Universitatea Politehnica Timişoara

# Layered LDPC Decoding Architectures:

## bridging the Gap from Algorithms to Implementations

**HABILITATION THESIS**

**Author: Oana AMĂRICĂI-BONCALO, PhD**

**– April 2019 –**

Universitatea Politehnica Timișoara

# Arhitecturi decodoare LDPC:
## studiu unitar pornind de la algoritmi până la implementare

**Teză ABILITARE**
**Autoare: Dr.Ing. Oana AMĂRICĂI-BONCALO**

**– Aprilie 2019 –**

## Rezumat

Această teză prezintă rezultatele academice ale autorului în intervalul 2013-2018. Direcția de cercetare este cea a decodoarelor LDPC care sunt folosite ca și mecanisme de detecție și corecție a eroriilor la nivelul interfețelor fizice de comunicație, precum și pentru memorii non-volatile semiconductoare. În cadrul codurilor LDPC, accentul cade asupra unei subclase - codurile cvasi-ciclice. Acestea au o structură regulată care permite explorarea unei varietăți mari de soluții arhitecturale. Din acest punct de vedere, ele au fost intens studiate de cercetători din zona proiectării de circuite dedicate. Codurile cvasi-ciclice LDPC prezintă o serie de proprietăți foarte interesante, precum toleranța ridicată intrinsecă la calculul cu erori. Acest fapt a motivat elaborarea unui proiect de cercetare - DIAMOND - Message Passing Iterative Decoders based on Imprecise Arithmetic for Multi-Objective Power-AreaDelay Optimization -, în colaborare cu grupuri de cercetători de la CEA-LETI Grenoble (coordonat de dr. Valentin Savin), și ENSEA Cergy-Pontoise (coordonat de prof. David Declercq). Proiectul DIAMOND a avut ca și țintă inițială elaborarea unor decodoare LDPC bazate pe aritmetică imprecisă. În vederea atingerii acestor obiective, primul pas a fost selectarea și implementarea unor versiuni arhitecturale de bază pentru aceste coduri. Urmărind punctele de variație, în încercarea de a maximiza efortul necesar proiectării și verificării unei astfel de arhitecturi hardware, am elaborat o metodologie prin care să avem posibilitatea descrierii și verificării unei întregi familii de astfel de decodoare LDPC. Descrierea familiei și machetele de verificare au la bază conceptele de șablon și etichete predefinite, precum și posibilitatea de definire de noi șabloane și etichete de către utilizator. În continuare am elaborat mai multe șabloane de decodoare LDPC, majoritatea folosind algorimul Min-Sum sau derivate ale acestuia, folosind planificarea de tip layered.

Numărul de biți corespunzător unui cuvânt de cod LDPC este de ordinul miilor. De asemenea, numărul de mesaje reprezentate pe 2-8 biți este cu un ordin de mărime mai mare decât lungimea cuvântului de cod. Din acest motiv, o atenție sporită trebuie acordată proiectării memoriei și mapării mesajelor în aceasta. Efortul nostru a fost în mare parte orientat asupra decodării LDPC ce folosește o planificare a procesării matricii de paritate de tip layered. Motivația din spatele acestei alegeri constă în eficiența stocării mesajelor în memorie, precum și prin prisma convergenței sporite în ceea ce privește performanța de decodificare. Performanța de decodificare este măsurată folosind metricile: rata de biți eronați, respectiv rata de cadre eronate. Pe lângă necesarul mare de memorie, numărul de unități de procesare este de la zeci pana la mii, funcție de nivelul de paralelizare ales pentru arhitectura. Așadar, am propus o serie de operații imprecise care să reducă costul și să îmbunătățească volumul de date decodificate pe secundă. Inițial, proiectul DIAMOND și-a propus să înlocuiască operațiile aritmetice exacte cu operații aritmetice imprecise. Ulterior, în urma rezultatelor cercetării, am decis că este mai eficient să introducem aceste imprecizii și la nivel de algoritm, respectiv la nivel de stocare în memorie a mesajelor (dat fiind numărul mare al acestora). Așadar, s-a materializat o direcție de cercetare cu multe rezultate în care au fost explorate limitele de imprecizie tolerate, precum și modalități eficiente

și sistematice de implementare a lor.

O problema critică în realizarea arhitecturii de memorie pentru un decodor LDPC este problema mapării mesajelor în bancuri de memorie și ordinea de access a acestora, în așa fel încât sa fie evitate hazardurile de tip citire-dupa-scriere (RAW). Acestea fac utilizarea resurselor hardware ineficientă, și sunt cu atât mai critice si greu de soluționat cu cât paralelismul este mai mare la nivelul arhitecturii. Au fost dezvoltate două direcții de abordare ale acestei probleme a maparii mesajelor în memorie și a găsirii unei planificari de accessare eficientă a acestora prin prisma ciclurilor de tact utile:

- Fiind dat un cod LDPC și un nivel de paralelizare, se pune problema găsirii unei mapări și a unei planificări a accesului optime. Memoria este organizată în bancuri de memorie implementate, folosind blocuri SRAM. Pentru soluționarea acestei probleme, am propus un set de algoritmi off-line care se bazează pe colorarea grafului și pe problema comis-voiajorului. De asemenea, pe partea de arhitectură, am introdus ca și suport pentru evitarea RAW-urilor, conceputul de mesaje reziduale.

- Dacă nu ne este impus codul LDPC, putem să abordăm această problemă de optimizare la nivelul construcției acestuia. Recent am propus o variantă algoritmului Progressive Edge Growth (PEG) bazată pe constrângeri arhitecturale (Arhitecture-aware Layered Progressive Edge Growth (AL-PEG)). Aceasta este o versiune extinsă a algoritmului de construcție PEG. Tehnica care stă la baza algoritmului PEG este o tehnică populară folosită și de alți cercetători.

Așadar, obiectivele inițiale ale proiectului DIAMOND, au fost lărgite și în final am obținut o colecție de familii de decodoare LDPC optimizate atât prin primsa operațiilor, cât și prin prisma stocării mesajelor în memorie, respectiv a accesului și procesării acestora. În viitor, vom cerceta și aspecte legate de introducerea deliberată de perturbații pentru a scădea costul, respectiv pentru a îmbunatăți performanța decodoarelor LDPC.

## Summary

This thesis presents the research and academic achievements during the 2014-2019 period. Modern communication and storage standards require efficient Forward Error Correction (FEC). Due to their excellent error correction capability, Quasy-Cyclic Low-Density Parity-check codes (QC-LDPC) are a class of codes employed in wireless standards, digital video broadcasting, and non-volatile semiconductor memories. This fact prompted the research direction we have pursued during the last 5 years, mainly the study of QC-LDPC decoder architecture trade-offs and optimizations. More specifically, within the framework of the project DIAMOND - Message Passing Iterative Decoders based on Imprecise Arithmetic for Multi-Objective Power-AreaDelay Optimization -, in collaboration with researchers from CEA-LETI Grenoble (dr. Valentin Savin), and ENSEA Cergy-Pontoise (prof. David Declercq), we have tried to exploit the advantages of implementing imprecise operations in Low-Density Parity-Check (LDPC) decoder architectures, in order to optimize the cost/area/power consumption. The original project goals – to develop hardware architectures that use imprecise arithmetic – have been largely expanded due to the very favorable research results. The contributions presented in this thesis closely follow the DIAMOND project.

The first step was to develop a collection of LDPC decoder architecture baselines for which the correct operations would be substituted with imprecise arithmetic operations. Due to the significant effort required for developing and verifying such an architecture for a single LDPC code, we have proposed and implemented a template based approach in order to be able to automate the design process. The proposed automatic methodology allows to describe the variation points across different LDPC code matrices, as well as to be able to verify the hardware design automatically. The associated know-how is embedded in the template, with the verification frameworks in SystemVerilog being reused for increased productivity and overall process quality. Several hardware templates corresponding to different memory organizations, schedulings, as well as parallelization degrees have been designed. The primary target has been layered scheduling for QC-LDPC, since it provides the most efficient message storage.

After successfully developing and comparing different template baseline decoder architectures, we have moved to the next step of replacing correct fixed point arithmetic operations with imprecise computation units. An important issue in LDPC design is represented by message storage, due to the large size of codewords: thousands to tens of thousands of messages, with each message in the range of 2-8 bits. This fact motivated extending the research scope to imprecise storage as well. Furthermore, our investigations have shown that investigating imprecise operations for the decoding process provides a more efficient means to optimize design and reduce cost. Thus, DIAMOND project's scope has been widened to investigate imprecise LDPC decoder operations, as well as imprecise storage.

Layered scheduling QC-LDPC offer a unique message memory mapping that reduces to half memory requirements for the extrinsic messages. Furthermore, it has the advantage of increased convergence. The drawback of using layered scheduling is represented by data hazards due to the late update effect caused by memory access time and pipeline. Furthermore, if implementation-wise, the message memory uses banks made of Static Random Access Memory (SRAM) blocks, the access patterns according to the code graph also introduce data conflicts. Hence, two problems need to be solved

in case such architecture choices and scheduling are present. We approached this problem from two directions:

- A set of offline algorithms has been proposed such that an almost optimum message memory mapping and access scheduling that avoid RAW hazards is generated. The message memory mapping represents a hyper-graph coloring problem, while avoiding RAW pipeline hazards represents a traveling salesman problem. In addition to this, since the RAW hazard problem is very constrained, we have also proposed adequate architecture support by using residue message information for correct decoder operation.
- Architecture aware code design for application where the LDPC code is not fixed. The proposed algorithm builds on a well known construction algorithm - Progressive Edge Growth (PEG). The proposed architecture aware PEG (AL-PEG) extends the original PEG by adding new constraints related to pipeline and message memory mapping. It tries to find a successful solution based on a given choice of hardware architecture and code parameters.

The DIAMOND project yielded successful collaborations and high quality research output; at the same time, it also laid the foundations for new research directions such as probabilistic decoding, design and verification of families of hardware architectures, fault tolerant design, etc.

## Acknowledgment

There are many people I had the opportunity of interacting and collaborating throughout the years. Still, I would like to pinpoint those that have inspired me and helped me grow up both as a person and as a professional. Thus, I will stick to a much shorter list. Through the years, Emanuel Popovici has been a friend on which I could always rely for various advices and support. He is also the person that gave me a vote of confidence and helped me develop my career after finishing mu PhD. I am also very grateful to Valentin Savin for all the advices and work that we managed to do together during the execution of the DIAMOND project (2014-2018). I sincerely hope to be able to work together with him in the future. Another key collaborator is David Declercq, distinguished researcher and a wonderful person. I managed to better grasp from him the role of a PhD advisor. From my coworkers, I would like to acknowledge the support of Marius Marcu, Pepi Mihancea and Mircea Popa. Over the years, I have been fortunate to work with highly skilled students, from which I would like to highlight one in particular - Gyorgy Antal Kolumban. His intelligence and perseverance are only surpassed by his professionalism. Sergiu Nimara is another important collaborator with whom I had many interesting research discussions.

Last but not least I owe a huge token of gratitude to my family that supported me unconditionally all the time: Sascha, Luca, Tutza, Galea, Mica, Vasile, Ioan, and Elena.

I am grateful to all of these people for giving me the opportunity to work with them, and for being part of my life!

# Contents



# List of Figures



# List of Tables

# Chapter 1

# Overview of author's research results

**Abstract:** *This Chapter presents the main research results in the 2010-2018 timeframe. In this period, the author has co-authored a number of 14 ISI journal papers, 48 conference papers, and 1 EPO/USPO patent application. Furthermore, she has been involved as principal investigator in two international research grants: the bilateral Romanian-French UEFISCDI-ANR project DIAMOND - Message Passing Iterative Decoders based on Imprecise Arithmetic for Multi-Objective Power-Area-Delay Optimization -, in collaboration with CEA-LETI Grenoble (dr. Valentin Savin), and ETIS Cergy-Pontoise (prof. David Declercq), and European Space Agency (ESA) Innovation Triangle Initiative (ITI) project REDOUBT - Reliable FPGA Datapath Design Using Control Techniques -, in collaboration with Technical University of Cluj-Napoca (dr. Zsofia Lendek). The most important scientific contributions during this period, which are the topic of this habilitation degree, are related to architectural improvements in the layered LDPC decoders. A summary of these contributions will be detailed in this thesis.*

## 1.1 Research Path

The author has graduated the Engineering Degree in Computer Engineering at Faculty of Automation and Computers, University Politehnica Timisoara, in 2006, and has finished her PhD in 2009, at the Department of Computers, University Politehnica Timisoara, in 2009, with the PhD Thesis "Simulation Based Reliability Assessment of Quantum Circuits". Since finishing her Phd studies, the author has worked in the following fields of expertise:

- Reliability and fault tolerance of digital circuits
- Hardware architectures for error correction codes, with a focus on Low Density Parity Check (LDPC) codes
- Digital arithmetic
- FPGA implementation of signal and image processing

In this fields, the author has the following scientifice results:

- 14 scientific papers in ISI indexed journals
- 48 scientific papers in conference proceedings
- 1 European Patent Office (EPO) and United States Patent Office (USPO) patent application
- 1 textbook and 2 bookchapters

Regarding the research activities, after graduation of her PhD studies, the authors has been involved as principal investigator in the following research projects:

1. DIAMOND - Message Passing Iterative Decoders based on Imprecise Arithmetic for Multi-Objective Power-Area-Delay Optimization - 2014-2017 - bilateral Romanian-French UEFISCDI-ANR PN-II-ID-JRP-RO-FR-2012-0109; project consortium: UPT - Romanian partner - and CEA-LETI Grenoble - principal investigator dr. Valentin Savin -, and Laboratoire des Equipes de Traitement de l'Information et Systèmes (ETIS), part of ENSEA and University of Cergy-Pontoise, - principal investigator prof. David Declercq, - French consortium -;

2. REDOUBT - Reliable FPGA Datapath Design Using Control Techniques - 2018-2019 - ESA Innovation Technology Initiative project number 4000123993/18/NL/CRS; project consortium UPT (lead) and Technical University of Cluj-Napoca (subcontracting) - principal investigator dr. Zsofia Lendek;

Furthermore, the author has been member in the following research projects:

1. Falx Daciae - Software Development Tools and Processes for Advanced Multimedia Applications on Mobile Phone Multi-Core Architectures - 2010-2012 - project type: POSCCE/A2-O2.1.1/449/11844 - consortium: Movidius Timisoara - principal investigator dr. Valentin Muresan - and UPT - principal investigator prof. Mihai Micea

2. CHIST-ERA GEMSCLAIM - GreenEr Mobile Systems by Cross LAyer Integrated energy Management - 2012-2015 - CHIST-ERA project - project consortium: University of Innsbruck(lead) - principal investigator prof. Thomas Fahringer -, UPT - principal investigator prof. Marius Marcu -, Queens University of Belfast - principal investigator prof. Dimitrios Nikolopoulos -, and RWTH Aachen - principal investigator prof. Reiner Leupers;

During these years, the author has been Handling Editor for "Microprocessors and Microsystems" journal (edited by Elsevier) - ISI journal with impact factor 1.049 -, and has reviewed papers in journals including: "IEEE Transactions on VLSI", "IEEE Transactions on Circuits and Systems I: Regular Papers ", "IEEE Access", "Integration, the VLSI journal", "Computers and Electrical Engineering", "Physical Communication".

The author has been invited in two long term (more than 2 weeks) research visits

- University College Cork (dr. Emanuel Popovici) - June-July 2012
- Univesity of Cergy-Pontoise (prof. David Declercq) - January-February 2016

The author has more than 100 independent citations in papers indexed in Scopus, with 40 independent citations in ISI Web of Science.

## 1.2 Summary of contributions

In this habilitation thesis, scientific and research contributions related to architectural design space exploration and optimization for layered LDPC decoder will be detailed. These contributions have been developed during the "DIAMOND - Message Passing Iterative Decoders based on Imprecise

Arithmetic for Multi-Objective Power-Area-Delay Optimization" research project, in collaboration with the research teams from CEA-LETI Grenoble, led by dr. Valentin Savin, and ENSEA Cergy-Pontoise, lead by Prof. David Declercq.
The scientific contributions related to these, can be classified as follows:

- **Architecture design space exploration of layered LDPC decoders** - Regarding this topic, the main contributions have targeted the following:
  - Efficient implementation of layered LDPC decoding architectures on FPGA devices; results corresponding to this contribution have been disseminated in the following conference paper:
    **O. Boncalo, A. Amaricai, A. Hera, and V. Savin, "Cost-efficient FPGA layered LDPC decoder with serial AP-LLR processing,"" in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014, pp. 1–6.**
  - An architectural design exploration tool for automatic generation of layered LDPC decoders and the corresponding verification infrastructure; results corresponding to this contribution have been disseminated in the following conference paper:
    **O. Boncalo, P. F. Mihancea, and A. Amaricai, "Template-based QC-LDPC decoder architecture generation,"" in *2015 10th International Conference on Information, Communications and Signal Processing (ICICS)*, Dec 2015, pp. 1–5.**
  - Ultra high throughput layered LDPC decoder, based on layer unrolling; results corresponding to this contribution have been disseminated in the following conference paper:
    **O. Boncalo and A. Amaricai, "Ultra high throughput unrolled layered architecture for QC-LDPC decoders," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, vol. 00, July 2017, pp. 225–230. [Online]. Available: "doi.ieeecomputersociety.org/10.1109/ISVLSI.2017.47"**

- **Optimization of layered LDPC decoder architectures based on imprecise computation and message storage** - - Regarding this topic, the main contributions have targeted the following:
  - Memory efficient approximate version for Self-Correcting Min-Sum (SCMS) LPDC decoding; results corresponding to this contribution have been disseminated in the following papers:

    **O. Boncalo, A. Amaricai, P. F. Mihancea, and V. Savin, "Memory trade-offs in layered self-corrected min-sum LDPC decoders," *Analog Integrated Circuits and Signal Processing*, vol. 87, no. 2, pp. 169–180, May 2016. [Online]. Available:** `https://doi.org/10.1007/s10470-015-0639-3`

**O. Boncalo, A. Amaricai, and V. Savin, "Memory efficient implementation of self-corrected min-sum ldpc decoder," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2014, pp. 295–298.**

– Imprecise one-hot based implementation of check-node unit based computation; results corresponding to this contribution have been disseminated in the following journal paper:

**O. Boncalo, A. Amaricai, V. Savin, D. Declercq, and F. Ghaffari, "Check node unit for LDPC decoders based on one-hot data representation of messages," *Electronics Letters*, vol. 51, no. 12, pp. 907–908, 2015.**

– Efficient architectures for Non-Surjective Finite Alphabet Iterative Decoder (NS-FAID) LDPC decoding; results corresponding to this contribution have been disseminated in the following papers:

**T. Nguyen-Ly, K. Le, F. Ghaffari, A. Amaricai, O. Boncalo, V. Savin, and D. Declercq, "Fpga design of high throughput LDPC decoder based on imprecise offset min-sum decoding," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, June 2015, pp. 1–4.**

**T. T. Nguyen-Ly, K. Le, V. Savin, D. Declercq, F. Ghaffari, and O. Boncalo, "Non-surjective finite alphabet iterative decoders," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.**

**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, March 2018**

**O. Boncalo, V. Savin, and A. Amaricai, "Unrolled layered architectures for non-surjective finite alphabet iterative decoders," in *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, Oct 2017, pp. 1–5.**

– In-between layer based stopping criterion for layered LDPC decoders; results corresponding to this contribution have been disseminated in the following papers and patent:

**A. Hera, O. Boncalo, C. Gavriliu, A. Amaricai, V. Savin, D. Declercq, and F. Ghaffari, "Analysis and implementation of on-the-fly stopping criteria for layered qc-ldpc decoders," in *2015 22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES)*, June 2015, pp. 287–291.**

**D. Declercq, V. Savin, O. Boncalo, and F. Ghaffari, "An imprecise stopping**

**criterion based on in-between layers partial syndromes,"** ***IEEE Communications Letters*****, vol. 22, no. 1, pp. 13–16, Jan 2018.**

**V. Savin, O. Boncalo, and D. Declercq, "Stopping criterion for decoding quasi-cyclic ldpc codes,"** ***European Patent Office EP3373488A1***

- **Memory conflict and pipeline hazard mitigation in layered LDPC decoders** - Regarding this aspect, the contributions have targeted the following:
  - Algorithms for message mapping in memory banks and pipeline related hazard removal for a given LDPC code (such as LDPC codes used in communication standards); the results related to this contribution have been disseminated in the following journal paper:

    **O.Boncalo, G.Kolumban-Antal, A.Amaricai, V.Savin, and D.Declercq, "Layered LDPC decoders with efficient memory access scheduling and mapping and built-in support for pipeline hazards mitigation"** ***IEEE Transactions on Circuits and Systems I: Regular Papers*****, pp. 1–14, 2018.**

  - Code construction methods based on progressive-edge growth (PEG) that outputs LDPC codes friendly for pipelined layered hardware architecture; results related to this contribution have been disseminated in the following journal paper:

    **O. Boncalo, G. Kolumban-Antal, D. Declercq, and V.Savin, "Code-design for efficient pipelined layered LDPC decoders with bank memory organization"** ***Microprocessors and Microsystems*****, vol. 63, pp. 216 – 225, 2018. [Online]. Available:** `http://www.sciencedirect.com/science/article/pii/S0141933118300863`

- **Miscelanous contributions to LDPC decoder implementations and optimizations**
  - Efficient probabilistic gradient descent bit flipping on QC-LDPC codes using imprecise units using flooding scheduling has been published din one conference paper, and one journal paper:

    **K. Le, D. Declercq, F. Ghaffari, L. Kessal, O. Boncalo, and V. Savin, "Variable-node-shift based architecture for probabilistic gradient descent bit flipping on qc-ldpc codes,"** ***IEEE Transactions on Circuits and Systems I: Regular Papers*****, vol. 65, no. 7, pp. 2183–2195, July 2018.**

  - Partially-parallel flooding scheduling implementation of a QC-LDPC decoder using imprecise storage has been published in one conference paper:

    **O. Boncalo, A. Amaricai, and S. Nimara, "Memory-centric flooded ldpc decoder architecture using non-surjective finite alphabet iterative decoding," in** ***2018 21st Euromicro Conference on Digital System Design (DSD)*****, Aug 2018, pp. 104–109.**

– Dynamic quantization with different decoding speeds and message storage mappings across iterations for a partially-parallel flooding scheduling architecture of a QC-LDPC has been published in one conference paper:

**O. Boncalo, "Qc-ldpc gear-like decoder architecture with multi-domain quantization," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 244–251**

## 1.3 Thesis outline

The thesis will be organized as follows: the following three Chapters - Chapter 2, Chapter 3 and Chapter 4 - will be dedicated to the detailed presentation of the main research and scientific contributions related to the LDPC decoder architecture tradeoffs and optimizations of layered LDPC decoders. Chapter 2 presents the main design exploration strategies for architectural optimization of layered LDPC decoders, as well as the process supporting it.

Contributions related to optimizations of layered LDPC decoders based on imprecise computations are the subject of Chapter 3. A short overview of contributions in the field of imprecise computation for LDPC decoders relying on other algorithms besides MS and other scheduling beside layered scheduling is discussed briefly in Section 3.6 of Chapter 3.

Chapter 4 is dedicated to optimizations related to mitigation of memory conflicts and pipeline related hazards in layered architectures: algorithms for conflict and hazard removal are presented, as well as LDPC code construction techniques based on progressive-edge growth (PEG) suited for layered architectures.

The last Chapter of this habilitation thesis will consists of main research directions that will be tackled by the author in its future academic career: an overview of future perspectives and challenges will be presented, and future steps will be detailed.

# Chapter 2

# Layered Decoder Architectures Design Space

**Abstract:** *This section presents the contributions on the hardware architecture side of the layered scheduling LDPC decoder implementations. We start first by defining the general notations and working principle, and we continue with discussing the implementation trade-offs, and proposed improvements, and we close this section with a brief discussion of the framework for automating design space exploration for hardware architectures in general, and for LDPCs as a use case. The optimizations proposed for the baseline layered scheduling decoder are introduced both on the processing unit level, and on the LDPC architecture level.*

## 2.1 Introduction

Modern communication and storage standards require efficient FEC. Given their excellent error correction capability QC-LDPC are a class of codes employed in wireless standards (WiFi [20], WPAN [10], WiMAX [7]), digital video broadcasting (DVB [21]) and flash memories [22]. Furthermore, they belong to a family of structured code with properties that allow different trade-offs with respect to the hardware implementation architecture's parallel message processing capabilities. Thus, an abundant stream of research works focus on different aspects of QC-LDPC decoding such as: algorithmic and scheduling aspects [23][24][25], approximate computation and storage [26][27][28][29], different architecture implementations[30] [31][32], [33], message and input quantization [34][28][1][16][29], code-design favorable for a target architecture [35][19][36], and off-line algorithms optimizing scheduling [37][38][39][40], and memory access [41][42][15]. Furthermore, interest has been shown into efficient implementations of standard codes[43][39][2] as well. Many of these results can complement each other, yielding a large design space with many different architecture choices.

Before discussing the architecture design choices of QC-LDPC decoder architectures, it is worthwhile emphasizing some of their key features: long codewords of thousands and tens of thousands of bits, structured linear codes with good decoding capability, standard codes typically have several rates specified, and even several code lengths. This organization prompts for the following decoder architecture design knobs:

- number of processing units: this choice is directly reflected in throughput and area, and it dictates the required memory bandwidth accessing the messages.
- number of input messages of the processing units: impacts the memory bandwidth and access

pattern, thus, introducing constraints on the memory hierarchy. Furthermore, it is directly reflected in the processing units complexity.

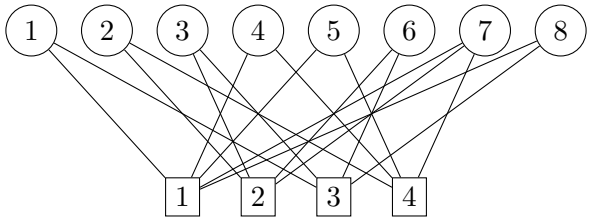
- message memory organization and message quantization: dictates the overall storage requirements. LDPC codes are efficient for codeword lengths of thousands, and tens of thousands of bits. Therefore, the storage requirements have to accommodate a comparable large number of messages.
- interconnect choice: constrained by the number of parallel units.
- processing units design: has two dimensions, one is related to the implementation of the arithmetic operations, and the second is related to the units re-use for several codes, and even working modes.
- number of consecutive layers processed together at one time: this is specific to the layered scheduling discussed in this thesis.
- number of decoders working in parallel: for Tbit throughput, the entire decoder architecture is unrolled several times.
- number of codewords that are processed by a unit in a time division manner (*i.e.* serialized processing).

This chapter is organized as follows: in Section 2.2 we start by providing a very brief introduction in LDPC codes and layered scheduling decoding principle, coupled with a summary of the notations used throughout this thesis. Then, in Section 2.3 we start by discussing the baseline layered architecture templates, and continue with the optimizations we have proposed. Since efficiency is a key requirement for any engineering work, we have invested effort in embedding knowledge in hardware design templates such that this knowledge is re-used. The result is a family of hardware design templates. The approach and one of the case studies are presented in Section 2.4. We conclude with an overview of the main scientific contributions related to QC-LDPC architecture implementations for layered scheduling decoders.

The technical contributions reviewed in this Chapter have been disseminated in the following venues:

- O. Boncalo, P. F. Mihancea, and A. Amaricai, "Template-based QC-LDPC decoder architecture generation," in *2015 10th International Conference on Information, Communications and Signal Processing (ICICS)*, Dec 2015, pp. 1–5.
- O. Boncalo and A. Amaricai, "Ultra high throughput unrolled layered architecture for QC-LDPC decoders," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, vol. 00, July 2017, pp. 225–230. [Online]. Available: "doi.ieeecomputersociety.org/10.1109/ISVLSI.2017.47"
- O. Boncalo, A. Amaricai, A. Hera, and V. Savin, "Cost-efficient FPGA layered LDPC decoder with serial AP-LLR processing," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014, pp. 1–6.




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

Figure 2.1: Tanner Graph with equivalent $H$ matrix representation – Example from [1]

## 2.2 Notations and the Layered scheduling decoding principle

LDPC codes are linear codes that can be described using a bipartite graph $\tilde{G}$, or alternatively using a sparse parity check matrix $H$ representation [44][45][46]. The bipartite graph has two kinds of nodes: variable-nodes, and check-nodes. The variable-nodes correspond to the columns of the parity check matrix $H$. More specifically, they correspond to the codeword bits. The check-nodes are the $H$ matrix lines, which dictate the parity check equations. The alternative representation is using a bipartite graph called the Tanner graph [46][47]. All non-zero $H$ matrix entries have a corresponding edge in the Tanner graph. The decoding process take place by iterative exchange of "beliefs" (*i.e.* messages) along the edges of the Tanner graph (see Fig. 2.1).

QC-LDPC codes are a class of structured LDPC codes that are obtained by *expanding* a base matrix $B$ by an expansion factor. Specifically, each $B$ matrix entry is replaced by either a circulant permutated matrix or by a zero matrix [47]. The non -1 entries from the base matrix $B$ are replaced by the unitary matrix shifted by the $B$ matrix value. The -1 entries are replaced by the zero matrix. The resulting structure is very favorable for hardware implementations, motivating their use in many standards for communication and video systems.

Belief Propagation (BP), also known as Sum-Product (SP), is proved to be optimal for cycle-free LDPC codes [48]. For the cycle-free case it outputs the Maximum A Posteriori (MAP) estimates of the coded bits. Nonetheless, two observations are in place: (1) practical LDPC codes have cycles, and (2) the implementation of BP has high computational complexity (*e.g.* logarithm, hyperbolic tangent). Another BP drawback is that it requires an accurate estimation of the Signal-to-Noise Ratio (SNR) - difficult to implement in many practical cases. MS overcomes these two limitations by: (i) using max-log approximations the check node messages, and (ii) for most of the usual channel models the SNR information is not needed [49]. In fact MS algorithm relies only on additions and comparisons, as its name suggests. Other MS variants are Offset Min-Sum (OMS), SCMS [50], Normalized Min-Sum (NMS) [51][52] try to compensate the overestimation of MS, by reducing the amplitude of some messages. Results discussed throughout this thesis use the hardware friendly MS and some of its variants.

As mentioned previously during LDPC decoding there are two kinds of message processing: variable-node and check-node. The simplest LDPC scheduling strategy is to first process all VN, followed by CN processing, yielding two distinct sub-phases. This decoding approach is called flood-

ing scheduling. Another representative scheduling is called layered scheduling. It groups several $H$ matrix lines together into what is called a layer. For the QC-LDPC decoders a layer corresponds to a line of the base matrix $B$, and the number of $H$ rows is equal to the expansion factor. In layered scheduling decoders, the $H$ matrix in processed layer by layer, and most importantly updated information is propagated after each layer processing. From this behavior stems the superior convergence (*i.e.* average number of iterations required o decode a codeword) and decoding performance of layered versus flooding scheduling [53]. Furthermore, implementation-wise it is possible to avoid storing variable-node messages all together. This is a significant advantage when discussing hardware implementations. In this thesis we focus on layered scheduling LDPC decoders. Thus, in order to facilitate the discussion, we start by presenting the layered decoder algorithm in Algo. 1, with the notations being used explained in Table 2.1.

**Algorithm 1** Layered scheduling decoding principle[15]

**Initialization:**
  **set** $\beta_{r,t}$ messages to 0
  **set** $\gamma_t$ to channel LLR values
  **set** $It = 0$ and $syndrome = false$
**while** $(It \leq It_{max})$ and $(syndrome = false)$ **do**
  **for all** $m \in M$ **do**
    **for all** $r \in N(m)$ **do**
      **Variable-Node:** compute $\alpha_{r,t}$ messages
        $\alpha_{r,t(r,n)} = \gamma_{t(r,n)} - \beta_{r,t(r,n)}, \forall n \in N(m)$
      **Check-Node:** compute $\beta_{r,t}$ messages
      $\beta_{r,t(r,n)} = \prod_{n' \in N(m) \setminus \{n\}} sign(\alpha_{r,t(r,n')}) \times$
        $\min_{n' \in N(m) \setminus \{n\}} \alpha_{r,t(r,n')}, \forall n \in N(m)$
      **AP-LLR update:** compute $\gamma_t$ messages
      $\gamma_{t(r,n)} = \alpha_{r,t(r,n)} + \beta_{r,t(r,n)}; \forall n \in N(m)$
  **compute** new *syndrome* according to sign($\gamma$) values
  **set** $It = It + 1$
**Offloading:** output $\leftarrow$ sign($\gamma$)

Note that the A-Posteriori Log-Likelihood Ratios (AP-LLR) messages (denoted by $\gamma$), store the contribution of all check-node messages plus the channel input - Log-Likelihood Ratios (LLR).

