Implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards

Cédric Marchand

THESE / UNIVERSITE DE BRETAGNE SUD
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE DE BRETAGNE SUD
Mention : STIC
Ecole doctorale SICMA

présenté par
Cédric Marchand
Laboratoire : Lab-STICC

Etude et implementation d'un décodeur LDPC pour les nouvelles normes de diffusion de television numérique (DVB-T2 et S2)

Thèse soutenue le 10 Janvier 2010
devant le jury composé de :

Jean-François Helard
Professeur des universités à l'INSA de Rennes / président

Christophe Jego
Professeur des universités à l'institut Polytechnique de Bordeaux / *rapporteur*

Renaut Pacalet
Directeur d'étude à Télécom Paris Tech / *rapporteur*

Diddier Lohy
Responsable innovation à NXP Caen / examinateur

Laura Conde-Canencia
Maitre de conférence à l'Université de Bretagne Sud / *Co-directeur de thèse*

Emmanuel Boutillon
Professeur des universités à l'Université de Bretagne Sud / Directeur de thèse

"If we knew what it was we were doing, it would not be called research, would it?"

Albert Einstein (1879-1955)

Remerciements

Je tiens tout d'abord à remercier Emmanuel Boutillon pour m'avoir encadré avec sérieux et clairvoyance. Je remercie aussi Laura Conde-Canencia pour son aide lors de la rédaction des différents articles et de la thèse.

Je remercie tous mes encadrants qui se sont succédés dans un contexte de plan social et de restructuration NXP: Patrick Lecluse initiateur de la thèse CIFFRE, Jean-Baptiste DORE qui m'a été d'une grande aide en début de thèse, puis Pascal Frederic et enfin Didier Lohy.

Je remercie mes collègues de travail au sein du group NXP: Arnaud, Pierre-Jean, Zair, Carol et Franck.

Je n'oublie pas mes collègues à l'Univerité de Bretagne Sud dont Sébastien, Antoine et Jérémie qui finissaient leur thèse et dont j'ai pu bénéficier des conseils avisés; Gizlaine et Rachid, collègues de promotion et amis qui ont commencés leur Master recherche et leur thèse en même temps que moi; Gorgiano mon voisin d'en face; Aswhani et sa bonne humeur, le personnel administratif et le personnel du restaurant universitaire.

Enfin, je remercie ma famille pour son soutient et Monika pour son support quotidien et ses bons petits plats.

Résumé

Les codes correcteurs d'erreurs LDPC ("Low Density Parity Check" ou matrice de parité à faible densité) font partie des codes en bloc perméttant de s'approcher à quelques dixième de dB de la limite de Shannon. Ces remarquables performances associées à leur relative simplicité de décodage rendent ces codes très attractifs pour les systèmes de transmissions numériques. C'est notamment le cas pour la norme de télédiffusion numérique par satellite (DVB-S2) et la norme de télédiffusion numérique terrestre (DVB-T2) qui utilisent un code LDPC irrégulier pour la protection de la transmission des données. Cette thèse porte sur l'optimisation de l'implémentation matérielle d'un décodeur LDPC pour les standards DVB-S2, -T2 et -C2. Après une étude de l'état de l'art, c'est le décodeur par couche (layered decoder) qui a été choisi comme architecture de base à l'implémentation du décodeur. Nous nous sommes ensuite confrontés au problème des conflits mémoires inhérents à la structure particulière des standards DVB-S2, -T2 et -C2. Deux nouvelles contributions ont été apportées à la résolution de ce problème. Une basée sur la constitution d'une matrice équivalente et l'autre basée sur la répétition de couches (layers). Les conflits mémoire dues au pipeline sont quant à eux suprimés à l'aide d'un ordonnancement des layers et des matrices identités. L'espace mémoire étant un différenciateur majeur de coût d'implémentation, la réduction au minimum de la taille mémoire a été étudiée. Une saturation optimisée et un partitionnement optimal des bancs mémoires ont permis une réduction significative de l'espace mémoire par rapport à l'état de l'art. De plus, l'utilisation de RAM simple port à la place de RAM double port est aussi proposé pour réduire le coût mémoire. En dernière partie, nous répondons à l'objectif d'un décodeur capable de décoder plusieurs flux pour un coût réduit par rapport á l'utilisation de multiples décodeurs.

Mot-clés: Codes LDPC, Implémentation, Conflits mémoire, DVB-S2

Abstract

LDPC codes are, like turbo-codes, able to achieve decoding performance close to the Shannon limit. The performance associated with relatively easy implementation makes this solution very attractive to the digital communication systems. This is the case for the Digital video broadcasting by satellite in the DVB-S2 standard that was the first standard including an LDPC.

This thesis subject is about the optimization of the implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards. After a state-of-the-art overview, the layered decoder is chosen as the basis architecture for the decoder implementation. We had to deal with the memory conflicts due to the matrix structure specific to the DVB-S2, -T2, -C2 standards. Two new contributions have been studied to solve the problem. The first is based on the construction of an equivalent matrix and the other relies on the repetition of layers. The conflicts inherent to the pipelined architecture are solved by an efficient scheduling found with the help of graph theories.

Memory size is a major point in term of area and consumption, therefore the reduction to a minimum of this memory is studied. A well defined saturation and an optimum partitioning of memory bank lead to a significant reduction compared to the state-of-the-art. Moreover, the use of single port RAM instead of dual port RAM is studied to reduce memory cost.

In the last chapter we answer to the need of a decoder able to decode in parallel x streams with a reduced cost compared to the use of x decoders.

Keywords: LDPC codes, implementation, memory conflicts, DVB-S2

Contents

Notations

LDPC codes notations:

x	: Encoder input of length $K : x = (x1, , xK)$.
c	: Encoder output (sent codewords).
C	: A channel code, i.e. the set of all the codewords: $c \in C$.
y	: Decoder input (received codewords).
K	: Number of information bits.
N	: Number of variables.
M	: Number of parity checks : $M = N - K$.
R	: Rate of the code $C : R = K/N$.
H	: A parity check matrix of size $(N - K)N$ of the code C.
d_c	: Maximum weight of the parity checks.
$M_{c \to v}$	: Check to variable message.
$M_{v \to c}$	: Variable to check mesage.

Mathematical expressions

$(.)^t$	: Stands for transposition of vectors.
$\lvert . \rvert$	: Stands for the cardinality of a set, or for the magnitude of a real value.
$\lceil . \rceil$	: Ceil operator.
$\lfloor . \rfloor$	: Floor operator.

Abbreviations

APP	: A-Posteriori Probability
ASIC	: Application Specific Integrated Circuit
AWGN	: Additive White Gaussian Noise
BCH	: Bose and Ray-Chaudhuri
BER	: Bit Error Rate
BPSK	: Binary Phase-Shift Keying
CMMB	: China Multimedia Mobile Broadcasting
CN	: Check Node
CNP	: Check Node Processor
DDSM	: Double Diagonal Sub-Matrix
DPRAM	: Dual Port RAM
DSNG	: Digital Satellite News Gathering
DTV	: Digital TeleVision
DVB	: Digital Video Broadcasting
DVB-S2	: Second generation DVB System for Satellite broadcasting and unicasting
DVB-T2	: ... Terrestrial ...
DVB-C2	: .. Cable ...
DVB-X2	: DVB-S2, -T2 and C2 standards

FEC	: Forward Error Correction
FER	: Frame Error Rate
FPGA	: Field Programmable Gate Array
GA	: Genetic Algorithm
HDCE	: Hardware Discrete Channel Emulator
HDTV	: High-Definition TeleVision
IM	: Identity Matrix
IP	: Intellectual property
IVN	Information Variable Node
LDPC	: Low Density Parity Check
LSB	: Less Significant Bit
LLR	: Log-Likelihood Ratio
MSB	: Most Significant Bit
MPEG	: Moving Picture Experts Group
NMS	: Normalized Min-Sum
OMS	: Offset Min-Sum
PVN	: Parity Variable Node
QPSK	: Quadrature Phase Shift Keying
RAM	: Random Access Memory
SNR	: Signal-to-Noise Ratio
SO	: Soft Output
SPRAM	: Single Port RAM
TV	: TeleVision
TNT	: Télévision Numérique Terrestre
TSP	: Traveling Salesman Problem
VN	: Variable Node

Introduction

In the early 90's, C. Berrou and A. Glavieux proposed a new scheme for channel decoding: the turbo codes. Turbo codes made possible to get close to the Shannon limit as never before.

Besides the major impact that the turbo codes have had on telecommunication systems, they also made researchers realize that iterative process existed. Hence Low-Density Parity-Check (LDPC) codes invented in the early sixties by Robert Gallager, have been resurrected in the mid nineties by David MacKay. In fact, the computing resources at that time were not powerful enough to exploit LDPC decoding, and LDPC codes have been forgotten for some decades. Thanks to exponential computing power capability (Moore law), nowadays LDPC decoding can be implemented with a reduced cost. Among all the published works on LDPC, the approach introduced in [5] led to the conception of structured codes which are now included in standards. Among the existing standards, we can distinguish standards using short frames (648, 1296 and 1944 bits for Wi-Fi) and standards using long frames (64800 bits for DVB-S2). The use of long frames makes it possible to get closer to the Shannon limit, but leads to delays that are not suitable for internet protocols or mobile phone communications. On the other hand, long frames are suitable for streaming or Digital Video Broadcasting (DVB). The 2^{nd} Generation Satellite Digital Video Broadcast (DVB-S2) standard ratified in 2005 was the first standard including an LDPC code as forward error corrector. The 2^{nd} Generation Terrestrial DVB (DVB-T2) standard was adopted in 2009 and the 2^{nd} Generation Cable DVB (DVB-C2) was adopted in 2010.

These three DVB standards include a common Forward Error Correction (FEC) block. The FEC is composed of a BCH codec and an LDPC codec. The FEC supports eleven code rates for the DVB-S2 standard and is reduced to six code rates for the DVB-T2 standards. The LDPC codes defined by the DVB-S2,-T2,-C2 standards are structured codes or architecture-aware codes (AA-LDPC [42]) and they can be efficiently implemented using the layered decoder architecture [10, 53] and [31]. The layered decoder benefits from three architecture improvements: parallelism of structured codes, turbo message passing, and Soft-Output (SO) based Node Processor (NP) [10, 53] and [31]. Even if the state-of-the-art of the decoder architecture converges to the layered decoder solution, the search of an efficient trade-off between area, cost, low consumption, high throughput and high performance makes the implementation of the LDPC decoder still a challenge. Furthermore, the designer has to deal with many possible choices of algorithms, parallelisms, quantization parameters, code rates and frame lengths. In this thesis, we

study the optimization of a layered LDPC decoder for the DVB-S2, -T2 and -C2 standards. We consider the DVB-S2 standard to compare results with the literature but our work can also be applied to the Wi-Fi and WiMAX LDPC standards or more generally, to any layered LDPC decoder. This thesis is organized as follows:

The first chapter gives the notation and background required for the understanding of the thesis. The basic concepts of digital communication are first introduced. We remind appropriate basic elements of a digital communication and highlight the channel decoder. Then the LDPC decoder is presented with the belief propagation algorithm. Sub-optimal algorithms are described, more specially the Min-Sum algorithm and related algorithms. After discussing on layered decoder principle and advantages, an architecture overview is given. The DVB-S2, -T2 and C2 standards are presented and more precisely the matrix construction followed by a state-of-the-art of implementations of LDPC decoders for these standards. In order to compare the efficiency of different algorithms and quantization options, we discuss the testing environment used for our simulations and implementations.

The second chapter is dedicated to the resolution of the memory update conflicts. Two kind of memory update conflicts are identified and solved separately. The conflicts due to the structures of the matrices are described and solved using two innovative solutions. The conflicts due to the pipelined structure are identified and solved by an efficient scheduling of the layers.

The third chapter is dedicated to memory optimization. The memory is first reduced in size. A careful study of the saturating process lead to a reduction in number of quantized bit and in memory size. Because of the many rates imposed by the standard, the extrinsic memory requires a wide address range and a wide word range. The solution provided by us gives the required flexibility to the memory with the minimum over cost in memory size. Then, we suggest solutions in order to implement single port RAMs instead of dual port RAMs.

The forth chapter proposes implementation solutions for multi stream decoders. We investigate solutions based on parallelism, sharing resources and adding a buffer to a variable iterative decoder.

A conclusion and perspectives are given at the end of this thesis.

Nobody told them it was impossible, so they made it.

Mark Twain (1867-1902)

1 Background

Summary:

This first chapter introduces the advanced techniques of channel decoding. After a brief overview of digital communication and channel decoder, notions and notations of Low Density Parity Check (LDPC) codes are presented. The layered decoder is then detailed. The LDPC included in the DVB-S2, -T2 and -S2 standards are presented with the state-of-the-art of exiting implementations. Finally, the performance and the testing environment of error correcting codes are discussed.

1.1 Basic concepts

This section mainly refers to the introduction of the book written by John G. Proakis [50] and presents the fundamentals of digital communication systems and gives a special attention to channel decoding.

1.1.1 Digital communication

The subject of digital communication involves the transmission of information in a digital form from a source that generates the information to one or more destinations.

Figure 1.1 illustrates the functional diagram and the basic elements of a digital communication system. In a digital communication, the messages

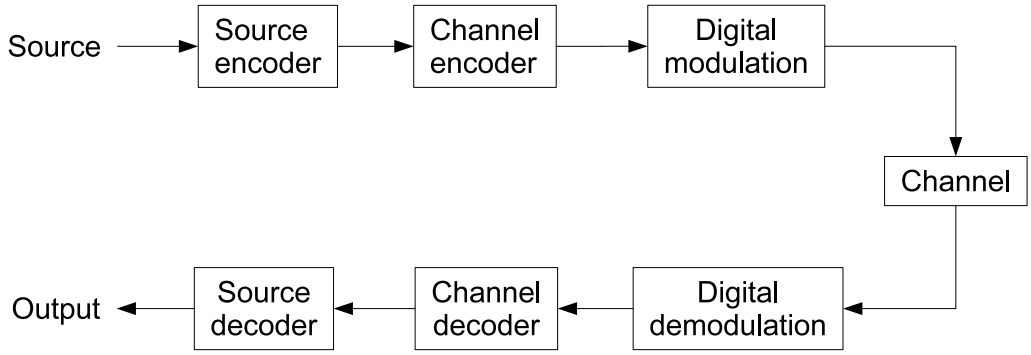


Figure 1.1: Basic elements of a digital communication system

produced by the source are converted into binary digits. The data are compressed through source encoding. The information sequence is passed to the *channel encoder*. The purpose of the channel encoder is to introduce, in a controlled manner, some redundancy in the binary information sequence that can be used at the receiver to overcome the effects of noise in the transmission of the signal through the channel. Thus, the added redundancy serves to increase the reliability of the received data. Encoding involves taking k information bits at a time and mapping them into a unique n-bit sequence called *code word*. The *code rate* of a code is defined by the rate k/n. The *digital modulator* maps the binary information sequence into a signal waveform. In this thesis, we consider the Binary Phase Shift Keying (BPSK) modulation, which maps the binary digit 0 into the waveform $s_0(t)$ and the binary digit 1 into waveform $s_1(t)$. $s_0(t)$ and $s_1(t)$ waveform are sinusoidal signals with one signal shifted by 180 degree from the other. For the rest of the thesis, the BPSK modulation is considered. The *communication channel* is the physical medium that is used to send the signal from the transmitter to the receiver. The *digital demodulator* estimates the transmitted data symbols. The generated sequence of numbers is them passed to the *channel decoder* which attempts to reconstruct the original information sequence from the knowledge of the code used by the channel encoder and the redundancy contained in the received data. Finally the *source decoder* decompress the data to give to original binary digits provided by the source.

1.1.2 Channel decoders

Prior to 1993, the best channel decoder constructions were serial concatenated codes based on an outer Reed-Solomon error correction code combined with an inner Viterbi-decoded short constraint length convolutional code

(used in the DVB-T standard [17]). In 1993, turbo codes were introduced by C. Berrou, A. Glavieux, and P. Thitimajshima (from Telecom-Bretagne, former ENST Bretagne, France) [3]. In a later paper [2], C. Berrou writes "R. Gallager and M. Tanner had already imagined coding and decoding techniques whose general principles are closely related," although the necessary calculations were impractical at that time. This is a reference to LDPC codes that are, like turbo codes, iterative decoders. LDPC codes had been discovered by Gallager [25] in 1963 and had been rediscovered by MacKay [41] in 1997. Compared with turbo codes, LDPC codes have simpler processing requirement and their decoding algorithm is inherently parallel. For long frames, LDPC decoders offer better performance than turbo codes. This makes LDPC decoders relevant for TV broadcasting standards. LDPC codes are included in the linear block codes family and thus heritate from the linear block codes proporties.

1.1.3 Linear block codes

A linear code is an error-correcting code for which any linear combination of codewords is another codeword of the code. Block codes are one of the two common types of channel codes (the other one being convolutional codes), which enable reliable transmission of digital data over unreliable communication channels subject to channel noise. A block code transforms a message consisting of a sequence of information symbols over an alphabet into a fixed-length sequence of n symbols, called a code word. In a linear block code, each input message has a fixed length of $k < n$ input symbols. The redundancy added to a message by transforming it into a larger code word enables a receiver to detect and correct errors in a transmitted code word, and to recover the original message. The redundancy is described in terms of its information rate, or more simply in terms of its code rate $r = k/n$.

1.1.4 LDPC codes

The rediscovery in the nineties of the Turbo codes and, more generally, the application of the iterative process in digital communications led to a revolution in the digital communication community. This step forward led to the rediscovery of Low Density Parity Check (LDPC) codes, discovered three decades before. Several recent standards include optional or mandatory LDPC coding methods. The first standard that included LDPC was the second generation Digital Video Broadcasting for Satellite (DVB-S2) standard (ratified in 2005 [18]).

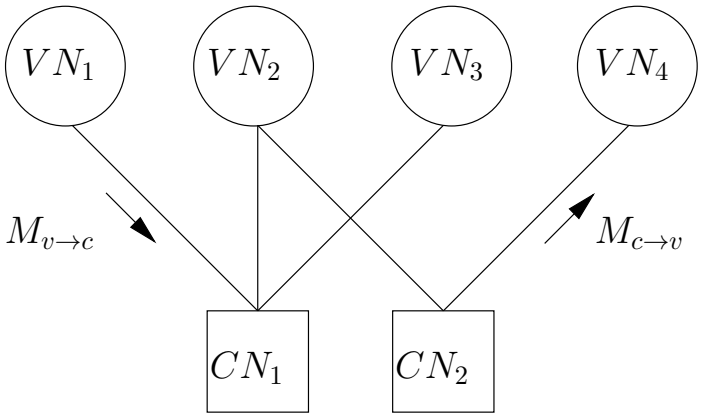


Figure 1.2: Tanner graph representation of **H**

LDPC codes are a class of linear block codes. The name comes from the characteristic of their parity-check matrix which contains only a few 1's in comparison to the amount of 0's. Their main advantage is that they provide a performance which is very close to the theoretical channel capacity [58]. An LDPC decoder is defined by its parity check matrix **H** of M rows by N columns. A column in **H** is associated to a codeword bit, and each row corresponds to a parity check equation. A nonzero element in a row means that the corresponding bit contributes to the parity check equation associated to the node.

The set of valid code words $x \in C$ have to satisfy the equation:

$$x\mathbf{H}^t = 0, \forall x \in C \tag{1.1}$$

An LDPC decoder can be described by a Tanner graph [63], a graphical representation of the associations between code bits and parity checks equation. Code bits are shown as so called variable nodes (VN) drawn as circles, parity check equations as check nodes (CN), represented by squares, with edges connecting them accordingly to the parity check matrix. Figure 1.2 shows a simple Tanner graph of matrix **H** .

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

Through the edge, messages are read by the nodes. A message going from CN to VN is called $M_{c\to v}$, and a message going from VN to CN is called $M_{v\to c}$.

1.1.5 Standard Belief Propagation LDPC decoding

In this subsection, the soft iterative decoding of binary LDPC codes is presented. The iterative process, first introduced by Gallager [25] and rediscovered by MacKay [41] is more known as the Belief Propagation (BP) algorithm. This algorithm propagates probability messages to the nodes through the edges. The probability gives a soft information about the state (0 or 1) of a VN. In the Log-BP algorithm, the probabilities are in the log domain and are called Log Likelihood Ratio (LLR). The LLRs are defined by the following equation:

$$LLR = \log\left(\frac{P(v=0)}{P(v=1)}\right) \tag{1.2}$$

where $P(v=x)$ is the probability that bit v is egual to x.

The order in which the nodes are updated is called the scheduling. The schedule proposed by Gallager is known as the flooding schedule [25]. Flooding schedule consists in four steps. First step is the initialization. Second step is the update of all the check nodes. The third step is the update of all the variable nodes. The fourth step consists in going back to step two until a codeword is found or a maximum number of iteration is reached. At the end of the iterative process, a hard decision is made on the VN to output the codeword. The update of a node means that a node reads the incoming messages and then updates the outgoing messages. The initialization, the VN update and the CN update are described hereafter. The test to find the codeword or stopping criteria is described in Section 4.4.3.

Initialization

Let x denote the transmitted BPSK symbol corresponding to a codeword bit, and let y denote the noisy received symbol, then $y = z + y$ where z is a Gaussian random variable with zero mean. Let us assume that $x = +1$ when the transmitted bit is 0 and x=-1 when the transmitted bit is 1. Let $LLR_{in} = \log\frac{p(x=+1|y)}{p(x=-1|y)}$ denote the a-priori LLR for the transmitted bit. The sign of LLR_{in} gives the hard decision on the transmitted bit, whereas the magnitude of LLR_{in} gives an indication on the reliability of the decision: the greater the magnitude is, the higher is the reliability. On a Gaussian channel LLR_{in} is given by:

$$LLR_{in} = 2y/\sigma^2$$

where σ is the variance of the received signal.

Decoding starts by assigning the a-priory LLR to all the outgoing edge $M_{v\to c}$ of every VNs.

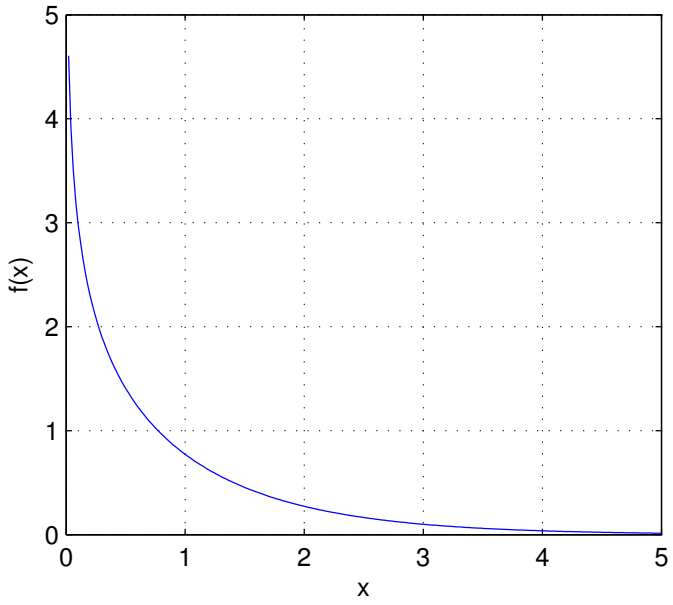


Figure 1.3: Representation of the f(.) function defined in equation 1.5

Check node update

The CN update is computed by applying the Bayes laws in the logarithmic domain. For implementation convenience, the sign (1.3) and the absolute value (1.4) of the messages are updated separately:

$$sign(M_{c\to v}) = \prod_{v'\in v_c/v} sign(M_{v'\to c}) \tag{1.3}$$

$$|M_{c\to v}| = f\left(\sum_{v'\in v_c/v} f(|M_{v'\to c}|)\right) \tag{1.4}$$

where v_c is the set of all the VN connected to the check node CN and v_c/v is v_c without v. The $f(.)$ function is expressed by Equation (1.5) and represented in Figure 1.3.

$$f(x) = -\ln\tanh(\frac{x}{2}) = \ln\frac{\exp x + 1}{\exp x - 1} \tag{1.5}$$

Variable node update

The VN update is split in two equations (1.6) and (1.7) to show the Soft Output (SO) value also called the A Posteriori Probability (APP). The SO

is calculated in (1.6) where the LLR_{in} value is the initial soft input. Then from this SO, the new $M_{v\to c}$ are computed by (1.7).

$$SO_v = LLR_{in} + \sum_{c\in c_v} M_{c\to v} \tag{1.6}$$

$$M_{v\to c} = SO_v - M_{c\to v} \tag{1.7}$$

The VN update is simple and with straight forward implementation. The check node update is more complex and with many possible sub-optimal algorithm and implementation options which are discussed in the next section

1.2 Sub-optimal algorithms

The $f(x)$ function described in equation (1.5) and used in (1.4) is difficult to implement.This function can be implemented us- ing look up table or linear pieceware approximation as in [33] but can also be implemented more efficiently by using a sub-optimal algorithm. The most used algorithms are improved version of the well known Min-Sum algorithm [24], such as the normalized min-sum algorithm [11], the offset min-sum algorithm [11], the λ -min algorithm [29] or the Self-Corrected Min-Sum algorithm [55].

1.2.1 The normalized Min-Sum algorithm and other related algorithms

A lot of researches tend to reduce computational complexity by simplifying the Check Node Processor (CNP). The Min-Sum algorithm is the simplest decoding method which approximates the sum-product algorithm by ignoring other terms except the most minimum incoming message. Therefore the complex computation of function $f(x)$ can be eliminated. The CN Min-Sum processing of Min-Sum algorithm can be expressed as:

$$|M^{new}_{c\to v}| \approx \min_{v'\in v_c/v} |M_{v'\to c}| \tag{1.8}$$

The Min-Sum algorithm can dramatically reduce the computation complexity. However, the resulting approximated magnitude is always overestimated. This inaccurate approximation causes significant degradation on the decoding performance. In the Normalized Min-Sum (NMS) algorithm [11], the output of the CN processor is multiplied by a normalization factor α. This compensates the overestimation of Min-Sum approximation:

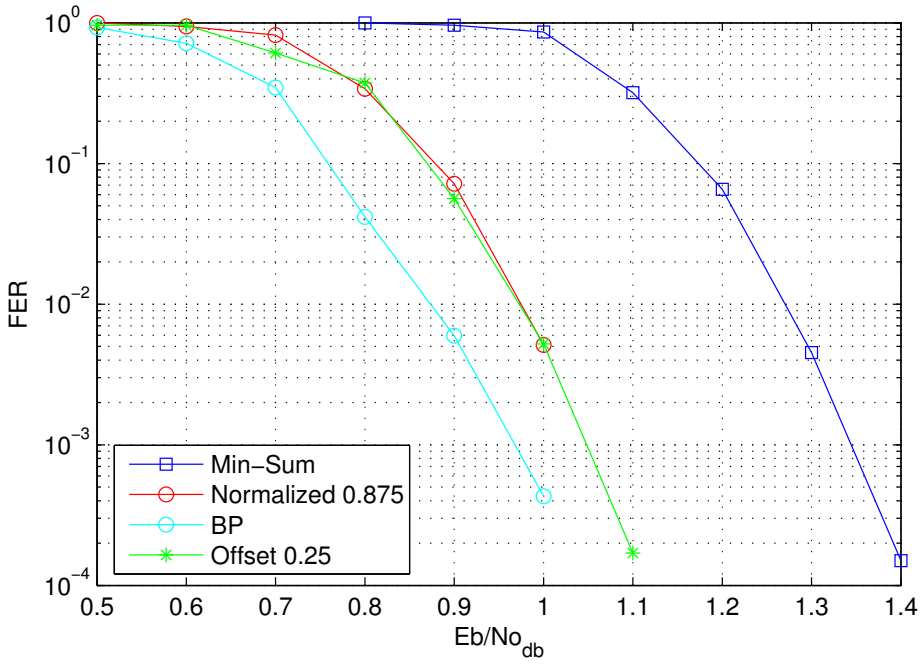


Figure 1.4: Performance comparison of the BP and some sub-optimal algorithm for the CN update.

$$|M_{c\to v}^{new}| \approx \alpha \min_{v'\in v_c/v} |M_{v'\to c}| \tag{1.9}$$

where α is the normalization factor, $0 < \alpha \leq 1$.

Another way to reduce the over-estimation of the outgoing message is to add a negative offset. In the Offset Min-Sum algorithm, equation (1.4) becomes:

$$|M_{c\to v}^{new}| \approx \max\left(\min_{v'\in v_c/v} |M_{v'\to c}| - \beta, 0\right) \tag{1.10}$$

where β is a positive constant.

Figure 1.4 shows the performance simulation of the BP optimal algorithm and the Min-Sum, NMS and OMS sub-optimal algorithms for comparison. The simulation is on a Gaussian channel with floating point computation on short frame N=16200 DVB-S2 matrix, code rate $r = 1/2$, 70 iterations and no outer BCH (Bose and Ray-Chauduri) code. The simulation shows that the Min-Sum algorithm is at 0.4 dB from the BP algorithm and the NMS and OMS are 0.1 dB from the BP algorithm.

Algorithm name	CN update
log-BP	$\|M_{c\to v}\| = f\left(\sum_{v'\in v_c/v} f(\|M_{v'\to c}\|)\right)$
Min-Sum [24]	$\|M_{c\to v}\| \approx \min_{v'\in v_c/v} \|M_{v'\to c}\|$
Offset Min-Sum [11]	$\|M_{c\to v}\| \approx \max\left(\min_{v'\in v_c/v} \|M_{v'\to c}\| - \beta, 0\right)$
Normalized Min-Sum [11]	$\|M_{c\to v}\| \approx \alpha \min_{v'\in v_c/v} \|M_{v'\to c}\|$
λ-Min [29]	$\|M_{c\to v}\| \approx f\left(\sum_{v'\in v_c^\lambda/v} f(\|M_{v'\to c}\|)\right)$
A-min* [35]	IF $v = \arg\min_{v'\in v_c} \|M_{v'\to c}\|$ THEN $\|M_{c\to v}\| \approx f\left(\sum_{v'\in v_c/v} f(\|M_{v'\to c}\|)\right)$ ELSE $\|M_{c\to v}\| \approx f\left(\sum_{v'\in v_c} f(\|M_{v'\to c}\|)\right)$
Self-Corrected Min-Sum [55]	IF $sign(M_{v'\to c}^{tmp}) \neq sign(M_{v'\to c}^{old})$ THEN $M_{v'\to c} = 0$ ELSE $M_{v'\to c} = M_{v'\to c}^{tmp}$ $\|M_{c\to v}\| \approx \min_{v'\in v_c/v} \|M_{v'\to c}\|$

Table 1.1: Check node update with different sub-obtimal algorithms

In the λ-min algorithm [29] only the λ most minimum $M_{c\to v}$ values are computed. Let v_c^λ be the subset of v_c which contains the λ VNs linked to CN c and having the smallest LLR magnitude. Let v_c^λ/v be v_c^λ without VN v. Equation 1.4 is then approximated by:

$$|M_{c\to v}| \approx f\left(\sum_{v'\in v_c^\lambda/v} f(|M_{v'\to c}|)\right) \tag{1.11}$$

Two cases will occur: if the VN belongs to the subset v_c^λ, then the $M_{v\to c}$ are processed over $\lambda - 1$ values of v_c^λ/v, otherwise the $M_{v\to c}$ are processed over the λ values of v_c^λ.

The Self-Corrected Min-Sum algorithm presented in [55] modifies the variable node processing by erasing unreliable $M_{v\to c}$ messages.

Table 1.1 summarizes the different sub-optimal algorithms for the CN update.

The advantages of these algorithms are the simplified computation of equation (1.17) and the compression of the $M_{c\to v}$ messages. Instead of storing all the $M_{c\to v}$ messages conected to one CN, two absolute values, the index of the minimum value and sign of the $M_{c\to v}$ messages need to be stored. This compression lead to significant memory saving with check node degree greater than 4. Although all these algorithms present different performances, the memory space they require to store the $M_{c\to v}$ messages is identical (considering $\lambda = 2$ for the λ-min algorithm). Hence, without loss of generality, for the rest of the thesis, we will consider in the NMS algorithm.

An implementation of the NMS algorithm can be easily improved to a λ-min algorithm or A-min* algorithm by adding look up tables. Many other improvements to the Min-Sum algorithm are proposed in the litterature.In [40], a self compensation technique update the normalization factor α as a function of the CN incoming messages. Because of the massive required computation, the method is simplified in [64] by defining a normalization factor for the min value α_{min} and an other factor for the other value. Furthermore, the factor is shunted after some iteration to improve the performance.

From the algorithms presented in table 1.1, The NMS algorithm is chosen for the following reasons:

- performance close to the BP algorithm
- easy implementation
- low implementation area
- extrinsic memory compression
- not sensitive to scaling in the LLR_{in} values
- possibility to improve performance by upgrading to A-min*,λ-min or Self-Corrected Min-Sum algorithm

1.2.2 Serial implementation of the NMS algorithm

The CN generates two different values: *min* and *submin*. The *min* value is the normalized minimum of all the incoming $M_{v\to c}$ values and the *submin* is the second normalized minimum. Let ind_{min} be the index of *min* i.e.,

$$ind_{min} = \arg \min_{v' \in v_c/v} |M_{v'\to c}|$$

For each $|M^{new}_{c\to v}|$ values, if the index of $M^{new}_{c\to v}$ is ind_{min} then $|M^{new}_{c\to v}| = submin$ else $|M^{new}_{c\to v}| = min$. The $M_{c\to v}$ from one CN can be compressed with four

elements, i.e. min, $submin$, ind_{min} and $sign(M_{c\to v}^{new})$. For matrices with a check node degree greater than four, memory saving becomes significant.

