PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DESIGN OF A DVB-S2 COMPLIANT LDPC DECODER FOR FPGA

Tesis para optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Guillermo Daniel Montaño Gamarra

ASESOR: MSc. Ing. Mario Andrés Raffo Jara

Lima, 2021

To my parents, Consuelo and Guillermo, my sister Sara,
my advisor Mario, the members of GuE PUCP and my best friends in life

”If I have seen further than others,
it is by standing upon the shoulders of giants.”

Issac Newton

Abstract

Low Density Parity Check codes presents itself as the dominant FEC code in terms of performance, having the nearest performance to the Shannon limit and proving its usefulness in the increasing range of applications and standards that already used it. Low power devices are not except of this rapid development, where it emerges the necessity of decoders of low power without totally sacrificing performance or resource usage.

The present work details the devolopment of a LDPC decoder compliant with the DVB-S2 standard for digital television, motivated for its already established use in uplink and downlink satellite applications and its great performance at large code lengths. This research presents the study of the min-sum algorithm and the design of the elements that conform the core decoder, including both functional units (variable and check nodes), memory blocks and routing network. In the context of DVB-S2, it focused exclusively in the prototyping of the inner LDPC decoder and targets FPGA as platform.

A variety of design strategies are applied in the design of the core, including the optimal selection of the architecture and the schedule policy, the design of the control unit as a Algorithmic State Machine (ASM) and the inclusion of specialized modules to reduce the number of clock cycles per decoding process, such as early stopping.

The selected features for this work are code length of 64800 bits and code rate equal to 1/2. The selected architecture is partially parallel with flooding schedule and operates over binary symbols (Galois field GF(2)). For testing, it assumes a channel with AWGN and BPSK modulation, so the demodulator feeds soft decision information of each symbol based on both assumptions.

The design has been validated using different verification methodologies according to complexity and predictability of each part or the whole system. Obtained results show the decoder, when configured for a maximum of 10 iterations, has a BER performance of 10^{-3} at a SNR of 2 dB, having an advantage of 1 dB respect to previous published works [1]. It uses 60363 slice LUT and 23552 slice registers when synthesized in the Virtex 7 xc7vx550t FPGA from Xilinx, a reduction of 10% in resource usage from [1]. It achieves a maximum frequency operation of 194 Mhz and a throughput of 142.99 Mbps at worst case. The top energy per bit rate is 18.344 nJ/bit.

Keywords: LDPC, min-sum algorithm, DVB-S2, low power, early stopping, flooding schedule.

Contents

List of Figures

List of Tables

Introduction

The advent of digital communications in modern life introduced the possibility of nullify, or at least minimize, the degrading effect of channel noise by introducing error correction techniques or FEC. With an ever increasing demand for more reliable, efficient and faster information exchange, correction codes are in a endless cycle of improvement. The latest established FEC is the Low Density Parity Check Codes (LDPC), recognized for its robustness and already in use in telecommunication standards.

The present work is focused in the design of a LDPC decoder compliant with the DVB-S2 standard, taking into account considerations of resource usage and power dissipation. This decoder is intended for a Field-Programmable Gate Array (FPGA) as a prototype for a future implementation in an Application-Specific Integrated Circuit (ASIC).

This document is structured in six parts: four chapters, conclusions and recommendations. The first chapter presents the state of art which serve as metric for the proposed design and the objectives that guide the development of it. The second chapter explores the theoretical framework required for the understanding of the focused application, covering two different fronts: the fundamentals of LDPC, belonging to information theory scope, and the hardware design applied to such kind of implementations, reviewing types of architecture and physical considerations (timing and area). This chapter finishes with the presentation of a general scheme of the decoder, based on the previous presented knowledge. The third chapter is the core of this work and focuses in the design of the decoder core, taking considerations of the previous chapter and indicating how each of the components is structured at a register-transfer level (RTL), to conclude with the description of the control unit, following the Algorithmic State Machine (ASM) approach. The fourth chapter introduces the results of simulations and evaluations executed over the core and compares it with selected works. Finally, some conclusions are provided, with the addition of recommendations for improvement and some guidelines for future works. Attached to this document, there are the appendixes that contain the source files of hardware description and other files used during development.

Chapter 1

Problem formulation

1.1 Motivation

Nowadays, digital communications has established as the core of multiple technologies: mobile telephony and internet, satellite connections and digital television, among others. All of these forms of communications are transmitted via noisy channels, especially present in wireless technologies. This forces the use of error correction codes, also known as Forward Error Correction (FEC) codes, with the purpose of adding redundant information to the message and making its restoration at the receiver possible [2].

One of the most used FEC codes are the Low Density Parity Check (LDPC) codes, a technique proposed in Robert Gallager's dissertation in the 60s [3]. Ignored in his time because of lack of computational capabilities, this type of codes had its resurgence in the mid 90s with the Mackay and Neal's paper [4] and more computing power available. Since then, LDPC codes had been under development until surpassing its immediate competitor, turbo codes, and proving they are the FEC code type with the nearest capacity to Shannon limit (i.e. the maximum theoretical capacity of a communication channel). Moreover, LDPC codes are the codification used in various standards such as 10GBase-T Ethernet, WiMax and NASA Deep Space Communication. Development continues with the objective of satisfying last technologies' requisites, for example, 5G connectivity or Cloud-RAN systems [5]. Satellite communications are another example in which this type of codes is incorporated into established standards, such as European digital television standard, also known as Digital Video Broadcasting - Satellite Generation 2 (DVB-S2) [6].

All this massive connectivity establishes access to information as a primary vehicle for social development. Deprivation of usage of Information and Communications Technologies (ITC)

causes the phenomenon called ”digital divide”, affecting low-income people and truncating its own development [7].

In Peru, digital divide is prevalent in geographically isolated towns, lacking or having a limited access to telecommunication. This situation presents the opportunity of reducing digital divide by satellite links, the same kind of technology whose utilization forces development of codes for error correction. There is a precedent in the Pontificia Universidad Católica del Perú (PUCP), represented by the Rural Telecommunications Group (GTR-PUCP by its acronym in Spanish), a multidisciplinary team dedicated to development and promotion of rural telecommunications in Latin America [8]. VHF/HF and long-distance Wi-Fi is usually preferred for deployment of rural telecommunications networks across Peru. Nonetheless, in [8], it is highlighted the multiple pros of satellite links, like easy and fast implantation in places of difficult access and its high availability and reliability.

1.2 State of art

Multiple generalizations of the original algorithms presented in [3] have been proposed and different platforms for implementing each one have been explored, trying to fit specific requirements [9]. The algorithms in existence can be divided in two types according to the approach for correcting the codeword: hard decision and soft decision. The former only considers the value of each bit, whereas the latter also takes into account the available information provided by the demodulator about probabilities of each bit or non-binary symbol (in the Galois field GF(q)) of taking determined values of the signal constellation used [9][10].

Both of them fulfill opposite demands in communication: Hard decision methods have poor performance in low SNR channels, but are simpler and faster than their soft counterparts. Soft decision decoders have higher complexity and are slower, in detriment of throughput. However, they are less sensitive to channel noise, making them ideal for wireless communication [10].

The most used binary LDPC decoding method is the min-sum algorithm (MSA), a sub-optimal simplification of the original sum-product algorithm (SPA). It is preferred by its simplicity and reliability, at cost of losing resistance against channel noise provided by the latter [9]. Accordingly, several corrections have been proposed for its application [5].

Specific to DVB-S2, in Table 1.1 a comparison between different designs is made, where it can be observed high throughput and medium code length is required. An special case is [11], where is implemented a complete DVB-S2 decoder adaptable for a vast range of code rates, effectively covering the whole standard’s specifications. All of these works employ the MSA as

base, demonstrating the real extend of this decoding form.

Table 1.1: Comparison between DVB-S2 LDPC decoding implementations

Ref.	Calcanhotto **[11]**	Patel *et al.* **[1]**	Pignoly *et al.* **[12]**
Code length	64800	1152	16384
Code rate	1/2	1/2	3/4
Throughput (Mbps)	339.286	2107	2130
Freq. operation (Mhz)	100.058	219.443	250
Resource usage			
Slice LUT	79524	67317	49121
Slice registers	-	27182	-
Flip-flops	45183	-	26903
RAM	405 blocks	-	50 blocks

Some other works explore different aspects of the decoding scheme. For example, in [13] a study is carried out for high level synthesis (HLS) tools and the speeding in hardware development reusing algorithms oriented to software execute in multi-core architectures. Another case is [14], whose authors propose multiple types of simplified algorithms and applies a series of techniques for minor power dissipation. Finally, in [15] a fast convergence LDPC decoder is developed, focusing in bringing good throughput level for short code lengths at low SNR. As can be observed in Table 1.2, each of the parameters under study is slightly prioritized over throughput level without completely ignoring it.

Table 1.2: Comparison between feature's specific LDPC decoding implementations

Ref.	Delomier *et al.* **[13]**	Kim *et al.* **[14]**	Sayed *et al.* **[15]**
Code length	648	648	512
Code rate	1/2	1/2	1/2
Throughput (Mbps)	64,8	110	1400
Freq. operation (Mhz)	333	100	250
Resource usage			
Logic elements	-	-	81844
Slice LUT	11097	19761	-
Flip-flops	20610	7081	-
RAM	133 blocks	24 blocks	19456 bits

1.3 Justification

Satellite communications for video broadcasting need high throughput and low latency to comply with high streams of data sent. To guarantee the effectiveness of a design, the first step is to elaborate a prototype based on tentative algorithms and verify functionality and meeting of requisites.

Naturally, this can be accomplished with usage of high parallelism implementations in graphic processing units (GPUs), with the advantage of having an exceptional flexibility without incurring in high non-recurring engineering (NRE) costs. Nevertheless, this implies high power dissipation, usually not available for mobile systems. On the other side, an application specific integrated circuit (ASIC) combines low power consumption, high level of parallelism achieved by hand-made RTL design and low usage area [5]. Despite that, the engineering costs are too elevated and only cost-effective if they are built in great quantities. Thus, it is non-viable to design a ASIC decoder for prototyping.

As another option, FPGA are situated between both alternatives, making possible to simplify the design compared with ASICs, even more considering strategies as algorithmic state machine and datapath (ASMD) and HLS. For fine-tuning modifications, aimed to power reduction, traditional digital design is possible, combining both techniques for best results. Additionally, FPGA implementations dissipate much less power than a solution based on GPUs, making appropriated for portable applications, which usually lack large energy storage [5][9]. More important, implementations for FPGA can benefit of its intrinsic parallelism also found in GPUs [16], meaning that throughput volume doesn't need to be sacrificed.

Consequently, a FPGA implementation of the decoder under study is presented as an attractive alternative for initial prototyping. For this work, utilization of FPGA for development offers the flexibility and easiness for design necessaries for the prototyping of a LDPC decoder, which would serve as the basis for a future ASIC implementation.

1.4 Objectives

1.4.1 Main objective

- Design of an architecture of a LDPC decoder for a Xilinx Virtex 7 FPGA.

1.4.2 Specific objectives

- Simulate min-sum algorithm in software MATLAB® and test performance.
- Design and synthesize the chosen algorithm in hardware description language Verilog HDL.
- Adjust design for reduced power dissipation and low hardware utilization
- Verify design using functional verification methodology and compare with state-of-art works.

Chapter 2

Fundamentals of LDPC codes and design strategies

As mentioned in the previous chapter, LDPC codes is the current FEC scheme utilized in different applications and still under research, begin possible to extend its capabilities and enhance its performance by a great margin.

This chapter is divided in two parts. The first part presents a background of LDPC codes and latest techniques in improvement of power usage, followed by some general methods used in hardware design. The second part defines the proposal of this work, illustrating a model solution based on previously published state of art works.

2.1 Generalities of error correction

2.1.1 Block codes

There are two families of coding schemes according to the nature of processed data: convolutional codes and block codes (also called algebraic codes). Both are well document in any information theory textbook [2][17][18] and both serve the same purpose: add redundant information to the message prior to its sending. The former operates over blocks of certain length, so the input data is divided for its processing (be encoding or decoding). In contrast, the latter processes data continuously (in a serial fashion).

The redundancy level added is defined by the rate k/n, where k is the number of bits of the original message and n the number of bits of the message encoded. Therefore, the encoder adds $n - k$ bits of redundancy and it is denoted by $C_b(n, k)$. For a message of length k there are 2^k different combinations and for each of them it exists a unique sequence of length n, establishing a

one-to-one pairing, so the decoder is able to recover the message applying the inverse procedure.

2.1.2 Linear block codes

A subclass of block codes are linear block codes, where the sum of two codewords is another codeword belonging to the coding space and all codewords can be represented by a linear combination of k vectors, independent between themselves [19]. Formally, the codewords are a vector subspace of all the combinations of length $\boldsymbol{n}$ [18].

Be the set of these vectors $\boldsymbol{g_0, g_1, ...g_{k-1}}$ and the message with bit sequence $\boldsymbol{m_0 m_1 ... m_{k-1}}$. All codewords **c** can be calculated as

$$\boldsymbol{c} = m_0\boldsymbol{g_0} + m_1\boldsymbol{g_1} + ... + m_{k-1}\boldsymbol{g_{k-1}} \tag{2.1}$$

or in matrix form as

$$c = mG \tag{2.2}$$

where **G** is the generator matrix, used for message encoding and completely defines the linear code.

2.1.3 Parity check matrix

The parity check matrix **H** is associated with the generator matrix and serves the purpose of checking if the received block belongs to the codeword set (if not, an transmission error is declared). It satisfies that each row of **H** is orthogonal to all rows of **G**, that is

$$G \times H^T = 0 \tag{2.3}$$

Be **s** the vector *syndrome*. Replacing 2.2 in 2.3 it is obtained

$$s = uG \times H^T = 0 \tag{2.4}$$

The *syndrome* is equal to zero if the received block has been generated with **G**, which means **c** is a valid codeword and the message has been received with no errors. Otherwise, if $s \neq 0$ then the block has been altered during transmission.

2.1.4 LDPC description

In particular, LDPC codes are a type of linear block code which main attribute is that its matrix **H** is sparse. This means it contains lots of '0's and very few '1's [17]. Another property is that LDPC implementations usually utilize schemes of long code length. This is produced by the fact the error probability decrease exponentially when the code length is increased, as well as increasing the minimum distance of the code [18]. As a direct consequence of this, the complexity of correction of a given sequence is drastically increased, making unfeasible to execute it in a deterministic fashion. In [3], an iterative approach is proposed, so the corrected message is determined in each step and is processed again until all the check-sums of the syndrome vector flag the output as free of errors.

