PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE ENGENHARIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

AUGUSTO CALCANHOTTO MENGARDA

CORE* LDPC PARA O PADRÃO DVB-S2 – *DIGITAL VIDEO BROADCASTING – SATELLITE GENERATION 2

Porto Alegre – RS, Brasil

2016

AUGUSTO CALCANHOTTO MENGARDA

CORE LDPC PARA O PADRÃO DVB-S2 – *DIGITAL VIDEO BROADCASTING – SATELLITE GENERATION 2*

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica do Rio Grande do Sul, como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Sinais, Sistemas e Tecnologia da Informação.

Linha de Pesquisa: Telecomunicações.

Orientador: Fernando César Comparsi de Castro

Porto Alegre – RS, Brasil

2016

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

CORE LDPC PARA O PADRÃO DVB-S2 - DIGITAL VIDEO BROADCASTING - SATELLITE GENERATION 2

CANDIDATO: AUGUSTO CALCANHOTTO MENGARDA

Esta Dissertação de Mestrado foi julgada para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica do Rio Grande do Sul.

DR. FERNANDO CÉSAR COMPARSI DE CASTRO - ORIENTADOR

BANCA EXAMINADORA

DR. AVELINO FRANCISCO ZORZO - PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO - PUCRS

DR. DARIO F. GUIMARÃES DE AZEVEDO - DO PPGEE/FENG - PUCRS

PUCRS
Campus Central
Av. Ipiranga, 6681 - Prédio 30 - Sala 103 - CEP: 90619-900
Telefone: (51) 3320.3540 - Fax: (51) 3320.3625
E-mail: engenharia.pg.eletrica@pucrs.br
www.pucrs.br/feng

Resumo

O padrão *Digital Video Broadcasting – Satellite Generation 2* (DVB-S2) é amplamente utilizado em comunicações via satélite, para operações nas áreas de defesa e de comunicação civil. Devido à distância entre transmissor e receptor, enlaces de comunicação via satélite operam com baixa relação sinal-ruído. Técnicas de *Forward Error Correction* (FEC) são de particular importância no desempenho de sistemas DVB-S2, garantindo a performance desejada. Esta dissertação de mestrado apresenta o desenvolvimento de um *core*, em lógica programável, de um *codec* LDPC (*Low-Density Parity-Check*) compatível com o padrão DVB-S2. O *core* opera com os dois tamanhos de frames e as vinte e uma taxas de codificação previstas no padrão. A dissertação aborda os principais desafios de implementação do *codec* em *hardware* e como os mesmos são enfrentados. Três versões da arquitetura proposta são implementadas e avaliadas, utilizando diferentes representações numéricas das variáveis do sistema, em ponto fixo. Os resultados de simulação do core VHDL são balizados através de simulações em linguagem de programação C, utilizando ponto flutuante. Os resultados obtidos demonstram que o *core* proposto apresenta desempenho equivalente ou superior aos relatados em literatura quando utilizada a menor representação numérica implementada. No entanto, quando avaliada a arquitetura de maior representação numérica, os resultados do core proposto nesta dissertação são significantemente superiores aos apresentados em literatura, e próximos aos resultados obtidos nas simulações em C, utilizando representação de 64 bits em ponto flutuante. Além das avaliações de desempenho, são apresentados os recursos de *hardware* utilizados para cada uma das três implementações propostas, sendo realizada a análise quanto a desempenho versus ocupação de recursos FPGA (*Field-Programmable Gate Array*).

Palavras-chave: Codificação de Canal, LDPC, DVB-S2, VHDL, FPGA

Abstract

Digital Video Broadcasting – Satellite Generation 2 (DVB-S2) standard is widely adopted for militar and civil communication. Due to the long distance between transmitter and receiver, satellite communication links operate with low signal to noise ratio. Forward Error Correction (FEC) techniques are of particular importance for DVB-S2 systems, ensuring the desired performance. This dissertation presents the development of a core, in hardware description language, of a LDPC (Low-Density Parity-Check) codec compatible with the DVB-S2 standard. The developed core operates with two sizes of frames and twenty-one encoding rates, as defined in the DVB-S2 standard. The dissertation addresses the main challenges regarding the codec implementation and how they are faced. Three versions of the proposed architecture are implemented and evaluated. Each version uses a different numerical representation for the codec variables. VHDL simulation results are compared with simulations in C programming language, which uses floating point. The results show that the proposed core has equivalent or superior performance to those works reported in the literature when using the architecture with the smallest numerical representation. However, when evaluated the architecture with the highest numerical representation, the obtained FEC performance is significantly better than those presented in the literature, and are close to the results obtained with 64 bits floating point representation. In addition to the performance evaluation, the use of the FPGA (*Field-Programmable Gate Array*) resources are presented for each one of the three implemented architectures. The analysis of performance versus FPGA resources is addressed.

Keywords: Channel coding, LDPC, DVB-S2, VHDL, FPGA.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACM	*Adaptative Coding and Modulation*
APSK	*Amplitude and Phase-Shift Keying*
BCH	*Bose-Chaudhuri-Hocquenghem*
BEC	*Backward Error Correction*
BPSK	*Binary Phase-Shift Key*
BS	*Barrel-Shifter*
dB	*Decibel*
DSP	*Digital Signal Processing*
DVB	*Digital Video Broadcasting*
DVB-RCS	*Digital Video Broadcasting – Return Channel System*
DVB-S2	*Digital Video Broadcasting – Satellite Generation 2*
FEC	*Forward Error Correction*
FPGA	*Field-Programmable Gate Array*
FSM	*Finite State Machine*
FU	*Functional Unit*
GPU	*Graphics Processing Unit*
IRA	*Irregular Repeat-Accumulate*
Kb	*Kilobit*
LDPC	*Low-Density Parity Check*
LLR	L*og-Likelihood Ratio*
LUT	*Look-Up Table*
Mb	*Megabit*
PCM	*Parity-Check Module*

PNAE Programa Nacional de Atividades Espaciais

PNBL Programa Nacional de Banda Larga

PNM Programa Nacional de Microeletrônica

PSK *Phase-Shift Keying*

QPSK *Quadrature Phase-Shift Keying*

RAM *Random Access Memory*

ROM *Read Only Memory*

SGDC Satélite Geoestacionário de Defesa e Comunicações Estratégicas

SPA *Sum-Product Algorithm*

SRL *Super Logic Region*

VHDL *VHSIC Hardware Description Language*

XOR *Exclusive-OR*

Sumário

1 Introdução

1.1 Agradecimentos

Esta dissertação de mestrado foi viabilizada através da implementação de uma Bolsa na Modalidade Mestrado, concedida através do ao Edital MCTI/CNPq Nº 20/2013 - PNM (GM e GD), relativo ao Programa Nacional de Microeletrônica. O projeto contemplado, intitulado “Desenvolvimento de IP core para correção de erro no standard DVB-S2 (Digital Video Broadcasting - Satellite Generation 2)”, enquadra-se no contexto das demandas por autonomia tecnológica nacional nas áreas de defesa e comunicações.

1.2 Motivação

É de amplo conhecimento que a demanda por acesso a serviços de comunicação de dados e de internet tem sido crescente nos últimos anos. Os grandes centros urbanos contam com infraestrutura e disponibilidade de serviços de diversas operadoras de telefonia. O mesmo não ocorre, no entanto, em pequenas localidades e em áreas rurais. Em muitos casos, não existe sequer disponibilidade de serviços de comunicação de dados e internet em tais localidades.

Neste contexto, foi estabelecido no Brasil o Programa Nacional de Banda Larga (PNBL), o qual busca prover acesso à Internet em áreas rurais, regiões remotas e localidades não atendidas pela infraestrutura de telecomunicações existente no País.

Em função da grande extensão territorial do Brasil, e do consequente número de localidades de difícil acesso, é de grande interesse a disponibilização de serviços de dados/internet através de satélites geoestacionários, bem como através de satélites de baixa órbita. Esta alternativa de acesso à Internet constitui, portanto, uma opção viável para o desenvolvimento destas regiões, muitas vezes economicamente ativas, embora segregadas dos grandes centros.

Desta forma está o PNBL relacionado ao Programa Nacional de Atividades Espaciais (PNAE). O PNAE prevê o lançamento e a operação de satélites geoestacionários e de baixa órbita, em uma escala de tempo que se estende até 2020.

O citado programa, que é um dos eixos da recente opção estratégica do Brasil pelo fortalecimento do cenário nacional das áreas de defesa e comunicações, conduz assim a pesquisa científica nacional em setores estratégicos a um patamar adequado de autonomia tecnológica.

O primeiro projeto do PNAE destina-se a implantar o Satélite Geoestacionário de Defesa e Comunicações Estratégicas (SGDC), projeto que teve origem, à época, nos Ministérios das Comunicações, da Defesa e de Ciência, Tecnologia e Inovação.

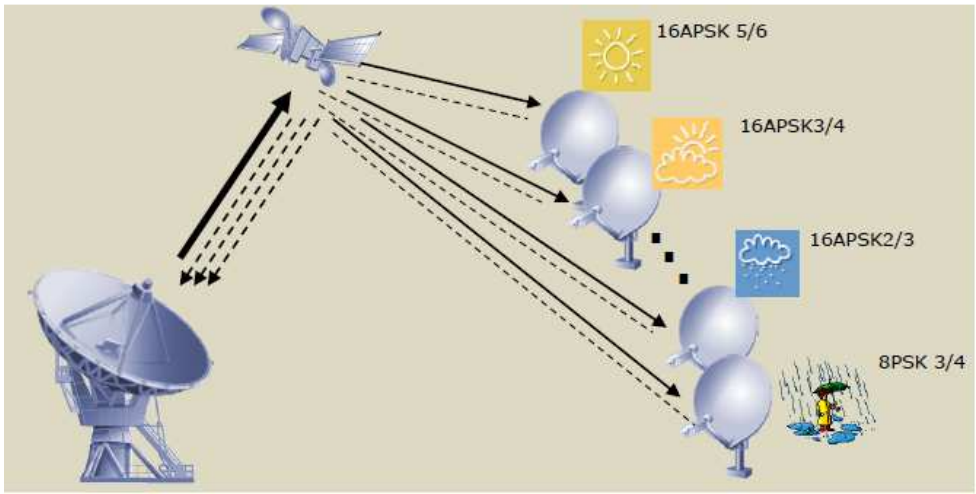
O SGDC utilizará a banda Ka, e permitirá ampliar a cobertura do PNBL, levando acesso à internet às regiões onde atualmente o sistema de telecomunicações é precário ou inexistente, além de atender às necessidades de comunicação estratégica, na área de defesa nacional.

Serviços de dados/internet através de satélite são baseados nos padrões *Digital Video Broadcasting* (DVB) (ETSI EN 302 307 V 1.2.1, 2009) (ETSI TR 102 376 V 1.1.1, 2005). Com a necessidade de maior eficiência nos serviços de comunicação utilizando satélites, foram desenvolvidos dois padrões: o padrão *Digital Video Broadcasting – Satellite Generation 2* (DVB-S2), responsável pelo *uplink* (*forward link*) e o *Digital Video Broadcasting – Return Channel System* (DVB-RCS), responsável pelo *downlink (return link).*

O padrão DVB-S2 apresenta uma interface de ar adaptativa, permitindo diferentes combinações de modulação e codificação, fator que possibilita a maximização do desempenho do sistema. A flexibilidade de modulação e codificação ocorre quando o sistema está operando no modo *Adaptative Coding and Modulation* (ACM) (ALBERTY, et al., 2007). Este modo de operação permite ao sistema se adaptar de acordo com as condições do canal de transmissão, sendo esta uma das principais diferenças com relação ao padrão anterior, o padrão DVB-S. Para que a adaptação do sistema ocorra, é necessário que

informações sobre o estado do canal estejam disponíveis. O *link* de retorno, DVB-RCS, é o responsável pelo envio de tais informações. A Figura 1.1 apresenta, de forma exemplificada, o modo como atua o modo ACM. A medida que o canal de transmissão sofre mudanças, as características do sistema se adaptam a essas condições. Quanto melhor for a condição do canal de transmissão, mais densa é a modulação e maior é a taxa de código utilizadas. A medida que as condições do canal forem piorando, a modulação vai se tornando menos densa e a taxa de código fica menor.

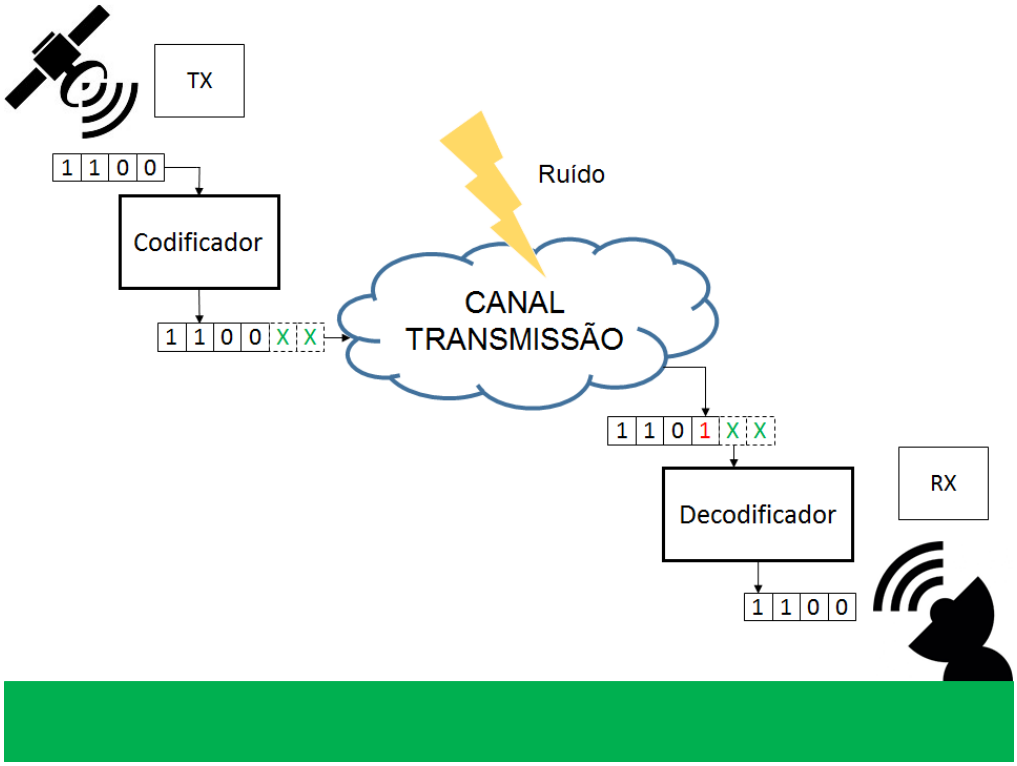
Figura 1.1 – Exemplo de funcionamento do Modo ACM



Fonte: ETSI (2007)

O DVB-S2 utiliza codificação do tipo *Forward Error Correction* (FEC), especialmente o código *Low-Density Parity Check* (LDPC), técnica bastante eficiente, de latência relativamente reduzida e capaz de atingir taxas de transmissão próximas à capacidade do canal, o que é altamente desejável quando se trata de enlaces via satélite. A Figura 1.2 ilustra o funcionamento de uma codificação do tipo FEC. No transmissor existe um bloco codificador que adiciona informação redundante à mensagem a ser transmitida, formando a palavra-código. Essa palavra-código é transmitida através do canal de transmissão e, mesmo que essa palavra-código seja recebida com erros no receptor, a adição dessa informação redundante vai auxiliar o receptor, através do bloco decodificador, a corrigir a palavra-código recebida com erro a partir de um algoritmo corretor, nesse caso o código LDPC.

Figura 1.2 – Contextualização da codificação FEC



Fonte: o autor

No contexto acima exposto, este trabalho tem como objetivo o desenvolvimento de *core* do codificador de canal LDPC em conformidade com o padrão DVB-S2. O desenvolvimento do *core*, em lógica programável, é balizado pelo compromisso entre o desempenho do LDPC e a ocupação de recursos da FPGA (*Field-Programmable Gate Array*).