To compute the sign, the property that the XOR of "all but one", is equal to the XOR of "all plus *the* one" is used, i.e.

$$\bigoplus_{v'\in v_c/v} x_{v'} = \bigoplus_{v'\in v_c} x_{v'} \oplus x_v \tag{1.12}$$

where x_v is a sign bit (0 for a positive value and 1 for negative value). The sign calculation of the outgoing messages is done in two steps. First, all the incoming messages are read. The recursive property of the XOR function:

$$\bigoplus_{i=1}^{d_c} x_i = x_{d_c} \oplus (x_{d_c-1} \oplus (x_{d_c-2} \oplus (x_{d_c-3} \oplus (\cdots \oplus (x_1) \cdots)))) \tag{1.13}$$

is used to compute the XOR of all the incoming messages serially. In the second step, the sign of the outgoing messages are computed serially by applying equation (1.12).

To compute the magnitude, the first step consists in a serial sorting of the incoming value to produce the min, $submin$ and ind_{min} values. During the second step, the min, $submin$ and ind_{min} values are used to serially compute the outgoing magnitude messages.

Figure 1.5 shows the serial CN processor implementation. The $M_{v\to c}$ values arrive serially in a two's complement form and are transformed in sign and magnitude representation to be computed separately. The figure is split horizontally and vertically: the upper part is for the sign computation and lower part for the magnitude computation. The left part represents the first step computation and the right part the second step. The upper part is a straight forward implementation of the sign computation where $signT = \oplus_{i=1}^{d_c} x_i$. The upper left part implement equation (1.13) in a first step. Then the right upper part implements equation (1.12) for the second step. The lower part implement the magnitude computation with in a first step, the sorting of the incoming magnitude in order to store the min, $submin$ and ind_{min} values until all the incoming messages are read. Then in the second step (lower right part) the min, $submin$ and ind_{min} values previously sorted are used to serially compute the new outgoing messages. Then, the sign and magnitude are transformed in two's complement for later computation. Step 1 and step 2 take d_c cycles to process and can easily be pipelined.

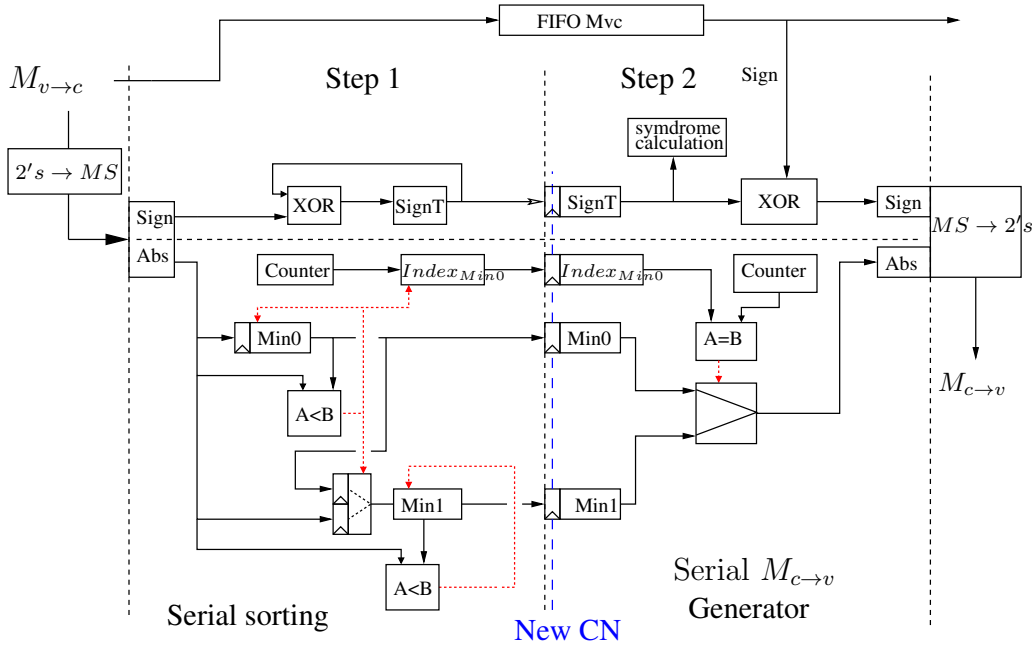


Figure 1.5: serial CN processor implementation.

1.3 LDPC Layered decoder

The layered decoder compared with the straight forward LDPC decoder benefits from three improvements: the turbo message passing schedule, a structured matrix and a Soft Output (SO) centric architecture.

1.3.1 The turbo message passing schedule

The turbo decoding messages passing introduced by Mansour [43, 42] is also known as shuffled iterative decoder [70], staggered decoding or gauss-seidel iteration. Turbo decoding applies a different message schedule than the two phase flooding schedule. In the case of the horizontal shuffle schedule, the basic idea is to process the CNs one by one and to pass the newly calculated messages immediately to the corresponding VNs. The VNs update their outgoing messages in the same sub iteration. The next CN will thus receive newly updated messages which improve the convergence speed. Figure 1.6 shows the Probability Density Function (PDF) of the number of iterations before a codeword is decoded for a flooding schedule and for an horizontal shuffle schedule. This figure shows a simulation for N=16200 bits, a code rate of 1/2 and a constant $E_b/N_o = 1dB$. Note that the average number of iterations (it_{avr}) to find a codeword is about two times smaller for the shuffle

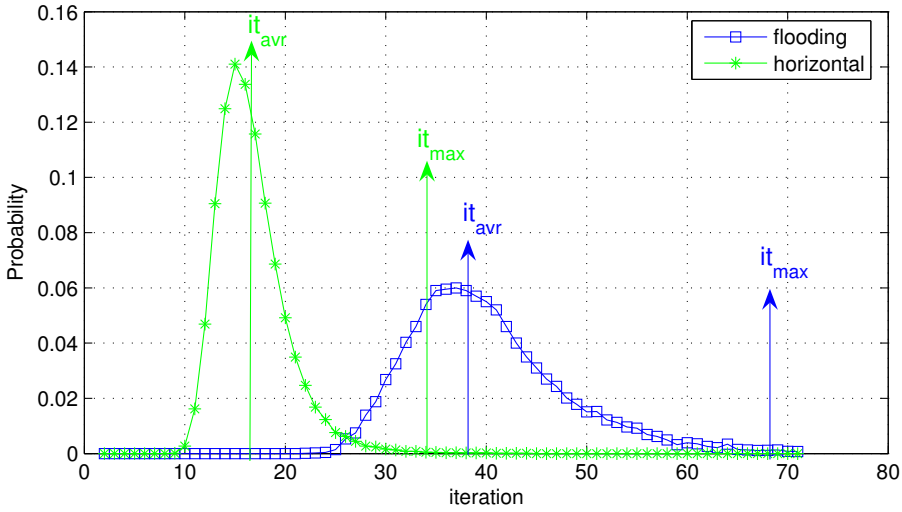


Figure 1.6: Probability Density Function of the number of iterations before a codeword is found

schedule than for the flooding schedule. The same observation can be done for the maximum number of iterations (it_{max}).

However, the main drawback of this schedule is that the CN are updated serially one by one leading to low throughput. The next subsection explains how this serial schedule can be partially parallelized.

1.3.2 Structured matrices

The semi-parallel horizontal Shuffle decoder is also known as Group horizontal Shuffle or Layered Horizontal shuffle [53]. Layered decoding uses the same principle as the Turbo decoding but instead of processing the CNs one by one, they are processed by groups. Let P be the size of the group. Then P CN are processed in parallel. This is possible only if the weights of the columns in a block does not exced one. The idea is to use identity matrices of size $P \times P$. The IEEE WiMAx standard [60] uses this structure and is well explained in [10] where an efficient architecture is also presented. Figure 1.7 shows the structure of the rate-2/3 short-frame DVB-S2 LDPC parity check matrix. This structured matrix is composed of shifted identity matrices of size $P = 360$, allowing for efficient parallel processing of up to 360 CNs.

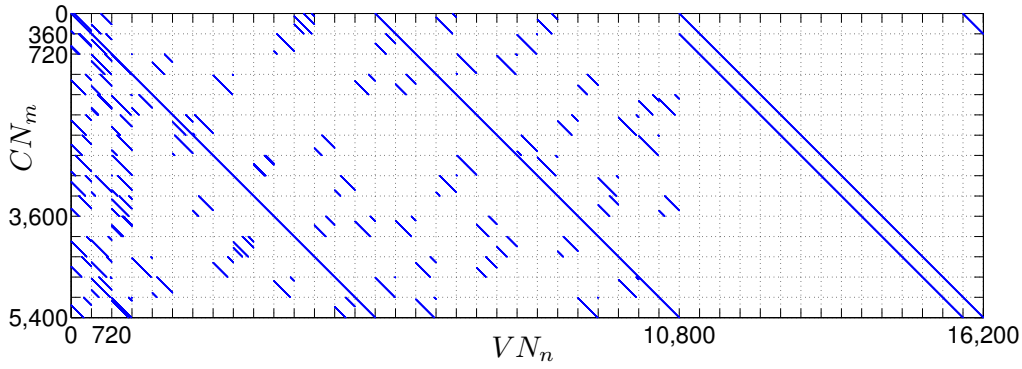


Figure 1.7: Block-structured rate-2/3 DVB-S2 matrix (N=16200 bits)

1.3.3 Soft Output (SO) centric decoder

In this subsection we explain how the soft output (SO) based check node processor (CNP) architecture is deduced. From (1.6) and (1.7), we can find the new equation:

$$SO_v = M_{v\to c} + M_{c\to v} \tag{1.14}$$

The update of the VNs connected to a given CN is done serially in three steps. First, the message from a VN to a CN ($M_{v\to c}$) is calculated as:

$$M_{v\to c} = SO_v - M^{old}_{c\to v} \tag{1.15}$$

The second step is the serial $M_{c\to v}$ update, where $M_{c\to v}$ is a message from CN to VN, and is also called extrinsic. Let v_c be the set of all the VNs connected to CN c and v_c/v be v_c without v. For implementation convenience, the sign and the absolute value of the messages $|M^{new}_{c\to v}|$ are updated separately:

$$sign(M^{new}_{c\to v}) = \prod_{v'\in v_c/v} sign(M_{v'\to c}) \tag{1.16}$$

$$|M^{new}_{c\to v}| = f\left(\sum_{v'\in v_c/v} f(|M_{v'\to c}|)\right) \tag{1.17}$$

where $f(x) = -\ln\tanh\left(\frac{x}{2}\right)$. The third step is the calculation of the SO_{new} value:

$$SO^{new}_v = M_{v\to c} + M^{new}_{c\to v} \tag{1.18}$$

The updated SO^{new}_v value can be used in the same iteration by another sub-iteration leading to convergence which is twice as fast as the flooding schedule [42].

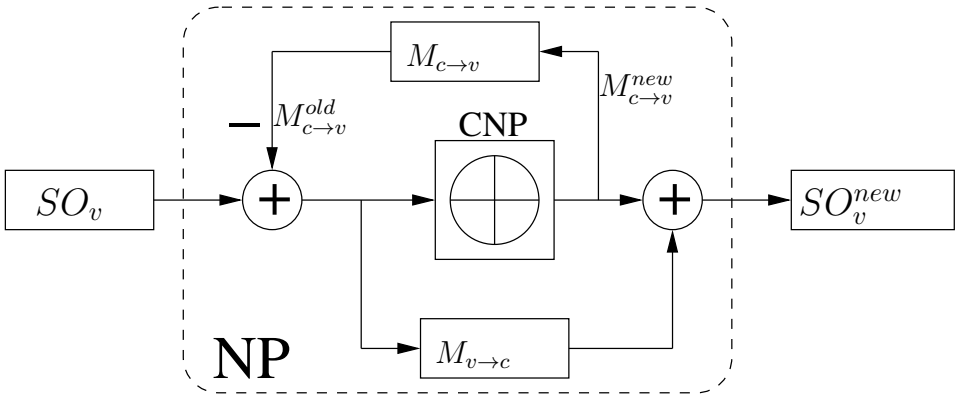


Figure 1.8: SO based Node Processor

1.3.4 Architecture overview

From equations (1.15) to (1.18), the Node Processor (NP) architecture Fig. 1.8 can be derived. The left adder of the architecture performs equation (1.15) and the right adder performs equation (1.18). The central part is in charge of the serial $M_{c\to v}$ update.

As the structured matrices are made of identity matrices of size P, P CNs are computed in parallel. Hence, the layered decoder architecture is based on P NPs that first read serially the Groups of P VNs linked to one layer and then write back the SO_v^{new} in the VNs.

The architecture proposed in Figure 1.9 is mainly based on the architecture of a layered decoder. The counter counts up to IM_{base} (i.e. the number of identity matrices in the base matrix). The ROM linked to the counter delivers the VG_i^{read} addresses and the associated shift value $Shift_i$ following the base matrix order. The VG_i^{write} adresse is given by VG_i^{read} which is delayed by $d_c + \epsilon$ cycles corresponding to latency to compute a new SO value. The size of the ROM dedicated to store VG_i and $Shift_i$ is thus $IM_{base} \times (log_2(N/P) + log_2(P))$.

Usually, a barrel shifter is in charge of shifting the SO values by $Shift_i$ before being processed, and after processing, another barrel shifter is in charge of shifting back the SO values in memory. In this architecture, the $\Delta Shift$ generator allows using one barrel shifter instead of two.

As there is no barrel shifter in charge of shifting back the SO values, at the next call of a VNG, the SOs in this group are already shifted by the shift value of the previous call ($Shift_i^{old}$). The $\Delta Shift$ value takes into account the shift value of the previous calls by doing the subtraction $\Delta Shift_i = Shift_i^{new} - Shift_i^{old}$. $Shift^{old}$ is stored in a RAM of size $N/P \times log2(P)$.

The $M_{v\to c}$ memory is implemented using a FIFO of size dc. For the SO

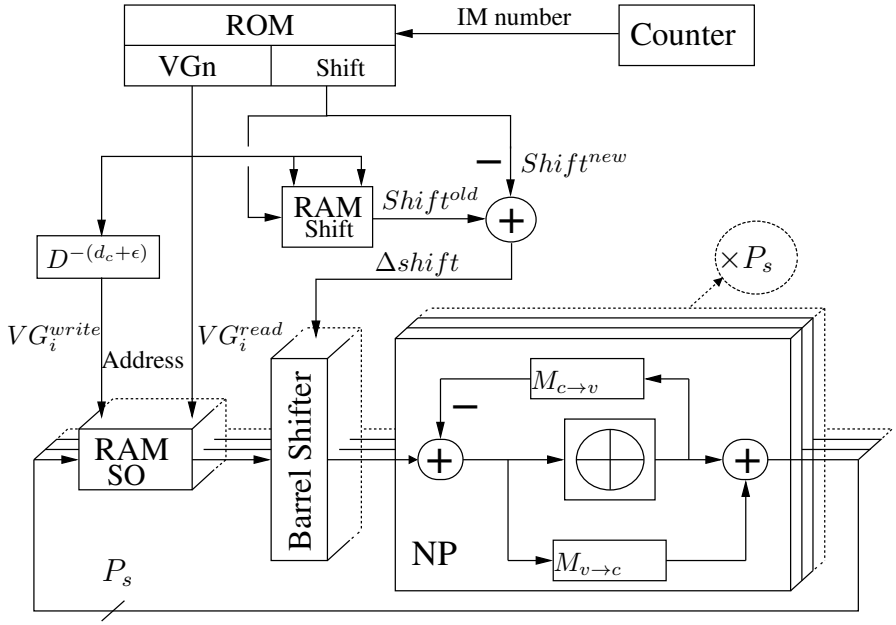


Figure 1.9: Layered decoder architecture

RAM, a dual-port RAM with one port for writing and another for reading is required for pipelining.

Figure 1.10 presents the detailed architecture of the CN update and the $M_{c\to v}$ memory for the implementation of the NMS algorithm. Note that the architecture includes the NMS update described in Figure 1.5.

1.4 The DVB-S2, -T2 and -C2 standards

1.4.1 Digital Terrestrial Television

With the establishment of the European Digital Video Broadcasting (DVB) standard and the American Television Systems Committee (ATSC) standard, digital TV (DTV) broadcasting is now a reality in several countries. The Terrestrial broadcasting or Digital Terrestrial Television (DTT), in France the "Television numérique terrestre" (TNT) are defined by the DVB-T standard [19]. Even if DTT requires a set top box equipment or a specific chip in the TV, it offers many advantages. Among them, thanks to the Forward Error Correction (FEC), DTT allows obtaining optimum picture when an analogue tuner would only allow a poor quality picture. With DTT, the high definition television (HDTH) is also possible. The format is the 1920 by 1080 pixel/frame format interlaced at 60 fields per second.

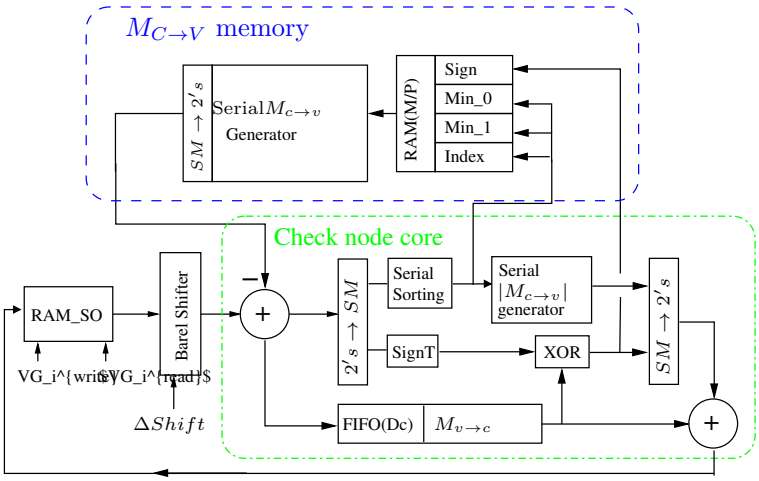


Figure 1.10: Detailed SO based Node Processor

1.4.2 DVB group

The DVB Project, founded in September 1993, is a consortium of public and private sector organizations in the television industry. Its aim is to establish the framework for the introduction of MPEG-2 based digital television services. Now comprising over 200 organizations from more than 25 countries around the world, DVB fosters market-led systems, which meet the real needs, and economic circumstances, of the consumer electronics and the broadcast industry.

DVB-S (EN 300 421 [17]) was introduced as a standard in 1994 and DVB-DSNG (EN 301 210 [18]) in 1997. The DVB-S standard specifies QPSK modulation and concatenated convolutional and Reed-Solomon channel coding, and is now used by most satellite operators worldwide for television and data broadcasting services. In addition to DVB-S format, the DVB-DSNG specifies the use of 8PSK and 16QAM modulation for satellite news gathering and contributing services.

The new standard for digital video broadcast features a powerful FEC system which enables transmission close to the theoretical limit (Shannon limit). DVB-S2 is a single and very flexible standard which covers a variety of applications by satellite and among them, a powerful FEC system based on LDPC codes concatenated with BCH codes, allowing Quasi-Error-Free operation at about 0,7dB to 1 dB from the Shannon limit, depending on the transmission mode (AWGN channel, modulation constrained Shannon limit). The 2^{nd} Generation Terrestrial DVB (DVB-T2) standard adopted in 2009 and the 2^{nd} Generation Cable DVB (DVB-C2) adopted in 2010 include a

Standard	lenght	code rates
DVB-S2	short	1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9
	long	1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, 9/10
DVB-T2	short	1/4, 1/2, 3/5*, 2/3, 3/4, 4/5, 5/6
	long	1/2, 3/5, 2/3*, 3/4, 4/5, 5/6
DVB-C2	short	1/2, 2/3, 3/4, 4/5, 5/6, 8/9
	long	2/3*, 3/4, 4/5, 5/6, 9/10

Table 1.2: code rates for DVB-S2, -T2, -C2 standards

common Forward Error Correction (FEC) block with the DVB-S2 standard.

1.4.3 The LDPC code in the DVB-S2, -T2 and -C2 standards

The DVB-S2, -T2, -C2 standards features variable coding and modulation to optimize bandwidth utilization based on the priority of the input data, e.g., SDTV could be delivered using a more robust setting than the corresponding HDTV service. These DVB standadards also features adaptive coding and modulation to allow flexibly adapting transmission parameters to the reception conditions of terminals, e.g., switching to a lower code rate during fading.

Code rates

The DVB-S2, -T2, -C2 standards [20, 22, 21] are characterized by a wide range of code rates (from 1/4 up to 9/10) as shown in table 1.2. Furthermore, FEC frame may have either 64800 bits (normal) or 16200 bits (short). Each code rate and frame length corresponds to an LDPC matrix: this is 21 matrices for the DVB-S2 standard, 13 matrices for the DVB-T2 standard and 11 matrices for the DVB-C2 standard. The matrices construction is identical for the three standards. The advantage is that the same LDPC decoder can be used for the 3 standards. Due to the fact that the decoder is identical for the 3 standards, hereafter DVB-X2 refers to DVB-2, -T2, -C2 standards.

Matrix construction

LDPC codes can be specified through their parity check matrices. The DVB-X2 LDPC codes are 64800 bits long. Therefore, a certain structure is imposed

[0]*upgraded matrix with better performance compared to DVB-S2 standard

on parity check matrices to facilitate the description of the codes and the encoding.

By inspecting the construction rules in [20, 22, 21], the DVB-S2, -T2 and -C2 parity check matrices consist in two distinctive parts: a random part dedicated to the systematic information, and a fixed part that belongs to the parity information. Two types of VN can be distinguished: the Information VN (IVN) and the Parity VN (PVN) corresponding to the systematic and parity bits respectively. The connectivity between every IVN_m and CN_j is defined by the standard encoding rule:

$$CN_j = CN_j \oplus IVN_m,\ j = (x + q(m \mod 360)) \mod M. \tag{1.19}$$

CN_j is the j^{th} parity bit, IVN_m it the m^{th} information code bit, q and M are code rate dependent parameters specified by the DBV-X2 standard. The variable x is given by the Tables defined in the standard.

Table 1.3 is a copy of the Appendix C of the DVB-S2 standard [20]. This table defines the x value used in Equation (1.19). The first line defines the connections of the information bits $m = 0 \times 360$ to $(1 \times 360) - 1$, the second line is for $m = 1 \times 360$ to $(2 \times 360) - 1$) and so on.

The fixed zigzag connectivity of the PVN and CN is defined by the encoding scheme:

$$CN_j = CN_j \oplus CN_{j-1},\ j = 1, 2, \ldots, N - 1. \tag{1.20}$$

This construction scheme results in a staircase lower triangular on the right part of the matrix. These type of LDPC codes with simple encoding procedure are also called Irregular Repeat Accumulate (IRA) codes [34] result in a linear complexity of encoding with respect to the frame length.

From the construction method defined in the standard, a parallelism of 360 can be deduced but it does not look like a prototype matrix ready for a layered decoder.

In the following subsection, we describe how it is possible to build a matrix, made of shifted identity matrices of size 360 (see Figure 1.7), by reordering the matrix.

Matrix reordering for layer decoding

The construction process of the new matrix relies on the permutation of the rows and columns in two steps: first the CN are permuted as described in the following equation:

0 2084 1613 1548 1286 1460 3196 4297 2481 3369 3451 4620 2622
1 122 1516 3448 2880 1407 1847 3799 3529 373 971 4358 3108
2 259 3399 929 2650 864 3996 3833 107 5287 164 3125 2350
3 342 3529
4 4198 2147
5 1880 4836
6 3864 4910
7 243 1542
8 3011 1436
9 2167 2512
10 4606 1003
11 2835 705
12 3426 2365
13 3848 2474
14 1360 1743
0 163 2536
1 2583 1180
2 1542 509
3 4418 1005
4 5212 5117
5 2155 2922
6 347 2696
7 226 4296
8 1560 487
9 3926 1640
10 149 2928
11 2364 563
12 635 688
13 231 1684
14 1129 3894

Table 1.3: matrix construction table for DVB-S2 standard, code rate of 2/3 and short frame lengh

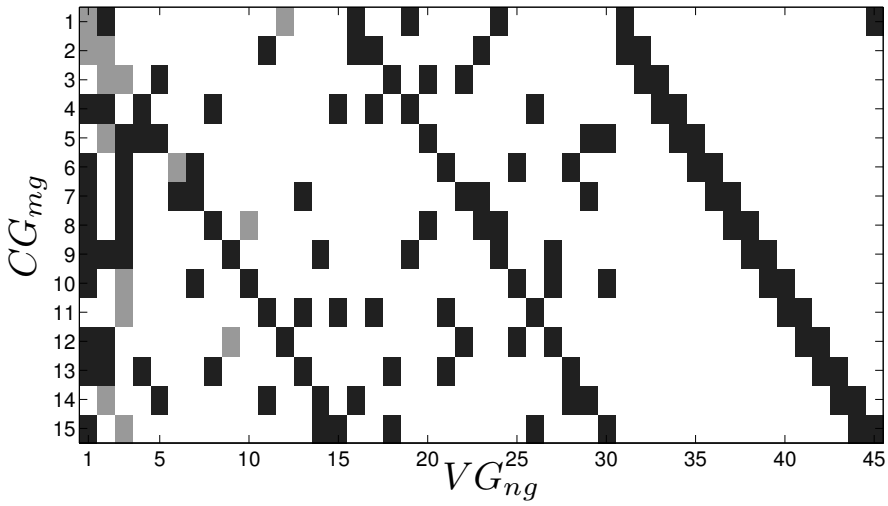


Figure 1.11: Base DVB-S2 matrix representation

$$\sigma(j+i\times 360) = i+(j\times q) \mod N,\ i=1,2\ldots,q,\ j=1,2,\ldots,360. \quad (1.21)$$

Then the PVN are permutated by following the same permutation equation. The obtained matrix is made of Identity Matrix (IM) sized 360.

The matrix construction and permutation process is illustrated in appendix A. Figure 1.11 illustrates the IM in the rate-2/3 short-frame matrix. VG_i denotes the i^{th} group of 360 VNs and CG_j denotes the j^{th} group of 360 CNs. A square denotes a permuted IM linking CG_j and VG_i. Let im be the identity matrix number. Each IM ($im = 1, 2, \ldots, IM^{total}$) can be best described by giving the shift value δ_{im}^{IM} and the coordinate of the IM in the matrix which is given by the CG_{im}^{IM} value and the VG_{im}^{IM} value.

There is a convenient way to obtain directly the shift value δ_{im}^{IM}, and the address (CG_{im}^{IM} and VG_{im}^{IM}) values from the standard value x given in the standard table 1.3. In this table, each x value corresponds to an identity matrix. The line number corresponds to the VG number $VG_{im}^{IM} = line$. The shift value of each im is given by:

$$\delta_{im}^{IM} = (360 - \lfloor x/q \rfloor) \mod 360. \quad (1.22)$$

and the check group number is given by:

$$CG_{im}^{IM} = x \mod q. \quad (1.23)$$

The connectivity of the PVN and CN is built by a staircase of identity matrices as in Figure 1.11($VG_{31}toVG_{45}$). Each identity matrix in the staircase has a shift value of 0. The up right IM is a special IM with a shift of 359 and the up right connectivity removed.

The permutation of the matrices allows obtaining a structured matrix ready to be used by a layered decoder. The DVB-S2 standard was the first standard including an LDPC decoder and many different implementation options have been tested in the State of the art.

1.4.4 State-of-the-art on DVB-S2 LDPC implementation

To our knowledge, the first implementation presented by F. Kienle et al [36] with synthesis results, benefits from the parallelism of $P = 360$. The implementation relies on a flooding schedule and an optimized update of CN of degree 2. Due to the varying node degree, the functional CN units process all incoming messages in a serial manner. This serial processing of the coming messages is a common point to most implementations recorded in our state of the art. Although it was able to provide a throughput far above from the mandatory rate of 90 Mbps, the high number of processing units and the long width of the barrel shifter require a huge silicon area. The throughput of this first implementation was more than required by the standard.

In [14],[27] the parallelism is reduced to any integer factor of 360 by using an efficient memory bank partitioning.

In all implementations recorded, the shuffling network implementation is based on the use of barrel shifters. The differentiation comes from the size and the number of shuffling networks. The size of the shuffling network will depend on the parallelism. In [14, 36] two shuffling networks are used (one for shuffle and the other to shuffle back), while in [27] the author saves one shuffling network. In [27] instead of shuffle back the data, the shift value is memorized and thanks to the linearity of the barrel shifter, the next shift value is computed by a simple substraction.

In [14] the authors presented the first implementation that benefit from layered decoder architecture. Implementations of layered decoder are also presented in [57, 68, 48, 66, 65]. A layered decoder requires matrices made of IM, but in the case of the DVB-X2 matrices, the superposition of matrices appears. The superposed matrices are not compatible with a layered decoder leading to conflicts and a patch is mandatory to solve the problem. In [57, 48], the problem is identified and solved by the computation of the message variation. The problem is also solved by using a 'delta' architecture

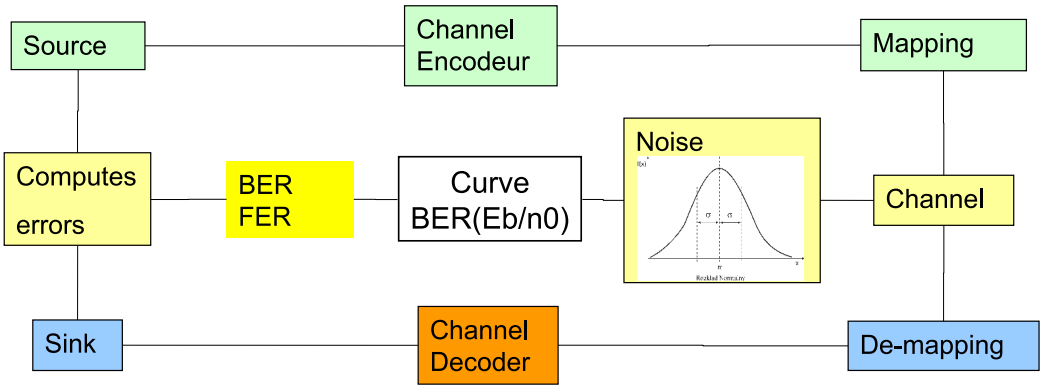


Figure 1.12: Block diagram of a testing environment

as described in [8] and in [53]. These solutions are efficient but compared to a standard layered decoder requires another access to the SO memory to compute the new SO ($SO^{new} = SO^{old} + delta$) which is costly in terms of memory size.

A vertical layer schedule version of the layered decoder is proposed in [47]. This architecture allows iterative message passing between the de-mapper and the LDPC decoder. In [23] an alternative approximation of CN algorithm for DVB-S2 LDPC decoder is proposed. The alternative approximation is based on the first series approximation of equation (1.5), the magnitude update of the $|M_{c\to v}|$ is expressed with $\log_2(x)$ and 2^x function which can be easily implemented by a shifter and apriority encoder.

From this state of the art, the choice of a layered decoder has been decided. Based on this choice, the research is focused on an efficient solution for the memory conflicts and an optimized implementation in term of area and consumption. In this thesis, a testing environment has been developed to test the performance of the different solutions. This testing environment is described in the next section.

1.5 Testing the performance of a decoder

Advanced digital wireless communication systems often require an appropriate trade-off between complexity and performance of an efficient iterative decoder design. In practice, from the floating point to the fixed-point hardware description, many parameters (reliability message length, digital word size, rounding and quantization operations, etc.) should be jointly optimized. However, these parameters interact in a non-linear way and the selection of the optimal algorithm is a very high time-consuming task. Usually, the ex-

pression of Bit Error Rate (BER) or the Frame Error Rate (FER) expressed in [50] is used to predict the performance of the system. The conventional solution is the Monte-Carlo simulation that evaluates the BER, which gives an estimation of the error correcting capability of the decoder. Figure 1.12 shows block diagrams of a testing environment using Monte-Carlo simulation. The source is random and the noise has a Gaussian distribution of variance depending on the E_b/N_0 ratio. In this section, two practical applications are presented to test an LDPC decoders.

1.5.1 Software and hardware simulation

The Monte-Carlo simulation method is traditionally performed by software programs. With this approach, a FER around 10^{-9} requires one or two weeks of simulation. To speed up these very long simulations, some software approaches are proposed, such as, a reduced Monte-Carlo simulation method [7] that re-runs only erroneous codewords obtained from an initial "classical" Monte-Carlo simulation. In [59] is proposed a technique called the distance-based method which is based on the direct evaluation of a distance between the soft output of the sub-optimal decoder and the soft output of a reference decoder. Although these methods reduce the simulation time, the software based execution (for instance, executing applications on a conventional CPU cluster) is still costly due to the high power consumption and physical space cost. Consequently, the solution used for this thesis is simulation by using a hardware accelerator based simulation. To combine the high speed of hardware simulation and the flexibility of software simulation, the testing environment is designed to be able to switch from software to hardware simulation at any time for C-VHDL co-simulation. Another advantage of this solution is the use of co-simulation for debugging.