In [3], it is provided a notation for LDPC codes, $C_{LDPC}(n, s, v)$, where n is the code length, s is the number of '1' per column and v is the number of '1' per row [18]. If the number of 1s is constant across all rows and columns, the matrix **H** is called regular. On the other hand, if this number is variable then **H** is called irregular. The irregular version of **H** has better BER performance, enhancing the decoding procedure. However, a regular matrix can benefit from structured construction methods, in contrast with random constructions. Also, it provides a simpler hardware encoder, accelerating data transmission. As it can be observed, the selection of the parity check matrix will depend on the priority assigned to encoding and decoding.

2.2 Tanner graph

The structure of the LDPC codes is defined by a bipartite graph, also known as Tanner graph, which is a graphical representation of the matrix **H**. In the Tanner graph, there are two types of nodes (divided in two rows) connected by edges: variable and check nodes. Each type of node is linked to the other by an edge where there is a non-null element in the matrix **H**. The variable nodes (VNs) correspond to the elements of the codeword, on the other hand, the check nodes (CN) represent the values of the parity-check equations evaluated with its associated variable nodes. It can be observed that VNs are associated with the columns of H and CNs with its rows. The exchange of messages is accomplished between CNs and VNs across the edges until the process is finished.

In the Tanner graph it can be observed *cycles*, paths which starts and finishes in the same node. These paths can produced what is known the self-confirmation of node's belief, which can degrade the overall performance [17]. In this way, it is preferred to eliminate cycles of short size (number of edges of the path).

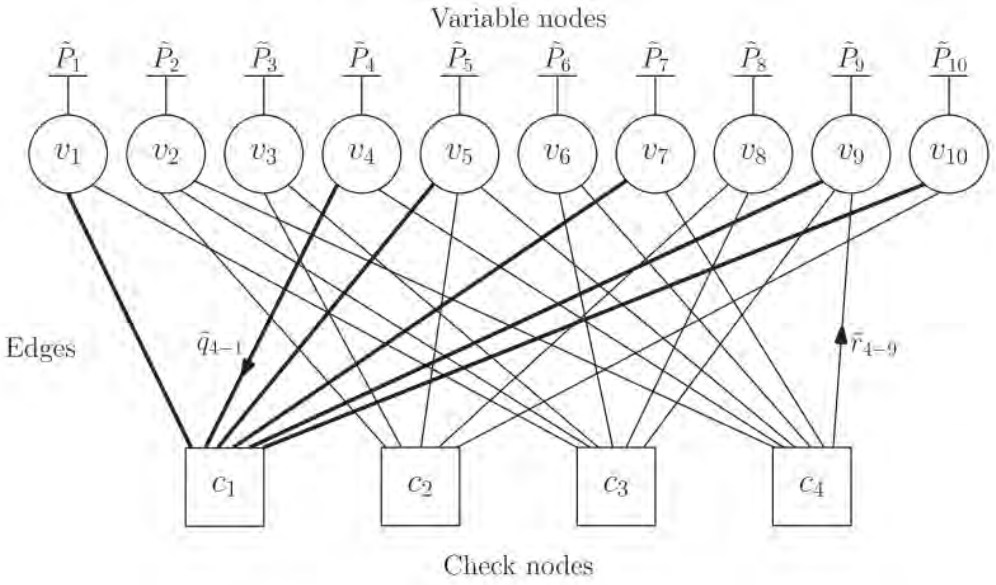

Figure 2.1: Tanner graph, taken from [17]

Whatever algorithm a particular implementation applies for the decoding, all of them follow the same general structure displayed in Algorithm 1 [5].

For Algorithm 1, be v_n a variable node, where $n = 0, 1, ..., N$ and c_m a check node, where $m = 0, 1, ..., N - K$. It is defined $N(m)$ as the subset of VNs adjacent to c_m and $M(n)$ as the subset of CNs adjacent to v_n.

Algorithm 1 General algorithm, inspired by [5]

1: **Initialization of VNs:**
2: **for** $i = 1, I_{max}$ **do**
3: **if** $s = Hc^T = 0$ **then**
4: break
5: **end if**
6: **CN Processing:** For a CN c_m, compute messages $r_{mn}^{i+1}/n \in M(n)$ based on received messages $q_{n'm}^{i}$ from $v_{n'}/n' \in N(m) \setminus n$
7: **VN Processing:** For a VN v_n, compute messages $q_{nm}^{i+1}/m \in N(m)$ based on received messages $r_{m'n}^{i+1}$ from $c_{m'}/m' \in M(n) \setminus m$
8: **Hard decoding:** For each v_n, compute message Q_n and update bit state
9: **end for**

The procedure can be clearly observed in the Tanner graph, starting with the initialization, which loads the information provided by the demodulator to the variable nodes. It follows the CN processing, which for the check node c_m it receives messages from all $v_n \in N(m)$ and send for each v_n in $N(m)$ a message based on information of all adjacent VNs except the same v_n.

For example, in Figure 2.1, during CN processing, c_1 would received messages from all v_n connected by edges ($n = 1, 4, 5, 7, 9, 10$). To compute the message r_{1-1} directed to v_1, it would

take all messages q_{n-1} except q_{1-1}. The same routine is applied to VN processing, with the difference that CN and VN roles are exchanged. In this step is v_n which receives and computes messages. Finally, in v_n the values Q_n are calculated and the state bit in that VN is determined.

2.3 Classification of algorithms

Multiple generalizations of the original algorithms presented by Gallager [3] have been proposed and different platforms for implementing each one have been explored, trying to fit specific requirements. The algorithms in existence can be divides in two types according to the approach for correcting the codeword: hard decision and soft decision. The former only considers the value of each bit, whereas the latter also take into account the available information provided by the demodulator about probabilities of each bit or baud of taking determined values of the communication constellation.

Both of them fulfill opposite demands in communication: Hard decision methods have poor performance in low SNR channels, but are simpler and faster than its counterpart. Soft decision decoders have higher complexity and are slower, in detriment of throughput. However, they are less sensitive to channel noise, making them ideal for wireless communication.

2.3.1 Hard decision algorithms

This kind of decoding methods usually are implemented in reliable channels, such as optical communications, because of its speed and low complexity at cost of considerable Bit Error Rate (BER) degradation. They are only defined for binary inputs (Galois Field, GF(2)) and don't take into consideration the magnitude of the value returned by the demodulator, only the sign of that value is quantified [17]. Consequently, in each iteration the bits of the received sequence are flipped if the check-sums of the check nodes adjacent to the variable node indicate it, without further processing.

In this class are present majority-logic and bit-flipping algorithms. One of the techniques proposed by Gallager are included in this category (Gallager A and B Bit Flipping Algorithms [20]).

2.3.2 Soft decision algorithms

This type of techniques can be classified in two, depending on the finite field (also called Galois field) where it operates. The decoding process is similar between both decoding schemes with significant difference in complexity. In general, if the decoding process is carried out over a

Galois Field of dimension q or GF(q), where $q > 2$, the state-of-art works implement its operations in the Fourier domain to simplify certain computations (convolutions, for example) and reduce computational cost [5]. In spite of that, each algorithm for GF(2) can be generalized for GF(q). More common is to decode the received message in its binary form, that is the same as GF(2) and its the same field the hard decision algorithms operates over. Thanks to its great performance, it concentrates most of research in the area and are the class of codes included in most standards.

2.4 Soft decision binary algorithms

2.4.1 Sum Product Algorithm (SPA)

Also called belief propagation, it is the original algorithm proposed by Gallager [3]. It is characterized by its heavy numerical complexity and low throughput level, with a substantial penalty in resource usage. Nevertheless, it has the best response against channel noise, achieving very low BER even at low signal-noise ratio (SNR).

2.4.2 Logarithmic Sum Product Algorithm (LSPA)

Due to its exceptional performance in low SNR channels, research has focused in simplifying SPA without degrading its robustness. The most complicated step is the CNs updating, which it is better operated in the loglikehood rate (LLR) domain, defined in [9] as

$$LLR(u|y) = \frac{p(u=0|y)}{p(u=1|y)} \tag{2.5}$$

which represent the probability of **u** of taking 0 or 1 having received **y**.

In the LLR domain, products or quotients are converted to additions or subtractions, simplifying the decoding process at cost of including complex mathematical functions (tanh and $\tanh^{-1}$), which are usually implemented with lookup tables.

2.4.3 Min Sum Algorithm (MSA)

A simplification of the LSPA processing in the CN step derived in this sub-optimal algorithm, which meant a simpler implementation at cost of decreasing performance (almost a 0.5 dB penalty) [5]. Nonetheless, MSA is the base for most of decoders implementations, with abundant research for recovering lost performance by using it coupled with other techniques in the final decoder (interleaver, BHC codes) or applying corrections over the hard decision step of the sequence.

Within the latter, there are included scaling, offsetting and self-correction as primary techniques [5].

Be γ_n the information of each received bit provided by the demodulator. For Algorithm 2, each value q_{nm} and r_{mn} is expressed in LLR domain.

Algorithm 2 Min-sum algorithm, inspired by [5][20]

1: **Initialization of VNs:** $q_{nm} = \gamma_n$, $c_n = sign(\gamma_n)$
2: **for** $i = 1, I_{max}$ **do**
3: **if** $s = Hc^T \neq 0$ **then**
4: break
5: **end if**
6: **CN Processing:** For a CN c_m,

$$r_{mn}^{i+1} = \left(\prod_{n' \in N(m) \backslash n} sign(q_{n'm}^{i}) \right) \min_{n' \in N(m) \backslash n} \left(|q_{n'm}^{i}| \right) \tag{2.6}$$

7: **VN Processing:** For a VN v_n,

$$q_{nm}^{i+1} = \gamma_n + \sum_{m' \in M(n) \backslash m} r_{m'n}^{i+1} \tag{2.7}$$

8: **Hard decoding:** For a VN v_n,

$$\begin{aligned} Q_n^{i+1} &= \gamma_n + \sum_{m \in M(n)} r_{mn}^{i+1} \\ c_n &= sign(Q_n^{i+1}) \end{aligned} \tag{2.8}$$

9: **end for**

2.5 Architectures

Another factor to take into consideration is how a specific algorithm is implemented in the chosen platform. Even centering for FPGAs, there is a vast design space, subject of study of multiple works [5, 9][21].

Of special interest is the fully parallel architecture, which places all VNs and CNs in hardware. This approach can achieve exceptionally high throughput, using the flooding schedule for node updating, at cost of inflexibility in terms of code rate and code length. These decoders are static in functionality, so they are only able to operate in a single configuration. Moreover, for large code lengths, the network's size that connects the VN layer and the CN layer is significantly increased, even taking more physical space than the nodes themselves.

In the other spectrum's side, a fully serial architecture only implements one node per type, which are multiplexed between each clock cycle to cover all the nodes defined in the Tanner

graph. As expected, it suffers of low throughput and increased logic complexity. In exchange, as all edges of the graph are stored in memory [9], this configuration is much more flexible than the previous one, allowing shifts of the decoding structure in run-time. Also, occupied space is lower because of the reduced number of nodes.

To obtain a balance between throughput and hardware usage, the partially parallel implementation places a specific number of nodes and multiplex them. This type of arrangement can take advantage of specific parity-check matrix structured to follow some construction techniques of the matrix **H** [9], e.g. quasi-cyclic LDPC codes [20].

2.6 Schedule policies

The schedule policy, also known as scheduling [9], scheduling scheme [20] or decoding schedule [5] is how and in which order the nodes are processed. The most typical used are the flooding and layered belief propagation, each of them covering different requisites.

2.6.1 Flooding scheme

Also called two-phased message-passing, this process update a whole layer of one type before proceed to the other one. In this case, an iteration is complete when both layers have been updated. This policy is used where a parallel architecture is implemented, in order to exploit to the fullest the architecture's capacity. The required control unit is much more simpler than for other policies.

2.6.2 Layered Belief Propagation

Other names for this policy are turbo-decoding message-passing and layered scheduling. In contrast with the flooding scheme, one type of node (usually CN) is processed in a way that adjacent VNs are updated in the next cycle. In this fashion, in the same clock cycle some CN and VN are been processed. Particularly, there is some limit to the level of parallelism it can be reached. However, layered scheduling converges at a faster rate than flooding scheduling (almost half of the iteration required), balancing throughput and power dissipation.

2.6.3 Other policies

Some schedules focus in other parameters of decoding. Such is the case of Informed Dynamic Scheduling, which inspects the messages sent and determines which node should be activated to obtain the greatest improvement in belief [9]. As expected, this adds another set of calculations

that must be performed, increasing hardware complexity. Despite its promising advantages over more traditional schedules, few works have focused in its developing.

Other methods are the Row-layered scheme and the Column-layered scheme, where the matrix **H** is divided in blocks of rows of columns, respectively. In case of row-layered policy, the messages from CN to VN derived in a iteration from one block are processed to calculate the messages from VN to CN of the next block, and vice versa for column-layered policy [20].

2.7 Design techniques

In this section, some methods for design of digital circuits are described, which are considered as guidelines for the implementation of the LDPC decoder and are of utmost importance in the impact of performance and scalability of this one.

2.7.1 Pipelining

Pipelining is applied over the combinational part of a circuit and consists in dividing a large combinational portion in smaller pieces, in order to process several tasks at the same time [22]. To avoid race conditions and ensure the timing of the signals, registers are added between each section. This means that in each clock cycle different tasks are executed and an new output is ready.

The main drawback of a pipelined circuit is that the delay to obtain the output is increased. In spite of that, the circuit's throughput is also increased, as the minimum period required is reduced. The relevance of this technique is that enables a gain in the maximum operation frequency reducing the execution time of critical paths, at cost of increasing the number of registers used and the occupied area by the chip.

2.7.2 Algorithmic State Machine and Datapath (ASMD) design

Complex problems require a top-down approach solution in order to be effectively solved. Starting with a simple representation (e.g. general algorithm) enables a full understanding of the problem and the subdivision in smaller, more manageable, blocks.