### 2.2.1 Message update formalization in layered scheduling decoders

According to the layered decoding principle, each variable-node $n$ is updated $d_v(n)$ times during an iteration. Consequently, the $\gamma_n$ message containing the equivalent of the contribution of all check-nodes and channel input is updated $d_v(n)$ times; thus, resulting in the *superior convergence of layered scheduling* decoding, as compared to flooding scheduling decoding, since the latter only performs one VN update per iteration [53]. In addition to this, in order to have correct layered scheduled decoding, the AP-LLR messages that are common for successive layers need to have their values updated, otherwise the check-node message contribution from the earlier scheduled layer is lost.

In this chapter we discuss message re-ordering issues. This are the subject of Chapter 4. The mitigation for reducing stalls due to hazards is that of employing multiple codewords.

Table 2.1: Notations

| Symbol | Description |
|---|---|
| $B$ | base matrix |
| $G$ | base graph corresponding to $B$ matrix |
| $z$ | circulant size (also called expansion factor) |
| $H$ | parity check matrix |
| $\widetilde{G}$ | Tanner graph |
| $It_{max}$ | maximum number of decoding iterations |
| $M$ | set of all variable nodes of matrix $B$ |
| $N$ | set of all check nodes of matrix $B$ |
| $m$ | check-node of base graph/base matrix |
| $n$ | variable-node of base graph/base matrix |
| $H(n)$ | set of $z$ variable-nodes of H corresponding to column $n$ of matrix $B$ |
| $H(m)$ | set of set of $z$ check-nodes of $H$ corresponding to row $m$ of matrix $B$ |
| $N(m)$ | the set of variable-nodes connected to check-node $m \in M$ |
| $M(n)$ | the set of check-nodes connected to variable-node $n \in N$ |
| $d_c(m)$ | check-node $m$ degree |
| $d_c^{max}$ | maximum check-node degree |
| $d_v(n)$ | variable node $n$ degree |
| $\gamma_t$ | $APP\text{-}LLR_t$ message |
| $\alpha_{r,t}$ | variable $t$ to check node $r$ message |
| $\beta_{r,t}$ | check $r$ to variable node $t$ message |
| $t(r;n)$ | the unique $t \in H(n)$ connected to $r \in H$ |
| $\tilde{*}$ | partial result |
| $\lvert\cdot\rvert$ | the cardinal of a set operator |

Table 2.2: Hardware architecture parameter notations

| Symbol | Description |
|---|---|
| $n_{banks}$ | number of single-port memory banks |
| $N_C$ | processing unit parallelism |
| $N_P$ | number of pipeline levels for the processing units |
| $n_{latency}$ | number of clock cycles from the time a $\gamma$ message is read, until it is available for subsequent processing |
| $pd$ | function computing the message processing duration based on the update order rule |
| $n_{layers}$ | number of layers processed simultaneously |
| $N_{CW}$ | number of codewords processed in a time division manner |

## 2.3 Generic layered scheduling decoder architectures: trade-offs

The main design choices in a typical layered decoder architecture from [15], revised and extended:

1. *Parallelism degree at processing unit level* –it refers to the number of input messages for a processing unit per clock cycle. By serial message processing, we understand one message per clock cycle [5,39,54], while fully parallel processing means that all $d_c/d_v$ messages are processed at the same time [29,55]. The in-between solution, referred hereafter as partially parallel operation ($1 < n_{banks} < d_c^{max}$) may be employed [15].

2. *Number of data-path pipeline levels* – $N_P$ accounted inside the $n_{latency}$ latency value. Although pipeline is a well known method for increasing working frequency, for layered scheduling, it also represents a source of data hazards [37,38]. Adding also single-port memories as building blocks for the AP-LLR memory increases the difficulty of finding of a message processing order that minimizes architecture stalls for cases with $n_{banks}$ >1 [15].

3. *Message update strategy* – determines the order in which the message are written back to memory at the end of CN processing. Three update strategies are proposed in literature: ($i$) unconstrained output processor, no relation between the message read order from the write-back order [38,56], ($ii$) First-In First-Out (FIFO) update, or in-order update the read and write-back order stay the same[37,57], and ($iii$) reversed write-back, which writes the messages in reverse order than their read access from memory banks[39,40][2]. All these choices have pros and cons. The first approach is the least constrained with respect to scheduling, but is the most expensive in terms of cost, if implemented by a register file (RF). The normal (FIFO) write-back is the natural choice for regular codes, cost efficient, and can be implemented using a simple shift register. If multi-code, and multi-rate need to be supported the shift register needs to be replaced by a FIFO. The reverse write-back is both cost efficient and it flexible. It can straightforward accommodate multi-rate and multi-codes, and it can be implemented using a stack. Thus, it can easily accommodate irregular standard codes that in many cases are quasi-regular in $d_c$, therefore allowing a simplified stack architecture [2].

4. *Number of layers processed in parallel* – $n_{layers}$ – for the case when all layers are processed by the hardware implementations, based on the number of $H$ lines processed per layer, two architectures have been proposed in literature: one line/layer [55], and all $z$ lines per layer, using 2 layers [58], and all layers [4].

5. *Number of codewords decoded at a time* – $N_{CW}$ – number of codewords that are processed in a time division time multiplexing fashion at one time inside the LDPC decoder unit. Typically one codeword is processed at a time. However, in order to improve hardware usage efficiency and minimize stall clock-cycles, several codewords can be processed. In this way, the pipeline latency for example can be compensate by processing data of different codewords. This approach comes at the price of extra storage requirements. However, as shown in [5], this overhead is technology and code dependent. For Field Programmable Gate Array (FPGA) technology, the Block RAM (BRAM) memory may accommodate extra codewords at the same cost.

### 2.3.1 Processing Unit Decoder Design [1]

[1] This subsection contains results and text partially reproduced from the conference paper [2]

As discussed in the previous section layered scheduling decoders design of QC-LDPC codes utilizing MS have a unique memory organizations, especially favorable for hardware implementations, as well as good decoding performance and convergence. Two possible implementations processing units implementations have been proposed:

- merged processing units - VCN. Merging both units has allowed for hand-made design implementations that exploit FPGA technology mapping, such as in the case of the work [2]. The serial processing of the a posteriori LLRs allowed the design of a high frequency VCN unit that replace data conversions as well as additions and comparators by look-up-tables implemented using distributed RAM. This is possible for (5,3) bits message quantizations as described in the paper:

  – O. Boncalo, A. Amaricai, A. Hera, and V. Savin, "Cost-efficient fpga layered ldpc decoder with serial ap-llr processing," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014, pp. 1–6.

  This merged VCN architecture has been generalized in the work [5] for an arbitrary number of input messages (Fig. 2.8).

- separate Variable-Node Unit (VNU) and CNU processing units: operations are computed separately. Two possibilities exist, based on the routing network position (*i.e.* before VNU processing – routing AP-LLRs [59], or after VNU processing – routing $\beta$ messages [15]. The difference lies in the size of this messages. The AP-LLR messages are larger in size than $\beta$ messages. Using Density Evolution the optimum difference considered is typically of 2 bits. For (6,4) bits message quantization, this amounts to 50% reduction of the routing network.

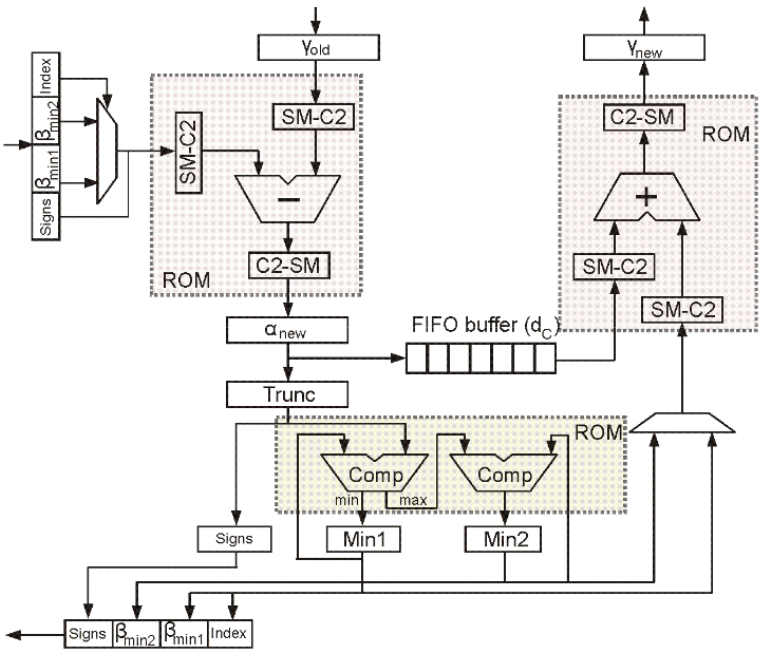



Figure 2.2: Serial processing unit of the a posteriori LLRs with FPGA specific optimizations merging VN and CN operations, which achieves high working frequencies by using ROM memories that replace data conversions as well as additions and comparators by look-up-tables implemented using distributed RAM [2].

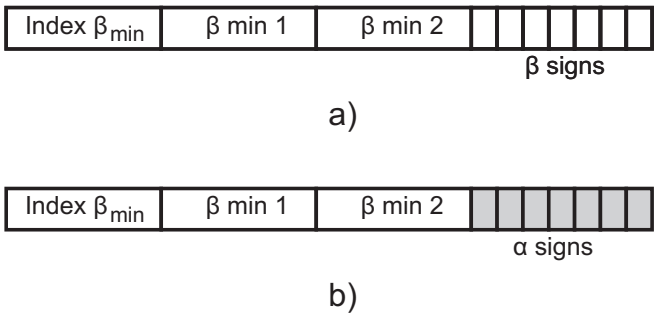



Figure 2.3: Compressed word format in MS (a) and SCMS (b) [3]

Other optimizations proposed for high rate codes, address the comparator design. The comparator is on the critical path of the CNU, and given the small number of bits per input (*e.g.* 2,3, 4 bits) we have proposed a one-hot encoding optimization in:

- O. Boncalo, A. Amaricai, V. Savin, D. Declercq, and F. Ghaffari, "Check node unit for LDPC decoders based on one-hot data representation of messages," *Electronics Letters*, vol. 51, no. 12, pp. 907–908, 2015.

This approach introduces a small impreciseness, and is suitable for LDPC codes with $d_c > 30$.

### 2.3.2 Layer Unroll Decoder [2]

When targeting throughputs in the range of tens and hundreds of Gbps, higher parallelism degrees are required. For flooding scheduling decoders, such a case has been presented in [60] for unrolling several iterations of fully parallel decoder architectures, and by employing multiple codewords being processed

[2]This subsection contains results and text partially reproduced from the conference paper [4]

Table 2.3: Regular $d_v = 3$ rate 3/4 code $B$ matrix $z = 108$, code size 1296, and girth (*i.e.* size of the shortest cycle in the Tanner graph) 8, with the girth multiplicity 31428 [4]

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 44 | 1 | 107 | 51 | 52 | 22 | 65 | 36 | 94 | 105 | 75 |
| 61 | 102 | 85 | 63 | 54 | 65 | 99 | 3 | 14 | 13 | 32 | 18 |
| 70 | 44 | 30 | 3 | 84 | 46 | 40 | 16 | 2 | 16 | 18 | 77 |

simultaneously inside the unrolled architecture. For layered scheduling, a two layer unroll architecture has been proposed in [58]. Our contribution is that of an unrolled decoder architecture for high rate QC-LDPC codes typically used in storage systems. It uses layered scheduling, targeting codes characterized by a very small number of non-zero elements for each layer. In addition to this it takes advantage of FPGA technology characteristics to optimize implementation cost. For this architecture data dependency among layers is compensated by using multiple codewords. The processing units are fully parallel. The unroll approach, followed by pipeline registers, provides a natural way for increasing the amount of parallelism within the architecture, resulting in a regular structure, with simplified routing, and control as depicted in Fig. 2.4. The architecture is presented in Fig. 2.5.

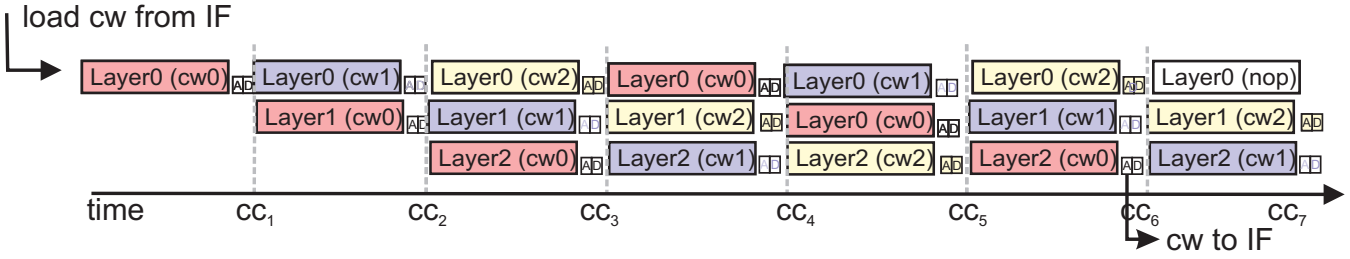



Figure 2.4: Pipeline chronogram showing the execution of 3 iterations, for three layers unrolled inside the proposed architecture. A total of three codewords are in different execution stages at a time [4].

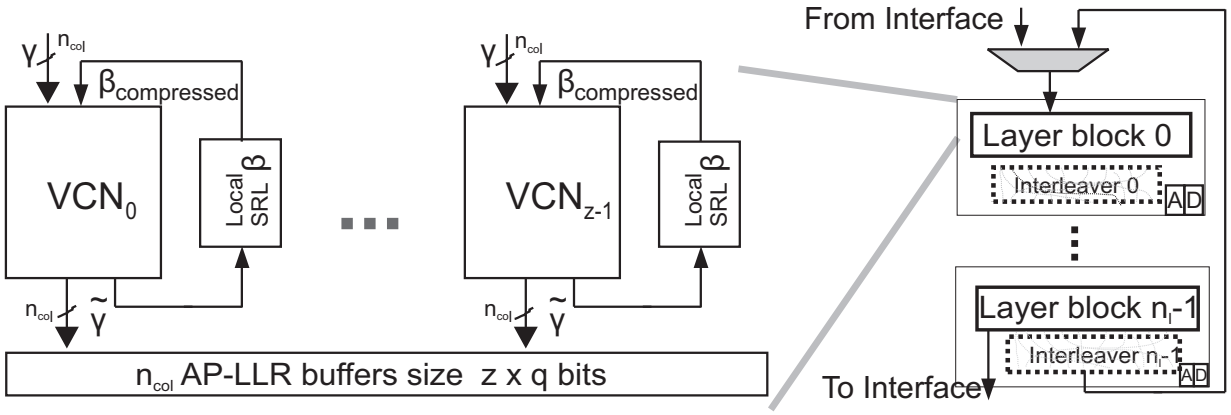



Figure 2.5: Unroll architecture overview n layers unrolled (right) and the detailing of a layer block implementation is described left. Note the $\beta$ messages are stored in local SRL register memories, while AP-LLRs are stored inside the in-between layer registers. Each layer block implementation has $z$ merged fully-parallel processing units [4].

The following observations are in place with respect to the FPGA results reported in Table 2.4 take advantage of the following :

- The Xilinx FPGA architectures features a look-up table (LUT) element that can be configured as logic, ROM/RAM, or SRL. The LUT SRL configuration used for storing CN messages is cost

Table 2.4: Resource and throughput estimates for the array code corresponding the code from Table 2.3, having 3 unrolled layers, with 3 codewords being be decoded at the same time, and an iteration latency of 3 cycles/codeword. Throughput (T) is computed for a number of 4 iterations corresponding to a FER below $10^{-4}$ [4]

| | Slice registers | Slice LUTs | | Freq [MHz] | Throughput [Gbps] |
|---|---|---|---|---|---|
| | | Logic | Memory | | |
| MS (4,6) | 36986 | 247411 | 7128 | 106 | 34.3 |
| MS (3,5) | 31154 | 213761 | 6480 | 114 | 36.9 |
| MS (2,4) | 25322 | 70036 | 5832 | 197 | 62.8 |
| SCMS:v2(3,5) | 31154 | 201428 | 6480 | 104 | 33.6 |
| SCMS:v2(2,4) | 25332 | 104284 | 5832 | 167 | 54.1 |

efficient [61]. In addition to this, AP-LLR memory is removed, as information is stored inside the pipeline registers of in-between stages.

- Due to the technology specific storage optimizations, memory no longer dominates the overall cost. Logic in the case of the unroll architecture from [4] becomes the main contributor.

- the latency bottleneck for the VCN stage is the comparator, which has 12 inputs for the code from Table 2.3. Cost and latency have been improved by the removal of the hardwired interconnect of each layer.

To sum up, the proposed architecture allows optimum throughput/cost ration achieving a throughput of up to 62 Gbps for an average of 4 iterations. This way, the proposed unrolled layered architecture can be used for newer generations of telecommunication standards, such as 5G networks, as well as the newer generations of non-volatile storage devices, where one major focus is represented by throughputs in range of tens of Gbps.

## 2.4 Framework for design space exploration of LDPC decoders [3]

Exploring various LDPC configurations for space / energy optimization requires to have complete, verified decoder implementations using a hardware description language (e.g. Verilog HDL, VHDL). However, since LDPC codes are tailored to specific applications, with different circuit design requirements (throughput, area, energy), as well as different decoding performance (Frame Error Rate (FER) curve error floor region for storage, and waterfall for wireless communication) a lot of "fitting" is required on for the baseline architecture. This translates to variations which are "customized" according to requirements, and much description that is duplicated in different LDPCs. As a result, the person in charge needs to manually manage all these separate descriptions with all the problems involved by code duplication e.g., hard to modify when a common modification must be propagated in all the replications of the common code (see Fig. 2.6).

[3] This subsection contains results and text partially reproduced from the conference paper [5]

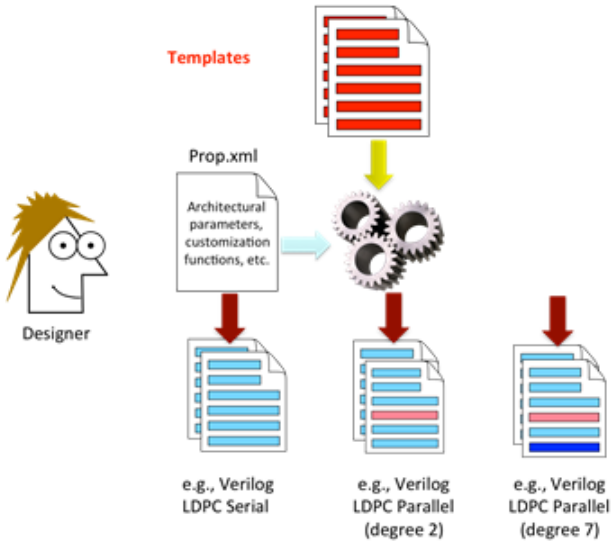



Figure 2.6: Overview of template-design based principle as described in [5]

### 2.4.1 The process

This approach builds in fact an entire process that uses both in-house and commercial software. The flow used for hardware design is presented in Fig. 2.7. What Tool and Environment for HDL Template Design (TeDi) allows is an efficient way to describe families application specific circuits, which have fine tune design optimizations, as well as application domain knowledge optimization that are reflected inside the architecture. This typically requires one or several software implementations of algorithms tuning architecture parameters and/or control configuration information, and are executed on-line. These software programs are in-house developed, and need to be seemingly integrated in the same flow as the vendor tools, such that the entire process is automated.

An important aspect lies in the fact that this "complete environment" also supports the verification process. This is important given that every change and variation needs to be properly verified. Furthermore, as discusses subsequently the idea is to speed up the learning curve and allow the designer and verification engineer to use/reuse scripts and HDL design language and legacy code, without the need to learn an entirely new language. Ultimately, a template Verilog HDL file is regarded from the tool point of view as a sort of text processor. It contains some annotations / commands that have some user inputs. these are replaced by text. From the user's point of view we have atypical design file with some annotations that help customize the more complex variations that cannot be handled using the generic design parameters and generate Verilog HDL statements. These annotations are hereafter referred as tags, and they can be either generated or user defined tags. Thus, both the adoption of this approach and the code's readability are improved.

Other concrete approaches targeting LDPC decoder architectures are presented in the works of [62] combining combines Verilog HDL with Phyton, the work of [63] relying on Matlab, the work from [64] using Vivado HLS, and works such as [65] using Simulink. The approach in [62] gives no insight on the template, or on the FPGA specific optimizations.

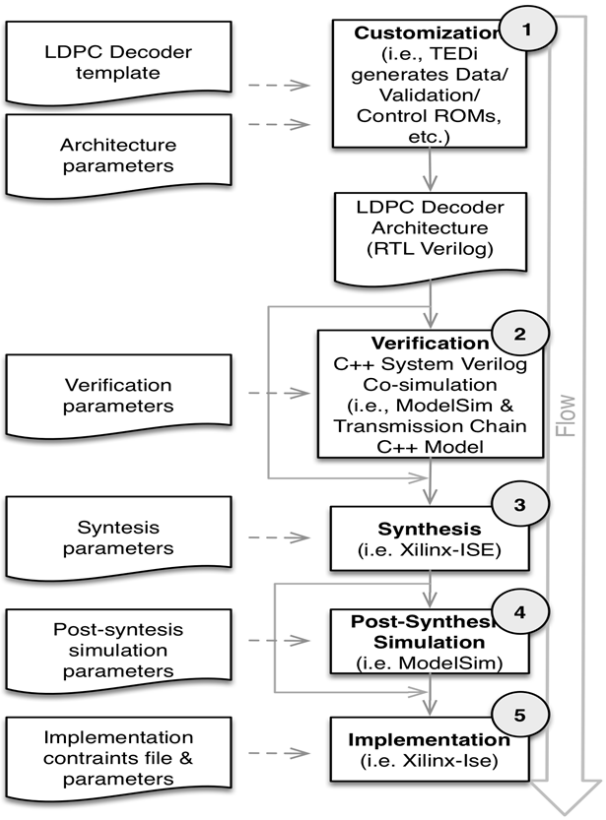



Figure 2.7: The process for design/verification/implementation emphasizing the tool integration for the particular case of LDPC decoders. The main inputs are the templates and a configuration file (*e.g.*, Prop.xml), and the process output is the entire description (*i.e.*, Verilog code) of the LDPC variation the user asked for via the configuration file (e.g., serial LDPC, parallel LDPC with a parallelization degree of 2, etc.), together with the scripts and verification files needed to validate the design and/or derive implementation results [5]

### 2.4.2 The template and tags

The idea of generating the description code of an entity starting from on a high-level template descriptions not knew. For software, good such references are the JSP [66] and the ASP [67] technologies that are based on the idea that each webpage displayed to a user (its HTML code) is actually generated from e template page. Similar hardware design approaches are [68] that relies on a set of directives, Genesis2 [69], which has a Perl script template that uses Perl statements to describe how various Verilog code sequences are combined for a complete design file. Our approach is different in the sense that the it provides a clean separation between the actual hardware description and the computational code for modeling variations of architecture family features. The latter is described using Java code semantics. Thus, the template is expressed in terms of language independent tag semantics mixed with native code (*i.e.* Verilog HDL) and the Java code is not mixed with the Verilog code. This separation is also possible when using EP3, to some extent. However, we offer the possibility to perform some Verilog HDL code analysis and to extend the tool analysis features. Other approaches, such as Chiesel, have relied on application specific languages [70] (Scala-based). To sum up, with respect to prior work, our approach provides a complete environment, covering all aspects from script calls and automation, to the description and verification of a family of LDPC decoders, providing the necessary means for fine design optimizations through the template design approach. Therefore, we have the means to

mine for a family of LDPC decoder architectures, with fine hand-made optimizations embedded in an automated process.

Semantically, a tag is going to be replaced by a piece of text (e.g., Verilog code) during the customization process performed by the application. The actual text is produced by a function (having the name specified by the identifier). Such a function is called customization function and the user can implement her own functions.

TeDi offers support for built-in tags such as: repetition, conditional, expression, replacement, external tool integration, template hierarchy, and the possibility for user defined tags for some application specific computations. The code analysis and Verilog-dependent tags addressing the following issues: To be more precise we targeted the following potential problem cases:

- detect an incompatible width of a vector wire connected to a vector port in a module instantiation statement
- repeated usage of the same part of a vector wire several times when connecting the wire to a port in a module instantiation statement
- unconnected parts of a vector wire connected to a port in a module instantiation statement

Warnings are issued, and possible copy-paste errors are identified.

### 2.4.3 A case study

This environment has been first validated for the architecture from Fig. 2.8, using different codes and parallelization degrees. Synthesis results are presented in Table 2.5 for Xilinx FPGA technology. The layered LDPC decoder architectures have yielded higher throughput than some of the hand-made LDPC decoder solutions from state-of-the art [12][13] and comparable cost. Note, that the bank allocation problem is solved using the [15] algorithm, while the pipeline problem can be solved by either re-arranging the access order of messages as suggested in [15], or by using multiple codewords decoding as presented in [3].

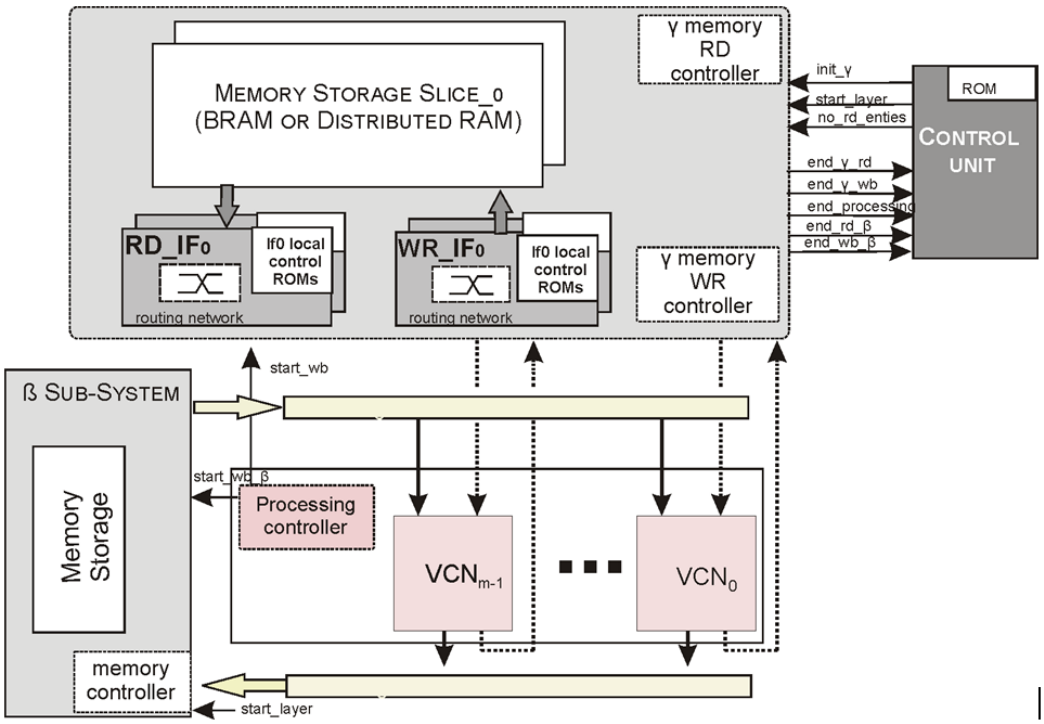



Figure 2.8: QC-LDPC merged VCN generic architecture customized by the unit parallelism parameter P, message quantization, MS variant, and $\beta$ message storage sub-system type (*i.e.* compressed or uncompressed message storage). The number of processing units is equal to the circulant size. [5]

Table 2.5: Synthesis results for the Xilinx Virtex-7 xc7vx485t FPGA device codes: WiMAX rate $\frac{1}{2}$ irregular code, dv=3, rate $\frac{1}{2}$ regular code, dv=4 rate $\frac{1}{2}$ regular code and dv=4, rate $\frac{3}{4}$, having coded bits is 2304 for WiMAX and 1296 for the regular codesand $z$ equal to 96 for WiMAX, 54 for rate $\frac{1}{2}$ regular codes and 27 rate $\frac{3}{4}$ regular codes, with message quantization is (4,6) [5].

| Code | Cost (LUT-FF pairs, BRAMs) | Freq [MHz] | T[coded] Mbps |
|---|---|---|---|
| Regular dv3 [r1/2, P=1] | 14185 pairs, 9 BRAM | 246 | 425 |
| Regular dv3 [r1/2, P=3] | 36832 pairs, 20 BRAM | 234 | 1084 |
| Regular dv3 [r1/2, P=6] | 77753 pairs, 66 BRAM | 235 | 2030 |
| Regular dv4 [r1/2, P=1] | 16249 pairs, 8 BRAM | 226 | 300 |
| Regular dv4 [r1/2, P=4] | 59637 pairs, 25 BRAM | 190 | 912 |
| Regular dv4[r3/4, P=1] | 9963 pairs, 6 BRAM | 225 | 295 |
| Regular dv4[r3/4, P=4] | 32713 pairs, 24 BRAM | 219 | 1051 |
| WiMAX [r1/2, P=1] | 28455 pairs, 16 BRAM | 240 | 630 |
| WiMAX [r1/2, P=2] | 56982 pairs, 24 BRAM | 218 | 1116 |
| WiMAX [r1/2, P=3] | 91050 pairs, 37 BRAM | 178 | 1318 |

## 2.5 Conclusions

This section presents we have presented the contributions on the hardware architecture side of the layered scheduling LDPC decoder implementations.We discuss the baseline architectures used in the rest of this thesis. The parameters and proposed improvements are also briefly presented. We close this

section with a brief discussion of the framework for automating design space exploration for hardware architectures in general, and for LDPCs in particular. We exploit this environment for all decoder variation evaluations discussed in subsequent sections.

# Chapter 3

# Imprecise Computation and Approximate Storage for Layered Scheduling LDPC Decoders

**Abstract:** *This chapter discusses the approximate computation techniques designed for layered QC-LDPC decoders. Although at first the prime targets are processing elements, later we have discovered they impact storage blocks as well. Thus, for partially-parallel decoder architectures, and for ASIC technology implementations, the gain in approximate memory storage is comparable, and in some cases larger than the one for the processing units. Therefore, we conclude that by jointly using approximate computation and storage, we can effectively optimize cost and throughput for the LDPC decoder architecture. Another important aspect touched down in this chapter is the issue of stopping the decoder.*

## 3.1 Introduction

In the era of information, where a plethora of existing and emerging services and products demanding connectivity, and having different communication requirements, efficient forward error correction (FEC) solutions are in demand. Due to their structure, making possible different degrees of parallelization, QC-LDPC have gained momentum and have been used in wireless standards (WiFi [20], WPAN [10], WiMAX [7]), digital video broadcasting (DVB [21]) and flash memories [22]. In this chapter, we address the generic layered decoders described in the previous chapter, and try to exploit their robustness to noise in order to obtain area and/or throughput and/or power consumption improvements with respect to the baseline. Thus, we consider both partially-parallel and very high throughput hardware architectures for our assessment decoding. Furthermore, as discussed subsequently the proposed approaches.

This chapter focuses on optimization techniques based on imprecise computation and message storage, as well as imprecise early decoding stopping criterion. These contributions can be summarized as follows:

- Memory efficient approximate version for Self-Correcting Min-Sum (SCMS) LPDC decoding; results corresponding to this contribution have been disseminated in the following papers:

  O. Boncalo, A. Amaricai, P. F. Mihancea, and V. Savin, "Memory trade-offs in layered self-corrected min-sum LDPC decoders," *Analog Integrated Circuits and Signal Processing*, vol. 87,

no. 2, pp. 169–180, May 2016. [Online]. Available: https://doi.org/10.1007/s10470-015-0639-3

O. Boncalo, A. Amaricai, and V. Savin, "Memory efficient implementation of self-corrected min-sum ldpc decoder," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2014, pp. 295–298.

- Imprecise one-hot based implementation of check-node unit based computation; results corresponding to this contribution have been disseminated in the following journal paper:

  O. Boncalo, A. Amaricai, V. Savin, D. Declercq, and F. Ghaffari, "Check node unit for LDPC decoders based on one-hot data representation of messages," *Electronics Letters*, vol. 51, no. 12, pp. 907–908, 2015.

- Efficient architectures for Non-Surjective Finite Alphabet Iterative Decoder (NS-FAID) LDPC decoding; results corresponding to this contribution have been disseminated in the following papers:

  T. Nguyen-Ly, K. Le, F. Ghaffari, A. Amaricai, O. Boncalo, V. Savin, and D. Declercq, "Fpga design of high throughput LDPC decoder based on imprecise offset min-sum decoding," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, June 2015, pp. 1–4.

  T. T. Nguyen-Ly, K. Le, V. Savin, D. Declercq, F. Ghaffari, and O. Boncalo, "Non-surjective finite alphabet iterative decoders," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

  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, March 2018

  O. Boncalo, V. Savin, and A. Amaricai, "Unrolled layered architectures for non-surjective finite alphabet iterative decoders," in *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, Oct 2017, pp. 1–5.