1.5.2 Test with all-zero codewords

Due to the linearity of the LDPC decoder, we can consider an 'all zero codeword' with a Binary Phase-Shift Keying (BPSK) transmission on a Gaussian channel. Fig. 1.13 shows the all zero codeword model for the test of FER or/and BER. The *channel emulator* block is in charge of adding a noise to the signal to emulate a noisy channel. For a Gaussian channel, an Additive White Gaussian Noise (AWGN) is added to the signal as in [6, 39]. We presented in [9] an Hardware Discrete Channel Emulator (HDCE) which is more detailed in Appendix B. In the *compute errors* block linked with the decoder output, each non-zero value is counted as a bit error. With the bit errors, the BER and FER are easily deduced and the index (SNR) is incremented

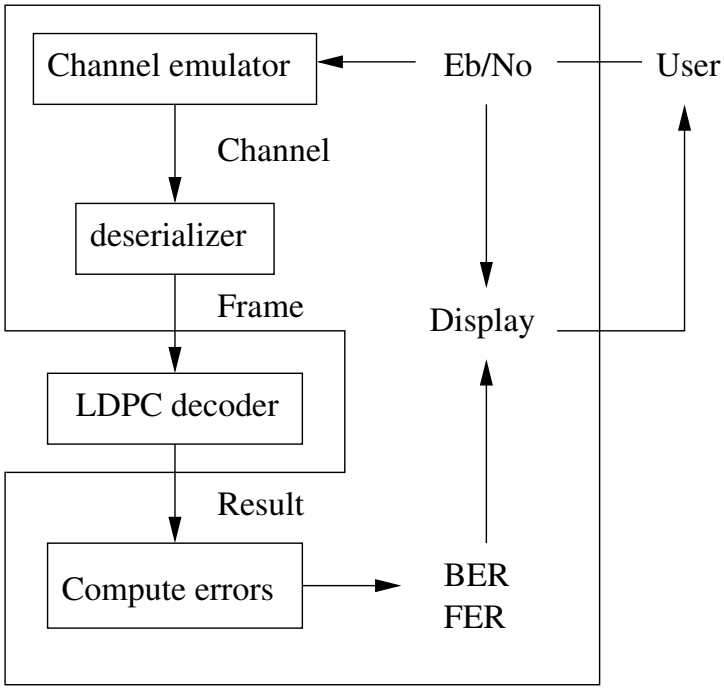


Figure 1.13: Test of an LDPC decoder, all-zero codeword model

when a maximum number of frame errors is reached. The testing environment including the HDCE is low cost and easy to design. The test patch can be included as a part of a decoder chip or IP for built-in SNR estimation and testing purposes at a low area cost. It would take less than 2% of the area of a DVB-S2 LDPC decoder described in [48] .

1.5.3 Test of a communication model

To get closer to a real communication, or to test the encoder, the test of the performance of the LDPC codes in a communication model becomes relevant. In Figure 1.14, the codeword from the encoder is sent serially to emulate a BPSK modulation on the AWGN channel. The FIFO stores the source words until the codeword is decoded. Then the *compute errors* block compares the two words and deduces bit errors. This model has been used to test a non binary LDPC code in the DAVINCI project [51].

1.5.4 Channel emulator

To emulate a Gaussian channel, usually an Accurate White Gaussian Noise (AWGN) is added to the signal as in [26, 6, 39]. An alternative is to use

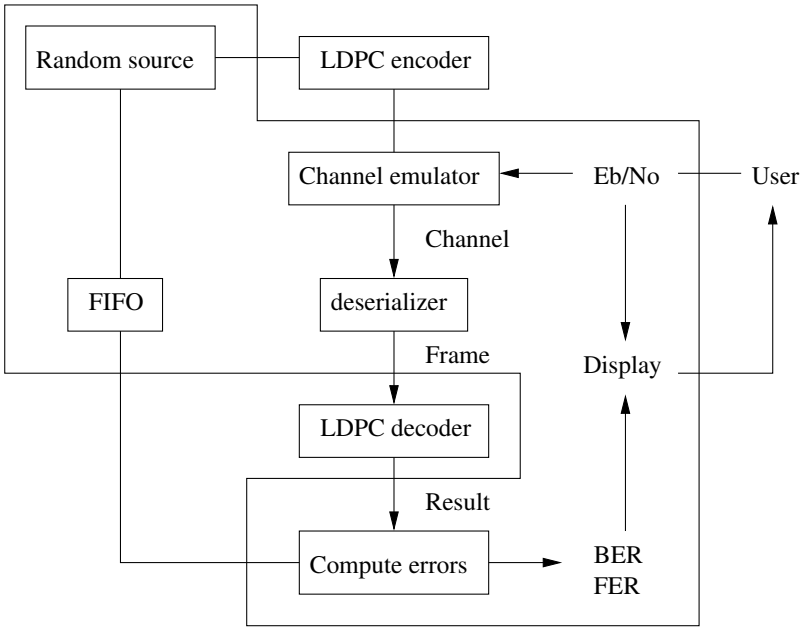


Figure 1.14: test of an LDPC decoder

the Hardware Discrete Channel Emulator (HDCE) presented in [9] and in Appendix B. The implementation of the HDCE was part of my master's degree work. The HDCE directly produce a value with the required distribution. The HDCE relies on the "Alias Method" [67] and the implementation has been optimized for high speed. The advantages of the HDCE are a reduced area cost, high speed, the possibility to emulate other channel and the possibility to replay sequence for debugging.

1.5.5 Interpreting results

The error rate of iteratively decoded codes has a typical shape such as sketched in Figure 1.15. Three regions can be easily distinguished. The first region is where the code is not efficient, even if the number of iterations is increased. The *waterfall region* is where the error rate has a huge negative slope. Ideally the waterfall region is vertical. The more the waterfall region is on the left side, better is the code. The *error floor* region is where the curve does not fall as quickly as before. The error floor appears at low BER, and is usually caused by trapping sets or pseudo-codewords. The error floor has to be at a BER as low as possible for a good code.

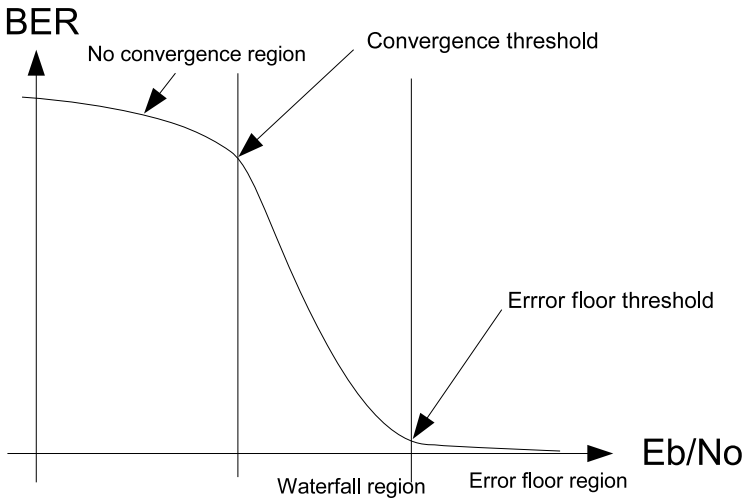


Figure 1.15: Typical regions in an error probability curve

The Shannon limit

Stated by Claude Shannon in 1948, the theorem describes the maximum possible efficiency of error-correcting methods versus level of noise interference and data corruption. The Shannon theorem states that given a noisy channel with channel capacity C and information transmitted at a rate R, then if $R < C$ there exist codes that allows the probability of error at the receiver to be made arbitrarily small. With a given coding rate r, it is possible to calculate the minimum possible $E_b/N_0(dB)$ value.

$$E_b/N_0(min)(dB) = 10\log((2^{2r} - 1)/2r) \tag{1.24}$$

1.5.6 Standard requirements

For the DVB-S2 standard, the standard requirement is to be within 1 dB from the Shannon limit at Quasi Error Free (QEF). The definition of QEF adopted for DVB-S.2 is "less than one uncorrected error-event per transmission hour at a level of a 5Mbit/s single TV service decode", approximately corresponding to a Transport Stream Packet Error Ratio PER $< 10^{-7}$ before de-multiplexer, where a packet is a MPEG packet of 188 bytes. In this thesis, we will approximate the QEF at FER $< 10^{-6}$ and BER$< 10^{-10}$. A simulation that gives results within the standard requirement must have the error floor threshold at BER$< 10^{-10}$ and $E_b/N_0(std)(dB) < E_b/N_0(min)(dB) + 1$.

Table 1.4 gives for the code rates used in the DVB-X2 standards the minimum achievable $E_b/N_0(min)(dB)$ value computed using equation 1.24.

code rate	1/4	1/2	2/3	3/4	4/5	5/6	9/10
$E_b/N_0(min)(dB)$	-1.9	0	1.3	1.9	2.4	2.6	3.2
$E_b/N_0(std)(dB)$	-0.9	1	2.3	2.9	3.4	3.6	4.2

Table 1.4: Shannon limit in function of the code rate

Table 1.4 also gives the maximum $E_b/N_0(std)(dB)$ value at $BER = 10^{-10}$ for a simulation to fulfill the standard requirements.

Take time to deliberate, but when the time for action has arrived, stop thinking and go in.

Napoléon Bonaparte
(1769-1821)

2
Memory update conflicts

Summary:

In this chapter, a pipelined layered LDPC decoder is considered as the best implementation solution for the DVB-X2 standard. The major drawback of using a layered decoder with DVB-X2 standards are the arising memory update conflicts due to the structure of the matrix and the pipelined architecture. Solving these memory conflicts requires efficient strategies to keep performance, high throughput and low hardware complexity. State-of-the-art solution includes complex control, hardware patch or idle cycle insertion. In this chapter, we propose alternative solutions based on the "divide and conquer" strategy to overcome the memory conflicts. Two kinds of conflicts are identified and solved separately: the conflicts due to the matrix structure and the conflicts due to pipelined architecture. The number of conflict due to the structure of the matrix is first reduced by a reordering mechanism of the matrix called split algorithm that creates a new structured matrix which reduces the parallelism. Then the remaining conflicts are solved by an equivalent matrix using added punctured bits. Another solution based on a repeat of the deficient layer and an "on time" write disable of the memory offers an even better performance. To solve the conflict due to pipelining, the split process is used again as a first step followed by an efficient scheduling of the layers and identity matrices.

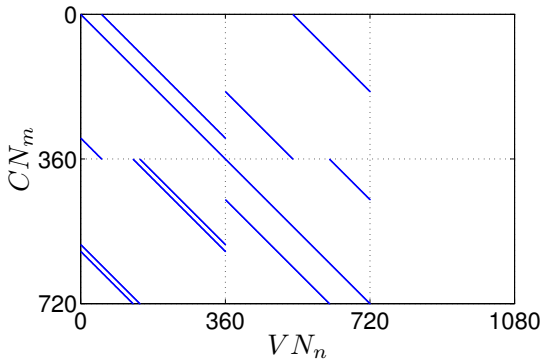


Figure 2.1: Zoom of a rate-$^2/_3$ DVB-T2 Matrix with N=16200

A memory update conflict occurs when a data is computed by a processing unit while this data has not been updated yet by another processing unit. Although DVB-X2 standards define structured parity check matrices, these matrices are not perfectly structured for layered decoder architecture, leading to conflicts in the SO memories. In the beginning of this chapter, our attention is focused on a particular type of conflict introduced by the existence of overlapped shifted identity matrices in the DVB-X2 parity check matrix structure. Then the memory update conflicts introduced by the use of a pipelined CNP are identified and solved by an efficient scheduling.

2.1 Conflicts due to the matrix structure

Figure 2.1 shows a zoom on the first 720 VNs and CNs of the DVB-T2 LDPC matrix illustrated in Figure 1.7. One can see that the first group of 360 CNs is linked twice to the first group of 360 VNs by two diagonals. The sub-matrices with a double diagonal in it will be called Double Diagonal Sub Matrix (DDSM). The DDSM are also called overlapped identity matrices or superposed sub-matrices and are also present in the LDPC matrices of the Chinese Mobile Multimedia Broadcasting (CMMB) standard [61].

Let us consider the case where two CNs (C_1 and C_2) are computed in one layer and connected to the same VN denoted v_{dd}. There are two updates of the same SO value. The calculation of the new SO (2.1) is deducted from equation (1.15) and (1.18). Assuming $\Delta M_{c\to v} = -M^{old}_{c\to v} + M^{new}_{c\to v}$ and using (2.1), the calculation of $SO^{new1}_{v_{dd}}$ and $SO^{new2}_{v_{dd}}$ in (2.2) and (2.3) are obtained respectively.

$$SO^{new}_{v} = SO^{old}_{v} - M^{old}_{c\to v} + M^{new}_{c\to v} \tag{2.1}$$

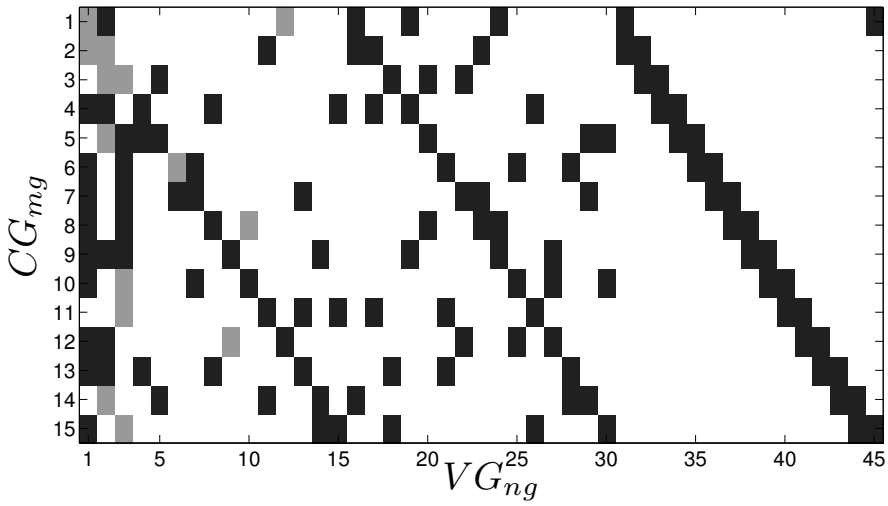


Figure 2.2: Base DVB-T2 matrix representation

$$SO_{v_{dd}}^{new_1} = SO_{v_{dd}}^{old} + \Delta M_{c_1 \to v_{dd}} \tag{2.2}$$

$$SO_{v_{dd}}^{new_2} = SO_{v_{dd}}^{old} + \Delta M_{c_2 \to v_{dd}} \tag{2.3}$$

Because the SO is updated serially in the layered architecture, the SO_{vdd}^{new2} will overwrite the SO_{vdd}^{new1} value. This conflict is equivalent to cut the $M_{c_1 \to vdd}$ message. This is called a cutting edge in [8] and will lead to significant performance degradation. With a straightforward implemented layered decoder, each DDSM will produce P cutting edges and dramatically annihilating the decoding performances.

Figure 2.2 illustrates the number of DDSMs in the rate-2/3 short-frame matrix. VG_i denotes the i^{th} group of 360 VNs and CG_j denotes the j^{th} group of 360 CNs. A black square represent a single permuted identity matrix linking CG_j and VG_i and a gray square corresponds to a DDSM. Note that there are 14 DDSMs in this example.

2.1.1 State-of-the-art

Among all the literature about DVB-X2 implementation, the problem of overlapped sub-matrices is rarely explicitly identified and solved. In [36], the problem is avoided by applying the Gauss-Seidel technique only on the staircase part of the parity check matrix. The architecture in [14] applies layers decoding but does not shows the treatment to superposed sub-matrices.

In [53] and [8] the authors present a solution to avoid memory update conflicts based on the computation of the variation (or delta) of the SO metrics to allow concurrent updates. The computation of this SO update ($SO^{new} = SO^{new} + \Delta M_{c \to v}$) requires an additional SO access which is costly to implement.

In [48], the problem of overlapped matrices is identified and solved by computing the variation only in case of overlapped matrices. In the architecture proposed the SO update ($SO^{new} = SO + delta$) is done locally before shifting back in the SO memory. This architecture save a triple access to the SO memory but require two barrel shifters.

In [57], the two updated SO of two overlapped CN ($SO^{new_1}_{v_{dd}}$ and $SO^{new_1}_{v_{dd}}$) are stored in two different addresses and the SO^{new} is updated with the following equation:

$$SO^{new} = SO^{new_1}_{v_{dd}} + SO^{new_2}_{v_{dd}} - SO^{old}. \tag{2.4}$$

By developing equation (2.4) we find:

$$SO^{new} = SO^{old} + \Delta M_{c_1 \to v_{dd}} + \Delta M_{c_2 \to v_{dd}}. \tag{2.5}$$

The two proposed method are more or less based on the computation of the varation of the SO metrics but the over-cost is limited to the concerned SO memory. In the previously described solutions, an heavy patch and two barrel shifters are used. We propose solutions without patch for an LDPC decoder architecture with one barrel shifter. In [30] is proposed a parallel updating among all the layers (processing of one CN per layer) and a serial updating CNs. The decoding of this algorithm is quite different from the original but it is efficient to solve conflict without performance loss and the implemented architecture is detailed. One drawback of this method can be the shuffling network that can not be a barrel shifter and is not detailed. In the next section we explain how we reduce the number of DDSMs.

2.2 Conflict resolution by group splitting

To achieve the minimum required throughput of 90 Mbps in the DVB-T2 standard, parallel processing of a fraction of the 360 CNs is enough (see Section 4.2. In [14], the authors have used 45 CNPs which lead to significant area reduction, therefore splitting the group of 360 CNs is considered. In [27] and [14], the splitting process has already been done implicitly through memory mapping. [27] describes how the memory banks are sized and organized in function of the parallelism. In the next subsection, we will show

how to reorder the structured matrices initially designed with IM of size 360 to matrices with a significant reduction of DDSM and with IM of size $360/S$, where S is the number of splits.

2.2.1 Construction of the sub-matrices

Let us define P_s as the number of CN working in parallel after a split. The values P, P_s and S are then linked by the equation:

$$S \times P_s = P$$

The construction process of the new matrix relies on the permutation of the rows and the columns in two steps. First a permutation of the rows (CNs) with the permutation defined as:

$$\sigma(i) = (i \mod S)P_s + \lfloor i/S \rfloor \tag{2.6}$$

where $\lfloor x \rfloor$ is the largest integer not greater than x. We first reorder the CN using (2.6) where i is the CN number. Then columns (VNs) are permuted in the same way.

Let us consider the example of a double diagonal sub matrix $\mathbf{H}_{\mathbf{2,6}}^{\mathbf{DDSM}}$ of size $P = 12$ in Figure 2.3(a), where the values in subscript are shift parameters of the two diagonals i.e.: first diagonal (D_1) is shifted by $\delta^{D_1} = 2$ and the second diagonal (D_2) is shifted by $\delta^{D_2} = 6$. This DDSM will produces 12 cutting edges. After reordering the rows using equation (2.6) with $S = 3$ and $P_s = 4$, we obtain the new matrix $\mathbf{H'_{2,6}}$ in Figure 2.3(b). Then a permutation of the columns gives the equivalent matrix shown in Figure 2.3(c). Note that $\mathbf{H''_{2,6}}$ is composed of shifted identity matrices of size $P_s = 4$ and can be best described by its base matrix:

$$\mathbf{H''_{base}} = \begin{bmatrix} \delta_2 & 0 & \delta_0 \\ \delta_1 & \delta_2 & 0 \\ 0 & \delta_1 & \delta_2 \end{bmatrix}$$

where a 0 is a null sub matrix and δ_i is a 4×4 IM right shifted by the i value.

In the general case, there is a convenient way to build the new base matrix layer by layer, using the shift value δ^{D_j} of the diagonal j before split. The column position vg of the sub shifted identity matrix is given by:

$$vg^{l,D_j} = (\delta^{D_j} + l) \mod S \tag{2.7}$$

where l is the layer index. The shift value is given by:

$$\delta^{l,D_j} = \lfloor \delta^{D_j}/S \rfloor + \lfloor (\delta^{D_j} \mod S + l)/S \rfloor \tag{2.8}$$

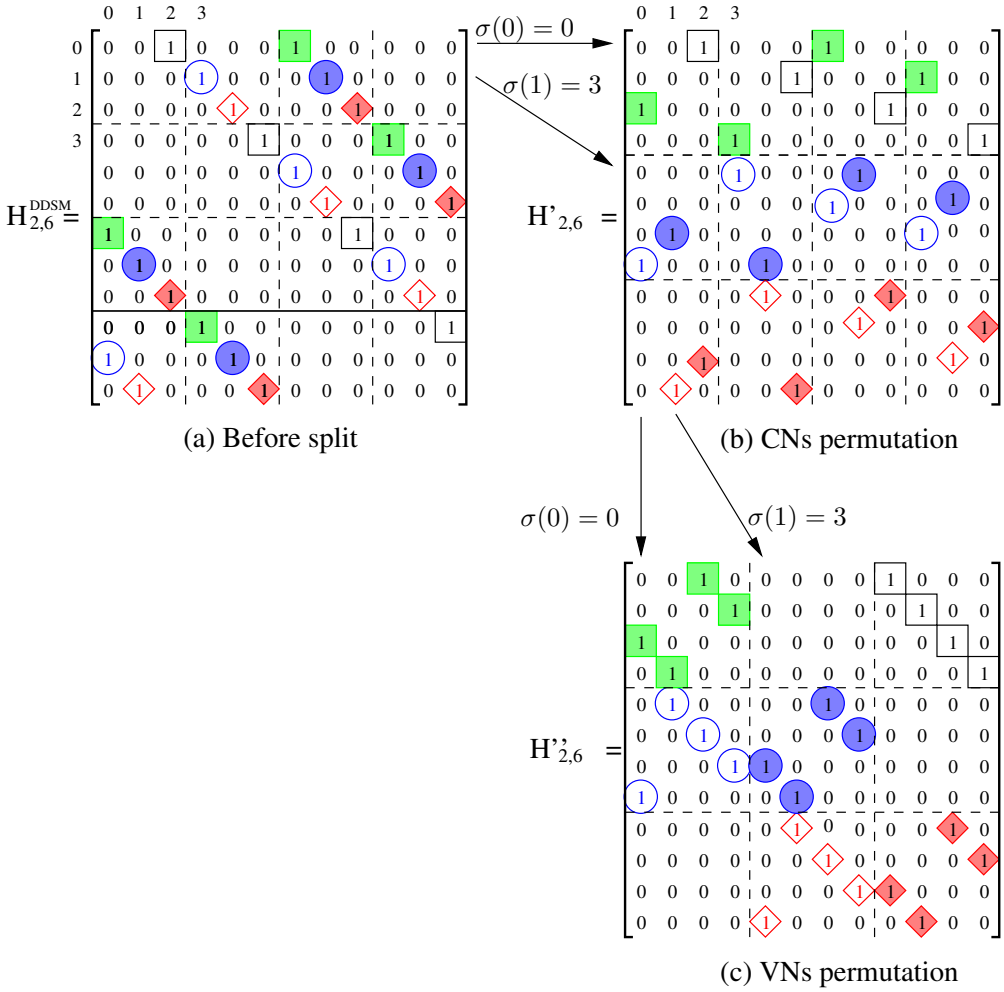


Figure 2.3: Shifted identity matrix

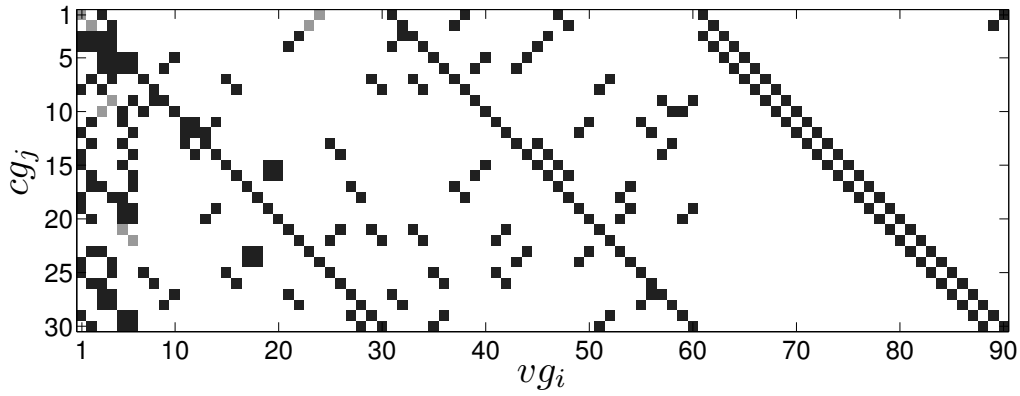


Figure 2.4: Split base DVB-T2 matrix representation

It is important to mention that the splitting of a DDSM does not always remove the double diagonals. If $\delta^{D_1} \mod S = \delta^{D_2} \mod S$ then equation (2.7) gives $vg^{l,D_1} = vg^{l,D_2}$ and there will remains DDSMs in the sub-matrices.

2.2.2 DDSM in DVB-X2 and simulation results

The rate-2/3 base DVB-T2 matrix (Figure 2.2) is split by a factor of two and the obtained matrix is shown in Figure 2.4. It can be observed that after the split, the number of grey squares are reduced from 14 to 8. In terms of cutting edges, this means a reduction from 14×360 cutting edges to 8×180. Tables 2.1 and 2.2 provides the equivalent number of DDSMs of size 360 as a function of the parallelism and the coding rate. An asterisk (*) in the table means that there are multiple overlapped identity matrices greater than two among the counted DDSMs. Table 2.1 provides results for short frames and Table 2.2 provides results for long frames DVB-T2 LDPC codes. Significant reduction of the number of cutting edges can be observed by the proposed group splitting method.

Figure 2.5 gives simulation results for a normalized Min-Sum fixed point layered decoder, with 30 iterations for short frames at a code rate of 2/3 in Additive White Gaussian Noise (AWGN) channel. We simulate an architecture where the channel value is quantified on 5 bits, the SO on 6 bits (see Chapter 3) and the normalization factor is 0.75. The curve denoted "p45 (0 cuts)" shows the error performance with a parallelism of 45 which produces no cutting edges. The parallelism of 60 and 90 (represented by "p60" and "p90" respectively) result in a performance loss of 1dB from the reference ("p45"). The dramatic performance loss motivates us to find a solution for

		Short Frame						
S	P_s	1/4	1/2	3/5	2/3	3/4	4/5	5/6
1	360	4	8	0	14	9	9	20*
2	180	1	2	0	4	5	8	13*
3	120	1	1	0	3	3	2	11
4	90	0	1	0	2	2	7	5*
5	72	1	1	0	5	1	1	1
6	60	0	0	0	1	1	2	6
8	45	0	1	0	0	2	2	4*
9	40	1	0	0	1	2	0	3
10	36	0	0	0	1	0	1	1

Table 2.1: Number of DDSM for N=16200

		Long Frame					
S	P_s	1/2	3/5	2/3	3/4	4/5	5/6
1	360	8	32*	12	23*	31*	35*
2	180	4	19	5	10	13	21
3	120	2	16	4	8	15	12
4	90	2	8	2	3	6	13
5	72	0	8	2	3	9	11
6	60	1	6	1	3	5	3
8	45	0	2	0	3	3	5
9	40	2	4	1	3	4	2
10	36	0	4	1	2	2	5

Table 2.2: Number of DDSM for N=64800

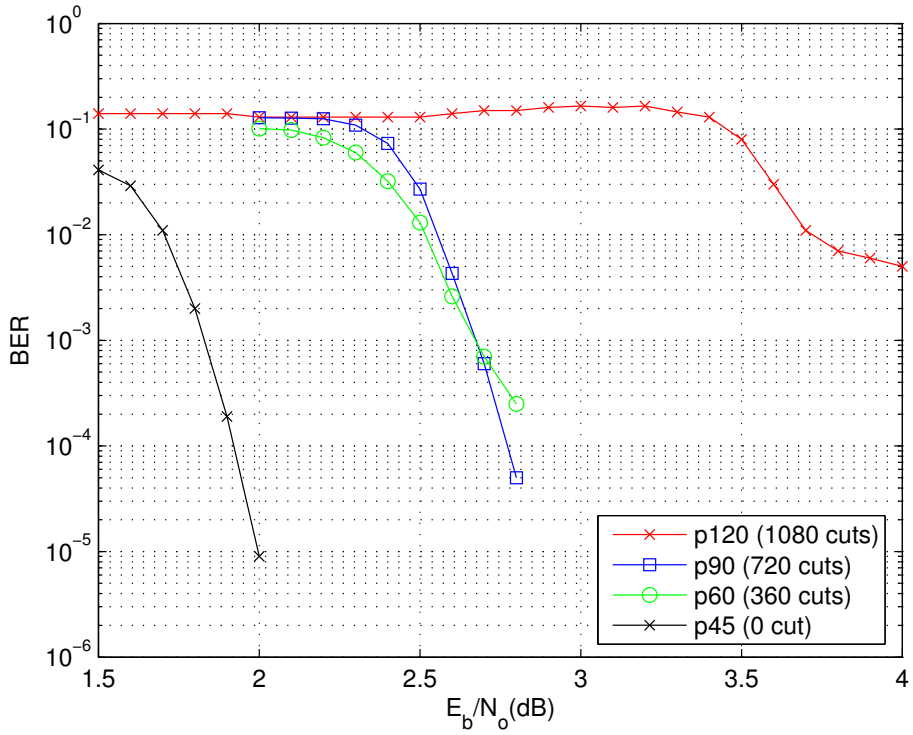


Figure 2.5: BER as a function of the parallelism on a fixed point simulation

the remaining DDSMs in the next section.

2.3 Parity check matrix equivalent

In [15] and [56] a method to transform a parity check matrix by the introduction of dummy VN of degree two is presented. With the help of this method, we build an equivalent matrix without DDSMs.

2.3.1 Principle of the split-extend process

Taking into consideration one parity equation (2.9), it can be split-extended into two equations (2.10) and (2.11) using a dummy VN p_0.

$$v_1 + v_2 + v_3 + v_4 = 0 \bmod 2 \tag{2.9}$$

$$v_1 + v_2 + v_3 + p_0 = 0 \bmod 2 \tag{2.10}$$

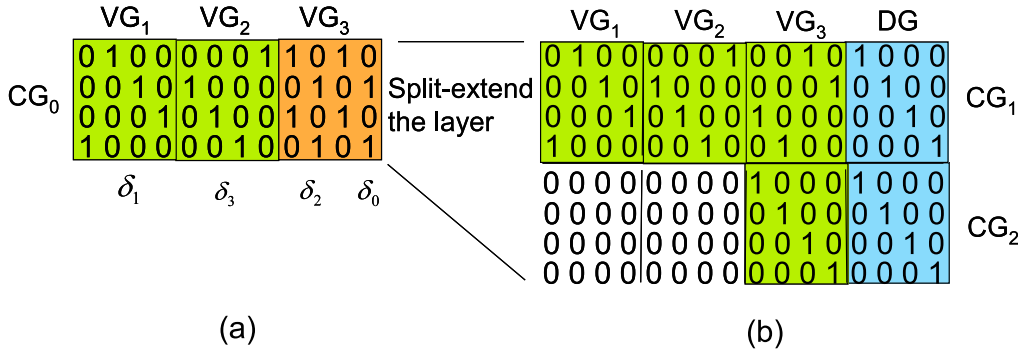


Figure 2.6: Principle of an extended parity check matrix

$$v_4 + p_0 = 0 \bmod 2 \tag{2.11}$$

A dummy VN has an initial LLR value equal to 0. The same splitting process can be applied to a layer. In Figure 2.6, a layer (a) with one DDSM is split-extended into two layers (b). The new equivalent matrix is extended by a a Group of Dummy VN (DG). After the split-extending process, the two layers are without DDSM. It is possible to keep the original matrix for the encoding, which means that there is no change in the architecture of the encoder. However, during the decoding process, the added VNs (dummy VN) are initialized with LLR values of 0 and the new matrix is used for the decoding process. By using the BP algorithm, a flooding schedule and enough iterations, the performance of the extended matrix is equivalent to the original one. The equivalent matrix is less effective for a layered decoder: the dummy VNs of degree two are used only for the communication between the split CNs. The messages going through the punctured VN are updated just one time during one iteration. This means one iteration latency is needed for the message to be sent from one split CN and received by the other. The next subsection presents the results of simulation on a fixed point normalized decoder as described in Chapter 3.

2.3.2 Simulation results

Figure 2.7 shows the simulation result keeping the same conditions as in Figure 2.5 but utilizing extended matrices. The values between round bracket represent the number of extended sub matrice. Note that the simulations are with short frame and without the outer BCH decoder. Simulation with standard frame and BCH correct the error floor and fulfill the standard limit. Simulations with short frame and without BCH have the advantage of faster

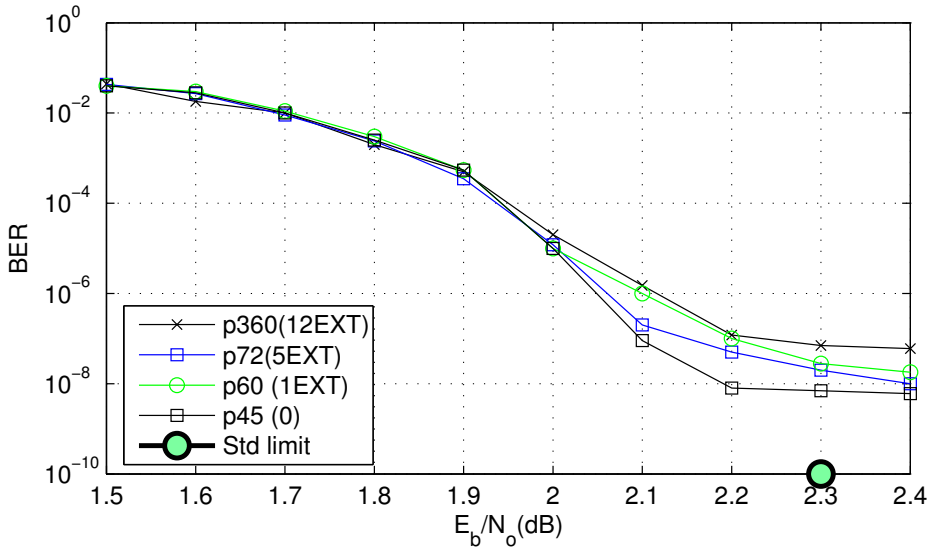


Figure 2.7: BER as a function of the parallelism with extended matrices

simulation. One can observe the performance degradation with the increase of the parallelism.

Figure 2.8 shows a simulation for long frames at parallelism of 40 which is the minimum required parallelism for a 200 MHz pipelined layered decoder to reach the expected 90 Mbps throughput (see Section 4.2). The simulation results are presented for rates 2/3, 3/4, 4/5 and 5/6. The curves represented by dashed lines are the references at a parallelism that gives no conflict (see Table 2.2). The curve represented by solid lines are the results with a parallelism of 40 and extended matrix. An error floor can be observed for rate 2/3, 3/4 and 4/5 this shows that the proposed solution introduces a significant degradation of performances.