In hardware design, the solution is presented in two well differentiated blocks, the *datapath* and the *control path*. The former realizes all required operations of the algorithm, whereas the latter determines when and which operations are executed. The usual way of designing the control path is by using a Finite State Machine (FSM), which is a sequential circuit with internal states.

Each state rules which action is executed and the transition between states is performed in each clock cycle [22].

A possible representation of a FSM is an Algorithmic State Machine (ASM) chart, with the advantages of being more descriptive than the traditional representation. In this manner, it is faster and simple to convert the general algorithm to a FSM, defining each state and its corresponding outputs [23].

Another benefit of using a ASM is the less intrusive inclusion of the register-transfer logic (RTL) operations of the datapath. In that situation, the chart is called algorithmic state machine and datapath (ASMD) and provides a less cumbersome method to transfer an algorithm to hardware synthesis, saving time and engineering costs. This alternative methodology has been extensively covered in [22][23], with the main difference with FSMD that operations in ASMD are not executed in a clock-by-clock basis, introducing a delay in register storage.

2.7.3 High Level Synthesis (HLS)

Design of RTL-level circuits are part of the core in a FPGA implementation, a common property shared with ASIC ones. This procedure quickly escalates with solution's complexity, what entails high non-recurring costs (NRE).

LDPC decoders are not exempt of this procedure, but another design strategy for FPGA implementations, called high-level synthesis (HLS) tools, recently have attracted interest because of its multiple advantages over other techniques. For example, HLS enables acceleration of hardware development times. These algorithms are implemented in a high-level programming language, such as Java or C++, in opposition to hardware description languages as Verilog (industry's standard for applications based on FPGA) or VHDL [10]. Algorithms designed for using in multicore platforms [13][16] can be reused for FPGA once transformed for these architectures. Aiming to compete with GPU-based decoders' high throughput, FPGA implementations have as its main advantage a reduced power consumption.

2.8 Design for low-power applications

The power dissipation of a digital design can quickly increment with a high number of logical elements and a continuously increasing clock frequency. Some measures are needed to balance the BER performance, throughput and power consumption in order to design a portable device.

Some common techniques include clock gating, which the disabling of the clock signal at clock level, to suspend a circuit operation. Usually considered a bad design practice, disabling the clock

signal stop the transistor of flipping states, effectively reducing power usage. Nonetheless, this should be made after completing the synthesis of the hardware, during power optimization assisted by software [22]. This procedure serves as the basis for the following described techniques, which also involve the suspending of a part of a circuit.

2.8.1 Early stopping scheme

When transmissions are sent in a low SNR channel, the number of corrupted bits exceeds the correction capacity of the FEC code used. As a consequence, the received sequence cannot be corrected and doesn't converge to a valid codeword, so it oscillates until it reaches the maximum of iterations specified in the algorithm. At that point, the decoding process is declared failed, wasting unnecessary clock cycles and power in an unrestorable message. Certain implementations found in literature [24][25][26] used a stopping policy when a undecodable codeword is detected, with the objective of reducing the number of iterations executed for such codeword, thus reducing total power dissipation and reducing time spent in decoding that message.

2.8.2 Forced convergence

Although certain bits of a codeword could be quickly decoded (in few iterations), the algorithm still updates its belief and reinforce those bits until the syndrome vector indicates the end of the process. This technique allows to stop the belief updating of variable nodes by deactivating them, if they have a strong belief value and certainly its value is determined to be correct. Therefore, power is saved in the subsequent iterations [27][28].

2.9 Digital Video Broadcasting - S2

The European standard for digital television (DVB-S2) establishes Low Density Parity Check Codes as part of a bigger decoding scheme, arguing its limited algebraic structure and easily adjustable degree of parallelism. It defines a wide range of code rates and 2 code lengths: 16200 (short frame) and 64800 (normal frame) bits per received block [29]. Specifically, this standard belongs to the Irregular Repeat and Accumulate (IRA) LDPC class [30].

It fixes 50 iterations for obtaining minimum bit error rate and adds a outer Bose–Chaudhuri–Hocquenghem (BHC) code as an redundant insurance to the already very high performance of LDPC inner stage.

2.10 Model solution

As mentioned in the previous chapter, this work has the objective of designing a LDPC decoder.

This work will implement the min-sum algorithm, following the main trend of research [30][31], which highlights its performance-complexity relation. A focus is made over Digital Video Broadcasting-S2 standard, which constitutes a major challenge for a low-power design. The main difficulty is the long code length in both of its versions (16200 and 64800 bits, respectively). This represents a top requirement in the specifications space, meaning that, for purpose of this work, it is considered an upper limit. The scope of this work will be the inner stage of a DVB decoder, with the purpose of counting with an already established framework reference covered in several publications [1][32][33].

Because of the exceptionally long code length determined in DVB-S2, a fully-parallel approach is not feasible. In order to comply with this requisite, a partially-parallel architecture will be used, combined with a flooding schedule for reduced hardware complexity. On the other hand, the ASMD methodology wil be employed by its simplicity to convert pseudocode to hardware description language. Furthermore, it does not require specialized software, in contrast with HLS, what results in a portable design across different platforms.

To extend the previous defined base, strategies aiming to reduce power dissipation will be applied. Its has been already presented the early stopping scheme and the forced convergence. Both of them focused in reducing the number of clock cycles for the decoding of a block, covering two extremes cases. The former covers undecodable sequences produced when transmitting over low SNR channels. The latter applies over high SNR channels, producing data streams with fewer errors.

In Figure 2.2, the block diagram of the proposal is displayed. It can be seen the main layers of variable nodes and check nodes, as well as the permutation network. This last block can be reconfigurable in function of the selected code rate and length. Because there is not necessity of overwriting the configuration for each code rate after the initial synthesis, they are stored in read only-memory. The syndrome calculation is performed from data provided by CNs and the convergence detector analyze the syndrome vector as a whole. Both of them send to the control unit if the iterations must be stopped.

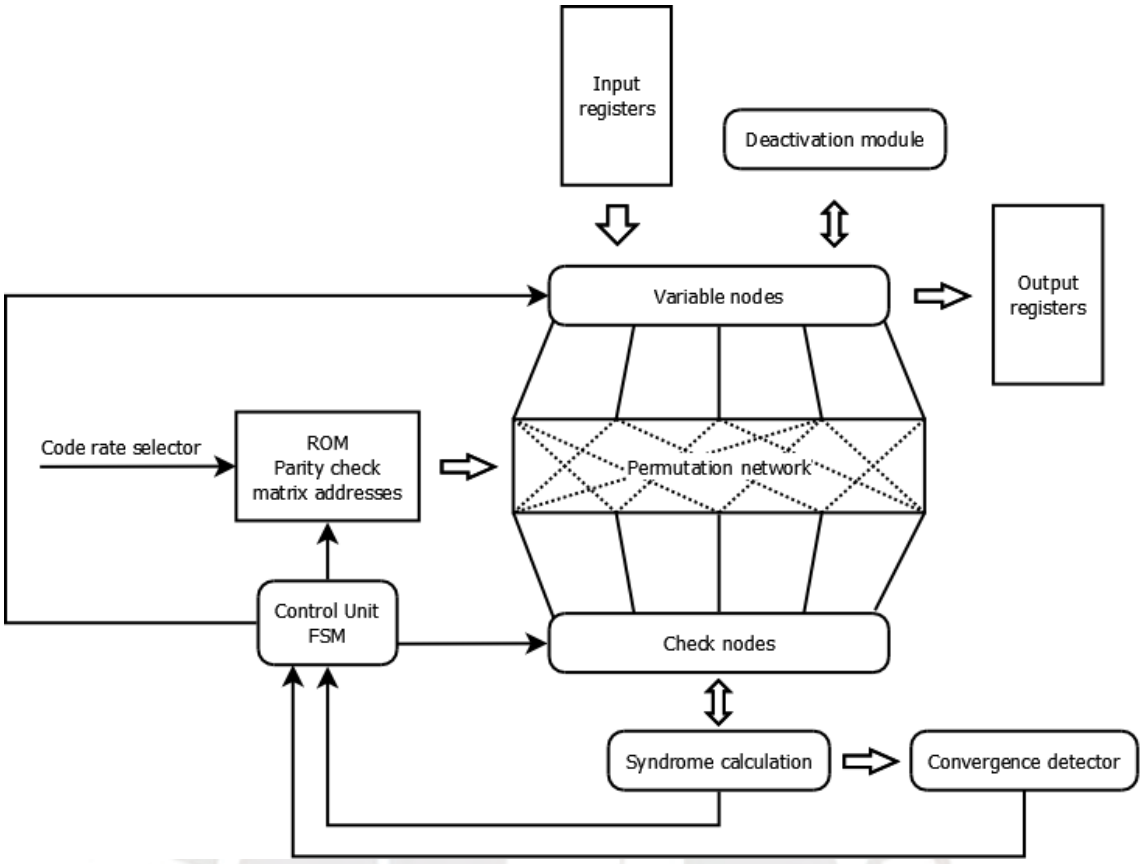

Figure 2.2: Block diagram

Chapter 3

Design of the LDPC decoder

The present chapter shows the design of the complete LDPC decoder. For this purpose, methodologies such as Algoritmic State Machine and Datapath (ASMD), pipelining and clock gating are used at hardware level. As explained in the previous chapter, at hardware description level some techniques are applied in order to improve performance (specifically, throughput), reduce power dissipation and optimize operation frequency.

This chapter is divided in three sections. The first part corresponds to the architecture of the decoder and the description of its behavior. The second part covers the design of the individual modules of the standard decoding operation, that is, without including the power-saving modules, which include the elemental functional units, the barrel shifter, the distribution of the ROM and RAM memories and the control unit. Finally, the third part introduces the power-saving modules and its effect in the control flow, being these modules the convergence analyzer for early stopping and the deactivation module for forced convergence.

3.1 Considerations

For the design, the FPGA in use will be the xc7vx550t of the Virtex 7 family from Xilinx company. It follows a partially parallel architecture with two-phased message-passing or flooding schedule. The selected features to be implemented are code length of 64800 bits, corresponding to the broadcast mode defined in [29] and code rate 1/2. It is assumed the maximum number of iterations is 10, unless otherwise stated. The verification of the design is executed following the functional verification framework elaborated in [34].

3.2 Architecture of decoder

This section presents the general scheme of the decoding core and explains its behavior during processing.

The decoder receives the LLR values sent by the demodulator by a serial port, and returns the estimated bits of the received sequence, having corrected the possible errors introduced during transmission.

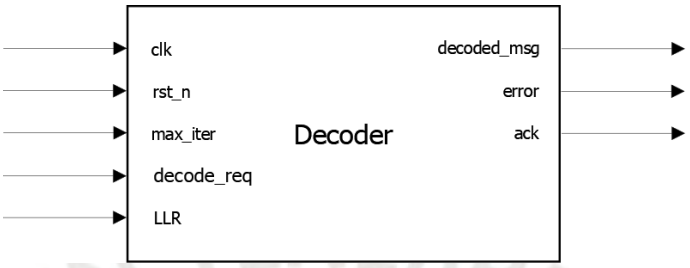

Figure 3.1: Decoder block, by the author, based on [11]

Table 3.1: List of input and output signals

Name of signal	N of bits	Description
		Inputs
clk	1	Clock
rst_n	1	Reset, low active
max_iter	6	Sets the maximum number of iterations before declared decoding as failed
decode_req	1	Requests a decoding petition and mark the beginning of a new input sequence
LLR	5	Serial input of LLR values from demodulator
		Outputs
decoded_msg	1	Serial output of decoded message
error	1	Flag that indicates a failed decoding
ack	1	Marks the end of decoding, start of new output sequence and decoder ready to received requests

The control unit is forever waiting a request signal that marks the beginning of a execution cycle. It starts the loading of the LLR values from the demodulator through the serial input interface. These values are written in the I/O buffer and are read again to be stored the memory bank, so it can be possible to be accessed in parallel. Once both processes have finished, the

decoder executes its processing, starting with the check node processing, according to algorithm 2.

During the check node processing, the check nodes start the reading of channel and v2c messages from each memory bank, and deliver the c2v messages and the corresponding checksum. The latter is sent to the parity check module, where the complete syndrome vector is evaluated to continue or stop the decoding. The c2v messages are written to the corresponding memory and, once finished, the variable node processing starts.

Once started the variable node processing, the variable nodes read the c2v messages from the memory banks and deliver the v2c messages to be once again written in the same RAMs. The hard decoding is executed in this phase, determining the new bit value held for each V and storing those values in a different RAM memory.

The control unit evaluates if the maximum number of iterations is not reached yet and if the evaluation of parity check module indicates a syndrome vector different from zero. If so, it orders the execution another iteration, beginning with the check node processing.

On the contrary, the decoding is finished, the bit values are read from RAM, written to the I/O buffer and sent by the serial output. Finally, a output serves as flag indicating if the decoding was successful or failed.

3.3 Units of standard MSA

3.3.1 Input/output buffer

The I/O buffer is a FIFO memory which stores the incoming LLR values from sent by the demodulator or the final bit values to be sent as decoder output. Both the input and the output are serial interfaces, so is not possible to have the required number of input values available to be accessed in parallel. The same problem is present for the output bits, so both kind of values are needed to be stored first in the memory bank before being read or written by the functional units.

3.3.2 Check node

The check node is responsible of the verification of the checksum of a set of variable nodes, as expressed in the parity-check matrix and the Tanner graph of the decoding scheme. As presented in Chapter 2, the equation of a CN c_m to calculate the message for each adjacent variable node v_n is

$$r_{mn}^{i+1} = \left(\prod_{n' \in N(m) \backslash n} sign(q_{n'm}^{i}) \right) \min_{n' \in N(m) \backslash n} \left(q_{n'm}^{i} \right) \tag{3.1}$$

which takes the minimum LLR value sent by the subset of VNs that excludes the VN receptor of that specific message.

The design of this module is based on works of Andrade [5] y Calcanhotto [11], which receives variable-to-check-node messages (v2c) one by one during the uploading stage and delivers one check-to-variable- node message (c2v) to each adjacent VN during the downloading stage. For each message receives or send, one clock cycle is used, for a total of $2 \times n$ clock cycles, where n is the number of adjacent VNs. It is assumed the v2c messages are in sign magnitude representation.