- In-between layer based stopping criterion for layered LDPC decoders; results corresponding to this contribution have been disseminated in the following papers and patent:

  A. Hera, O. Boncalo, C. Gavriliu, A. Amaricai, V. Savin, D. Declercq, and F. Ghaffari, "Analysis and implementation of on-the-fly stopping criteria for layered qc-ldpc decoders," in *2015 22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES)*, June 2015, pp. 287–291.

  D. Declercq, V. Savin, O. Boncalo, and F. Ghaffari, "An imprecise stopping criterion based

on in-between layers partial syndromes," *IEEE Communications Letters*, vol. 22, no. 1, pp. 13–16, Jan 2018.

V. Savin, O. Boncalo, and D. Declercq, "Stopping criterion for decoding quasi-cyclic ldpc codes," *European Patent Office EP3373488A1*

- Some other miscellaneous contributions for imprecise computation QC-LDPC decoder architectures:

K. Le, D. Declercq, F. Ghaffari, L. Kessal, O. Boncalo, and V. Savin, "Variable-node-shift based architecture for probabilistic gradient descent bit flipping on qc-ldpc codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 7, pp. 2183–2195, July 2018.

O. Boncalo, A. Amaricai, and S. Nimara, "Memory-centric flooded ldpc decoder architecture using non-surjective finite alphabet iterative decoding," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, Aug 2018, pp. 104–109.

O. Boncalo, "Qc-ldpc gear-like decoder architecture with multi-domain quantization," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 244–251

This chapter is organized as follows: Section 3.2 discusses the SCMS imprecise erasure bits computation rules, as well as a bunch of memory optimizations for compressed CNU ($\beta$) message storage, Section 3.3 discusses the Non-surjective Finite Alphabet Iterative Decoder (NS-FAID) framework for imprecise memory message storage, Section 3.4 discusses imprecise early termination criteria that can be computed during a layer decoding processing with minimum hardware resources and no memory overhead, Section 3.5 discusses the on-hot encoding suitable for high code rate CNU comparator processing, last some other imprecise computation contributions for flooding scheduling QC-LDPC decoders and other decoding algorithms besides MS-based decoding. In the ending of this chapter, in section 3.6, we briefly introduce some additional contributions for flooding scheduling decoders using either MS or PGDBF decoding algorithms.

## 3.2 Modified SCMS [1]

The MS algorithm is favored for hardware implementations since the processing units are built out of simple additions and comparisons on messages which are represented using a small number of bits (*i.e.* 3, 4, 5, 6 bits). The SCMS algorithm is a variant of the MS algorithm that improves decoding performance by exploiting the concept of unreliable messages [50]. These messages have their magnitude reduced, while the sign is preserved. The criterion for flagging the variable node messages ($\alpha$) as unreliable is the condition that its sign changes between two consecutive iterations. In this case, an erasure flag is set and the message is "erased", meaning that its amplitude is reduced. Typically this reduction is made to zero. Thus, the decoding advantage of SCMS comes at the price of the erasure bit computation, as well as two additional bits per message stored: the erasure flag, and

[1] This subsection contains results and text partially reproduced from the following papers [3][6]

the sign of the variable node from the previous iteration, on the storage side. This of-course is a course rain implementation, where the memory overhead, without the usage of compressed messages is of 50% for four bit message quantization. Given, the large number of messages (*i.e.*, in the thousands) this additional cost is non-negligible

Variable node computation for SCMS algorithms [3]:

$$\alpha_{i,j}^{new} = \gamma_i^{old} - \beta_{i,j}^{old}, \tag{3.1a}$$

$$e_{i,j}^{new} = (\sim e_{i,j}^{old}) \cdot (s_{i,j}^{new} \oplus s_{i,j}^{old}), \tag{3.1b}$$

$$\alpha_{i,j}^{SC} = (e_{i,j}^{new} = 1) ? \ 0 : \ \alpha_{i,j}^{new}, \tag{3.1c}$$

The contribution for memory efficient SCMS decoder implementation addresses the reduction of the storage requirement overhead of the erasure bits. For this purpose, two variants have been proposed:

- SCMS-V1 – eliminates the need for check node messages' signs storage.
- SCMS-V2 – is based on a novel imprecise self- correction rule, which allows the reduction of the erasure bits.

These improvements have first been proposed in [6]. The first one discards the check- node message signs, and only stored variable- node message signs and erasure bits. The second discards the erasure bits, as well, and tries to approximate them by considering all messages with magnitude zero as erased messages during the previous iteration execution. Note the for SCMS-V2, the memory requirements are the same as for MS. The work from [6] discusses results for WiMAX (1152, 2304) code, having message quantizations of three bits for check-node and variable node, and 5 bits for AP-LLR, – (3,5). The FPGA implementation results revealed that the performance of the modified SCMS and the MS decoder implementations are approximately the same. In terms of decoding performance, the modified SCMS has a performance degradation of approximate 0.1 dB lower with respect to baseline SCMS, while having an improvement of approximatively 0.5 dB with respect to MS.

SCMS-V1 relies on the storage of only the variable-node message signs - $sign(\alpha_{i,j}^{old})$ -, and recomputing at the beginning of the current iterations the signs of the old check-node message - $sign(\beta_{i,j}^{old})$ -. This way, storage of $sign(\beta_{i,j}^{old})$ is no longer required. The variable node computation associated to SCMS-V1 becomes as following [3, 6]:

Variable node computation for SCMS-V1 algorithm [3, 6]:

$$sign(\beta_{i,j}^{old}) = \oplus s_{i,k}^{old}) \tag{3.2a}$$

$$\alpha_{i,j}^{new} = \gamma_i^{old} - \beta_{i,j}^{old}, \tag{3.2b}$$

$$e_{i,j}^{new} = (\sim e_{i,j}^{old}) \cdot (s_{i,j}^{new} \oplus s_{i,j}^{old}), \tag{3.2c}$$

$$\alpha_{i,j}^{SC} = (e_{i,j}^{new} = 1) ? \ 0 : \ \alpha_{i,j}^{new}, \tag{3.2d}$$

Thus, the stored check-node message is comprised of its corresponding magnitude, the sign of the variable-node message and erasure bits. SCMS-V1 does not represent an algorithmic change in the SCMS based LDPC decoding. Therefore, the error correction capability associated to it is the same as the conventional SCMS. The single difference of SCMS-V1 is represented by the re-computation

of the signs of the check-node message at the beginning of the current iteration, using XOR based operations.

Regarding SCMS-V2, it targets the additional memory footprint improvement, by removing the requirement for erasure bits storage. This reduction in the memory word associated to the check-node message is performed by approximating the erasure bit based on the magnitude of the check-node message: if the magnitude of the $\beta_{i,j}^{old}$ is equal to zero, we assume that an erasure has taken place during the previous iteration, and therefore, we consider a valid erasure bit. The variable node computation corresponding to SCMS-V2 becomes as follows:

Variable node computation for SCMS-V2 algorithm [3, 6]:

$$sign(\beta_{i,j}^{old}) = \oplus s_{i,k}^{old}), \tag{3.3a}$$

$$\alpha_{i,j}^{new} = \gamma_i^{old} - \beta_{i,j}^{old}, \tag{3.3b}$$

$$e_{i,j}^{old} = (\beta_{i,j}^{old} = 0)?\ 1\ :\ 0, \tag{3.3c}$$

$$e_{i,j}^{new} = (\sim e_{i,j}^{old}) \cdot (s_{i,j}^{new} \oplus s_{i,j}^{old}), \tag{3.3d}$$

$$\alpha_{i,j}^{SC} = (e_{i,j}^{new} = 1)?\ 0\ :\ \alpha_{i,j}^{new}, \tag{3.3e}$$

Regarding a layered LDPC decoding architecture, with a compressed word used for check-node message storage, consisting of the index of the first minimum, magnitude of the first minimum - $\beta_{min1}$ -, magnitude of the second minimum - $\beta_{min2}$ -, and the signs, the estimation corresponding to an erasure operation in the previous iteration is based on whether the first minimum is equal to 0. Therefore, for architectures using compressed word for check-node message storage, the variable node computation becomes:

Variable node computation for SCMS-V2 algorithm with compressed check node message storage [3, 6]:

$$sign(\beta_{i,j}^{old}) = \oplus s_{i,k}^{old}), \tag{3.4a}$$

$$\alpha_{i,j}^{new} = \gamma_i^{old} - \beta_{i,j}^{old}, \tag{3.4b}$$

$$e_{i,j}^{old} = (|\beta_{min1} = 0|)?\ 1\ :\ 0, \tag{3.4c}$$

$$e_{i,j}^{new} = (\sim e_{i,j}^{old}) \cdot (s_{i,j}^{new} \oplus s_{i,j}^{old}), \tag{3.4d}$$

$$\alpha_{i,j}^{SC} = (e_{i,j}^{new} = 1)?\ 0\ :\ \alpha_{i,j}^{new}, \tag{3.4e}$$

Figure 3.1 presents the memory word reduction for the SCMS-V1 and SCMS-V2 with respect to conventional SCMS memory word, when compressed check-node message storage is employed. Sizes for different check-node degrees of compressed check-node messages, for MS, SCMS, SCMS-V1, and SCMS-V2 are depicted in Table 3.1. Figure 3.1 and Table 3.1 show that for the SCMS-V2, the compressed memory word has the same size as in the case of MS LDPC decoding, with improvements between 44-56% with respect to the conventional SCMS. Furthermore, SCMS-V1 has reduced memory word between 22% and 28% compared with conventional SCMS, with the same error correction capability, as no change in the algorithm is performed.

Implementation of the two proposed versions of SCMS - SCMS-V1, SCMS-V2 - requires the modification of processing units within the LDPC decoders, according to equations 3.2 and 3.4. We will detail the modification of the processing units corresponding to a layered LDPC decoding architecture,

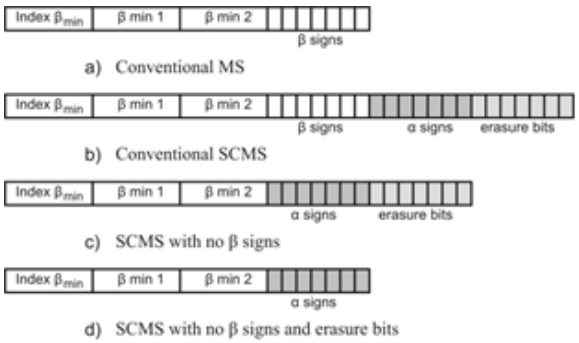



Figure 3.1: Compressed check-node message format and size for a $d_c = 7$ LDPC code, as exemplified in [6]: (a) for MS, it consists of index of first minimum, magnitude of first minimum, magnitude of second minimum, and the signs of check-node messages; (b)for conventional SCMS, it consists of index of first minimum, magnitude of first minimum, magnitude of second minimum, signs of check-node messages, signs of variable-node messages, and erasure bits; (c) for SCMS-V1, it consists of index of first minimum, magnitude of first minimum, magnitude of second minimum, signs of variable-node messages, and erasure bits; (d) or SCMS-V2, it consists of index of first minimum, magnitude of first minimum, magnitude of second minimum, and the signs of variable-node messages

Table 3.1: Size of the compressed check-node message words for different values of check-node degree $d_c$ used in MS, conventional SCMS, SCMS-V1 and SCMS-V2, as exemplified in [3] ; a quantization of 4 bits has been considered for check-node representation;

| Code | $d_{c\_max}$ | MS | SCMS | SCMS-V1 | SCMS-V2 |
|---|---|---|---|---|---|
| Regular dv3 rate 1/2 | 6 | 15 | 27 | 21 | 15 |
| Regular dv3 rate 2/3 | 9 | 19 | 37 | 28 | 19 |
| Regular dv3 rate 3/4 | 12 | 22 | 46 | 34 | 22 |
| Regular dv3 rate 5/6 | 18 | 29 | 65 | 47 | 29 |
| WiMAX rate 1/2 | 7 | 16 | 30 | 23 | 16 |
| WiMAX rate 2/3 | 10 | 20 | 40 | 30 | 20 |
| WiMAX rate 3/4 | 15 | 25 | 55 | 40 | 25 |
| WiMAX rate 5/6 | 20 | 31 | 71 | 51 | 31 |

with compressed check-node message storage and merged VCN. These modifications are depicted in Figures 3.2, 3.3, 3.4, and detailed in [3]:

- The VCN for conventional SCMS contains an erasure detection logic - marked in Figure 3.2 with red circle - and a multiplexer, which, based on the result of the erasure detection logic, performs the amplitude reduction of the variable-node message;
- The VCN for SCMS-V1 contains a regeneration logic, consisting of XOR gates, which recomputes the sign of the check-node messages from the previous iteration based on the sign of the stored variable node message; the erasure detection logic remains the same as in the case of the conventional SCMS - Figure 3.3;
- The VCN for SCMS-V2 contains an additional erasure approximation logic, which estimates the erasure bits, as presented in equations (3.4) - Figure 3.4; the regeneration of the check-node messages based from the variable-node message stored in memory, and the erasure detection

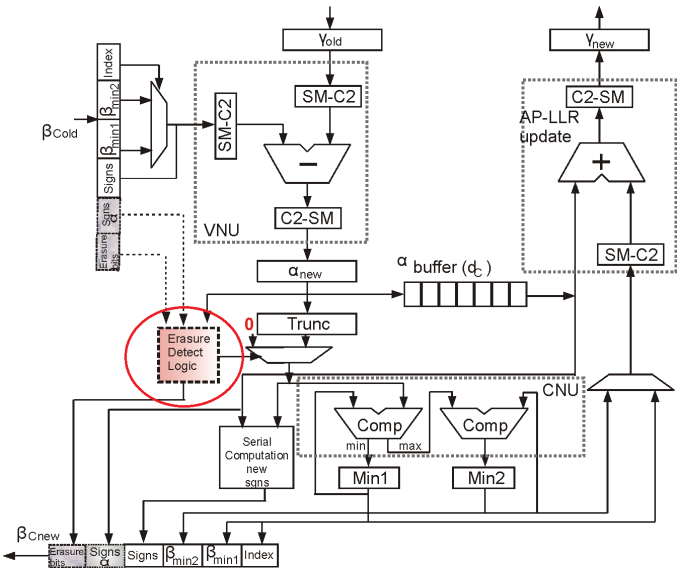



Figure 3.2: Merged VCN unit for conventional SCMS based layered LDPC decoder, as in [3]; the erasure detection logic represents the main difference with respect to the VCN unit corresponding to MS

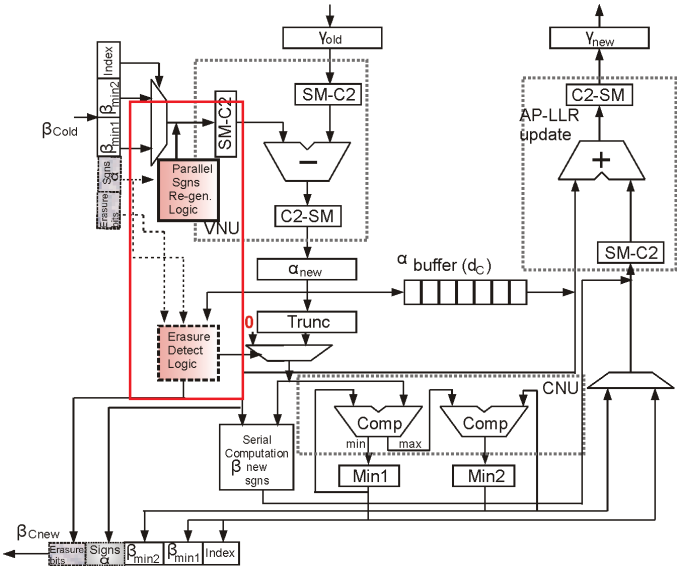



Figure 3.3: Merged VCN unit for SCMS-V1 based layered LDPC decoder, as in [3]; the merged VCN unit of the SCMS-V1 has an additional XOR based re-computation block of the check-node message signs

logic remains identical as in the VCN of the SCMS-V1.

Because SCMS-V2 represents an approximation based modification of the conventional SCMS algorithm, error correction capability may be affected. We have performed analysis in terms of BER for MS, conventional SCMS - same as SCMS-V1 -, and SCMS-V2. The analysis has been performed for 8 LDPC codes: 4 regular LDPC codes with variable node degree $d_v = 3$, rates $1/2, 2/3, 3/4, 5/6$ and 4 irregular WiMAX codes, rates $1/2, 2/3, 3/4, 5/6$. The codeword size is 2304 bits, with expansion factor $z = 96$, with considered quantizations $(6, 4)$ and $(5, 3)$. Decoding performance curve are depicted in Figures , for irregular codes, and in Figures , for regular codes. These results are detailed in [3]. These figures indicate that the SCMS-V2 has only a slight decoding performance degradation with respect

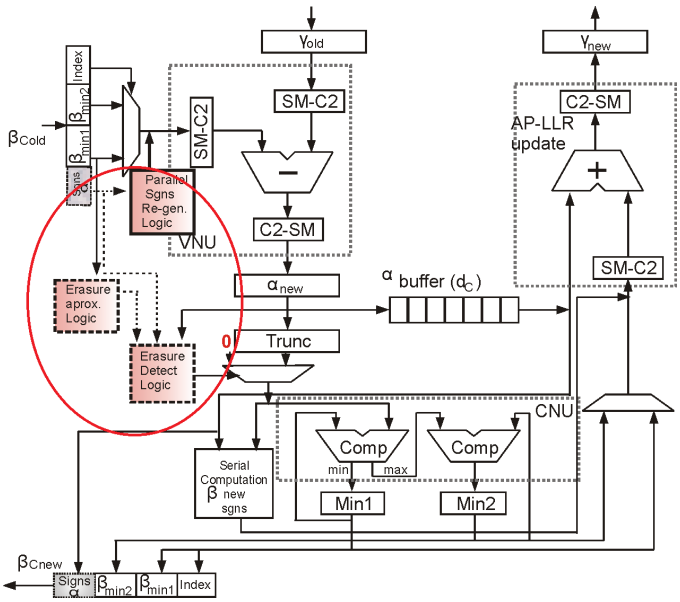



Figure 3.4: Merged VCN unit for SCMS-V2 based layered LDPC decoder, as in [3]; the merged VCN unit of the the SCMS-V2 contains an additional erasure estimation logic

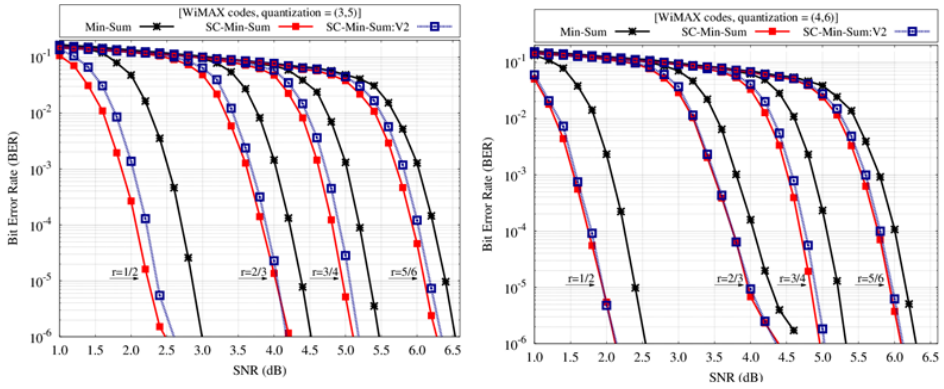



Figure 3.5: Bit-Error-Rate curves of MS, conventional SCMS, and SCMS-V2 algorithms for irregular WiMAX LDPC codes: quantization $(5, 3)$ left, and quantization $(6, 4)$ right - as presented in [3]

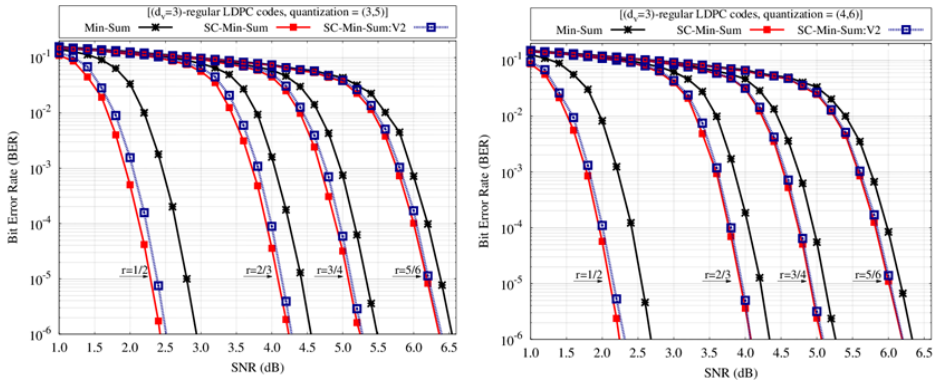



Figure 3.6: Bit-Error-Rate curves of MS, conventional SCMS, and SCMS-V2 algorithms for regular $d_v = 3$ LDPC codes: quantization $(5, 3)$ left, and quantization $(6, 4)$ right - as presented in [3]

to conventional SCMS, with less than 0.1 dB degradation in the waterfall region. With respect to the MS, both SCMS versions present a significant improvement in the error correction capability, with 0.15 dB to 0.5 dB improvement for regular codes, and between 0.3 dB to 0.6 dB for WiMAX irregular

LDPC codes.

We have evaluated the four LDPC decoding algorithms - MS, conventional SCMS, SCMS-V1, SCMS-V2 - regarding the cost of their architecture implementation. We have synthesized for FPGA technology a number of 64 decoders, with the following parameters:

- 8 LDPC codes: 4 regular codes with variable node degree $d_v = 3$ and 4 irregular WiMAX codes; all the 8 LDPC codes have a codeword size of 2304 bits;
- 2 quantizations: $(6, 4)$ and $(5, 3)$
- 4 decoding methods: MS, conventional SCMS, SCMS-V1, SCMS-V2

The Verilog HDL source code for all of the 64 LDPC decoding architectures have been generated using the LDPC decoding generator tool TEDI [5], which has been described in detailed in previous chapter. The synthesis process has been performed using Xilinx ISE 14.7 tool for a Xilinx Virtex-7 Virtex-7 VX485T, speed grade-2, device. The corresponding results, presented in [3], detail the logic utilization of the 64 LDPC decoders, expressed in LUT-FF pairs, and the memory utilization, expressed in the number of used BRAM memory blocks. Regarding logic resource consumption, the architectures corresponding to MS and SCMS-V2 LDPC decoders have similar cost, and with up to 33% improvement with respect to SCMS-V1. Regarding SCMS-V1, it present a cost reduction of up to 20% with respect to conventional SCMS. Regarding memory utilization, the LDPC decoders implementing MS and SCMS-V2 have similar cost, with an improvement of up to 39% with respect to conventional SCMS; SCMS-V1 has a reduced cost compared to conventional SCMS of up to 21%.

Therefore, the introduction of a novel imprecise erasure approximation rule for Self-Corrected Min-Sum LDPC decoding algorithm has led to significant cost reduction in the hardware architecture corresponding to a layered LDPC decoder, while having only a slight error correction capability degradation, of less than 0.1 dB.

## 3.3 NS-FAID

### 3.3.1 Overview and General Idea

Finite Alphabet Iterative Decoder (FAID) has been introduced by Planjery, Declercq and Vasic [71], as a variable-node update method targeted at breaking harmful cycles in LDPC decoders, called trapping sets. FAID represents a generalization of offset min-sum LDPC decoding that relies on specific look-up tables in the variable-node processing; the main advantage of FAID decoders is represented by improved performance in the error floor region, making them suitable for non-volatile storage devices. The first versions of FAID decoders have been developed for regular LDPC codes with variable node degree $d_v = 3$, and use quantizations of 3 bits for both check-node messages and variable-node messages.

A prior development of the Non-Surjective Finite Alphabet Iterative Decoder (NS-FAID) is represented by Partially Offset Min-Sum (POMS) and the Imprecise Partially Offset Min-Sum (I-POMS), presented in [27]. Regarding POMS, it performs the offset subtraction associated to the offset MS only when the message is odd. Therefore, the check nude message for POMS becomes as follows [27]:

$$\beta^*_{r,t(r,n)} = \prod_{n' \in N(m)\setminus\{n\}} sign(\alpha_{r,t(r,n')}) \times \min_{n' \in N(m)\setminus\{n\}} \alpha_{r,t(r,n')}; \tag{3.5a}$$

$$\beta_{r,t(r,n)} = \begin{cases} \beta^*_{r,t(r,n)}; & LSB\left(\beta^*_{r,t(r,n)}\right) = 0 \\ \beta^*_{r,t(r,n)} - 1; & LSB\left(\beta^*_{r,t(r,n)}\right) = 1 \end{cases}; \tag{3.5b}$$

Therefore, the least significant bit of the newly computed check-node message will always equal to 0. Thus, it does not need storage of this bit in the check-node message memory: if a quantisation of 4 bits for the check-node message is employed, only 3 bits will be stored in the memory. Results presented in [27] for a layered decoding architecture for a regular $(3, 6)$ LDPC code using fully parallel processing units and uncompressed check-node message indicate that a memory saving of 25% with respect to a MS or Offset MS implementation is obtained. Regarding the error correction capability, the POMS decoder has decoding performance between the MS and Offset MS.

Non Surjective - Finite Alphabet Iterative Decoding (NS-FAID) is a method that builds on the conventional MS, while performing a tight fitting on the message storage based on an off-line analysis using the density evolution method. In a sense it can be regarded as a generalization of the normalized min-sum, offset min-sum methods, and later the dual-domain quantization min-sum [28]. In the following we will briefly discus the NS-FAID concept.

NS-FAID extends the concept of FAID presented in [71]; the main optimization target for NS-FAID is represented by the reduction of quantisation the variable-node messages, and thus, reducing memory and routing logic footprint, without degradation of the error correction capability of the LDPC decoder. Formally a $(2Q+1)$-level FAID, with Q being a positive integer, has been defined in [16] as a 4-tuple$(\mathcal{M},\mathcal{I},\Phi_v,\Phi_c)$, where:

- $\mathcal{M} = \{-Q, \ldots, -1, 0, +1, \ldots, +Q\}$ is the alphabet of the exchanged messages, also referred to as the decoder alphabet,
- $\mathcal{I} \subseteq \mathcal{M}$ is the input alphabet of the decoder, *i.e.*, the set of all possible values of the quantized soft information inputted to the decoder,
- $\Phi_v$ and $\Phi_c$ denote the update rules for variable-nodes and check-nodes,

respectively.

As mentioned before the CN-update function remains unchanged, while $\Phi_v : \mathcal{I} \times \mathcal{M}^{d_v-1} \to \mathcal{M}$, becomes for a VN of degree $d_v$ as [16]:

$$\Phi_v(\gamma, m_1, \ldots, m_{d_v-1}) = F\left(\gamma + \sum_{j=1}^{d_v-1} m_j\right) \tag{2}$$

NS-FAID is a FAID with a non surjective framing function $F\colon Z \to \mathcal{M}$ [16][8]. The main idea is to have the image set of $F$, denoted by $Im(F) \subset mathcalM$, a strict subset of $\mathcal{M}$. This would lessen the storage requirements for the exchanged messages, as well as reduce of the size of the interconnect network between memory and the processing units. To sum up, the basic idea is to have two working domains in terms of quantization: a higher domain for the VNU process and the AP-LLR update, an

a lower domain for the CNU processing. The crossing between the two domains is made by means of the so-called framing and de-framing functions. From the memory storage perspective, these work as compression/decompression tables. The CN processing is the same for any FAID decoder, however, the message quantization, as discussed earlier is lower than in the VN processing case.

### 3.3.2 Theoretical analysis

Density evolution (DE) has been performed using the method described in [16][8] and the use cases considered are for both irregular WiMAX LDPC codes with rate 1/2 [7], and regular NS-FAIDs are optimized for $(d_v = 3, d_c = 6)$-regular LDPC codes [8], and for for $(d_v = 3, d_c = 12)$-regular LDPC codes [9]. For irregular codes, the results for the MS decoder (indicated as NS-FAID-444), as well as NS-FAID decoders have been presented in [16], and are shown in Table 3.2, with the framing functions' LUTs of the best NS-FAID-$w_2w_3w_6$ described in Table 3.3 (`Lx` in Table 3.3 corresponds to `LUTx` in Table 3.2). For regular codes, Table 3.4 summarizes the 10 best NS-FAIDs optimized using DE for $(3, 12)$-regular LDPC codes [9].

Decoding performance curves for WiMAX rate 1/2 code and Additive White Gaussian Noise (AWGN) channel are depicted in Fig. 3.7 for some selected NS-FAID decoders from Table 3.2 [8]. Simulations from Fig. 3.7, show that the NS-FAID-433 and the NS-FAID-432 decoders outperform the MS decoder by 0.3 dB and 0.15 dB (at BER=$10^{-5}$), the NS-FAIDs-333 decoder improves the BER performance by 0.12 dB, and the NS-FAIDs-332 exhibits similar BER performance, while the NS-FAID-222 decoder, which is the most implementation efficient, exhibits a significant BER degradation of $\approx 1$ dB.

### 3.3.3 Implementation results considerations

For the hardware implementation side integrating an NS-FAID kernel boils down to placing framing and de-framing Look-up Tables at the input and output of variable node processing, as well as de-framing LUT for AP-LLR update as depicted in Fig. 3.8 [9]. Therefore, we conclude that the 3 architecture blocks that are changed due to NS-FAID integration are: check-node $\beta$-memory block, saturation and framing with two's complement to sign-magnitude conversion (SAT/FRA), de-framing with merged sign-magnitude to two's complement conversion /DE-FRA, and the CNUs. The $\beta$-memory block and CNUs are changed in the sense that the number of message bits is lower than the one considered for the baseline MS decoder –NS-FAID-444 presented in Fig. 3.9. In order to assess the impact of integrating the NS-FAID concept into the VLSI design, we have made the component-level comparisons of NS-FAID implementations in [8]. Area results have been provided for the 3 aforementioned blocks for maximal decoder operating clock frequency. Note that for NS-FAID-2 and NS-FAID-3 decoders, the $\beta$-memory block occupies 49:7% and 73:5%, respectively, as compared to NS-FAID-444. Gains even more significant are obtained for the CNU block, of 24:4% and 33:3%, respectively, as compared to NS-FAID-444.

The concept of NS-FAID has proven its advantages for the unrolled layered decoding architectures detailed in [4]. The unrolled layer based LDPC decoder using NS-FAID decoding has been detailed in [9]. As the architecture presented in [8], the unrolled layer LDPC architecture employs fully parallel processing units. For the decoder depicted in [9], a merged VCN processing unit has been employed. This unit contains the following [9]:

- **Variable Node Processing** - in this processing unit, the check node $\beta$ messages are read from the local shift register implemented check-node memory, that are stored in framed form - using a small quantization - ; these messages are represented using sign-magnitude representation. The arithmetic operation required are the subtraction of $\beta$ messages from the AP-LLR $\tilde{\gamma}$, which is done in full precision. Therefore, a de-frame operation on the the $\beta$ messages is required. Furthermore, the conversion from sign-magnitude to two's complement format is also required. NS-FAID rules are employed for the implementation of the de-frame look-up tables. Furthermore, in order to reduce the delay in the critical path, the de-frame and the conversion from sign-magnitude to two's complement are merged into a single function, implemented using a single look-up table.

- **Check-node processing** - Check-node processing requires comparison of the absolute values of newly updated check-node messages $\alpha$, which, according to the NS-FAID concept, can be performed in the low quantization domain. In order to do the corresponding arithmetic operations in the low quantisation domain, framing on the $\alpha$ messages is necessary. Furthermore, the comparisons within the check-node unit required absolute values, so conversions from two's complement to sign-magnitude needs to be performed. As in the variable-node processing, the framing function and the integer format conversion are merged into a single format, and thus, a single look-up table for it is employed. The resulting check-node messages $\beta$ are stored in the memory in the low-quantisation domain, in the framed form.

- **AP-LLR update** - The processing according to the AP-LLR update require the addition, in full precision and using two's complement format, of the newly updated variable-node message and the newly updated check-node message, in order to obtain the AP-LLR message. Therefore, a de-frame operation, as well as conversion from sign-magnitude to two's complement, on the check-node message $\beta$ is required. As in the case of variable-node processing and check-node processing, these operations are combined in a single look-up table.

An unrolled layered LDPC decoder, for a regular $d_v = 3$ array LDPC code, has been implemented [9], using NS-FAID based update in the processing units. For this code, the best NS-FAID tables are depicted in Table 3.4. These tables have been obtained using the theoretical analysis based on density evolution, described in [16]. The synthesis results for FPGA technology, using a Xilinx ISE tool, and Xilinx Virtex-7 FPGA device, for both $w = 2$ and $w = 3$ indicate throughput-to-area ratio of between to 20% and up to 125% with respect to a MS decoder that employs a quantisation of 4 bit for the check-node message. Thus, NS-FAID based LDPC decoding is highly appropriate for the unrolled layer architecture, with improvements in both cost and throughput, while having increased - for $w = 3$ - or slightly degraded - for $w = 2$ -, error correction capability with respect to the MS decoding.