Para atingir o objetivo proposto neste trabalho, é identificada a arquitetura mais adequada à implementação do *core* do codificador de canal LDPC em conformidade com o padrão DVB-S2, assim como são avaliadas diferentes representações numéricas das variáveis envolvidas no sistema em ponto fixo.

O desenvolvimento é realizado em linguagem de descrição de *hardware VHSIC Hardware Description Language* (VHDL) (IEEE STD 1076, 2000) e implementado em *Field-Programmable Gate Array* (FPGA) (XILINX ISE, 2016).

A dissertação está organizada como segue: o Capítulo 2 apresenta a fundamentação teórica do trabalho, abordando os principais conceitos de

Codificação de Canal, tais como o Teorema de Shannon e Códigos de Bloco Binários. O Capítulo 3 apresenta o código LDPC presente no padrão DVB-S2 e os respectivos algoritmos de codificação e decodificação. O Capítulo 4 apresenta a arquitetura do codificador e a arquitetura do decodificador do código LDPC DVB-S2 implementados neste trabalho. O Capítulo 5 apresenta os resultados de simulação em linguagem de programação C e em linguagem de descrição de *hardware* VHDL, bem como os resultados de síntese. Finalizando a dissertação, o Capítulo 6 apresenta as conclusões e sugestões para trabalhos futuros.

2 Codificação de Canal

Neste capítulo é apresentada uma revisão de codificadores de canal. As informações *neste* capítulo são baseadas na apostila de De Castro e De Castro (DE CASTRO & DE CASTRO, 2001).

Uma vez que a informação digital é enviada através de um canal de transmissão, ruídos e interferências inerentes a qualquer canal prático degradam o sinal, de forma que os dados transmitidos chegam ao receptor contendo erros.

O usuário do sistema de transmissão digital estabelece uma taxa de erro máxima aceitável, acima da qual os dados não são considerados utilizáveis. Usualmente, a taxa de erro máxima aceitável corresponde a uma mensagem errada a cada 1×10^{6} mensagens recebidas, ou seja, uma taxa de erro de 1×10^{-6}.

Em um sistema digital, o subsistema responsável por manter a taxa de erro dentro de um limite máximo aceitável pelo usuário é o Codificador de Canal. Em 1948, através do trabalho denominado Teorema Fundamental de Shannon (SHANNON, 1948) , foi demonstrada a possibilidade do uso de codificação para o controle com eficiência da taxa de erros de um sistemas de comunicação digital. Segundo Shannon, se a taxa (velocidade) de transmissão $R\ [bits/s]$ da informação a ser enviada pelo canal é menor do que uma dada quantidade $C\ [bits/s]$, denominada de Capacidade do Canal, a comunicação através do canal poderá ser estabelecida com uma probabilidade de erro tão baixa quanto se deseje, através da utilização de um código adequado para correção de erro.

Supondo que o único agente degradante do canal seja o ruído $\eta(t)$ com distribuição de probabilidade Gaussiana (canal Gaussiano), a Lei de Shannon-Hartley, decorrente do Teorema Fundamental de Shannon, estabelece que a capacidade C deste tipo de canal é dada por

$$C = B \log_2 \left(1 + \frac{P}{N}\right) [bits/s], \tag{2.1}$$

onde B é a largura de banda do canal em Hz, P é a potência do sinal transmitido e N é potência do ruído gaussiano adicionado ao sinal no canal. É possível interpretar também P como a potência do sinal recebido no receptor e N, a potência do ruído na entrada do receptor.

Em geral, como a densidade espectral de $\eta(t)$, dada por $|\Im\{\eta(t)\}|^2$, é uma constante $\eta_0/2$ dentro do intervalo de frequência $-B \leq f \leq B$, o ruído pode ser considerado branco (TAUB & SCHILLING, 1986), e a potência do ruído pode ser aproximada por $N = \eta_0 B$, sendo o $\Im\{\cdot\}$ o operador Transformada de Fourier (CARLSON, 1965).

É instrutivo verificar o que acontece com a Capacidade do Canal quando a largura de banda B do canal é aumentada ao infinito. Da Equação 2.1 com $N = \eta_0 B$ tem-se:

$$C = B \log_2 \left(1 + \frac{P}{\eta_0 B}\right) [bits/s], \tag{2.2}$$

que pode ser reescrita como

$$C = \frac{P}{\eta_0} \log_2 \left(1 + \frac{P}{\eta_0 B}\right)^{\frac{\eta_0 B}{P}} [bits/s]. \tag{2.3}$$

Quando $B \to \infty$ tem-se

$$C|_{B\to\infty} = \lim_{B\to\infty} \frac{P}{\eta_0} \log_2 \left(1 + \frac{P}{\eta_0 B}\right)^{\frac{\eta_0 B}{P}} [bits/s], \tag{2.4}$$

mas

$$\lim_{x\to 0}(1 + x)^{1/x} = e \tag{2.5}$$

e, fazendo $x = P/\eta_0 B$, da Equação 2.4 e da Equação 2.5 tem-se

$$C|_{B\to\infty} = \frac{P}{\eta_0} \log_2 e = 1.44 \frac{P}{\eta_0} [bits/s]. \tag{2.6}$$

A Equação 2.6 é conhecida como Limite de Shannon, e define a máxima taxa de transmissão para um canal cuja largura de banda seja suficientemente grande, tal que não apresente qualquer atenuação ao espectro do sinal que transporta a informação a ser transmitida.

O Teorema Fundamental de Shannon, entretanto, apenas demonstra que, se $R \leq C$, poderá existir um código corretor de erro tal que a informação possa ser transmitida através do canal com uma taxa de erro arbitrariamente baixa, mas não especifica como construir tal código corretor. Considera-se a maior utilidade prática do Teorema Fundamental de Shannon, demonstrar que para $R > C$ não é possível transmitir informação sem erro através do canal, mesmo que se utilize o mais poderoso código corretor de erro que possa ser concebido.

É importante destacar que, muitas vezes, o maior valor possível para a taxa de transmissão R é dado pela complexidade computacional do código corretor necessário para que o valor de R possa ser alcançado, e não pela Capacidade do Canal em si.

2.1 Codificadores de Canal

O Codificador de Canal tem como propósito introduzir, na informação a ser transmitida, de maneira controlada, uma determinada quantidade de informação redundante, de tal forma que, no receptor, esta informação redundante possa ser utilizada para detectar e corrigir erros decorrentes de ruído e interferência, os quais inevitavelmente afetam o sinal quando este é transmitido através do Canal de Transmissão. A redundância adicionada à informação a ser transmitida serve para aumentar a confiabilidade da informação recebida e, assim, melhorar a fidelidade do sinal recebido no receptor. (DE CASTRO & DE CASTRO, 2001)

Uma forma de introduzir informação redundante é tomar um conjunto de K *bits* de informação na entrada do Codificador de Canal do *encoder* e mapear cada mensagem de K *bits* em uma sequência de N *bits*, sendo $N > K$, sequência esta denominada de palavra-código, formando assim um *codebook* com 2^K palavras-códigos. Cada palavra-código é formada por K *bits* de informação acrescidos de $N - K$ *bits*, denominados de *bits* de paridade.

O mapeamento é definido de modo a maximizar a distância de Hamming (DE CASTRO & DE CASTRO, 2001) entre todas as palavras-códigos contidas no *codebook*, de modo a maximizar a dessemelhança entre elas, reduzindo a incerteza do *decoder* ao identificar qual palavra-código foi recebida no receptor.

O *decoder* no receptor do enlace usa como referência o mesmo mapeamento do *codebook* do *encoder* no transmissor do enlace. Uma vez recebida uma palavra-código, o *decoder* determina a distância de Hamming entre ela e todas as palavras-código do *codebook*. Aquela palavra-código do *codebook* que resultar na menor distância de Hamming entre ela e a palavra-código recebida é considerada como a mais provável de corresponder à palavra-código originalmente transmitida.

Os dois métodos de codificação de canal mais conhecidos são o *Backward Error Correction* (BEC), que efetua apenas a detecção do erro, ou seja, quando um erro é detectado, o transmissor é requisitado a retransmitir a mensagem. Esse método é simples e estabelece pequenos requisitos nas propriedades do código corretor. Por outro lado, o BEC exige comunicação do tipo *duplex* e causa indesejável retardo na transmissão. O segundo método é o *Forward Error Correction* (FEC), que requer que o decodificador seja, também, capaz de corrigir um determinado número de erros. Como os códigos FEC requerem apenas comunicação *simplex*, são especialmente atrativos em sistemas de comunicação sem fio. (TAUB & SCHILLING, 1986)

Existem duas grandes classes de códigos corretores de erro: os códigos de blocos e os códigos convolucionais. Existe também a possibilidade de combinar essas duas classes, criando os chamados códigos concatenados.

Para este trabalho, apenas os códigos de bloco e, em específico, os códigos de bloco binários, são de interesse, uma vez que os códigos LDPC, utilizados no padrão DVB-S2, são um subgrupo dos códigos de bloco binários .

2.2 Códigos de Bloco Binários

Um código de bloco binário $\boldsymbol{\theta}\{\cdot\}$ mapeia um conjunto $\boldsymbol{X} = \{\underline{x}_i\} = \{\underline{x}_0, \underline{x}_1, \cdots, \underline{x}_{M-1}\}$ de $M = 2^K$ mensagens binárias, cada uma delas com K *bits*, em um conjunto $\boldsymbol{C} = \{c_i\} = \{\underline{c}_0, \underline{c}_1, \cdots, \underline{c}_{M-1}\}$ de M palavras-código binárias, cada uma delas com N *bits*, onde $N > K$. Um código de bloco binário $\boldsymbol{\theta}\{\cdot\}$ cujas mensagens a serem codificadas apresentam K *bits* e são mapeadas em palavras-código de N *bits* é representado pelo operador $\boldsymbol{\theta}(N, K)\{\cdot\}$ ou simplesmente $\boldsymbol{\theta}(N, K)$**.**

A razão K/N, denominada Razão de Codificação, ou Taxa de Código, é um parâmetro importante no contexto de Códigos de Bloco, pois é um indicador da eficiência do Código, visto que quanto menor for a razão K/N , maior terá sido a quantidade de *bits* de paridade adicionados ao conteúdo de informação presente nas mensagens codificadas em palavras-código por $\boldsymbol{\theta}(N, K)$ (DE CASTRO & DE CASTRO, 2001).

Um código $\boldsymbol{\theta}(N, K)$ é sistemático quando cada palavra-código de N *bits* é formada pelos K *bits* da respectiva mensagem associada, acrescidos por justaposição de r *bits* adicionais destinados ao controle e correção de erros, denominados de *bits* de paridade. Em um código sistemático, portanto, cada mensagem contendo K *bits* de informação é expandida em uma palavra-código de $N = K + r$ *bits*, onde r é o número de *bits* representativos da informação redundante adicionada, visando o controle e a correção de erro.

O peso de uma palavra-código é definido como o número de dígitos "1" nela existentes. Por exemplo, sejam $\underline{c}_i = [0\ 1\ 0\ 1]$ e $c_j = [1\ 0\ 0\ 0]$, o peso da palavra-código $\underline{c}_i$ é 2 e o peso da palavra-código $\underline{c}_j$ é 1.

Supondo que $\underline{c}_i$ e $\underline{c}_j$ sejam duas palavras-código quaisquer do código $\boldsymbol{\theta}(N, K)$, uma possível medida da diferença entre as duas palavras-código é o número de *bits* em posições correspondentes que diferem entre si. Esta medida é denominada Distância de Hamming (DE CASTRO & DE CASTRO, 2001) e denotada por d_{ij}. Por exemplo, sejam $\underline{c}_i = [0\ 1\ 0\ 1]$ e $c_j = [1\ 0\ 0\ 0]$, então $d_{ij} = 3$.

A Distância de Hamming d_{ij} sempre satisfaz a condição $0 < d_{ij} \leq N$, $i \neq j$, para duas palavras-códigos $\underline{c_i}$ e $\underline{c_j}$, ambas de N *bits*. Por definição, em um código $\boldsymbol{\theta}(N, K)$, $\underline{c_i} \neq \underline{c_j}$, $\forall i$ e $\forall j$ com $i \neq j$.

O menor valor no conjunto $\{d_{ij}\},\ i, j = 0, 1, \cdots, M-1,\ i \neq j,\ M = 2^K$ é denominado Distância Mínima do Código e denotado por d_{min}.

A Distância de Hamming d_{ij} é uma medida do grau de separação entre duas palavras-códigos $\underline{c_i}$ e $\underline{c_j}$. Por consequência, o grau de correlação temporal entre dois sinais modulados $v_i(t)$ e $v_j(t)$ gerados no modulador de um transmissor digital em decorrência de $\underline{c_i}$ e $\underline{c_j}$ é implicitamente associado à d_{ij} (PROAKIS, 1995). d_{min} está associado, portanto, à capacidade do código $\boldsymbol{\theta}(N, K)$ em identificar palavras-código demoduladas no receptor quando estas são recebidas em erro como consequência do ruído e interferência no canal. Ou seja, quanto maior d_{min}, maior a capacidade do código $\boldsymbol{\theta}(N, K)$ em detectar e corrigir erros.

Seja $\boldsymbol{\theta}(N, K)$ um código corretor binário, cuja menor distância entre palavras seja d_{min}. Seja d o número máximo de erros que $\boldsymbol{\theta}(N, K)$ é capaz de detetar, e seja t o número máximo de erros que $\boldsymbol{\theta}(N, K)$ é capaz de corrigir. Então:

$$d = d_{min} - 1, \tag{2.7}$$

$$t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor, \text{ e} \tag{2.8}$$

$$d_{min} < N - K + 1, \tag{2.9}$$

sendo $\lfloor \cdot \rfloor$ o operador que resulta no inteiro mais próximo e menor que o argumento (ASH, 1967) (PROAKIS, 1995).

2.2.1 Matriz Geradora de um Código $\boldsymbol{\theta}(N, K)$

Seja a i-ésima mensagem de um código binário $\boldsymbol{\theta}(N, K)$ representada pelo vetor $\underline{x_i} = [x_{i0}, x_{i1}, \cdots, x_{i(k-1)}]$, e seja a i-ésima palavra-código de $\boldsymbol{\theta}(N, K)$ representada pelo vetor $\underline{c_i} = [c_{i0}, c_{i1}, \cdots, c_{i(k-1)}]$, onde $i = 0, 1, \cdots, M-1$, $M = 2^K$.

O processo de codificação da mensagem $\underline{x_i} = [x_{i0}, x_{i1}, \cdots, x_{i(k-1)}]$ na respectiva palavra-código $\underline{c_i} = [c_{i0}, c_{i1}, \cdots, c_{i(k-1)}]$ efetuado por um código binário $\boldsymbol{\theta}(N, K)$ pode ser representado em forma matricial por

$$\underline{c_i} = \underline{x_i}\boldsymbol{G}, \tag{2.10}$$

onde a matriz $G_{K\times N}$ é denominada de matriz geradora do código $\boldsymbol{\theta}(N, K)$ e é dada por

$$\boldsymbol{G} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0(N-1)} \\ g_{10} & g_{11} & \cdots & g_{1(N-1)} \\ \vdots & \vdots & & \vdots \\ g_{(K-1)0} & g_{(K-1)1} & \cdots & g_{(K-1)(N-1)} \end{bmatrix}. \tag{2.11}$$

A matriz $\boldsymbol{G}$ pode ser interpretada como um conjunto de K vetores linhas $\underline{g_j}$, $j = 0, 1, \cdots, K-1$, tal que

$$\boldsymbol{G} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0(N-1)} \\ g_{10} & g_{11} & \cdots & g_{1(N-1)} \\ \vdots & \vdots & & \vdots \\ g_{(K-1)0} & g_{(K-1)1} & \cdots & g_{(K-1)(N-1)} \end{bmatrix} = \begin{bmatrix} \leftarrow & \underline{g_0} & \rightarrow \\ \leftarrow & \underline{g_1} & \rightarrow \\ \vdots & \vdots & \vdots \\ \leftarrow & \underline{g_{(K-1)}} & \rightarrow \end{bmatrix}. \tag{2.12}$$

Assim, a partir da Equação 2.10 e da Equação 2.12, pode-se verificar que cada palavra-código $\underline{c_i} = [c_{i0}, c_{i1}, \cdots, c_{i(k-1)}]$ é uma combinação linear dos vetores $\underline{g_j}$ com coeficientes da combinação linear determinados pela mensagem associada $\underline{x_i} = [x_{i0}, x_{i1}, \cdots, x_{i(k-1)}]$, ou seja:

$$\underline{c_i} = x_{i0}\underline{g_0} + x_{i1}\underline{g_1} + \cdots + x_{i(K-1)}\underline{g_{(K-1)}}. \tag{2.13}$$

É possível demonstrar que o conjunto C de 2^K palavras-código de um código $\boldsymbol{\theta}(N, K)$ é um sub-espaço vetorial de dimensão K (CLARK JR & CAIN, 1988), (PETERSON & WELDON JR, 1990) e (LIN & COSTELLO, Error Control Coding, 1983). Logo, os K vetores-linha $\underline{g_j}$ que formam a matriz $\boldsymbol{G}$ devem ser

linearmente independentes para que possam, conforme estabelece a Equação 2.13, gerar o sub-espaço C em K dimensões. Ou seja, o conjunto de vetores $\underline{g_j}$ é uma base para o sub-espaço C.