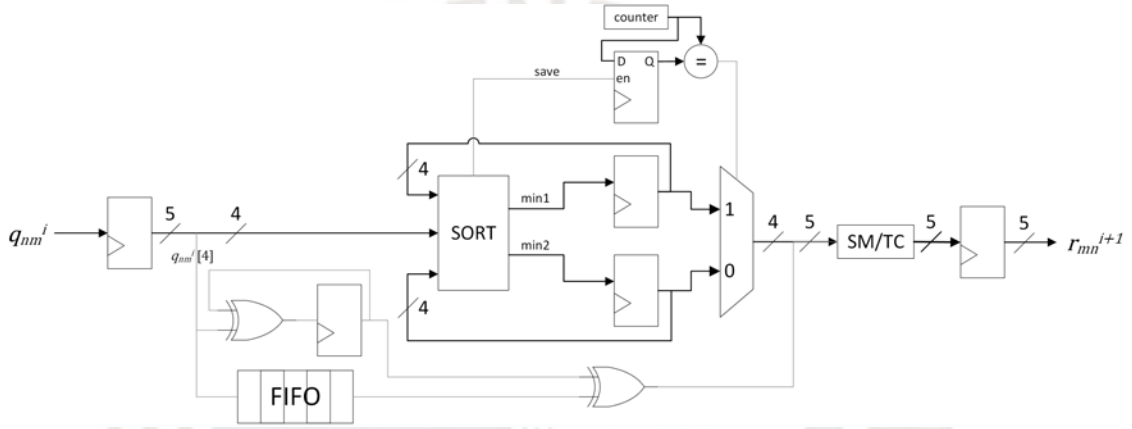

Figure 3.2: Check node module, minimum v2c search and formatting

In figure 3.2, the block diagram of the proposed check node is observed. During the uploading stage, the sign of each v2c message is stored in a FIFO memory and a XOR operation is executed between each entry sign and the sign accumulator. The magnitude of the message is processed in a sorting module, where it is compared with the current first and second minimum values found. The sorting block outputs the two new minimum values.

At the end of the uploading stage, the search of the minimum v2c magnitude finds the two minimums of the set, covering all possible outputs. The first minimum covers all VN with LLR greater than theirs and the second minimum is returned for the VN which sent the first minimum LLR. It cannot take the first one because that one have been excluded of the VN set. To determinate which node receives the second minimum magnitude message, the entry order of a v2c message is saved if the values corresponds to the new first minimum value. The sign of c2v output is calculated applying the XOR operator between the original sign stored in FIFO memory and the sign accumulator. Finally, the c2v is converted to two complement representation, ready to be sent.

The check node module serves another purpose, exploiting its connectivity with its adjacent

nodes, and its the fact that the value of the sign accumulator is equal to the checksum of the adjacent values. Taking advantage of this, the sign accumulator is sent to the parity checker, becoming part of the syndrome vector.

3.3.3 Variable node

The variable node executes the updating of the LLR of each received bit, according to the messages sent by the set of adjacent check nodes for that VN. The equation of a VN v_n to calculate the messages for each adjacent check node c_m is

$$q_{nm}^{i+1} = \gamma_n + \sum_{m' \in M(n) \backslash m} r_{m'n}^{i+1} \tag{3.2}$$

which is the sum of the LLR provided by the demodulator and the value sent by the set of CNs that excludes the CN receptor of that specific message.

It also determines the new binary value held according to the information of the check node, described by equations

$$\begin{aligned} Q_n^{i+1} &= \gamma_n + \sum_{m \in M(n)} r_{mn}^{i+1} \\ c_n &= sign(Q_n^{i+1}) \end{aligned} \tag{3.3}$$

where the '0' or '1' of the the bit is the sign of the sum of the LLR of the channel and the whole set of messages sent by the adjacent CNs.

Notice that both equations 3.2 and 3.3 represent the same operations, with the difference of which VN are included in the second term of the expression. Rewriting equation 3.2 in terms of equation 3.3, this can be expressed as

$$q_{nm}^{i+1} = Q_n^{i+1} - r_{mn}^{i+1} \tag{3.4}$$

which is the equation to be included in this design.

Similarly to the check node, the design of this unit is based on works of Andrade [5] y Calcanhotto [11], where a c2v message is taken one by one during the uploading stage and a v2c message is delivered during the downloading stage, for a total of $2 \times m$ clock cycles, where m is the number of adjacent CNs. It is assumed the c2v messages are in two-complement representation.

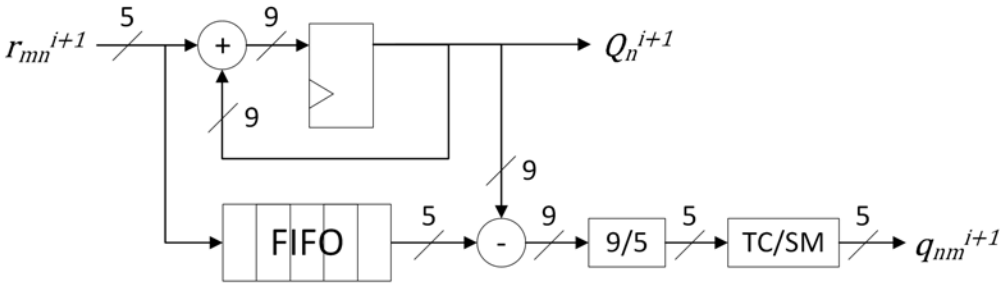

Figure 3.3: Variable node module

In figure 3.3 the block diagram of the proposed variable node is presented. It works similarly to the check node, with the same uploading and downloading stage. During the former, the LLR from demodulator (gamma term in eq. 3.2) and the recevied c2v messages are added to the sum accumulator, while only the c2v are stored in a FIFO memory. The new bit value is taken from the sign of the total sum of this step. In the latter, following 3.4, the returned v2c values are calculated subtracting the stored c2v from the final sum. The c2v messages are then converted to sign magnitude representation before being sent to the CNs. The sign of the updated LLR value is stored in memory.

3.3.4 Parity node

As presented in chapter 2, the DVB-S2 standard makes use of the LDPC codes with a common property: each check node is adjacent to two consecutive check nodes by two variable nodes of degree two, one VN for each of them, as presented in [35]. This relation is shown in figure 3.4.a

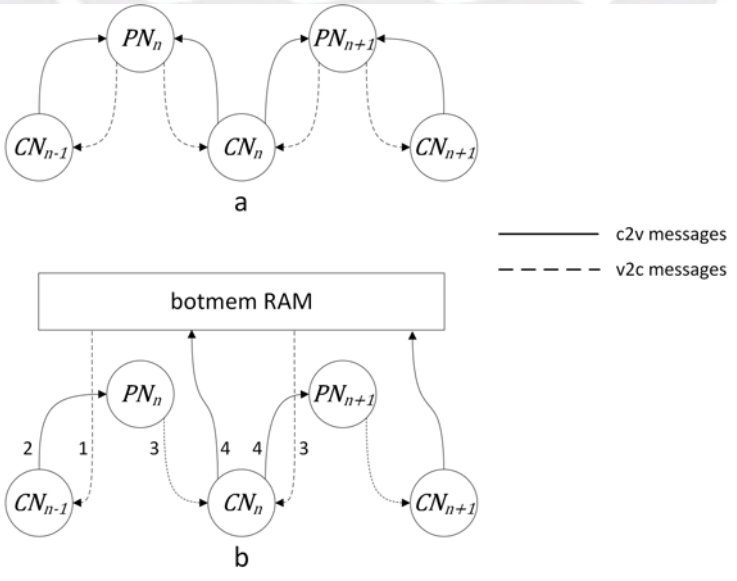

Figure 3.4: Message exchange between CN and PN. Adapted from [35]

This means that, for a variable node of degree 2 (or parity node, onwards), the v2c message sent to the first adjacent CN is equal to the c2v message received from the second adjacent CN, plus the received LLR from the channel for that node. The same applies for the other adjacent CN. For this VN, be $M(n)$ the subset of adjacent CN, where $M(n) = \{m_1, m_2\}$. Replacing it in 3.2, the equation is simplified to obtain 3.5.

$$q_{nm}^{i+1} = \gamma_n + \sum_{m' \in M(n) \setminus m} r_{m'n}^{i+1} \tag{3.2}$$

$$\begin{aligned} q_{nm_1}^{i+1} &= \gamma_n + r_{m_2 n}^{i+1} \\ q_{nm_2}^{i+1} &= \gamma_n + r_{m_1 n}^{i+1} \end{aligned} \tag{3.5}$$

From the equations 3.5, it can be observed the original degree-2 VN has been simplified to a single adder. Considering the property shown in 3.4, this scheme can be exploited to send the c2v message from a CN_{n-1} to the next CN_n, immediately after finishing the processing in the former node.

Based in [35], the resulting design is a module which takes the two c2v messages sent to the parity node and the LLR from the channel, as displayed in 3.5. As the output of the parity node is connected to another check node, the output of the adder is truncated and converted to sign-magnitude representation.

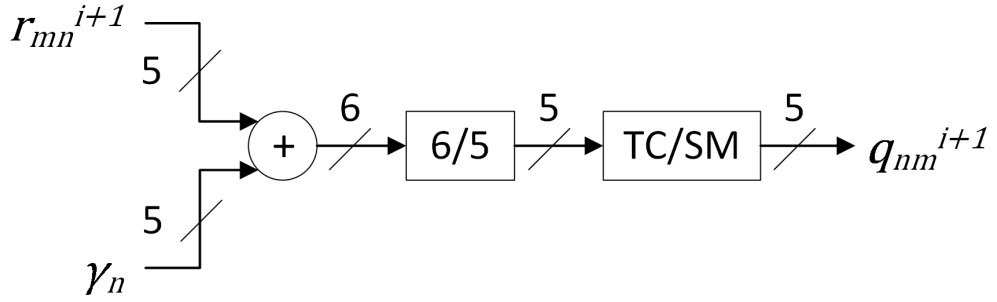

Figure 3.5: Parity node module

3.3.5 Parity check module

The parity check module (PCM) serves the purpose of holding the checksums of the CNs during the duration of each iteration. The checksums sent by the check nodes altogether form the syndrome vector, which, as presented in equation 2.4, is equal to zero, it indicates the decoded sequence has been corrected. Therefore, The parity check module take all the checksums and compares them to determine if the decoding process must continue or not. The result of this

comparison is communicated to the control unit.

This unit is designed following a tree structure, so the input fan-in is not too large, otherwise it could introduce important delays and impose a penalty in frequency. This structure can be exploited inserting registers between each layer, considering the minimum clock cycles used in check node processing.

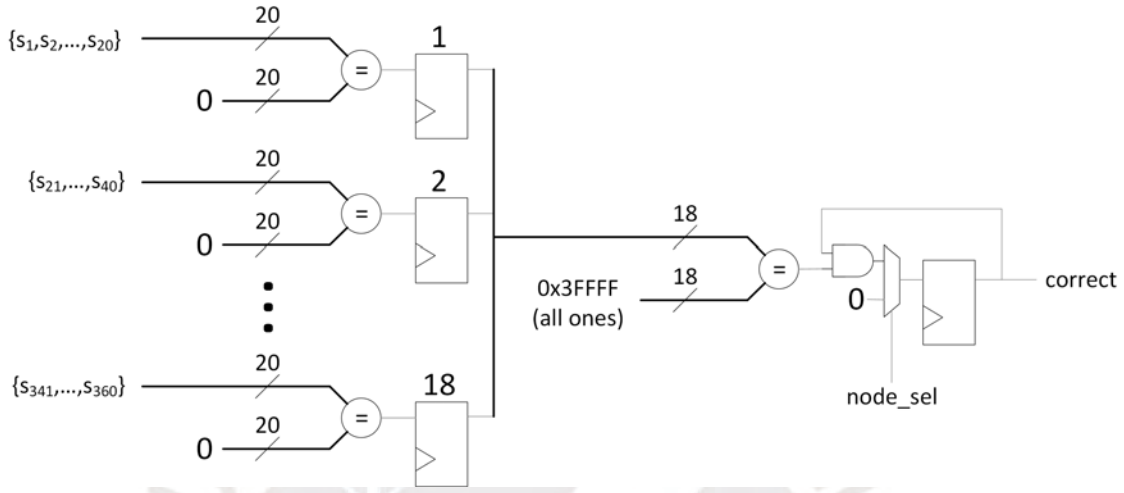

Figure 3.6: Parity check module diagram, by the author

The designed PCM is shown in figure 3.6, where it can be seen comparators are separated in two layers. The first layer they take equal subdivisions of all available bits at the moment, equal to M (number of parallel functional units). In the second layer, a single comparator takes the results of the previous ones and compare them against a all-ones constant of fixed width, equal to the number of instantiated comparators in the previous layer.

For each set of VN processed, the flag value is registered and sequentially updated by a AND gate, so if in one set the partial syndrome vector is different from zero, the flag will be set to zero no matter the incoming comparison results. At the final of each VN processing, the output register is synchronously reset by an auxiliary multiplexer, preparing it for the next iteration.

3.3.6 Memory bank

The DVB-S2 standard defines a careful structured parity check matrix following the Irregular Repeat Accumulate (IRA) scheme, providing a simplified encoding process and a memory scheme that takes advantage of the structure. As presented in [36] and applied in multiple works [5][11], the memory is conceptually distributed in two parts. The former, called top RAM, references the sub-matrix A and contains the edges between CNs and VNs of degree greater than two, that is, the information nodes. The later is named bottom RAM and references the sub-matrix B. Unlike top

RAM, it contains edge between CNs and parity nodes (VNs of degree less or equal than two).

In [36], an example about how the edges of the parity check matrix would be mapped in the memory bank. In figure 3.7 the matrix H is shown and the sub-matrices A and B are identified.

$$H = \left[\begin{array}{cccccccccccc|cccccc} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right]$$

A B

Figure 3.7: Example of parity check matrix, from [36]

The restriction imposed over the matrix [29], established that, for a set of M information bits (belonging to the sub-matrix A), the adjacent CNs of the VNs are determined as

$$\begin{aligned} M(v_1) &= \{a_0, a_1, a_2, ..., a_{d_v}\} \\ M(v_i)) &= \{(a_0 + (i-1)q) mod(N-K), \\ &\quad (a_1 + (i-1)q) mod(N-K), \\ &\quad (a_2 + (i-1)q) mod(N-K), \\ &\quad (a_{d_v} + (i-1)q) mod(N-K)\}/i \leq M \end{aligned} \tag{3.6}$$

where $q = \frac{N-K}{M}$ and M is fixed to 360 in DVB-S2. In general, the first set $M(v_1)$ is randomly chosen [29] and its those indexes are the only ones that need to be stored to define the whole group, reducing memory requirements.