## 3.4 Early termination criteria for Layered scheduling decoders[2]

### 3.4.1 Imprecise on-the-fly criteria for early decoding termination decoders[3]

[2]This subsection contains results and text partially reproduced from the conference paper [12], journal publication [13], and patent Application[72]

[3]This subsection contains results and text partially reproduced from the conference paper [12]

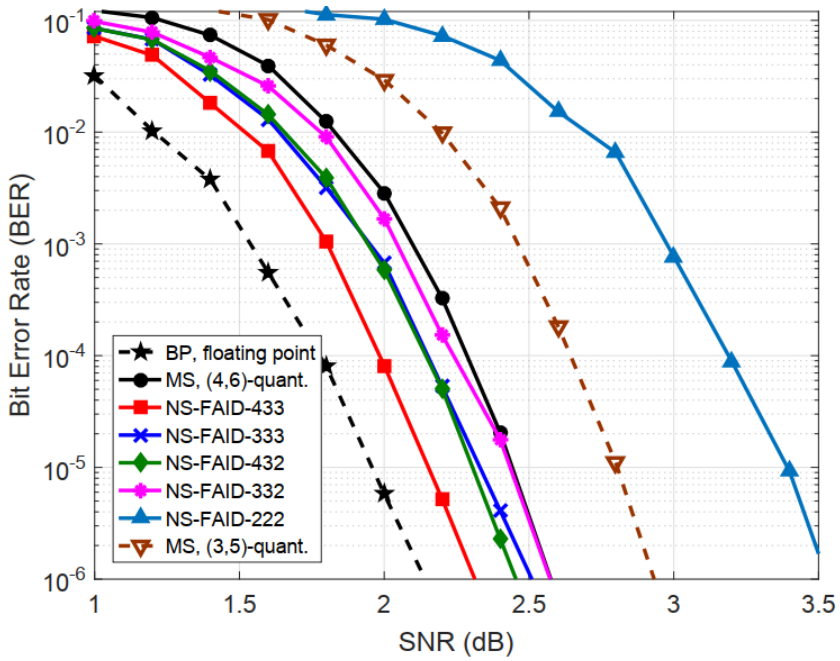



Figure 3.7: Bit Error Rate (BER) statistic Monte-Carlo simulation curves for the irregular WiMAX code of rate 1/2, with base matrix of size 12 × 24 [7], and expansion factor z = 96, thus resulting in a codeword length of 2304 bits for selected NS-FAID decoders from table 3.2, as in [8]

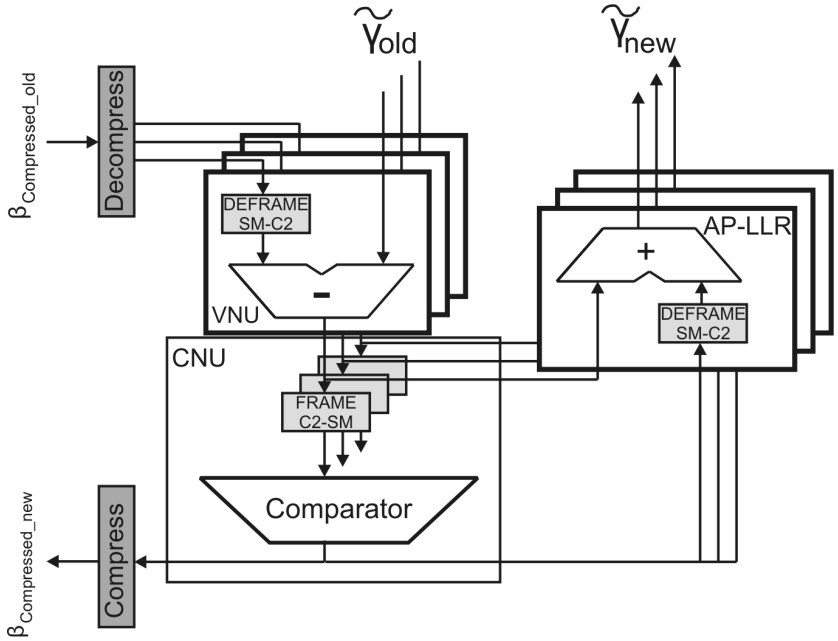



Figure 3.8: VNU overview for NS-FAID operation with framing and de-framing LUTS, as in [9]. For increased implementation cost efficiency the sign-magnitude to two's complement conversions required are merged as well.

An important aspect to QC-LDPC decoder implementations is the stopping rule used to stop a codeword decoding. Two practical cases exist: (i) decode for a fixed number of iterations (ii) stop the decoding based on the decoder state – the conventional approach is to check for the syndrome (i.e. all the parity checks are satisfied for all the equations), or if the maximum allowed number of iterations is reached in the situation when the syndrome checks is not satisfied after a predefined, fixed number of iterations. The later is especially convenient since the average number of iterations for decoding a

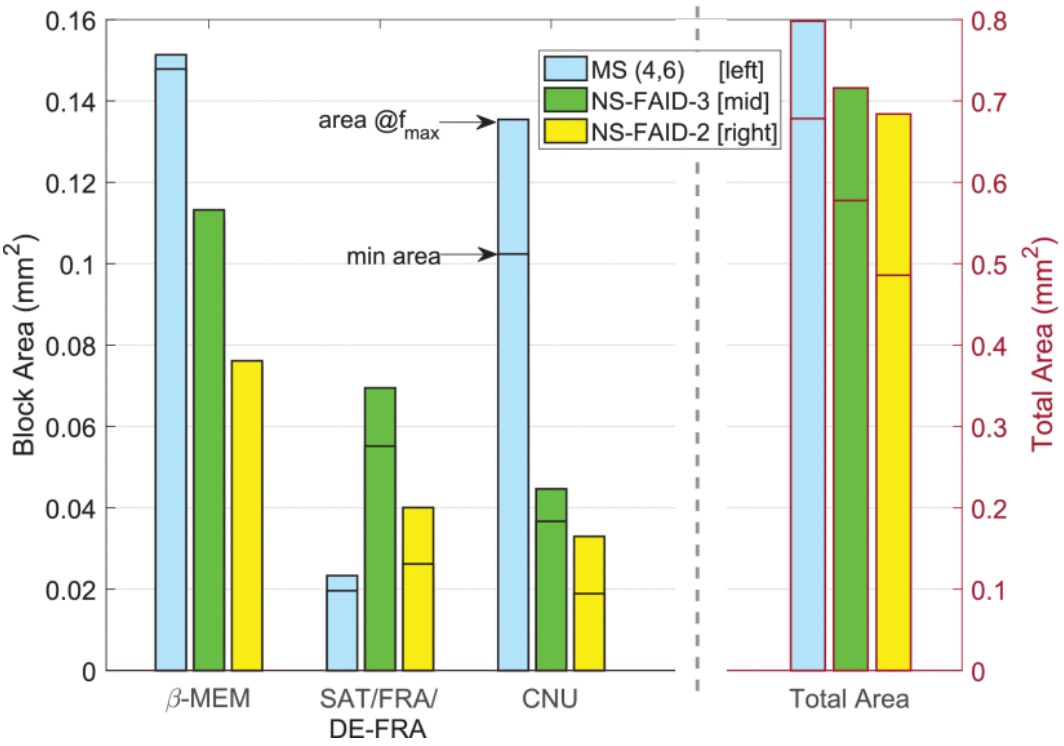



Figure 3.9: Component-level area comparison (full-layers architecture, with uncompressed CN messages), as in [8]. The min area corresponds to the case of relaxed timing constraint. ASIC post-synthesis implementation results on 65nm CMOS technology.

codeword, which is a SNR, codeword data, and message quantization dependent value, is considerably smaller than the maximum allowed number of decoding iterations allowed for any codeword. Thus, the stopping the decoder earlier will lead to significant reduction in energy consumption [73][74][75][76][77]. For layered architectures decoders a straightforward early-termination implementation requires extra hardware resources both for processing (*i.e.* routing and parity check computation of the AP-LLR sign bits), as well as storage (*i.e.* additional codeword length bits for storing the AP-LLR signs for the previous iteration). Furthermore, an extra iteration needs to be executed in parallel with the syndrome computation, or a the iteration suspended for syndrome computation. The later, in the worst case, doubles the execution time for a codeword decoding. Alternative methods for stopping the decoder, that overcome the conventional syndrome's computation limitations have been proposed in literature [73–77], and are referred to as "on-the-fly syndrome" computation because they estimate the state of the decoding process by some criteria computed during normal decoding iteration processing, and do not use additional storage for the AP-LLR signs from the previous iteration; thus, are said to be computed "on-the-fly". The efficiency of these strategies for deciding early decoding termination is characterized by its overhead (*i.e.* small energy consumption overhead, low cost), and of its impact on decoding performance (*i.e.* the target is to have negligible or no degradation).

The approaches proposed characterize the decoder state in terms of: AP-LLR messages' signs fluctuations [73], parity check constraints for the AP-LLR signs of consecutive layers [74], AP-LLR magnitudes exceeding a pre-defined threshold [75], both AP-LLR signs and AP-LLR magnitudes [76], or both AP-LLR magnitudes, and signs, as well as check-node message evolution between two consecutive iterations [77]. We have proposed a new method for on-the-fly stopping criterion based on parity check equations that are monitored layer by layer, for a number of $n_L$ consecutive layers.

Table 3.2: Theoretical hardware complexity versus decoding performance trade-Off for optimized irregular NS-FAIDs corresponding to WiMAX rate 1/2 variable node distributions. Decoding performance is measured as a SNR gain(+) or loss (-) with respect to baseline MS decoder, and is presented in column 6. Complexity is expressed in terms of bits needed for message representation for variable node messages (column 7), and check-node messages in both the uncompressed message format (column 8), and compressed format (column 9). The information from column 1 encoded as NS-FAID-$w_2w_3w_6$ is used to denote the ensemble of NS-FAIDs defined by a triplet of framing functions $F_2$, $F_3$, $F_6$, corresponding to variable node-degrees dv=2, 3, and 6, with message bit-lengths $w_2$, $w_3$, and $w_6$. The baseline MS is depicted as NS-FAID-444. The framing functions are depicted in columns 2, 3, and 4, while the $\eta$-threshold value (in dB) and the corresponding gain factor $\mu$ are shown in column 5 [16].

| NS-FAIDs Ensemble | Framing functions applied to | | | SNR-thres (dB) (& gain factor $\mu$) @BER $= 10^{-6}$ | SNR gain/loss (+/− dB) | Memory size reduction (%) | | |
|---|---|---|---|---|---|---|---|---|
| | $d_v = 2$ | $d_v = 3$ | $d_v = 6$ | | | VN-mess. | CN-messages | |
| | | | | | | | uncomp. | comp. |
| NS-FAID-444 | LUT0 | LUT0 | LUT0 | 1.374 ($\mu$=3.2) | 0.000 | 0.00 | 0.00 | 0.00 |
| NS-FAID-443 | LUT0 | LUT0 | LUT1 | 1.073 ($\mu$=2.9) | +0.301 | −9.87 | 0.00 | 0.00 |
| NS-FAID-434 | LUT0 | LUT8 | LUT0 | 1.073 ($\mu$=2.8) | +0.301 | −7.90 | 0.00 | 0.00 |
| NS-FAID-344 | LUT10 | LUT0 | LUT0 | 1.203 ($\mu$=2.6) | +0.171 | −7.24 | 0.00 | 0.00 |
| NS-FAID-442 | LUT0 | LUT0 | LUT15 | 1.224 ($\mu$=3.1) | +0.150 | −19.74 | 0.00 | 0.00 |
| NS-FAID-424 | LUT0 | LUT15 | LUT0 | 1.352 ($\mu$=3.1) | +0.021 | −15.79 | 0.00 | 0.00 |
| NS-FAID-244 | LUT18 | LUT0 | LUT0 | 2.106 ($\mu$=2.3) | −0.732 | −14.47 | 0.00 | 0.00 |
| NS-FAID-432 | LUT0 | LUT3 | LUT17 | 1.188 ($\mu$=3.0) | +0.186 | −27.63 | 0.00 | 0.00 |
| NS-FAID-423 | LUT0 | LUT17 | LUT2 | 1.262 ($\mu$=3.1) | +0.112 | −25.66 | 0.00 | 0.00 |
| NS-FAID-324 | LUT4 | LUT17 | LUT0 | 1.464 ($\mu$=2.6) | −0.091 | −23.03 | 0.00 | 0.00 |
| NS-FAID-342 | LUT10 | LUT0 | LUT17 | 1.278 ($\mu$=2.6) | +0.096 | −26.97 | 0.00 | 0.00 |
| NS-FAID-234 | LUT21 | LUT6 | LUT0 | 1.998 ($\mu$=2.7) | −0.624 | −22.37 | 0.00 | 0.00 |
| NS-FAID-243 | LUT21 | LUT0 | LUT5 | 1.997 ($\mu$=2.8) | −0.624 | −24.34 | 0.00 | 0.00 |
| NS-FAID-422 | LUT0 | LUT17 | LUT19 | 1.509 ($\mu$=3.4) | −0.135 | −35.52 | 0.00 | 0.00 |
| NS-FAID-242 | LUT21 | LUT0 | LUT15 | 2.049 ($\mu$=2.9) | −0.676 | −34.21 | 0.00 | 0.00 |
| NS-FAID-224 | LUT21 | LUT15 | LUT0 | 2.155 ($\mu$=2.9) | −0.781 | −30.27 | 0.00 | 0.00 |
| NS-FAID-433 | LUT0 | LUT9 | LUT8 | 1.015 ($\mu$=2.8) | +0.359 | −17.76 | 0.00 | 0.00 |
| NS-FAID-343 | LUT10 | LUT0 | LUT8 | 1.085 ($\mu$=2.4) | +0.289 | −17.11 | 0.00 | 0.00 |
| NS-FAID-334 | LUT10 | LUT8 | LUT0 | 1.102 ($\mu$=2.3) | +0.272 | −15.13 | 0.00 | 0.00 |
| NS-FAID-233 | LUT21 | LUT7 | LUT7 | 2.046 ($\mu$=2.6) | −0.672 | −32.24 | −25.00 | −13.04 |
| NS-FAID-323 | LUT11 | LUT17 | LUT5 | 1.457 ($\mu$=2.9) | −0.083 | −32.90 | −25.00 | −13.04 |
| NS-FAID-332 | LUT10 | LUT9 | LUT17 | 1.273 ($\mu$=2.6) | +0.101 | −34.87 | −25.00 | −13.04 |
| NS-FAID-333 | LUT10 | LUT10 | LUT9 | 1.110 ($\mu$=2.4) | +0.264 | −25.00 | −25.00 | −13.04 |
| NS-FAID-223 | LUT20 | LUT17 | LUT12 | 2.154 ($\mu$=2.9) | −0.780 | −40.13 | −25.00 | −13.04 |
| NS-FAID-232 | LUT21 | LUT7 | LUT14 | 2.080 ($\mu$=2.7) | −0.706 | −42.11 | −25.00 | −13.04 |
| NS-FAID-322 | LUT13 | LUT17 | LUT19 | 1.667 ($\mu$=3.4) | −0.293 | −42.76 | −25.00 | −13.04 |
| NS-FAID-222 | LUT18 | LUT16 | LUT16 | 2.299 ($\mu$=2.3) | −0.925 | −50.00 | −50.00 | −26.09 |

Table 3.3: LUTs used by NS-FAIDs in Table 3.2. L**x** is short for LUT**x** from Table 3.2 [16]

| $m$ | L0 | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 | L9 | L10 | L11 | L12 | L13 | L14 | L15 | L16 | L17 | L18 | L19 | L20 | L21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ±1 | ±1 | ±1 | ±1 | ±1 | ±1 | ±2 | ±2 | ±2 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| 2 | 2 | 1 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| 3 | 3 | 1 | 2 | 2 | 2 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 2 | 2 | 1 | 1 | 1 | 1 | 5 | 2 | 2 | 2 |
| 4 | 4 | 3 | 2 | 3 | 4 | 4 | 4 | 2 | 2 | 3 | 3 | 4 | 2 | 5 | 1 | 1 | 5 | 7 | 5 | 2 | 2 | 6 |
| 5 | 5 | 3 | 4 | 3 | 4 | 4 | 4 | 6 | 7 | 3 | 7 | 4 | 2 | 5 | 6 | 7 | 5 | 7 | 5 | 2 | 7 | 6 |
| 6 | 6 | 7 | 4 | 7 | 7 | 4 | 7 | 6 | 7 | 7 | 7 | 7 | 2 | 7 | 6 | 7 | 5 | 7 | 5 | 2 | 7 | 6 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 7 | 5 | 7 | 5 | 7 | 7 | 6 |
| $w$ | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

Note that LLR messages fluctuate during layered decoding, since an AP-LLR is updated $d_v$ degree number of times during an iteration [12] (see reference for detailed description of algorithm). We have evaluated the performance of the proposed early decoding termination criterion, with the state-of-the art from literature with respect to decoding performance using statistical Monte-Carlo simulations

Table 3.4: Best NS-FAIDs for $(3, 12)$-regular LDPC codes, as in [9]

| $w = 2$ | | $w = 3$ | |
|---|---|---|---|
| $F$ (LUT) | SNR-thres (dB) & gain factor $\mu$ | $F$ (LUT) | SNR-thres (dB) & gain factor $\mu$ |
| $[1, 1, 1, 1, 1, 6, 6, 6]$ | 4.651 (9.0) | $[0, 1, 1, 3, 3, 3, 7, 7]$ | 4.206 (5.8) |
| $[2, 2, 2, 2, 2, 2, 2, 7]$ | 4.717 (11.2) | $[0, 1, 1, 3, 3, 3, 3, 7]$ | 4.217 (5.6) |
| $[2, 2, 2, 2, 2, 2, 7, 7]$ | 4.730 (9.6) | $[0, 1, 1, 3, 3, 6, 6, 6]$ | 4.240 (5.0) |
| $[3, 3, 3, 3, 3, 3, 3, 7]$ | 4.812 (13.3) | $[1, 1, 1, 3, 3, 4, 4, 7]$ | 4.241 (7.4) |
| $[1, 1, 1, 1, 1, 7, 7, 7]$ | 4.848 (8.2) | $[0, 0, 2, 2, 4, 4, 4, 7]$ | 4.244 (6.7) |
| $[2, 2, 2, 2, 2, 6, 6, 6]$ | 4.849 (9.1) | $[1, 1, 2, 2, 4, 4, 4, 7]$ | 4.256 (7.4) |
| $[1, 1, 1, 1, 1, 1, 7, 7]$ | 4.920 (11.9) | $[0, 1, 2, 2, 2, 6, 6, 6]$ | 4.259 (4.6) |
| $[2, 2, 2, 2, 2, 7, 7, 7]$ | 4.966 (10.0) | $[1, 1, 1, 2, 4, 4, 4, 7]$ | 4.259 (7.3) |
| $[1, 1, 1, 1, 6, 6, 6, 6]$ | 4.973 (8.4) | $[1, 1, 1, 3, 3, 5, 5, 7]$ | 4.261 (7.3) |
| $[3, 3, 3, 3, 3, 3, 7, 7]$ | 4.991 (13.6) | $[0, 0, 2, 2, 3, 3, 7, 7]$ | 4.264 (6.6) |
| | | $w = 4$ | |
| | | NS-FAID-444 decoder | 4.326 (6.4) |

Table 3.5: Selected NS-FAID decoders synthesis results for FPGA technology, Virtex7 board, target Device xc7vx485t-2-ffg1761 for the unrolled layered architecture proposed in [4], and a regular array code, rate 3/4, having code size 1296 bits. Results show a TAR improvement ratio of 20% up to 125% fro NS-FAID with respect to baseline MS. as disseminated in [9]

| FAID Ensemble | Slice registers | Slice LUTs | | LUT-FF pairs | Freq [MHz] | T (4 it.) [GBps] | TAR [MHz/LUT-FF pairs] |
|---|---|---|---|---|---|---|---|
| | | Logic | Memory | | | | |
| MS-444 | 36986 | 247411 | 7128 | 271888 | 106 | 34.344 | 0.1263 |
| $[1, 1, 1, 1, 1, 6, 6, 6]$ | 29178 | 163524 | 5832 | 187595 | 137.523 | 44.388 | 0.2366 |
| $[2, 2, 2, 2, 2, 6, 6, 6]$ | 29233 | 150688 | 5832 | 174057 | 149.373 | 48.276 | 0.2775 |
| $[1, 1, 1, 3, 3, 4, 4, 7]$ | 29826 | 215413 | 6480 | 245228 | 117.277 | 37.908 | 0.1759 |
| $[1, 1, 1, 3, 3, 5, 5, 7]$ | 29826 | 224806 | 6480 | 254621 | 118.356 | 38.232 | 0.15 |

(Fig. 3.10), average iteration number (Fig. 3.12), and implementation cost (Fig. 3.11).

To sum up we have proposed a new method for on-the-fly stopping criterion having: (i) decoder performance similar for all analyzed codes, (ii) small implementation cost dependence on the underlying LDPC architecture characteristics (i.e. pipeline); (iii) cost overhead is small ($\approx$ 1% cost overhead with respect to a low cost serial AP-LLR processing MS decoder such as [2]).

### 3.4.2 In-Between Layers-Partial Syndrome [4]

The works [13][72] address the problem of early QC-LDPC decoding stopping criterion for layered scheduling decoders, with the multi-objective design goals of more safeness with low hardware cost and minimum latency. For this purpose we have proposed a new on-the-fly measure in the decoder, called in-between layers partial syndrome (IBL-PS), and have defined a family of stopping criteria, which allows different trade-offs between complexity, latency and performance. The proposed criterion has been validated using numerical simulation results, and compared against existing solutions. Our

[4]This subsection contains results and text partially reproduced from the journal publication [13], and patent Application[72]

| Code | Early termination method | | | |
|---|---|---|---|---|
| | *Proposed* | *OTF Syndrome* | *SDA* | *Sun 2008* |
| **WiMAX Rate ½** | 1.2e-04 at 2.6 dB<br>4.3e-05 at 3.0 dB | - | 10x - 100x at<br>2.6, 2.8 , 3.0 dB | - |
| **WiMAX Rate 5/6** | - | - | 3.9e-05 at 6.8 dB<br>2.8e-05 at 7.0 dB | - |
| ***Dv=3* Rate 1/2 - no FER degradation** | | | | |
| ***Dv=3* Rate ¾** | 1.8e-04 at 5.8 dB<br>2.1e-04 at 6.2 dB | - | - | 1.0e-05 at 6.2 dB |
| ***Dv=4* Rate ½** | - | 10x - 100x at<br>3.6, 3.8, 4.0 dB | - | - |
| ***Dv=4* Rate ¾** | - | 10x - 100x at<br>5.8, 6.0, 6.2 dB | 10x - 100x at<br>6.0 and 6.2 dB | - |
| **WPAN Rate ½** | - | - | 1.9e-03 at 3.0 dB | - |
| **WPAN Rate 3/4** | - | - | 1.0e-04 at 6.2 dB<br>2.8e-05 at 6.6 dB | - |
| **WPAN Rate 7/8** | - | 10x - 100x at<br>7.4, 7.8, 8,0 dB | 2.0e-04 at 8.0 dB | - |

Figure 3.10: Decoding performance for various standard codes WiMAX [7], WPAN [10] for different code rates and four non-standard regular matrices (rates 1/2 and 3/4) [11], and a maximum allowed number of 20 iteration. Simulation results show that a negligible SNR degradation $< 0.2$ dB is observable for the proposed criteria [12].

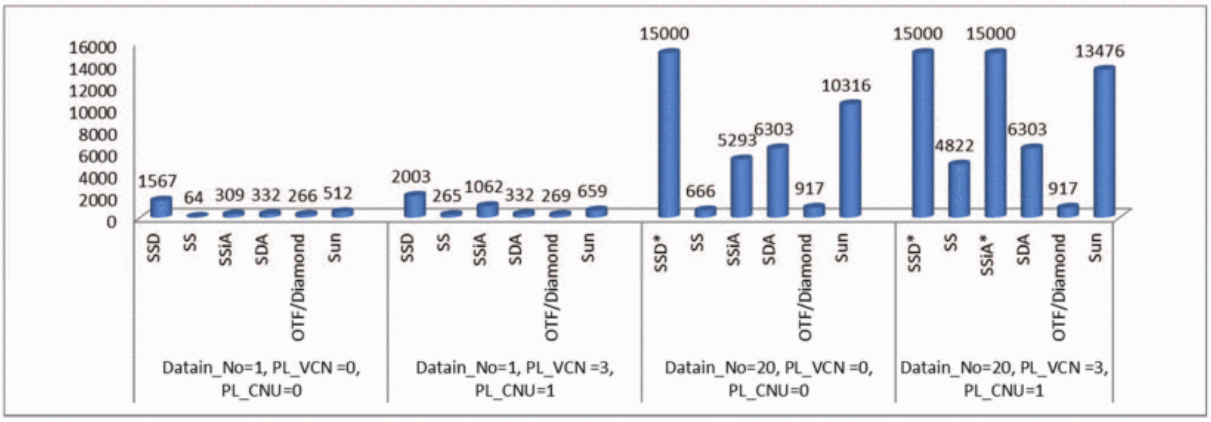



Figure 3.11: synthesis results for WiMAX codes, with circulant size 96 for Xilinx Virtex 7-7vx330tffg 1157–3 FPGA platform using Xilinx ISE 14.7. The following hardware architecture parameters: the number of AP-LLR messages processed at once, and the number of pipeline levels. The cost exceeding 15000 LUT-FF pairs has been marked them with (*) and saturated – as in [12].

findings suggest that IBL-PS surpasses existing solutions, and can be as safe as the full-syndrome detection, down to frame error rates (FER) as low as FER=$10^{-8}$.

The work [13] addresses the case of protograph LDPC of type-I, *i.e.* when $b_{i,j} \in \{0, 1\}$. As discussed in 2.2, $b_{i,j} \geq 0$, $\forall i = 1 \dots M_b$, $\forall j = 1 \dots N_b$, is an entry in the base matrix $B$. In addition to this, an alternative name of the base matrix $B$ is that of *protograph* [78]. Tha parity check matrix $H$ of a QC-LDPC code, of size $(M, N) = (M_b\, L, N_b\, L)$, can be organized in groups of organized in groups of $L$ consecutive parity-checks, corresponding to one row of circulants, also called *layers*.

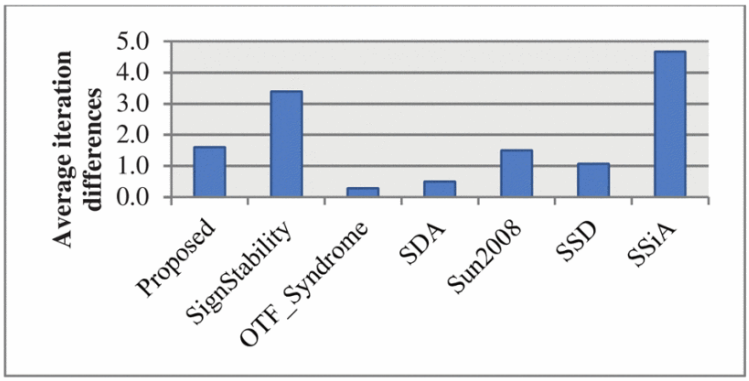



Figure 3.12: Difference of average iteration increase compared to the baseline. Note that the proposed criterion provides similar performance for all codes analyzed, yielding an almost constant average iteration increase of around 1.6 iterations [12]

At the end of each layer processing, the AP-LLR messages are updated, from which the hard decision vector is computed, $\hat{\boldsymbol{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)^T \in \{0,1\}^N$. The syndrome vector is defined as $\boldsymbol{s} = H \times \hat{\boldsymbol{x}}$. We say that all the parity checks are satisfied if $\boldsymbol{s} = \mathbf{0}$. In this event, the vector $\hat{\boldsymbol{x}}$ is a codeword of $H$. We use the submatrix $H_i$, $\forall i = 1 \ldots M_b$, corresponding to the parity check equations of a layer, to define a *partial syndrome* vector $\boldsymbol{s}_i = H_i \times \hat{\boldsymbol{x}}$. $\boldsymbol{s}_i$ measures the satisfiability of all the parity check equations within layer $H_i$. Similarly the $(2\,L, N)$ submatrix corresponding to the concatenation of two consecutive layers $H_{i,i+1} = [H_i\,;\, H_{i+1}]$, $\forall i = 1 \ldots M_b$, where the indexes are taken modulo-$M_b$. The in-between layers partial syndrome (IBL-PS) is defined in [13] as follows.

**Definition 3.1.** *Let $\hat{\boldsymbol{x}}_i = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$ be the hard decision vector, computed after processing of layer $H_i$ and before processing of layer $H_{i+1}$. The IBL-PS is defined as*

$$\boldsymbol{s}_{i,i+1} = H_{i,i+1} \times \hat{\boldsymbol{x}}_i \tag{3.6}$$

The use of to use the partial syndrome $\boldsymbol{s}_i$ as a measure to define a stopping criterion has been used in other works such as [74], and since its computation is performed on the latest hard-decision values, during layer processing, it has been called *on-the-fly syndrome* (OTF syndrome). Implementation wise, the $\boldsymbol{s}_i$, $\forall i = 1 \ldots M_b$ computation is of low complexity, translates to adding $L$ XOR gates, as opposed to computing the whole syndrome of $H$ that requires all the parity checks to be computed, as well as the additional routing network. Furthermore, since it is computed in parallel with layer decoding, it does not incur any degradation on the iteration latency. The proposed IBL-PS measure can also be computed on-the-fly, from the AP-LLR sign values, by computing the parity-checks within layer $H_i$ when data is written back to the AP-LLR memory (*i.e.*, after current layer processing), and those within layer $H_{i+1}$ when data is read from the AP-LLR memory (*i.e.*, before next layer processing). Thus, we ensure that the parity-checks within both layers are verified on the same hard decision vector. We have further extended the definition of IBL-PS to *generalized layers* (GL) [72]. A generalized layer refers to any group of consecutive $B$ rows that are non-overlapping with each other. Thus, within this group any column of $B$ has at most one non-negative entry within these rows.

The parity check approach from [74] computes on-the-fly $\boldsymbol{s}_i$, $\forall i = 1 \ldots M_b$, over the course of an iteration, and stops the decoder is all partial $\boldsymbol{s}_i$ are satisfied. Although implementation wise this approach is of very low complexity and cost overhead, this decoding criterion's impact pretty much

varies with the choice of code, and for some cases is unsafe leading to large decoding performance (bit error rate and frame error rate) degradation.

In our work [13], we have proposed to use the new on-the-fly IBL-PS measures. Furthermore, we were able to define a family of stopping criteria that is parametrized by $\theta \geq 1$, as follows:

**Stopping Criterion 1.** *The layered decoding stops when $\theta$ consecutive IBL-PSs are satisfied, i.e. if for some $i$,*

$$\textit{IBL-PS}(\theta) \Leftrightarrow \boldsymbol{s}_{i,i+1} = \boldsymbol{s}_{i+1,i+2} = \ldots = \boldsymbol{s}_{i+\theta-1,i+\theta} = \mathbf{0} \tag{3.7}$$

The parameter $\theta$ takes minimum value 1, and may be span across several iterations, if $\theta > M_b$. By increasing the value of $\theta$ will make the stopping criterion safer and safer, and will also increase iteration time. Of-course, the smallest latency corresponds to the case of $\theta = 1$. The synthesis results for a layered QC-LDPC decoder architecture using IBL-PS stopping criterion, have shown that the cost overhead incurred by this criterion is less than 2% of the global layered architecture implementation cost. Furthermore, for $\theta > 1$ the implementation cost is the same. Moreover, we this value can be dynamically reconfigured if needed.

The reduced complexity stopping criteria are unsafe, in the sense that there are decoding situations when the decoder is stopped, and the hard decision at the output is not a valid codeword. This is a major drawback of low-complexity stopping criteria. In order to gain better insight into this issue we can analyze specific structures of the Tanner graph of the LDPC code having certain graph topologies that are cycles or trapping sets [79]. Vectors satisfying the partial syndrome $\boldsymbol{s}_{i,i+1}$ , have been referred to as a codeword of $H_{i,i+1}$, are quite simple to characterize in the case of two *full layers*. As detailed in [13] a first observation is that QC-LDPC codes well adapted to the IBL-PS(1) should have the property that any two consecutive layers have the maximum possible girth $g = 12$, with the lowest multiplicity of those 12-cycles.

In [13] we analyze a QC-LDPC code satisfying the aforementioned properties for the IBL-PS(1) criterion. The code design algorithm used is from [80]. Table 3.6 shows the cycle distribution of $H$ and the three associated submatrices for the rate $R = 1/2$ array QC-LDPC code, of length $N = 768$ bits, designed specifically for the IBL-PS(1) stopping criterion having a $(M_b, N_b) = (3, 6)$ array-type base matrix $B$, with $b_{i,j} = 1\ \forall i, j$, and circulant size is $L = 128$. Our findings suggest that although the global girth of the LDPC graph is $g = 10$, it is in fact possible to ensure that the combination of two layers form subgraphs with girth $g = 12$.