2.3.3 Performance improvement

To improve the performance, it is important to find first where the error comes from, which can be difficult to detect as this error occurs on average only once in every 10^8 bits.

Problem description

Let us analyse what does happen exactly on an example. Figure 2.9 is a Tanner representation of the figure 2.6. In this example, VG_3 and VG'_3

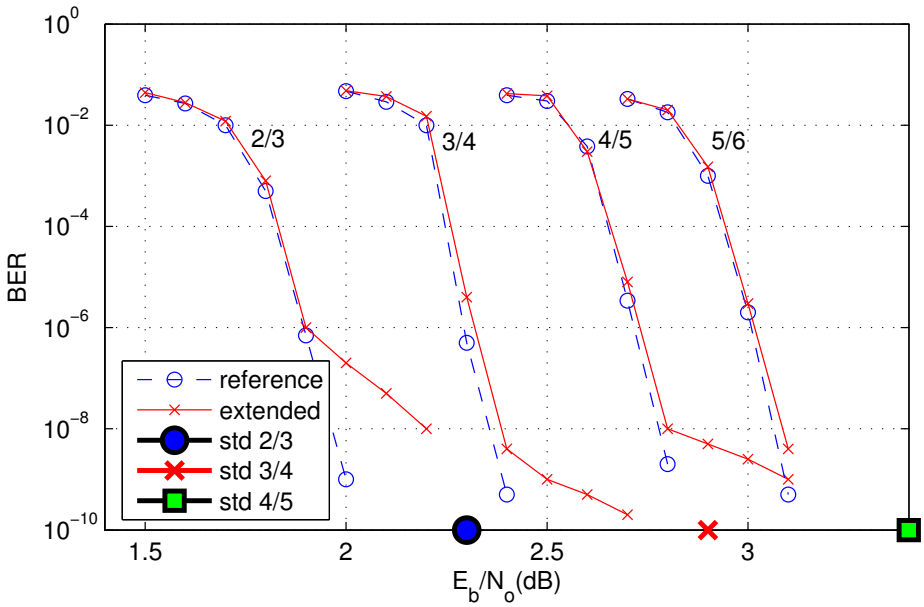


Figure 2.8: BER for standard frames with parallelism of 40 with extended matrices

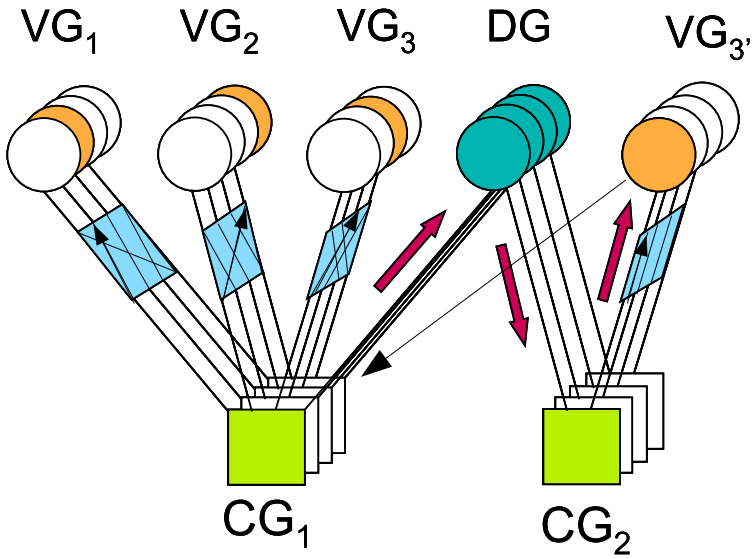


Figure 2.9: Tanner representation of Figure 2.6 (b)

represent the same variable group but they are differentiated in the Tanner graph for better understanding.

Let us follow the message passing evolution of the update of one split CN (The CNs and VNs used during this update are colored in the Tanner graph of Figure 2.9). At the first sub-iteration it_0 and first layer, the Dummy Variable (DV) get the extrinsic message from V_1, V_2 and V_3. Supposing a min-sum algorithm and the V_3 is a minimum value, then V_3 message is transmitted to DV. During processing of the second layer, the V_3 message is transmitted to V_3' through DV. In fact, when the minimum is in V_1, V_2 or V_3, the message passing works exactly like if the parity check were not split. Supposing now that the minimum is in V_3', this minimum is transmitted at the DV only during the second layer processing, and the first layer of it_0 does not take benefit of this message. The first layer will benefit from this message only at the next iteration it_1. This is where the problem comes from: with this matrix configuration, there is one iteration delay for the reading of the V_3' value. The probability that this delay leads to a mistake is very low, but in practice, produces the error floor.

Proposed solution

To sum-up, the root of the problem is that $DV = V_3'(it-1)$ and should be ideally $DV = V_3'(it)$. If during the read process of layer CG_1, we read directly the V_3' value instead of the VD value, then we read the last V_3' updated value and the problem is solved. It is not allowed to write two times at the same address during updating of one layer (it will cause overwrite) but it is allowed to read two times at the same address without problem.

Simulation results

Figure 2.10 is a simulation in similar condition as in Figure 2.8 but for short frames. The simulation results are presented for rates 2/3, 3/4 and 5/6. The new curves are less than 0.05 dB from the reference.

Figure 2.11 is a simulation in similar condition as in Figure 2.8. At the difference of Figure 2.8, the performances are well in the standard requirements, even without BCH.

The values given in Table 2.1 and 2.2 gives the number DDSM and also gives the number of DV that have to be added to solve the conflicts. For a given parallelism, the maximum value of the line multiplied by 360 gives the required number of DV to add to the architecture as an over cost. The SO memory over cost is 2.2 % for a parallelism $P_s = 40$, 3.3 % for $P_s = 60$, 8.8 % for $P_s = 120$ and 20 % for $P_s = 360$.

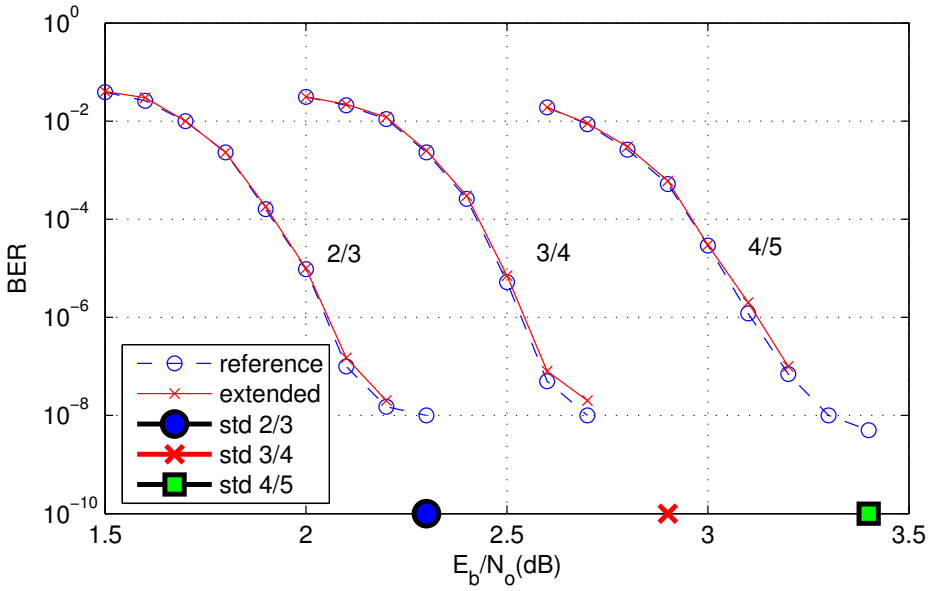


Figure 2.10: BER for short frames with a parallelism of 40 with extended matrices

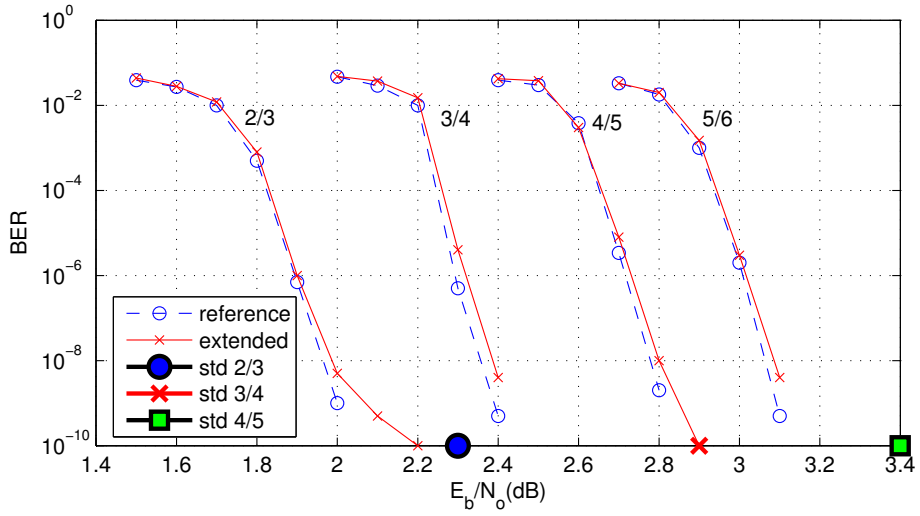


Figure 2.11: BER for long frames with a parallelism of 40 with extended matrices

Another drawback of the solution we proposed comes from a throughput reduction that can become significant in case of high parallelism. The throughput is reduced due to the two added identity matrices for each DDSMs. For example, in case of a rate-5/6 short frame with a parallelism of 40, the 3 DDSMs are removed by adding 6 identity matrices. Comparing with the 2055 identity matrices, this leads to a throughput reduction factor of 2055/(2055+6) which is negligible. To have a simple architecture, we choose the solution to have a constant CN degree (d_c). This means that for added layer of $D_c = 2$, null operation are added to have $D_c = const$. The resulting throughput is now reduced proportionally to the ratio number of DDSMs over the number of layers. In case of a single stream decoder the proposed solution is efficient because the required parallelism is low which means a few DDSM to solve. In case of a multi stream decoder (See Section 4.5), the required throughput is multiplied by the number of streams, which increase the required parallelism and consequently the number of DDSM. With a high number of DDSM, the throughput loss becomes significant and another solution has to be considered.

2.4 Conflict Resolution by Layer duplication

The equivalent extended solution is efficient in term of performance but reduce the decoding throughput. Since the DDSM are solved by the processing of two layers instead of one, the question arises " can we solve the DDSM and further benefits from the second layer processing?". Iterative decoding with replicas [69] shows that it is possible to update a layer two times during an iteration. The idea is to recall the layers that have DDSM and use a write disable when needed.

In this section, the split process (see Section 2.2) is kept to reduce the number of DDSM and then the layers that include conflicts are now simply duplicated and the conflicts are solved thanks to an efficient "on time" write disable of the memory. The advantage is that the added layer solves the DDSM problem, while still taking part in the decoding process.

This section is organized as follows: we first explain the principle of layer duplication and on time write disable. Then we present and solve the problem that arise in the $M_{c\to v}$ messages. Finally, the architecture and performance results are presented.

2.4.1 Conflict resolution by Write Disabling the memory

The principle of conflict resolution by write disabling is to control appropriately the writing of the updated SO values when there is a DDSM. Let us recall first where the memory update conflict comes from (see Section 2.1). In order to facilitate the explanation, we recall equations (2.2) and (2.3).

$$SO_{v_{dd}}^{new_1} = SO_{v_{dd}}^{old} + \Delta M_{c_1 \to v_{dd}} \tag{2.12}$$

$$SO_{v_{dd}}^{new_2} = SO_{v_{dd}}^{old} + \Delta M_{c_2 \to v_{dd}} \tag{2.13}$$

The $SO_{v_{dd}}$ value is updated sequentially two times by equation (2.12) and then equation (2.13). The result of the layer update is:

$$SO_{v_{dd}}^{L_{R1}} = SO_{v_{dd}}^{old} + \Delta M_{c_2 \to v_{dd}} \tag{2.14}$$

To benefit from the two updates, Equation (2.13) should be:

$$SO_{v_{dd}} = SO_{v_{dd}}^{old} + \Delta M_{c_2 \to v_{dd}} + \Delta M_{c_1 \to v_{dd}} \tag{2.15}$$

After a first layer computation, we duplicate the layer, and compute again this layer. By applying Equation (2.12) to the result of the first layer update ($SO_{v_{dd}}^{L_{R1}}$) we obtain Equation (2.15) which is the result we are looking for. Then normally, the second update apply Equation (2.13) and overwrite the result were looking for. To overcome this problem, the SO memory is write disabled during the second update by a write disable, SO is kept unchanged.

In Figure 2.12(a), a sub iteration layer has one DDSM and the SOs are read and written in the same order as they appear in the matrix, from left to right. The result of the diagonal D_2 in the DDSM will overwrite the result of the diagonal D_1 (D_1 in dash line shows the overwrite). Figure 2.12 (b) and (c) is an example where the layer labeled L_{R1} is the first occurrence and L_{R2} is the second occurrence. During the first layer, the result from the left diagonal (D_1) is overwritten but the result of the right diagonal (D_2) is updated. During the second layer occurrence, a write disable signal is generated to disable D_2 update. This time, the result of D_1 is updated. With the recall process associated with an on time write disable, the memory overwrite still occurs but it is corrected thanks to the layer duplication.

2.4.2 Scheduling of the layers

The suspension points between Figure 2.12 (b) and (c) indicate layers between L_{R1} and L_{R2}. These layers are included to avoid memory conflicts due

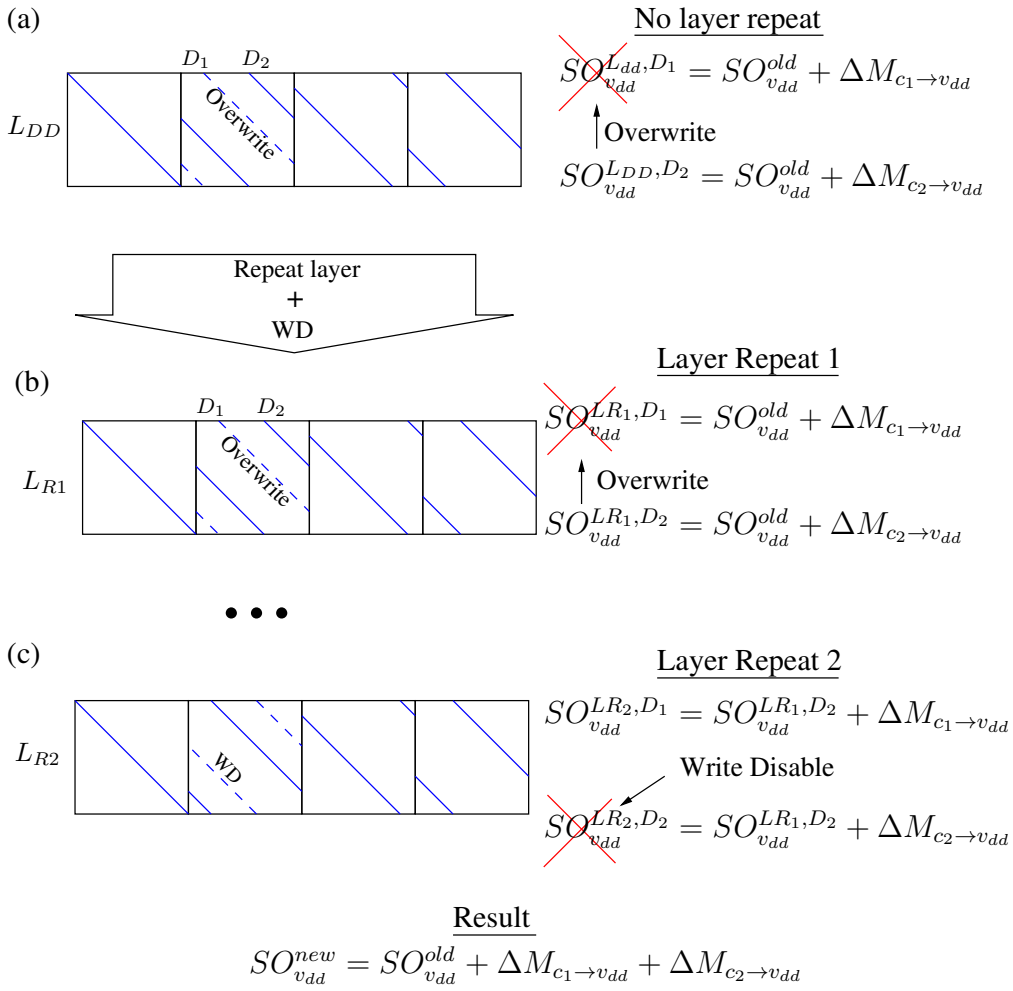


Figure 2.12: repeat layer principle

to pipeline [46] between L_{R1} and L_{R2}. Furthermore, because L_{R1} and L_{R2} use the same SO values, if L_{R2} would be just after L_{R1} the result of L_{R2} is identical to L_{R1} and there is no progress in the iterative process. If between L_{R1} and L_{R2}, the SO values are updated, then L_{R2} benefits from these new updated values. In order to benefit from more updated SO values, an efficient scheduling of the layers would include a maximum number of layers between L_{R1} and L_{R2}. During the next iteration, in order for L_{R1} to benefit from L_{R2} updating, L_{R2} and L_{R1} should also include a maximum number of layers. Ideally, L_{R1} and L_{R2} should be separate by half the number of layers.

This repeat layer principle is quite simple. Nevertheless while the SO Memory is write disabled, the $M_{c\to v}$ is still updated which will lead to errors. In terms of hardware implementation, specific mechanisms are required in order to keep coherence between $M_{c\to v}$ memory and SO memory.

2.4.3 Write disabling in the $M_{c\to v}$ memory

During one sub iteration with one DDSM, from equations (2.12) and (2.13), the SO_v^{new2} will overwrite the SO_v^{new1} value. The edge of the Tanner graph connecting c_1 to v_{dd} is virtually absent in the decoding process. Nevertheless, at the next iteration from (1.15), the $M_{v\to c_1}$ is given by:

$$M_{vdd\to c_1}^{it} = SO_v - M_{c_1\to v_{dd}}^{old} \tag{2.16}$$

The edge $c_1 \to v_{dd}$ has been cut during the previous iteration, this means that $M_{c_1\to v_{dd}}^{old}$ should not provide the $M_{c_1\to v_{dd}}^{new}$ value computed during the previous iteration. A possible implementation is to Write Disable (WD) the $M_{c\to v}^{MEM}$ memory when storing $M_{c_1\to vdd}^{new}$. This process keeps $M_{c\to v}$ at address of diagonal one (D_1) in the $M_{c\to v}$ memory at the previous value.

Let us now focus on the DDSMs in the repeated layers in Fig 2.12 (b) and (c). During the L_{R1} (b) computation, the $M_{c\to v}$ memory at address (D_1) is write disabled and during the L_{R2} (c) computation, the $M_{c\to v}$ memory at address (D_1) is write disabled.

Write disable algorithm

There is a need for an on time WD of the SO memory and the $M_{c\to v}$ memory in function of the layer occurrence (L_{R1} and L_{R2}) and the diagonal number in the DDSM (D_1 and D_2). This is summed-up in Algorithm 1.

This algorithm is efficient for no compressed $M_{c\to v}$ memory where each edge has an address in the memory. In case of a min-sum algorithm, there is no more access to D_1 and D_2 because all the edges of one layer are compressed and stored in one address. This problem is solved in the next subsection.

Algorithm 1 Write disable algorithm

```
if LR1&D1 then
  WD M_{c→v} MEMORY ← 1
  WD SO MEMORY ← 1
else if LR2 & D2 then
  WD M_{c→v} MEMORY ← 1
  WD SO MEMORY ← 1
else
  WD M_{c→v} MEMORY ← 0
  WD SO MEMORY ← 0
end if
```

2.4.4 Write disabling the $M_{c\to v}$ memory when a Min-Sum algorithm is used

When a Min-Sum algorithm is used, The d_c $M_{c\to v}$ messages linking a CN are compressed in the min, $submin$, ind_{min} and $sign$ values(see Section 1.2.1). The goal of Algorithm 1 is to WD one specific $M_{c\to v}$ message which is not possible on compressed $M_{c\to v}$ messages. One solution is to add two additional memories for $M_{c\to v}(D_1)$ and $M_{c\to v}(D_2)$. The $M_{c\to v}(D_1)$ memory is updated only when there is no overwrite of D_1 which means second layer and first diagonal.The $M_{c\to v}(D_2)$ memory is updated only when there is no WD of D_2 which means first layer and second diagonal. The algorithm 1 is updated in Algorithm 2.

During the reading of the $M_{c\to v}$ memory, the $M_{c\to v}(D_1)$ memory is selected every time a D_1 is detected (same process for D_2). The next challenge is to implement efficiently the new algorithm.

2.4.5 Simulations and memory size results

Simulation results

Figure 2.13 illustrates simulation results for a rate-2/3, short frame with a normalized min-sum algorithm and 25 iterations. By reordering the matrix [45], [46] and Section 2.2, it is possible to greatly reduce the number of DDSM (at the cost of parallelism reduction). For example, a parallelism $p = 72$ gives 5 DDSMs and produce a curve closer to the reference than p=360. The reference curve is without conflict and a parallelism $p = 45$. The presented simulation shows better results than in [45] where a matrix expending process is used. Note that we used short frame without BCH as reference curve for faster simulation giving results above the standard

Algorithm 2 Write disable algorithm for min-sum

if $Layer = L_{R1}$ **then**

  if D_1 **then**
    WD SO MEMORY $\leftarrow 1$
  else if D_2 **then**
    WE $M_{c\rightarrow v}(D_2)$ MEMORY $\leftarrow 1$
  end if
else if $Layer = L_{R2}$ **then**

  if D_1 **then**
    WE $M_{c\rightarrow v}(D_1)$ MEMORY $\leftarrow 1$
  else if D_2 **then**
    WD SO MEMORY $\leftarrow 1$
  end if
else
  WD SO MEMORY $\leftarrow 0$
  WE $M_{c\rightarrow v}(D_2)$ MEMORY $\leftarrow 0$
  WE $M_{c\rightarrow v}(D_1)$ MEMORY $\leftarrow 0$
end if

requirement. If standard frame and BCH were used, the performance would be in the standard requirements.

To compare the curve at a constant throughput, in Figure 2.14 the number of iteration is reduced in function of the number of added layers. The reference curve is without conflict and 30 iterations. With a parallelism of 120 (3 DDSM), 3 layers are repeated over the 15 layers. To keep a constant throughput, the number of iterations is reduced by a factor $15/(15+3)$, which gives 25 iterations. With a parallelism of 360 (14 DDSM), the number of iteration is reduced by $15/(15+14)$ which gives 16 iterations and the curve is at only 0.1 dB away from the reference.

Memory size

When using compressed $M_{c\rightarrow v}$ memory, there is a need to store the $M_{c\rightarrow v}$ values for every DDSM. The Memory over cost in bits is equal to two times the $M_{c\rightarrow v}$ word size times the number of DDSM times the parallelism P. In case of a parallelism of 360 the memory over cost becomes significant i.e. 126 Kbits for long frame and rate 5/6.

When using a not compressed $M_{c\rightarrow v}$ memory, there is no more need of

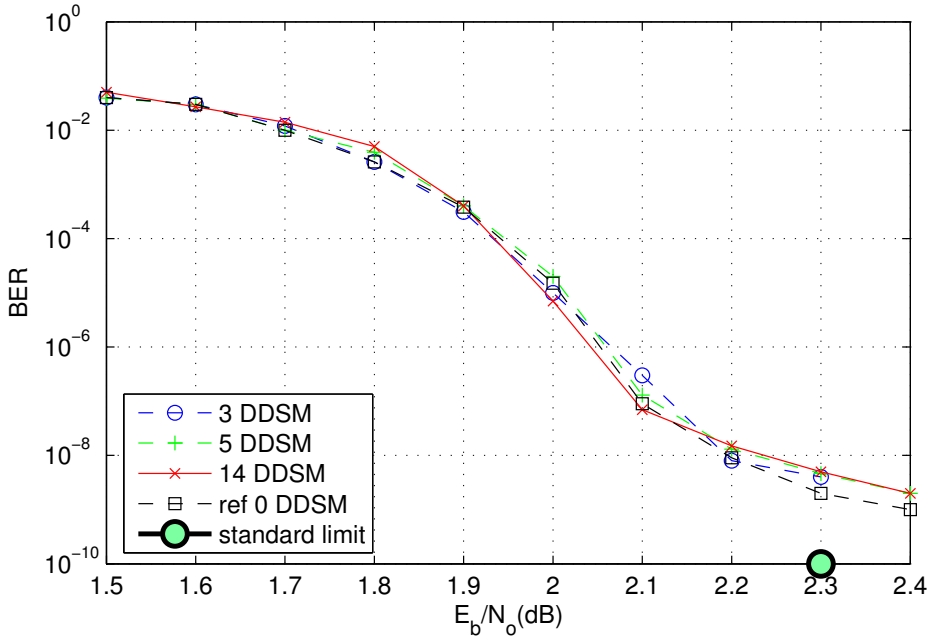


Figure 2.13: BER for short frame, 25 it, r=2/3, N=16200, without BCH

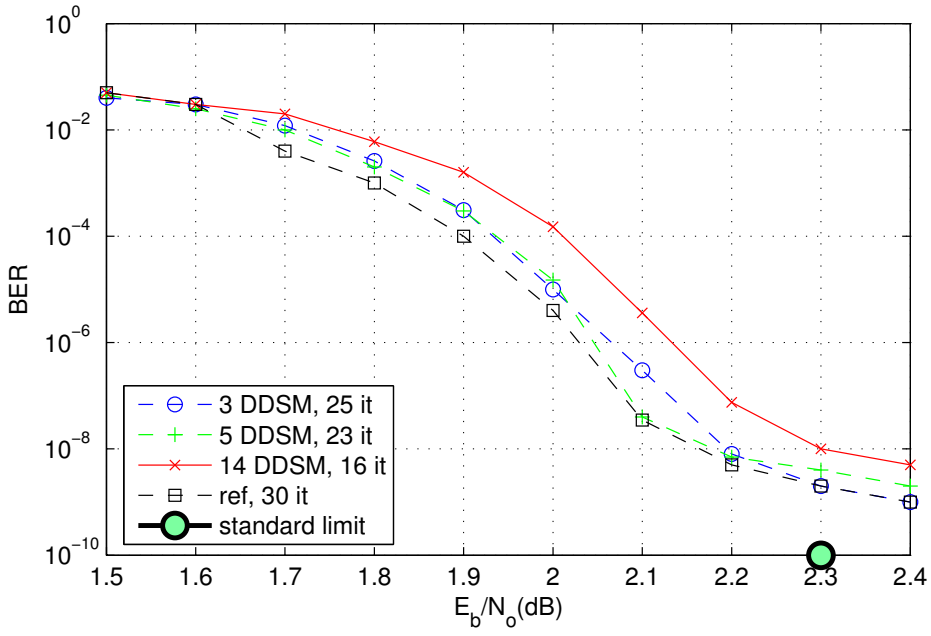


Figure 2.14: BER for short frame, constant throughput, R=2/3, N=16200, without BCH

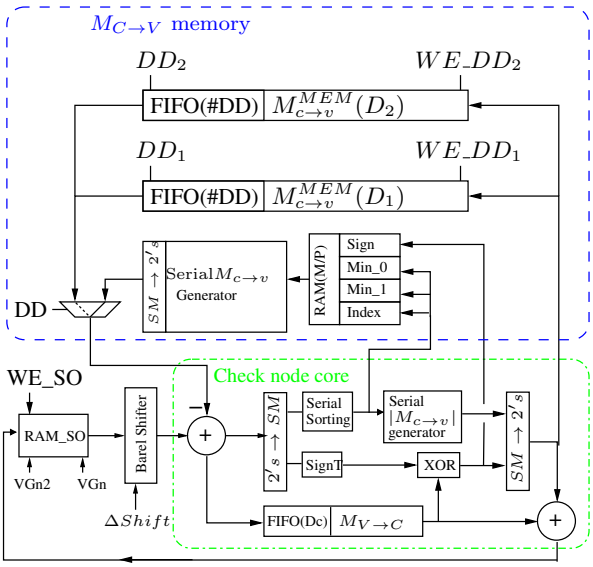


Figure 2.15: write disable architecture

adding specific memory. Considering the memory size optimization described in [44] and in Chapter 3, a compressed version requires 875 Kbits and a non compressed version requires 1100 Kbits. Adding 126 Kbits on the compressed side, the non compressed solution still leads to a 10 % memory over cost but gives alternative to the Min-Sum algorithm. An alternative to Min-Sum algorithm can be interesting to improve performance and to solve the error floor problem for the low code rates.

2.4.6 The write-disable architecture

In this subsection, we describe the three main modifications added to the layered decoder architecture 1.3.4 to implement the WD algorithm.

Layer duplicating

In the architecture overview (see Section 1.3.4, the $M_{c\to v}$ memory can be made of a FIFO memory as a $M_{c\to v}$ update occurs only once during one iteration. In case of repeated layers, the $M_{c\to v}$ are updated more that once and the $M_{c\to v}$ read and write must be at the same address as during the first call. This can be implemented using a RAM and an ad hoc address generator. From the counter and the ROM_{dc}, the layer number signal is given. The ROM_{Mcv} gives a constant address when a layer is repeated.

XQ5VLX85	LUT	LUT RAM	BRAM
Node Processor	143	2	0
sorting	37	0	0
gen $m_{c\to v}$	34	0	0
fifo $m_{v\to c}$	12	2	0
$m_{c\to v}$ memory	46	0	3
Total 1 node	189	2	3
Total 45 nodes	8586	90	135
Control	667	3	0
block SO RAM	360	0	22
Channel RAM	48	0	25
Barrel shifter	945	0	0
Total	11005	93	182
Percentage [%]	5	1	50

Table 2.3: Synthesis Results for DVB-S2 LDPC decoder

Cutting edge and repeat layer detection

The cutting-edge detector principle relays on the detection of DDSM. When there is a DDSM, during the serial read of the VNGs, two equal $VNGn$ values follows. This can be easily detected at the output of ROM_{VNGn} where the base matrix is stored. A repeat layer detector is implemented by using a dedicated ROM. In this ROM, for each layer is stored if the layer is a repeat layer and if the layer is the first or the second occurrence.

Knowing the status of a layer (normal, L_{R1} or L_{R2}) and the status of an IM (normal, D_1 or D_2), it is possible deducted from the Algorithm 2 the states of the WD signals (WD_{SO} , WE_D_1 and WE_D_2)

2.4.7 Synthesis results on FPGA

The architecture presented in Figure 2.15 with finite precision options described in Section 3 was synthesized on a Virtex-V Pro FPGA (XQ5VLX110) from Xilinx, for validation purposes. The system decodes long frames of code rate 2/3. Table 2.3 reports the hardware resources required. The clock frequency is 200 MHz, the average number of iterations is 20 and the throughput is 90 Mbit/s, which allows the decoding of two simultaneous High-Definition Television (HDTV) streams.

2.4.8 Conclusion

To deal with the memory conflict problem, we proposed a solution based on repeating the layers with DDSM. The repeating process is associated with an efficient write disable of the memory to overcome the conflict problem. This technique shows in worst case performance at 0.1db from the reference without throughput loss. The implementation area benefits from the use of only one barrel shifter and a simple memory access at the price of a slight memory increase and control over cost. This technique shows, in worst case, performance at 0.1db from the reference without throughput loss. The next step is to solve the conflicts due to the pipelining.

2.5 Memory update conflicts due to pipeline

Many of the current LDPC implementations of DVB-X2 or WiMAX [60] standard use the so-called layered architecture combined with pipeline [62, 54, 14, 57]. However, the pipeline process may introduce memory update conflicts.

In [53] and [8] the authors present a solution based on the computation of the variation (or delta) of the SO metrics to allow concurrent updates. The computation of this SO update ($SO^{new} = SO^{old} + delta$) is efficient to solve this kind of conflict but needs either a costly additional memory access or an increase in the clock frequency by a factor of two. In [62], the use of idle time is proposed to deal with the conflicts. However this solution decreases the throughput. The scheduling of the SO in [54] reduced the use of idle time by using CNP able to deliver its outputs values in a different order than its input values, which increase the complexity of the CNP architecture. Finally, we should also note that [14], [54] propose an appropriate scheduling of the check node as a solution to avoid pipeline conflict but in none of them, the idea is fully developed.

In this section, we focus on the conflicts due to the pipelining of a layered decoder and we propose solutions with an example on the DVB-T2 matrices. In a first step the splitting process (see Section 2.2) solve by itself a part of the pipeline conflicts. In order to find an efficient scheduling to solve the remaining conflict, we show that the research of an efficient scheduling is equivalent to the well known "Traveling Salesman Problem". Thus, all the numerous methods described in the literature to solve the former problem can be used for our scheduling problem. In this section, we present scheduling results using a genetic algorithm.

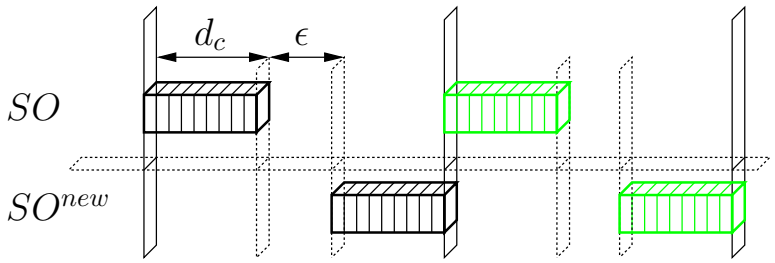


Figure 2.16: Chronogram of a non pipelined CNP

2.5.1 Non pipelined CNP

The chronogram in Figure 2.16 illustrates a non-pipelined CNP. The CNP first reads the SO. Then after a given number of clock cycles ϵ, i.e. the CNP latency, the CNP writes back the result of the calculation. We can see on this chronogram that the CNP starts to read the new set of variable nodes $v_{c_{i+1}}$ only when all the previous v_{c_i} have been calculated. The corresponding throughput is given by:

$$D_1 = \frac{K.F_{clk}}{(2d_c + \epsilon).\frac{M}{P}.N_{it}} \quad bit.s^{-1} \tag{2.17}$$

where N_{it} is the number of iterations to decode a codeword, M is the number of CN, P is the number of CNPs working in parallel, d_c is the average number of VNs linked to a CN, F_{clk} is the clock frequency and K is the number of information bits in a codeword.