Qualquer matriz geradora $\boldsymbol{G}$ de um código $\boldsymbol{\theta}(N,K)$ pode, através de operações elementares em suas linhas e permutações em suas colunas, ser reduzida à forma sistemática quando, então, o código gerado é sistemático. Uma matriz geradora $\boldsymbol{G}$ encontra-se na forma sistemática quando

$$\boldsymbol{G} = [I_{\boldsymbol{K}}|\boldsymbol{P}] = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & \cdots & 0 & p_{00} & p_{01} & \cdots & p_{0(N-K-1)} \\ 0 & 1 & 0 & \cdots & 0 & p_{10} & p_{11} & \cdots & p_{1(N-K-1)} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & p_{(K-1)0} & p_{(K-1)0} & \cdots & p_{(K-1)(N-K-1)} \end{array}\right], \tag{2.14}$$

onde $I_{\boldsymbol{K}}$ é a matriz identidade de ordem $K \times K$ e $\boldsymbol{P}$ é a matriz de ordem $K \times (N - K)$ responsável pela determinação dos $N - K$ *bits* de paridade que estarão presentes na palavra-código $\underline{c_i}$ de N *bits*, formada a partir dos K *bits* da mensagem $\underline{x_i}$.

2.2.2 Matriz Paridade de um Código $\boldsymbol{\theta}(N,K)$

Seja um código $\boldsymbol{\theta}(N,K)$ com matriz geradora $\boldsymbol{G}$ dada na forma sistemática, isto é,

$$\boldsymbol{G} = [I_{\boldsymbol{K}}|\boldsymbol{P}]. \tag{2.15}$$

Conforme Seção 2.2.1, a i-ésima palavra-código $\underline{c_i} = [c_{i0}, c_{i1}, \cdots, c_{i(k-1)}]$ relaciona-se com a respectiva mensagem $\underline{x_i} = [x_{i0}, x_{i1}, \cdots, x_{i(k-1)}]$ através de $\underline{c_i} = \underline{x_i}\boldsymbol{G}$.

Dado que $\boldsymbol{G}$ encontra-se na forma sistemática, a palavra-código $\underline{c_i}$ pode ser decomposta em

$$\underline{c_i} = [\underline{x_i} \quad \underline{a_i}], \tag{2.16}$$

onde $\underline{a_i} = x_i\boldsymbol{P}$ é o vetor-linha que contém os $N - K$ *bits* de paridade de $\underline{c_i}$.

Visto que $\underline{a_i} = x_i \boldsymbol{P}$ e, considerando que a soma em $\boldsymbol{GF}(2)$ é uma operação módulo 2, então pode-se escrever que

$$x_i \boldsymbol{P} + \underline{a_i} = 0. \tag{2.17}$$

A Equação 2.17 pode ser escrita na forma matricial, conforme

$$[\underline{x_i} \quad \underline{a_i}] \begin{bmatrix} \boldsymbol{P} \\ I_{\boldsymbol{N-K}} \end{bmatrix} = 0. \tag{2.18}$$

Definindo $\boldsymbol{H}^T = \begin{bmatrix} \boldsymbol{P} \\ I_{\boldsymbol{N-K}} \end{bmatrix}$ tem-se, da Equação 2.18, que

$$\underline{c_i} \boldsymbol{H}^T = 0, \tag{2.19}$$

sendo

$$\boldsymbol{H} = (\boldsymbol{H}^T)^T = \begin{bmatrix} \boldsymbol{P} \\ I_{\boldsymbol{N-K}} \end{bmatrix}^T = [\boldsymbol{P}^T \ | \ (I_{N-K})^T] = [\boldsymbol{P}^T \ | \ I_{N-K}]. \tag{2.20}$$

Portanto, da Equação 2.19, infere-se que cada palavra do código $\boldsymbol{\theta}(N, K)$ é ortogonal a cada linha da matriz $\boldsymbol{H}$. Se $\underline{u} \cdot \underline{v}^T = 0$, então os vetores $\underline{u}$ e $\underline{v}$ são ortogonais (CHEN, 1984). Em consequência, como as palavras do código $\boldsymbol{\theta}(N, K)$ são geradas por $\boldsymbol{G}$, então

$$\boldsymbol{G}\boldsymbol{H}^T = 0. \tag{2.21}$$

A partir da Equação 2.19 pode-se notar que a matriz $\boldsymbol{H}$ pode ser usada no receptor digital para detectar se a palavra-código foi recebida corretamente ou com erro, como consequência da degradação imposta pelo canal de transmissão. Sempre que a palavra-código $\underline{y_i}$ recebida no receptor digital resultar em $\underline{y_i}\boldsymbol{H}^T \neq 0$, pode-se concluir que $\underline{y_i}$ apresenta erros. Por este motivo, $\boldsymbol{H}_{(N-K)\times N}$ é denominada Matriz de Paridade do código.

2.2.3 Decodificação pela Mínima Distância

No receptor digital, os N *bits* provenientes do demodulador, correspondentes à i-ésima palavra-código $\underline{y_i}$ recebida, são entregues ao decodificador do código $\boldsymbol{\theta}(N, K)$. O decodificador compara $\underline{y_i}$ com as $M = 2^K$

possíveis palavras-código $\underline{c}_j$ de $\boldsymbol{\theta}(N, K)$, $j = 0, 1, \cdots, M-1$, e decide em favor daquela palavra-código que é mais próxima da palavra-código recebida, em termos da Distância de Hamming.

Um decodificador baseado no critério de distância mínima é denominado Decodificador de Máxima Verossimilhança, ou Decodificador ML (*Maximum-Likelihood*).

Embora a decodificação ML possa ser realizada, comparando $\underline{y}_i$ com as $M = 2^K$ possíveis palavras do código, existe uma maneira mais eficiente de implementar um decodificador ML, aproveitando as propriedades da matriz de paridade $\boldsymbol{H}_{(N-K)\times N}$ de um código $\boldsymbol{\theta}(N, K)$, denominada Decodificação por Arranjo Padrão (*Standard Array Decoding*).

Seja $\underline{c}_i$ a palavra-código transmitida pelo transmissor digital através do canal de transmissão, e seja $\underline{y}$ a palavra-código recebida, resultante na saída do demodulador do receptor digital. Devido à degradação do sinal no canal, em consequência de ruído e de interferência, a palavra-código $\underline{y}$ recebida pode conter erros, de modo que $\underline{y}$ pode ser expressa por

$$\underline{y} = \underline{c}_i + \underline{e}\ , \tag{2.22}$$

onde $\underline{e}$ é o vetor-linha de N *bits* que representa o padrão de erro resultante da degradação do sinal no canal, ou seja, $\underline{e}$ é o vetor que, ao ser somado à palavra código originalmente transmitida, corresponde ao tipo (padrão) de erro ocorrido, resultando em $\underline{y}$. O peso do padrão de erro é a Distância de Hamming calculada entre $\underline{y}$ e $\underline{c}_i$.

Multiplicando os dois lados da Equação 2.22 por $\boldsymbol{H}^T$ obtém-se

$$\underline{y}\boldsymbol{H}^T = \left(\underline{c}_i + \underline{e}\right)\boldsymbol{H}^T = \underline{c}_i\boldsymbol{H}^T + \underline{e}\boldsymbol{H}^T = \underline{e}\boldsymbol{H}^T \tag{2.23}$$

A partir da Equação 2.23 define-se o vetor $N-K$ dimensional $\underline{s}$, denominado síndrome do padrão de erro, ou simplesmente síndrome, conforme

$$\underline{s} = \underline{e}\boldsymbol{H}^T. \tag{2.24}$$

É importante enfatizar que o conjunto de síndromes $\{\underline{s}\}$ é determinado pelo conjunto de padrões de erro $\{\underline{e}\}$, mas não pelo conjunto $\boldsymbol{C}$ de palavras-código transmitidas, como pode-se inferir da Equação 2.23.

Visto que $\underline{e}$ é um vetor de N *bits*, existem 2^N possíveis padrões de erro no conjunto $\{\underline{e}\}$. Observe-se que, no entanto, $\underline{s}$ é um vetor de $N-K$ *bits*, de onde, portanto, existem 2^{N-K} possíveis síndromes no conjunto $\{\underline{s}\}$. Em consequência, a Equação 2.24 mapeia diferentes padrões de erro $\underline{e}$ na mesma síndrome $\underline{s}$.

A Decodificação por Arranjo Padrão (AP) resulta em uma tabela, denominada Tabela de Síndromes, a qual é implementada em ROM (*Read Only Memory)* no receptor digital. A Tabela de Síndromes é consultada pelo decodificador ML para identificação e correção de erro em cada palavra-código $\underline{y}$ recebida.

2.3 Códigos Low-Density Parity-Check

Os códigos *Low-Density Parity-Check* (LDPC) são uma subcategoria dos códigos de bloco lineares e foram, originalmente, introduzidos por Gallager (GALLAGER, 1963) nos anos 1960. À época, os códigos LDPC não foram muito utilizados, devido à falta de um algoritmo decodificador eficiente e também em função de limitações de *hardware* (*hardware* de baixa capacidade computacional). Nos anos 1990, os códigos LDPC foram redescobertos e demonstraram desempenho próximo do limite de Shannon.

Além do notável desempenho, o processo de codificação e decodificação adotado pelos códigos LDPC é muito menos complexo, quando comparado à outra classe de códigos cujo desempenho aproxima-se do Limite de Shannon, os códigos Turbo (BERROU & THITIMAJSHIMA, 1993).

Outro fator importante a ser observado é a presença de estruturas de código altamente paralelas nos códigos LDPC, as quais são extremamente adequadas para desenvolvimento em FPGA.

Em decorrência das características supracitadas, o LDPC é adotado em diversos padrões de sistemas de comunicações, sendo o DVB-S2 um deles.

Assim como nos demais códigos de bloco, os códigos LDPC utilizam uma matriz geradora $\boldsymbol{G}$ para a codificação das mensagens, e uma matriz paridade $\boldsymbol{H}$ para a decodificação do código.

De acordo com (LIN & COSTELLO, Error Control Coding, 1983), diferentemente das matrizes paridade $\boldsymbol{H}$ dos demais códigos de bloco, no caso do código LDPC a matriz paridade possui as seguintes propriedades:

- duas linhas ou colunas não possuem mais do que um elemento não-zero em comum, e
- os pesos de uma linha ou coluna são pequenos, em comparação ao tamanho do código.

Se os pesos das linhas e das coluna são constantes, então a matriz $\boldsymbol{H}$ descreve um código LDPC regular; caso contrário, a matriz $\boldsymbol{H}$ descreve um código LDPC irregular, sendo este o caso do padrão DVB-S2.

Todas as propriedades apresentadas nessa seção são aplicáveis aos códigos LDPC.

As técnicas que serão discutidas no próximo Capítulo, no entanto, diferem das apresentadas nessa seção, uma vez que a estrutura dos códigos LDPC no padrão DVB-S2 possibilita uma implementação de codificação e decodificação mais eficiente.

3 Códigos LDPC no Padrão DVB-S2

Os códigos LDPC utilizados no padrão DVB-S2 possuem dois tamanhos de palavra-código, no contexto de codificação LDPC, equivalentemente denominados de *frame* (ETSI TR 102 376 V 1.1.1, 2005). Os *frames* normais possuem tamanho de bloco $N = 64800$ *bits* e os *frames* pequenos possuem tamanho de bloco $N = 16200$ *bits*. Onze taxas de códigos são especificadas nos *frames* normais[1] e dez nos *frames* pequenos[2].

Apesar das matrizes de paridade $\boldsymbol{H}$, escolhidas por cada padrão, serem esparsas, suas respectivas matrizes geradoras não são, causando uma grande complexidade de codificação. O padrão DVB-S2, no entanto, adota uma estrutura especial da matriz $\boldsymbol{H}$, chamada de *Irregular Repeat-Accumulate* (IRA) (JIN, KAHNDEKAR, & MCELIECE, 2000), para reduzir os requerimentos de memória e a complexidade do codificador, uma vez que essa estrutura de matriz $\boldsymbol{H}$ permite ao codificador a utilização da mesma, e não de uma matriz geradora $\boldsymbol{G}$. A matriz $\boldsymbol{H}$ pode ser representada por duas matrizes, $\boldsymbol{A}$ e $\boldsymbol{B}$ concatenadas, conforme

$$\boldsymbol{H}_{(N-K)\times N} = \left[\boldsymbol{A}_{(N-K)\times N} | \boldsymbol{B}_{(N-K)\times(N-K)}\right], \tag{3.1}$$

onde K é o tamanho em *bits* da mensagem e $\boldsymbol{B}$ é uma matriz triangular inferior em forma de escadaria. A matriz $\boldsymbol{A}$ é uma matriz esparsa, onde a posição dos elementos não-zero é especificada nos anexos B e C do padrão (ETSI EN 302 307 V 1.2.1, 2009). Além disso, o padrão introduz uma periodicidade de $M = 360$ posições de *bits* na submatriz $\boldsymbol{A}$ para reduzir ainda mais os requisitos de memória.

[1] 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9 e 9/10
[2] 1/5, 1/3, 2/5, 4/9, 3/5, 2/3, 11/15, 7/9, 37/49 e 8/9

$$\boldsymbol{H} = \left[\begin{array}{cccc|cccccc} a_{00} & a_{01} & \cdots & a_{0,K-1} & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ a_{10} & a_{11} & \cdots & a_{1,K-1} & 1 & 1 & 0 & & & \vdots \\ \vdots & & & \vdots & 0 & 1 & 1 & \ddots & & \vdots \\ \vdots & & & \vdots & \vdots & \ddots & \ddots & \ddots & 0 & \vdots \\ a_{N-K-2,0} & a_{N-K-2,1} & \cdots & a_{N-K-2,K-1} & \vdots & & \ddots & 1 & 1 & 0 \\ a_{N-K-1,0} & a_{N-K-1,1} & \cdots & a_{N-K-1,K-1} & 0 & \cdots & \cdots & 0 & 1 & 1 \end{array}\right] \quad (3.2)$$

A condição de periodicidade divide a submatriz $\boldsymbol{A}$ em grupos de $M = 360$ colunas. Para cada grupo, os locais dos elementos não-zero localizados na primeira coluna são dados pelos anexos B e C do padrão. Seja $c_0, c_1, \ldots, c_{d_b-1}$ o conjunto de locais dos elementos não-zero da primeira coluna, ou a coluna mais à esquerda do grupo, onde d_b é o número de elementos não-zero nessa primeira coluna, também chamado de grau de nodo de bit. Para cada uma das $M - 1 = 359$ outras colunas, os locais dos elementos não-zero da i-ésima coluna do grupo são dados por

$$\begin{gathered} (c_0 + i \times p) \bmod (N-K), \\ (c_1 + i \times p) \bmod (N-K), \\ (c_2 + i \times p) \bmod (N-K), \\ \vdots \\ (c_{d_b-1} + i \times p) \bmod (N-K) \end{gathered} \quad (3.3)$$

onde $N - K$ é o número de *bits* de paridade e $p = \frac{N-K}{M}$ é uma constante dependente do código, conforme mostra a Tabela 3.1. Os valores da Tabela 3.1 são obtidos a partir das diretrizes do usuário do padrão DVB-S2 (ETSI TR 102 376 V 1.1.1, 2005).