Two types of analysis have been carried out. First, the regular code from Table 3.6 have been considered. Second, we have considered the protograph QC-LDPC codes from the WIMAX standard [7], which is an irregular code with code length $N = 2304$ and $M_b = 12$ layers. We have studied the effect of the proposed IBL-PS stopping criterion with respect to other on-the-fly computed stopping criteria from state-of-the art with respect to decoding performance and average iteration count for the additive white Gaussian noise (AWGN) channel with signal to noise ratio $E_b/N_0$. The decoding algorithm used for this assessment is offset Min-Sum [79], with layered scheduling and a maximum of 50 decoding iterations [13].

Although, a plethora of low-complexity imprecise stopping criteria proposed in the literature, they monitor either sign changes such as *sign stability* (SS) proposed in [73], or parity checks using sign information such as the on-the-fly syndrome check (OTF) proposed in [74], or monitor intrinsic or extrinsic message amplitude variations [75–77].

| $(M_b, N_b) = (3, 6)$ $L = 128$ | | | | |
|---|---|---|---|---|
| | 10-cycles | 12 cycles | 14-cycles | 16-cycles |
| $H_{1,2}$ | | 2 688 | | 49 536 |
| $H_{2,3}$ | | 5 248 | | 46 848 |
| $H_{3,1}$ | | 4 224 | | 47 488 |
| $H$ | 9 600 | 93 312 | 756 224 | 6 299 008 |

Table 3.6: Statistics of the rate $R = 1/2$LDPC code designed specifically for the IBL-PS(1) stopping criterion having a $(M_b, N_b) = (3, 6)$ array-type base matrix $B$, with $b_{i,j} = 1\ \forall i, j$, and circulant size is $L = 128$, which results in an expanded code length $N = 768$ bits [13]

| $E_b/N_0$ | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 | 3.00 |
|---|---|---|---|---|---|---|---|
| $\overline{It}$ | 11.18 | 8.14 | 6.79 | 5.94 | 5.32 | 4.84 | 4.47 |

Table 3.7: Convergence Rate for the Wimax Code, rate 1/2, with code length $N = 2304$ and $M_b = 12$, as in [13]

The SS criterion monitors the sign fluctuations of at each layer $H_i$, by comparing the signs of the AP-LLRs before and after layer processing. If signs don't change a counter denoted by $\theta$ is incremented, otherwise the counter is reset. Thus, this criterion is denoted by SS($\theta$) in order to emphasize the relation with the maximum counter value that decides the stopping of the decoder. This criterion is almost safe, and in many cases leads to an increase of the average iteration count for the decoding process. A counter based approach is used for the OTF criterion, where a counter is incremented by 1 if a partial syndrome $\boldsymbol{s}_i$ is satisfied, and reset to zero if not satisfied. In [74], the maximum value of the counter $\theta$ is fixed to $\theta = M_b$ and the counter was reset at the beginning of each iteration, while in [12], we have considered a safer condition with $\theta > M_b$. The weakness of the OTF($\theta$) syndrome comes from the fact that the hard decision vector $\hat{\boldsymbol{x}}$ may change oscillate between consecutive layer updates. These are not missed by the OTF, rendering the criterion unsafe. The decoding performance statistic curves comparing the IBL-PS stopping criterion are shown in Fig. 3.13. With respect to average iteration count for different stopping criteria that have the same error correction performance, *i.e.* Syndrome based, SS(3), OTF(9) and IBL-PS(2) the analysis results from [13] show that the IBL-PS(2) stopping criterion allows reducing the average number of iterations by up to 30% compared to the SS(3) criterion, and up to 108% compared to the OTF(9) criterion.

The second case considered for the stopping criteria analysis is that of the WIMAX standard [7] rate 1/2 QC-LDPC codes protograph, an irregular code with code length $N = 2304$ and $M_b = 12$ layers. Different from the above case, the layers in WiMAX rate 1/2 code contain all-zero blocks, which we expect to lessen the safeness of the IBL-PS criterion. For the purpose of this analysis, we have computed the average number of iterations for all stopping criteria with different values of $\theta$, and select the value of $\theta$ for which the average number of iterations matches the one of the full syndrome, indicated in Table 3.7. The conclusion of this analysis is that the only criterion which approaches the safeness of the full syndrome is the IBL-PS(5).

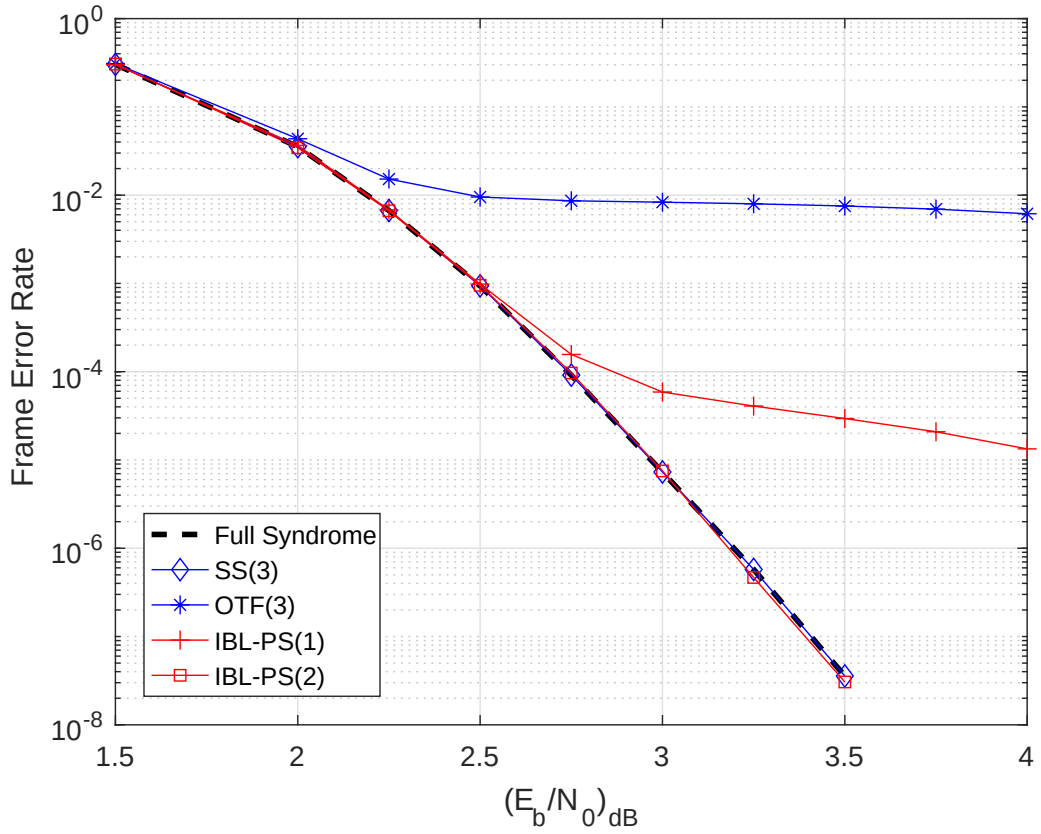



Figure 3.13: Different stopping criteria on a $(M_b, N_b) = (3, 6)$, $N = 768$ QC-LDPC code. The IBL-PS(1) stopping criterion does not introduce any performance loss compared to the full syndrome until the frame error rate (FER) reaches an error floor at FER $\approx 10^{-4}$, and the IBL-PS(2) is as safe as the SS(3), with no performance degradation compared with the full syndrome check. The OTF(9)'s performance is similar to that of the other stopping criteria down to FER=$10^{-6}$, as in [13].

To sum up we have proposed an early stopping criterion – called in-between layers partial syndrome – for layered QC-LDPC decoders, aiming at low hardware cost, minimum latency and an improved safeness compared to other similar criteria. Furthermore, we have defined a family of stopping criteria using this new measure, with different trade-offs between latency and performance. The analysis from [13] shows that our imprecise stopping criterion is sufficiently safe to be considered in practical applications, while surpassing existing solutions from the state-of-the-art. Dissemination wise, we have submitted this principle for patenting purposes. Two patent applications have been filled, one for US and one for an EPO patent.

## 3.5 One-Hot encoding CNU implementation [5]

In [14] we have proposed a novel check node unit architecture suitable for very high rate for LDPC decoders. Its distinct feature is the fact that it avoids the usage of carry based comparators for the computation of the required first and second minimum values. In order to do so it uses a a one-hot representation of the input messages' magnitude, obtained by $q$-to-$2^q$ decoders. The message magnitude is converted to an array of bits with only one non-zero bit position corresponding to the input value (e.g. the 3-bit value 2 is represented by the 8-bit vector 00000100). After converting the magnitude in one-hot representation an OR tree (e.g. a bitwise OR between 1, 2, and 5 will result in the vector 00110010), and a modified leading zero counter (LZC) is used to compute the first and second minimums. The least significant "one" value position in the vector obtained after the OR operation represents corresponds to the value of the first minimum, while the second least significant one is assigned as the second minimum. The conventional LZC computes only the position of the least significant one. Hence, we modify it such that it computes the required "one" bit value positions. The computation is imprecise, since this approach cannot distinct the case when both minimums are of equal value. We have analyzed the proposed approach in terms of both FPGA technology implementation cost, as well as decoding performance degradation due to the aforementioned impreciseness. The most favorable conclusion is obtained for high rate codes, for which results have shown as up to 30% hardware cost improvement, higher working frequency, at the expense of negligible decoding capability degradation with respect to standard implementations. The detailed proposed CNU building blocks are presented in Fig. 3.14.

Implementation results for FPGA technology are depicted in Table 3.8 for cost estimates, and Table 3.9 for frequency estimates. Note that the proposed implementation of the CNU presents a 9% to 30% better cost with respect to the baseline implementation for $d_c$ values greater than 10. The decoding performance is analyzed in Fig. 3.15. It can be seen that at BER = 1E-5, the SNR degradation induced by the imprecise CNU varies between 0.25 dB (rate 1/2) and 0.15 dB (rate 9/10) for the MS decoder, and between 0.12 dB (rate 1/2) and 0.08 dB (rate 9/10) for the SCMS decoder [14].

To sum up the CNU proposed in [14] is best suited for LDPC codes with high $d_c$, as it presents significant cost reduction, lower latency, while the loss in decoding performance is negligible.

[5] This subsection contains results and text partially reproduced from the journal publication [14]

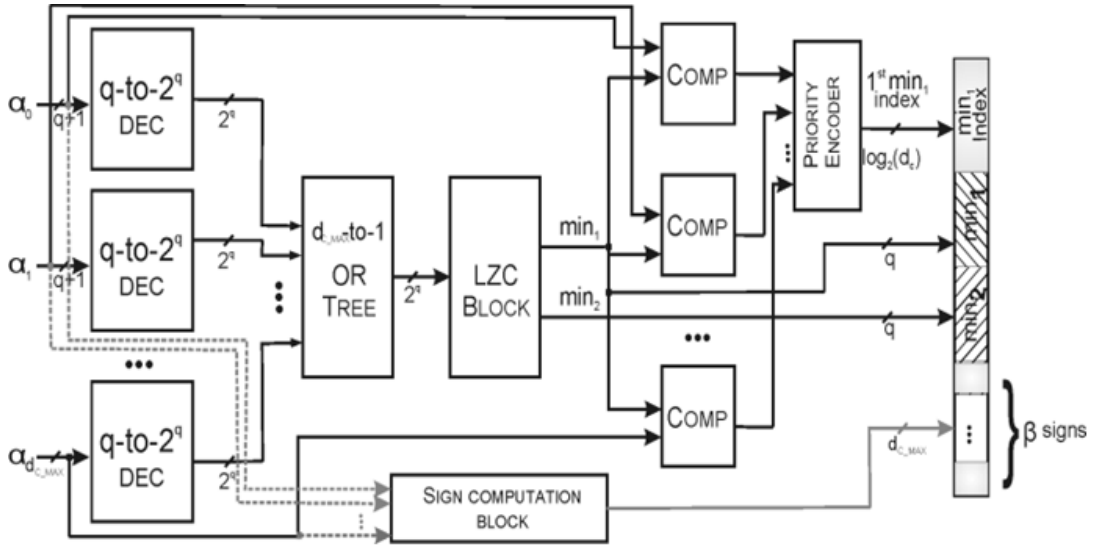



Figure 3.14: Architecture of the proposed one-hot encoding CNU implementation using imprecise comparator, as in as in [14]. The decoders are used to perform the conversion from of the magnitude in one-hot implementation. The LZC block is used to compute the first and second minimums. The index of the first minimum is a two stage computation: first compare the first minimum with the inputs, and then use a priority encoder during the second stage.

| $d_c$ | 10 | 15 | 20 | 32 | 40 | 64 | 72 |
|---|---|---|---|---|---|---|---|
| baseline | 141 | 195 | 274 | 430 | 591 | 873 | 976 |
| proposed | 113 | 155 | 204 | 332 | 383 | 612 | 685 |

Table 3.8: Cost estimates expressed in LU-FF pairs for the baseline [17] and the proposed CNU for Virtex-7 FPGA (in MHz) , as in [14]

| $d_c$ | 10 | 15 | 20 | 32 | 40 | 64 | 72 |
|---|---|---|---|---|---|---|---|
| baseline | 272 | 246 | 209 | 197 | 182 | 166 | 153 |
| proposed | 254 | 216 | 197 | 185 | 197 | 176 | 175 |

Table 3.9: Frequency estimates for the baseline [17] and the proposed CNU for Virtex-7 FPGA (in MHz) , as in [14]

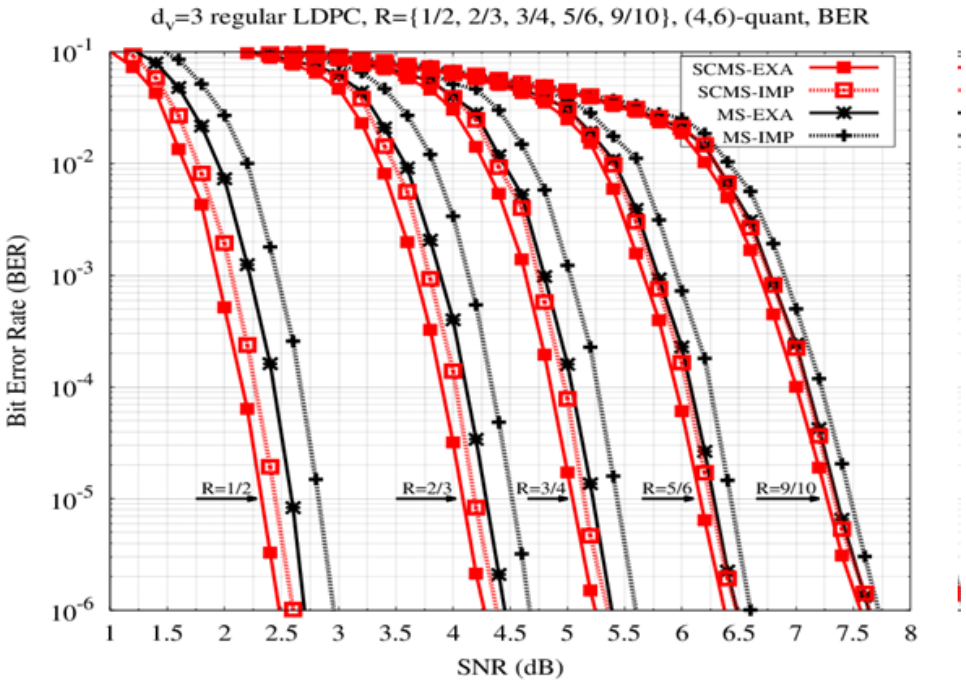



Figure 3.15: The bit error rate (BER) performance over the Additive White Gaussian Noise (AWGN) channel with quadrature phase-shift keying (QPSK) modulation for regular LDPC codes with dv = 3 and dc = 6, 9, 12, 18, 30, corresponding to coding rates R = 1/2, 2/3, 3/4, 5/6, 9/10, having exchanged messages quantized on 4 bits, as in [14]. Solid curves correspond to the exact CNU and dashed curves to the proposed imprecise CNU.

## 3.6 Miscellaneous contributions to imprecise computation of QC-LDPC decoders

This section discusses additional imprecise computation and approximate storage contributions to QC-LDPC decoding proposed for other scheduling types such as flooding [1][18], as well as other decoding algorithms besides MS and MS-like [59]. These have been disseminated in the following publications and are briefly described in the remaining of this section.

### 3.6.1 Gear-like decoding for QC-LDPC codes using flooding scheduling [6]

The contributions from Section 3.5, Section 3.3, and Section 3.2 propose techniques and architecture changes that aim to reduce the cost and/or latency of layered decoder architectures. One common aspect is that all iterations have the same latency, and all use the same optimizations. The work from [1], tries to decrease overall decoder latency by speculating the changes in message amplitudes during the decoding process. Taking advantage of this observation, it proposes a multi-domain quantization for a partially-parallel QC-LDPC decoder, such that memory bandwidth is utilized efficiently, and the overall decoding latency is decreased for the case when no early termination criteria is used. Typically two approaches are utilized for stopping a decoder. The first if to run it for a fixed number of iterations, and the second is to check the syndrome after each iteration, and if it is satisfied stop the decoder, otherwise continue until the maximum allowed number of iterations is executed. As stated before, the proposed decoder should be able to change quantization dynamically, where by

[6] This subsection contains results and text partially reproduced from the conference publication [1]

"dynamic" we understand that during a codeword decoding, different iterations are allowed to used different quantizations. Our aim is to allow the trade-off of speedup and energy efficient processing for error correction capability degradation. Although, in principle any number of quantization domain changes can be implemented, in practice cost-wise this number should be small. In [1], we use 2 quantization modes: a high resolution based $q$-bit messages, and a low resolution $q/2$-bit messages. Furthermore, for reasons explained in [8][27], in the lower quantization mode, the variable node units operate on $q/2+1$ resolution, while check node units process $q/2$-bit messages. For the decoding performance analysis we consider both the Additive White Gaussian Noise (AWGN) and Binary Symetric Channel (BSC) channel models. Implementation wise, we validate this approach on a full decoder implementation. Flooding scheduling decoding for an iteration is a two-step process. First, all check-nodes are processed and the new messages passed to their neighbors, followed in the second step by variable-node processing of the check-node messages and channel input. Subsequently all variable-nodes pass new messages to their neighbors. Note that the MS algorithm is used this time as well, meaning that the VNU and CNU processing is more or less the same as with layered scheduling MS.

**Algorithm 2** Flooding Min-Sum decoding principle, as described in [1]

Initialization
**for all** $n \in V$ **do**
    $\gamma_n = Channel_n$
    $\beta_{m,n} = 0$
    $\alpha_{p,n} = Channel_n,\ p \in H(n)$
Iterative message exchange
**for** $it = 0, Itmax$ **do**
    **for all** $m \in W$ **do**
        $\tilde{\beta}_{m,n} = \prod_{\acute{n} \in H(m) \setminus n} sign\left(\alpha_{m,\acute{n}}\right) \cdot \min_{\acute{n} \in H(m) \setminus n} |\alpha_{m,\acute{n}}|$ ▷ Check node $CN_m$ message processing.
    **for all** $n \in V$ **do**
        $\tilde{\alpha}_{p,n} = \gamma_n + \sum_{\acute{p} \in H(n) \setminus p} \beta_{\acute{p},n}$
        ▷ Variable Node $VN_n$ message processing.
    **for all** $n \in V$ **do**
        $\tilde{\gamma}_n = \gamma_n + \sum_{m \in H(n)} \beta_{m,n}$
        ▷ $AP-LLR_n$ message update.

By analyzing the amplitude evolution of the exchange messages for several QC-LDPC codes, we noticed that during the first iterations, messages have typically small amplitudes, and that during the last iterations, most message amplitudes have converged to the maximum representable values. By analyzing this behavior we have proposed that a codeword's decoding is split in three phases with respect to the number of quantization bits used [1] :

$Ph_1$ Low number of quantization bits for both storage and operation - use $q_R$ bits;

$Ph_2$ High quantization domain for both storage and operation - use $q_F$ bits;

$Ph_3$ Low quantization domain for both storage and operation - scale down messages to $q_R$ bits before executing Phase 3;

Related works addressing the issue of reduced quantization domain approach have been proposed in[28], and [27]. These rely on a form of compression/decompression functions that reduced message storage by 1 bit while gaining performance, and reducing resource usage. Furthermore, it has been proven that dual-domain quantization for the VNU/CNU operation can in fact not only reduce storage requirements, but also improve decoding performance [16]. Hence, we also use this approach when working with lower quantized messages (*i.e.* during phases $Ph_1$ and $Ph_3$, the VNU operating on $\frac{q_F}{2}+1$ bits, while the CNU operates in the $\frac{q_F}{2}$ bits. For the high message quantization, during processing phase $Ph_2$, both units operate on on $q_F$ bit messages. In order to take full advantage of the times when lower quantization messages are employed, during phases $Ph_1$ and $Ph_3$, two messages are processed by both the VNUs and the CNUs instead of one message, as during phase $Ph_2$. This essentially means that iteration processing speed can be improved by a factor $2\times$ for phases $Ph_1$ and $Ph_3$, with respect to $Ph_2$. Implementation-wise, this *dynamic multi-domain quantization* (i.e. different quantizations between iterations, and during the reduced quantization iterations, dual-quantization domain) has been achieved by designing *Single Instruction Multiple Data (SIMD)* processing units [1]. It is worth emphasizing we target a high memory efficiency (denoted by $\mu$) for the extrinsic message memory. A key decision when discussing implementation cost for the SIMD units is the message mapping between the VNU working with $\frac{q_F}{2}+1$ bit messages and CNU working with $\frac{q_F}{2}$ bit messages. The simplest mapping is the one from [27]. The $\tilde{}$ notation denotes current iteration messages (outputs $\rightarrow$ memory). This information is summarized in Table 3.10. Note that only the most significant 2 bits of $\tilde{\alpha}$ messages from the VNU output during last iteration of $Ph_2$ are saved to memory. The last is zeroed, hence is a hidden 0 from the memory operation perspective. CNU operation is performed on the number of $\alpha$ message bits retrieved from memory. For $Ph_1$, $Ph_3$ this means 2 bit message processing. $Ph_2$ works with 4 bit message. Note that during phases $Ph_1$ and $Ph_3$ the VNU is operating on $\frac{q_F}{2}+1$ bits, which processes $2\times\beta$ messages on $\frac{q_F}{2}$ bits and outputs $2\times\alpha$ messages represented on $\frac{q_F}{2}$ bits. Also, the VNU uses $1\times\beta$ message on $q_F$ bits, with the same output resolution for $\alpha$ messages during $Ph_2$. The CNUs operate either in the $\frac{q_F}{2}$ bits domain processing 2 messages at a time, or in the $q_F$ bits higher quantization domain processing 1 message at a time. The $\tilde{}$ notation denotes updated messages (outputs $\rightarrow$ memory) of the current iteration. For the concrete

We have implemented this approach on a flooding scheduling decoder architecture. The baseline uses serial processing units (one message is being processed at a time). We have kept the same memory mapping and replaced the processing units with SIMD units. Furthermore, we have modified the control unit such that the SIMD command is generated. As argued before, during the first and last phase, the decoder processes $2\times$ messages for both types of units (*i.e.* VNU and CNU), and utilizes the maximum theoretical bandwidth. Furthermore, for limiting the decoding performance degradation, we use dual-domain quantization during $Ph_1$, and $Ph_3$. We have analyzed the impact on decoding performance of the proposed approach, up to the case when up to 50% of the total iterations are reduced quantization iterations for different QC-LDPC codes, by means of Monte Carlo simulations. The BER/FER performance curves show that there is no visible degradation in the waterfall. With respect to cos, the synthesis results for the SIMD units have shown an increase (17% and 36.6% respectively) with respect to the baseline, while working frequencies stay in the same range. To sum up, we have proposed a decoder implementation fusing dynamic quantization and SIMD units able to process either 1 or two messages at a time, and allowing runtime efficient trade-off of decoding

Table 3.10: The message mappings for each phase, and when transitioning from one phase to the next. The phases transition $Ph_1 \rightarrow Ph_2$ suggests that during the last iteration of phase $Ph_1$ VNU messages are stored on 3 bits to memory. The transition $Ph_2 \rightarrow Ph_3$ should be interpreted as: $\gamma$ channel messages are represented as +2 if the sign of the channel message is +, or -2 if it is equal to -. Only the most significant 2 bits of $\tilde{\alpha}$ messages from the VNU output during last iteration of $Ph_2$ are saved to memory. The last is zeroed, hence is a hidden 0 from the memory operation perspective. CNU operation is performed on the number of $\alpha$ message bits retrieved from memory. For $Ph_1$, $Ph_3$ this means 2 bit message processing. $Ph_2$ works with 4 bit message. As in [1]

| **To** $\rightarrow$ **From** $\downarrow$ | $Ph_1$ | $Ph_2$ | $Ph_3$ |
|---|---|---|---|
| $Ph_1$ | $\gamma \rightarrow \gamma_3\gamma_2\gamma_1\gamma_0$<br>$\tilde{\alpha} \rightarrow \tilde{\alpha_2}\tilde{\alpha_1}0$ | $\gamma \rightarrow \gamma_3\gamma_2\gamma_1\gamma_0$<br>$\tilde{\alpha} \rightarrow \tilde{\alpha_2}\tilde{\alpha_1}\tilde{\alpha_0}$ | $NA$ |
| $Ph_2$ | $NA$ | $\gamma \rightarrow \gamma_3\gamma_2\gamma_1\gamma_0$<br>$\tilde{\alpha} \rightarrow \tilde{\alpha_3}\tilde{\alpha_2}\tilde{\alpha_1}\tilde{\alpha_0}$ | $\gamma \rightarrow \gamma_3 10$<br>$\tilde{\alpha} \rightarrow \tilde{\alpha_3}\tilde{\alpha_2}$ |
| $Ph_3$ | $NA$ | $NA$ | $\gamma \rightarrow \gamma_3 10$<br>$\tilde{\alpha} \rightarrow \tilde{\alpha_2}\tilde{\alpha_1}$ |

performance for decoding speed.

### 3.6.2 Probabilistic Gradient Descent Bit Flipping Decoder Using Variable Node Shift Architecture [7]

The contribution from [59] discusses a different class of LDPC decoding algorithms – hard-decision decoders. In particular it focuses on a recently proposed decoder Probabilistic gradient descent bit-flipping (PGDBF) [81]. PGDBF has gained attention due to its error correction properties that are approaching the performance of soft-information decoders on the BSC channel. Its advantage is given by its probabilistic nature, which if implemented straightforwardly with pseudo-random noise generators comes at an extremely large overhead compared to the non-probabilistic gradient descent bit flipping (GDBF) implementation [82]. The paper [59] proposes a new fully parallel flooding architecture that can favorably approximate the *probabilistic* behavior without the need to implement noise generators, called variable-node-shift architecture (VNSA). Due to the structure nature of QC-LDPC codes, messages are cyclic shifted in order to be processed by the appropriate VNU, and then back to memory. In a fully parallel implementation, we ca take advantage of this observation, and induce a probabilistic effect by using circular shifting and several types of VN processing. More specifically, we can shift messages in a linear shift register fashion, such that a message undergoes different VN types of processing every iteration. Although, any type of shifting can be implemented for flooding scheduling fully parallel decoders, circular shifting modulo $z$ of one position every iteration is the low cost approach, that has proven sufficient for inducing the probabilistic behavior. In this way, by carefully handpicking the VN types of processing, we were able to further improve the decoding of PGDBF, compared to existing hardware implementations, and implementation-cost wise achieve a complexity below that of the GDBF. These findings are the result of ASIC synthesis and by Monte-Carlo decoding simulations.

[7] This subsection contains results and text partially reproduced from the journal publication [59]

According to VNSA architecture, the VNs and CNs are mapped to different VNUs and CNUs during each iteration. The mapping rule is simple: circular $z$ cyclic shift by one position. Hence, in VNSA a VN (CN) is operated in 2 consecutive VNUs (CNUs) (*i.e.* VNUs(CNUs) that are in the same column (row) of the base matrix, and their indexes in the expended matrix are consecutive modulo $z$). If all Variable Node Units (VNUs) are identical, the decoding is identical to that of a generic flooding scheduling implementation. Note that we consider a generic architecture, one that always computes a VN (CN) using the same (fixed) VNU (CNU).

PGDBF is a decoding algorithm that relies on a probabilistic input signal to have two distinct VNU types of processing. The effect of this probabilistic decision is that a VNU may behave in 2 different ways, that is, it may have 2 different results in 2 iterations, even if the input messages are identical. Since in the VNSA architecture, the VNs and CNs are mapped to different VNUs and CNUs during each iteration, it becomes a natural candidate for efficient realization of the PGDBF decoding algorithm. Furthermore, it becomes apparent that for fully parallel decoder implementations the VNSA distributes the different behaviors in different VNUs, instead of implementing a complex VNU with multiple behaviors, and employs a permutation rule for routing different messages to different VNU types during each iteration such that the desired "effect" is induced. PGDBF needs a probabilistic effect. Implementation wise, the simplest permutation is cyclic shift by one position. In terms of decoding performance, it has proven sufficient. Therefore, a VNSA-PGDBF implementation has the advantage of PGDBF, and the complexity of GDBF, since the expensive signal noise generator has been reduced. VNSA-PGDBF complexity can be reduced by replacing type-2 VNU by other VNU – type-3. The type-3 VNU does not compute the energy value and all computing circuitry is reduced. Hence, it behaves like a conventional type-2 VNU with 0 at the probabilistic input signal. In other words, it simply forwards the input message values to the next iteration.

In [59] we have introduced a new architecture Variable Node Shift Architecture (VNSA), suitable for efficient PGDBF decoding implementation of QC-LDPC codes. Two implementations have been realized:

- VNSA-PGDBF that uses two types of VNUs and a simple circular shift by one position of consecutive nodes. Th 2 types of VNUs that have been designed are simpler than those of the conventional PGDBF implementation. Consequence of this, ASIC synthesis results have shown that VNSA-PGDBF reduces the decoder complexity up to 11% less than the one of the deterministic Gradient Descent Bit Flipping (GDBF) decoder.

- VNSA-IM-PGDBF that is the imprecise version of the previously discussed solution. It replaces type-2 VNUs with type-3 VNUs, which are trivial. The synthesis results have shown that this approach further reduces the hardware cost to 18% with respect to GDBF.

From the decoding performance perspective, Monte-Carlo simulations have shown that the VNSA-PGDBF solution has the error correction as good as the optimized PGDBF implementation in the literature, while VNSA-IM-PGDBF, with imprecise maximum finder has an even better performance than VNSA-PGDBF. Therefore, VNSA-IM-PGDBF has proven a very good decoding capability, with low complexity and ultra high decoding throughput, making it suitable for the next communication and storage standards.

### 3.6.3 NS-FAID In Memory Centric Flooded LDPC Decoders [8]

As discussed in section 3.3 using framing and de-framing functions allows for storage requirements reduction, and for parallel processing units implementation, with a large number of units implemented in hardware it also creates incentives for improving working frequency and reducing implementation cost, all these coupled with similar or improved error correction performance with respect to Min-Sum. Although our main target have been layered architectures [16][8][9], the NS-FAID concept can be applied for flooding scheduling decoder implementations as well. Note that different from layered scheduling decoding where only check-node messages are stored to memory, for flooding scheduling decoders both the check-node and the variable node messages are stored. Consequence of this, compressing both of them leads to even larger memory savings. For the study in [18], we use a memory centric decoder having $B$ matrix number of rows of Check Node Units (CNUs), and $B$ matrix number of columns of VNUs. Thus, the parallelization in terms of number of units is rather small (*i.e.* circulant size $z$ $\times$ smaller) with respect to a fully parallel flooding scheduling implementation. This decoder architecture has been proposed for array codes, with an especially designed off-line scheduling algorithm that enables efficient overlapping between VNU and CNU processing [83]. The efficiency of the scheduling solution depends strongly on the choice of QC-LDPC code matrix. Note the focus of the study from . The throughput to area (TAR) metric improvement for this memory centric architecture is in the range 25% up to 110% compared to baseline Mi-Sum utilizing 4 bits quantization messages.

The partially parallel decoder implementation is memory centric. We have used the decoder architecture proposed in proposed in [83], without with employing the algorithm that computes the start indexes for the VNU and CNU operation, such that the execution of check-node and variable-node message memories. as well as their corresponding message storage is overlapped. Thus, the iteration time is $2 \times z$. Note that other orthogonal optimizations for this architecture have been proposed, such as vector folding [84], or the multiple codeword processing optimization for FPGA [85]. However, we focus in [18] our goal is to assess the impact of integrating NS-FAID kernels on logic and message storage, without considering other complementary architecture optimizations. Furthermore, we also evaluated the impact of integrating NS-FAID in the baseline for different types of hardware architecture choices and scheduling: layered scheduling, such as the one in [29] and the one using the unrolling principle in [9]. By using the $\Delta TAR$ metric, er can assess the opportunity of implementing NA-FAID- based MS decoders for different architectures. After careful analysis of the synthesis results as well as $\Delta TAR$, we have noticed that for [29] the main driver of $TAR$ increase is the working frequency increase. Since, the increase in frequency from $w = 2$ to $w = 3$ is rather modest, the $\Delta TAR$ improvement follows the same trend- from 42% to 58%. For architectures with a massive parallelism, such as [9], For the architecture in this paper, as well as the one in [9], the $\Delta TAR$ increase is mainly driven by cost savings, since using smaller quantization messages reduces processing unit complexity, especially for the CNU side. This is especially obvious for $w = 2$ bit messages, where both cost savings and an increase in working frequency with respect to the baseline architecture is also obtained.