In this scheme, the VNs read the RAM in a in-order bank-aligned way, where if the VNs v_m of this block are adjacent to CNs $c_n/n \in N(m)$, the messages c2v sent to VN v_i are stored across d_c adjacent rows and in column i.

The CNs do not follow this ordered reading schedule, having its adjacent nodes scattered in random (not adjacent) rows and not in the same column. However, for a given set of M parallel functional units and, consequently, M CNs, if the first CN reads the messages v2c of its adjacent VNs from rows $a_1, a_2, ..., a_{d_c}$, the $M-1$ check nodes will read from the same rows. Also, for each row, the if the first VN v_m of the set $m \in M(n)$ of CN c_n is assigned the column j, then the VN of that row that for CN c_{n+1} is assigned the column $j+1$, and so on. This difference in order

of reading is represented in figure 3.8, left image (a), that is the distribution of top RAM.

To determined which positions are needed to be read to feed each CN, it is only necessary to know the rows where the messages are stored, and the offset respect to the index of the CN that results in the column of a given VN.

At this point, it is clear how the RA structure simplifies hardware requirement. For each group of M CNs, it is only required to store the rows where the VNs store the messages, and the offset of each column, reducing static memory requirements. Additionally, because a fixed routing mapping for the message exchange hasn't been used, the same hardware distribution can be used for other parity check matrices that comply with the RA structure. DVB-S2 takes advantage of these restrictions and defines multiple code rates, each of them with different row indexes and offsets for each M set [29].

This structure enables a partially parallel approach to the placing of the functional units, where the number of nodes in parallel can be any integer divisor of M [37].

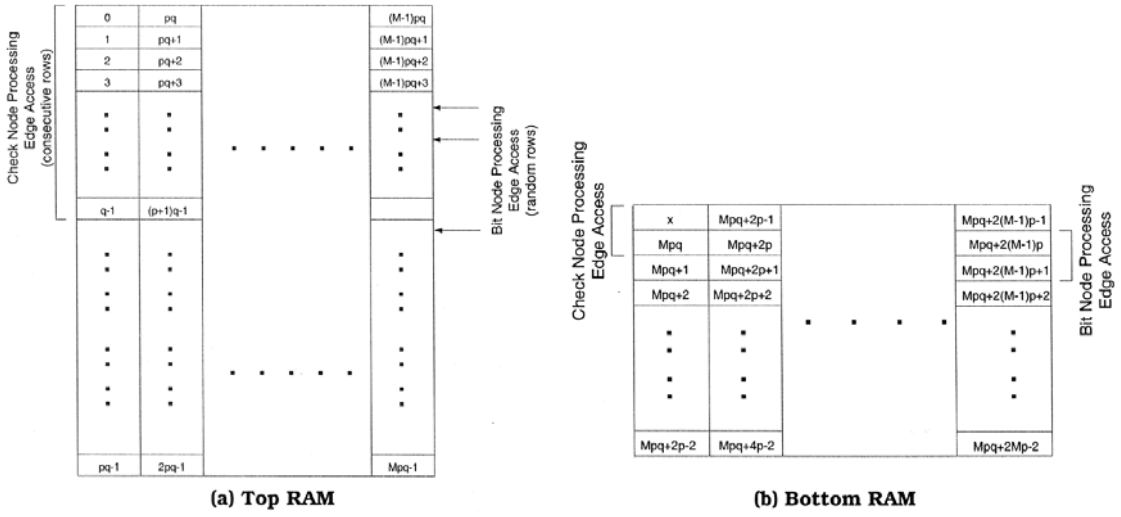

Figure 3.8: Conceptual division of RAM bank, from [36]

In short, at a conceptual level, each column of the memory bank is assigned to a functional unit, where both VNs and CNs access to transmit v2c and c2v messages, respectively. Overall, how each message is mapped in RAM is presented in figure 3.9. It can be seen that, to implement the shift feature described for reading of v2c, a barrel shifter is embed between the memory bank and the functional units.

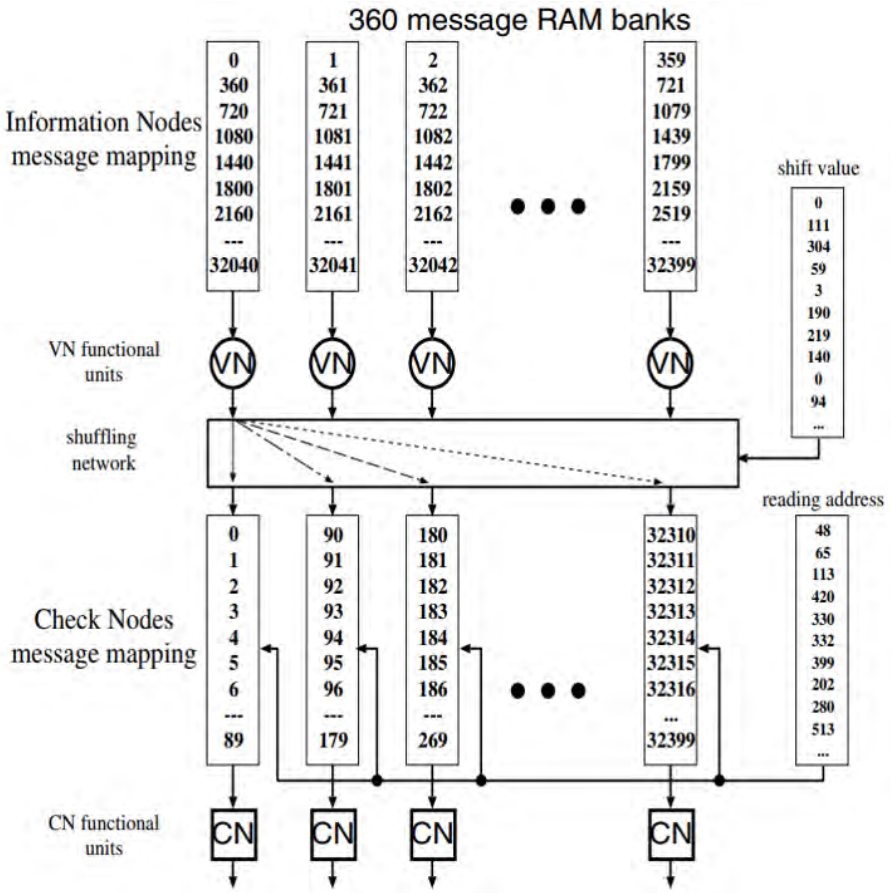

Figure 3.9: RAM distribution scheme, from [35]

d

3.3.7 Barrel shifter

To take advantage of the special properties of the parity check matrix from DVB-S2 standard, a barrel shifter is added between the functional units and the memory bank.

It serves the same purpose of a shift register, with the main difference that is formed almost entirely by combinational elements. It takes as inputs the data bus to be shifted and the number of positions to be shifted, as each message take a constant width and it is not required to shift the data in a bit by bit basis but in blocks of the same width than LLRs. As expected, it only takes one clock cycle to update the output (registered in the subsequent module) at cost of a decreased operation frequency.

Formed by successive multiplexers, each of them shift the data by a number of blocks equal to a integer divisor of M or not shift at all, covering the whole spectrum from 0 to $M - 1$. Some intermediate registers are added to improve the frequency operation. This means the output is delayed by one clock cycle compared with the input.

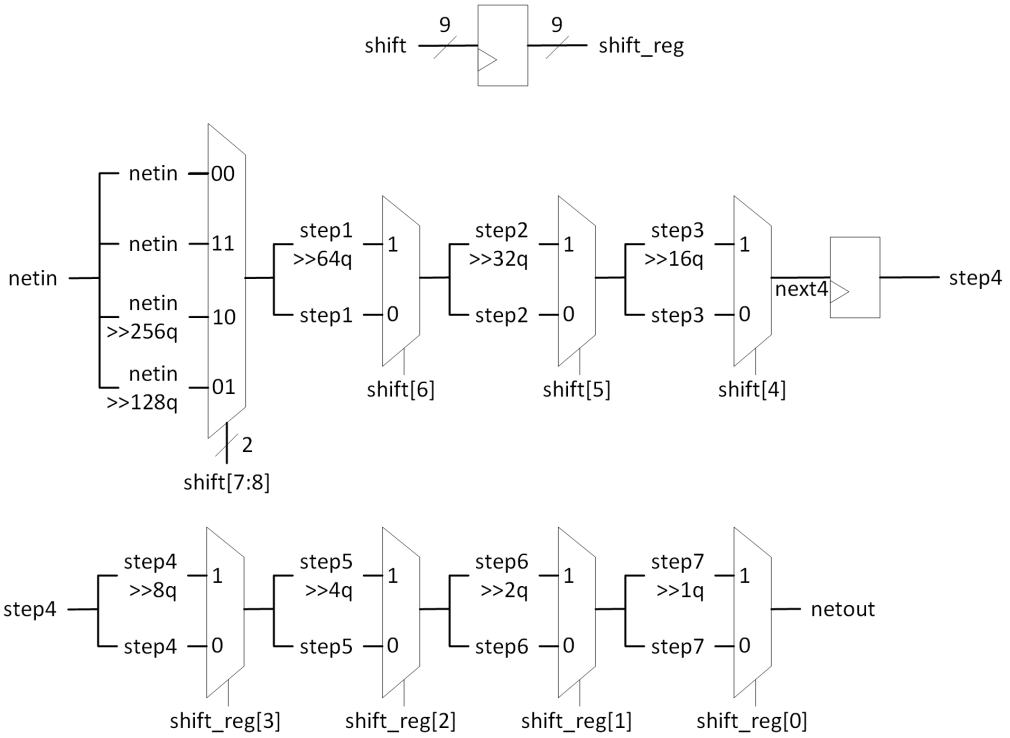

Figure 3.10: Barrel shifter diagram, by the author

3.3.8 Control unit

The control unit is based in the work of Calcanhotto [11], which follows the structure of the general LDPC algorithm and is easily adaptable to an algorithmic state machine.

It takes a total of six states, where three of them explicitly belong to the LDPC algorithm. In figure 3.11 it is presented the finite state machine representation of the control unit, without showing internal signals.

The states featured in the FSM are:

1. **IDLE**: Default state after a general reset or power on. Waits until a request of decoding arrives and starts the general execution.

2. **BUFF**: State where the LLRs are buffered in a FIFO, waits until a fixed number of inputs has been loaded.

3. **LOAD**: State where the buffered LLR are read and stored in memory bank.

4. **CHECK STEP**: State that executes the check node processing

5. **VAR STEP**: State that executes the variable node processing

6. **VALID**: State where it is syndrome vector is checked and determines if another iteration must be execute or the decoding is finished. If the decoding is finished, loads the output stream to a FIFO and return them serially.

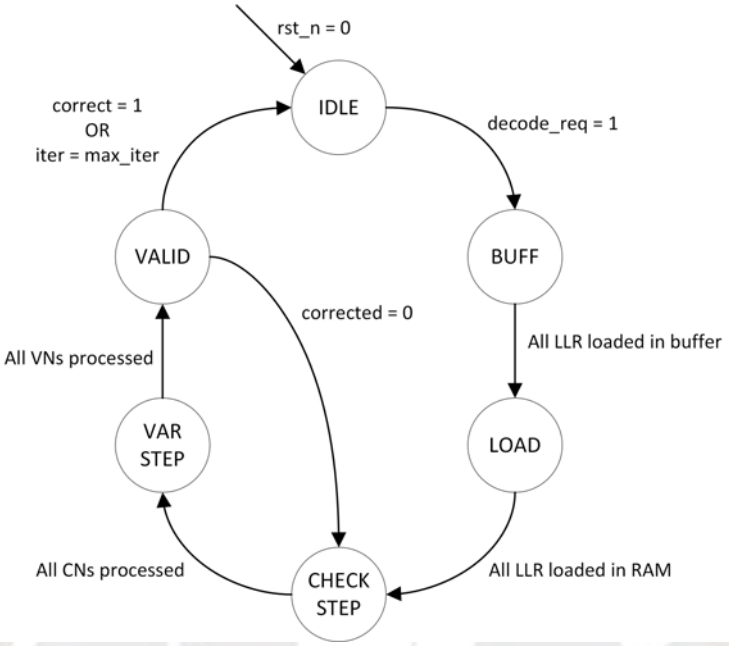

Figure 3.11: Finite State Machine (FSM) diagram, based on [11]

A list of the main I/O signals of the control unit is displayed in table 3.2. Additionally, a more detailed representation of the control unit can be seen in the algorithmic state machine diagram, in figure 3.12, where intermediate signals between FSM and datapath are shown.