Tabela 3.1 - Valores de p nos códigos LDPC do padrão DVB-S2

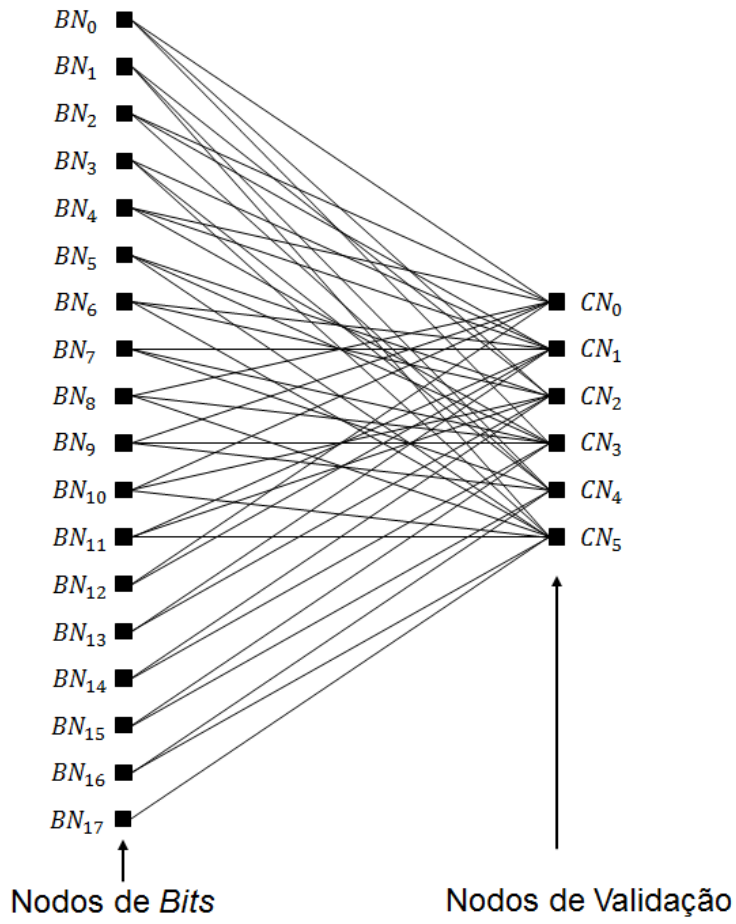
$N = 64800$		$N = 16200$	
Taxa do código	p	Taxa do código	p
1/4	135	1/5	36
1/3	120	1/3	30
2/5	108	2/5	27
1/2	90	4/9	25
3/5	72	3/5	18
2/3	60	2/3	15
3/4	45	11/15	12
4/5	36	7/9	10
5/6	30	37/49	8
8/9	20	8/9	5
9/10	18	-	

Fonte: ETSI (2007).

A matriz de paridade $\boldsymbol{H}$ pode ser representada com a ajuda dos gráficos de Tanner. Em 1981, Tanner (TANNER, 1981) desenvolveu um método para representar os códigos LDPC de forma gráfica, o que possibilitou pesquisas adicionais quanto ao método iterativo de decodificação. Um exemplo do gráfico de Tanner é apresentado na Figura 3.1, sendo a matriz de paridade $\boldsymbol{H}$ da Equação 3.4 sua respectiva matriz de origem.

$$\boldsymbol{H} = \left[\begin{array}{cccccccccccc|cccccc} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \tag{3.4}$$

Figura 3.1 - Exemplo de Gráfico de Tanner



Fonte: o autor.

O gráfico de Tanner é constituído por N nodos de *bit* (BN) e $N - K$ nodos de validação (CN). O índice j do nodo BN_j corresponde ao índice da respectiva coluna na matriz $\boldsymbol{H}$. O índice i do nodo CN_i corresponde ao índice da respectiva linha da matriz $\boldsymbol{H}$**.** Uma linha interliga o nodo BN_j ao nodo CN_i do gráfico de Tanner sempre que $h_{ij} \neq 0$. Por exemplo, considerando a matriz $\boldsymbol{H}$ da Equação 3.4, existe um elemento não-zero na linha 1, coluna 1, logo existe uma linha conectando o nodo de validação CN_0 ao nodo de bit BN_0, conforme ilustra a Figura 3.1.

3.1 *Encoder* LDPC no padrão DVB-S2

Os códigos LDPC são sistemáticos, possuindo o formato $\underline{c} = [\underline{x} \mid \underline{p}]$, com os *bits* de mensagem $\underline{x} = [x_0, x_1, \cdots, x_{(k-1)}]$ associados à submatriz $\boldsymbol{A}$ e os *bits* de paridade $\underline{p} = [p_0, p_1, \cdots, p_{(N-K-1)}]$ associados à submatriz $\boldsymbol{B}$. Em termos de representação por gráfico de Tanner, significa que os nodos de validação (CN) são conectados a dois tipos de nodos de *bits* (BN): os nodos de informação (IN), representando os K *bits* da mensagem e os nodos de paridade (PN), representando os $N - K$ *bits* de paridade.

A estrutura bidiagonal da submatriz $\boldsymbol{B}$ corresponde a uma conectividade em ziguezague entre os nodos de validação (CN) e os nodos de paridade (PN), permitindo a computação dos *bits* de paridade diretamente da submatriz $\boldsymbol{A}$, de um modo recursivo, de acordo com

$$
\begin{aligned}
p_0 &= a_{0,0}x_0 \oplus a_{0,1}x_1 \oplus \ldots \oplus a_{0,K-1}x_{K-1} \\
p_1 &= a_{1,0}x_0 \oplus a_{1,1}x_1 \oplus \ldots \oplus a_{1,K-1}x_{K-1} \oplus p_0 \\
p_2 &= a_{2,0}x_0 \oplus a_{2,1}x \oplus \ldots \oplus a_{2,K-1}x_{K-1} \oplus p_1 \\
&\vdots \\
p_{N-K-1} &= a_{N-K-1,0}x_0 \oplus a_{N-K-1,1}x_1 \oplus \ldots \oplus a_{1,K-1}x_{K-1} \oplus p_{N-K-2},
\end{aligned}
\tag{3.5}
$$

onde $\oplus$ é o operador de soma em módulo dois. Para códigos LDCP-IRA, uma vez que a submatriz $\boldsymbol{A}$ é esparsa, a complexidade computacional será $O(n)$ (JIN, KAHNDEKAR, & MCELIECE, 2000).

De acordo com (GOMES, GONÇALVES, SILVA, FALCAO, & MAIA, 2007), manipulando corretamente as Equações 3.5, e considerando a periodicidade $M = 360$ comum em todas as taxas de códigos do LDPC do padrão DVB-S2, obtém-se um algoritmo compartilhado por todas as opções de taxas de código, capaz de realizar a computação parcialmente paralela dos *bits* de paridade.

Seja $IN(l)$ os índices dos *bits* de informação que fazem parte do nodo de paridade l, ou, em termos da representação do gráfico de Tanner, os índices

de IN que estão conectados em CN_l. Seja CN_c os índices de CN que estão conectados em CN_c, com as letras l e c significando linhas e colunas da matriz $\boldsymbol{H}$, respectivamente. Dessa maneira, as Equações 3.5 podem ser reescritas conforme segue,

$$\begin{aligned} p_0 &= \bigoplus_{z \in IN(0)} x_z = S_0 \\ p_1 &= \bigoplus_{z \in IN(1)} x_z \oplus p_0 = S_1 \oplus S_0 \\ p_2 &= S_2 \oplus p_1 = S_2 \oplus S_1 \oplus S_0 \\ &\vdots \\ p_l &= S_l \oplus p_{l-1} = \bigoplus_{i=0}^{l} S_i \end{aligned} \tag{3.6}$$

para $l = 0, \cdots, N - K - 1$ e

$$S_l = \bigoplus_{z \in IN(l)} x_z \tag{3.7}$$

A Equação 3.7 pode ser computada recursivamente para cada CN conforme,

Para $l = 0 : (N - K - 1)$

Para cada $z \in IN(l)$ **fazer**

$$S_l = S_l \oplus x_z \tag{3.8}$$

Fim Para

Fim Para

com $S_l = 0$ sendo o valor inicial. Ao invés de processar os CNs um a um, o que requereria um conhecimento *a priori* de todos os *bits* de informação, é possível inverter a ordem de processamento, de horizontal para vertical. Então, para cada novo bit x de mensagem recebido pelo codificador, os valores S_l são atualizados de acordo com

Para $c = 0 : (K - 1)$

Para cada $l \in CN(c)$ **fazer**

$$S_l = S_l \oplus x_z \tag{3.9}$$

Fim Para

Fim Para

Considerando a periodicidade do código LDPC, cada iteração da Equação 3.9 pode ser realizada simultaneamente por $M = 360$ CNs. De acordo com a Equação 3.3, para cada grupo de M *bits* de mensagem $\underline{x} = [x_c, x_{c+1}, \cdots, x_{c+M-1}]$ recebidos, sendo $c \bmod M = 0$, é necessário saber apenas $CN(c)$ para encontrar $CN(c + j)$, sendo $j = 1, \cdots, M - 1$. De fato, se CN_l está conectado a IN_c, por exemplo, $l \in CN(c)$, então, $[(l + j \times p) \bmod (N - K)] \in CN(c + j)$.

Como consequência, o procedimento iterativo de codificação pode ser reescrito como

Para $c = 0 : M : (K - M)$

Para cada $l \in CN(c)$ **fazer**

$$\begin{aligned}
S_l &= S_l \oplus x_c \\
S_{(l+p) \bmod (N-K)} &= S_{(l+p) \bmod (N-K)} \oplus x_{c+1} \\
S_{(l+2\times p) \bmod (N-K)} &= S_{(l+2\times p) \bmod (N-K)} \oplus x_{c+2} \\
&\vdots \\
S_{(l+(M-1)\times p) \bmod (N-K)} &= S_{(l+(M-1)\times p) \bmod (N-K)} \oplus x_{c+M-1}
\end{aligned} \tag{3.10}$$

Fim Para

Fim Para

Se os valores de S_l forem armazenados na seguinte matriz binária,

$$\boldsymbol{S} = \begin{bmatrix} S_0 & S_p & S_{2p} & \cdots & S_{(M-1)p} \\ S_1 & S_{p+1} & S_{2p+1} & \cdots & S_{(M-1)p+1} \\ S_2 & S_{p+2} & S_{2p+2} & \cdots & S_{(M-1)p+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{p-1} & S_{2p-1} & S_{3p-1} & \cdots & S_{N-K-1} \end{bmatrix}, \tag{3.11}$$

é possível observar que todos os valores S_l atualizados nas Equações 3.10 estão na mesma linha de $\boldsymbol{S}$. O único problema é que os valores não estão na ordem desejada para realizar a computação recursiva especificada nas Equações 3.10. Entretanto, de acordo com a estrutura de $\boldsymbol{S}$, é necessário apenas a realização de um deslocamento cíclico da linha de $\boldsymbol{S}$ correspondente ou, de uma forma alternativa, uma rotação do vetor de tamanho M da mensagem.

Considerando a linha requerida, $\boldsymbol{S}(l \bmod p, :)$ e o vetor da mensagem $\underline{x} = [x_c, x_{c+1}, \cdots, x_{c+M-1}]$, a iteração das Equações 3.10 pode ser assim reescrita como

$$\boldsymbol{S}(l \bmod p, :) = \boldsymbol{S}(l \bmod p, :) \oplus rot_{l\ div\ p}(\underline{x}), \tag{3.12}$$

onde $rot_a(\cdot)$ é um deslocamento circular para a direita de a *bits*.

Uma vez que todos os valores S_l são conhecidos, é possível computar os bits de paridade p_l de acordo com as operações recursivas das Equações 3.6. Porém, esse método implica em um atraso muito grande do codificador. É possível diminuir esse atraso utilizando abordagem vetorial. Computando a soma em módulo dois de todas as colunas de $\boldsymbol{S}$, se obtém o vetor

$$\underline{s} = \begin{bmatrix} \bigoplus_{i=0}^{p-1} S_i, & \bigoplus_{i=p}^{2p-1} S_i, \cdots, & \bigoplus_{i=358p}^{359p-1} S_i & \bigoplus_{i=359p}^{360p-1} S_i \end{bmatrix}_{(1\times M)}, \tag{3.13}$$

o qual pode ser transformado utilizando as seguintes operações em GF(2):

$$\boldsymbol{L} \times \underline{s} = \begin{bmatrix} \bigoplus_{i=0}^{p-1} S_i, & \bigoplus_{i=p}^{2p-1} S_i, \cdots, & \bigoplus_{i=358p}^{359p-1} S_i & \bigoplus_{i=359p}^{360p-1} S_i \end{bmatrix}^T = \underline{s}' \tag{3.14}$$

onde $\boldsymbol{L}$ é uma matriz triangular inferior unitária de dimensão $(M \times M)$ conforme

$$\boldsymbol{L} = \begin{bmatrix} 1 & & & & \\ & 1 & & 0 & \\ & & \ddots & & \\ & 1 & & 1 & \\ & & & & 1 \end{bmatrix}. \tag{3.15}$$

O vetor $\underline{s}'$ é deslocado logicamente para a esquerda em uma posição para a obtenção de:

$$\begin{bmatrix} & p-1 & 2p-1 & & 359p-1 \\ 0, & \bigoplus_{i=0} S_i, & \bigoplus_{i=p} S_i, \cdots, & & \bigoplus_{i=358p} S_i \end{bmatrix} = [0, p_{p-1}, p_{2p-1}, \cdots, p_{359p-1}]. \tag{3.16}$$

Tendo calculado os valores da matriz $\boldsymbol{S}$ e também tendo somado em módulo dois os valores da coluna da matriz $\boldsymbol{S}$, é possível calcular $M = 360$ *bits* de paridade por vez, conforme segue

$$\begin{aligned} [p_0, p_p, p_{2p}, \cdots, p_{359p-1}] &= [0, p_{p-1}, p_{2p-1}, \cdots, p_{359p-1}] \ \oplus \boldsymbol{S}(0,:) \\ [p_1, p_{p+1}, p_{2p+1}, \cdots, p_{359p+1}] &= [p_0, p_p, p_{2p}, \cdots, p_{359p-1}] \quad \oplus \boldsymbol{S}(1,:) \\ [p_2, p_{p-1}, p_{2p-1}, \cdots, p_{359p+2}] &= [p_1, p_{p+1}, p_{2p+1}, \cdots, p_{359p+1}] \oplus \boldsymbol{S}(2,:) \\ &\vdots \\ [p_{p-1}, p_{2p-1}, p_{3p-1}, \cdots, p_{N-K-1}] &= [0, p_{p-1}, p_{2p-1}, \cdots, p_{359p-1}] \ \oplus \boldsymbol{S}(p-1,:). \end{aligned} \tag{3.17}$$

A palavra-código codificada é a concatenação dos *bits* da mensagem com os *bits* de paridade, de tal forma que a palavra-código $\underline{c}$ de *N bits* resultante possui a seguinte forma

$$\underline{c} = [x_0, x_1, \dots, x_{K-1}, p_0, p_1, \dots, p_{N-K-1}]. \tag{3.18}$$

3.2 *Decoder* LDPC no padrão DVB-S2

A decodificação dos códigos LDPC no padrão DVB-S2 é uma decodificação do tipo *soft-decision* (LIN & COSTELLO, Error Control Coding, 1983). O algoritmo utilizado para a decodificação dos códigos LDPC é um algoritmo de passagem de mensagem, onde as mensagens (números reais) são passadas entre os dois conjuntos de nodos CN e BN. Essas mensagens são atualizadas nos nodos através de operações matemáticas. Para simplificar os

cálculos, as entradas do sistema são valores de *log-likelihood ratio* (LLR). Seja $\underline{c} = [c_0, c_1, \cdots, c_{N-1}]$ a palavra-código transmitida e $\underline{y}$ a sequência *soft-decision* recebida. O valor LLR, denotado λ_l, para cada bit da sequência é dado por

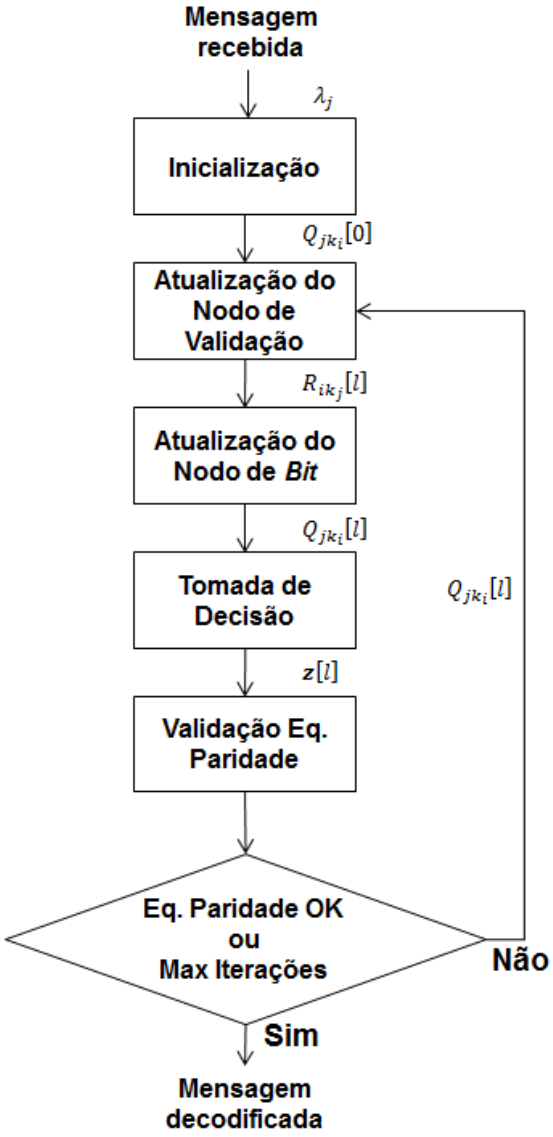
$$\lambda_l = \log\left(\frac{P(c_l = 0|y)}{P(c_l = 1|y)}\right) \tag{3.19}$$

Um valor negativo de λ_l indica que a probabilidade do evento "*bit* recebido $= 1$" é maior que a probabilidade do evento "*bit* recebido $= 0$". Um valor positivo para λ_l indica o contrário. Quanto maior for o valor absoluto de λ_l, maior a probabilidade do evento.