[8] This subsection contains results and text partially reproduced from the conference publication [18]

Table 3.11: TAR Improvement with respect to the baseline architecture ($\Delta TAR$) of different NS-FAID Decoder Architectures. The [18] and [9] use a $(3, 12)$-regular LDPC code, while (3,6)-regular LDPC code. The length of all codes is 1296 bits. As in [18]

| Architecture | $\Delta TAR$ for $w = 3$ | $\Delta TAR$ for $w = 2$ |
|---|---|---|
| Memory-centric flooding [18] | 24% | 110% |
| Layered [29] | 42% | 58% |
| Unrolled layered [9] | 39% | 119% |

## 3.7 Conclusions

This chapter has discussed the issue of approximate computation in layered QC-LDPC decoders. Although at first the prime targets have been processing elements, later we have concluded that imprecise storage has a significant impact on the decoder implementation. Thus, for partially-parallel decoder architectures, and for ASIC technology implementations, the gain in approximate memory storage has proved comparable, and in some cases larger than the one obtained when focusing solely on the processing units. Therefore, we conclude that by jointly using approximate computation and storage, we can effectively optimize cost and throughput for the LDPC decoder architecture. Another important aspect touched down in this chapter is the issue of stopping the decoder. With respect to this we have two early termination criteria proposed, having different cost and safeness properties. Last, but not least, in section 3.6 we have presented some additional imprecise computation contributions for flooding scheduling QC-LDPC decoder architectures, using both MS, and PGDBF decoding algorithms.

# Chapter 4

# Layered Scheduling Memory Hierarchy Trade-offs

**Abstract:** *This chapter discusses the issue of memory design, as well as other related aspects of QC-LDPC decoder architectures using layered scheduling. During this discussion two main approaches are presented. The first we address the case of a given code and set of architecture parameters for which we must optimize the HUE metric. Second, we use the alternative approach, when the choice of code is not fixed, and we can do architecture-aware code design for a set of code parameters and a set of architecture parameters and assumptions.*

## 4.1 Introduction

A typical LDPC decoder architecture consists of memory modules for message storing, an interconnection network, as well as multiple parallel processing units. A key property of LDPC codes is represented by their capability to accommodate wide degrees of parallelism, that results in different throughput-cost trade-offs. Increasing throughput in layered LDPC architectures can be achieved by: *(i)* increasing parallelism, and *(ii)* increasing working frequencies. Regarding the former, this strategy leads to increased number of simultaneous memory accesses, which can be achieved by: *(1)* multi-port memories, and *(2)* multiple single port memory banks. Single port memories represent the low cost option independent of the technology choice. Furthermore, for FPGA devices, the most efficient implementation for memories - the Block RAM - does not allow more than two ports. However, using the multiple bank based organization may lead to memory access conflicts, as multiple messages may be required to be accessed simultaneously from the same bank. In order to increase working frequencies, pipelining the processing units is employed. However, this leads to delayed message write-backs in memories, and thus, to pipeline related hazards, such as read-after-write (RAW). For both memory access conflicts and pipeline related hazards, introducing stall cycles – idle clock cycles – is required. Therefore, a penalty in the iteration latency is obtained, as well as an inefficient usage of hardware resources.

Two approaches have been proposed in literature. On the one hand off-line algorithms that try to reduce memory access conflicts have been proposed [41][42][86][86][87][15], as well as solutions for pipeline hazard mitigation solutions through memory message access reordering strategies [37][38][39][15]. This alternative corresponds to the case of a given QC-LDPC code and a set of hardware architecture parameters. Alternatively, if the choice of code is allowed, we can proceed to a code construction solution. This is the approach presented in [19][35][36]. Note that finding a successful

solution for the second approach depends strongly on the choice of hardware architecture parameters, and code parameters.

This chapter is organized as follows: section 1 discusses the problem of message mapping and access for layered scheduling decoding, section 2 presents Notations, Definitions and Metrics, section 3 is focused on off-line tools used for already existing QC-LDPC codes, section 4 addresses the alternative solution, namely hardware-aware code design for obtaining a very high HUE. Last, some concluding remarks.

The contributions presented in this chapter have been disseminated in two journal publications:

- O.Boncalo, G.Kolumban-Antal, A.Amaricai, V.Savin, and D.Declercq, "Layered LDPC decoders with efficient memory access scheduling and mapping and built-in support for pipeline hazards mitigation" *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2018.

- O. Boncalo, G. Kolumban-Antal, D. Declercq, and V.Savin, "Code-design for efficient pipelined layered LDPC decoders with bank memory organization" *Microprocessors and Microsystems*, vol. 63, pp. 216 – 225, 2018. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0141933118300863`

This chapter attempts to offer an unified presentation of the message memory mapping and scheduling for hazard avoidance problem mitigation from both code-design and off-line dedicated algorithm for a fixed code perspective, under given generic layered scheduling decoder architecture constraints.

## 4.2 Notations, Definitions and Metrics [1]

### 4.2.1 Notations, Definitions

As discussed previously, architecture aware code construction allows the finding od an architecture friendly QC-LDPC code that enables maximum parallelism at processing unit level for a given processing latency, as well as a conflict free (AP-LLR) message mapping, and thus it optimizes hardware resource usage, while minimizing iteration processing latency. Before presenting the actual results, we start by describing the tools needed for formulating the problem. The message mapping into memory banks can be regarded as a graph coloring problem, as noted in [42]. Each memory bank has a corresponding color $c$, with $\{1 \dots n_{banks}\}$. We used $n_{banks}$ to denote the total number of memory banks. Note that this is an architecture parameters, with maximum value equal to $d_c^{max}$. Given a base graph $G$, and taking into account the quay-cyclic nature of QC-LDPC codes with a single non-zero entry for each column of the expanded graph $\tilde{G}$, the problem of finding a suitable mapping of AP-LLR messages is equivalent to coloring all the edges of each variable node using two restrictions:

- **AC1**: the maximum allowed number of colors used for coloring all the edges connected to a check-node $m$ is $n_{banks}$;

- **AC2**: all edges connected to a variable node $n$ are colored using the same color;

[1] This subsection contains definitions and text partially reproduced from the journal publications [15] and [19]. The notations have been unified such that they better reflect the contributions regarding optimized memory organization and scheduling of layered scheduling decoder architectures

In addition to this, we add the assumption that is a message is read from a bank, then its corresponding updated value is written back to the same bank several cycles later.

Note that we ended up edge coloring the bipartite graph $G$, corresponding to the base matrix $B$, having two types of nodes: variable-nodes and check-nodes. In order to formally describe the output of the edge-coloring process we define the concept of a coloring function, and denoted it by $\xi$.

**Definition 4.1.** *Given a set of variable nodes $N$, an $n_{banks}$ coloring of $N$ is a function $\xi\colon N \rightarrow \{1 \dots n_{banks}\}$, where $\xi(n) = c$ return the color $c$ assigned to all edges connected to variable node $n$.*

Consequently, we denote the set of variable nodes having their edges colored with a color $c$, as $N_\xi(c) = \{n | n \in N,\ \xi(n,m) = c,\ \forall m \in M(n)\}$. For simplicity, since the condition for correct coloring is that all edges connected to a variable-node $n \in N$ are colored with the same color (*i.e.* $\xi(n,m) = c,\ \forall m \in M(n)$), we used the simplified notation $\xi(n) = c$ instead of the edge perspective: $\xi(n,m) = c,\ \forall m \in M(n)$. In addition to this, instead of saying all edges of $n$ colored with color $c$, we say that variable $n$ has a corresponding color $c$ assigned to it. Similarly, when referring to the set of variable nodes colored with $c$, and connected to check-node $m$, we use the set notation $N_\xi(m,c) = N(m) \cap N_\xi(c)$.

For a check node $m \in M$, of a base graph $G$, colored by a function $\xi$, the *read access window* is the time needed for *reading* all the (AP-LLR) for check node $m$ processing from their corresponding memory banks. If no message access conflicts appear in any bank, the access window value is equal to the maximum number of neighboring variable nodes of $m$ with the same color assigned to them, and denote it by $T_G(\xi, m)$. This value can be lower bounds by:

$$T_G(\xi, m) = \max_{1 \leq c \leq N_C} |N_\xi(m,c)| \geq \left\lceil \frac{d_c(m)}{N_C} \right\rceil \tag{4.1}$$

For all check-nodes that are processed during an iteration processing, the sum of all the AP-LLR message *read access windows* from memory is denoted by $T_G(\xi)$, defined and lower bounded as:

$$T_G(\xi) = \sum_{m \in M} T_G(\xi, m) \geq \sum_{m \in M} \left\lceil \frac{d_c(m)}{N_C} \right\rceil \tag{4.2}$$

Let us denote all the edges of the base graph $G$, colored with $c$ by $\mathcal{N}_E(c)$, equal to the sum of the variable nodes degrees that have color $c$ assigned to them. $\mathcal{N}_E(c)$ can be expressed as:

$$\mathcal{N}_E(c) = \sum_{n \in N_\xi(c)} d_v(n) \tag{4.3}$$

After completing the task of mapping messages to memory banks, the next issue to address is the order in which messages need to be accesses and written back to memory. Several observations are in place regarding this second issue:

- the order in which check-nodes are processed (*i.e.* are mapped into layers) is not important, given that all check-nodes are processed once and only once during an iteration. Slight differences can be noted for the decoding performance. In general, they are considered irrelevant. Thus, we can assume:

  **A1**: The processing order of check-nodes of a base graph $G$, can be made arbitrary, and is denoted by the function $\sigma_G$.

- the order in which AP-LLR messages mapped to bank $c$, which are required for an arbitrary check-node $m$ processing can be made in which-ever order, under the assumption that:
  **A2**: All AP-LLR messages corresponding the variable nodes $n \in N(m)$ connected to check-node $m$ are accessed before reading any AP-LLR message required for the next check-node of the scheduling $\sigma$.
  To sum up, the AP-LLT message read for check-node $m$ is an atomic operation.

**Definition 4.2.** *A check-node ordering function is a bijection $\sigma\colon M \to \{1 \dots |M|\}$ that maps each check-node to a layer, determining the check-node processing order. Check node number refers to the natural ordering, while layer number refers to the rescheduled ordering. Indexing starts from 1.*

A clock cycle when a read access is performed from the AP-LLR memory bank $c$ is called a emphread clock cycle. Alternatively, if no read access is performed, we say that a *stall clock cycle* with respect to bank $c$ takes place.

With these assumptions in mind, we further note that providing such a message read access scheduling, is in fact equivalent to visiting all edges $(m, n)$ of the graph $G$, and assigning access orders for their corresponding memory bank read. Assigning these access time orders, can be done in a check-node by check-node manner, taking into account a given check-node scheduling $\sigma$. In addition to this, since the access order is with respect to a memory bank, we can say that we are ordering all variable nodes $n \in N(m)$ for each color separately, under the assumption **A2**. Thus, we obtain a read access scheduling function $\psi(m, n)$ defined as follows:

**Definition 4.3.** *Given a base graph $G$, colored with a coloring function $\xi$, the read access scheduling function $\psi(m, n)$ indicates the processing order of variable node $n$, connected to a check node $m$ and colored with color $\xi(n)$. Indexing starts from clock cycle 1.*

Note that so far we have only touched on the issue of AP-LLR message read access scheduling. After the variable-node and check-node processing, the value of the AP-LLR message is updated such that the current layer contribution is added. The order in which messages are written back is given by the processing duration function of a variable node $n \in N(m)$, denoted by $pd(m, n)$, gives the number of clock cycles from the moment the AP-LLR message corresponding to variable node $n$ is read, until it is written back to memory. The expression for the $pd$ function reflects the choice of write back strategy for the decoder architecture. Furthermore, it is a decoder architecture parameter since it dictates the storage type for the variable-node messages inside used during a layer's processing, and more specifically during the AP-LLR update. If the variable-node messages are stored in a first-in first-out (FIFO) memory, it means that their update order for an arbitrary message scheduled the $i$-th for memory access read is the same as the read order [19]:

$$pd_n(i, i_{max}) = i_{max} + n_{banks} - 1 \tag{4.4}$$

When the a stack is used, the update order is revered with respect to the AP-LLR memory read order:

$$pd_r(i, i_{max}) = 2 \cdot (i_{max} - i) + n_{banks} \tag{4.5}$$

Where $i_{max}$ is used to denote the total number of reads per banks required for check node $m$ processing.

### 4.2.2 Metrics

Having the most important concepts introduced in the previous subsection, we continue by describing the metrics used to assess the efficiency of the output results. For this purpose we first start by describing the concept of perfectly colorable bipartite graph $G$.

**Definition 4.4.** *A base graph $G$ is* perfectly colorable *by $\xi$ if $\forall m \in M, \forall c \in \{1 \dots n_{banks}\}$, we have:*

$$|N_\xi(m,c)| \leq \left\lceil \frac{d_c(m)}{n_{banks}} \right\rceil \tag{4.6}$$

**Definition 4.5.** *A base graph $G$, colored with an $n_{banks}$ coloring function $\xi$ is a* balanced colored graph *if $\forall m \in M, \forall c \in \{1 \dots N_C\}$:*

$$|N_\xi(m,c)| = \frac{d_c(m)}{N_C} \tag{4.7}$$

Hence, a balanced colored graph is also perfectly colored by its coloring function.

**Definition 4.6.** *Given a base graph $G$, colored with $n_{banks}$ coloring function $\xi$, and a processing order $\psi$ of $G$ that introduces no read after write hazards, the Hardware Usage Efficiency (HUE) of the decoder is defined as:*

$$HUE(G,\xi,\psi) = \frac{|E|}{N_C \cdot T_G(\xi)} \cdot 100\ [\%] \tag{4.8}$$

A Hardware Usage Efficiency of 100% means no additional stalls due to message access conflicts in memory banks, and corresponds to the case of a balanced colored graph.

Throughput is the other important measure for a decoder's efficiency reflecting both scheduling, memory organization and message mapping, as well as implementation efficiency. Let $Stall(\psi)$ account for all pipeline stalls introduced during an iteration due to scheduling.

**Definition 4.7.** *For a graph $G$, with coloring function $\xi$, and scheduling $\psi$, having an architecture processing latency of $n_{latency}$, the decoding throughput can be expressed as:*

$$T(G,\xi,\psi) = \frac{F_{clk} \cdot |N| \cdot z}{(T_G(\xi) + Stall(\psi)) \cdot N_{It} + n_{latency}} [bps] \tag{4.9}$$

From the scheduling and mapping point of view, what we try to optimize is the iteration latency, or iteration duration. Note that this measure can have the same value for two distinct layered decoder implementations. Furthermore, it can be computed off-line without any implementation at hand. It is relevant for a first assessment of the scheduling and mapping quality. It has been defined in [8] as follows:

**Definition 4.8.** *Given a base graph $G$, a given $n_{banks}$ coloring function $\xi$, and a scheduling of $G$ given by the $\sigma$ and $\psi$ pair, we define the* decoder iteration latency *as the total number of clock cycles needed to read, process and write-back all the AP-LLR messages, including those needed in order to avoid any potential hazards in-between the execution of successive iterations, and denote it by $I_G(\xi,\sigma,\psi)$.*

Based on the decoder iteration latency, we can refine it into another more global metric that accounts for both mapping conflicts and pipeline stalls for the overall decoder architecture– the hardware usage efficiency. It characterizes the dataflow inside the decoder, and how much of the memory bandwidth is used efficiently.

**Definition 4.9.** *The Hardware Usage Efficiency (HUE) is defined as the fraction of the useful memory access clock cycles out of the total number of available memory accesses ($I_G(\xi, \sigma, \psi) \cdot n_{banks}$).*

$$HUE(G, \sigma) = \frac{\sum_{m \in M} d_c(m)}{I_G(\xi, \sigma, \psi) \cdot n_{banks}} \cdot 100 \tag{4.10}$$

With these two measures at hand, the throughput (T) can also be expressed as [15]:

$$T = (|N| \times F_{clk}) \,/\, (N_{It} \times I_G(\xi, \sigma, \psi)) \tag{4.11}$$

Note that independent of the equation used, the throughput is linearly dependent on the working frequency ($F_{clk}$).

## 4.3 Pipeline related Data hazards and Message Mapping Problems in Layered Scheduling Decoding [2]

### 4.3.1 Problem statement

The key architecture parameters when designing a partially parallel LDPC decoder are memory design building blocks, pipeline, and parallelism. Furthermore, of great importance is the choice of decoding scheduling that dictates the memory access patterns and the memory blocks parameters constraints (*e.g.* width, number of access ports, number of storage locations, access order, update delay). It is well understood that the most inexpensive memory building block is the single-port memory SRAM, while the high speed and also larger cost is register based storage, such as register files (RF). With this idea in mind, we have developed a set of off-line algorithms that aim at optimizing the memory access patterns and memory message mapping such that the iteration latency is minimized [15]. Furthermore, results showed that in spite of the best effort, some code matrices did not have the potential for obtaining good HUE results. The empirical findings have been reinforced by the results presented in AL-PEG paper on code constriction methods targeting 100%, or close to 100% HUE. More specifically in [19] we have extracted two necessary properties of the input parameters for building the QC-LDPC code, such that code construction is possible.

As a workaround this issue, we introduced flexibility inside the underling hardware architecture, such that the pipeline constraint can be relaxed. In order to perform a thorough evaluation, we have considered a hardware architecture with the following characteristics: memory is organized in $n_{banks}$, with each input AP-LLR message being accessed fro a different bank. Furthermore, the memory banks are implemented using SRAM tiles, which are single port. Given the structure of the QC-LDPC code, we consider the case of $Z$ circulant size number of parallel processing units (*i.e.* VNU and CNU processing units). According to the layered decoding principle, each variable-node $n$ is updated $d_v(n)$ times during an iteration. Consequently, the $\gamma_n$ message containing the equivalent of the contribution of all check-nodes and channel input is updated $d_v(n)$ times. This leads to the *superior convergence of layered scheduling* decoding, as compared to flooded decoding that only performs one VN update per iteration [53]. One pitfall, consequence of this fact is that in order to have correct layered scheduled decoding, the AP-LLR messages that are common for successive layers need to have their values updated, otherwise the check-node message contribution from the earlier scheduled layer

[2] This subsection contains definitions, figures and text partially reproduced from the journal publications [15].

is lost. This constraint in processing and message memory access and update is tightly related to pipelining. Although, pipeline is a very good technique for increasing working frequency, it is also responsible for causing a delayed update effect, that may give rise to data hazards, such as read-after-write (RAW). Thus, in terms of pipelining and memory access latency, we consider the case of an arbitrary number expressed by the $n_{latency}$ value parameter.

Note that the message mapping in single port banks, and the problem of mitigating pipeline related hazards are in fact two distinct problems. The reason is related to architecture choices. The works that have investigated pipeline mitigation hazards, discussed a serial architecture, where serial is understood as one message processed by each unit at a time [37][38][39][40]. The mapping related work deals with an architecture that is based on fully parallel processing units. Therefore, we start with first discussing the mapping approaches and then continue with the pipeline hazard mitigation algorithms.

### 4.3.2 Off-line algorithm optimizations for a given code

The message mapping into memory banks can be translated into a graph coloring problem. The closest related work to the proposed solution in this work is [87]. It uses edge coloring of the resulted conflict graph and color it using $d_c^{max}$+1 colors (*i.e.* $d_c^{max}$+1 banks). This approach is valid if the resulting conflict graph is a simple graph. A key difference with respect to our work [15] is represented by the number of processing units that work in parallel in the hardware LDPC decoder architecture. We use $z$ parallel units. consequence of this, the coloring is not performed on a conflict graph, instead the $G$ graph, which corresponds to $B$ base matrix needs to be colored. Furthermore, the choice of relatively small number of processing units working in parallel (*e.g.* 4, 6, 16) from [87] increases the odds that the resulting graph used for the coloring problem does not contain 4 cycles. he 4-cycle free requirement in [87] ensures that the colored graph is a simple graph, subject to Vizing's theorem that guarantees the existence of a mapping solution for at most $n_{banks} + 1$ memory banks. Otherwise, as noted in the concluding remarks of [87] the associated coloring graph contains multiple edges and Vizing's theorem does not longer apply. The LDPC case study from [87] features a non-binary LDPC, with $d_v = 2$ and without 4-cycles. However, even for QC-LDPC codes with $d_v = 2$, most $G$ graphs associated with the base matrix B contain 4-cycles. Moreover, for QC-LDPC codes with $d_v > 2$, the conflict graph is actually a hypergraph, thus transforming the message mapping problem into a hypergraph coloring problem. Furthermore, the reduced number of units imposes different design considerations. In this case, the size of the control read only memories becomes comparable to that of message memories. Hence, techniques for reducing its size are needed. In addition to this, different options for the read stage can be used, such as the anticipated reading of some messages for latter use. Other related approaches are: the pioneering work recognizing the mapping problem for Turbo decoders from [41], which uses a simulated annealing based solution, the work from [42] where it has been expressed into an equivalent conflict graph coloring problem subject to Vizing's theorem. The conflicts that could not be resolved are usually settled by stalls or, as proposed in [88], by introducing registers and extra multiplexing logic. Alternatively, the multiple read multiple write (MRMW) strategy eliminates the needs for extra registers; this strategy relies on reading and writing the messages in different memory banks. Its drawback is the additional hardware overhead in both routing and control.

### 4.3.3 Code-construction based approaches

There are several methods to construct LDPC codes. One such method is the Progressive Edge Growth (PEG) algorithm proposed in [89]. Its aim is to construct $\tilde{G}$ with as large as possible girths. The girth of a graph is given by the length of the smallest cycle of the graph. The original PEG algorithm has been adapted to build QC-LDPC codes [90][91]. Furthermore, PEG variants exists, one particular example is that of [92] that tries to build QC-LDPC codes taking into account the underlying harware architecture features for serial processing VNU and CNU units. The specific harware implementation case addresses is of $n_{banks} = 1$, with write-back order of AP-LLR messages identical to the read order. This modified PEG variant addresses only the case of mitigating RAW hazard caused by pipeline by code-design.

The work from [93] targets software-defined radio codes. It uses code-design to build a code with conflict-free mapping. More specifically, the condition proposed is that the number of AP-LLR messages grouped into a memory bank must be approximatively the same [93]. In a sense this is the intuition behind the theoretical property from [19], a property that can be computed a-priori to the actual code construction. In addition to this it is worthwhile emphasizing that the approach presented identifies in fact only a sub-set of possible solutions. In [19] we show the existence of situations when the aforementioned requirement is *not* fulfilled, but conflict-free mapping is possible. Furthermore, [93] does not cover the pipeline issue. An addition approach that tries with the same goals as our has been proposed by [36]. From the code construction point of view it from [19] in the sense that it colors edges first, and then builds the graph. Furthermore, the pipeline issue is not handles by code-design. Instead is an off-line step performed after the code-construction. Additionally, the work from [19], also identifies two important theoretical properties, that can be computed before actual code-construction PEG is run, in order to check whether the constrction is at all possible. Thus, we conclude that the contribution from [19] is the most complete approach addressing architecture-aware code-design for layered scheduling QC-LDPC decoding proposed so far.

## 4.4 Residue Based Layered Decoding and the supporting Off-line algorithms for improving the HUE metric [3]

### 4.4.1 Residue Based Layered Decoding

Since even a small percentage of missed updates may cause significant performance degradation [57], the contribution of all variable-nodes must be added to the AP-LLR by adding the corresponding newly computed check-node message. However, due to data dependency, in some cases we need to wait for this update to finish inside de decoder. Instead of waiting, we compute the amount of this contribution and store it in the form of a residue. During subsequent updates to the same AP-LLR message for which the residue has been computed, we make sure to add this contribution as well. Note that it is possible that the residue for the same AP-LLR is accumulated for more than one AP-LLR update. The length of the sequence of AP-LLR updates for which the residue is stored is denoted by the architecture parameter $n_\Omega$. The actual computation of the residue is describes as [15]:

$$\delta_{t(r',n)} = \beta^{new}_{r',t(r',n)} - \beta^{old}_{r',t(r',n)},\ \forall n \in M(m') \tag{4.12}$$

[3] This subsection contains definitions, figures and text partially reproduced from the journal publications [15].

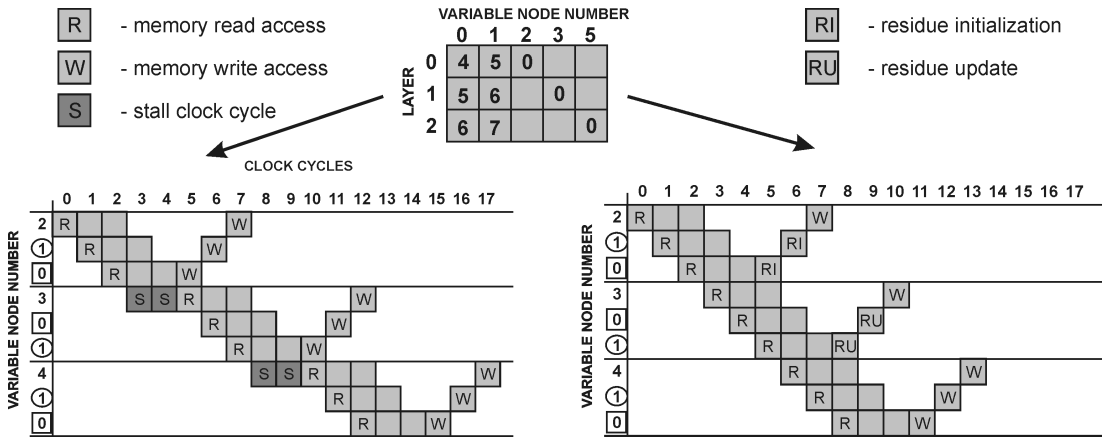



Figure 4.1: Data-flow for $n_{latency} = 4$ conventional layered decoder (left) and residue-based layered decoder with $n_\Omega = \infty$ (right), as in [15]

where $r' \in H(m')$, and the tags *new* and *old* refer to the currently and previously computed check-node message during check-node $m'$ processing. We say that we have an overlapping update for some variable-node $n$ of length $n_\Omega$. In order to be able to described the residue-based layered decoding principle we start by introducing the following definitions [15]:

**Definition 4.10.** *Given a 3-uple $G\langle\sigma, \xi, \psi\rangle$, the sequence of overlapping updates for a variable-node $n$, denoted by $M_\Omega(n)$, is the maximal set of check-nodes $\{m_1, \ldots, m_i, \ldots, m_l\}$, with $l \geq 2$, such that $\Omega(n, m_{i-1}, m_i) = true$, $\forall 2 \leq i \leq l$.*

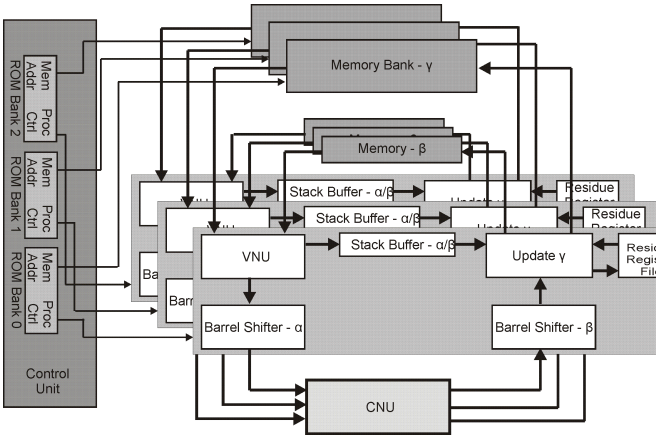
Furthermore, we define the following boolean operators $fist_\Omega(n, m)$ and $last_\Omega(n, m)$, which return *true* iff $l_\Omega(n, m) > 0$ and $m$ is the first, and respectively the last, element of the unique overlapping sequence it belongs to.

The changed layered decoding scheduling algorithm is depicted in Algo. 3 with the modified operations highlighted by the word modified. Furthermore, the example from Fig. 4.1 shows a data-flow execution. It becomes apparent that the overlapping updates are carried out in a flooded schedule fashion, while non-overlapping updates are carried out in a conventional layered schedule fashion from the AP-LLR updating procedure.

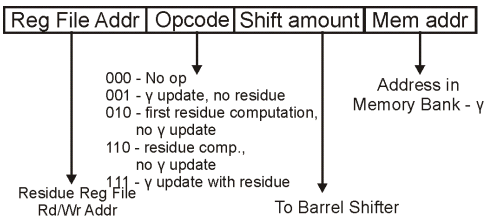
The employed layered architecture is depicted in Fig. 4.2a, having three key parameters $\langle n_{banks}, n_{latency}, n_\Omega\rangle$. Different than the architecture from section 2.4 – Fig. 2.8, the work from [15] employs separate processing units – VNU, CNU and AP-LLR update – , while the Barrel Shifters (BS) are used to route the $\alpha$ and $\beta$ messages, instead of the AP-LLRs, used in most layered decoding architectures. This is a cost-effective solution since the $\alpha$ and $\beta$ messages use a lower quantization with respect to the AP-LLR messages. The correct data interpretation and processing corresponding to the layered LDPC decoding is handled by control signals and specific opcode messages from the control unit. This includes: (*i*) AP-LLR memory addresses, (*ii*) shift amounts for the two barrel shifters, (*iii*) control signals for processing units, and (*iv*) control signals for stack buffers. The control unit contains $n_{banks}$ Read Only Memories (ROM), with the entries being depicted in Fig. 4.2a-b. ROM contents are generated off-line by the proposed bank mapping and scheduling algorithm, that will be presented in Section 4.4.2.

In order to relax the scheduling constraints, we employ the residue based layered decoding scheduling. The architectural modifications with respect to the baseline layered scheduling decoder consist

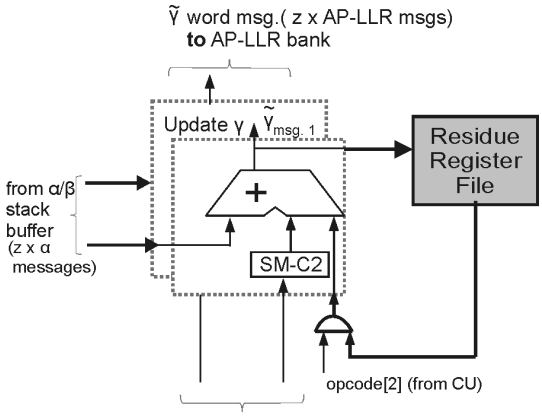
 




(a) Layered QC-LDPC architecture with $n_{banks} = 3$ single port memory banks for AP-LLR memory




(b) Control ROM data entry




(c) AP-LLR update unit

Figure 4.2: Generic Layered QC-LDPC architecture for residue-based LDPC decoder: (a) Overview; (b) Control ROM data entry for AP-LLR bank $i$; (c) AP-LLR processing block corresponding to $z \times \tilde{\gamma}$ messages updated to bank $i$, as in [15]

of:

1. AP-LLR update unit – supports three types of operations: $(i)$ conventional AP-LLR update, $(ii)$ storage of the residue $\delta$ message, and $(iii)$ AP-LLR update with residue based on the opcode field.

2. Extra storage for the $\delta$ residue message with appropriate control signals.

3. Control ROM *opcode* information, as depicted in Fig. 4.2b.

Note that the $\alpha$ stack buffer re-used for storing the previous $\beta$ message, needed during the residual computation. For this purpose, it has been renamed as $\alpha/\beta$ stack buffer to emphasize its double-role. In addition to this, we emphasize that everything is embedded in the control information that is computed off-line by the set of algorithms (*i.e.* no hardware support is required for checking that the maximum overlap length $n_\Omega$ is reached, or for differentiating between the different information stored in the $\alpha/\beta$ stack buffer).