2.5.2 Pipelined CNP

Pipelining allows a more efficient use of the CNP and an increase of the throughput [62], [4] and [54]. The pipelining consists in reading the v_{c_i} of one sub-iteration while writing on $v_{c_{i-1}}$ the result of the previous sub-iteration. This means that as soon as the reading of one sub-iteration is finished, a new one is started. The chronogram is given in Figure 2.17 and the corresponding throughput is given by the following equation:

$$D_2 = \frac{K.F_{Clk}}{d_c.\frac{M}{P}.N_{it} + d_c + \epsilon} \quad bit.s^{-1} \tag{2.18}$$

The pipelined architecture offers at least two times greater throughput compared to the non pipelined one: $\frac{D_2}{D_1} \approx 2 + \frac{\epsilon}{d_c}$. We will see in the next subsection that this architecture can lead to memory conflicts.

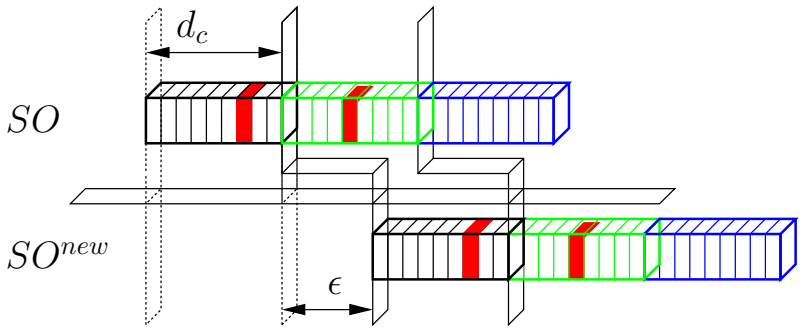


Figure 2.17: Chronogram of a pipelined CNP without idle time

2.5.3 The problem of memory update conflicts

In this subsection, we describe the memory update conflicts and we differentiate two types of conflicts.

"Type $i+1$" memory conflict: In Figure 2.17 a common variable SO_{com} (filled square) is used in two successive sub-iterations. During the second sub-iteration, the SO_{com} is still not updated from the previous sub-iteration and the result of the current sub-iteration will overwrite the result of the previous sub-iteration. This is also known as a 'cutting edge' problem because it is equivalent to a cut in an edge of the Tanner graph representation of the matrix. Let vg_{cg_i} be the set of all the vg_i connected to cg_j. During pipelining, the P CNPs write on vg_{cg_i} while the P CNPs read on vg_{cg_j}. The layers cg_i and cg_j can work one after the other without memory access conflict when the two groups don't share any common variable [1]. Mathematically speaking, this constraint can be expressed by:

$$vg_{cg_i} \cap vg_{cg_j} = \emptyset$$

For example, we can check in Figure 2.2 that there is no vg_i in common between the set of vg_i linked to the group of check node number one cg_1 and eleven cg_{11}:

$$\{vg_{cg_1}\} \cap \{vg_{cg_{11}}\} =$$

$$\{1,2,12,16,19,24,31,45\} \cap \{3,11,13,15,17,21,26,40,41\} = \emptyset$$

Thus the decoding of cg_1 and cg_{11} can be processed consecutively without memory update conflict. To avoid memory update conflicts, we have to take care that at any time, two sub-iterations does not share any SO in common.

[1] As mentioned in [54], if $dc > \epsilon$, it is still possible to avoid memory conflict between two groups sharing the same common variable SO_{com} by an appropriate scheduling of the SO_{com} inside the two consecutive layers.

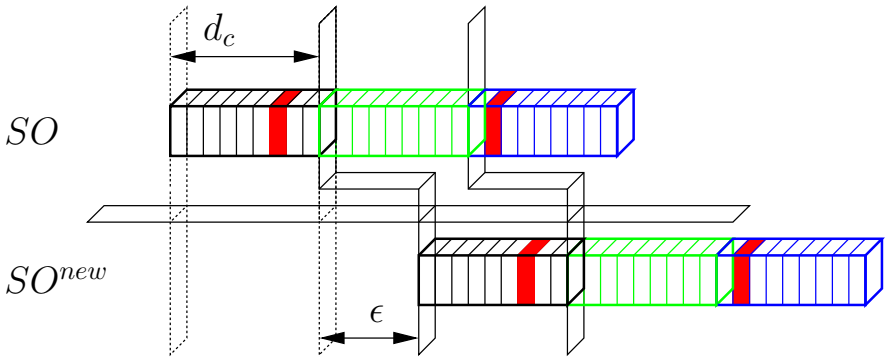


Figure 2.18: Conflict due to pipelining at $i+2$

"Type $i+2$" memory conflict: Figure 2.18 illustrates that the same consideration must be taken with the $i+2$ sub-iteration because of the latency ϵ. In this figure, one value of SO generated by layer i is written in memory after it is read by layer $i+2$. This situation also leads to a memory conflict and can be avoided by appropriate scheduling of the layers.

2.5.4 Conflict reduction by group splitting

Section 2.2 explaines how to split an identity matrix in multiple sub-identity matrices. This process was used to reduce the number of memory update conflicts due to the structure (Section 2.1). Again, the split process is used to solve memory update conflict but the causes and the reasons are different.

The Figure 2.4 illustrates the new base matrix after splitting by a factor of 2. We can see that the new base matrix is sparser than the base matrix in Figure 2.2 (in terms of identity matrix density). In fact, the number of identity matrices increases by 2 while the size of the base matrix is increased by 2×2. Increasing the split decreases the risk of memory conflicts due to pipelining. An appropriate scheduling of the layers can solve the remaining conflicts.

2.5.5 Conflict resolution by scheduling

A schedule provides timing information about a series of arranged events. To avoid the cutting edge conflict, we explore the scheduling of the layers. After defining the scheduling strategy, we show that this problem is an instance of the well known "Traveling Salesman Problem". This problem can be efficiently solved by a Genetic Algorithm (GA).

Scheduling of the layers

The schedule is done in the set of groups of CNs $cg = \{cg_1, cg_2, ..., cg_{mg}\}$. We define a schedule sequence index π, where π is one permutation in the set $\{1, 2, ..., mg\}$. The number of conflicts due to the pipelining between two check node groups cg_i and cg_j is given by equation (2.19).

$$c(i,j) = |vg_{cg_i} \cap vg_{cg_j}| \tag{2.19}$$

The number of conflicts after one full iteration using scheduling π is given by equation (2.20).

$$c_{it}(\pi) = \left(\sum_{i=1}^{i=mg-1} c(\pi(i), \pi(i+1)) \right) + c(\pi(mg), \pi(1)) \tag{2.20}$$

where the second term is the cost between the last layer of an iteration and the first layer of the following iteration. We have to find the optimal permutation π_{opt} that gives the smallest number of conflicts. This can be translated into an optimization problem which consists in the minimization of the cost function:

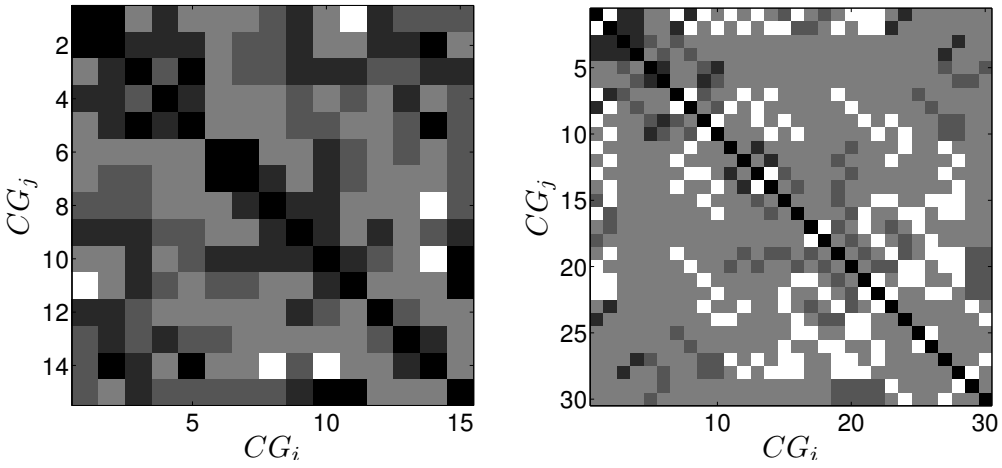
$$\pi_{opt} = \arg\min\{c_{it}(\pi), \pi \in \Pi\} \tag{2.21}$$

where Π is the group of all the possible permutations of π. The use of graph theory is dedicated to solve this kind of problem.

The Traveling Salesman Problem (TPS)

Finding the schedule to avoid cutting edge is described by the minimization problem (2.21). An equivalent formulation in terms of graph theory is: given a complete weighted graph (where the node would represent a group of CNs, and the number of cutting edge would be the cost of the edge), find a Hamiltonian cycle with the least cost. An Hamiltonian cycle consist in visiting each node exactly once and also return to the starting node. This problem is also known as a TSP [38]. The TSP statement is as follows: given a number of cities and the cost of traveling from one city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the starting city. In our case a town is a group of check node and the traveling cost is the number of cutting edges (2.19).

The first step is to build the cost matrix $H_c = \{H_c(i,j) = c(i,j), i,j \in [1, mg]^2\}$. This matrix gives the number of cutting edges for each possible couple of groups of CN cg_i and cg_j. The cost matrix for a rate 2/3 short frame is illustrated in Figure 2.19(a). On this graphical representation, a



(a) Without split

(b) With a split factor of 2

Figure 2.19: Cost matrix

white square is for no cutting edge and a grey square means one (light grey) or more (darker grey) cutting edges.

We can see in Figure 2.19(a) that there are only three couples $cg_{1,11}$, $cg_{8,14}$ and $cg_{10,14}$ that can perform consecutively without conflicts. Figure 2.19(b) shows the cost matrix after a split factor of 2. The new matrix offers more possible couples without memory conflict (from 3% to 20% after splitting by two). The split process gives fewer cutting edges and a greater degree of freedom for scheduling.

The problem of trying all permutations ($|Pi| = mg!$) and selecting the minimum cost (2.21) is NP hard in $O(mg!)$. For a long frame of rate $^1/_4$ and split 4, the number of cities are 540. Thus a suboptimal or heuristic algorithm is needed to solve the problem.

Principle of Genetic Algorithm

Genetic algorithms [32, 38] use techniques inspired by evolutionary biology such as inheritance, mutation, selection and crossover. The genetic algorithm process is summarized in Figure 2.20. First, many solutions are randomly generated to form an initial population. Then individual solutions are selected through a two round tournament selection using the fitness function given by equation (2.20). The next step is to generate the 'children' through a two point crossover between solutions previously selected. The last step is the mutation of the children for diversity purpose by swapping two randomly

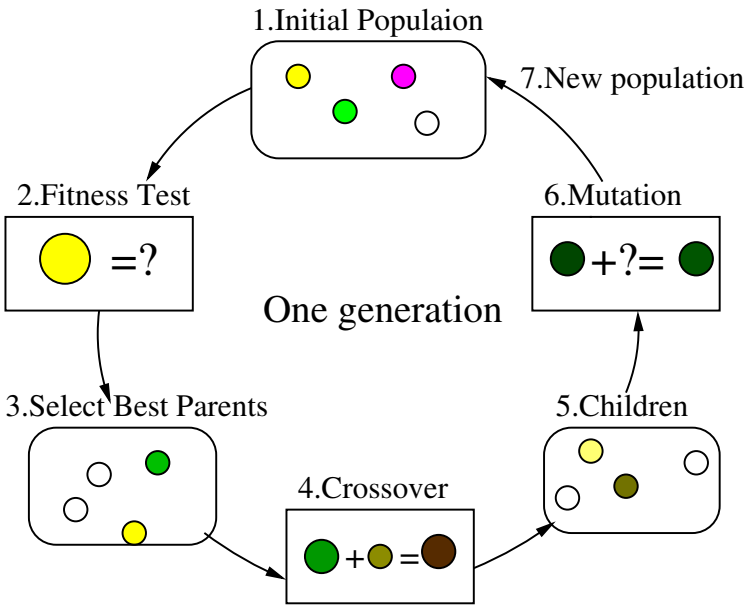


Figure 2.20: Genetic algorithm

chosen cities. This generational process is repeated until a null cost is found for an individual solution or a fixed number of generations is reached.

The result of the genetic algorithm applied to the scheduling problem is presented in the next section.

Simulation results

In this section, we present the result of the genetic algorithm. The maximum number of generation and the initial population were set to 1000. Every solution was obtained in less than 15 minutes on a standard desktop processing unit.

In order to take into account the "type $i+2$" cutting edge defined in section II.D, i.e $vg_{cg_i} \cap vg_{cg_{i+2}} = \emptyset$, we modify the cost function c_{it} defined in (2.20) as:

$$c'_{it}(\pi) = \alpha c_{it}(\pi) + \sum_{i=1}^{mg} c(\pi(i), \pi((i+2) \bmod mg)) \tag{2.22}$$

where $\alpha \in N^+$ and is high enough to give an absolute priority to the $i+1$ conflicts against the $i+2$ conflicts. The next subsection gives some results using a genetic algorithm to find an efficient schedule.

Table 2.4 and Table 2.5 present the results found for the DVB-T2 LDPC decoder. In these tables, a '0' means that the genetic algorithm found no

		Code rate						
S	P_s	1/4	1/2	3/5	2/3	3/4	4/5	5/6
1	360	1	1	0	0	0	0	0
2	180	2	2	1	0	1	0	0
3	120	2	2	1	1	1	1	0
4	90	2	2	1	1	1	2	1
5	72	2	2	1	1	2	2	1
6	60	2	2	2	2	2	2	1
8	45	2	2	2	2	2	2	1
9	40	2	2	2	2	2	2	2

Table 2.4: Scheduling solutions for short frames

		Code rate					
S	P_s	1/2	3/5	2/3	3/4	4/5	5/6
1	360	2	1	1	1	0	0
2	180	2	1	2	1	1	0
3	120	2	1	2	2	1	1
4	90	2	1	2	2	1	1
5	72	2	1	2	2	1	1
6	60	2	2	2	2	2	1
8	45	2	2	2	2	2	2
9	40	2	2	2	2	2	2

Table 2.5: Scheduling solutions for long frames

solution that avoids "type i+1" conflict. A '1' (respectively '2') means that it found a scheduling solution without type $i+1$ conflict (respectively without type $i+1$ and type $i+2$ conflicts). We can check that for $P_s = 40$, there are schedules without conflicts at $i+2$ for all code rates and frame types. Note that, after a scheduling at $i+1$, the remaining conflicts due to the latency ϵ can be avoided using a scheduling of the vg_i inside the layers [54]. This option allows a parallelism of up to 120 for long frames and 90 for short frames.

Scheduling of the identity matrices

During the sorting process of the min-sum algorithm in equation (1.8), the arriving order of the VG have no incidence in the result, so it is possible

to reorder the VG in a layer without consequences. Table 2.4 and Table 2.5 show that for high parallelism there is no layer scheduling that solve equation 2.20 (type $i+1$ conflict). Even if two successive layers have a common VG, a conflict doesn't necessary occurs as in figure 2.17. In the figure 2.17, if the common VG (colored square) of the first layer is read at the start of the layer, and the common VG is at the end of the second layer, then no conflict occurs. To generalize, if $\epsilon < dc - 1$ then a solution exist by reordering the IM in the layers. If $\epsilon > dc - 2$ then there is no solution for type $i+1$ conflict and idle time have to be added to the reordering of the IM to avoid conflict. Moreover, if $\epsilon > dc - 2$, then there are possibility of conflict between a layer i and layer $i+2$ as in Figure 2.18 (type $i+2$ conflict), this conflict can be solved by identity matrix scheduling.

2.5.6 Conclusion

Pipelining a layered decoder doubles the throughput but leads to memory conflicts. The split process reduces the parallelism and creates sparser base matrices. Using the new base matrices, schedules without conflicts can be found. Due to the huge amount of combinations, a genetic algorithm is used to find the best schedule. This algorithm finds schedules that avoid conflicts with the next sub-iteration $(i+1)$ and with the second next sub-iteration $(i+2)$. Combined with a scheduling of the identity matrices in the layers, the whole scheduling process is able to solve all conflicts due to pipeline. The proposed solution only requires an efficient scheduling and requires no change or added patch to the layered architecture. Although this section explains the process for matrices defined by the DVB-T2 standard, the same process can be used for structured matrices such as the ones defined by the WiMAX standard.

2.6 Combining layers duplication and scheduling

Section 2.4 solves efficiently memory conflict due to the matrices structure. Section 2.5 solves efficiently memory conflicts due to the pipelining. Can we combine the two methods to solve all the memory update conflicts? The split process explained in Section 2.2 is a common tool as a first step for the resolution of the two causes of conflicts. The split conflict is compatible with the two methods. Then in a second step, the resolution of pipe conflict is solved by minimizing the cost of an Hamiltonian cycle (process all the

layers only once during an iteration and return to the starting layer). The second step to solve conflicts due to the structure is to duplicate layers. The consequence is that the iteration cycle is not any more Hamiltonian because layers are repeated during one iteration. Thanks to the flexibility of the Genetic Algorithm (GA) used to solve the scheduling problem, the data of the problem has just to be updated with new layers (the duplicated layers) without any changes in the GA algorithm.

2.7 Conclusion

In this chapter we considered the problem of memory update conflicts due to the DVB-S2, -T2 and -C2 matrices structure and conflicts due to the use of a pipelined layered decoder. Conflicts due to the structure has been solved thank to two innovative solutions and Conflicts due to pipeline has been solved by an efficient scheduling of the layers. The *split procees* associated with a *layer repeat* architecture and an efficient *layer scheduling* can be efficiently combined for a memory update conflict free solution. Now that the memory conflict problems are solved, it is possible to think on reducing the implementation cost by optimizing the architecture.

If you want a thing done well,
do it yourself.

Napoléon Bonaparte
(1769-1821)

3
Memory optimization

Summary :

In this chapter, the optimization of the memory size is studied. The DVB-S2 standard is considered to compare results with the literature. Our work can also be applied to the DVB-T2 and -C2 standard, the Wi-Fi and WiMAX LDPC standards or more generally to any layered LDPC decoder. The first step concerning the memory reduction is the choice of a sub-optimal algorithm. The already well-known Min-Sum algorithm [24] and its variants significantly reduce the memory needs by the compression of the extrinsic messages. Another way to reduce the memory needs is to limit the word size by saturation. In the state-of-the-art, the way the SO and the extrinsic messages are saturated is rarely explicitly explained. In this chapter, we provide first some discussion on efficient saturation of the extrinsic messages and the SO values. To complete the discussion, we also introduce a methodology to optimize the size of the extrinsic memory. In a second step, still is the field of memory optimization, as Dual port RAM is much more costly than Single port RAM, methodologies are given to implement the layered decoder by using only single port RAM.

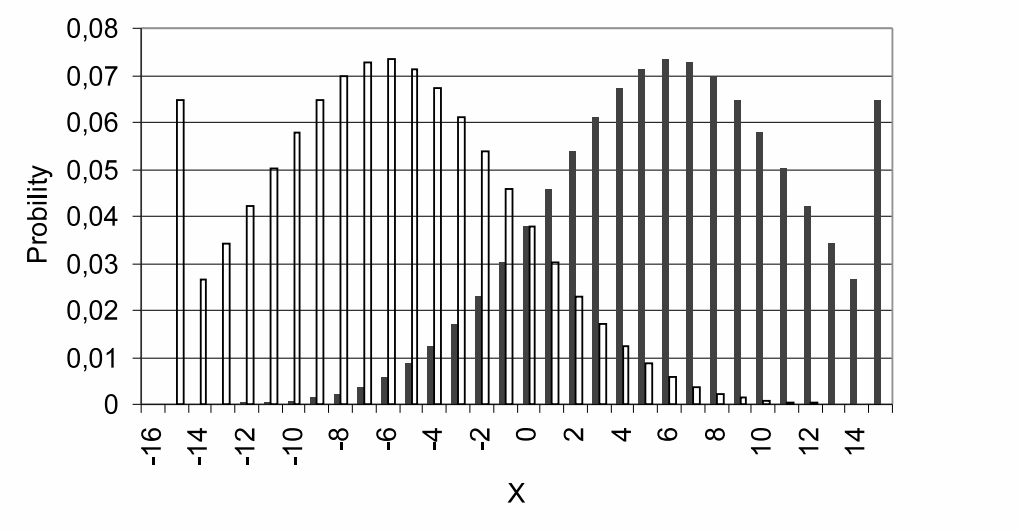


Figure 3.1: Resulting distribution of a quantified BPSK modulation

The area occupied by the memory of an the implementation of the DVB-S2 LDPC decoder is around 75 %. Thus area optimization leads to the optimization of the memory. The memory size depends on tree factors: the quantification, the way it is implemented and the technology used. The saturation process is first studied as a mean to reduce quantization.

3.1 Saturation of the stored values

A SO value is the sum of the channel LLR with all the incoming extrinsic messages. Considering the case of an LDPC code of the DVB-S2 standard, the maximum variable node degree (d_v) is 13. As an exemple, considering that the channel LLR and the $M_{c\to v}$ are quantized on 6 bits, the SO values should be quantized on at least $6 + \lceil \log_2(13) \rceil = 10$ bits to prevent overflows. However, to avoid prohibitive word size, efficient saturation of channel LLRs lead to a reduction of the overall quantization. Then a saturation of the SO values and the $M_{c\to v}$ are considered.

3.1.1 Channel LLR saturation

For floating point simulation, it is known that the decoders using the Normalized Min-Sum algorithm are not sensitive to scaling in the LLR_{in} values. During the initializing process, the equation $LLR_{in} = 2y/\sigma^2$ can be simplified

to $LLR_{in} = y$, saving the need to compute the variance.The received y value is quantized in a way to have integer values at the input of the decoder. We assume here that the quantized value of y denoted by LLR_q is represented on n_{LLR} bits and the quantification function is defined as:

$$LLR(y)_q = \left\lfloor sat(y, R) \times \frac{2^{n_{LLR}-1} - 1}{R} + 0.5 \right\rfloor, \tag{3.1}$$

where $sat(a, b) = a$ if a belongs to $[-b, b]$ and $sat(a, b) = sign(a) \times b$ otherwise. The R value is the interval range of quantization (y is quantized between $[-R, R]$) and represent also the saturation threshold value. Considering the BPSK modulation, we saturate y at $R = 1 + \beta$.

Figure B.3 shows the Probability Density Function of a quantized BPSK modulation (-1 and +1) that is perturbed by an Additive White Gaussian Noise (AWGN) of variance $\sigma = 0.866$ (corresponding to Eb/No = 2 dB). The channel is quantized on 5 bits and the saturation threshold is $R = 1 + \beta = 2,47$. The distribution filled in black shows the +1 offset, and the unfilled distribution is the -1 offset. The quantized distribution varies from $LLR_q^{min} = -(2^4 - 1)$ to $LLR_q^{max} = 2^4 - 1$. The problematic is to find the saturation threshold providing the best performance for for a given number of bits of quantization. If the saturation threshold is low, then the information given by the tail of the Gaussian curve are saturated. If the threshold is high then the quantization error increases.

The saturation limit $1 + \beta$ can be calculated so that the proportion of saturated values is equal to the average proportion of the other values. The average proportion of a given value is $1/(2^{n_{LLR}} - 1)$(probability of a value in an uniform distribution). On the other side of the equality, the Cumulate Distributive Function (CDF) of a -1 offset distribution applied to the negative saturation limit will give the proportion of saturated values for a -1 offset signal. The equality can be written as:

$$\frac{1}{2}\left[1 + erf\left(\frac{-\beta}{\sqrt{2}\sigma}\right)\right] = \frac{1}{2^{n_{LLR}} - 1} \tag{3.2}$$

From equation (3.2), β can be deduced:

$$\beta = \sigma \times \sqrt{2}\left(erf^{-1}\left(\frac{2^{n_{LLR}} - 1}{2^{n_{LLR}} + 1}\right)\right) \tag{3.3}$$

Thus the β value is a function of n_{LLR} and is proportional to σ. By applying equations (3.3) and (3.1), the optimum saturation threshold and the scaling factor can be computed. The problem of this solution is that an

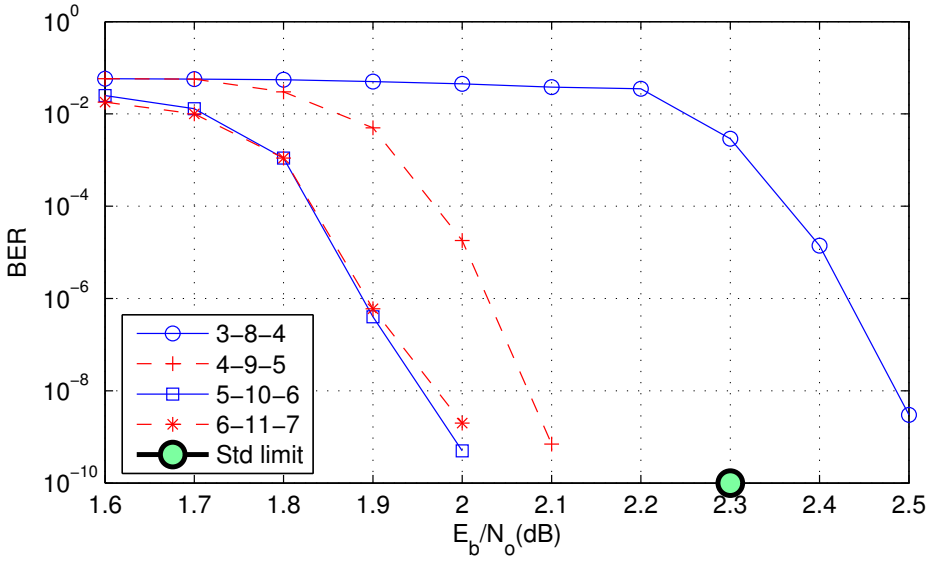


Figure 3.2: BER simulation for a rate 2/3

adaptative quantization of y is needed that requires a channel estimation of σ .

In fact, to prevent the σ computation, an optimal scaling factor is calculated for a given SNR. If the performance requirement is reached for a given SNR, then for higher SNR, the quantization would be sub-optimal. However, even with sub-optimal quantization, with higher SNR, the performance are still improving. For each code rate, a constant scaling factor ω and saturation value $1 + \beta$ can be pre-computed saving the need for the σ value.

Effects of LLR_{in} saturation on BER performance

Fig. 3.2 shows the simulation results for a normalized Min-Sum fixed point layered decoder, with a maximum of 30 iterations, long frame, code rates 2/3 in Additive White Gaussian Noise channel. The normalization factor is 0.75. Let us consider the following notation: a 3-8-4 configuration refers to a channel LLR quantized on 3 bits, an SO value word size of 8 bits and a $M_{c\to v}$ word size of 4 bits. We also depicted the standard limit at 1 dB from the Shannon limit in E_b/N_0 for code rate 2/3.

The quantification values of the $M_{c\to v}$ and SO are not optimized and chosen large enough to not affect the results of the channel quantification. Fig. 3.2 shows that a quantification on 4 or 5 bit of the LLR_{in} is enough to fulfill the standard requirements.

3.1.2 SO saturation

Once the LLR_{in} quantized, they are stored in an SO memory wich will evoluate with the iteration process. This SO memory need to be saturated to limit their size

The problem of SO saturation

We first consider the saturation case where $SO_{max} < SO_v^{new}$ during the SO update (1.18). The saturation process will bound SO_v^{new} to the SO_{max} value. This will introduce an error ϵ_v in the SO_v^{new} value ($\epsilon = SO_v^{new} - SO_{max}$). During the next iteration, the new $M'_{v\to c}$ value will be $M'_{v\to c} = SO_v - M_{c\to v} = M_{v\to c} - \epsilon_v$.

We now consider the worst case: during an iteration, SO_v is saturated at $+SO_{max}$, each CN confirms a positive $M_{c\to v}$ value, and d_v=13 (i.e. SO_v is saturated 13 times). At the beginning of the next iteration, $SO_v = SO_{max}$. From (1.15) and (1.18), we can deduce that $SO_{new} = SO_{old} + \Delta M_{c\to v}$ where $\Delta M_{c\to v} = M_{c\to v}^{new} - M_{c\to v}^{old}$. If $\Delta M_{c\to v} < 0$, the SO value decreases. The SO value can even decrease 13 times and change its sign. To summarize, when SO is saturated, the SO value cannot increase, but it can decrease. The saturation introduces a non-linearity that can produce pseudo-codewords and an error floor. A solution has to be found to overcome this problem.

A solution for SO saturation

The solution that we propose was first introduced in [16] and relies partially on the A Priory Probability (APP) based decoding algorithm [24]. The APP-variable decoding algorithm simplifies equation (1.15) to:

$$M_{v\to c} = SO_v \tag{3.4}$$

which greatly reduces the architecture complexity but introduces significant performance loss. The idea is to use equation (3.4) only when there is saturation. This leads to the APP-SO saturation algorithm, which is described as follows:

Algorithm 3 APP-SO saturation algorithm

if $SO_v = Max_{SO}$ **then**
  $M_{v\to c} = SO_v$
else
  $M_{v\to c} = SO_v - M_{c\to v}$
end if

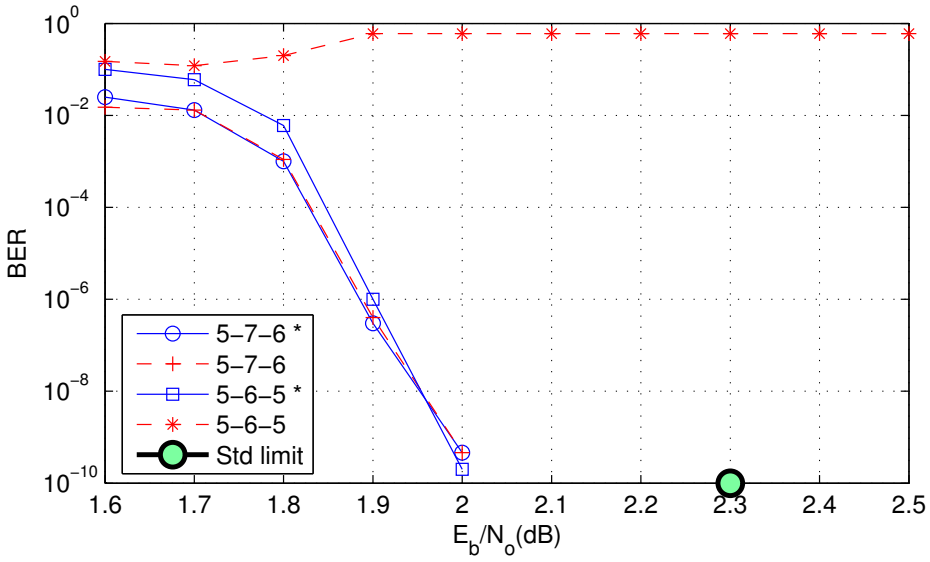


Figure 3.3: BER simulation for a rate 2/3 showing effects of SO saturation

With this algorithm, if a *SO* is saturated, then the *SO* will remind at the maximal value until the end of the decoding process.

Effects of SO saturation on BER performance

Fig. 3.3 shows the simulation results in the same conditions as in Fig. 3.2. An asterisk symbol in the legend means that the APP-SO algorithm is used. The results shows that the APP-SO algorithm, compared to a standard solution shows perfomance improvement only if the extrinsic values are satured (curves 5-6-5* and 5-6-5).

3.1.3 Saturation of the extrinsic messages

Figure 3.4 shows the SO based node processor. One can see that the newly updated extrinsic $M_{c\to v}^{new}$ is used to compute SO_v^{new} from equation (1.18) and $M_{c\to v}^{new}$ is also stored in the extrinsic memory for the calculation of $M_{v\to c}$ (equation 1.15) at the next iteration. Any saturation on the value $M_{c\to v}^{new}$ responsible for the *SO* update would not produce area savings and would degrade performance. This is the reason why we do not saturate this value. On the other hand, saturation of the $M_{c\to v}^{new}$ message that are stored in a memory would lead to significant area saving. Furthermore, the saturation of the $M_{c\to v}^{new}$ stored in the extrinsic memory is much less critical because it

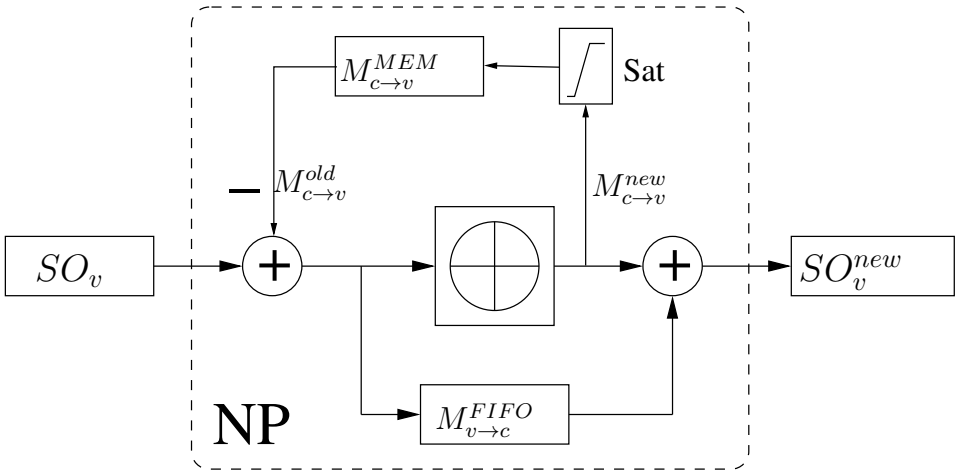


Figure 3.4: NP with saturation of the extrinsic messages

will be used only once during an iteration to compute an $M_{v\to c}$ and this $M_{v\to c}$ will affect SO^{new} only if it is a minimum value (see equation 1.8).