A saída do decodificador LDPC é uma estimativa da palavra binária correspondente à mensagem originalmente transmitida e, adicionalmente, um valor lógico indicando se a decodificação foi terminada com sucesso ou se um erro ainda existe na mensagem decodificada. Tal decisão é determinada pelas equações de paridade da Matriz $\boldsymbol{H}$. O processo de decodificação pode ser visualizado também através de gráficos de Tanner.

Um dos algoritmos mais conhecidos para decodificar os códigos LDPC é o Algoritmo Soma-Produto (SPA) (LIN & COSTELLO, Error Control Coding, 2nd Ed, 2004). O SPA é um algoritmo de troca de passagem de mensagem, onde as mensagens, que são valores reais, são passadas entre os nodos de validação e os nodos de *bits*. Os algoritmos descritos na sequência são similares aos apresentados por Masera et al. (MASERA, QUAGLIO, & VACCA, 2005), exceto pela função ψ^{-1} no passo de atualização dos nodos de validação, a qual é substituída pela função ψ no passo de atualização dos nodos de *bits*. Os passos da decodificação seguem de acordo com Eroz et al. (EROZ, SUN, & LEE, 2004), ilustrados pelo diagrama de fluxo apresentado na Figura 3.2 e a seguir descritos.

Figura 3.2 - Diagrama de fluxo do SPA

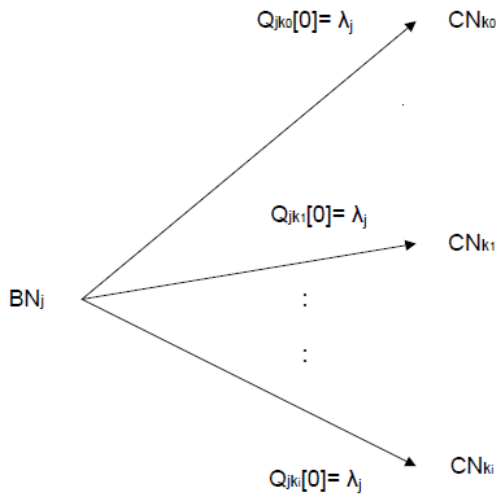


Fonte: o autor.

1. Inicialização:

Seja N o tamanho do bloco e λ_j o LLR do sinal recebido, com $j = 0, 1, \dots, N-1$. Seja $Q_{jk_i}[l]$ a mensagem a ser enviada do nodo de bit BN_j para o CN_{k_i} durante a l-ésima iteração, onde k_i é o índice do nodo de validação que tem uma linha conectando-o ao nodo de bit BN_j, $i = 0, 1, \dots, d_b - 1$ e d_b é o grau de nodo de bit BN_j. A Figura 3.3 ilustra o gráfico de Tanner no passo da inicialização do algoritmo SPA.

Figura 3.3 - Passo de Inicialização do SPA



Fonte: Loi, Kung (2010).

Utilizando a matriz $\boldsymbol{H}$, esse passo é equivalente a atribuir λ_j a todos os elementos não-zero na coluna j da matriz $\boldsymbol{H}$

$$\boldsymbol{H}[0]_b = \begin{bmatrix} h_{00} \cdot \lambda_0 & h_{01} \cdot \lambda_1 & \cdots & h_{0,N-1} \cdot \lambda_{N-1} \\ h_{10} \cdot \lambda_0 & h_{11} \cdot \lambda_1 & \cdots & h_{1,N-1} \cdot \lambda_{N-1} \\ \vdots & & & \vdots \\ h_{N-K-1,0} \cdot \lambda_0 & h_{N-K-1,1} \cdot \lambda_1 & \cdots & h_{N-K-1,N-1} \cdot \lambda_{N-1} \end{bmatrix}, \qquad (3.20)$$

onde $h_{ij} = 0$ ou 1 é o elemento na i-ésima linha e j-ésima coluna em $\boldsymbol{H}$ e $\boldsymbol{H}[0]_b$ é a matriz $\boldsymbol{H}$ inicializada.

2. Atualização do Nodo de Validação:

Seja $R_{ik_j}[l]$ a mensagem a ser enviada de CN_i para BN_{k_j} durante a l-ésima iteração, onde k_j é o índice do nodo de bit que tem uma linha conectando-o ao CN_i, $j = 0, 1, \ldots, d_c - 1$ e d_c é o grau de nodo de checagem de CN_i. Seja $B[i]$ o conjunto dos índices de BN de todas as mensagens que chegam no CN_i dos BNs conectados a eles, ou seja, o conjunto de todos os k_j índices de CN_i. O conjunto $R_{ik_j}[l]$ a ser enviado de CN_i para BN_{k_j} durante a l-ésima iteração é dado por

$$R_{ik_j}[l] = \left[\sum_{m\in B[i]} \psi(Q_{mi}[l]) - \psi(Q_{k_ji}[l])\right] \cdot \left[\prod_{m\in B[i]} \operatorname{sgn}(Q_{mi}[l]) \times \operatorname{sgn}(Q_{k_ji}[l])\right], \tag{3.21}$$

onde

$$\psi(x) = -\ln\left(\tanh\left|\frac{x}{2}\right|\right) = \ln\left(\frac{(1+e^{-|x|})}{(1-e^{-|x|})}\right), \tag{3.22}$$

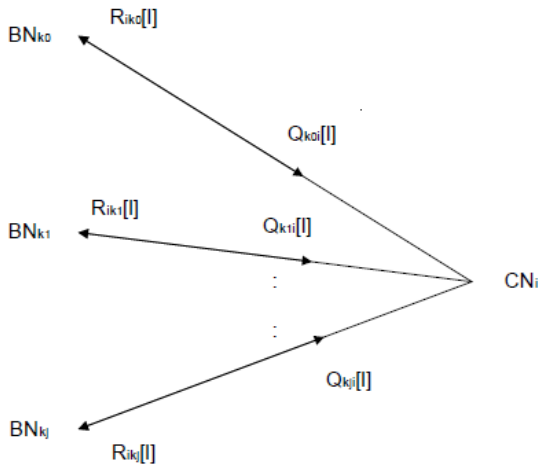
e

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{se } x < 0 \\ 0 & \text{se } x = 0. \\ 1 & \text{se } x > 0 \end{cases} \tag{3.23}$$

$Q_{mi}[l]$ é o conjunto dos nodos de bit BN_{k_j} que chegam ao nodo de validação CN_i e $Q_{k_ji}[l]$ é o nodo de bit BN_{k_j}.

Em outras palavras, a atualização dos nodos de validação é definida em duas etapas: o cálculo da magnitude e a definição do sinal. No cálculo da magnitude, para cada CN, é realizado o somatório dos valores de $\psi(Q_{mi}[l])$ e subtraído o valor de $\psi(Q_{k_ji}[l])$ correspondente. Similarmente, a parte do sinal é computada pelo produtório de $\operatorname{sgn}(Q_{mi}[l])$, multiplicado por $\operatorname{sgn}(Q_{k_ji}[l])$. A Figura 3.4 ilustra esse passo graficamente.

Figura 3.4 - Passo da Atualização do Nodo de Validação do SPA



Fonte: Loi, Kung (2010).

Utilizando a matriz $\boldsymbol{H}$ como referência, esse passo utiliza todos os elementos não-zero de cada linha de $\boldsymbol{H}[l]_b$ para o cálculo de $R_{ik_j}[l]$, conforme (3.21), para a i-ésima linha e k_j-ésima coluna da matriz $\boldsymbol{H}[l]_c$, onde $\boldsymbol{H}[l]_c$ é a matriz $\boldsymbol{H}$ resultante após a l-ésima iteração da atualização do nodo de checagem.

3. Passo de Atualização do Nodo de Bit:

Seja $C[j]$ o conjunto de índices de CN de todas as mensagens que chegam ao BN_j dos CNs conectados a ele, ou seja, o conjunto de todos os índices k_i de BN_j. A atualização do nodo de checagem $Q_{jk_i}[l]$ é dada por

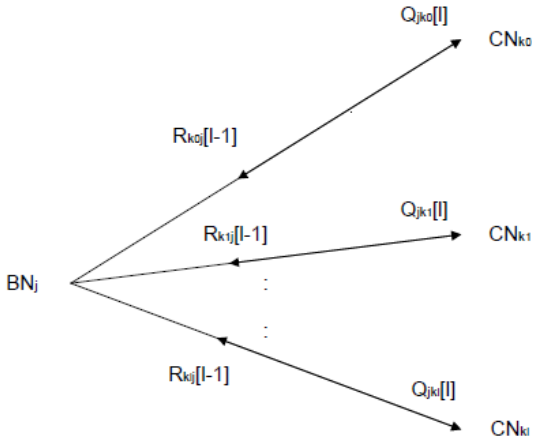
$$Q_{jk_i}[l] = \lambda_j + \sum_{m \in C[j]} \operatorname{sgn}\big(R_{mj}[l-1]\big) \cdot \psi\big(R_{mj}[l-1]\big) - \operatorname{sgn}(R_{k_ij}[l-1]) \cdot \psi(R_{k_ij}[l-1]), \tag{3.24}$$

onde $R_{mj}[l-1]$ é o conjunto de nodos de validação CN_{k_ij} que chegam ao nodo de bit BN_j e $R_{k_ij}[l-1]$ é o nodo de validação CN_{k_j}.

Essa equação é similar à equação de atualização do nodo de checagem, exceto que os cálculos da magnitude e do sinal estão dentro do somatório, e que este resultado é adicionado ao valor LLR λ_j. A Figura 3.5 mostra graficamente este passo.

Utilizando a matriz $\boldsymbol{H}$ como referência, este passo utiliza todos os elementos não-zero de cada coluna de $\boldsymbol{H}[l]_c$ para o cálculo de $Q_{jk_i}[l]$, conforme a Equação 3.24, para a k_i-ésima linha e j-ésima coluna da matriz $\boldsymbol{H}[l]_b$, onde $\boldsymbol{H}[l]_b$ é a matriz $\boldsymbol{H}$ resultante após a l-ésima iteração do passo de atualização do nodo de bit.

Figura 3.5 - Passo da Atualização do Nodo de Bit *do SPA*



Fonte: Loi, Kung (2010)

4. Tomada de Decisão:

Após a atualização do nodo de bit, a sequência da palavra-código candidata $\underline{S}[l] = [S_0, S_1, \cdots, S_j, \cdots, S_{N-1}]$, no formato de *soft-decision* é computado da seguinte forma

$$S_j = \lambda_j + \sum_{m \in C[j]} \operatorname{sgn}\big(R_{mj}[l-1]\big) \cdot \psi\big(R_{mj}[l-1]\big). \qquad (3.25)$$

A Equação 3.25 é equivalente à primeira parte da Equação 3.24. Subsequentemente, a sequência $\boldsymbol{S}$ é decodificada em uma sequência de *hard-decision*, $\underline{z}[l] = [z_0, z_1, \cdots, z_j, \cdots, z_{N-1}]$, conforme segue

$$z_j = \begin{cases} 0 & se \quad S_j \geq 0 \\ 1 & se \quad S_j < 0 \end{cases}. \qquad (3.26)$$

A sequência $\boldsymbol{z}$ resultante é uma palavra-código candidata, formada por mensagem e paridade, e é utilizada para verificar se as equações de paridade são satisfeitas ou não. Se todas as equações de paridades forem satisfeitas, então a parte contendo a mensagem é a saída do decodificador. Mais especificamente, o decodificador tem como saída $\underline{z} = [z_0, z_1, \cdots, z_{K-1}]$. Caso as equações de paridade não tiverem sido satisfeitas, os passos 2, 3 e 4 são repetidos até todas as equações serem satisfeitas, ou até que um número pré-determinado de iterações seja realizado, caso em que um erro na decodificação é declarado.

4 *Core* Para Códigos LDPC no Padrão DVB-S2

Neste capítulo são apresentados os detalhes da arquitetura do codificador e do decodificador LDPC DVB-S2. Na Seção 4.1 são descritas as características da FPGA utilizada, na Seção 4.2 é apresentada a arquitetura do codificador LDPC DVB-S2 implementado, enquanto que na Seção 4.3 é apresentada a arquitetura do decodificador LDPC DVB-S2 implementado.

4.1 Arquitetura da FPGA

O projeto do Encoder e o projeto do Decoder LDPC DVB-S2 a serem apresentados nas próximas seções (Seção 4.2 e Seção 4.3, respectivamente), são implementados na FPGA Xilinx Kintex-7 XC7K160T.

Nesta seção será apresentada uma visão geral da arquitetura da FPGA Xilinx Kintex-7 XC7K160T, necessária ao entendimento dos resultados de síntese apresentados no próximo capítulo, Seção 5.3.

XC7K160T é um dispositivo da família Kintex-7, que, juntamente com a família Artix-7 e a família Virtex-7, formam a série 7 de FPGAs da Xilinx. A Tabela 4.1 apresenta os recursos mais relevantes da FPGA Kintex 7, no contexto deste trabalho.

Tabela 4.1 - Recursos da FPGA Kintex 7

Células lógicas	Blocos Lógicos Configuráveis (CLBs)		*Slices* de DSP	Blocos de Blocos de RAM		
	Slices	Max RAM Distribuída (Kb)		18 Kb	36 Kb	Max (Kb)
162,240	25,350	2,188	600	650	325	11,700

Fonte: o autor.

Cada *slice* de uma FPGA da série 7 da Xilinx contém quatro *Look-Up Tables* (LUT) e oito *flip-flops*, mas apenas alguns s*lices* podem usar suas LUTs como Memória de Acesso Randômico (RAM) distribuída ou *Super Logic Regions* (SRL).

Cada *slice* de Processamento Digital de Sinal (DSP) contém um *pre-adder*, um multiplicador 25 × 18, um somador e um acumulador.

Os blocos de RAM são fundamentalmente de tamanho 36 Kb. Cada bloco pode ser usado também como dois blocos de 18 Kb independentes.