### 4.4.2 The Off-line Algorithms

The off-line algorithms proposed in [15] optimizes the message memory mapping into memory banks, as well as message accesses such that the iteration duration $I_G$ is minimized, and resource usage is maximized. The process and set of algorithms is depicted in Fig. 4.3. The inputs and outputs are as follows:

- Inputs:

**Algorithm 3** Residue-Based Layered scheduling decoding principle[15]

**Initialization:**
**set** $\beta_{r,t}$ messages to 0
**set** $\gamma_t$ to channel LLR values
**set** $It = 0$ and $syndrome = false$
**while** $(It \leq It_{max})$ and $(syndrome = false)$ **do**
  **for all** $m \in M$ **do**
    **for all** $r \in N(m)$ **do**
      **Modified Variable-Node:**
      $\alpha_{r,t(r,n)} = \gamma_{t(r,n)} - \beta^{old}_{r,t(r,n)}, \forall n \in N(m)$
      **if** $first_\Omega(n,m)$ **then**
        $\delta_{t(r,n)} = 0$
      **Check-Node:** compute $\beta_{r,t}$ messages
      $\beta_{r,t(r,n)} = \prod_{n' \in N(m)\setminus\{n\}} sign(\alpha_{r,t(r,n')}) \times \min_{n' \in N(m)\setminus\{n\}} \alpha_{r,t(r,n')}, \forall n \in N(m)$
      **Modified AP-LLR Update:**
      **if** $last_\Omega(n,m)$ **then**
        $\gamma_{t(r,n)} = \alpha_{r,t(r,n)} + \beta^{new}_{r,t(r,n)} + \delta_{t(r,n)}, \forall n \in N(m)$
      **else**
        **if** $|M_\Omega(n)| > 1$ **then**
          $\delta_{t(r,n)} += \beta^{new}_{r,t(r,n)} - \beta^{old}_{r,t(r,n)}, \forall n \in N(m)$
        **else**
          $\gamma_{t(r,n)} = \alpha_{r,t(r,n)} + \beta^{new}_{r,t(r,n)}, , \forall n \in N(m)$
  **compute** new *syndrome* according to sign($\gamma$) values
  **set** $It = It + 1$
**Offloading:** output $\leftarrow$ sign($\gamma$)

  - code parameter: the base graph $G$ and its characteristics: $|M|$, $|N|$
  - architecture specific parameter: the update rule for the AP-LLR messages, which describes the order in which AP-LLRs are written-back to memory with respect to their read access. It is expressed formally by the function $pd(m, n)$.
  - architecture specific parameter number of banks – $n_{banks}$ value,
  - architecture specific parameter the pipeline update delay expressed in clock cycles – $n_{latency}$
  - the maximum allowed number of successive residual updates $n_\Omega$

- Outputs:
  - ROM file which contains the addresses for writing the channel input LLRs inside the memory during decoder initialization and offloading – the message mapping information is used for generating this file;
  - ROM file for each bank with the access information for each AP-LLR block message read - both message mapping and $\psi$ function is used to generate this information;
  - ROM memory with the check-node order and number of reads per check-node, and the in-between check-node stall information – the $\sigma$ function is needed, as well as the hazard avoidance stall information based on the current $\psi$ function;
  - the number of register file entries, as well as opcode information inside the ROM file for each bank with the access is required in case of $n_\Omega \geq 1$.

The sequence of execution of the set of off-line algorithms, depicted in Fig. 4.3, can be partitioned in 3 steps (for detailed description see [15]):

**Step 1:** aims at obtaining a check-node order, $\sigma$, by minimizing the number of overlapping updates. The check node ordering is a combinatorial optimization problem, that can be expressed as a Traveling Salesman Problem (TSP) (see [94] for TSP description). This TSP genetic solver involving crossover and mutation is a typical approach for obtaining $\sigma$, and it has been used with different update rules and architecture latency constraints [38][40][37]. The common variable nodes between consecutive check nodes are the main source of pipeline stalls. Hence, we define a fitness function that depends on the characteristics of the proposed residue-based architecture and it accounts for the sequences of consecutive common check nodes taht are allowed by the $n_\Omega$ architecture parameter. The crossover is inspired from [95]. We select a random divide $i_d$ point between 2 and $|M| - 1$, and we form the child check node ordering for the first $i_d$ layers, by selecting the check-nodes mapped in the first $i_d$ layers of the first parent, while the rest of the layers is filled with the remaining check-nodes that are ordered according to the $\sigma$ ordering of the second parent. *Mutation* is a simple permutation of a small number of layers. The output of this step is the reordered base graph $G\langle\sigma, \_, \_\rangle$.

**Step 2:** compute the actual message bank allocation by transforming the problem in a coloring problem. The coloring function $\xi$ is determined using the proposed modified Misra-Gries algorithm followed by a genetic optimization. Hence, the output is $G\langle\sigma, \xi, \_\rangle$. Each color corresponds to an AP-LLR memory bank; hence, bank mapping is inferred from the coloring result. The steps 1 and 2 are interchangeable.

**Step 3:** aims at determining the order in which the neighboring variable nodes of a given check-node are to be processed. This order has no impact on the layered LDPC decoding algorithm, however, it is important for the architecture since it may give rise to stalls due to RAW hazards. For variable node processing the order and more specifically the clock cycle during which the AP-LLR messages are read is determined. The check-node messages are not shared between layers, hence no conflict potential exists for the later. This is computed by a constructive in-layer re-ordering algorithm followed by specially designed tabu search algorithm. The iteration duration can be computed at the end of this step.

The iteration duration can be computed at the end of step 3.

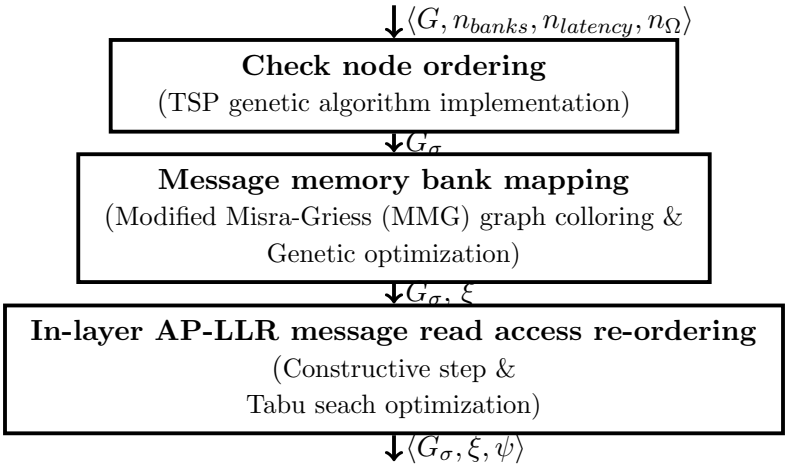



Figure 4.3: The execution order of the off-line algorithms, from [15]

### 4.4.3 Design Space Explorations and Discussions

The design space exploration comprises of 3 parts:

**First,** the evaluation from [15] start with running the off-line algorithms for the following input parameter values:

- code matrix: 6 standard codes have been considered: WiMAX [7] rates: 1/2, 2/3, 3/4, 5/6 of code length 2304 bits, $z = 96$, and $|M| = 24$ columns base matrix $B$ columns, with $d_c^{max}$ equal to 7 having $d_c^{max}$ equal to 7, 10, 15, and 20 respectively -, and two long codes - DVB-S2X codes of rate 28/45 and the DVB-S2X rate 140/180 [21]. The DVB-S2X codes have 180 columns in the base matrix, a code length of 64800, and $d_c^{max} = 10$ for rate 28/45 and $d_c^{max} = 20$ for rate 140/180;
- $pd$ – corresponding to reversed write-back rule;
- 4 values for the $n_{latency}$ parameter – $\{2, 3, 4, 5\}$,
- 4 up to 6 values for the $n_{bank}$ parameter such that the maximum theoretical HUE is maximized (*i.e.* a divisor of $d_{max}^c$)
- 3 different $n_\Omega$'s – $\{0, 1, \infty\}$.

With respect to the overlapping parameter $n_\Omega$, and the set of off-line algorithms, four distinct scenarios were considered for the HUE comparison:

- the natural order message read access scheduling without any architecture support ($n_\Omega = 0$),
- the case when the off-line algorithm is used for message mapping and optimized scheduling, however no architecture support for overlapping updates exists ($n_\Omega = 0$),
- the case when the off-line algorithms are used and the architecture supports only length 1 overlapping sequences ($n_\Omega = 1$),
- the case, when we use both the off-line scheduling algorithms, and we have unbounded overlapping sequences ($n_\Omega = \infty$).

Results are presented in Fig. 4.5.

**Second,** the Monte-Carlo simulation assessing the impact of the residue–based scheduling overlapping and the message access re-orderings in terms of FER statistics has been reported for the Additive White Gaussian Boise (AWGN) channel for 30 decoding iterations. The plotted the statistical curves for the frame error rate (FER) for WiMAX rate 3/4 code, with $n_{banks} = 3$, and $n_{latency} = 4$, using reverse write-back for three distinct values of $n_\Omega$: 0, 1, and Infinity ($\infty$) are presented in Fig. 4.4. No significant degradation can been noted between the plotted scenario curves.

**Third,** FPGA implementation results and other state-of the-art are presented in Table 4.1. The TAR metric suggests that in spite of the additional RF and pipelining implementation cost overhead, the residue-based layered scheduling decoders have a superior performance, due to the efficient scheduling that allows integration of additional pipeline levels.

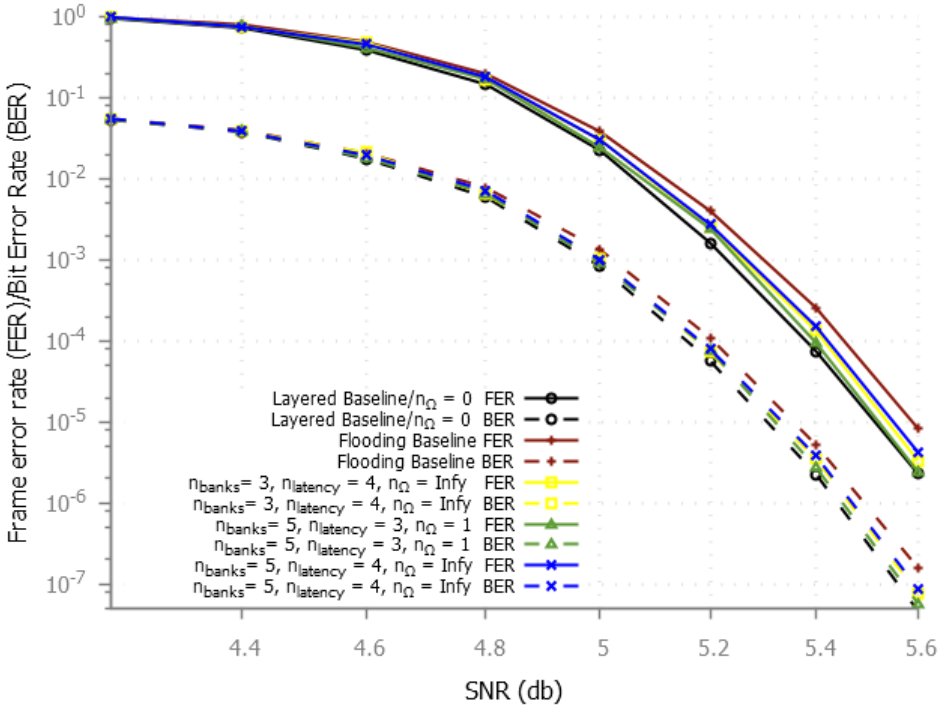
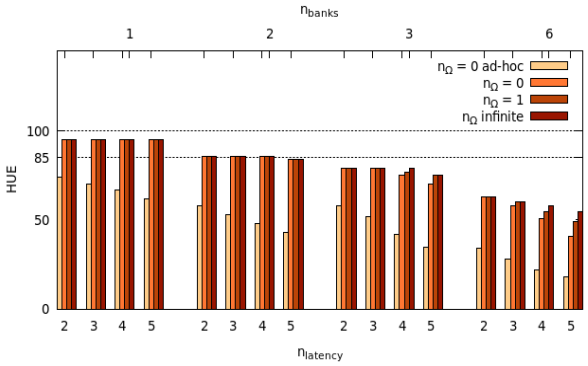
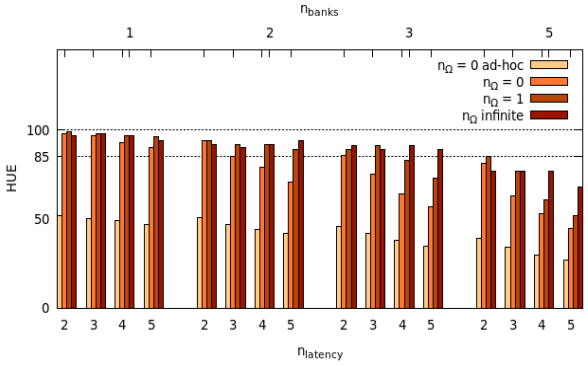


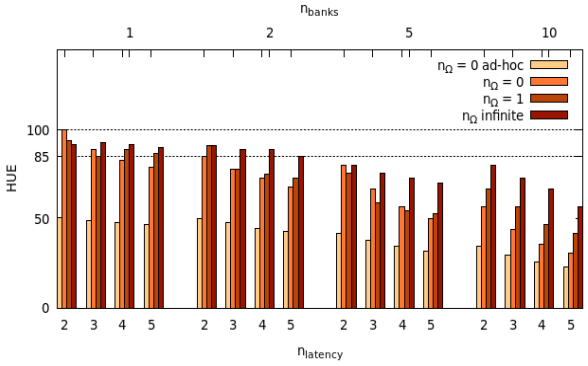

Figure 4.4: Frame error rate (FER) for WiMAX rate 3/4 code, with $n_{banks} = 3$, and $n_{latency} = 4$, using reverse write-back for three distinct values of $n_\Omega$: 0, 1, and Infinity ($\infty$) [15]



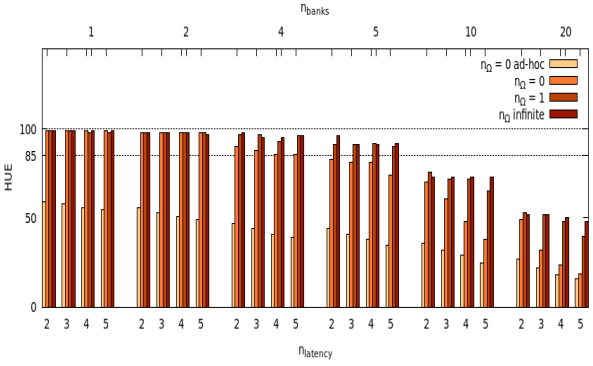

(a) HUE – WiMAX rate 1/2



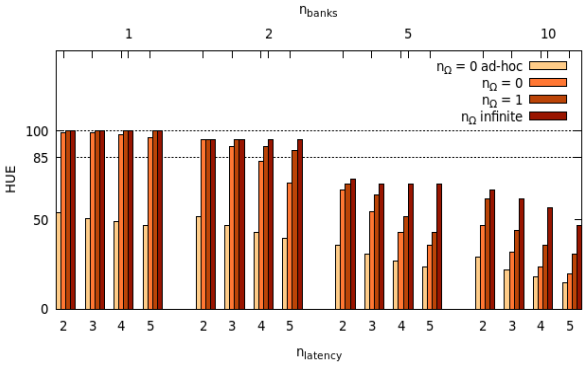

(b) HUE – WiMAX rate 3/4



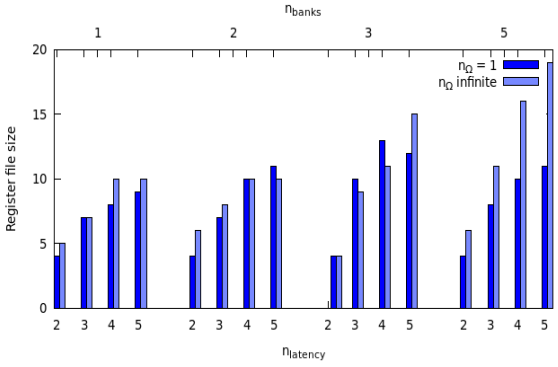

(c) HUE – WiMAX rate 5/6



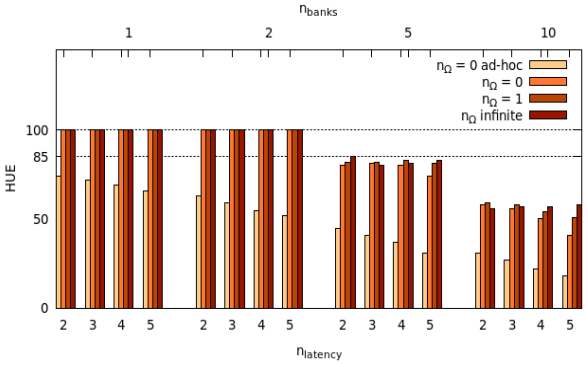

(d) HUE – DVB-s2x rate 140/180



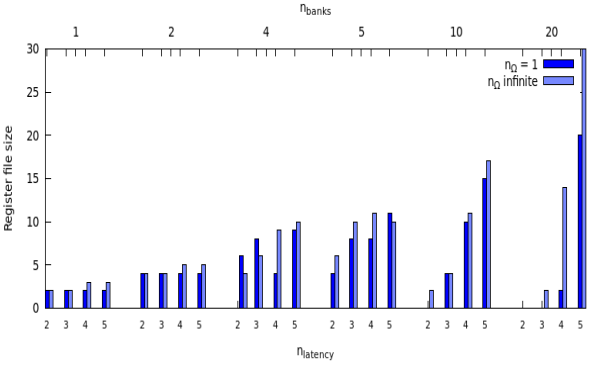

(e) HUE – WiMAX rate 2/3




(f) Total number of RF entries – WiMAX rate 3/4




(g) HUE – DVB-s2x rate 28/45




(h) Total number of RF entries – DVB-s2x rate 140/180

Figure 4.5: Evaluation for the proposed off-line algorithms for 6 standard codes using the HUE metric. (f), (h) report the total number of RF entries needed for residue-based scheduling, [15]

Table 4.1: Implementation results for WiMAX and DVB codes for the selected $\langle n_{banks}, n_{latency}, n_{\Omega} \rangle$ configuration parameters and related works, [15]

| | $\langle n_{banks}, n_{latency}, n_{\Omega} \rangle$ | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Decoder | $\langle 3,2,0 \rangle$ | $\langle 5,2,0 \rangle$ | $\langle 3,4,\infty \rangle$ | $\langle 5,4,\infty \rangle$ | $\langle 5,2,0 \rangle$ | $\langle 5,4,\infty \rangle$ | [29] | [96] | [97] | [84] | [27] |
| Code size[bit] | 2304 (WiMAX rate 3/4) | | | | 64800 | | 2304[1,2] | 2304 | 2304[3] | 1536[1,2,3] | 1296[1,2,3] |
| Device | Virtex-7 VC707 (xc7vx485tffg1761-2) | | | | | | Zynq7000 | Zynq7020 | Virtex 5 | Virtex 2 | Virtex 6 |
| Slices | 8576 | 13070 | 12496 | 18748 | 55511 | 59874 | 9776 | 4446 | 5583 | 6102 | NA |
| Slice regs. | 15543 | 17017 | 26925 | 35013 | 76008 | 112613 | NA | 3086 | 3992 | 9263 | 23352 |
| Slice LUTs. | 29688 | 46136 | 40700 | 63832 | 188145 | 198810 | NA | 13555 | 18542 | 9698 | 95188 |
| No. BRAMs | 40.5 | 67.5 | 40.5 | 67.5 | 252.5 | 252.5 | 0 | 24 | 160 | 24 | 144 |
| $f_{max}$[Mhz] | 90.9 | 75.1 | 142.8 | 125 | 45.5 | 80 | 98 | 150 | 126 | 149.8 | 164 |
| $n_{iter}$ | 10 | | | | | | 20 | 10 | 7.5 | 3 | 20 |
| $I_G$[cc] | 33 | 21 | 31 | 22 | 189 | 173 | 12 | 180 | 42 | 64 | 6 |
| T[Mbps] | 634.7 | 824.9 | 1061.7 | 1309 | 1525.7 | 2996.5 | 527 | 47-341 | 432 | 830.6 | 1770 |
| $T^N$[Gbps] | 6.34 | 8.24 | 10.77 | 13.09 | 15.26 | 29.96 | 10.5 | 1.920 | 6.912 | 35.95 | 31.1 |
| TAR[Mbps/slices] | 0.74 | 0.63 | 0.84 | 0.69 | 0.27 | 0.50 | 1.078 | 0.431 | 1.238 | 0.589 | NA |
| Quantization[bit] | 4 | | | | | | 4 | 4 | 2 | 8 | 4 |

(1) special $B$ matrix properties needed; (2) no flexibility with respect to changing $B$; (3) regular (3,6) code; NA - not accounted

## 4.5 Layered Scheduling Aware Code Design for Pipelined Architectures with Memory-Bank based Memory Organization [4]

### 4.5.1 Theoretical constraints

The coloring function $\xi$ of the base graph abstracts the mapping of the AP-LLR messages into $n_{banks}$ memory banks. During an iteration processing the check-nodes mapped into layers, hereafter simply referred as layers, are processes one by one. During layer processing, firs all the AP-LLR messages are read from memory. These messaged correspond to the neighbored variable nodes of the current check node. Next, variable node processing followed by check-node processing takes place. For, the later in MS-based decoding the first and second minimum of all $\alpha$ variable-node messages of the currently processed check-node is computed. Afterwards, comes the update, which dictates the order in which messages are written back to memory. Note that for all these processing types, pipeline might be used. Hence their response is delayed a number of cycles (equal to the number of pipeline levels).

Let $\mathcal{S}$ denote the sum of the processing durations, of all AP-LLR messages:

$$\mathcal{S} = \sum_{(m,n)\in E} pd(\psi(m,n), T_G(\xi, m)) \tag{4.13}$$

Alternatively the Eq. (4.13) for $\mathcal{S}$ can be rewritten in terms of the graph construction input parameters – the update rule $pd$, the check node degrees, and the number of colors:

$$\mathcal{S} = \sum_{m\in M} \sum_{k=1}^{\frac{d_c(m)}{n_{banks}}} (n_{banks} \cdot pd(k, \frac{d_c(m)}{n_{banks}})) \tag{4.14}$$

The pipeline theoretical constraint for the code matrix parameters is summarized in the following Proposition and has been proposed in [19]:

**Proposition 1.** *Let $G$ be a base graph, balanced colored by an $N_C$ coloring function $\xi$, assuming that the read $rd$ and write $wb$ functions corresponding to $G$ do not introduce pipeline hazards. Than the following relation is true:*

$$\mathcal{S} \leq \frac{|N| \cdot |E|}{n_{banks}} \tag{4.15}$$

*Proof (Proposition 1):* Let us consider a variable node $n$, and its neighbor check nodes $M(n) = \{m_1, m_2, \dots m_k\}$ where $k = d_v(n)$. The pipeline constraints can be expressed using the read (rd) and write (wb) time access functions as a system of inequalities:

$$\forall 1 \leq i < k, wb(m_i, n) + 1 \leq rd(m_{i+1}, n)$$
$$wb(m_k, n) + 1 \leq rd(m_1, n) + T_G(\xi)$$

Adding the relations results, and moving the sum of read clock cycles to the right side, results in:

$$\sum_{m\in M(n)} (wb(m,n) - rd(m,n) + 1) \leq T_G(\xi) \tag{4.16}$$

By expressing inequality (4.16) in terms of access duration, simple transformation, and then summing over all variable nodes we obtain:

$$\mathcal{S} \leq |N| \cdot \sum_{m\in M} T_G(\xi, m)$$

[4] This subsection contains definitions, figures and text partially reproduced from the journal publications [19].

Since $G$ is a balanced colored base graph, $T_G(\xi) = \frac{|E|}{n_{banks}}$; hence, we obtain the inequality from (4.15). □

The theoretical constraint for a code that allows successful message bank mapping is presented in the next two Propositions [19]:

A code is said to be $(d_c, d_v)$ regular if $d_c(m) = d_c \; \forall m \in M$, and $d_v(n) = d_v \forall n \in N$, resulting in $d_c \cdot |M| = d_v \cdot |N| = |E|$.

**Proposition 2.** *Given $G(V, E)$, a $(d_c, d_v)$ regular base graph that is perfectly colored with an $n_{banks}$ coloring function $\xi$, having $N_C \mid d_c$, then the following statement is true:*

$$n_{banks} \mid |N| \tag{4.17}$$

The demonstration is trivial. Note that perfect colorability for a $(d_c, d_v)$ regular base graph, $n_{banks} \mid d_c$ colors, implies that the number of colors divides the number of variable nodes ($n_{banks} \nmid |N|$).

**Proposition 3.** *Let $G$ be a graph and $\xi$ an $n_{banks}$ balanced coloring function of $G$, then the following sentences are true:*

$$\forall m \in M, n_{banks} \mid d_c(m) \tag{4.18}$$

$$\forall c_1, c_2 \in \{1 \dots n_{banks}\}, \mathcal{N}_E(c_1) = \mathcal{N}_E(c_2) \tag{4.19}$$

*where $\mathcal{N}_E(c)$ is used to denote the sum of edges colored with an arbitrary color $c$.*

*Proof (Proposition 3):* Consequence of the balanced colored property of $G$, the number of colored variable nodes connected to a check node $m$ is equal for all colors.

$$\forall m \in M, \forall c_1, c_2 \in \{1 \dots n_{banks}\}, N_\xi(m, c_1) = N_\xi(m, c_2) \tag{4.20}$$

This means that the number of colors divides all the check node degrees, and eq. (4.18) is satisfied.

Because edges connected to a variable node are colored with the same color as the variable node itself, by summing eq. (4.20) for all check nodes we obtain eq. (4.19). □

We stress out that the pipeline and coloring constraints are necessary, but, not sufficient. This means that even if the constraints are satisfied it does not guarantee that code construction succeeds. However, they are important in the sense that they clearly show whether it makes sense to even attempt to build a code with a certain set of parameters.

### 4.5.2 AL-PEG

We start by checking the graph properties that are required for perfect colorability and for pipeline execution without RAW hazards: Proposition 3, and Eq. (4.15) and Proposition 1 for pipeline respectively. If any of the two necessary conditions are not satisfied, a different lifting degree $z$ is chosen, or some other graph parameter is changed. According to Proposition 3 the sum of variable node degrees needs to be balanced for each colors such that the graph construction succeeds. In order to achieve this requirement, we modify the PEG graph construction approach by adding a preliminary step that does the coloring such that the condition of Proposition 3 is met. Having variable nodes colored (we emphasize that the edges are in fact colored, but since the color needs to be the same for all edges

connected to a variable node, we use the loose terminology that the variable node is colored). Having different pre-colored set of variable nodes, we proceed to the actual graph construction. The PEG constructs the graph by expanding the tree to build a set of variable-nodes candidates, denoted by $\mathcal{L}_m$. The base graph ($G$) and the exponent matrix ($P$) are built in a single-step expansion. The construction is done in a check-node by check-node manner.

The criterion for selecting the $\mathcal{L}_m$ set candidates is the girth. For the girth filter we use the two-step approach proposed in [91]. AL-PEG adds new filters during the section process such that:

- RAW hazard avoidance between layers, and between successive iterations:

$$wb(m_1, n) < rd(m_2, n) \tag{4.21}$$

$$wb(m_2, n) < \sum_{m \in M} R_G(\xi, m) + rd(m_1, n) \tag{4.22}$$

- Selects all variable nodes that are colored with an available color that has the highest number of free edges. This strategy guarantees that the number of connected edges per color is balanced during construction (*i.e.* $\mathcal{N}_E(\xi(n)) - \widetilde{\mathcal{N}}_E(\xi(n)))$. $\mathcal{L}_m = \{n \in \mathcal{L}_m | Eq.(4.1) holds\ ,\ \mathcal{N}_E(\xi(n)) - \widetilde{\mathcal{N}}_E(\xi(n)) = \max_{n' \in \mathcal{L}_m} (\mathcal{N}_E(\xi(n')) - \widetilde{\mathcal{N}}_E(\xi(n')))\}$
  The $\widetilde{}$ notation distinguishes between the base graph during the construction phase (with $\widetilde{}$), and the input parameters of the base graph (without$\widetilde{}$).

If several candidates exist in the $\mathcal{L}_m$ set, the algorithm uses randomization to make the final choice. The pre-coloring step is depicted in Algorithm 4.

**Algorithm 4** AL-PEG: Graph construction ([19])

```
INPUT: |M|, |N|, z, n_banks, n_latency, pd function
Check pipeline constraint Proposition 1
Check coloring constraint Proposition 3, 2
ξ_best ← {Random initial coloring}          ▷ Determine a variable node coloring
EQUALIZE(1, N, ∞)          ▷ Group the variable nodes in sets corresponding to each color.
Using the a priori colored variable nodes build the graph.
for all m ∈ M do                            ▷ Check nodes are constructed one by one
    while d̃_c(m) < d_c(m) do
        L_m ← select(m)
        n ← random(L_m)                     ▷ selection of n with associated p offset
        connect(m, n, p)
OUTPUT: ξ, φ, P
```

AL-PEG uses a similar approach as suggested in [92]. The base graph is constructed in a check node by check node manner. This approach allows the straightforward inclusion of the pipeline constraint, as noted by [89]. The variable nodes to be connected have been already colored by the EQUALIZE function described shortly.

**EQUALIZE: determine $\xi$**

There are some cases of input variable node degrees when a balanced coloring function cannot be found. Then, the absolute deviation of the number of edges per color, denoted by $\delta(\xi, c)$, is minimized:

$$\delta(\xi, c) = |\mathcal{N}_E(c) - \mathcal{N}_E^{avg}| \tag{4.23}$$

The notation $\mathcal{N}_E^{avg}$ stands for the average value of $\mathcal{N}_E(c)$, $\forall c \in \{1 \dots n_{banks}\}$. We use two measures to characterize the global deviation of the number of edges per color: the maximum absolute deviation denoted with $\delta_{max}(\xi)$, and the mean absolute deviation denoted with $MAD(\xi)$. The first is used to measure the distance from the balanced coloring solution, and it directly influences in the number of stalls required to balance message bank read accesses. The second metric is needed since the total number of stalls is computed for all colors, not only for the one corresponding to the maximum deviation. Note that two different coloring solutions, having the same $\delta_{max}(\xi)$ and different $MAD(\xi)$ may yield different iteration durations. Both $\delta_{max}(\xi)$ and $MAD(\xi)$ need to be minimized during the coloring function selection process. Our findings suggest that even if balanced coloring is not achieved, codes with high HUE can be obtained if these measures are made relatively small.

**Algorithm 5** AL-PEG pre-coloring step: Select $\xi$, as in [19]

```
1: function EQUALIZE(c, N̄, Δ_max)
2:     found ← false
3:     if c < n_banks then                                  ▷ the first n_banks − 1 colors
4:         Δ ← 0
5:         while Δ ≤ Δ_max and not found do
6:             KS = select_sets(N̄, N_E^avg − Δ) ∪ select_sets(N̄, N_E^avg + Δ)
7:             for all N_s ∈ KS do
8:                 color(c, N_s, ξ)
9:                 if EQUALIZE(c + 1, N̄ \ N_s, Δ) then
10:                    found ← true
11:            Δ ← Δ + 1
12:    else                                                 ▷ The last color is reached
13:        color(c, N̄, ξ)
14:        if δ(ξ, c) ≤ Δ_max then
15:            if MAD(ξ) < MAD(ξ_best) then
16:                ξ_best ← ξ                               ▷ Best solution was found
17:                found ← true                             ▷ Exit from the function
18:    return found
19: ξ_best ← {Random initial coloring}                      ▷ The program start
20: EQUALIZE(1, N, ∞)
```

After the variable nodes are colored by EQUALIZE, the actual construction begins (lines 7-11 from Algo. 4). the most representative functions used during construction are: select and connect. $connect(m, n)$ connects check node $m$ with variable node $n$ through an edge. Furthermore it sets the $P_{n,m}$ such that the girth is maximal. From the $\psi$ function point of view, the read access is also set. It also means that the read access ( $rd(m, n)$) time and write-back access ($wb(m, n)$) time for $n$ corresponding to the check-node $m$ processing is set.

$select(m)$ This function is responsible to find the candidate variable node $n$ to be connected to the check-node $m$. In addition to this, it also performs the task of finding the value of $p$ of the exponent matrix $P$, which tries to maximize the girth of the expanded graph of $H$. This is PEG default filtered referred to in this work as – Girth filter. Furthermore, it is the responsibility of the *select* function to determine if there are variable nodes with free degree equal to the number of *available* check nodes. These, variable nodes are unavoidable connections in the output graph $G$.

Next, we narrow the set of candidate nodes $\mathcal{L}_m$ by applying the following filters by priority order:

- *Last connection filter:* Manages the unavoidable connections corresponding to the remaining unconnected variable nodes with the free degree of $n$ equal to the number of remaining unconnected check-nodes. Thus, all the variable nodes $n$ with $d_v(n) - \widetilde{d}_v(n)$ equal to the number of check nodes with at least one missing connection are connected to check node $m$. Since these are unavoidable connections the $p$ value is set random. Afterwards, the pipeline and coloring constraints are verified. If they are satisfied than the algorithm continues, otherwise it fails.