Effects of extrinsic message saturation on BER performance

Fig. 3.5 shows the simulation results in the same conditions as in Fig. 3.2. Simulations show that the $M_{c\to v}$ with the same quantification as the LLR_{in} ($n_{LLR} = n_{ext}$) gives result equivalent to higher quantification. A reduction of the quantification lower than the LLR_{in} quantification lead to dramatic performance loss.

3.1.4 Combining the saturation processes

Simulations show that the combinaison of the SO saturation process and the $M_{c\to v}$ saturation process lead to better performance that when they are implemented separately. The combinaison of our saturation process allows the use of fewer bits than the usual 6-8-6 configuration.

3.1.5 Saturation optimization conclusion

The analysis of the saturation process shows that a better trade-off between word size and performance can be obtained with an efficient saturation of the LLR_{in},the SO and the extrinsic values. Simulations show the robustness of our saturations allowing for the use of fewer bits than the usual 6-8-6 configuration. Simulations with channel values quantified on 5 bits, SO

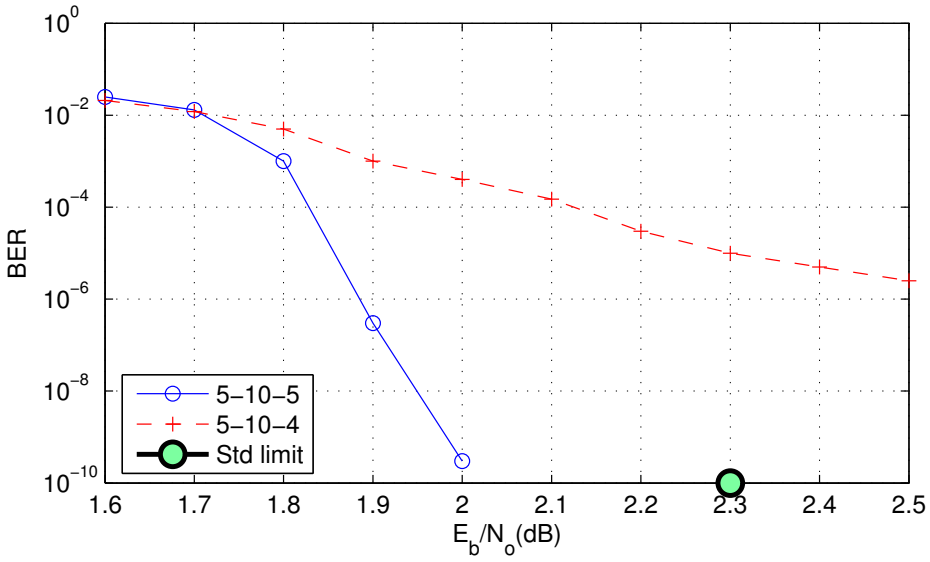


Figure 3.5: BER simulation for a rate 2/3 showing effects of extrinsic messages

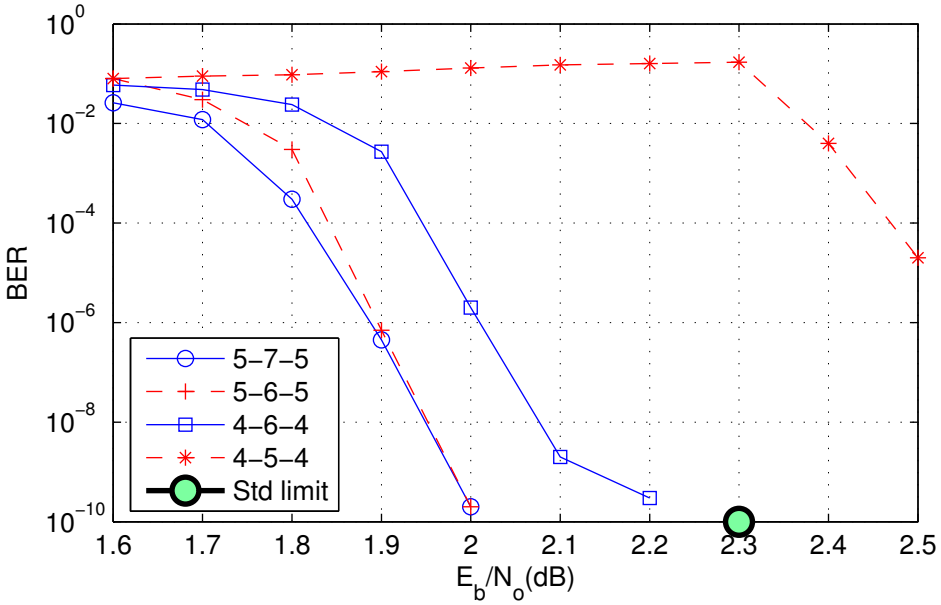


Figure 3.6: BER simulation for a rate 2/3 when combining SO and extrinsic saturation

Rate	M	W_{Sign}	W_{Ind}	$W_{Mc\to v}$	Memory
1/4	48600	4	2	14	680400
1/3	43200	5	3	16	691200
2/5	38880	6	3	17	660960
1/2	32400	7	3	18	583200
3/5	25920	9	3	21	544320
2/3	21600	10	4	22	475200
3/4	16200	14	4	26	421200
4/5	12960	18	5	31	401760
5/6	10800	22	5	35	378000
8/9	7200	27	5	40	288000
9/10	6480	30	5	43	278640

Table 3.1: Memory size of extrinsic

on 6 bits and $M_{c\to v}^{old}$ on 5 bits gives the same performance as other known implementation with fewer bits.

3.2 Optimizing the size of the extrinsic memory

This section focuses on the design of an optimal implementation for eleven different code rates. Because the extrinsic memory size requirements strongly depend on the coding rate, a memory dealing with many coding rate is an issue.

3.2.1 Extrinsic memory size requirements

The memory requirements of each CN is determined by the $M_{c\to v}^{old}$ messages needed for the CN computation. In the case of the normalized Min-Sum algorithm, the $M_{c\to v}^{old}$ values are compressed with min, $submin$, ind_{min} and $signM_{c\to v}$. In terms of memory, one address must be allocated for every CN which means that the RAM address range (R_{RAM}) is given by the number of CNs (M). The RAM word size (W_{RAM}) is given by the size of the compressed $M_{c\to v}^{old}$ values. If we denote by W_h the word size of h, then $W_{Mc\to v} = W_{|min|} + W_{|submin|} + W_{ind} + W_{sign}$. Table 3.1 presents the required memory capacity ($M \times W_{Mc\to v}$) for each rate. To calculate $W_{M_{c\to v}}$, we fix the value of $W_{|min|}$ and $W_{|submin|}$ to 4 (witout sign). To deal with the eleven code rates of the

standard, a simple implementation would define R_{RAM} with the maximum M value, and W_{RAM} with the maximum $W_{M_{c\to v}}$ in Table 3.1. Here, the total memory capacity would give: $48600 \times 43 = 2089800$ bits. For rate 1/4, 67% of word bits are wasted but addresses are fully used. On the other hand, for rate 9/10, words bits are fully used but 86 % of the addresses are wasted. Theoretically, a memory size of 691200 bits (maximum memory size in Table 3.1) would be enough to cover all the rates. An implementation solution has to be found for a better utilization of the memory.

3.2.2 Optimization principle

The idea is to add flexibility to both the address range and the word size. For this, we benefit from the fact that the RAM that stores the compressed $M_{c\to v}^{old}$ value is needed only once per layer. As the delay to compute the next layer is d_c cycles, we can use up to d_c cycles to fetch the data in the memory. A word can be split into two if we take two cycles to fetch the data, and split in three if we take three cycles. If we consider a single port RAM to implement the memory, up to $\lfloor d_c/2 \rfloor$ cycles can be used to read data, and $\lfloor d_c/2 \rfloor$ cycles to write new data.

Let us consider the example of a memory bank of size $48600(R_{RAM}) \times 22(W_{RAM})$. In a first configuration, where one cycle is used, we have a memory size of 48600×22 which fits to rates 1/4, 1/3, 1/2, 3/5, and 2/3. In a second configuration, where two cycles are used, and two words of size 22 are fetched at consecutive addresses, we have the equivalent of a memory of size 24300×44 which fits to rates 3/4, 4/5, 5/6, 8/9 and 9/10. The total memory size for the two-cycle option is equal to $48600 \times 22 = 106920$ bits. This constitutes a memory savings of 50% compared to the straightforward implementation.

3.2.3 Results of optimization

The previously described process can be generalized for different word sizes. Table 3.2 gives an example with $W_{RAM} = 9$. For each rate, the number of cycles is given by $n_{cycles} = \lceil W_{Mc\to v}/W_{RAM} \rceil$, and R_{RAM} is deduced from $R_{RAM} = n_{cycles} \times M$. The global RAM range ($R_{RAM}^{global}$) is given by the maximum R_{RAM} in Table 3.2 and the total memory capacity is $R_{RAM}^{global} \times W_{RAM} = 97200 \times 9 = 874800$ bits.

Fig. 3.7 shows the total memory capacity as a function of the word length W_{RAM}. There are local minima for word sizes 1, 9, 14, 18 and 21 bits. As the number of clock cycle to fetch $M_{c\to v}^{old}$ is bounded by $\lfloor d_c/2 \rfloor$, the possible solutions are limited to W_{RAM} greater than 7. A word size of 9 bits gives the

Rate	M	$W_{Mc\to v}$	n_{cycles}	R_{RAM}
1/4	48600	14	2	97200
1/3	43200	16	2	86400
2/5	38880	17	2	77760
1/2	32400	18	2	64800
3/5	25920	21	3	77760
2/3	21600	22	3	64800
3/4	16200	26	3	48600
4/5	12960	31	4	51840
5/6	10800	35	4	43220
8/9	7200	40	5	36000
9/10	6480	43	5	32400

Table 3.2: Memory capacity of the extrinsic message with $W_{RAM} = 9$

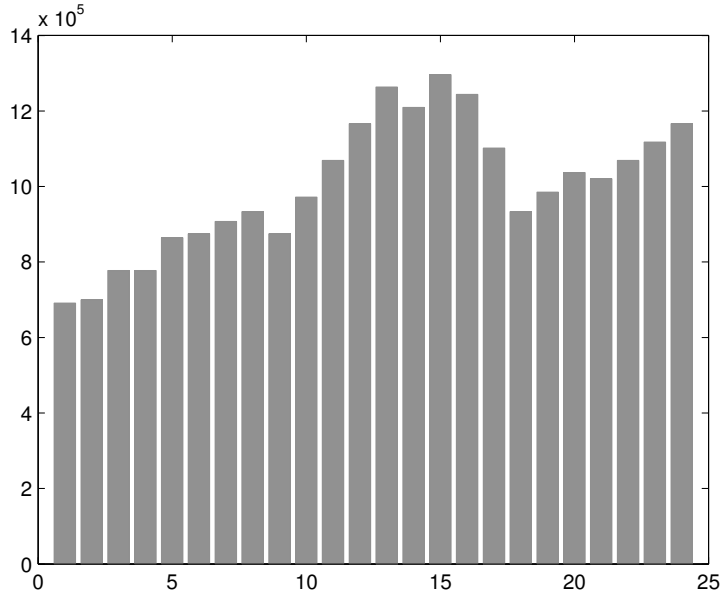


Figure 3.7: Memory capacity as a function of W_{RAM}

best memory optimization of 874800 bits. This is only 26 % more than the theoretical minimum.

We used the same memory optimization for an LDPC decoder that implement a BP algorithm using $M_{c\to v}$ values that are not compressed (all the $M_{c\to v}$ values are stored). We found a theoretical minimum of 1188000 bits, and a total memory size of 1296000 bits with words of size 10.

3.2.4 Case of the sum-product algorithm

When using a sum-product algorithm [49] [28] [23] instead of a Min-sum algorithm, the check node update equation (1.17) is computed for each $M_{c\to v}$ value and each $M_{c\to v}$ value are stored. The same process as the previously described can by used but a simpler and more efficient implementation can be used. We can consider a dual port ram which is read every cycle to fetch a $M_{c\to v}^{old}$ value and write simultaneously a $M_{c\to v}^{new}$ value to be stored. With the constraint of 11 code rates, 5 bits quantification and dual port ram, the memory requirement is given by the code rate that require the maximum number of $M_{c\to v}$ values. The code rate 5/6 requires storing 237600 values of 5 bit which give 1.2 MBits memory requirement. Although this size is 26% higher than the solution that we proposed for the Min-Sum algorithm, the over cost can worst it, considering the performance increase especially at low code rate. Implementation of a FIFO memory with single port modules for allowing simultaneous read and write operations is presented in [1]. This solution requires one memory banks for even addresses and another memory banks for odd addresses and lead to area saving compared to the use of dual port ram.

3.2.5 $M_{c\to v}$ memory optimization conclusion

The $M_{c\to v}$ memory optimization shows that the implementation of LDPC decoder for multiple code rate and single port RAM to store the $M_{c\to v}$ is an issue.

A careful implementation of the Min-sum algorithm gives result only 26 % more than the theoretical minimum compared to a straight forward implementation with 200% over cost.

The sum-product algorithm lead to a 26% over cost compared to the Min-Sum algorithm, but considering performance increase with low code rate, the implementation of the sum-product can be considered.

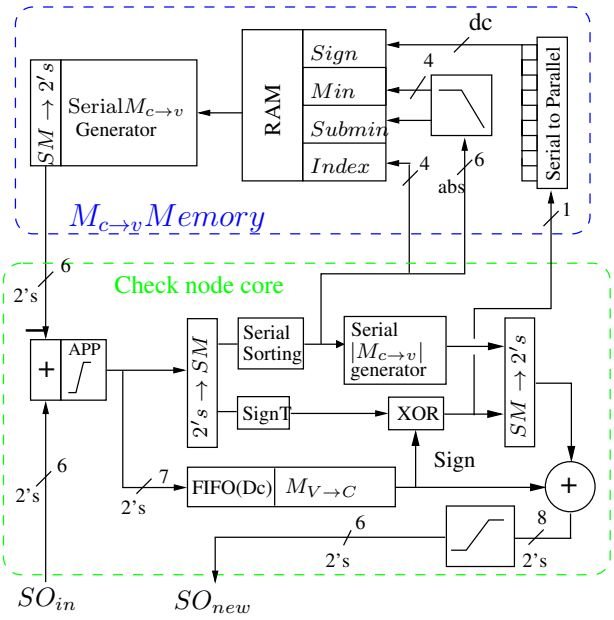


Figure 3.8: Finite precision of the NP architecture

3.3 Finite precision architecture of the layered decoder

Fig. 3.8 presents the finite precision architecture of the NP (Fig. 1.8). Word size, type (signed or absolute) and direction are detailed for every signal connection. The architecture implements the normalized Min-Sum algorithm.

In the Check node core, The $M_{v\rightarrow c}$ values arrive serially in two's complement representation from the adder modified to implement algorithm 3. They are transformed to sign and magnitude representation so that the sign and magnitude of the messages can be computed separately. The serial sorting of the incoming magnitude values is implemented, in order to give Min, $SubMin$ and $Index$ values until all the incoming messages are read. In the serial $M_{c\rightarrow v}$ block, the Min, SubMin and index values previously sorted are used to serially compute the new outgoing messages. An XOR function is computed recursively on the incoming sign bits in the $Signt$, until all the incoming messages are read. Next, an XOR function is computed in XOR block, between the output of $Signt$ block and the sign of the outgoing message $M_{v\rightarrow c}$ from the FIFO memory of size d_c. The sign and magnitude from XOR block and serial $M_{c\rightarrow v}$ block, respectively, are transformed into a two's complement representation ready for subsequent computations.

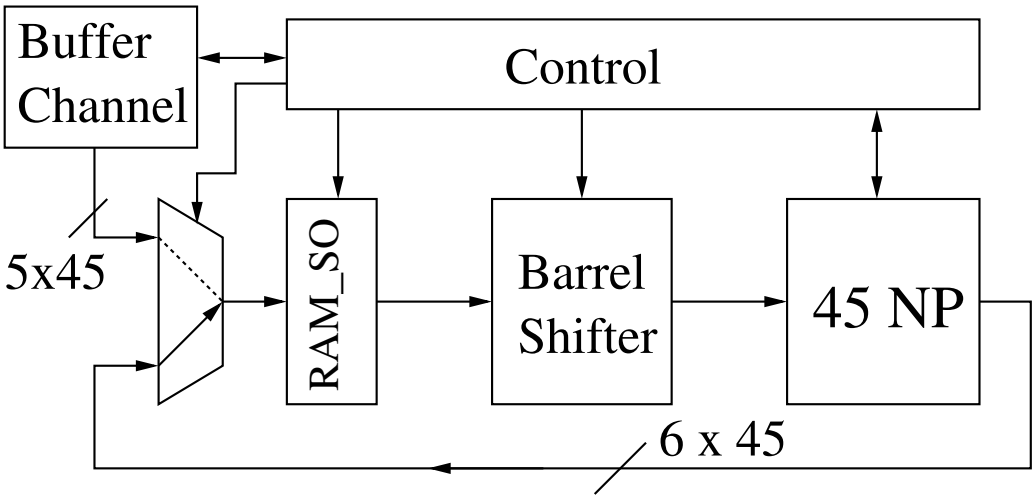


Figure 3.9: Layered decoder architecture

In the $M_{c\to v}$ memory block, the Min, $Submin$, $index$ and $sign$ of each $M_{c\to v}$ linked to one check node are stored in a RAM. From the Min, $Submin$, $index$ and $sign$ values, the serial $M_{c\to v}$ generator computes the two's complement representation of the $M_{c\to v}$ value.Min and $Submin$ are quantified on 4 bits (absolute values) which gives $M_{c\to v}$ values quantified on 5 bits (with the sign bit). Note that as $signt$ gives the result of the parity check equation of line and the syndrome can easily be computed by using the result of $signt$. The computation of the syndrome allows deciding an early termination of the decoding process leading to a significant reduction of the number of iterations.

Fig. 3.9 is an overview of the proposed layered decoder architecture (see [46] and [45] for a more detailed description). In this figure, the NP block is made of 45 NP (Fig. 3.8) working in parallel. The Barrel Shifter shifts seven words of size 45. The RAM_{SO} block stores the SO values. Thanks to a systematic syndrome calculation, it is possible to use a built-in stopping rule while decoding in a variable-iteration decoder. The addition of a buffer on the decoder input allows the exploitation of the variations in the decoding time in the different frames. A preemptive buffer control as in [52] is used to reduce the buffer size. Note that the quantification reduction also lead to area reductions of the node processor and the barrel shifter. The latency in the Check node core is also reduced due to complexity reduction of the addition and comparison computation.

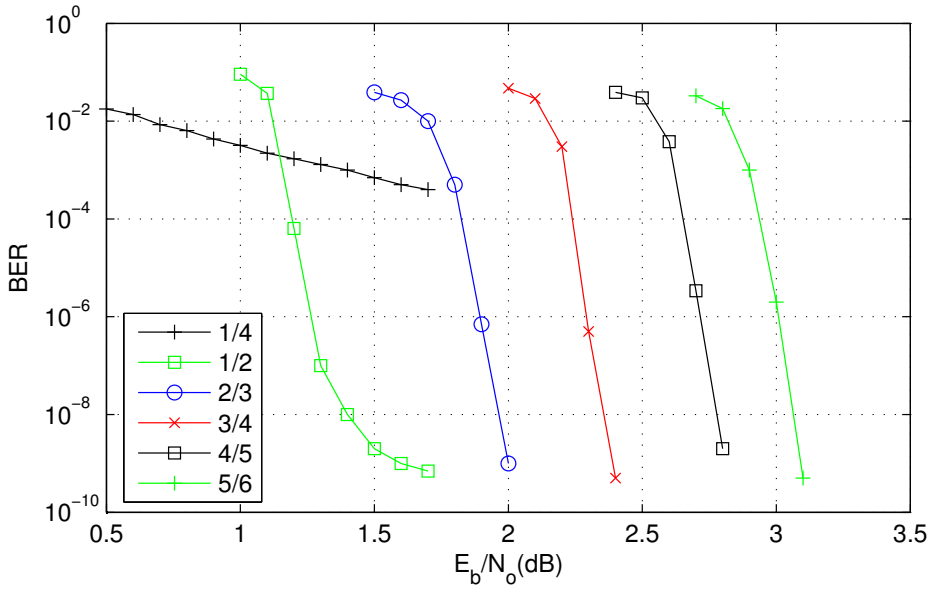


Figure 3.10: BER for long frames

3.4 Results of memory optimization

3.4.1 Monte-Carlo Simulations results

Fig. 3.10 shows simulation results for code rates 1/4, 1/2, 2/3, 3/4, 4/5, 5/6 and 5-6-5 configuration. The code rates 1/2, 2/3, 3/4, 4/5 and 5/6 show results that fulfil the standard requirements. The code rate 1/4 shows poor performances. The code rates 1/4, 1/3 and 2/5 have a check node degree smaller than 7 (4, 5 and 6 respectively) which lead to an error floor with the normalized min-sum algorithm. One can note that code rate 1/2 with a check node degree of 7 start producing an error floor that can be corrected with the BCH outer decoder. This error floor problem can be solved by implementing an A-min* or λ-min algorithm instead of the normalized Min-Sum in the CNP, with no change in the rest of the architecture. One can also implement a Sum-product algorithm as in [33] combined with the proposed saturation processes.

3.4.2 Synthesis results on FPGA

The architecture presented in Fig. 3.9 was synthesized on a Virtex-V Pro FPGA (XQ5VLX110) from Xilinx, for validation purposes. The system decodes long frames of code rate 2/3. Table 3.3 gives the hardware resources

XQ5VLX85	LUT	LUT RAM	BRAM
Node Processor	143	2	0
sorting	37	0	0
gen $m_{c\to v}$	34	0	0
fifo $m_{v\to c}$	12	2	0
$m_{c\to v}$ memory	46	0	3
Total 1 node	189	2	3
Total 45 nodes	8586	90	135
Control	667	3	0
block SO RAM	360	0	22
Channel RAM	48	0	25
Barrel shifter	945	0	0
Total	11005	93	182
Percentage [%]	5	1	50

Table 3.3: Synthesis Results for DVB-S2 LDPC decoder on Virtex-V

required. The clock frequency is 200 MHz, the average number of iterations is 20 and the throughput is 90 Mbit/s, which allows for the decoding of at least two simultaneous High-Definition Television (HDTV) streams.

3.4.3 Memory capacity comparison

Table 3.4 shows the number of bits for the main memory units in the latest published DVB-S2 decoder IPs [48, 66, 65]. Note that no information is provided on the ROM memories that store the matrices for every rate.

Paper	[48]	[66]	[65]	This
Parallelism	180	360	180	45
Air Throughput[Mb/s]	180	135	135	90
Extrinsic quantification [bits]	6	6	6	5
SO_{ram} quantification [bits]	10	8	8	6
Channel quantification [bits]	6	6	6	5
Buffer capacity [Mbits]	0	0	0	0.65
Total capacity[Mbits]	2.8	2.83	3.18	2.65

Table 3.4: Memory capacity comparison

In our architecture, the ROM capacity is 0.1 Mbits. A buffer of size two frames is used to store the channel LLR values. The buffer combined with a variable iterative decoder 4.4.3 allows increasing the throughput by two. Our architecture provides memory saving of 28% compared to [48], for the 5-6-5 configuration (for the 4-6-4 configuration the memory savings is 40%).

3.5 A single port RAM architecture

During the implementation process, the choice between single port and dual port RAM comes into the picture depending on the technologies used. Area, cost, consumption and the complexity parameters have to be also taken into account.

3.5.1 Single port ram, dual port ram, pseudo dual port ram and dual port RAM

The difference between Single Port RAM (SPRAM) and Dual Port RAM (DPRAM) is that SPRAM can be accessed at one address at one time, thus you can read or write only one memory cell during each clock cycle. Dual port RAM has ability to simultaneously read and write different memory cells at different addresses. SPRAM uses a 6 transistors basic RAM cell, while the dual port RAM cell uses 8 transistors cells for memory. So the area of SPRAM is much smaller than the area of DPRAM cell and is also more power efficient. To gain the advantages of both SPRAM (less area) and DPRAM (high speed), Pseudo DPRAM is introduced. Pseudo DPRAM can read and write the data in the same clock, using rising and falling edges for the operations respectively, and using SPRAM memory cell for the storage of data.

3.5.2 Memories in ASICS and FPGA

In FPGAs, most RAM structures are naturally dual-ported, e.g. the Virtex Block RAMs. On the other hand, in the ASIC library you can only instantiate some single port RAM and RAM which can be written in one port and read from the other port. This is a powerful FPGA argument if a true dual-port memory is needed. Because the use of DPRAM is not an issue on FPGA, an implementation on an FPGA of an LDPC decoder for prototyping and testing purpose is a convenient solution. However, the finale purpose of the layered decoder is to be implemented on ASICs for mass production at reduced cost. The studies of a SPRAM solution benefits from reduced area

and the fact that the layered decoder will have to be implemented later for ASIC solution. We discuss hereafter on existing solutions to change from DPRAM to SPRAM.

3.5.3 Implementation of dual port RAM with single Port

For some applications 2 SPRAMs can be used in an alternate buffer configuration. For dual read ports and a single write port, two RAMs are used and ones always write to both of them together, but read from them separately, with each treated as a separate read port. It is also possible to time-multiplex the two ports, which will take a double-speed clock and extra logic for the multiplexing.

3.5.4 FIFO memory with single port memory modules

In [1] is presented a FIFO memory with single port memory modules for allowing simultaneous read and writes operations. The method includes the following steps: (a) providing a first SPRAM for an even address of a read or write operation; (b) providing a second SPRAM for an odd address of a read or write operation; (c) alternating even address and odd address; and (d) when both a read request and a write request reach either the first single port memory module or the second single port memory module at a clock cycle, fulfilling the read request at the current clock cycle and fulfilling the write request at the next clock cycle.

3.5.5 Single port memories banks for the SO memories

By looking at an application, it is possible to see whether a full dual-port RAM is really needed, or whether one is dealing with a special case where it is possible to segregate it into independent parts. The memory can be build from multiple smaller RAMs and decode logic added to allow doing a simultaneous read and writing access, as long as they address separate RAMs. If the likely access pattern is known, the chance of collisions can be reduced by choosing which address bits select a RAM and which ones select a word in the RAM. In our case, the access schedule to memory is well known, and memory conflicts can be predicted and avoided by making sure that distinct memory bank are addresses for read and write operations at any time. Furthermore the order of the memory access can be modified if needed (layer scheduling and VG scheduling). The number of possible combination is huge (number of memory bank, addressing scheme in memory banks, layer

scheduling and VG scheduling). Finding the best solution will requires the help of graph theory but even with the help of graph theory, some parameters and implementation options have to be defined to reduce the space solution.

In [36],to avoid read/write conflicts, the SO memories are partitioned in 4 RAMs. A buffer is added to hold the data if writing is not possible due to conflict. The two least significant bits of address determines the assignment to a partition. This allows a simple control flow, which just has to compare the reading and writing addresses of the current clock cycle.

With a reduced parallelism compare to [36], a reduced number of RAM conflict can be expected. Our goal is to optimize the best addressing schedule so that no RAM conflicts arise and no buffer overhead is required.

3.6 Layer scheduling for single port RAM

The order or schedule of the SO arriving in the CNP does not change the result of the CNP (see Section 1.2.1) and an efficient scheduling of the SO can solve the read/write conflicts. For ease of understanding we use an example, we then generalize to general case giving problem and solution.

3.6.1 An example with two memory banks

In this example, a structured matrix with $d_c = 8$ is considered and the number of VG with even address is equal to the VG with odd addresses (4 are even and 4 are odd). Let us focus on one layer with the following VG address 1, 2, 4, 5, 8, 10, 13, 15. Usually the schedule follow the VG address in the increasing order. The schedule is modified such as an even number is followed by an odd number i.e. 1, 2, 5, 4, 13, 8, 15. The same process is used for the following layers. On this example the layered decoder has a delay of 9, which means that 9 cycles are needed between reading a value and writing the value. When the result of an SO at an even address is written back, an SO at an odd address is read. If all the even address are stored in memory bank (MB1) and all the odd address are stored in memory bank two (MB2) then it is possible to use single port RAM without risk of conflict and only two memory banks. In case of delay that is even, then a buffer of depth one is required in order that the writing operation fall in the previous described solution.

As the Less Significant Bit (LSB) address give the parity (odd or even), the implementation is straight forward by storing the SO in MB1 or MB2 depending on the SO address LSB. The benefit is the use of SPRAM instead of DPRAM. The drawback of this solution is that a structured matrix with

a specific constraint is required. This could be an interesting constraint for the design of architecture aware codes.

3.6.2 Generalization for DVB-X2 matrices

There is not always the same number of odd and even address in a layer. A solution is to switch one or some address from even to odd, or odd to even in order to fall in the configuration of the example. This switching of address would lead to an implementation over cost and even by trying all the possibilities, there is sometime no solution.

To give much more possibility, 4 memory banks are used as in [36]. The implementation is fixed and simple with the two LSB addressing the 4 memory banks. To solve read/write conflicts, the fact that VGs in a layer can be permuted is used. If there is a read/write conflict between two VG, one of then is permuted and provided that the permutation does not lead to another conflict then the conflict is solved. As the number of possible permutation is huge, a brute force approach is not possible and the help of graph theory is needed.

3.6.3 Genetic algorithm to solve scheduling problem

The advantage of using the Genetic algorithm (GA) is that we already used this powerful tool to solve memory update conflicts by the scheduling of the layers 2.5. Some modifications are required but the tool is ready. This time, the VG in a layer are permuted and the schedule of the layer is fixed. This work is not finalized yet.

3.7 Conclusion

In this chapter, a memory optimization for a layered LDPC decoder has been presented. A first approach to memory savings was to analyze the saturation problem in the layered decoder. A second approach relied on the word split of the extrinsic memory. We developed a finite precision layered decoder architecture that implements the proposed saturation process. This architecture outperforms the state-of-the-art in terms of memory needs while satisfying the standard requirements in terms of performances. A methodology is also given in order to implement single port RAM instead of dual port RAM in the layer decoder leading to significant space saving. Even if the DVB-S2 standard is considered in this chapter, the proposed techniques

can be extended to the DVB-T2 and -C2 standards and, more generally, to any layered LDPC decoder.

Logic will get you from A to B. Imagination will take you everywhere.

Albert Einstein (1879-1955)

4

Multi Stream LDPC decoder

Summary:

In this chapter, the goal is to find a way to reduce the cost of decoding x streams by using one decoder compared to the solution of using x decoders. The state of the art gives solutions based on increasing the parallelism, sharing resources or inserting a buffer to a variable iteration decoder. These three options are considered and developed to satisfy our case study. Then an architecture supporting multiple streams is presented

4.1 Introduction

Multi stream decoder can be interpreted in two ways. First, it can be interpreted as a decoder able to decode multiple kinds of standards. To be more specific, for television broadcasting, many stream standards exist:

- Standard-Definition Television (SDTV) streams,
- Extended-Definition Television (EDTV) streams,
- High-Definition Television (HDTV) streams,
- Three Dimensions Television (3D TV) streams.

From the LDPC decoder point of view, the difference between these standards is only in terms of required throughput. The bandwidth is highly dependent on the standards listed here:

- the bandwidth consumption of SDTV (720 x 480 pixels) is about 5 Mbps,
- the bandwidth of HD TV (1920 x 1080 pixels) stream is about 13 Mbps,
- 3D Blu-ray uses about 50% more throughput than HDTV depending on the compression,
- in case of 3D TV based one HD image per eye; the bandwidth is the double of one HD TV stream.

Between single SDTV stream and a 3D TV stream, there is a factor 5 in throughput requirement. In this interpretation, a multi stream decoder has to fulfill the highest required throughput.

The term *Multi stream* decoder can also be interpreted as a decoder decoding simultaneously x streams. Double or twin tuners make the difference on the market when customers want to record a movie while watching another. A video surveillance application can require up to 32 streams at 30 frames per second simultaneously. The required throughput in these cases are simply multiplied by x. In both interpretations, multi stream LDPC decoder is a matter of throughput. Combining the two interpretations in one decoder, i.e. a dual stream 3D TV decoder, the throughput will vary from one STD stream to two 3D TV streams, which makes a factor 10 in throughput variation. The decoder will be designed for the worst case which means the highest throughput but energy saving should be managed when the decoder is under exploited. A straight forward implementation of a multi stream decoder would use x decoders in parallel.

In this chapter, we present the case of one decoder decoding x streams and optimize area saving compared to the straight forward solution. The one decoder solution is studied through three axis. The first one gives flexibility to the throughput of a given layered decoder by reordering the matrix in function parallelism. The second axe presents an architecture allowing sharing resources for a dual stream LDPC decoder. The third option presents a variable iterative decoder associated with a buffer allowing increasing nearly by two of the throughput. Finally, an architecture is presented.