Mais informações sobre a FPGA utilizada neste desenvolvimento podem ser obtidas em (7-SERIES FPGA OVERVIEW, 2016) e (KINTEX-7 FPGAs DATA SHEET, 2016).

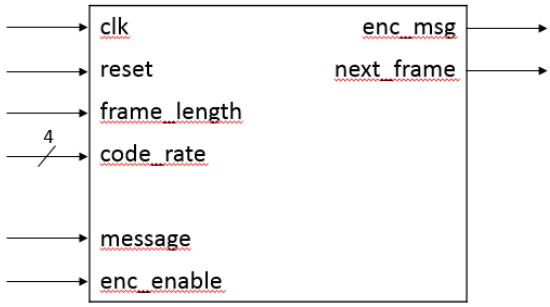
4.2 Arquitetura do *Encoder* LDPC DVB-S2

O *Encoder* LDPC DVB-S2 é responsável por gerar os *bits* de paridade de uma determinada mensagem $\underline{x}$, a partir das equações de paridade, conforme a Equação 3.5 e também a Equação 4.1, formando, de forma sistemática, a palavra-código $\underline{c}$.

$$
\begin{aligned}
p_0 &= a_{0,0}x_0 \oplus a_{0,1}x_1 \oplus \ldots \oplus a_{0,K-1}x_{K-1} \\
p_1 &= a_{1,0}x_0 \oplus a_{1,1}x_1 \oplus \ldots \oplus a_{1,K-1}x_{K-1} \oplus p_0 \\
p_2 &= a_{2,0}x_0 \oplus a_{2,1}x \oplus \ldots \oplus a_{2,K-1}x_{K-1} \oplus p_1 \qquad (4.1) \\
&\vdots \\
p_{N-K-1} &= a_{N-K-1,0}x_0 \oplus a_{N-K-1,1}x_1 \oplus \ldots \oplus a_{1,K-1}x_{K-1} \oplus p_{N-K-2}
\end{aligned}
$$

A Figura 4.1 apresenta as entradas e as saídas do codificador. A Tabela 4.2 descreve o significado de cada entrada e de cada saída.

Figura 4.1 - Entradas e saídas do codificador LDPC



Fonte: o autor.

Tabela 4.2 - Descrição das entradas e saídas do codificador

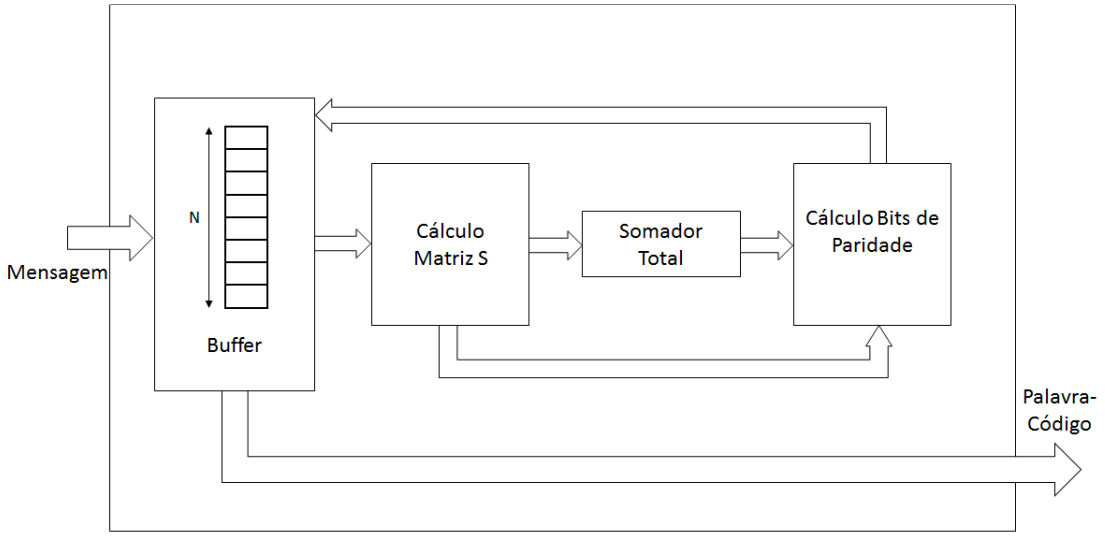
Nº de *bits*	Nome	Descrição
Sinais de entrada do codificador		
1	clk	Clock
1	reset	Reset
1	frame_length	Seleciona entre os *frames* normais ou pequenos (0 – *frame* normal; 1 – *frame* pequeno)
4	code_rate	Seleciona a taxa do código ($0000_b \rightarrow 1010_b$ para os *frames* normais; $0000_b \rightarrow 1001_b$ para os *frames* pequenos; em ordem crescente da taxa do código
1	message	Valores da mensagem de tamanho 1 bit no formato serial
1	enc_enable	Marca o início de um *frame* de entrada
Sinais de saída do codificador		
1	enc_msg	Saída serial da mensagem codificada
1	next_frame	codificador pronto para um próximo *frame*

Fonte: o autor.

Uma vez que cada uma das vinte e uma diferentes combinações de tamanhos de bloco e taxas de código tem conjuntos de equações de paridades diferentes, é necessário um codificador que gere cada conjunto separadamente. Se cada um dos conjuntos de equações de paridade fosse implementado separadamente em uma FPGA, seriam necessários muitos recursos de *hardware*. A arquitetura do codificador apresentada nessa seção possui a habilidade de gerar os *bits* de paridade de cada um dos códigos LDPC no padrão DVB-S2 de forma otimizada, sem a necessidade de implementação das equações de paridade em *hardware*. As equações são implementadas de acordo com a taxa de codificação desejada, baseadas na matriz de paridade $\boldsymbol{H}$, armazenada em memória ROM.

A arquitetura do codificador é baseada no codificador apresentado por (GOMES, GONÇALVES, SILVA, FALCAO, & MAIA, 2007). A Figura 4.2 mostra o diagrama de blocos do codificador LDPC. O codificador contém quatro elementos principais: *Buffer,* Bloco de Cálculo da Matriz S, Somador Total, Bloco de Cálculo de *Bits* de Paridade.

Figura 4.2 - Diagrama de Blocos do Codificador



Fonte: o autor.

O bloco Cálculo Matriz $\boldsymbol{S}$ é responsável pela computação dos valores S_l, conforme a Equação 3.12. O bloco Somador Total realiza a soma em módulo dois de todas as colunas da Matriz $\boldsymbol{S}$, local onde os valores S_l estão armazenados, de acordo com a Equação 3.13. Por fim, o bloco Cálculo *Bits* de Paridade realiza o cálculo dos *bits* de paridade, conforme a Equação 3.16.

O processo de codificação de uma mensagem de K bits, determinado pela taxa de código, começa com a inserção da mensagem a ser codificada no Buffer de forma serial. Após a inserção dos bits da mensagem no Buffer, o bloco Cálculo Matriz $\boldsymbol{S}$ acessa os valores contidos no Buffer em blocos de 360 bits, desloca-os por um determinado número de posições, de acordo com valores que estão armazenados em uma memória ROM interna para o cálculo dos valores S_l na Equação 3.12.

Não é necessário esperar o fim da computação de todos os valores S_l para a obtenção da soma em módulo dois de todas as colunas de $\boldsymbol{S}$, o que implicaria em um atraso muito grande. O bloco Somador Total computa a soma em módulo dois do vetor de mensagem recém deslocado com o valor anteriormente acumulado, de acordo com a Equação 3.13, ao mesmo tempo em que os valores S_l são atualizados.

Após o acúmulo da soma de todas as colunas de $\boldsymbol{S}$ no bloco Somador Total, o vetor resultante do acúmulo da soma em módulo dois é enviado ao bloco

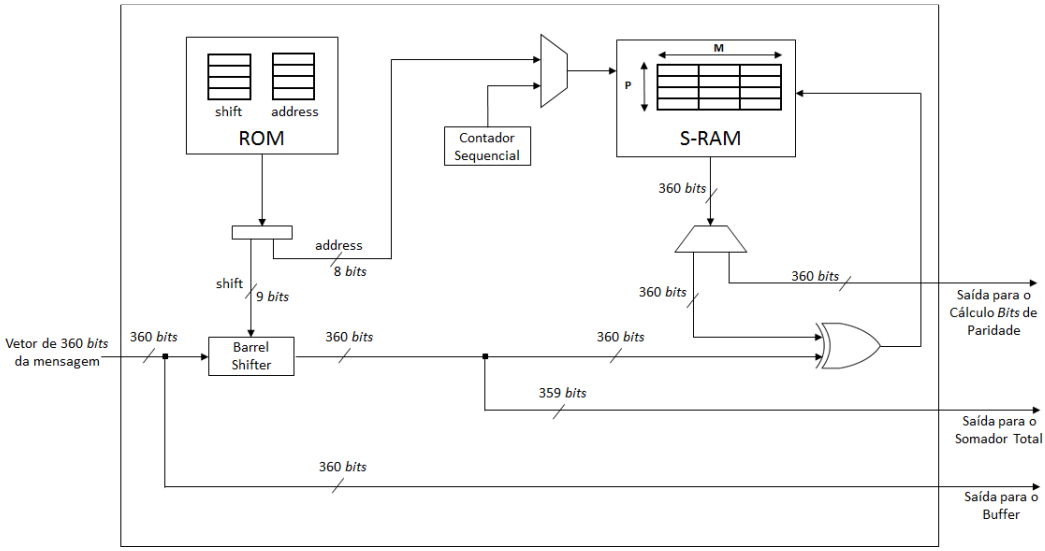
Cálculo Bits de Paridade e, juntamente com os vetores recebidos do Bloco Cálculo da Matriz $\boldsymbol{S}$, são somados em módulo 2 bit a bit, gerando 360 bits de paridade por ciclo de clock. Valores estes que são armazenados no Buffer juntamente com a Mensagem original a ser transmitida, formando a Palavra-Código para ser enviada ao canal de transmissão.

4.2.1 Arquitetura do Bloco “Cálculo Matriz S”

O Bloco Cálculo Matriz S realiza o cálculo da Equação 3.10 e a Equação 3.12, responsáveis pela atualização dos valores da matriz $\boldsymbol{S}$.

O diagrama de blocos do Bloco Cálculo Matriz $\boldsymbol{S}$ é mostrado na Figura 4.3, e é composto basicamente por uma memória ROM, uma memória RAM *dual-port*, um *Barrel-Shifter* (BS) e uma porta lógica XOR.

Figura 4.3 - Diagrama de Blocos do Bloco Cálculo Matriz ***S***



Fonte: o autor

A memória RAM S-RAM representa a matriz $\boldsymbol{S}$ e é projetada para o pior caso das taxas de código, caso em que a taxa de código $R = 1/4$ para o tamanho de frame normal, onde $p = 135$. Assim sendo, a S-RAM deve ter a dimensão de 135×360 células.

Com base na taxa de código e no tamanho do frame, são selecionados os valores apropriados na memória ROM correspondente à respectiva taxa de código. Os valores de shift indicam a quantidade de posições que o vetor $\underline{x}$ deve ser deslocado ciclicamente para a direita, enquanto que os valores de address indicam o índice da linha da matriz $\boldsymbol{S}$ a ser atualizada. Os valores da ROM correspondentes de shift e address $(l \div p)$ são lidos sequencialmente para realizar o Cálculo da Matriz $\boldsymbol{S}$ de acordo com a Equação 3.10 e a Equação 3.12. A Seção 4.3.3.2 detalha como o valor de shift é obtido.

O algoritmo para a atualização dos valores da memória S-RAM é apresentado no quadro abaixo (Algoritmo 4.1), e é baseado na Equação 3.10 e na Equação 3.12.

Algoritmo 4.1 Algoritmo de Cálculo dos valores da Matriz $\boldsymbol{S}$

Iniciar todas as posições de $\boldsymbol{S}$ em 0

Pegar o primeiro $\underline{x}$ do *Buffer* de Mensagem

Para cada uma das linhas no apêndice A

  Para $l =$ cada elemento em cada linha **fazer**

    $\boldsymbol{S}(l \bmod p, :) = \boldsymbol{S}(l \bmod p, :) \oplus rot_{l \div p}(\underline{x})$

  Fim para

  $\underline{x} =$ próximo $\underline{x}$ do *Buffer* de Mensagem

Fim para

No Algoritmo 4.1, $\boldsymbol{S}(x, :)$ é o vetor de 360 bits correspondente à linha x da S-RAM, $rot_{l \div p}(\underline{x})$ é o vetor $\underline{x}$ deslocado para a direita ciclicamente em $l \div p$ posições, l é um valor obtido do Apêndice A para uma determinada taxa de código e $\oplus$ é o operador de soma de módulo dois bit a bit.

O algoritmo começa com todas as células da S-RAM contendo valor zero. Assim que os vetores de 360 *bits* são lidos do *Buffer* e inseridos no Bloco Cálculo Matriz $\boldsymbol{S}$, a S-RAM é atualizada de acordo com a Equação 3.12.

O *Barrel Shifter* realiza a operação $rot_{l \div p}(\underline{x})$, rotacionado ciclicamente o vetor $\underline{x}$ de 360 *bits* em $l \operatorname{div} p$ posições, onde y é o valor armazenado no *shift* da ROM naquele ciclo de *clock*. A porta lógica XOR realiza a soma em módulo dois *bit* a *bit* do vetor rotacionado pelo *Barrel Shifter* com o valor contido na linha da S-RAM, cujo índice é obtido pelo valor *address* armazenado também na ROM no mesmo ciclo de *clock*. O resultado obtido pela soma é armazenado na mesma linha da S-RAM, atualizando o valor da mesma. Além de ser uma das entradas da porta lógica XOR, os primeiros 359 *bits* do vetor rotacionado pelo *Barrel Shifter* são enviados para o bloco Somador Total.

De acordo com o Algoritmo 4.1, um vetor $\underline{x}$ atualiza o mesmo número de linhas da S-RAM que a quantidade de elementos de uma linha do Apêndice A, que é o grau de nodos de bit d_b de cada grupo de nodos de bit. Por exemplo, na Tabela A.1, a primeira linha contém doze elementos, portanto, um vetor $\underline{x}$ de 360 *bits* de mensagem vai atualizar doze linhas da S-RAM. Quando essas doze linhas foram atualizadas, um novo vetor $\underline{x}$ de 360 *bits* de mensagem vai atualizar outras doze linhas da S-RAM porque a segunda linha da Tabela A.1 também possui doze elementos. Quando todos os valores no Apêndice A tiverem sido utilizados, significa que todos os vetores de 360 *bits* da mensagem foram inseridos no bloco Cálculo Matriz $\boldsymbol{S}$ e a S-RAM foi completamente atualizada.

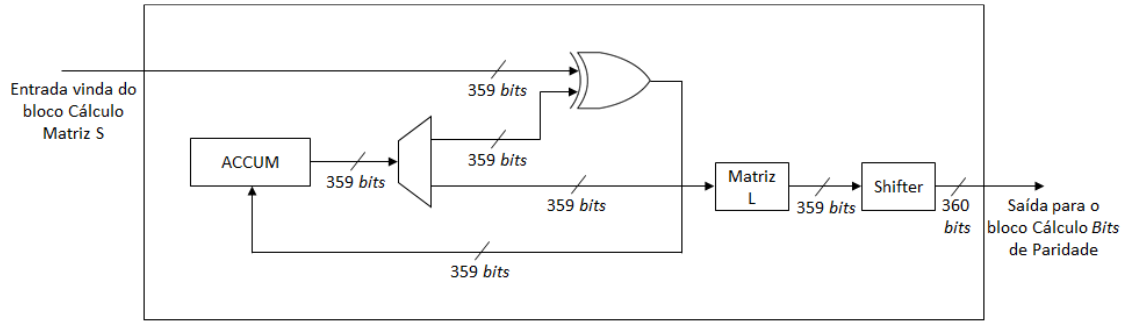
Uma vez que a S-RAM está completamente atualizada, as p linhas da S-RAM são lidas sequencialmente, uma a cada ciclo de *clock* através do Contador Sequencial, e os vetores correspondentes são enviados ao bloco Cálculo *Bits* de Paridade.