Table 3.2: List of I/O signals of the ASM

Name of signal	**Description**
	Inputs
correct	Indicates if the syndrome vector is equal to zero and the bit stream is free of errors
	Outputs
fifo_load_wr	Enables reading of the input FIFO
fifo_load_rd	Enables writing of the input FIFO
fifo_unload_wr	Enables reading of the output FIFO
fifo_unload_rd	Enables writing of the output FIFO
ram_fifo_ena	Selects data of input FIFO as input of RAM to be stored
net_ena	Enables the network shifter, used only during the check node step
node_sel	Select between VNs (0) or CNs (0)

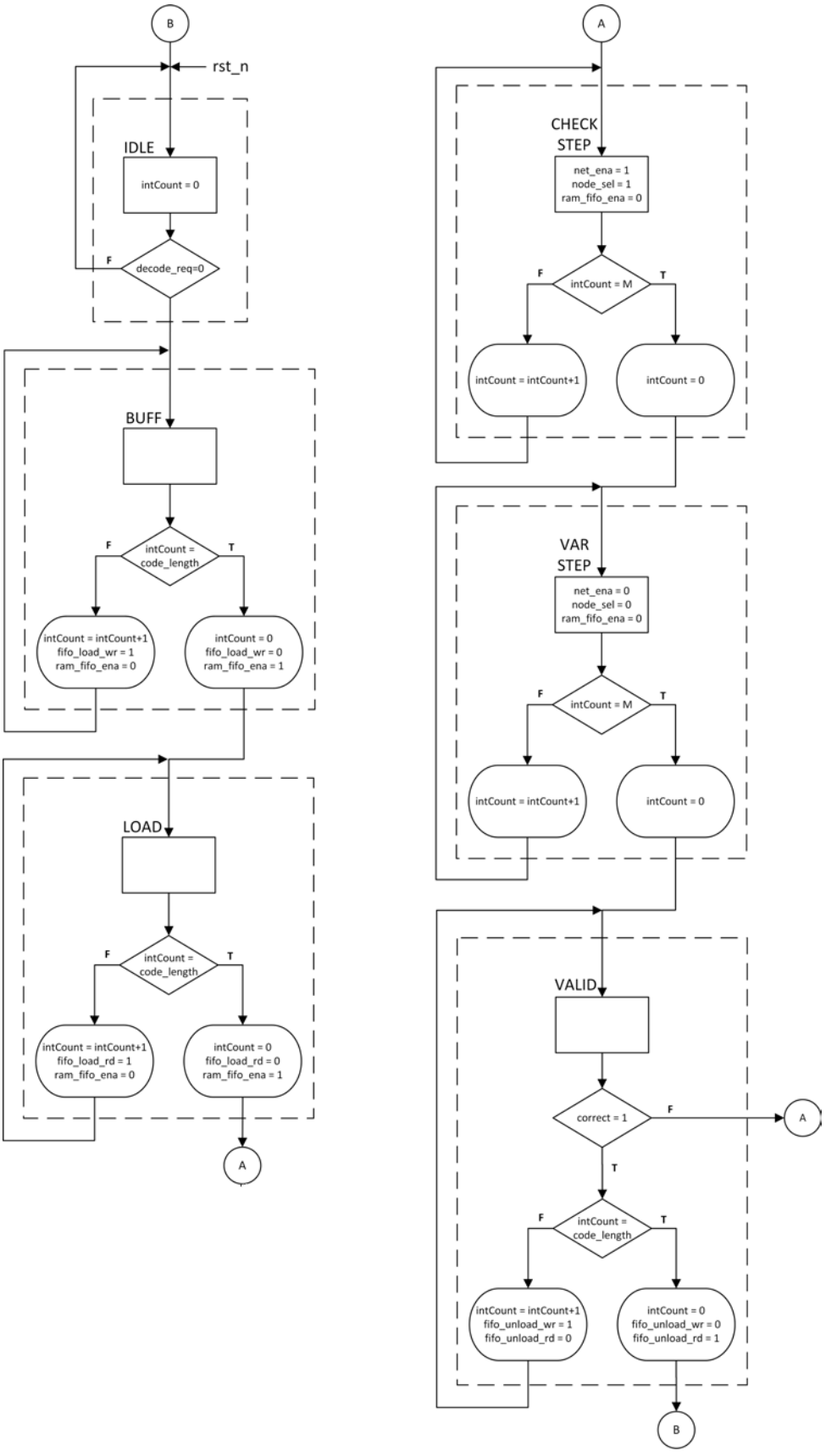

Figure 3.12: Algorithmic State Machine (ASM) diagram, by the author

3.4 Power saving modules

In this section the application of two techniques will be presented as an inclusion of the main core previously designed. Both techniques are incorporated with the objective of reducing power dissipation, each of them with different characteristics.

Firstly, the early stopping module is be covered, also explaining how affects the control and functional units. The same analysis is repeated for the forced convergence module, and how both methods effectively save power using different approaches.

3.4.1 Early stopping module

As it name indicates, the early stopping scheme stops the decoding before it finishes, that is, before a the bit sequence is completely corrected and the maximum number of iterations are reached. Under the assumption the demodulator has a static performance (i.e. it is set to a fixed SNR), this module determines when the SNR is too low to perform a successful decoding. At the same time, to avoid degrading performance in case of false positives, that is, when the stream hold a high SNR but is flagged as low SNR, the module does not execute the function of stopping current iterations.

From the work of Condo *et al*, this module takes two metrics and process them, following an algorithm described in [25]. With these parameters, extracted from node messages, it can be determined if the decoding will finish before the number of iterations exceed the configured maximum.

The first metric (SYN) is calculated from the syndrome vector and is defined as the sum of checksums (i.e the values of the syndrome), expressed as

$$SYN^{i+1} = \sum_{n}^{M} c_n, c_n = sign\left(Q_n^{i+1}\right) \tag{3.7}$$

The second metric (CNMM) is defined as the sum of the minimum c2v message returned by the CNs of the evaluated set, expressed as

$$CNMM^{i+1} = \sum_{m}^{M} \min_{n \in N(m)} \left(q_{nm}^{i}\right) \tag{3.8}$$

and is taken from the first minimum register in the check node (figure 3.12.

Both parameters are evaluated during the VALID state of the FSM, which now must include the algorithm defined in [25]. This algorithm depends in the comparison of SYN and CNMM with three thresholds, which are defined as

$$T1 = N * (1 - R) * 2^{-6}$$
$$T2 = N * (1 - R) * 2^{bits_f} \tag{3.9}$$
$$T3 = 2 * T1$$

where N is the code length, R is the code rate and $bits_f$ is the length of the fractional part of each LLR, in this case equal to 4.

Algorithm 3 Early stopping algorithm, from [25]

```
Enter VALID state
Activate IDD (Imposible Decoding Detection)
if correct = 1 then
    break
else
    if IDD active then
        if i ≥ 2 and (CNMM^i > T2 or SYN^i < T3) then
            Deactivate IDD
        else
            if CNMM^i < CNMM^(i-1) and SYN^i < SYN^(i-1) then
                CNT = CNT + 1
            else
                CNT = 0
            end if
        end if
        if CNT = I_esc then
            Stop decoding
        end if
        if i ≥ 0.6 * I_max and SYN^i > T1 then
            Stop decoding
        end if
    end if
end if
```

In figure 3.13, it is proposed a design for the early stopping module, where the same tree logic applied in the PCM module is repeated here, avoiding combinational components with large fan-in. Moreover, pipelining is applied introducing registers between each layer, taking into account that the number of steps is less than the number of clock cycles in each node processing, in standard operation.

In the top area (a), the SYN metric is calculated according to equation, and divided in two layers, considering 20 single-bit inputs of maximum fan-in. The CNMM metric is determined in bottom area (b), divided in two more layers than the design for SYN, because each input has a width of 5 bits, and the maximum fan-in is of 5 inputs.

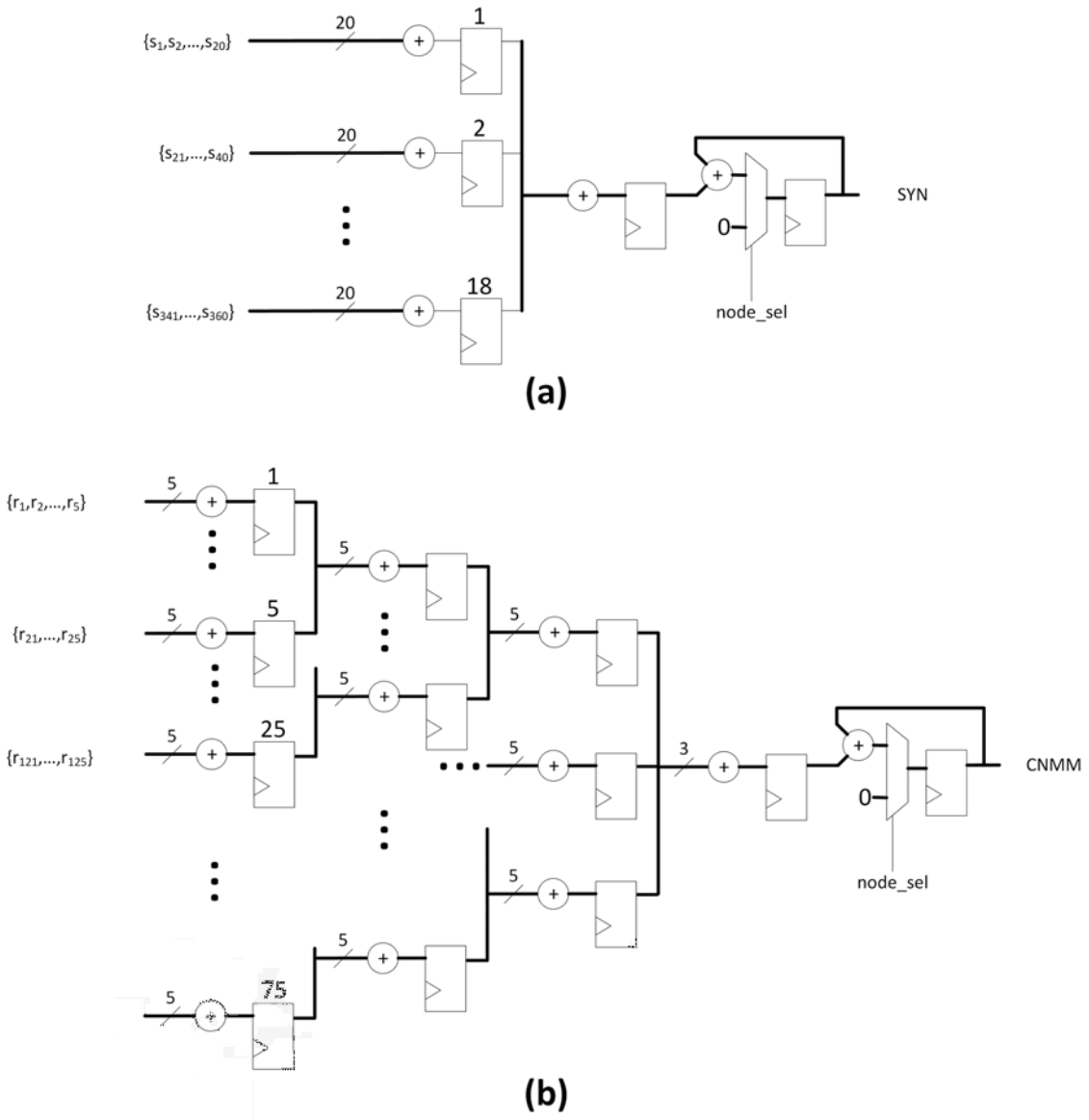

Figure 3.13: Early stopping module diagram, based on [25]

3.4.2 Forced convergence module

Forced convergence is a technique which deactivates a group of nodes, reducing the number of active nodes at a time, with the objective of reducing decoding complexity. This is based in the fact that a large number of VNs is quickly decoded, condition that is most satisfied by high SNR channels. When a VN has reached a high enough belief, its updating can be skipped. As consequence, the syndrome vector reaches a stable value (i.e converges to zero) faster than under normal operation.

How this technique reduce power dissipation is explained by less clock cycles in two scopes, which are the deactivated nodes (using clock gating) and fewer iterations for each input stream.

As described lines above, the belief magnitude of each VN is monitored if it surpasses a certain threshold [28], indicated in

$$\left|Q_n^{i+1}\right| > t_v \tag{3.10}$$

where t_v is extracted of simulations and suggested in [28] to be equal to 7.

The iteration when a VN is selected to deactivation, the v2c messages returned are saturated according to its sign, that is, the maximum possible magnitude is assigned. This is accomplish by appending a final stage to the output of the variable node, where the v2c message to be stored is selected according to the result of the comparison in equation 3.10. This message is stored in RAM and the VN is deactivated, so that position in RAM is not written again until end of decoding.

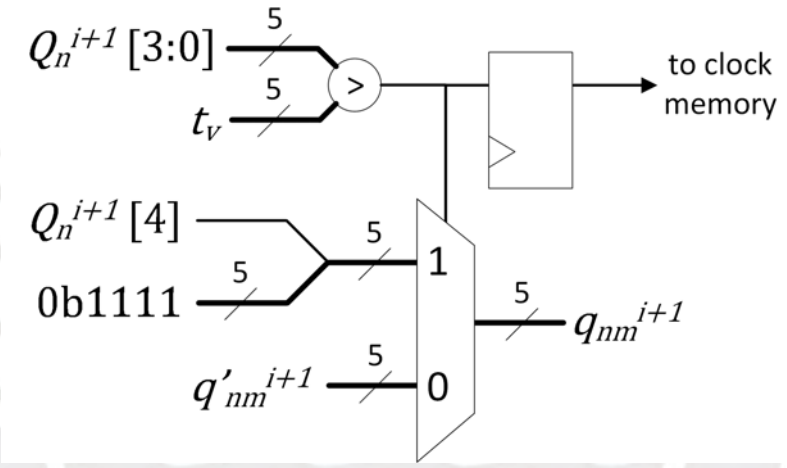

Figure 3.14: Variable node - output stage, by the author

The same result of the comparison is saved in a auxiliary memory and is read in next iterations to mark a variable node as active or not. Naturally, this has an impact in resource usage (1 bit register per VN).

Chapter 4

Verification and results

In this chapter, the applied methodology used for each module verification is presented, followed by the results of the synthesis and simulation of the proposed architecture. In the first and second section, the verification of the designed modules is covered, classified by the utilized methodology, direct verification in the first section and functional verification in the second. Next, the third section presents the results of the synthesis and compares its features with selected works. Finally, a performance and power analysis is displayed in the fourth and last section

4.1 Direct verification

This style of verification has been used for the individuals blocks of the design, because of the predictable operation and intrinsic simplicity of those. Most of them are of fully combinational logic and are verified by visual checking of the console report (for output accuracy) and the wave monitor (for timing analysis, also useful for synchronization of each block when integrated in the core). For executing the simulations, the software ISim Simulator from Xilinx has been used.

4.1.1 Barrel shifter

This combinational block has been tested with a static stimuli in the input port, equal to a known pattern, and a varying value in the shift input, which covers the whole set of cases, this being possible because of a reduced number of test cases (360 shift values). The output port was checked by visual inspection, where it was expected that the assigned pattern was located in a determined position.