4.2 The parallelism option

The matrices given in the DVB-S2, -T2 and -C2 standards are structured for a parallelism of 360. The split process of the matrices used to reduce the number of conflicts [45] [46] described in Chapter 2 can also be used to gain

P_s	36	40	45	60	72	90	120	180	360
100 MHz	42	47	53	70	84	106	141	212	423
200 MHz	84	94	106	141	169	212	282	423	845
400 MHz	169	188	212	282	338	423	564	846	1691

Table 4.1: Throughput (Mbit/s) as a function of the parallelism P_s and the Clock frequency with $it_{cst} = 17$ and $\epsilon = 6$

flexibility in the parallelism and thus in the throughput. The parallelism can be reduced by using the splitting process to $P = P_s$ provided that $360 = P_s \times S$ where P_s and S are natural numbers. Note that in [13] a method is proposed in which the parallelism factor does not have to be a divider of the original expansion size (P =360). This solution gives freedom of choosing a parallelism factor that fits exactly the required throughput. The throughput of a pipelined layered LDPC decoder is given by the following equation:

$$D_2 = \frac{K.F_{Clk}}{d_c.\frac{M}{P_s}.it_{cst} + d_c + \epsilon} \quad bit.s^{-1} \tag{4.1}$$

where it_{cst} is the average number of iterations to decode a codeword, M is the number of CN, P_s is the number of CNPs working in parallel, d_c is the average number of VNs linked to a CN, F_{clk} is the clock frequency, K is the number of information bits in a codeword and ϵ is the CN latency.

Table 4.1 gives an idea of the throughput in function of the parallelism and the clock frequency when using standard frame and coding rate set at 2/3, $\epsilon = 6$. A buffer is added to the decoder as presented in Section 4.4 which reduces $it_{cst} = it_{max} = 30$ to $it_{cst} = 17 = it_{avr}$.

4.2.1 Area saving compared with x decoders and conclusion

To give an idea of area saving, we compare the parallelism solution with a x decoders solution. In our case of study, an LDPC decoder for one HD stream gives in term of area 75 % of memory, 20 % of CNP and 5 % of control. To decode a dual stream for a given throughput, two decoders decode two frames simultaneously with a parallelism of p while one decoder decodes one frame at a time with parallelism of $2p$. For the one decoder solution, the parallelism is doubled but the control and the memory required is unchanged. Compared to a two decoders solution, the area saving is 40 %. Generalizing with x streams, the parallelism option save $x - 1$ memories. The area saving is 53%

for $x = 3$ and 60% for $x = 4$.

The parallelism option allows to fit with the various throughput requirements and provides significant memory and area savings compared to a x decoder solution.

4.3 Share resources in a dual stream decoder

A Block-interlaced LDPC decoder is proposed in [12] with an architecture based on the flooding schedule. The interlacing technique processes two frames simultaneously. In this section we apply the interlacing technique to a layered decoder and we discuss its advantages and drawbacks.

4.3.1 Sharing principle

As shown in Section 1.2.2 the check node update is split into two steps of d_c cycles. First step, the data are read, and second step, the updated data are written back. These two steps can easily be pipelined by overlapping the update of two layers. When one layer is updated in step one, another layer is updated in step two. The pipelined scheme in Figure 2.17 doubles the decoder throughput compared with the throughput in Figure 2.16.

Instead of overlapping two layers of the same frame, the decoding of two consecutive frames is interlaced. While a layer is reading frame one in step one, another layer is writing frame two in step two. For layered decoders, block interlacing keeps the CNP unchanged but requires doubling the LLR and $M_c \rightarrow v$ memories so that the CNPs can switch between the two memories as they switch between the two different frames.

4.3.2 Advantages, drawbacks and conclusion

This solution has four advantages compared to a pipeline solution. First advantage is that, the conflicts due to pipeline as describe in Section 2.5 are avoided because there is no SO in common between two consecutive layers. Second, there is no more need of a simultaneous read and write access to the LLR memories. Instead of using dual-port RAM for the LLR memory, single port RAM can be used without reading and writing conflict risk , leading to significant area saving. Third advantage is that the area is saving p CNP compared to a two (not pipelined) decoder solution. Fourth advantage: Table 2.4 shows that there is no solution for conflicts due to pipeline with high parallelism and block interleaving avoids this conflict.

There are two drawbacks of this solution. First drawback is that at constant throughput, compared with 2 pipelined decoders, an interlaced decoder does not lead to area saving. The interlaced solution needs as much memory as the two decoder option and to have the same throughput as one pipelined decoder, the parallelism of the interlaced decoder need to be doubled. Second drawback is that the streams need to be at the same rate. In case of two streams at different rates, which means the d_c are not equal, idle time has to be inserted to the smallest d_c so that the pipelining process can work. The rate problem can be solved by adding a buffer and some kind of control so that only frames of the same stream are decoded simultaneously but the buffer size need to be nearly doubled compare to a non interlaced solution.

These drawbacks make this solution not relevant for low parallelism, nevertheless in case of high parallelism, this solution becomes more relevant. In particular, with a parallelism of 180 or 360, there is no simple solution to solve conflicts due to pipeline (See Table 2.4) and the interlaced architecture solves the problem.

4.4 Use of a buffer

Low-density parity-check (LDPC) decoders can be designed in a way they stop after a variable number of iterations, dependent on the difficulty of decoding noisy received words. The number of iterations the decoder spends on a given frame determines both the probability of successful decoding, and the time expended. Thanks to the systematic syndrome calculation, it is possible to use a built-in stopping rule while decoding in a variable-iteration decoder. The addition of a small buffer on the decoder input allows the exploitation of the variations in the decoding time of the different frames. Whereas the speed of an LDPC decoder without a buffer is determined by the maximum number of iteration it_{max} frames, the speed of a variable-iteration decoder with sufficient buffering is determined by average number of iteration it_{avr}. The addition of an input buffer to an iterative decoder is presented in [52] and is summarized in Figure 4.1. The buffer, the variable iteration decoder and the preemptive buffer control are described in the following subsections.

4.4.1 FIFO buffer principle

FIFO (First In First Out) memories have acquired larger interest in VLSI design in telecom applications. Such macro cells provide an efficient way to interface two asynchronous system, buffering data between subsystems

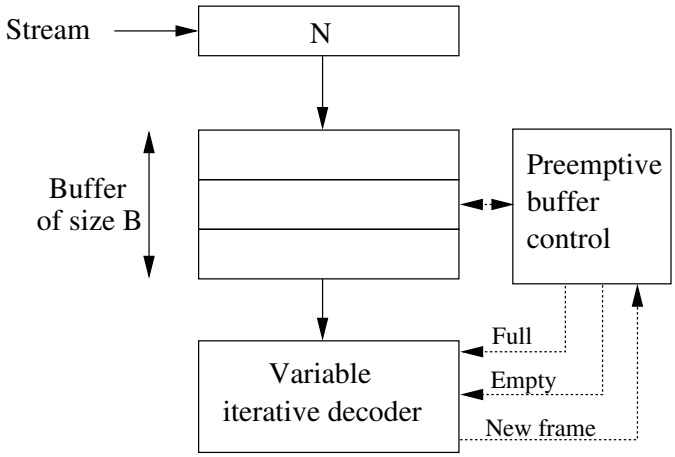


Figure 4.1: Variable iteration decoder with a buffer

operating at different data rates, thus increasing the flexibility in data transmission. In fact, data in FIFO memories are read in the same order in which they have been written, without the need of providing the address of the data.

Different approaches have been considered in literature to design FIFO memory cells. The architecture is based on SPRAM cell as described in Section 3.5.4. A generic architecture is defined whose size can be modified with slight modifications to the basic structure. In order to obtain such a result, two blocks are defined: the memory block which instantiate SPRAM and the control logic block. The control logic is in charge of detecting two particular states of the FIFO which are *full* and *empty* that will have effects on the decoding process. The *empty* message says to the decoder that after the current decoding, the decoder can switch to a stand-by mode to save energy consumption. The use of the *full* signal is discussed in the following subsection.

4.4.2 Preemptive buffer control

Before the buffer gets full, the *full* signal is sent to the decoder for an early termination of the decoding process and the *first in* data of the FIFO is sent to the decoder to let one space in the buffer to avoid a dramatic overflow. An overflow would lead to the lost of one frame while the effect of stopping the decoding process is limited. The effect of stopping the decoding is limited for three reasons: first reason is if the word under decoding is hard to decode (pseudo codeword or cycle trap), spending more iteration would not give solution. Second reason is that the degree of parity bits is only two and the

convergence of bit with low degree is slower than nodes with higher degree. For this reason, the errors are most probably in the parity bits and not in the information bits; this means that even if the stopping criterion is not valid, the information bits can be valid. Third reason is that in case that just a little more iteration would have been necessary to solve remaining bits, then the BCH can solve the last erroneous bits if the number of erroneous bit is lower than the BCH capability. Compared to a standard buffer, the preemptive buffer control significantly reduces the size of the buffer and reduce the average number of iteration to decode even with a buffer of size one.

4.4.3 Variable iterative decoder

A variable iterative decoder is able to detect that a codeword has been found. This solution allows starting the decoding of a new frame. The detection of a valid codeword, in theory is done by computing the syndrome and checking equation (1.1). In practice, the syndrome computation can be done during the sign calculation of the $M_{c\to v}$ message (See Section 1.2.2). The first step of the sign calculation gives:

$$signt(v') = \prod_{v' \in v_c} sign(M_{v'\to c}) \tag{4.2}$$

which is in fact the hard decision of the parity check of CN c. If all the hard decisions of all the check nodes are valid then a codeword has been found and the iterative process can be stopped. This stopping criterion is easy to implement on a layered decoder and requires a very low cost control. Because the decoding time of the variable iteration decoder is not constant, a buffer is required at the input of the decoder to smooth the decoding time variation.

4.4.4 FIFO Buffer size

A buffer of size one would be in charge of storing the LLR_{in} of every bit of the codeword. The size of a codeword of a DVB-X2 decoder is N=64800 and the quantification of the LLR_{in} in our architecture is 5. The storage requirement of a buffer of size x is $64800 \times 5 \times x$ which makes the buffer size the most critical point of the solution in terms of area. Therefore, the buffer size must be chosen carefully.

The calculation of the optimal buffer size and the average time to decode can be estimated using queuing theories and the probability distribution of the number of iterations (see Figure 1.6). Queuing theory is the mathematical study of waiting lines, or queues. The theory permits the derivation and calculation of several performance measurements including the average

waiting time in the queue, the expected number waiting and the probability of encountering the system in certain states such as empty and full.

The calculation of the buffering requirement by using queuing theories has been already presented successfully in [37]. In this subsection, a more empirical approach is described to determine the optimal buffer size. In fact, a simple patch has been added to the already existing simulation environment of the LDPC decoder and a Monte-Carlo simulation has been lunched for different buffer sizes. The goal of the simulation is to find the best compromise between buffer size, number of iteration and performance loss. We present first a simple patch simulating a single stream, and then we generalize with a patch simulating multiple streams.

The algorithm patch can be described by the three following points: a new frame is added to the buffer every it_{cst} iterations; the buffer is decremented every time a codeword has been found; and if the buffer is full, then the decoder stops the decoding of the current frame and takes a new frame in the buffer to relax the buffer.

Algorithm 4 buffer simulation

$it \leftarrow it + 1$
if $it = it_{cst}$ **then**
  $Buffer \leftarrow Buffer + 1$
  $it \leftarrow 0$
end if
if $(word\_decoded = \textbf{true}) AND (Buffer > 0)$ **then**
  $Buffer \leftarrow Buffer - 1$
end if
if $buffer > buffer\_size - 1$ **then**
  decode_next()
  word_decoded=**true**
  $it \leftarrow 0$
end if

The patch is sum-up in Algorithm 4 where it_{cst} is the average number of iteration a decoder spend to decode a codeword. Figure 1.6 shows the PDF of the number of iteration before a codeword is found. On the figure it_{max} and it_{avr} are also presented . The it_{max} value is the maximum number of iterations that a decoder spend to decode a codeword and the it_{avr} value is the average number of iterations before a codeword is found. On a decoder with a constant number of iteration to decode, it_{cst} is equal to it_{max}. The advantage of a buffer associated with a variable iterative decoder is that it_{cst} get closer to it_{avr} but even with a huge buffer size, it_{cst} can not be lower than

the average number of iteration it_{avr} to decode a codeword. Thus, it_{cst} will be between it_{avr} and it_{max}. As shown in equation (4.1), the throughput is directly proportional to it_{cst} and has to be as low as possible. The goal of the simulations is to optimize it_{cst} and *buffer_size* to a minimum with the same performance as a decoder with a constant number of iteration to decode fixed at $it_{cst} = it_{max} = 30$.

Simulations on a decoder with a parallelism of 40 show us that a buffer of size two gives already the required performances. The performances can be slightly improved by using a buffer of size 3 and a buffer of size greater than 3 does not give significant improvement. With a buffer size two, $it_{cst} = 17$ which is close to $it_{avr} = 16$ and nearly the half of it_{max} thus nearly double the throughput compared to a constant iterative decoder. Note that once it_{cst} is found, one should check if Equation (4.1) gives a throughput higher than the required throughput, and if not, increases the parallelism in proportion. Note also that the simulation emulates on an ideal case making some assumptions which are simplifying the architecture constraints. Among the simplifications, we consider that the FIFO buffer is read and written without latency, the decoder can stop a decoding process without delay, and decoder load a new word without latency. These assumptions can be acceptable if the latencies are negligible compared to the time unit of the simulation which is expressed in terms of iterations.

A SDRAM with a transfer rate of 64bit per cycle would require at least 6075 cycles to transfer the 64800×6 bits of one frame which is roughly equivalent to one iteration time for a rate 2/3 with a parallelism of 40.

In case of multi stream, if the streams are serialized, then Algorithm 4 can be used without modifications. If the streams arrive in parallel, then the patch is modified as follow: n frames are added to the buffer every $n \times it_{cst}$ iterations. The Algorithm 4 is upgraded to Algorithm 5 where RAM_{write} and RAM_{read} represent the time to write (read) a new frame in number of iteration.

For dual stream, with a decoder with doubled parallelism, simulations using Algorithm 5 show us that a buffer of size 5 fulfill the required performances.

4.4.5 Advantages and drawbacks

The only drawback of this solution is the buffer size. The main advantage of this solution is the improved throughput by nearly two. It is also possible to get advantage of additional iterations to decode a codeword if the buffer is empty by increasing it_{max}. In the same way of thinking, it_{max} can evolve in function of the buffer state. If the buffer is about to get full then decrease

Algorithm 5 buffer simulation 2

$it \leftarrow it + 1$
if $it = n \times it_{cst}$ **then**
  $it \leftarrow 0$
end if
if $it = n \times it_{cst} - RAM_{write}$ **then**
  $Buffer \leftarrow Buffer + n$
end if
if $it = RAM_{read} AND (Buffer > 0)$ **then**
  $Buffer \leftarrow Buffer - 1$
end if
if $buffer > buffer\_size - 1$ **then**
  decode_next()
  $it \leftarrow 0$
end if

it_{max} and if the buffer is about to be empty then increase it_{max}.

4.4.6 Implementation issue

The parallelism option and the addition of a buffer give the throughput and flexibility required by the decoding of multi streams. The parallelism option has been integrated in the design of the VHDL code as a parameter. To change parallelism, the parallelism parameter is changed and the VHDL is synthesized again to implement the modification.

The circular buffer can be implemented using a FIFO associated with an appropriate control. The control is in charge of sending the *full* signal and the *empty* signal. In case of multi stream, for example dual stream decoder, the FIFO can be modified to have two inputs and one output. Figure 4.2 shows an architecture example where two streams are filling the top of the buffer. Another buffer is added to the output of the decoder to synchronize with the BCH. The output buffer stores the hard decision of the decoder and the buffer control is in charge of reordering the stream one and two.

The buffer can be implemented in Synchronous Dynamic RAMs (SDRAMs) to save cost. SDRAM devices are low-cost, high-density storage resources that are widely available. To reduce the read and write operation latency, the SDRAM can be read and write by using a burst-mode. For a dual stream decoder, two memory banks are written simultaneously while one memory bank is read. Two memory banks are added as FIFO storage (See Section 4.4.4). This makes a total of 5 memory banks.

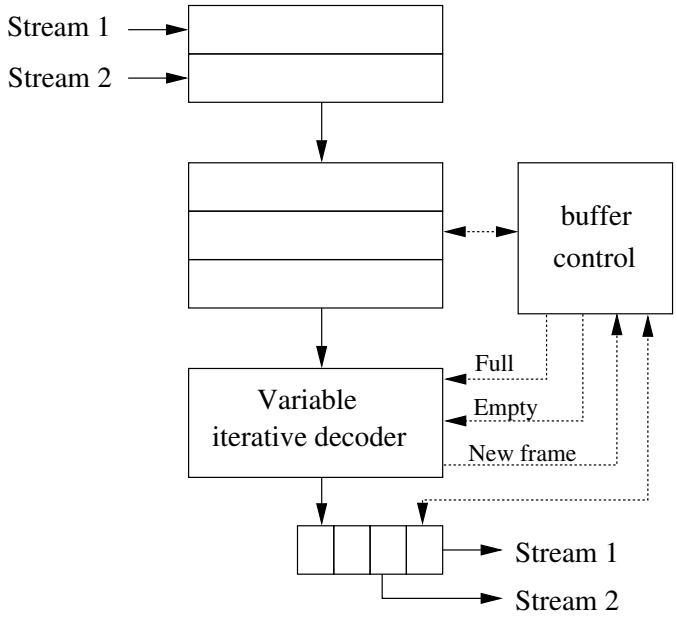


Figure 4.2: Variable iteration decoder with a buffer

The loading of a new word in the decoder can be time consuming and reduce the throughput. One solution is to use two SO memories in "ping pong". While one memory is used for the decoding process, the other one is loading a new word. When a codeword is found, the two memories are switched.

4.5 Conclusion

The fact of sharing resources to decode two streams simultaneously gives more drawbacks than advantages, especially at low parallelism. The parallelism option combined with the addition of a buffer gives the throughput and flexibility required by the decoding of multi streams. The parallelism of the decoder is chosen in function of the required throughput. To deal with the variable throughput requirement, a buffer added to a variable iterative decoder gives furthermore the advantage of reducing the average number of iterations the decoder spend to decode a codeword.

An investment in knowledge always pays the best interest.

Benjamin Franklin
(1706 - 1790)

5 Conclusion

In this thesis, we provide solutions for implementing an LDPC decoder for the second generation Satellite Digital Video Broadcast (DVB-S2) standard, the DVB-T2 (terrestrial) standard and the DVB-C2 (Cable) standard.

In **Chapter 1** we introduced the advanced techniques of channel decoding. After a brief overview of digital communication and channel decoder, notions and notations of Low Density Parity Check (LDPC) codes have been presented and more specifically the layered decoder and the Believe Propagation (BP) sub-algorithms are detailed. Then the LDPC included in the DVB-S2, -T2 and -C2 standards is presented with the state-of-the-art of existing implementations. Finally, we present a testing environment designed to evalutate the performance of error correcting codes. From the state-of-the-art of exiting implementations, the layered decoder is identified as the most efficient architecture. This architecture uses the Normalized Min-Sum algorithm for the check node update because of its good performance and low area cost of implementation. In the aim of most efficient implementation, the check node processors are pipelined to double the throughput.

The implementation of this defined architecture led in practice to memory update conflicts. Two types of memory update conflicts have been identified in **Chapter 2**: the conflicts inherent to the presence of overlapped sub-matrices and the conflicts due to the pipelined architecture. To improved the exiting solutions, we decided to find a pach free solution to keep the efficiency of the layered decoder.

We proposed two innovative solutions to solve the conflicts inherent to the overlapped-sub-matrices. The first solution that we presented in [45] is based on a "divide and conquer" strategy. The number of conflict is first reduced by a reordering mechanism of the matrix called split algorithm that creates a new equivalent structured matrix with reduced parallelism. With this first step, lower is the parallelism and lower is the number conflicts.

The remaining conflicts are solved with a "split-extended" matrix. The split extend proced operates on deficient layer by cutting the layer in two separate layers. Each obtened layer contain one of the overlapped submatrices. To keep the check equation valid, the two layers are extended with a commun group of dummy bits. This first solution is efficient especially for layered decoder that can deal with not constant check node degree and low parallelism.

The other solution is in a patent process and it is based on a repeat of the deficient layers and an "on time" write disable of the memories. This second solution is especially efficient for architecture using not compressed extrinsic memory and is efficient even at high parallelism.

The two proposed solution are not 100% patch free, but the added complexity is low and require just a slight increase of control. This added control is not in the critical path and does not degrade the throughput.

In Section 2.5 and in [46], we solve the conflicts due to pipelining. The split process is used again as a first step. This first step reduced conflict constraints allowing finding solutions by a scheduling of the layers. The layer scheduling required the need of graph theory to be solved in a reaonable computing time. We showed that the research of an efficient scheduling is equivalent to the well known "Traveling Salesman Problem" (TSP). Then a genetic algorithm heuritic has been used to solve the TSP.

All the proposed method are compatible between each other. It means that the methods can be combined in a single implementation to solve both conflicts due the overlapped matrices and conflicts due to pipelining.

After the memory conflict problems solved, we focused on finding different ways of optimizing the layered decoder implementation in **Chapter 3**. As 80% of the decoder is memory, we explored how to reduce the memory size. The memory is directly proportional to the word length and we reduce it by a specific saturation. We identified the saturation effects determinig which value are less sensitive to aturation and we conteract unwanted effect of saturation. cost in memory and .interaction between the SO values and the extrinsic messages. Then we presented a finite precision layered decoder architecture that implements the proposed saturation processes. With all the code rates, extrinsic memory address range and word size are subject to

significant variation. We discussed the best way to implement this variation with the minimum memory over cost. The saturation process and extrinsic memory optimization have been presented in [44] and implemented on an FPGA giving performances in the standard requirement and significant area reduction compared to the state-of-the-art. After that, still in the memory optimization field, we discussed efficient solutions to implement the LDPC decoder using only single port RAM instead of dual port RAM. Once the conflicts have been solved and the layered decoder has been optimized, we finally focused on the multi-stream problem.

In **chapter 4**, we explored the problematic of finding the best way to implement a decoder that decodes multiple streams simultaneously. In our proposed solution, the parallelism of the decoder allows to reach the required throughput while the addition of a buffer to one variable iteration decoder gives an answer to the flexible throughput requirement.

5.1 Produced work

Besides the publication and searching process [45, 46, 44], we also produced generic tools useful for the testing and designing of an LDPC decoder. First of all, we produced a general ANSI-C model of an LDPC decoder with the associated testing environment to test the possible algorithms and scheduling options. The following step was to design another ANI-C model close to the layered architecture for co-simulation C-VHDL. Then we have implemented on FPGA two versions of LDPC decoders. The first decoder implemented is a generic layered decoder. This decoder is able to decode any true layered codes such as codes defined for the WiMAX and WiFi standards. The implemented decoder is also able to decode DVB-S2, -T2 and -C2 codes but only for some parallelisms where no overlapped sub-matrices appear. This version allows also scheduling layers and thus avoiding memory update conflicts due to pipeline as described in Section 2.5. An upgraded version of the layered decoder allowing repeating layers and controlling the write enable of the SO and extrinsic memory have been designed. This second version is able to deal with the conflict due to overlapped sub-matrices and it is relevant for the DVB-X2 standards and also for the Chinese Multimedia Mobile Broadcasting (CMMB) standard.

To test the efficiency of the LDPC decoder even at low bit-error rate (below 10^{-10}), a testing environment has been designed. The testing environment has been designed in C and VHDL which allows C-VHDL co-simulation. The testing environment is an efficient tool to test, to design and to debug an LDPC decoder or in fact any channel decoder.

5.2 Perspectives

Future work will be dedicated to the hardware implementation optimization (area, frequency and performance) of the proposed decoder architecture and to the evaluation of its performance at low bit error rate. The implementation of a Bose and Ray-Chaudhuri (BCH) decoder is also considered to complete the transmission chain of the testing environment. Implementing and testing a multi stream decoder with the associated buffer and control should be done to confirm our estimation of the optimal buffer size (see Section 4.4.4).

The performance of low code rates (1/4, 1/3, and 2/5) should be slightly improved to fulfill the standard requirement. One solution consists in upgrading the Normalized Min-Sum Algorithm to a $\lambda - min$ or $A - min*$ algorithm. To achieve this, a patch with Look up table can be added to the check node processors without significant modification in the architecture. An upgraded version of the existing layered decoder is also considered. This upgraded version would be able to decode all rates and would implement the structure of the extrinsic message memory proposed in Section 3.2. This version would integrate the upgraded Min-sum algorithm that fulfills the DVB-X2 requirement for low rate codes.

Standards give lots of challenges to today's researchers and to all working in this field to find improvement options. An improving solution which would provide better reception to the customers and would enable to offer better services by keeping the costs in a reasonable range or even decrease it. The current economic crisis, the demanding market needs urge the companies and the researches to find answers and respond to these great challenges. This thesis has a potential answer to a small part of these challenges, but certainly an important one.

Simplicity is the ultimate sophistication.

Leonardo da Vinci
(1452-1519)

A

DVB-S2 matrices construction

The DVB-X2 standard provide a guidline to construct matrices. The resulting matrices are structured but they are not made of shifted identity matrices as the matrices defined by the WiFi or WiMAX standards [60] and they are not ready for layered decoding. Some permutations are required to make the identity matrices appear. In this appendix, a simple example is provided to visualise the initial structure of the matrices, the required permutations and the final structure.

A.1 Standard matrices construction

The matrices defined by the standard fall in the IRA codes types. The matrices are caracterized by two parts: the information part is linking the Information VNs (IVN) to the CNs and the parity part linking the Parity VNs(PVN) to the CNs. The standard provides a matrix construction protocol for the information part relaying on the use of a table and an equation. For ease of understanding, in this example a parallelism of 8 instead of a parallelism of 360 and codeword length of 40 instead of 16200 are given. In this example, coding rate is 2/5 and q = 3.

Table A.1 has only three values in it but will be used as a table from the standard (Table 1.3). The values in the first line give the connection between the first eight VNs (IVN_0 to IVN_7) and the CNs. A value x in the table is

2	7
1	

Table A.1: example table

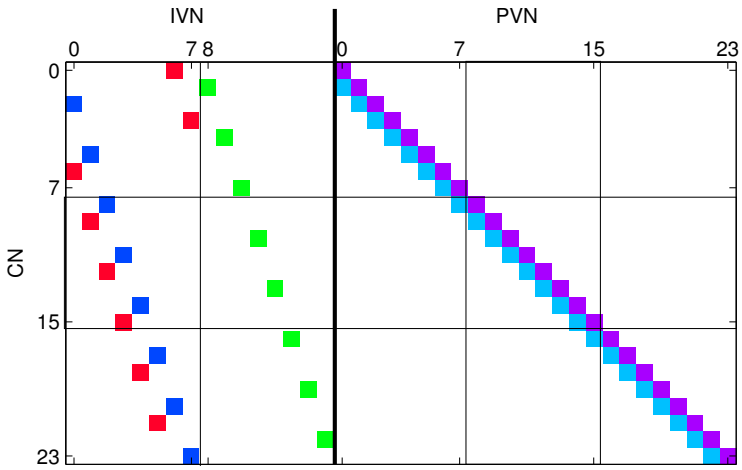


Figure A.1: matrix resulting from Table A.1

computed in Equation (1.19) with the new parameters p=8 and q=3.

$$CN_j = CN_j \oplus IVN_m,\ j = (x + q(m \mod 8)) \mod M. \tag{A.1}$$

The first value in the first line of the table by applying Equation (A.1) will give the connections between VN_0(m=0) and CN_2, VN_1(m=1) and CN_5, and so on. The value in the second line gives the connection between the second group of eight VNs (VN_8 to VN_{15}) and the CNS.

The fixed zigzag connectivity of the PVN and CN is defined by the encoding scheme given by equation A.2.

$$CN_j = CN_j \oplus CN_{j-1},\ j = 1, 2, \ldots, N-1. \tag{A.2}$$

Figure A.1 shows the resulting parity check matrix. For ease of lisibility, the 0 in the parity check matrix are not presented and the 1 are represented by a colored square. The square colors correspond with the x value given in Table A.1.

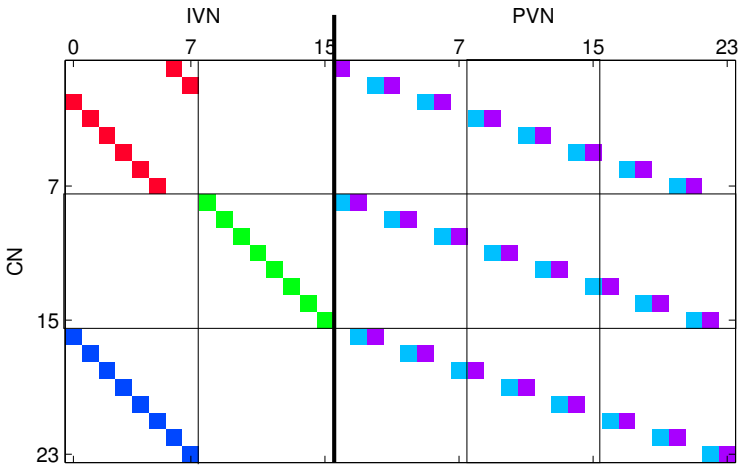


Figure A.2: First permutation

A.2 Matrix permutations for layered structure

The permutation process is splitted in two steps. First a permutation of the CNs with the permutation equation A.3

$$\sigma(j + i \times 8) = i + (j \times Q) \mod N,\ i = 1, 2, 3\ j = 1, 2, \ldots, 8. \qquad \text{(A.3)}$$

Figure A.2 shows the result of this first permutation. It can be seen that the information part of the matrix is now made of identity matrices but the parity part needs a permutation of the PVN. The second step is a permutation of the PVN by following the permutation of Equation (A.3).

Figure A.3 shows the resulting matrix made of IM of size 8. One can observe the particularity of the IM in the upper right corner where the connection between the first CN (CN_0) and the last PVN (PVN_{23}) does not exist. This particularity can be easily implemented but can not be ignored.

From the layered decoder point of view, this permutation allows an efficient decoding of the DVB-X2 matrices. One will note that a permutation of the PVN is required before the decoding process and a patch is needed for the upper right identity matrix.

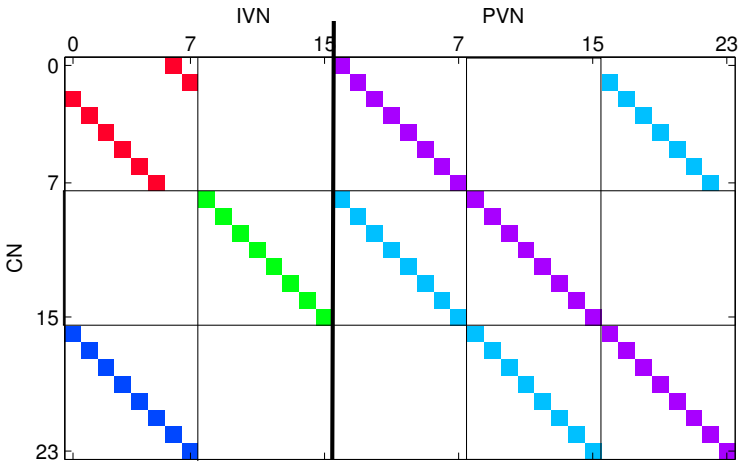


Figure A.3: Second permutation

It does not matter how slowly you go so long as you do not stop.

Confucius (551BC-479BC)

B

Hardware Discrete Channel Emulator

B.1 Introduction

The emulation environment named Hardware Discrete Channel Emulator (HDCE) has been developed as a coherent framework to emulate on a hardware device (FPGA as the implementation platform) and simulate on a computer the effect of an Additive White Gaussian Noise (AWGN) in a base band channel. The HDCE is able to generate more than 180 M samples per second for a very low hardware cost, which has been achieved in an efficient architecture. Using the HDCE, the performance evaluation of a coding scheme for a BER of 10^{-9} requires only one minute of emulation time.

In the state-of-the-art, a Random Value (RV) with an uniform distribution is generated, and this uniform distribution is modified to obtain a gaussian distribution [26, 6, 39]. This accurate white gaussian noise is added to a signal and then quantified. With the HDCE, a quantified output with the required distribution is directly produced.

The uniform distribution is generated using a Linear Feedback Shift Register (LFSR) presented in the first section. Then the alias method is developed and the architecture is presented.

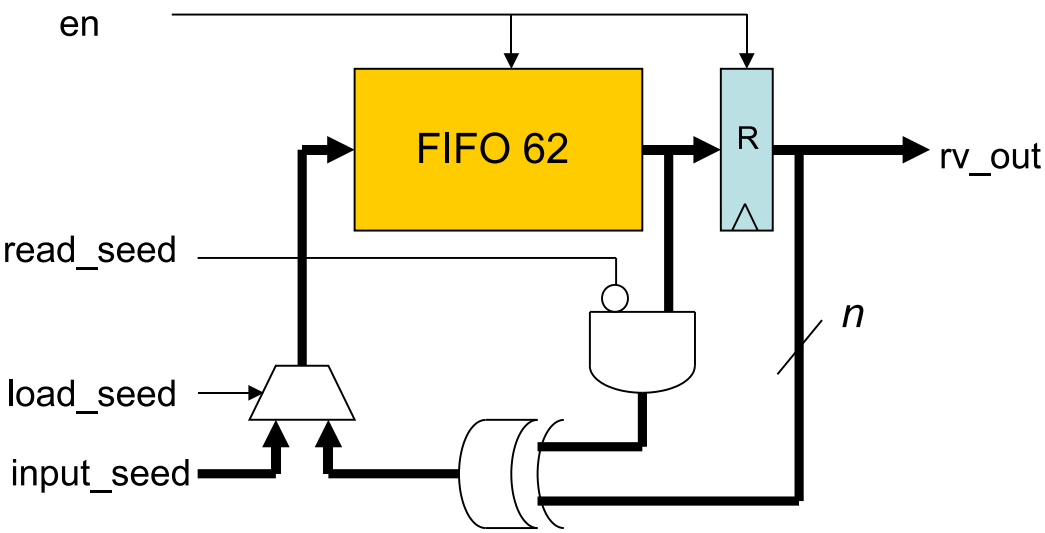


Figure B.1: LFSR

B.2 Linear Feedback Shift Register

A Linear Feedback Shift Register (LFSR) is a shift register whose input is a linear function of its previous state. LFSR can produce a sequence of bits which appears random. In Fig. B.1, LFSR are concatened in a FIFO giving an efficient implementation of a RV generator. Two control signals *read_seed* and *load_seed* are added to allow the possibility to load the LFSR and to read the LFRS in order to be able to replay a sequence.