4.2.2 Arquitetura do Bloco “Somador Total”

O Bloco Somador Total realiza o cálculo da Equação 3.13, da Equação 3.14 e da Equação 3.15.

O diagrama de blocos do Bloco Somador Total é mostrado na Figura 4.4, e é composto basicamente por uma porta lógica XOR, um registrador acumulador (ACCUM), o bloco Matriz L e um *Shifter*.

Figura 4.4 - Diagrama de Blocos do bloco Somador Total



Fonte: o autor.

A porta lógica XOR realiza a Equação 3.13, que é a soma em módulo dois bit a bit das colunas da matriz $\boldsymbol{S}$, ou a soma das colunas da S-RAM do bloco Cálculo Matriz S. Uma das entradas da porta lógica XOR é o vetor de 359 bits vindo do Bloco Cálculo Matriz S. A outra entrada da porta lógica XOR é o valor acumulado no registrador ACCUM. O registrador ACCUM armazena um vetor de 359 bits e tem como valor inicial um vetor de bits zero. A cada ciclo de clock, um vetor vindo do bloco Cálculo Matriz S é somado em módulo dois com o vetor armazenado temporariamente no registrador ACCUM. O vetor resultante dessa soma é armazenado novamente no registrador ACCUM para ser somado com o próximo vetor proveniente do bloco Cálculo Matriz S.

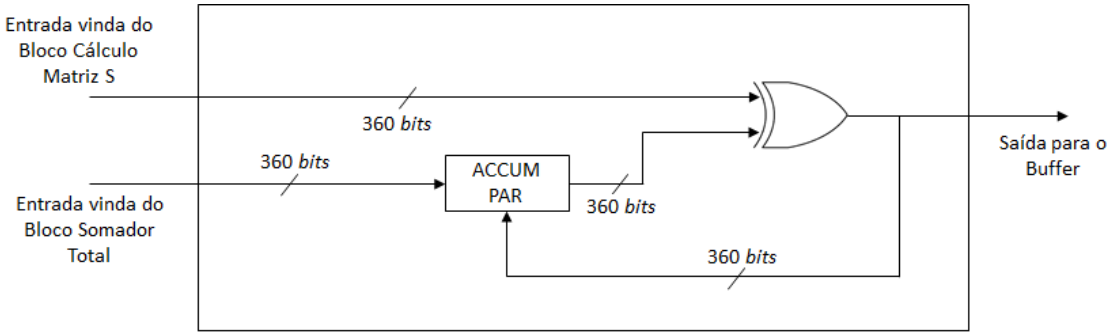
Assim que todos os vetores vindos do bloco Cálculo Matriz S foram acumulados no registrador ACCUM, o vetor resultante é enviado ao bloco Matriz L que realizará a Equação 3.14, que é a multiplicação do vetor resultante por uma matriz triangular inferior unitária de dimensão $(M \times M)$. Como o vetor possui 359 bits, a matriz L deve ter dimensões 359 bits × 359 bits. O vetor resultante da multiplicação será enviado ao bloco Shifter, que o deslocará logicamente para a direita em uma posição, conforme a Equação 3.15. É em função desta operação de deslocamento que esse bloco utiliza vetores de 359 bits, e não de 360 bits, uma vez que o último bit do vetor de 360 bits é descartado. O vetor resultante do bloco Shifter é enviado ao bloco Cálculo Bits de Paridade.

4.2.3 Arquitetura do Bloco “Cálculo *Bits* de Paridade”

O bloco Cálculo *Bits* de Paridade realiza o cálculo da Equação 3.16.

O diagrama do bloco Cálculo *Bits* de Paridade é mostrado na Figura 4.5 e é composto basicamente por uma porta lógica XOR e um registrador acumulador (ACCUM PAR).

Figura 4.5 - Diagrama de Blocos do Bloco Calculador de Paridades



Fonte: o autor.

Após a matriz $\boldsymbol{S}$ ter sido completamente atualizada, através do bloco Cálculo Matriz S e de suas colunas terem sido somadas em módulo dois bit a bit na Equação 3.13 através do bloco Somador Total, é possível calcular os bits de paridade da mensagem.

O vetor proveniente do bloco Somador Total é armazenado temporariamente no registrador ACCUM PAR e é uma das entradas da porta lógica XOR. A outra entrada da porta lógica XOR é o vetor armazenado na linha 0 da S-RAM. O vetor resultante da soma de módulo dois são os primeiros 360 bits de paridade da mensagem e são armazenados no Buffer para, na sequência, serem transmitidos. O vetor resultante também é enviado para o registrador ACCUM PAR. O novo valor armazenado no registrador é, de novo, uma das entradas da porta lógica XOR. A outra entrada da porta lógica XOR é o vetor armazenado na linha 1 da S-RAM e o vetor resultante da soma em módulo dois bit a bit é o segundo conjunto de 360 bits de paridade da mensagem, e também o valor armazenado no registrador ACCUM PAR para ser somado em módulo dois com o vetor armazenado na linha 2 da S-RAM. A iteração continua até que as p linhas da S-RAM sejam inseridas no bloco Cálculo Bits de Paridade e, por consequência, todos os $N - K$ bits de paridade terem sido calculados.

4.2.4 Arquitetura do *Buffer*

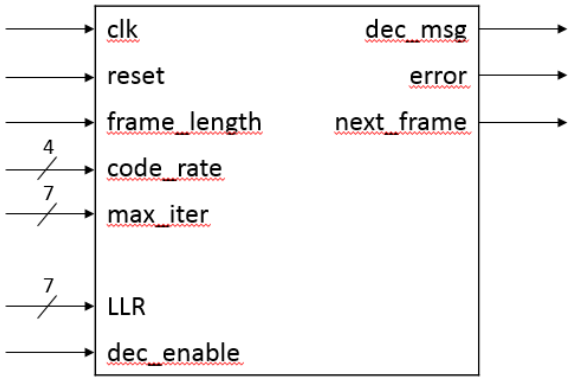
O *Buffer* é utilizado para armazenar os valores binários da mensagem a ser transmitida que chegam ao codificador, e também os *bits* de paridade que são gerados pelo codificador. O DVB-S2 prevê vinte e uma taxas de código diferentes, com vinte e um tamanhos K de mensagem possíveis, mas apenas dois tamanhos diferentes de palavra-código, $N = 64800$ *bits* e $N = 16200$ *bits*, de onde se conclui que é necessário prever um *Buffer* que armazene o maior tamanho de palavra-código N, ou seja, $N = 64800$ *bits*.

4.3 Arquitetura do *Decoder* LDPC DVB-S2

O *Decoder* LDPC DVB-S2 é responsável por receber os valores de *log-likelihood ratio* (LLR) vindos do canal de transmissão, conforme apresentado na Equação 3.19, e estimar a palavra binária correspondente à mensagem originalmente transmitida.

A Figura 4.6 apresenta as entradas e as saídas do decodificador. A Tabela 4.3 descreve o significado de cada entrada e de cada saída, em detalhes.

Figura 4.6 - Entradas e saídas do decodificador LDPC



Fonte: o autor.

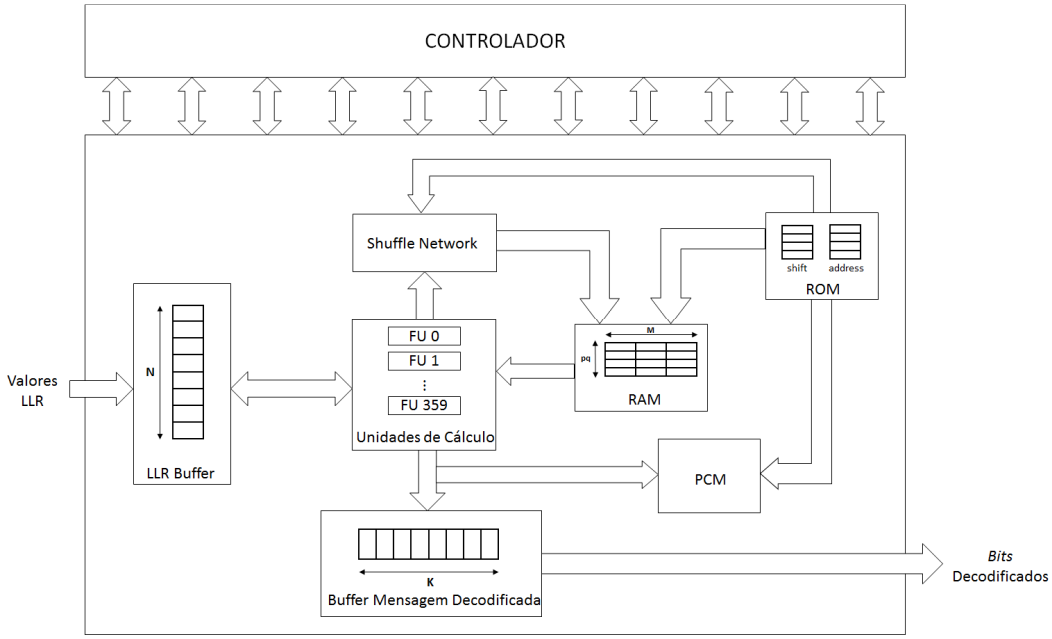
Tabela 4.3 - Descrição das entradas e saídas do decodificador

Nº de *bits*	Nome	Descrição
Sinais de entrada do decodificador		
1	clk	Clock
1	reset	Reset
1	frame_length	Seleciona entre os *frames* normais ou pequenos (0 – *frame* normal; 1 – *frame* pequeno)
4	code_rate	Seleciona a taxa do código ($0000_b \rightarrow 1010_b$ para os *frames* normais; $0000_b \rightarrow 1001_b$ para os *frames* pequenos; em ordem crescente da taxa do código
7	max_iter	Determina o número máximo de iterações realizadas pelo decodificador
7	LLR	Valores LLR de tamanho 7 *bits* no formato serial
1	dec_enable	Marca o início de um *frame* de entrada
Sinais de saída do decodificador		
1	dec_msg	Saída serial da mensagem decodificada
1	error	Indica a ocorrência ou não de erro na decodificação
1	next_frame	Decodificador pronto para um próximo *frame*

Fonte: o autor.

A arquitetura do decodificador é baseada no esquema de mapeamento de memória, de acordo com o apresentado por (EROZ, SUN, & LEE, 2006). A Figura 4.7 mostra o diagrama de blocos do decodificador LDPC. O decodificador contém oito elementos principais: *Buffer* LLR, Unidades Funcionais (FUs), *Shuffle Network*, ROM, RAM (Random Access Memory), Módulo de Verificação de Paridades (PCM), *Buffer* de Mensagem Decodificada e Controlador.

Figura 4.7 - Diagrama de blocos do decodificador



Fonte: o autor.

Recapitulando os passos do algoritmo SPA, conforme mostrados no Capítulo 3, durante o processo de inicialização, os valores LLR são inseridos no *Buffer* LLR de forma serial. De acordo com (ZHANG, WANG, & PARHI, 2001), apenas 6 *bits* são necessários para representar os valores LLR com perdas não muito significativas na performance. Neste trabalho foram utilizados 7 *bits* para representar os valores LLR, pois com 7 *bits* obteve-se uma melhora considerável na performance do decodificador. Para cada 360 valores de LLR coletados, os valores são comprimidos pelas FUs em um vetor de 5 *bits*, para a correta utilização do bloco ψ, conforme detalhado na Seção 4.3.4.1, e são encaminhados para o bloco *Shuffle Network*, bloco responsável por posicionar os valores vindos da FU nas corretas posições da memória RAM. Esses passos serão detalhados na sequência do capítulo.

No passo de Atualização dos Nodos de Validação, os valores são lidos da RAM e processados nas FUs de acordo com a Equação 3.21. Os valores resultantes são enviados para o bloco *Shuffle Network,* onde são colocados na posição correta da memória RAM para a realização do passo de Atualização dos Nodos de Bit.

No passo de Atualização dos Nodos de Bit, os valores são novamente lidos da RAM e processados nas FUs, mas dessa vez utilizando a Equação 3.24.

Para a Atualização dos Nodos de Bit são necessários os valores LLR, desta forma o *Buffer* LLR também é lido. Ao término dos cálculos, os valores resultantes são enviados ao bloco *Shuffle Network,* onde são colocados na posição correta da memória RAM para a realização do passo de Atualização dos Nodos de Validação, caso seja necessária uma outra iteração no processo de decodificação.

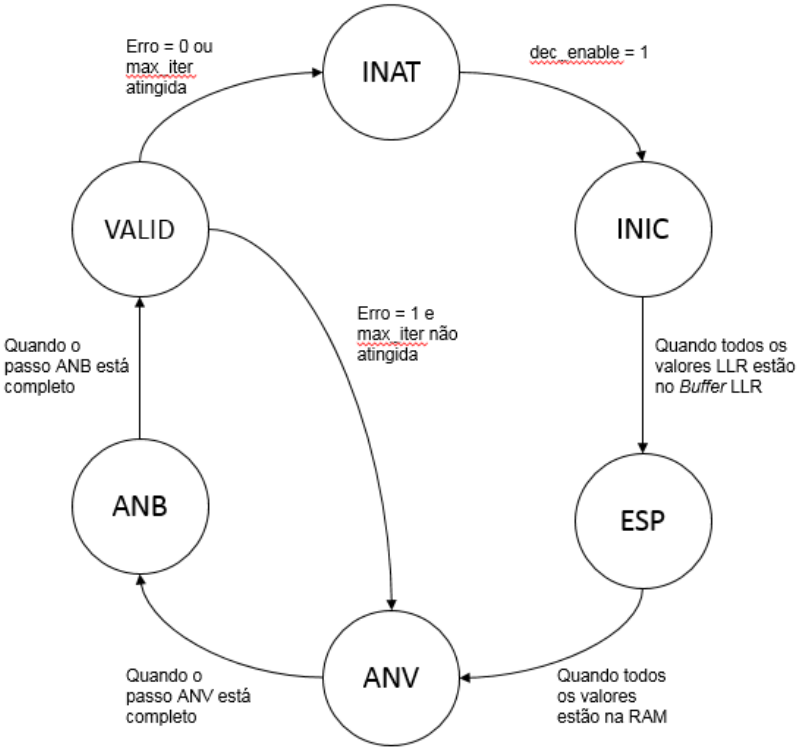
Durante o passo de Atualização dos Nodos de Bit*,* o decodificador também realiza o processo de Tomada de Decisão, já que as FUs, para computar a Equação 3.24, necessariamente computam a Equação 3.25, conforme mostrado no Capítulo 3. Além disso, conforme a Equação 3.26, os elementos da sequência de *hard-decision* da palavra-código candidata $\underline{z}[l]$ são equivalentes ao sinal dos elementos da sequência $\underline{S}[l]$, portanto, apenas os *bits* de sinal de $\underline{S}[l]$ são enviados das FUs para o PCM, bloco responsável pela verificação das equações de paridade do código. As FUs podem gerar porções de 360 *bits* de sinal da sequência completa $\underline{z}[l]$ por vez, e esses *bits* são enviados ao PCM assim que são gerados. As porções equivalentes à parte da mensagem da palavra-código são simultaneamente armazenadas no *Buffer* de Mensagem de Decodificação assim que são geradas.

O PCM verifica as equações de paridade e sua saída de erro indica se as equações de paridade foram satisfeitas ou não. O sinal indicando se houve erro ou não na decodificação é enviado ao Controlador para indicar se a decodificação continua em pelo menos mais uma iteração. Caso a saída de erro indique que todas as equações de paridade foram satisfeitas, a parte da mensagem da palavra-código candidata que estava armazenada no *Buffer* de Mensagem Decodificada corresponde à mensagem decodificada e é, então, enviada à saída do decodificador. A saída é feita de forma serial. Simultaneamente, um novo conjunto de valores LLR devem ser inseridos no decodificador. Caso a saída do PCM indique que nem todas as equações de paridade foram satisfeitas, o decodificador retorna ao passo de Atualização dos Nodos de Validação e itera até todas as equações serem satisfeitas ou até atingir um número máximo pré-estabelecido de iterações.

4.3.1 Arquitetura do Controlador

O Controlador é uma máquina de estados finita (FSM) que controla o fluxo de dados do decodificador, coordenando a operação de todos os sete outros elementos do decodificador, como mostra a Figura 4.7. As ligações entre o controlador e todos os demais blocos do sistema não são mostradas no diagrama de blocos do decodificador para efeito de clareza na interpretação da figura. O diagrama de transição de estados do Controlador é mostrado na Figura 4.8.