- *Pipeline filter:*enforces that no pipeline RAW hazard appears. The read and the write-back clock cycles of the previous check nodes have been fixed by previous calls to the *connect* function. $\mathcal{L}_m = \{n \in \mathcal{L}_m | Eqs.(4.21), (4.22)\ hold\ for\ \psi(m,n) = \left|\widetilde{\mathcal{N}}_E(\xi(n))\right| + 1\}$

- *Color filter:* besides the legal colors available for check-node $m$, we also need to avoid color overuse. The strategy ensured that the number of connected edges per color is balanced during construction $n$, by selecting all variable nodes having $\mathcal{N}_E(\xi(n)) - \widetilde{\mathcal{N}}_E(\xi(n))$ maximal.

$$\mathcal{L}_m = \{n \in \mathcal{L}_m | Eq.(4.1) holds\ ,\ \mathcal{N}_E(\xi(n)) - \widetilde{\mathcal{N}}_E(\xi(n)) = \max_{n' \in \mathcal{L}_m} (\mathcal{N}_E(\xi(n')) - \widetilde{\mathcal{N}}_E(\xi(n')))\}$$

- *Girth filter:* reduces $\mathcal{L}_m$ to the variable node candidates that would introduce the maximum girth into the Tanner graph (cyclic $z$-lifting). The initial value of $p$ is random. We use the two-step approach proposed by [91]. During this step, the offset value of $p$ is set. The Tanner graph cycle length is computed as suggested in [47].

- *Largest free degree filter:* reduces $\mathcal{L}_m$ to variable nodes with largest free degree $(d_v(n) - \widetilde{d}_v(n))$ – similar to [98].

The randomization points allow the selection of the higher girth and HUE value solutions from by multiple runs of the graph construction – lines 7-11 of Algo. 4. From the algorithm complexity point of view, EQUALIZE is a simple Breath-First Search in the base graph $G(V, E)$, having complexity $O(|V| + |E|)$, where $E$ is the set of edges, and $V$ is the set of nodes of the base graph $G$. For the graph construction algorithm, the most complex operation is the girth detection in the Tanner graph. It can be approximate by $O(|E| \cdot g \cdot z^2)$, by taking into account that in most cases $z^2 \gg (|V| + |E|$, and girth values seldom exceed 12.

**Results and Discussion**

In [19] we have pursued a complete analysis starting from code construction, continuing with Monte-Carlo simulations to extract FER/BER statistics for decoding performance, and implementation results as well. Here, we only report HUE and girth metrics, due to space limitations.We compare the code generation results to the case of standard PEG code construction followed by off-line algorithms for memory message bank-mapping, followed by message access scheduling compared (denoted by PEG-QC+CP) against AL-PEG. Table 4.2 also shows the case of PEG-QC+Ad-hoc– meaning conventional PEG-QC code construction and without any off-line algorithms. Instead, the natural order

message mapping followed by natural order message access scheduling are employed. For all tree cases 500 codes have been generated, and the one with the highest girth, and lowest girth multiplicity is selected. In order to make this analysis meaningful, we have selected a wide range of input parameters corresponding to both regular and $d_v$, $d_c$ irregular codes of different code lengths as follows:

Irregular codes input code parameters are using the degree distribution polynomials for variable ($\lambda$) and check-node ($\rho$) degrees of the WiMAX standard [7] codes having the corresponding code rates:

- Irregular code rate $\frac{1}{2}$ ($\rho(x) = 0.333333 \cdot x^6 + 0.666667 \cdot x^5$, $\lambda(x) = 0.208333 \cdot x^5 + 0.333333 \cdot x^2 + 0.458333 \cdot x^1$)
- Irregular code rate $\frac{2}{3}$ ($\rho(x) = 1 \cdot x^9$, $\lambda(x) = 0.208333 \cdot x^5 + 0.5 \cdot x^2 + 0.291667 \cdot x^1$),
- Irregular code rate $\frac{3}{4}$ ($\rho(x) = 0.166667 \cdot x^{14} + 0.833333 \cdot x^{13}$, $\lambda(x) = 0.75 \cdot x^3 + 0.0416667 \cdot x^2 + 0.208333 \cdot x^1$),
- Irregular code rate $\frac{5}{6}$ ($\rho(x) = 1 \cdot x^{19}$, $\lambda(x) = 0.458333 \cdot x^3 + 0.416667 \cdot x^2 + 0.125 \cdot x^1$),

The regular code, and the very high rate code:

- Regular code rate $\frac{1}{2}$ ($d_v = 3$, $d_c = 6$)
- Irregular code rate $\frac{15}{17}$ ($\rho(x) = 0.125 \cdot x^{37} + 0.125 \cdot x^{34} + 0.25 \cdot x^{33} + 0.375 \cdot x^{32} + 0.125 \cdot x^{31}$, $\lambda(x) = 1 \cdot x^3$)

As depicted in 4.2, the AL-PEG algorithm achieves the maximum theoretical HUE for all codes for the case $N_C = d_c^{max}$, and $N_P = 2$. Note that only $d_c$ regular codes can reach 100% HUE. The standard PEG-QC with CP optimizations HUE is the range $79\% - 50\%$ is CP methods are used, while the standard PEG-QC code construction without any optimizations and only relying on natural order is in the range $44\% - 21\%$ otherwise. It becomes apparent that for large values of $n_{banks}$ efficient HUE can only be achieved through dedicated code design.

## 4.6 Conclusions

In this chapter we have discussed the issue of memory design, as well as other related aspects message scheduling and memory access conflicts of QC-LDPC decoder architectures using layered scheduling. The discussion cover two distinct mitigation approaches. The first has addressed the case of finding an optimum message memory mapping and message access scheduling given a fixed LDPC code and set of architecture parameters. The target is to optimize the HUE metric and maximize throughput. Second, we have discussed the alternative approach, when the choice of code is not fixed, and we can do architecture-aware code design for a set of code and architecture parameters, as well as a set of architecture related assumptions.

Table 4.2: Girth and HUE results for 500 code construction runs, as in [19].

| Code parameters | Method | Girth | Girth multiplicity | HUE[%] |
|---|---|---|---|---|
| Regular rate $\frac{1}{2}$ code, length 1296, $z = 54$, $d_v = 3$, $d_c = 6$, $n_{banks} = 6$, $n_{latency} = 2$ | AL-PEG | 10 | 8694 | 100 |
| | PEG-QC + CP | 10 | 8856 | 55 |
| | PEG-QC + Ad-Hoc | 10 | 8856 | 28 |
| Irregular rate $\frac{15}{17}$ code, length 9520, $z = 70$, $n_{banks} = 38$, $n_{latency} = 2$ | AL-PEG | 6 | 122080 | 89 |
| | PEG-QC + CP | 6 | 126910 | 42 |
| | PEG-QC + Ad-Hoc | 6 | 126910 | 17 |
| Irregular rate $\frac{1}{2}$ code, length 2304, $z = 32$, $n_{banks} = 4$, $n_{latency} = 2$ | AL-PEG | 8 | 4416 | 79 |
| | PEG-QC + CP | 8 | 3072 | 79 |
| | PEG-QC + Ad-Hoc | 8 | 3072 | 44 |
| Irregular rate $\frac{2}{3}$ code, length 2304, $z = 32$, $n_{banks} = 10$, $n_{latency} = 2$ | AL-PEG | 8 | 56304 | 100 |
| | PEG-QC + CP | 8 | 57472 | 57 |
| | PEG-QC + Ad-Hoc | 8 | 57472 | 24 |
| Irregular rate $\frac{3}{4}$ code, length 2304, $z = 32$, $n_{banks} = 15$, $n_{latency} = 2$ | AL-PEG | 6 | 2304 | 94 |
| | PEG-QC + CP | 6 | 672 | 50 |
| | PEG-QC + Ad-Hoc | 6 | 672 | 21 |
| Irregular rate $\frac{5}{6}$ code, length 2304, $z = 32$, $n_{banks} = 20$, $n_{latency} = 3$ | AL-PEG | 6 | 11008 | 100 |
| | PEG-QC + CP | 6 | 11584 | 48 |
| | PEG-QC + Ad-Hoc | 6 | 11584 | 22 |
| Irregular rate $\frac{5}{6}$ code, length 2304, $z = 32$, $n_{banks} = 5$, $n_{latency} = 2$ | AL-PEG | 6 | 10848 | 100 |
| | PEG-QC + CP | 6 | 11520 | 94 |
| | PEG-QC + Ad-Hoc | 6 | 11520 | 49 |
| Irregular rate $\frac{5}{6}$ code, length 2304, $z = 32$, $n_{banks} = 5$, $n_{latency} = 3$ | AL-PEG | 6 | 10496 | 100 |
| | PEG-QC + CP | 6 | 11520 | 92 |
| | PEG-QC + Ad-Hoc | 6 | 11520 | 44 |
| Irregular rate $\frac{5}{6}$ code, length 2304, $z = 32$, $n_{banks} = 5$, $n_{latency} = 4$ | AL-PEG | 6 | 11264 | 94 |
| | PEG-QC + CP | 6 | 11520 | 83 |
| | PEG-QC + Ad-Hoc | 6 | 11520 | 40 |

# Chapter 5

# General Conclusion and Next Steps

In this Chapter, I will describe some future research and teaching directions based on the achievements so far and the goals I hope to accomplish in short and medium term. From the teaching perspective my primary goals are to improve the content of the study materials and to explore different ways to motivate students to get involved and accomplish more interesting tasks in area related to digital design, hardware verification, and communication systems. For the research part, I plan to extend my field of knowledge to control engineering aspects, as well as information theory. On the implementation side, I plan to continue my pursue towards optimized, design automation driven, reusable design for FPGA technology. All techniques that support this goal are future research interests. I have started my PhD more than 10 years ago with the hope of joining an environment that enables me to continuously grow as a professional. I found dealing with problems and learning of state-of-the art achievements very motivating and fulfilling. The thing I found most exciting about this role is the freedom in terms of technical work goals and directions. By means of this thesis I hope to be able to gain an extra degree of freedom to do my work, and the possibility to help guide and support some other's people dreams that share common aspirations such as mine. While these sound nice in principle, I am aware from future experience, that success is a multi-objective function of human resources involved (*i.e.* students, and collaborators), material resources (*i.e.* research funding, equipment, etc.), momentum (*i.e.* right time window for a collaborations), and some degree of luck, such that the optimum results are achieved (*i.e.* high quality engineering and research results and people satisfaction). Although, the road gets a bit bumpy, I strongly believe that with the right degree of perseverance, and enough flexibility, and by attempting to make things better with each increment, in the end we can succeed in reaching our goals.

During the last 8 years I have been fortunate to collaborate with many different researchers that influenced both at the personal level and especially the professional side. I believe that for further growing as a person and researcher I must continue to invest effort in meeting and collaborating with new people from other research groups, as well as to increase my mobility by means of research visits. In this way I can expand my horizons.

## 5.1 Challenges and Future Research Directions

### 5.1.1 Research and Teaching at UPT

When I have started my first teaching activity I was very enthusiastic about creating new teaching materials for students. Those materials have been designed from scratch. Students were suppose to follow through, and provide answers for a fixed number of assignments. Now, I have a different view. I think that there are a lot of interesting materials out there, I try as much as possible to avoid "re-inventing the wheel", and try to incorporate as much as already there knowledge as possible in the "red thread" that I as a teacher provide for a course. Furthermore, I try to be flexible, and offer them choices as much as possible, for the assignments they wish to complete. This, I hope will enable them to be more motivated in ticking the knowledge boxes" of knowhow.

For the application part, the concrete actions include:

- different material choices for the same lab assignment;
- bonus points by elective assignments – 2-3 students group projects;

For the exam part:

- short assignments and quizzes during the course period, that help maintain the interest of student during a lecture time, and emphasize the most important concepts discussed during the lecture;
- bonus points by elective assignments;

For the examination side – the possibility of choosing between different examination subjects (*e.g.* 3 out of 4 questions need to be solved from the examination sheet).

Another important aspect is content. With respect to this, during the last 10 years at UPT, I have contributed to introducing Register Transfer Level (RTL) methodology, as well as FPGA lab materials or the 1st year student for the Digital Logics course, as well as an elective new course – Hardware Verification and Validation for the 4th year undergraduate students teaching System Verilog, System Verilog Assertions and Universal Verification Methodology (UVM). For the wireless communication elective course, for 4th year undergraduate students, I have introduced OMNET++ lab activities for studying algorithms and standards related to wireless communication. I will continue to try and add new exciting technologies, as well as try to include as much flexibility and choice as possible during the teaching process. The challenge with respect to teaching is the ever-increasing number of students enrolled for a course, which makes the overall task tiring and lowers quality despite best efforts. In addition to this, research, writing grant proposals, and papers is also a must. Teaching and research activities are sometimes conflicting goals, since both contend for the same time resource budget.

### 5.1.2 Research Directions

Future research directions will be related to the development of novel techniques for fault tolerant digital processing, storage and communication that work on un-reliable components. This direction will combine the research developed in the two international projects that I have led:

- Bilateral Romanian-France UEFISCDI-ANR project DIAMOND - this project researched the implementation of fault tolerant mechanisms based on LDPC error correction codes for digital communication and storage

- ESA Innovation Triangle Initiative project REDOUBT - this project aimed at developing novel fault tolerant techniques for processing data-path using control theory

For the forward error correction LDPC decoders, several directions of research span. First on the hardware acceleration side, given la long simulation time that is needed for validating performance in the error floor, at very low bit error rates, it would be highly beneficial to have those carried on an FPGA platform. The design of such a system needs to be carefully considered such that the gains in accelerating the decoder architecture are not lost due to moving large blocks of data between memory and the decoder unit. This data corresponds to the encoded random codewords and reference. Furthermore, as stressed out in [71], certain data patters are more interesting to exercise. An efficient algorithm capable of extracting them, as well as an efficient way to integrate it in the the hardware-software solution needs to be considered. To sum up, a perspective future direction consists in building a co-design solution that accelerates analysis of decoding performance of QC-LDPC decoders comprising of a set of algorithms that help identify and analyze the input data patters that make suitable candidates for error floor performance characterization by using the information of the decoder cycle structures where errors become "trapped", and the decoder stuck, and a decoder FPGA infrastructure for collecting state information and running a large number of codeword decodings.

A second direction is related to harnessing the effects of "noisy" behavior during decoding by finding efficient implementation methods such as the ones proposed in [59]. The main idea is to find low-cost solutions for emulating noise. As argued in the work of [99] LDPC decoding actually benefits from executing on noisy hardware, and exhibits improved decoding performance. The sources of this noise, as shown in [59] can be imprecise implementations (in many cases reduced complexity implementations of the baseline operations) that yield lower cost than the actual baselines. So far, most of the effort has been focused on bit-flip decoders relying on flooding scheduling decoding. A future research direction would be to try and harness this effect and approach for soft message decoding algorithms such as MS. The target codes are QC-LDPC, and for this purpose we plan to make use of the accumulated know-how, and existing decoder architectures for both flooding and layered scheduling decoding.

The ESA Innovation Triangle Initiative project REDOUBT - opens a novel research direction with many possible future developments on the topic of novel fault tolerant techniques for processing data-path using control theory. For this research direction both theoretical and digital circuit design research opportunities are abundant since the approach is entirely novel.

To sum up, there are a lot of exciting things that can be investigated. In the end, the factor deciding the actual course of action will be funding and the human resource available for materializing these ideas. Only with proper support, and continuous effort in managing future collaborations, and in attracting new resources, both financial and people, the future steps can be realized. This is why, the activity of striving to obtain this resources will be prioritized during the period to come. Different from our course of action so far, we will try to have a greater interaction with industry, and try to improve collaboration in this direction for both the research and the teaching part.

# Bibliography

[1] O. Boncalo, "Qc-ldpc gear-like decoder architecture with multi-domain quantization," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 244–251.

[2] O. Boncalo, A. Amaricai, A. Hera, and V. Savin, "Cost-efficient fpga layered ldpc decoder with serial ap-llr processing," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014, pp. 1–6.

[3] O. Boncalo, A. Amaricai, P. F. Mihancea, and V. Savin, "Memory trade-offs in layered self-corrected min-sum ldpc decoders," *Analog Integrated Circuits and Signal Processing*, vol. 87, no. 2, pp. 169–180, May 2016. [Online]. Available: https://doi.org/10.1007/s10470-015-0639-3

[4] O. Boncalo and A. Amaricai, "Ultra high throughput unrolled layered architecture for qc-ldpc decoders," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, vol. 00, July 2017, pp. 225–230. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ISVLSI.2017.47

[5] O. Boncalo, P. F. Mihancea, and A. Amaricai, "Template-based qc-ldpc decoder architecture generation," in *2015 10th International Conference on Information, Communications and Signal Processing (ICICS)*, Dec 2015, pp. 1–5.

[6] O. Boncalo, A. Amaricai, and V. Savin, "Memory efficient implementation of self-corrected min-sum ldpc decoder," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2014, pp. 295–298.

[7] "Air interface for fixed and mobile broadband wireless access systems physical and medium access control layers," *IEEE Std 802.16e-2005*, 2005.

[8] 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, March 2018.

[9] O. Boncalo, V. Savin, and A. Amaricai, "Unrolled layered architectures for non-surjective finite alphabet iterative decoders," in *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, Oct 2017, pp. 1–5.

[10] "Wireless medium access control (MAC) and physi- cal layer (PHY) specifications for high rate wireless personal area networks (WPANs) amendment 2: Millimeter-wave-based alternative physical layer extension," *IEEE 802.15.3c-2009*, available online:. [Online]. Available: http://standards.ieee.org/getieee802/download/802.15.3c-2009.pdf

[11] F.-R. project, "Report on reliability aware synthesis and ldpc decoders built with unreliable components," no. 6.1, 2015, available online:. [Online]. Available: www.i-risc.eu

[12] A. Hera, O. Boncalo, C. Gavriliu, A. Amaricai, V. Savin, D. Declercq, and F. Ghaffari, "Analysis and implementation of on-the-fly stopping criteria for layered qc ldpc decoders," in *2015 22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES)*, June 2015, pp. 287–291.

[13] D. Declercq, V. Savin, O. Boncalo, and F. Ghaffari, "An imprecise stopping criterion based on in-between layers partial syndromes," *IEEE Communications Letters*, vol. 22, no. 1, pp. 13–16, Jan 2018.

[14] O. Boncalo, A. Amaricai, V. Savin, D. Declercq, and F. Ghaffari, "Check node unit for ldpc decoders based on one-hot data representation of messages," *Electronics Letters*, vol. 51, no. 12, pp. 907–908, 2015.

[15] O. Boncalo, G. Kolumban-Antal, A. Amaricai, V. Savin, and D. Declercq, "Layered ldpc decoders with efficient memory access scheduling and mapping and built-in support for pipeline hazards mitigation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2018.

[16] T. T. Nguyen-Ly, K. Le, V. Savin, D. Declercq, F. Ghaffari, and O. Boncalo, "Non-surjective finite alphabet iterative decoders," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[17] Y. S. Park, D. Blaauw, D. Sylvester, and Z. Zhang, "Low-power high-throughput ldpc decoder using non-refresh embedded dram," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 3, pp. 783–794, March 2014.

[18] O. Boncalo, A. Amaricai, and S. Nimara, "Memory-centric flooded ldpc decoder architecture using non-surjective finite alphabet iterative decoding," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, Aug 2018, pp. 104–109.

[19] O. Boncalo, G. Kolumban-Antal, D. Declercq, and V. Savin, "Code-design for efficient pipelined layered ldpc decoders with bank memory organization," *Microprocessors and Microsystems*, vol. 63, pp. 216 – 225, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933118300863

[20] "Wireless LAN medium access control (MAC) and physical layer (PHY) specification," *IEEE Std. 802.11*, 1997.

[21] "DVB-the family of international standards for Digital Video Broadcasting, second generation framin structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering, and other broadband satelite applications, Part 2: DVB-S2 extensions (DVB-S2X)," Oct 2014.

[22] K. C. Ho, C. L. Chen, and H. C. Chang, "A 520k (18900, 17010) Array Dispersion LDPC Decoder Architectures for NAND Flash Memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 4, pp. 1293–1304, April 2016.

[23] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Lin, "Flexible ldpc decoder design for multigigabit-per-second applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 1, pp. 116–124, Jan 2010.

[24] S. Usman, M. M. Mansour, and A. Chehab, "Interlaced column-row message-passing schedule for decoding ldpc codes," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

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

[26] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A bit-serial approximate min-sum ldpc decoder and fpga implementation," in *2006 IEEE International Symposium on Circuits and Systems*, May 2006, pp. 4 pp.–.

[27] T. Nguyen-Ly, K. Le, F. Ghaffari, A. Amaricai, O. Boncalo, V. Savin, and D. Declercq, "Fpga design of high throughput ldpc decoder based on imprecise offset min-sum decoding," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, June 2015, pp. 1–4.

[28] S. Abu-Surra, E. Pisek, T. Henige, and S. Rajagopal, "Low-power dual quantization-domain decoding for ldpc codes," in *2014 IEEE Global Communications Conference*, Dec 2014, pp. 3151–3156.

[29] T. T. Nguyen-Ly, V. Savin, X. Popon, and D. Declercq, "High throughput fpga implementation for regular non-surjective finite alphabet iterative decoders," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 961–966.

[30] M. Ardakani and F. R. Kschischang, "Gear-shift decoding," *IEEE Transactions on Communications*, vol. 54, no. 6, pp. 1143–1143, June 2006.

[31] B. Xiang, R. Shen, A. Pan, D. Bao, and X. Zeng, "An area-efficient and low-power multirate decoder for quasi-cyclic low-density parity-check codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 10, pp. 1447–1460, Oct 2010.

[32] D. Oh and K. K. Parhi, "Min-sum decoder architectures with reduced word length for ldpc codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 1, pp. 105–115, Jan 2010.

[33] 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)*, Aug 2016, pp. 230–237.

[34] X. Zhang and P. H. Siegel, "Quantized min-sum decoders with low error floor for ldpc codes," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, July 2012, pp. 2871–2875.

[35] Y. L. Ueng, Y. L. Wang, L. S. Kan, C. J. Yang, and Y. H. Su, "Jointly designed architecture-aware ldpc convolutional codes and memory-based shuffled decoder architecture," *IEEE Transactions on Signal Processing*, vol. 60, no. 8, pp. 4387–4402, Aug 2012.

[36] E. Dupraz, F. Leduc-Primeau, and F. Gagnon, "Low-latency ldpc decoding achieved by code and architecture co-design," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, Dec 2018, pp. 1–5.

[37] Z. Wu and K. Su, "Updating conflict solution for pipelined layered ldpc decoder," in *Signal Processing, Communications and Computing (ICSPCC), 2015 IEEE International Conference on*, Sept 2015, pp. 1–6.

[38] 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*, Oct 2009, pp. 220–225.

[39] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, "A scalable decoder architecture for ieee 802.11n ldpc codes," in *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, Nov 2007, pp. 3270–3274.

[40] Z. Wu, D. Liu, and Y. Zhang, "Matrix reordering techniques for memory conflict reduction for pipelined qc-ldpc decoder," in *2014 IEEE/CIC International Conference on Communications in China (ICCC)*, Oct 2014, pp. 354–359.

[41] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and ldpc decoder architectures," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2002–2009, Sept 2004.

[42] E. Amador, R. Pacalet, and V. Rezard, "Optimum ldpc decoder: A memory architecture problem," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, July 2009, pp. 891–896.

[43] X. Zhao, Z. Chen, X. Peng, D. Zhou, and S. Goto, "Dvb-t2 ldpc decoder with perfect conflict resolution," in *Proceedings of Technical Program of 2012 VLSI Design, Automation and Test*, April 2012, pp. 1–4.

[44] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[45] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[46] T. Richardson and R. Urbanke, "The renaissance of gallager's low-density parity-check codes," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 126–131, Aug 2003.

[47] M. P. C. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, Aug 2004.

[48] R. G. Gallager, "Low density parity check codes," MIT Press, Cambridge, 1963, research Monograph series.

[49] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, May 1999.

[50] V. Savin, "Self-corrected min-sum decoding of ldpc codes," in *2008 IEEE International Symposium on Information Theory*, July 2008, pp. 146–150.

[51] J. Chen and M. P. Fossorier, "Near optimum universal belief propagation based decoding of low density parity check codes," *IEEE Trans. on Communications*, vol. 50, no. 3, pp. 406–414, 2002.

[52] E. Eleftheriou, "Reduced-complexity decoding algorithm for low-density parity-check codes," *Electronics Letters*, vol. 37, pp. 102–104(2), January 2001. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/el_20010077

[53] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of ldpc codes," in *IEEE Workshop onSignal Processing Systems, 2004. SIPS 2004.*, Oct 2004, pp. 107–112.

[54] J. Jin and C. y. Tsui, "An energy efficient layered decoding architecture for LDPC decoder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 8, pp. 1185–1195, Aug 2010.

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

[56] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, "A minimum-latency block-serial architecture of a decoder for IEEE 802.11n LDPC codes," in *2007 IFIP International Conference on Very Large Scale Integration*, Oct 2007, pp. 236–241.

[57] C. Condo, A. Baghdadi, and G. Masera, "Reducing the Dissipated Energy in Multi-standard Turbo and LDPC Decoders," *Circuits, Systems, and Signal Processing*, vol. 34, no. 5, pp. 1571–1593, May 2015.

[58] Y. Sun, G. Wang, and J. R. Cavallaro, "Multi-layer parallel decoding algorithm and vlsi architecture for quasi-cyclic ldpc codes," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, May 2011, pp. 1776–1779.

[59] K. Le, D. Declercq, F. Ghaffari, L. Kessal, O. Boncalo, and V. Savin, "Variable-node-shift based architecture for probabilistic gradient descent bit flipping on qc-ldpc codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 7, pp. 2183–2195, July 2018.

[60] A. Balatsoukas-Stimming, M. Meidlinger, R. Ghanaatian, G. Matz, and A. Burg, "A fully-unrolled ldpc decoder based on quantized message passing," in *2015 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2015, pp. 1–6.

[61] "Virtex7 data sheet," 2016. [Online]. Available: www.xilinx.com

[62] J. Demma and P. Athanas, "A hardware generator for factor graph applications," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, Dec 2014, pp. 1–8.

[63] G. Falcao, M. Gomes, J. Goncalves, P. Faia, and V. Silva, "Hdl library of processing units for an automatic ldpc decoder design," in *2006 Ph.D. Research in Microelectronics and Electronics*, June 2006, pp. 349–352.

[64] Y. Delomier, B. Le Gal, J. Crenne, and C. Jego, "Model-based design of efficient ldpc decoder architectures," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, Dec 2018, pp. 1–5.

[65] D. M. Pham and S. M. Aziz, "An automated design methodology for fpga-based multi-gbps ldpc decoders," in *2012 15th International Conference on Computer and Information Technology (ICCIT)*, Dec 2012, pp. 495–499.

[66] Oracle, "Java server pages technology," 2015, available online:. [Online]. Available: www.oracle.com/technetwork/java/javaee/jsp/index.html

[67] Microsoft, "Asp technology feature overview," 2015, available online:. [Online]. Available: msdn.microsoft.com/en-us/library/ms972202.aspx

[68] G. Spivey, "Ep3: An extensible perl preprocessor," in *Proceedings International Verilog HDL Conference and VHDL International Users Forum*, March 1998, pp. 106–113.

[69] O. Shacham, S. Galal, S. Sankaranarayanan, M. Wachs, J. Brunhaver, A. Vassiliev, M. Horowitz, A. Danowitz, W. Qadeer, and S. Richardson, "Avoiding game over: Bringing design to the next level," in *DAC Design Automation Conference 2012*, June 2012, pp. 623–629.

[70] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, June 2012, pp. 1212–1221.

[71] S. K. Planjery, S. K. Chilappagari, B. Vasić, D. Declercq, and L. Danjean, "Iterative decoding beyond belief propagation," in *2010 Information Theory and Applications Workshop (ITA)*, Jan 2010, pp. 1–10.

[72] V. Savin, O. Boncalo, and D. Declercq, "Stopping criterion for decoding quasi-cyclic ldpc codes," *European Patent Office EP3373488A1*.

[73] J. Li, G. He, H. Hou, Z. Zhang, and J. Ma, "Memory efficient layered decoder design with early termination for ldpc codes," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, May 2011, pp. 2697–2700.

[74] E. Amador, R. Knopp, R. Pacalet, and V. Rezard, "On-the-fly syndrome check for ldpc decoders," in *2010 6th International Conference on Wireless and Mobile Communications*, Sep. 2010, pp. 33–37.

[75] C. . Lin, T. . Huang, C. . Chen, and S. . Lin, "Efficient layer stopping technique for layered ldpc decoding," *Electronics Letters*, vol. 49, no. 16, pp. 994–996, Aug 2013.

[76] Sun and J. R. Cavallaro, "A low-power 1-gbps reconfigurable ldpc decoder design for multiple 4g wireless standards," in *2008 IEEE International SOC Conference*, Sep. 2008, pp. 367–370.

[77] M. Ferrari, S. Bellini, and A. Tomasoni, "Safe early stopping for layered ldpc decoding," *IEEE Communications Letters*, vol. 19, no. 3, pp. 315–318, March 2015.

[78] G. Liva, S. Song, L. Lan, Y. Zhang, S. Lin, and W. E. Ryan, "Design of ldpc codes: A survey and new results," *Journal of Communications Software and Systems*, vol. 2, no. 3, pp. 191–211, 2017.

[79] "Copyright," in *Academic Press Library in Mobile and Wireless Communications*, D. Declerq, M. Fossorier, and E. Biglieri, Eds. Oxford: Academic Press, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123964991000169

[80] A. Venkiah, D. Declercq, and C. Poulliat, "Design of cages with a randomized progressive edge-growth algorithm," *IEEE Communications Letters*, vol. 12, no. 4, pp. 301–303, April 2008.

[81] O. A. Rasheed, P. Ivaniš, and B. Vasić, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, Sep. 2014.

[82] K. Le, D. Declercq, F. Ghaffari, C. Spagnol, E. Popovici, P. Ivanis, and B. Vasic, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1494–1497.

[83] Z. Wang and Z. Cui, "A memory efficient partially parallel decoder architecture for quasi-cyclic LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 483–488, April 2007.

[84] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory System Optimization for FPGA-Based Implementation of Quasi-Cyclic LDPC Codes Decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 1, pp. 98–111, Jan 2011.

[85] S. Nimara, O. Boncalo, A. Amaricai, and M. Popa, "Fpga architecture of multi-codeword LDPC decoder with efficient BRAM utilization," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2016, pp. 1–4.

[86] A. Briki, C. Chavet, and P. Coussy, "A conflict-free memory mapping approach to design parallel hardware interleaver architectures with optimized network and controller," in *SiPS 2013 Proceedings*, Oct 2013, pp. 201–206.

[87] S. U. Reehman, C. Chavet, P. Coussy, and A. Sani, "In-place memory mapping approach for optimized parallel hardware interleaver architectures," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 896–899.

[88] A. Briki, C. Chavet, P. Coussy, and E. Martin, "A design approach dedicated to network-based and conflict-free parallel interleavers," in *Proceedings of the Great Lakes Symposium on VLSI*, ser. GLSVLSI '12. New York, NY, USA: ACM, 2012, pp. 153–158. [Online]. Available: http://doi.acm.org/10.1145/2206781.2206819

[89] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth tanner graphs," in *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, vol. 2, 2001, pp. 995–1001 vol.2.

[90] Z. Li and B. V. K. V. Kumar, "A class of good quasi-cyclic low-density parity check codes based on progressive edge growth graph," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 2, Nov 2004, pp. 1990–1994 Vol.2.

[91] X. Q. Jiang, M. H. Lee, H. M. Wang, J. Li, and M. Wen, "Modified peg algorithm for large girth quasi-cyclic protograph ldpc codes," in *2016 International Conference on Computing, Networking and Communications (ICNC)*, Feb 2016, pp. 1–5.

[92] J. Zhang, M. Dong, and Y. Jin, "A qc-ldpc construction algorithm for increasing the throughput of layered decoders," in *Communication Technology (ICCT), 2013 15th IEEE International Conference on*, Nov 2013, pp. 604–608.

[93] Y. Zhu and C. Chakrabarti, "Architecture-aware ldpc code design for multiprocessor software defined radio systems," *IEEE Transactions on Signal Processing*, vol. 57, no. 9, pp. 3679–3692, Sept 2009.

[94] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," *Oper. Res.*, vol. 11, no. 6, pp. 972–989, Dec. 1963. [Online]. Available: http://dx.doi.org/10.1287/opre.11.6.972

[95] G. Ucoluk, "Genetic algorithm solution of the TSP avoiding special crossover and mutation," *Intelligent Automation & Soft Computing*, vol. 8, no. 3, pp. 265–272, 2002.

[96] S. Yeşil and M. Arslan, "Dual port ram based layered decoding for Multi Rate Quasi-Cyclic LDPC codes," in *2014 12th International Conference on Signal Processing (ICSP)*, Oct 2014, pp. 1524–1530.

[97] V. A. Chandrasetty and S. M. Aziz, "Resource efficient LDPC decoders for multimedia communication," *Integration, the VLSI Journal*, vol. 48, pp. 213 – 220, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926014000613

[98] J. Martinez-Mateo, D. Elkouss, and V. Martin, "Improved construction of irregular progressive edge-growth tanner graphs," *IEEE Communications Letters*, vol. 14, no. 12, pp. 1155–1157, December 2010.

[99] C. K. Ngassa, V. Savin, and D. Declercq, "Min-sum-based decoders running on noisy hardware," in *2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 1879–1884.