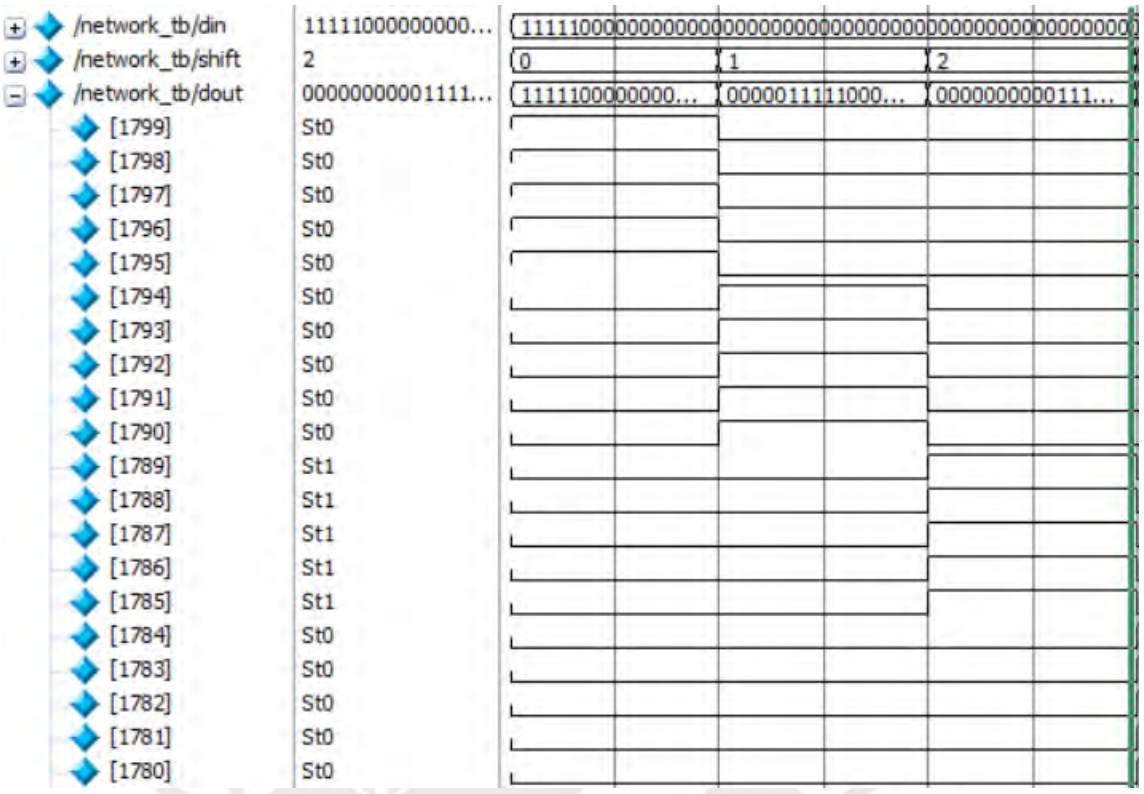

Figure 4.1: Wave monitor of the barrel shifter under verification, by the author

In figure 4.1, the pattern selected was a sequence of ones equal of width equal to the shift step of the barrel, followed by zeros. In the output port, it can be observed a ladder structure of ones that appears to be descending, that can be easily recognized as the assigned pattern being shifted by the shift value in the input.

4.1.2 Check node

For the CN unit, a series of random stimuli was applied in the variable to check node message (v2c) input, varying in quantity according to the different degrees (d_v) found in DVB-S2. The verification consisted in confirming the selection of the first and second minimum magnitude and the assignation of the correct sign for the outputs, as shown in the console report in 4.1, where 6 v2c messages are received after the reset signal. As shown in figure 4.2, it takes $2m + 3$ clock cycles for processing the c2v messages of m v2c inputs, including clock cycles for pipelining, provided there are not delays for reading/writing in memory.

Table 4.1: Console report of a CN operation cycle, by the author

```
=====Iteration [0]=====
Inputs
v2c 0: sign[0] dec[ 7] bin[00111]
v2c 1: sign[1] dec[12] bin[11100]
v2c 2: sign[1] dec[ 2] bin[10010]
v2c 3: sign[0] dec[10] bin[01010]
v2c 4: sign[1] dec[13] bin[11101]
v2c 5: sign[1] dec[12] bin[11100]

Outputs
c2v 0: duv[bin.00010 dec.  2] gm[          2]
c2v 1: duv[bin.11110 dec. -2] gm[         -2]
c2v 2: duv[bin.11001 dec. -7] gm[         -7]
c2v 3: duv[bin.00010 dec.  2] gm[          2]
c2v 4: duv[bin.11110 dec. -2] gm[         -2]
c2v 5: duv[bin.11110 dec. -2] gm[         -2]
-------------------------------
Testbench finished with success
```

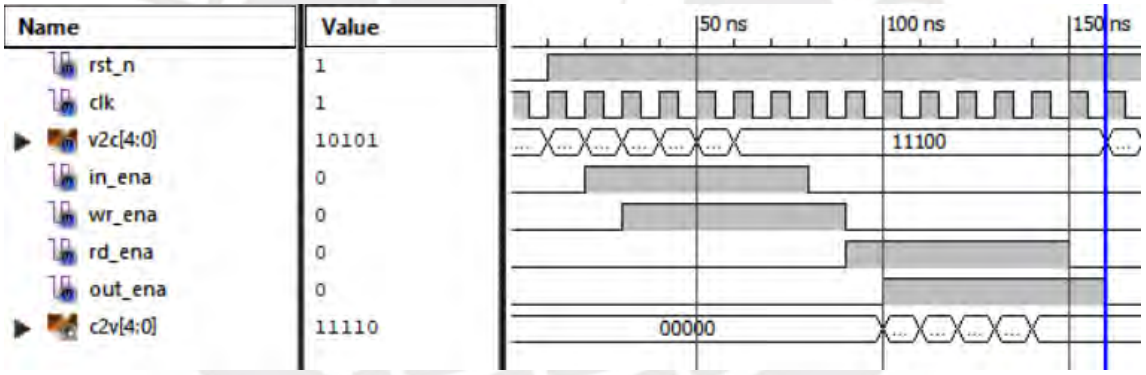

Figure 4.2: Wave monitor of a CN operation cycle, by the author

4.1.3 Variable node

Similar to the CN block, for the VN unit random stimuli were applied in the check to variable node message (v2c) input. It was tested that the correct total sum was calculated, the hard decoded sign was correctly propagated from the total sum, and the outputs were properly truncated if it was required, as explained in chapter 3. As shown in figure 4.3, it takes $2m + 3$ clock cycles for processing the c2v messages of m v2c inputs, including clock cycles for pipelining and the loading of the gamma value from the channel.

Table 4.2: Console report of a VN operation cycle, by the author

```
=====Iteration [0]=====
Inputs
c2v 0: dec[  8] bin[01000]
c2v 1: dec[  2] bin[00010]
c2v 2: dec[  4] bin[00100]
c2v 3: dec[  6] bin[00110]
c2v 4: dec[-15] bin[10001]
c2v 5: dec[-13] bin[10011]

Final sum Q dec[   7] bin[000000111]
Sign of Q [0]

Outputs
v2c 0: duv[bin.10001 sign.1 mag. 1] gm[sign.1 mag.15]
v2c 1: duv[bin.00101 sign.0 mag. 5] gm[sign.0 mag. 5]
v2c 2: duv[bin.00011 sign.0 mag. 3] gm[sign.0 mag. 3]
v2c 3: duv[bin.00001 sign.0 mag. 1] gm[sign.0 mag. 1]
v2c 4: duv[bin.01111 sign.0 mag.15] gm[sign.0 mag.15]
v2c 5: duv[bin.01111 sign.0 mag.15] gm[sign.0 mag.15]
------------------------------
Testbench finished with success
```

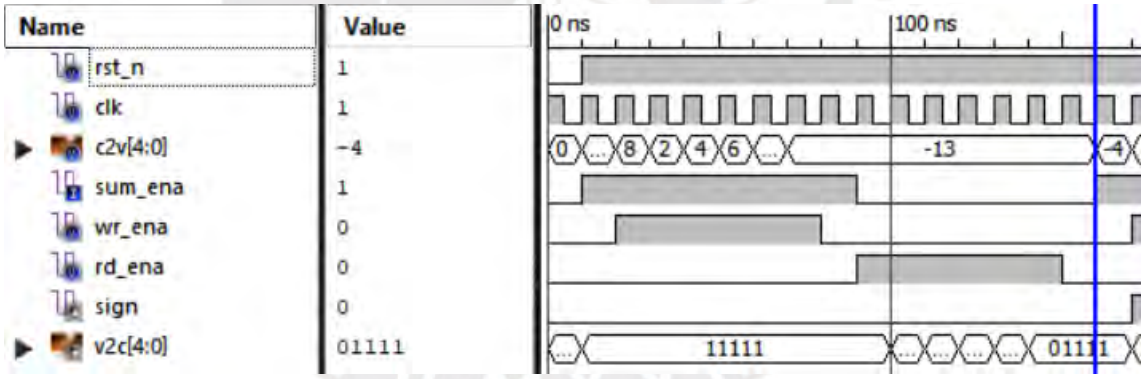

Figure 4.3: Wave monitor of a VN operation cycle, by the author

4.2 Functional verification of the decoder

In contrast with individual components, the decoder core presents a higher level of complexity and a larger number of stimuli combinations, making necessary to apply a functional methodology in replace of the direct testing approach. In figure 4.4 is displayed the block diagram of the testing environment, where it is defined 4 parts:

- Random stimuli generator: a random generator of LLR values, fed to the designed

architecture by the input serial port.

- Golden model: a implementation in software MATLAB that imitates the behavior of the design in hardware, so its output is the same as the expected output of the core in hardware (including possible decoding errors).
- DUV: Design under verification: the proposed implementation of the LDPC decoder in FPGA.
- Checker: It compares the output values returned by the golden model and the DUV and checks if are equal or not.

To simplify the development of the verification environment, the checker has been reduced to a file analyzer. Both the outputs of the golden model and the DUV are saved in a text file and the checker only returns if both files are equal or not.

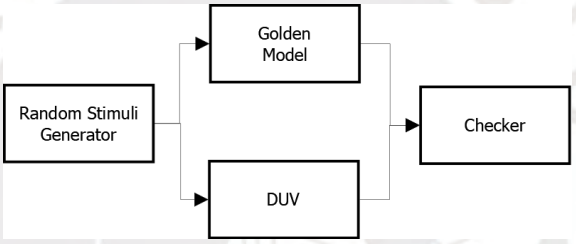

Figure 4.4: Scheme of verification environment

Furthermore, a simulation of the decoder has been executed to verify visually the correct synchronization of each component, reflected in figure 4.5, where each it can be observed the loading and unloading of the input frame and the corrected output frame, respectively. Also, the state changes of the control unit can be observed, being evident a bottleneck in the process identified in data loading and unloading.

4.3 Synthesis results

The synthesis of the LDPC decoder was made using ISE Design Suite from Xilinx. For the analysis, it is considered the resource usage, divided in slices LUT and registers, as well as block RAMs. Additionally, the maximum frequency of operation is taken from the results of the Timing report from the same software. Other parameters include the effective throughput and latency per

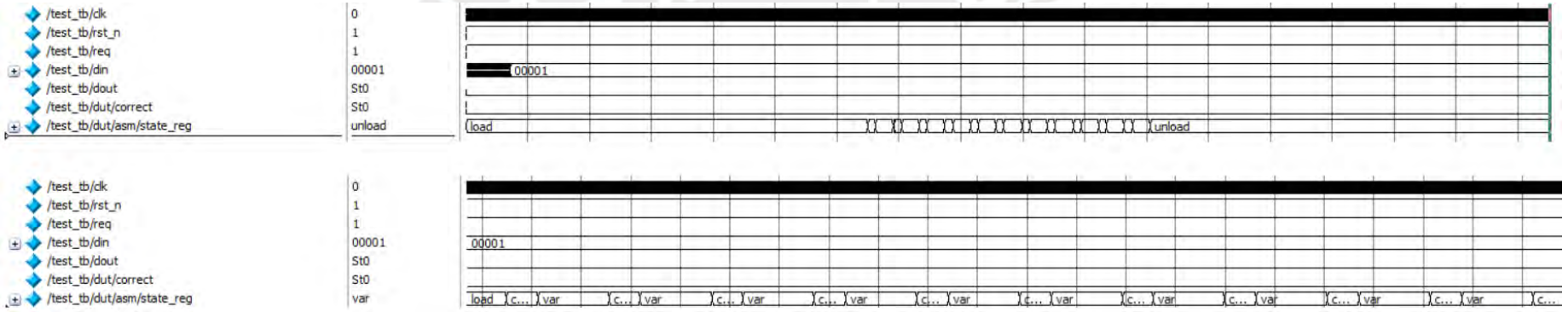

Figure 4.5: Monitor wave of a decoding process

frame, calculated using equations

$$\begin{aligned} \text{throughput} &= f_{max} * \frac{1}{(\text{number of clock cycles between two frames})} * (\text{frame length}) \\ \text{latency} &= \frac{1}{f_{max}} * (\text{number of clock cycles per frame}) \end{aligned} \tag{4.1}$$

However, most works related to DVB-S2 decoder design report throughput in function of time spend only in decoding, not counting input/output of the frame, which is scope of the interface. In this work, "throughput" will refer to this metric, expressed by the equation

$$\text{throughput} = f_{max} * \frac{1}{(\text{number of clock cycles for decoding one frame})} * (\text{frame length}) \tag{4.2}$$

and will be used in the comparison with state of art. As well as that, the first throughput mentioned will be referred as "effective throughput".

Considering the frame length of 64800 bits, a code rate of 1/2 and a rate of one cycle per bit received/send, the input frame loading into the decoder takes 64800 cycles, 2400 cycles per iteration during decoder (24000 cycles in total) and 32400 cycles for unloading the output frame (of half the length), for a total of 121200 cycles for latency. The decoder supports the unloading of the current frame concurrently with the loading of the next frame, so the effective throughput is based in 88800 cycles per frame. On the other hand, the theoretical throughput is calculated considering only 24000 cycles for the decoding.

In table 4.3, a comparison is made between this proposal and the works of Patel *et al.* [1] and Calcanhotto *et al.* [11]. The maximum frequency operation is 194.188 Mhz, slightly less than the other two works. However, compared with [1] and [11], a reduction of 10% and 24% in resource usage (Slice LUT) has been reached, respectively.