B.3 The alias method algorithm

The alias method was initially proposed by A. J. Walker in [67]. The alias method algorithm can be described in two steps. In the first step, two independent uniform distributions $Ps0$ and Prv are generated. In the second step, a test is performed on Prv. If $Prv < Threshold(Ps0)$, then the system output is $Ps0$. Otherwise, the system produces the alternative value given by $Ps1 = alias(Ps0)$. The *alias* values and the *threshold* values are stored in the alias table. With a given alias table, a distribution can be predicted. It is also possible to give an alias table in function of the required distribution. The computation of the alias table can be done by software and then the alias table can be updated.

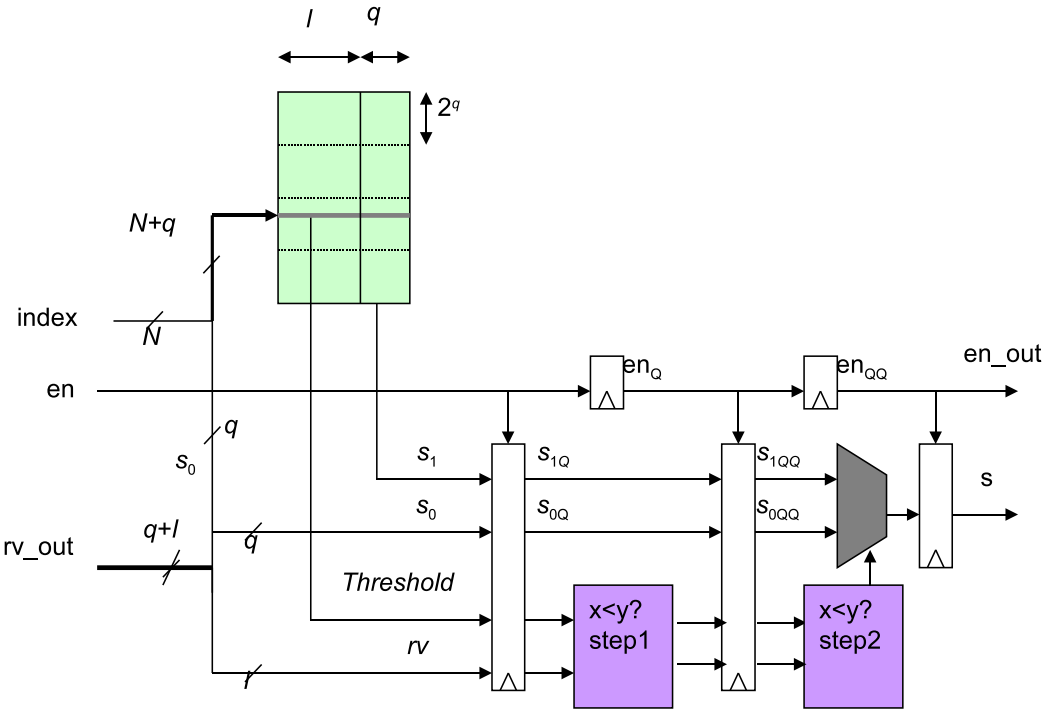


Figure B.2: HDCE architecture

B.4 The HDCE architecture

The implementation of the alias method is straight forward by following the alias method algorithm. A RV generator is instanciated by the use of LFSR and produces the $Ps0$ and Prv signals. A RAM stores the alias table and produces an *Alias* signal and a *Treshold* signal as a function of $Ps0$. A comparator compares the two values *Treshold* and Prv and produces a signal which drives a multiplexeur that selects the Pso signal or the *Alias* signal. Figure B.2 shows an improved HDCE architecture. In this architecture, one uniform distribution is split to generate the two uniform distributions Ps0 and Prv. The comparator is pipelined to decrease the delay and an index is added to address different distributions.

B.5 Resulting distribution

Figure B.3 shows the Probability Density Fonction of the HDCE output with the following parameters: channel quantization on 5 bits, threshold quantification on 10 bits and two distributions. The two ditributions are

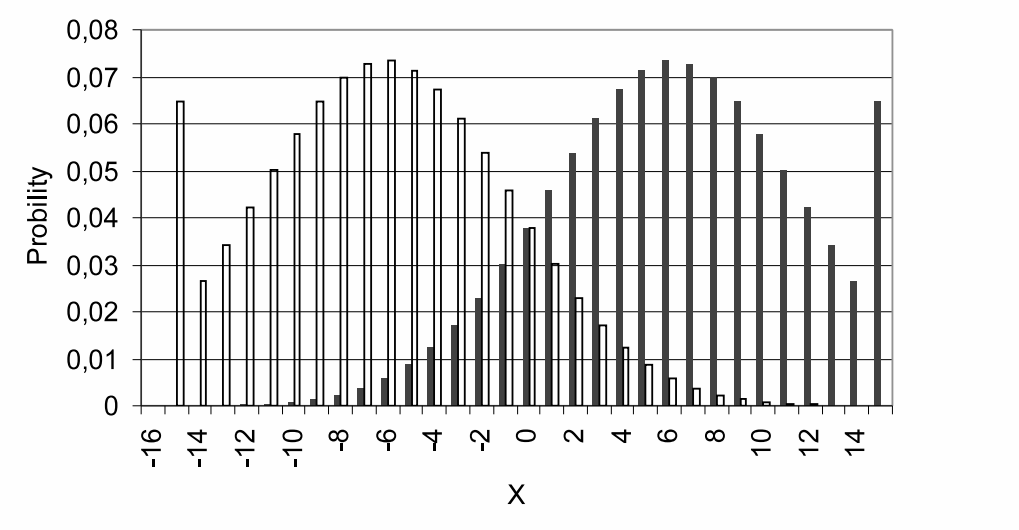


Figure B.3: Resulting distribution of a BPSK modulation emulated by HDCE

emulating a BPSK modulation (-1 and +1) which is perturbed by an AWGN of variance $\sigma = 0.866$ (corresponding to a Eb/No = 2 dB).

Rien ne vaut la recherche lorsque l'on veut trouver quelque chose.

J.R.R.Tolkien (1892-1973)

C Résumé étendu

Résumé

This is a french extented summary of this thesis, as required by the publisher.

C.1 Introduction

Shannon a montré en 1948 qu'il existe une limite au débit d'information transmis en présence de bruit, appelé capacité du canal, mais il n'a pas explicité les moyens de l'approcher. Quarante ans plus tard, Claude Berrou et Alain Glavieuxl [3] ont montré comment réussir à s'approcher de la limite de Shannon. Cette révolution a ouvert de nombreuses voies de recherches dans le domaine des codes correcteurs d'erreurs et notamment la redécouverte par MacKay [41] des codes Gallager [25] aussi appelés codes LDPC (Low density Parity Check) du fait de la faible densité de la matrice de parité. La compréhension de l'algorithme de décodage d'un code LDPC est facilité par sa représation graphique ou graphe de Tanner [63]. Dans un graphe de Tanner, chaque symbole du mot de code est représenté par un nœud appelé Nœud Variable (NV), et chaque contrainte de parité correspondant à chaque ligne de la matrice de parité est représentée par un Nœud de Parité (NP). Les NP et NV sont connectés a chaque fois qu'un 1 apparait dans la matrice de parité. Pendant le processus de décodage, des messages sont échangés entre les VNs et CNs.

Grâce à l'utilisation de matrices structurées, le décodage des codes LDPC a été facilité et les codes LDPC sont maintenant utilisés dans de nombreux standards. Le premier standard à inclure un code LDPC a été la norme de transmission vidéo par satellite de seconde génération (DVB-S2 pour Digital Video Broadcasting by Satellite of 2^{nd} generation) [20]. Les codes LDPC sont maintenant inclus dans de nombreux standard dont les standard WiFi, WiMAX [60], STiMi, DVB-T2 (T pour terrestre)[22] et DVB-C2(C pour Cable)[21]. Ces deux derniers standards ont la particuliaritée d'utiliser les mêmes matrices que les matrices du standard DVB-S2. Les decodeurs LDPC pour les standards DVB-S2, -T2 et -C2 étant identiques, les trois standards seront denommés standards DVB-X2.

Cette thèse porte sur l'optimisation de l'implémentation matérielle d'un décodeur LDPC pour les standards DVB-X2. Après une étude de l'état de l'art, c'est le "layered decoder" ou décodeur par couche qui a été choisi comme architecture de base à l'implémentation du décodeur. Nous nous sommes ensuite confrontés à deux types de conflits de mise à jour mémoire. Les conflicts inhérents à la structure particulière aux standards DVB-X2 sont résolus grâce à deux nouvelles contributions. Une est basée sur la constitution d'une matrice équivalente et l'autre basée sur la répétition de couches (layers). Les conflits mémoire dus au pipeline sont quant à eux suprimés à l'aide d'un ordonnancement des couches et des matrices identités. L'espace mémoire étant un différenciateur majeur de coût d'implémentation, la réduction au minimum de la taille mémoire a été étudiée. Une saturation précise et un partitionnement optimal des bancs mémoires ont permi une réduction significative par rapport à l'état de l'art. De plus, l'utilisation de RAM simple port à la place de RAM double port est étudiée pour réduire le coût mémoire. En dernière partie, nous répondons à l'objectif d'un décodeur capable de décoder plusieurs flux pour un coût réduit par rapport à l'utiliation de multiples décodeurs.

C.2 Pré-requis

Ce premier chapitre présente les termes et pré-requis utilisés dans cette thèse. Nous présentons d'abord d'une manière générale une transmission numérique puis un décodeur de canal. Le principe du décodeur itératif est brièvement rappelé puis le décodeur LDPC et le "layered" décodeur ou décodeur par couche sont davantage expliqués. Le standard DVB est ensuite décrit et plus particulièrement la construction des matrices LDPC à partir du standard est développé. Enfin l'environnement permettant le test du decodeur est présenté.

C.3 Les conflits de mise à jour de la mémoire

Les conflits de mise à jour mémoire ont lieu lorsque deux unités de calcul mettent à jour simultanément une même donnée en mémoire. Le résultat de la première unité de calcul sera alors ecrasée par le second résultat. Dans ce chapitre, deux types de conflits mémoire sont identifiés puis résolus: les conflits dus à la structure de la matrice et les conflits dus au pipelining du CNP.

C.3.1 Conflits dus à la structure

Les matrices définies par les standards DVB-X2 ne sont pas idéalement structurés pour un décodage par couches. En effet, l'existance de Matrice Identités (MI) Superposés entraînent un écrasement des Probabilités A Postériorie (PAP) mis à jour par la première MI (MI_1) par la seconde (MI_2). Pour résoudre ce problème, dans une première étape, une réduction du parallélisme entraîne une forte réduction du nombre de conflits. puis pour la deuxième étape, deux solutions innovantes sont proposées pour résoudre les conflits restants.

Réduction du paralléllisme

La réduction du parallélisme est réalisée à l'aide de la permutation de la matrice de parité de manière à obtenir des matrices identités réduitent par un facteur S et donc de taille $P_s = 360/S$ avec P et s entiers naturels. Ce procédé permet de séparer les MI Supperposées (MIS) avec une probabilité de $1 - 1/S$. Donc une diminution du paralléllisme entraîne une réduction de la probabilité de MIS. Cependant, cette solution à elle seule ne suffit pas car les simulations montrent que toutes les MIS doivent être résolus sous peine de performances qui ne satisfont pas le standard.

Matrice équivalente

Le procédé décrit ici repose sur la construction d'une matrice équivalente. Les couches avec une MIS sont coupées en deux, de manière à séparer les deux IMs de la MIS dans les deux couches obtenues. Pour créer la matrice équivalente, des bits poinçonnés sont additionnés. Ces bits additionnés entrainent un besoin en mémoire supplémentaire ce qui est un désavantage de cette solution. Un autre désavantage de cette solution est un retard dans la propagation des messages qui entraine un plancher d'erreur. Une modification lors de la lecture permet de régler ce plancher d'erreur et de rendre cette méthode

efficace pour résoudre un nombre restraint de MIS. Cette méthode se révèle moins efficace avec un parallélisme élevé où le nombre de MIS à résoudre est important, entrainant une baisse notable des performances et surtout une baisse de débit. Etant donné notre volonté de pouvoir faire face à des débits élevés avec de forts degrés de parallélisme une autre solution a été envisagée.

Répétition des couches

Cette solution consiste à répéter les couches qui comportent des MIS. Pour eviter l'écrasement des données de MI_1, un signal empèche l'écriture de MI_2. Les données de MI_2 seront ensuite mises à jour lors de la répétition. Cette méthode entraîne une faible perte de performance (inférieure à 0.1 dB) même avec un parallélisme maximum et sans perte de débit.

C.3.2 Conflits dus au pipelining

Le pipelining des processeurs permet de doubler le débit mais il y a un risque d'utiliser des Probabilités A Postériori(PAP) qui ne sont par encore mis à jour. Pour cela, il suffit d'éviter d'avoir des PAPs en commun entre deux couches qui se suivent. Il est possible de changer l'ordre ou l'ordonnancement des couches de manière à réduire le nombre de conflits. Avec un parallélisme de 360, il n'y a pas de solution sans conflit possible, mais en réduisant le parallélisme, des solutions peuvent être trouvées. En réduisant le parallélisme par un facteur s, le nombre de couches est mutiplié par s et l'ensemble des combinaisons possibles devient impossible à calculer par la force brute (évaluation de l'ensemble des possibilitées) et il devient nécessaire de faire appel à la théorie des graphes pour trouver une solution dans un temps limité. L'utilisation d'un algorithme génétique permet de rechercher des solutions pour tous les parallélismes. elle permet également, après une simple modification de la fonction de coût, de trouver des solutions pour qu'il n'y ait pas de PAPs en commun entre une couche et les deux suivantes à tous moments.

C.4 Optimisation de la taille mémoire

Par optimisation est sous entendue la taille du décodeur et sa consommation. Bien entendu cette optimisation doit se faire sans dégradation des performances. Etant donné que 70 à 80 % du décodeur est occupé par l'espace mémoire, l'optimisation du décodeur passe essentiellement par l'optimisation de la mémoire. L'optimisation de la mémoire est réalisée sous trois approches.

D'une part, l'optimisation de la taille des mots à enregistrer (et à traiter) entraîne une diminution de la taille mémoire (et une diminution de la complexité des processeurs). D'autre part, la flexibilité demandée par les rendements de codage entraîne l'utilisation d'une mémoire importante pour stocker les méssages extrinsèques qui peut être réduite par une légère modification de l'architecture et un choix judicieux de la taille des bancs mémoire. Enfin, l'implémentation de la mémoire en utilisant uniquement des RAM simple port entraîne une diminution de surface et de consommation par rapport à l'utilisation de RAM double ports.

C.4.1 Optimisation de la taille des mots

L'optimisation de la taille des mots est réalisée à l'aide d'une saturation efficace des mots lors des différentes étapes de calculs. Lors de l'initialisation, la saturation et le facteur d'échelle sont calculés au mieux pour une utilisation optimal de la mémoire. Les PAPs sont saturées en tenant compte de la corrélation entre les PAPs et les messages extrinsèques à l'aide d'un algorithme spécifique. Enfin les extrinsèques sont saturés avant d'être mémorisés.

C.4.2 Optimisation des bancs mémoire des extrinsèques

Une première optimisation de la taille mémoire des extrinsèques consiste a utiliser un algorithme de type Min-Sum. Au lieu de mémoriser les messages de toutes les connexions d'un NP, seulement la valeur absolue du plus petit message, son index, la seconde plus petite valeur et le signe des messages sont enregistrés. Le gain est d'autant plus important que le degré des NP (d_c) est important. La taille de la mémoire des extrinsèques pour une matrice est donnée par la taille du mot(fonction de d_c et de la quantification des extrinsèques) multipliée par la taille de l'adressage (définie par le nombre de NPs M).

Le problème est que d_c et le nombre de NP varie en fonction du taux de codage. Pour satisfaire cette variation, une solution consiste à choisir une mémoire avec la taille de mots maximal multipliée par la valeur maximal de M. La solution proposée consiste à pouvoir reconfigurer des bancs mémoires de manière à faire varier la taille des mots et de l'adressage. Une taille de banc optimale est ensuite déterminée.

C.4.3 Utilisation de RAM simple port

L'utilisation de RAM simple port à la place de RAM double port permet de faire un gain de surface et une réduction de consommation. L'utilisation

de RAM simple port est surtout problématique pour la mémoire PAP. La solution proposée repose sur un ordonancement des couches et si besoin, un ordonnancement des groupes de variables a l'intérieur des couches.

C.5 Un décodeur de flux multiple

Un décodeur de flux multiple peut être vu comme un décodeur capable de décoder plusieurs types de flux. Un décodeur peut être susceptible de décoder un flux de télévision standard (SDTV) puis un flux de télévision haute définission (HD TV) ou bien encore un flux de télévision en trois dimensions (3D TV). Du point de vue du décodeur LDPC, la différence entre ces trois flux s'interpréte en terme de débit. En effet la différence de débit entre un flux SDTV et 3D TV est un rapport de l'ordre de 1 à 5. Un décodeur de flux multiple peut aussi être interprété comme un décodeur capable de traiter x flux simultanément. A nouveau, au niveau du décodeur cela peut être interprété en terme de débit. Le problème peut donc être vu comme un problème de débits variable.

Nous allons traiter le décodeur de flux multiple sous trois angles: le parallélisme, le partage des ressources et l'addition d'un buffer à un decodeur itératif.

C.5.1 Parallélisme

La variation du parallélisme permet de répondre facilement à une demande de débit supplémentaire. Cependant, une fois un parallisme établi, le parallélisme ne peut pas être modifié et le débit sera toujours maximal.

C.5.2 Partage des ressources

Le traitement de deux flux simultanément permet de partager les ressources des processeurs de calcul. Cependant la mémoire ne peut pas être partagé et le partage des processeurs de calcul n'apporte pas de gain par rapport à un processeur pipeliné. L'avantage de cette solution peut venir d'une diminution des contraintes d'accés mémoire et de ce fait, peut fournir des solutions aux conflcts mémoire en cas de fort parallélisme.

C.5.3 Addition d'un buffer à un décodeur itératif variable

Un décodeur itératif variable est capable de détecter si un mot de code a été trouvé et ainsi arrêter le décodage en cours et commencer le décodage d'un nouveau mot ou bien s'arrêter pour économiser de l'énergie. L'avantage de ce type de décodeur vient du fait que le nombre d'itérations pour décoder un mot de code est réduit d'une valeur maximale à une valeur moyenne. L'inconvénient est qu'un buffer est néscessaire en entrée et en sortie du décodeur pour palier au caractère asynchrone du décodeur. La taille du buffer est réduite au minimum par simulation sans perte de performance et un débit doublé par rapport à un décodeur à iterations constantes. L'addition d'un buffer permet aussi de répondre au caractère non constant du débit imposé par de multiple flux.

C.6 Conclusion

Pendant cette thèse nous avons cherché à optimiser l'implémentation d'un décodeur LDPC pour les standards DVB-S2, -T2 et -C2. Nous avons d'abord implementé un décodeur par couche sur FPGA capable de décoder des codes de type "layered" comme ceux definis par les standards WiFi ou WiMAX. Ce décodeur intègre les optimisations concernant la taille mémoire. Une deuxième version améliorée permet de décoder des flux des standards DVB-S2, -T2 et -C2. Ce nouveau décodeur est capable de répéter les couches et de désactiver l'écriture des mémoire PAP et extrinsèque permettant ainsi de résoudre efficacement les conflicts de mise à jour dus à la structure particulière des matrices des standards DVB-X2.

C.6.1 Applications

Les optimisations mémoires sont applicables pour tous les décodeur de type "layered", et il en est de même pour la résolution des conflicts de mis à jour de la mémoire dus au pipeline des processeurs. La résolution des conflicts de mise à jour mémoire dus aux IM supperposées peut être étendue aux standard STiMi qui comporte lui aussi des MI supperposés. L'etude concernant les décodeurs multi flux peut, quant à elle, être etendue à l'ensemble des décodeurs itératifs.

C.6.2 Perspectives

L'architecture proposée peut être opimisée en surface, vitesse et performance. Les performances pour les codes de rendement de codage faible (1/4, 1/3 et 2/5) doivent être améliorées pour remplir les conditions inposées par les standards. Une solution simple consiste à utiliser une version améliorée de l'algorithme Min-Sum tel que l'agorithme $\lambda - min$ ou $A - min*$. L'environnement de test peut être etendue en incluant la possibilité d'émuler un canal de Rayleigh et en intégrant un code BCH pour compléter l'émulation de la transmission numérique.

List of Figures

List of Tables

Bibliography

[1] Andreev, Alexander, Bolotov, Anatoli, Scepanovic, and Ranko. Fifo memory with single port memory modules for allowing simultaneous read and write operations. *US patent 7181563*, February 2007.

[2] C. Berrou. The ten-year-old turbo codes are entering into service. In *Communication Magazine, IEEE*, pages 110–116, August 2003.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. volume 2, pages 1064–1070 vol.2, 1993.

[4] T. Bhatt, V. Sundaramurthy, V. Stolpman, and D. McCain. Pipelined block-serial decoder architecture for structured LDPC codes. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 4, page IV, Toulouse, France, May 2006.

[5] E. Boutillon, J. Castura, and F.R. Kschischang. Decoder first code design. In *In Proc. 2nd International Symposium on Turbo Codes & Related Topics*, pages 459–462, Brest, France, September 2000.

[6] E. Boutillon, J.L Danger, and A. Ghazel. Design of high speed AWGN communication channel emulator. In *Proc. AICSP*, volume 34, pages 133–142, 2003.

[7] E. Boutillon, C. Douillard, and G. Montorsi. Iterative decoding of concatenated convolutional codes: Implementation issues. In *Proceedings of the IEEE*, volume 95, June 2007.

[8] E. Boutillon and F. Guilloud. LDPC decoder, corresponding method, system and computer program. *US patent 7,174,495 B2*, February 2007.

[9] E. Boutillon, Y. Tang, C. Marchand, and P. Bomel. Hardware discrete channel emulator. In *IEEE International Conference on High Perfor-*

mance Computing and Simulation (HPCS 2010), pages 452–458, Caen, France, June 2010.

[10] T. Brack, M. Alles, F. Kienle, and N. Wehn. A synthesizable IP core for WIMAX 802.16e LDPC code decoding. In *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pages 1–5, Helsinki, Finland, September 2006.

[11] J. Chen and M. Fossorier. Density evolution of two improved bp-based algorithms for LDPC decoding. *IEEE Communication letters*, March 2002.

[12] A. Darabiha, A. C. Carusone, and F. R. Kschischang. Block-interlaced ldpc decoder with reduced interconnect complexity. *IEEE transaction on circuits and systems-II: Express Brief*, 55:74–78, January 2008.

[13] J. Dielissen and A. Hekstra. Non-fractional parallelism in LDPC decoder implementations. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 1–6, May 2007.

[14] J. Dielissen, A. Hekstra, and V. Berg. Low cost LDPC decoder for DVB-S2. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 2, pages 1–6, Munich, Germany, March 2006.

[15] D. Divsalar, S. Dolinar, and C. Jones. Construction of protrograph LDPC codes with linear minimum distance. In *Information Theory, 2006 IEEE International Symposium on*, pages 664–668, Washington, USA, July. 2006.

[16] J.B. Doré. *Optimisation conjointe de codes LDPC et de leurs architecture de decodage et mise en oeuvre sur FPGA*. PhD thesis, INSA, Rennes, France, 2007.

[17] Digital Video Broadcasting (DVB). Framing structure, channel coding and modulation for 11/12 ghz satellite services. *ETSI EN 300 421 (V1.1.2)*, 1994.

[18] Digital Video Broadcasting (DVB). Digital video broadcasting (dvb); framing structure, channel coding and modulation for digital satellite news gathering (dsng) and other contribution applications by satellite. *ETSI EN 301 210*, 1997.

[19] Digital Video Broadcasting (DVB). Framing structure, channel coding and modulation for digital terrestrial television. *ETSI EN 300 744 (v1.6.1*, 2009.

[20] Digital Video Broadcasting (DVB). Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (DVB-S2). *European Standard (Telecommunications series)ETSI EN 302 307 V1.2.1 (2009-08)*, 2009.

[21] Digital Video Broadcasting (DVB). Frame structure channel coding and modulation for a second generation digital transmission system for cable systems (DVB-C2). *DVB Document A138*, 2010.

[22] Digital Video Broadcasting (DVB). Frame structure channel coding and modulation for the second generation digital terrestrial television broadcasting system (DVB-T2). *DVB Document A122*, 2010.

[23] O. Eljamaly and P. Sweeney. Alternative approximation of check node algorithm for DVB-S2 LDPC decoder. In *Second International Conference on Systems and Networks Communications (ICSNC 2007)*, pages 157–162, October 2007.

[24] M.P.C Fossorier, M. Mihaljevic, and H. Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on communications*, 47:673–680, May 1999.

[25] R.G. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, Cambridge, 1963.

[26] A. Ghazel, E. Boutillon, J.L Danger, and G.Gulak. Design and performance analysis of a high speed awgn communication channel emulator. In *Proc. PACRIM*, volume 2, pages 374–377, 2001.

[27] M. Gomes, G. Falcao, V. Silva, V. Ferreira, A. Sengo, and M. Falcao. Flexible parallel architecture for DVB-S2 LDPC decoders. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 3265–3269, Washington, USA, November 2007.

[28] M. Gones, G. Falcao, J. Goncalves, V. Silva, M. Falcao, and P. Faia. HDL library of processing units for generic and DVB-S2 LDPC decoding. In *International Conference on Signal Processing and Multimdia Applications (SIGMAP2006)*, Setubal, Portugal, 2006.

[29] F. Guilloud, E. Boutillon, and J.-L. Danger. lambda-min decoding algorithm of regular and irregular LDPC codes. *Proceedings of the 3rd International Symposium on Turbo Codes and Related Topics*, September 2003.

[30] K. Guo, Y. Hei, and S. Qiao. A parallel-layered belief-propagation decoder for non-layered ldpc codes. In *Journal of Communications*, volume 5, pages 400–408, May 2010.

[31] D.E. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, pages 107–112, Austin, USA, October 2004.

[32] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[33] Xiao-Yu Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia. Efficient implementations of the sum-product algorithm for decoding LDPC codes. In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, volume 2, pages 1036–1036E vol.2, 2001.

[34] H. Jin, K. handekar, and R. McEliece. Irregular repeat-accumulate codes. In *In Proc. 2nd International Symposium on Turbo Codes & Related Topics*, pages 1–8, Brest, France, September 2000.

[35] C. Jones, E. Valles, M. Smith, and J. Villasenor. Approximate-min* constraint node updating for LDPC code decoding. In *IEEE Military Communication Conference*, pages 157–162, October 2003.

[36] F. Kienle, T. Brack, and N. Wehn. A synthesizable IP core for DVB-S2 LDPC code decoding. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 100–105, Munich, Germany, March 2005. IEEE Computer Society.

[37] S. L. Fogal, S. Dolinar, and K. Andrews. Buffering requirement for variable-iteration ldpc decoder. In *Information Theory and Application Workshop,ITA.2008*, pages 523–530, San Diego, CA, January 2008.

[38] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artif. Intell. Rev.*, 13(2):129–170, 1999.

[39] D. Lee, W. Luk, J.V. Villasenor, G. Zhang, and P. Leong. A hardware gaussian noise generator using the box-muller method and its error analysis. In *IEEE Trans Computers*, volume 55, pages 659–671, 2006.

[40] Y.C Liao, C.C Lin, H.C Chang, and C.W Liu. Self-compensation technique for simplified belief-propagation algorithm. In *IEEE Transaction on Signal Processing*, pages 3061–3072, vol.55, June 2007.

[41] D.J.C. MacKay and R.M. Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 33:457–458, March 1997.

[42] M. M. Mansour and N. R. Shanbhag. High-throughput LDPC decoders. *IEEE Transactions on Very Large Scale Integration VLSI Systems*, 11:976–996, December 2003.

[43] M.M. Mansour and N.R. Shanbhag. Low-power VLSI decoder architectures for LDPC codes. In *Low Power Electronics and Design, 2002. ISLPED '02. Proceedings of the 2002 International Symposium on*, pages 284–289, Monterey, USA, August 2002.

[44] C. Marchand, L. Conde-Canencia, and E. Boutillon. Architecture and finite precision optimization for layered LDPC decoders. In *Signal Processing Systems, 2010. SiPS 2010. IEEE Workshop on*, San Francisco, USA, October 2010.

[45] C. Marchand, J.-B. Doré, L. Conde-Canencia, and E. Boutillon. Conflict resolution by matrix reordering for DVB-T2 LDPC decoders. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–6, Honolulu, USA, November 2009.

[46] C. Marchand, J.-B. Doré, L. Conde-Canencia, and E. Boutillon. Conflict resolution for pipelined layered LDPC decoders. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 220–225, Tampere, Finlande, November 2009.

[47] L. Meng, C Abdel Nour, C. Jego, and C. Douillard. Design and FPGA prototype of a bit-interleaved coded modulation receiver for the DVB-T2 standard. In *IEEE workshop on Signal Processing System. SIPS 2010*, pages ???–???, Sam Francisco, USA, October 2010.

[48] S. Muller, M. Schreger, M. Kabutz, M. Alles, F. Kienle, and N. Wehn. A novel LDPC decoder for DVB-S2 IP. In *Design, Automation & Test in Europe Conference & Exhibition,2009. Date '09.*, Nice, France, April 2009.

[49] S. Papaharalabos and P.T. Mathiopoulos. Simplified sum-product algorithm for decoding LDPC codes with optimale performance. *Electronics letters*, 45:536–539, June 2009.

[50] J.G. Proakis. *Digital communication, Fourth Edition.* McGraw-Hill International Editions, 2000.

[51] DAVINCI project. *Design And Versatil Implementation of Non-binary wireless Communications based on Innovative LDPD Codes, (DAVINCI)*. DAVINCI project, 2010.

[52] M. Rovini and A. Martinez. On the addition of an input buffer to an iterative decoder for LDPC codes. In *IEEE 65th Vehicular Technologie Conference,VTC2007*, pages 1995–1999, Dublin, Ireland, April 2007.

[53] M. Rovini, F. Rossi, P. Ciao, N. L'Insalata, and L. Fanucci. Layered decoding of non-layered LDPC codes. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 537–544, Dubrovnick, Croatia, September 2006.

[54] Massimo Rovini, Giuseppe Gentile, Francesco Rossi, and Luca Fanucci. A minimum-latency block-serial architecture of a decoder for IEEE 802.11n LDPC codes. In *Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on*, pages 236–241, Atlanta, USA, October 2007.

[55] V. Savin. Self-corrected min-sum decoding of ldpc codes. pages 146–150, 2008.

[56] V. Savin. Split-extended ldpc codes for coded cooperation. In *International Symposium on Information Theory and its Applications*, pages ???–???, Taichung, Taiwan, October 2010.

[57] A. Segard, F. Verdier, D. Declercq, and P. Urard. A DVB-S2 compliant LDPC decoder integrating the horizontal shuffle schedule. In *IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2006)*, Tottori, Japan, December 2006.

[58] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379423, 1948.

[59] A. Singh, A. Al-Ghouwayel1, G. Masera, and E. Boutillon. A new performance evaluation metric for sub-optimal iterative decoders. *IEEE Communications letters*, 13:513–515, 2009.

[60] IEEE std. Air interface for fixed and mobile broadband wireless access systems. In *P802.16e/D12 Draft*, pages 100–105, Washington, DC, USA, 2005. IEEE.

[61] IEEE Std. Wireless lan medium access control (MAC) and physical layer (PHY) specifications-enhancement for higher throughput(draft). *IEEE P802.11n/D1.05*, October 2006.

[62] Y. Sun, M. Karkooti, and J.R. Cavallaro. High throughput, parallel, scalable LDPC encoder/decoder architecture for OFDM systems. In *Design, Applications, Integration and Software, 2006 IEEE Dallas/CAS Workshop on*, pages 39–42, Richarson, USA, October 2006.

[63] R. Tanner. A recursive approach to low complexity codes. *Information Theory, IEEE Transactions on*, 27:533–547, September 1981.

[64] C.J Tsai and M.C Chen. Efficient ldpc decoder implementation for DVB-S2 system. In *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*, pages 37–40, June 2010.

[65] P. Urard, L. Paumier, V. Heinrich, N. Raina, and N. Chawla. A 360mw 105b/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes enabling satellite- transmission portble devices. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 310–311, San Francisco, USA, February 2008.

[66] P. Urard, E. Yeo, L. Paumier, P. Georgelin, T. Michel, V. Lebars, E. Lantreibecq, and B. Gupta. A 135mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes. In *Solid-State Circuit Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pages 446–447, Sam Francisco, USA, February 2005.

[67] A.J. Walker. An efficient method for generating discrete random variables with general distributions. In *ACM Transaction on Mathematical Software (TOMS)*, volume 3, September 1977.

[68] Y. Yan, B. Dan, H. Shuangqu, Bo X., C. Yun, and Z Xiaoyang. A cost efficient LDPC decoder for DVB-S2. In *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*, pages 1007 – 1010, Changsa,China, October 2009.

[69] J. Zhang, Y. Wang, M. P. C. Fossorier, and J. S. Yedidia. Iterative decoding with replicas. *Information Theory, IEEE Transactions on*, 53:1644–1663, May 2007.

[70] Juntan Zhang and M.P.C. Fossorier. Shuffled iterative decoding. *Communications, IEEE Transactions on*, 53:209–213, February 2005.