Table 4.3: Comparison of features of LDPC decoders

Ref.	This work	Patel *et al.* **[1]**	Calcanhotto *et al.* **[11]**
Code length	64800	1152	64800
Code rate	1/2	1/2	1/2
Throughput (Mbps)	518.907	2107	339.286
Freq. operation (Mhz)	194.188	219.443	100.058
Resource usage			
Slice LUT	60363	67317	79524
Slice registers	23552	27182	-
Flip-flops	-	-	45183
Block RAM	243	-	405

In function of the maximum frequency of operation, the worst-case throughput and latency are

$$\begin{aligned} \text{effective throughput} &= 194.188\ \text{Mhz} * \frac{1}{88800} * 64800\ \text{bits} = 142.99\ \text{Mbps} \\ \text{latency} &= \frac{1}{194.188\ \text{Mhz}} * 121200 = 0.624\ \text{ms} \end{aligned} \tag{4.3}$$

It is necessary to take into account the significant delay introduced by the data uploading by serial port, where it is spend 70% of the clock cycles considered for effective throughput. The average case would have to consider the effect of the early stopping module and a metric to weigh each SNR case.

4.4 Performance analysis

The proposed design was evaluated to compare the efficacy of the decoder in correcting errors at different SNR. It was selected the evalation method from [1], taking point samples from 1 to 5 dB with steps of 1 dB. For each point, a total of 100 random input frames were tested.

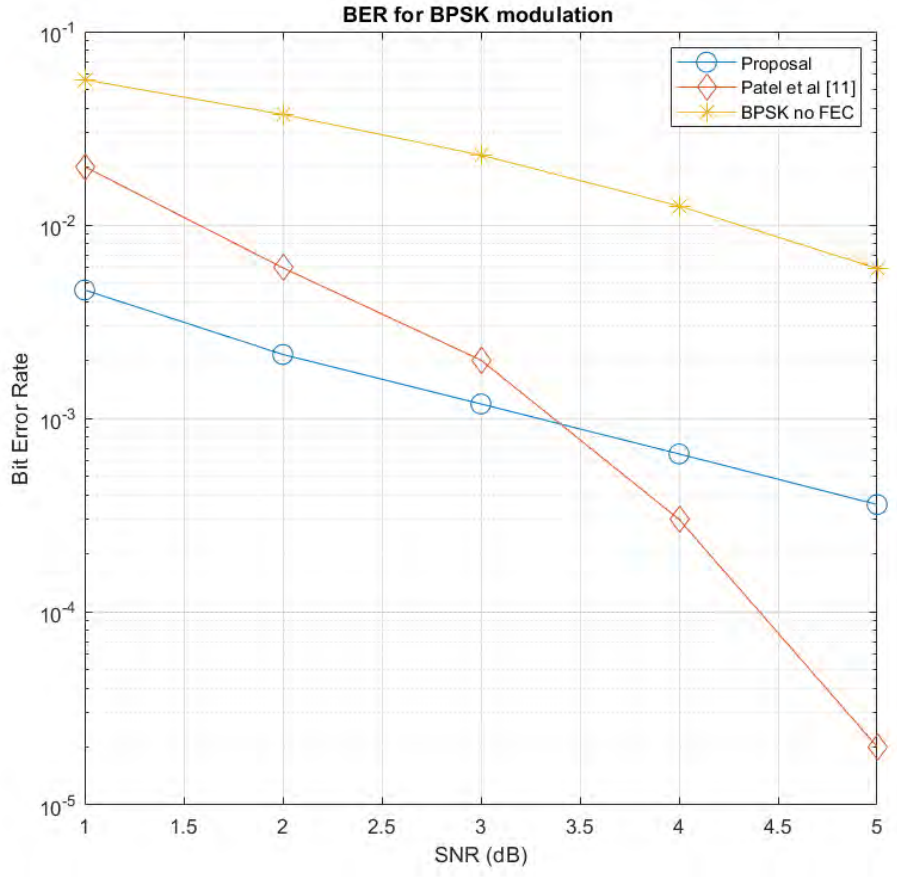

Figure 4.6: Bit Error Rate plot

In figure 4.6, the results of the testbench are displayed and a comparison is made with the work of Patel *et al.* [1] .It can be observed a better performance in the SNR region from 1 to 3 dB, with a difference of 1 dB above 10^{-3} BER. This effect must be caused by the difference between code lengths between both designs, where a large code length has the intrinsic advantage of having more robustness. However, the decoder has a poor performance above SNR of 4 dB. with a little-steep slope, which suggest a high error floor.

A consequence of this is the limitation of the possible applications where it can be applied. For example, it would be non-viable for satellite optical communications, where BER as low as 10^{-11} is mandatory [12], although it is competitive for low SNR, low throughput video satellite uplinks. Moreover, this result must be considered in the larger scheme of DVB-S2, where the inner LDPC core is supported by a outer BCH decoder and an interleaver, in order to achieve those levels of performance.

4.5 Power analysis

Finally, the designed decoder has been evaluated with the XPower Analyzer tool by Xilinx, whose results for power dissipation are shown in table 4.4.

Table 4.4: Power analysis results

Type	**Power (W)**
Clock	0.184
Logic	0.465
Signals	0.980
BRAMs	0.862
Static leakage	0.132
Total	2.623

From the total dissipated power, it is calculated the energy per bit rate, where at worst case is equal to

$$\text{energy per bit } = \frac{\text{power} * 1\text{s}}{\text{throughput} * 1\text{s}} = \frac{2.623 \text{ W}}{142.99 \text{ Mbps}} = 18.344 \text{ nJ/bit} \tag{4.4}$$

At explained before, in the synthesis results, it is the uploading and downloading stage which take most of processing time, in contrast with each decoding iteration (64800 cycle for uploading - one cycle per bit, 2400 cycles per iteration during decoding). Disabling most of the circuit slightly decrease power dissipation at cost of a increased use of logic elements, concluding deactivation

of nodes have a minimal effect in saving power at a non-negligible logic element cost. The early stopping scheme have a more prevalent outcome, even if the number of clock cycles for decoding is still overwhelmed.

Conclusions

- A LDPC decoder compliant with standard DVB-S2 has been designed, covering the inner stage of the DVB-S2 decoder, which includes the design of the functional units, a interconnection network between both types of units and an ASM for the control unit. It has been included a memory scheme that takes advantage of the optimization from the standard.
- Both functional units corresponding to the variable and check nodes have been developed with a low resource usage and no downtime during node processing.
- Pipelining has been applied large combinational paths, p, particularly in the parity check module, taking into consideration not to impact the timing of the rest of the circuit.
- A simulation in software MATLAB, which replicates the behavior of the decoder in hardware, has been developed with the resulting selection of parameters for quantization applied in the hardware design and the performance analysis of the min-sum algorithm once implemented.
- It has been achieved the prototyping of a LDPC decoder with a latency of 0.624 ms and a throughput of 142.99 Mbps, with a lower resource usage than other state-of-art works [1][12] and a similar performance [1].
- Specialized modules have been added to the original design in standard operation, aiming to reduce power dissipation. The early stopping scheme is the block with the highest impact in achieving this. Most power is not dissipated during decoding, as explained in Chapter 4.
- A power analysis has been executed with a result of 18.3 nJ/bit for worst operation case, not taking into account the expected reduction of iterations by early termination.
- A simulation of the designed architecture has been made for verification and its has been compared with the results of the previously developed golden model for validation.

Recommendations

- An alternative interface can be integrated in the core to avoid the significant delay introduced by the uploading and downloading of data by serial ports, eliminating the bottleneck provoked in those states.
- It is possible to adapt the presented core to a multi-symbol demodulation domain by converting the functional units into a Galois Field of 2^m, enabling its incorporation into more recent applications, such as 5G technology.
- It is recommended to implement this design in a physical device, with the objective of verify the obtained results in the simulation software ISim Simulator. Furthermore, an appropriate serial interface should be added for input and output frames, to approximate more closely a real-time operation.
- The design can be modified to accept different design parameters, for example, number of bits for quantization, with the objective of having a more flexible design, adaptable to more demanding applications not limited by resource or timing constrains.
- It is suggested to incorporate different strategies at physical level to achieve a reduction of dissipated power, such as unreliable memory or gear shifting [5], to complement the logic strategies applied.

Bibliography

[1] D. J. Patel and P. Engineer, "Design and implementation of quasi cyclic low density parity check (QC-LDPC) code on FPGA," in *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 181–185, March 2017.

[2] M. Borda, *Fundamentals in Information Theory and Coding*. Berlin Heidelberg: Springer-Verlag, 2011.

[3] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, Jan 1962.

[4] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronic Letters*, vol. 32, no. 18, 1996.

[5] J. Andrade, *Design Space Exploration of LDPC Decoders on Programmable and Reconfigurable Architectures*. PhD thesis, University of Coimbra, september 2015.

[6] G. Falcão, *Parallel Algorithms and Architectures for LDPC Decoding*. PhD thesis, University of Coimbra, july 2010.

[7] C. P. Carrasco, "Las tecnologías de la información y comunicaciones (TIC) y la brecha digital: su impacto en la sociedad del conocimiento del Perú ," in *Quipukamayoc*, vol. 15, pp. 65–74, Universidad Nacional Mayor de San Marcos, 2008.

[8] Grupo de Telecomunicaciones Rurales, *Redes Inalámbricas para Zonas Rurales*. Pontificia Universidad Católica del Perú, 2 ed., february 2011.

[9] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Survey of FPGA-Based LDPC Decoders," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 1098–1122, Secondquarter 2016.

[10] J. Andrade, N. George, K. Karras, D. Novo, F. Pratas, L. Sousa, P. Ienne, G. Falcao, and V. Silva, "Design Space Exploration of LDPC Decoders Using High-Level Synthesis," *IEEE Access*, vol. 5, pp. 14600–14615, 2017.

[11] A. Calcanhotto, "LDPC core for DVB-S2 standard," Master's thesis, Pontificia Universidad Católica de Río Grande del Sur, 2016.

[12] V. Pignoly, B. Le Gal, C. Jego, and B. Gadat, "High data rate and flexible hardware QC-LDPC decoder for satellite optical communications," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, pp. 1–5, Dec 2018.

[13] Y. Delomier, B. Le Gal, J. Crenne, and C. Jego, "From multicore LDPC decoder implementations to FPGA decoder architectures: a case study," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 89–92, Dec 2018.

[14] Min Hyuk Kim, Tae Doo Park, Chul Seong Kim and Ji Won Jung, "An FPGA design of low power LDPC decoder for high-speed wireless LAN," in *2010 IEEE 12th International Conference on Communication Technology*, pp. 1460–1463, Nov 2010.

[15] M. A. Sayed, Liu Rongke, and Zhao Ling, "Design and implementation of scalable throughput fast convergence LDPC decoder," in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Oct 2016.

[16] 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)*, pp. 1–5, Dec 2018.

[17] C. Berrou, *Codes and Turbo Codes*. Paris New York: Springer-Verlag Paris, 2010.

[18] J. Moreira, *Essentials of error-control coding*. West Sussex, England: John Wiley & Sons, 2006.

[19] Y. Jiang, *A practical guide to error-control coding using Matlab*. Boston: Artech House, 2010.

[20] X. Zhang, *VLSI architectures for modern error-correcting codes*. Boca Raton, FL: CRC Press, 2015.

[21] P. Schläfer, C. Weis, N. Wehn, and M. Alles, "Design Space of Flexible Multigigabit LDPC Decoders," *VLSI Des.*, vol. 2012, pp. 4:4–4:4, Jan. 2012.

[22] P. P. Chu, *RTL hardware design using VHDL: coding for efficiency, portability, and scalability*. John Wiley & Sons, 2006.

[23] D. Green, *Modern Logic Design*. Electronic systems engineering series, Addison-Wesley, 1986.

[24] T. Mohsenin, H. Shirani-mehr, and B. Baas, "Low power LDPC decoder with efficient stopping scheme for undecodable blocks," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pp. 1780–1783, May 2011.

[25] G. M. C. Condo, A. Baghdadi, "Energy-efficient multi-standard early stopping criterion for low-density-parity-check iterative decoding," *IET Communications*, vol. 8, p. 2171–2180, 2014.

[26] J. Sodha, "Early stopping criterion for LDPC," in *2017 International Conference on Circuits, System and Simulation (ICCSS)*, pp. 134–137, July 2017.

[27] B. J. Choi and M. H. Sunwoo, "Efficient Forced Convergence algorithm for low power LDPC decoders," in *2013 International SoC Design Conference (ISOCC)*, pp. 372–373, Nov 2013.

[28] B. J. Choi and M. H. Sunwoo, "Simplified forced convergence decoding algorithm for low power LDPC decoders," in *2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 663–666, Nov 2014.

[29] European Telecommunications Standards Institute, "Digital Video Broadcasting (DVB) Implementation guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications," tech. rep., ETSI, 2015.

[30] J. Andrade, G. Falcao, V. Silva, J. P. Barreto, N. Goncalves, and V. Savin, "Near-LSPA performance at MSA complexity," in *2013 IEEE International Conference on Communications (ICC)*, pp. 3281–3285, June 2013.

[31] S. Scholl, P. Schläfer, and N. Wehn, "Saturated Min-sum Decoding: An "Afterburner" for LDPC Decoder Hardware," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, (San Jose, CA, USA), pp. 1219–1224, EDA Consortium, 2016.

[32] R. Purnamasari, H. Wijanto, and I. Hidayat, "Design and implementation of LDPC (Low Density Parity Check) coding technique on FPGA (Field Programmable Gate Array) for

DVB-S2 (Digital Video Broadcasting-Satellite)," in *2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology*, pp. 83–88, Nov 2014.

[33] M. Y. Zinchenko, A. M. Levadniy, and Y. A. Grebenko, "Development of the LDPC coder-decoder of the DVB-S2 standard on FPGA," in *2018 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, pp. 1–3, July 2018.

[34] F. Plasencia, "Functional verification framework of an AES encryption module," B.S. Thesis, Pontificia Universidad Católica del Perú, 2017.

[35] F. Kienle, T. Brack, and N. Wehn, "A synthesizable IP core for DVB-S2 LDPC code decoding," in *Design, Automation and Test in Europe*, pp. 100–105, IEEE, 2005.

[36] M. Eroz, Feng-Wen Sun, and Lin-Nan Lee, "An innovative low-density parity-check code design with near-Shannon-limit performance and simple implementation," *IEEE Transactions on Communications*, vol. 54, pp. 13–17, Jan 2006.

[37] G. Falcao, M. Gomes, V. Silva, L. Sousa, and J. Cacheira, "Configurable M-factor VLSI DVB-S2 LDPC decoder architecture with optimized memory tiling design," *EURASIP Journal on wireless communications and networking*, vol. 2012, no. 1, p. 98, 2012.