Figura 4.8 - Diagrama de estados da FSM do Controlador



Fonte: o autor.

Os seis estados do diagrama de estados do controlador são INAT, INIC, ESP, ANV, ANB e VALID.

O fluxo do diagrama de estados do decodificador inicia no estado INAT, onde o decodificador está inativo, esperando que sejam inseridos *frames* para a decodificação. Quando a entrada dec_enable está ativa, o controlador entra no estado INIC, onde os valores LLR são inseridos no decodificador e armazenados no *Buffer* LLR. O decodificador permanece nesse estado até que os 64800

valores de LLR, para os *frames* normais, ou os 16200 valores de LLR, para os *frames* pequenos, sejam inseridos no decodificador, quando o decodificador muda seu estado para ESP. O estado ESP é um estado de transição, onde o decodificador já recebeu todos os valores de LLR, mas ainda não está pronto para realizar nenhum tipo de cálculo, pois os valores de LLR armazenados na RAM. Uma vez que todos os valores de LLR estão nas posições corretas na RAM, o controlador passa para o estado ANV, onde são realizados os cálculos do passo de Atualização dos Nodos de Validação. Após os cálculos terem sido realizados, o controlador passa para o estado ANB, onde o passo de Atualização dos Nodos de Bit é realizado. Uma vez terminados os cálculos desse passo, o controlador passa para o estado VALID, onde o PCM verifica as equações de paridades. Caso as equações tenham sido satisfeitas, ou seja, $erro = 0$, então o controlador entra para o estado INAT e espera o próximo *frame* de valores LLR enquanto a mensagem decodificada é enviada à saída do decodificador. Caso as equações não tenham sido satisfeitas, ou seja, $erro = 1$, o controlador retorna ao estado ANV para repetir os estado ANV, ANB e VALID. Se o número máximo de iterações é atingido durante o estado VALID, o controlador passa para o estado INAT e a mensagem decodificada é enviada à saída do decodificador juntamente com sinalização de mensagem decodificada com erro, através da *flag* $error = 1$.

4.3.2 Arquitetura da RAM e da ROM

Para uma alta taxa de transferência, a implementação ideal do decodificador deveria ter uma FU para cada nodo de bit e uma FU para cada nodo de validação, onde cada FU executaria os cálculos da Equação 3.21 e da Equação 3.24 independentemente, implementando uma arquitetura completamente paralela. O passo de Atualização dos Nodos de Validação e o passo de Atualização dos Nodos de Bit, no entanto, nunca são executados simultaneamente e, por isso, as FUs são utilizadas para executar tanto os cálculos da Equação 3.21 quanto da Equação 3.24. Nesse caso, são necessárias uma FU para cada nodo de bit, uma vez que $N > N - K$. No padrão DVB-S2, contudo, $N = 64800$ para os frames normais e $N = 16200$ para os frames

pequenos. Para este caso, seriam necessárias 64800 FUs, o que é impraticável em uma implementação em hardware devido às limitações de recursos em uma FPGA.

Para endereçar este problema, Eroz et al. (EROZ, SUN, & LEE, 2006) propõe aproveitar o fator periodicidade $M = 360$ da matriz de paridade $\boldsymbol{H}$, conforme mostrado no Capítulo 3, e organizar a RAM de uma forma apropriada, resultando em um sistema que pode executar a decodificação de uma forma bastante eficiente utilizando apenas 360 FUs.

4.3.2.1 Esquema de Mapeamento da Memória

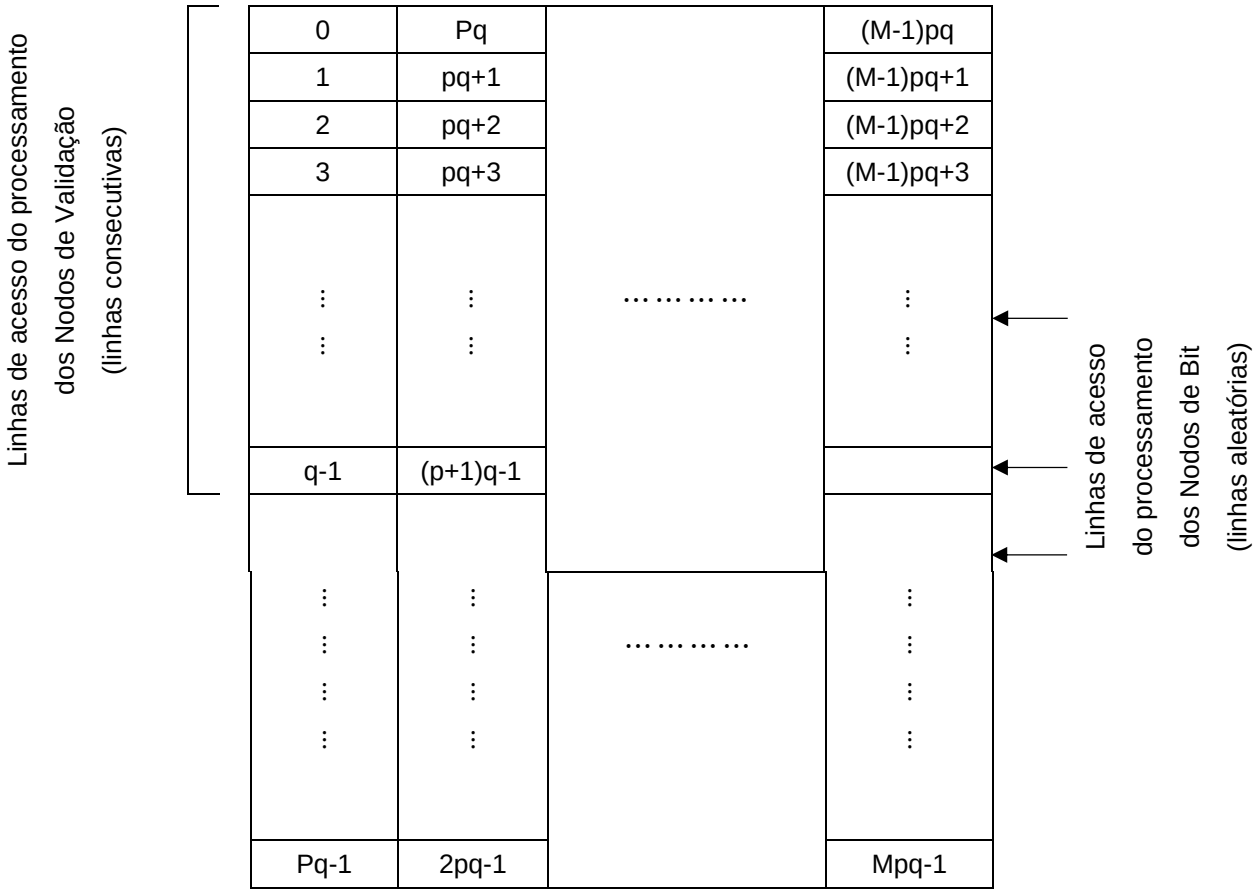
O esquema de mapeamento de memória é baseado no esquema apresentado por (EROZ, SUN, & LEE, 2006). Cada linha no gráfico de Tanner, ou cada elemento não-zero da matriz $\boldsymbol{H}$, é mapeado para uma posição na RAM, que age como uma caixa de mensagens para os valores R e Q da Equação 3.21 e da Equação 3.24, respectivamente, serem armazenados. A RAM é virtualmente dividida entre top RAM e bottom RAM. A top RAM corresponde aos elementos não-zero da submatriz $\boldsymbol{A}$ e a bottom RAM corresponde aos elementos não-zero da submatriz $\boldsymbol{B}$. Durante o passo de Atualização dos Nodos de Validação e o passo de Atualização dos Nodos de Bits, as FUs leem os valores da RAM, os processam e os reescrevem na RAM. Os locais dos quais as FUs leem da RAM dependem da Matriz $\boldsymbol{H}$.

Eroz et al. (EROZ, SUN, & LEE, 2006) sugerem que, caso a RAM seja organizada conforme mostram as Figuras 4.9 e 4.10, e cada elemento não-zero da matriz $\boldsymbol{H}$ seja mapeado corretamente em cada célula da RAM, o acesso, durante o passo de Atualização dos Nodos de Validação, é feito de forma sequencial, de q linhas da *top* RAM seguidas de duas linhas da *bottom* RAM. Na Figura 4.9 e na Figura 4.10, p e q são valores específicos de cada taxa de código. p é mostrado na Tabela 3.1 e na Tabela 4.4 e representa o número de grupos de $M = 360$ nodos de validação e q é mostrado na Tabela 4.4 e corresponde ao peso de cada linha da submatriz $\boldsymbol{A}$. Organizando a RAM de acordo com a Figura 4.9 e a Figura 4.10, apenas $M = 360$ FUs são necessárias na implementação e cada FU acessa e processa apenas os valores armazenados em uma coluna da

top RAM e uma coluna da *bottom* RAM. Nas Figuras 4.9 e 4.10, p e q são valores específicos de cada taxa de código. O valor p é mostrado nas Tabelas 3.1 e 4.4, e representa o número de grupos de $M = 360$ nodos de validação. O valor q é mostrado na Tabela 4.4 e corresponde ao peso de cada linha da submatriz $\boldsymbol{A}$. Organizando a RAM de acordo com as Figuras 4.9 e 4.10, apenas $M = 360$ FUs são necessárias para a implementação, e cada FU acessa e processa apenas os valores armazenados em uma coluna da *top* RAM e uma coluna da *bottom* RAM.

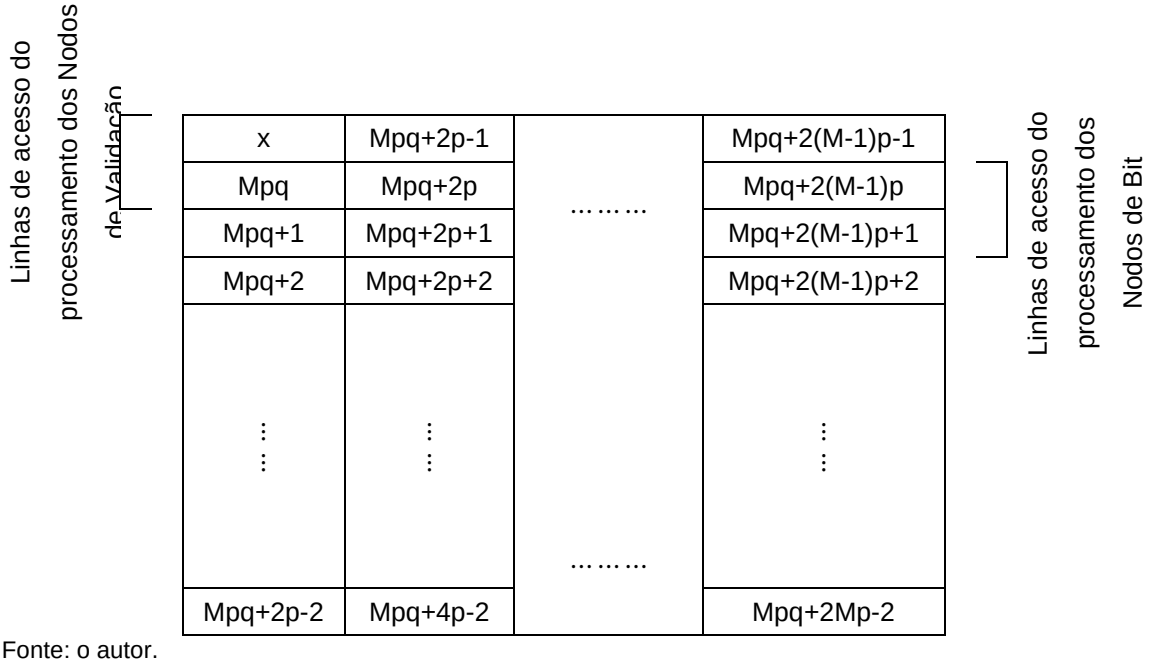
Durante o passo de Atualização dos Nodos de Bit, as linhas que precisam ser processadas estão localizadas de forma aleatória na *top* RAM seguidas de um acesso sequencial de linhas na *bottom* RAM. Dessa forma, os endereços das linhas a serem acessados durante a Atualização dos Nodos de Bit necessitam ser armazenados na ROM para cada uma das taxas especificadas pelo padrão.

Figura 4.9 - Posicionamento das linhas e acesso da top RAM



Fonte: o autor.

Figura 4.10 - Posicionamento das linhas e acesso da bottom RAM



Fonte: o autor.

Tabela 4.4 - Tamanho da RAM de todos os tamanhos de bloco e taxas de código no DVB-S2

Tamanho do bloco	Taxa do código	Nº de grupos de nodos de validação	Grau dos nodos de validação	Nº de linhas na *top* RAM	Nº de linhas de RAM
N		$p = \frac{N-K}{M}$	d_c	$q = d_c - 2$	$pq + 2p$
64800	$1/4$	135	4	2	540
	$1/3$	120	5	3	600
	$2/5$	108	6	4	648
	$1/2$	90	7	5	630
	$3/5$	72	11	9	792
	$2/3$	60	10	8	600
	$3/4$	45	14	12	630
	$4/5$	36	18	16	648
	$5/6$	30	22	20	660
	$8/9$	20	27	25	540
	$9/10$	18	30	28	540
16200	$1/5$	36	3.75*	1.75	135
	$1/3$	30	5	3	150
	$2/5$	27	6	4	162
	$4/9$	25	5.4*	3.4	135
	$3/5$	18	11	9	198
	$2/3$	15	10	8	150
	$11/15$	12	11*	9	132
	$7/9$	10	12.5*	10.5	125
	$37/49$	8	17.125*	15.125	137
	$8/9$	5	27	25	135

*Caso Especial de Taxas de Código

Fonte: o autor.

De acordo com a matriz $\boldsymbol{H}$, durante o passo de Atualização dos Nodos de Validação, cada linha da top RAM ou da bottom RAM corresponde ao conjunto que consiste em um elemento não-zero de cada linha de um grupo de nodos de validação em $\boldsymbol{H}$, onde um grupo de nodos de validação é o conjunto das linhas $i, i+p, i+2p, \cdots, i+(M-1)p$ da matriz $\boldsymbol{H}$ e $i = 0, 1, 2, \cdots, p-1$ é o índice do grupo de nodos de validação. Por isso, quando se acessa sequencialmente as linhas da top RAM, o decodificador está acessando os valores das caixas de mensagem que correspondem a cada elemento não-zero das linhas em um grupo de nodos de validação na submatriz $\boldsymbol{A}$. Quando se

acessa as linhas da bottom RAM, o decodificador está acessando os valores da caixa de mensagem que correspondem a cada elemento não-zero em um grupo de nodos de validação na submatriz $\boldsymbol{B}$.

Em outras palavras, se a célula 0 na Figura 4.9 corresponde a um elemento não-zero na linha 0 da submatriz $\boldsymbol{A}$, então as células $1, 2, \cdots q-1$ correspondem aos outros elementos não-zero na linha 0 da submatriz $\boldsymbol{A}$. Além disso, as células $pq, 2pq, \cdots, (M-1)pq$ da Figura 4.9 correspondem a um elemento não-zero nas linhas $p, 2p, \cdots, (M-1)p$ da submatriz $\boldsymbol{A}$, respectivamente.

Durante o passo de Atualização dos Nodos de Bit, cada linha na *top* RAM corresponde ao conjunto que consiste nos elementos não-zero de cada coluna de um grupo de nodos de bit, onde um grupo de nodos de bit é o conjunto de colunas $i, i+1, i+2, \cdots, i+(M-1)$ da submatriz $\boldsymbol{A}$, onde $i = 0, M, 2M, \cdots, K-M$ é o índice da primeira coluna de um determinado grupo de nodos de bit.

Do Capítulo 3, se os locais dos elementos não-zero da coluna mais à esquerda de um grupo de nodos de bit na submatriz $\boldsymbol{A}$ (primeira coluna desse grupo de nodos de bit), são $c_0, c_1, \cdots, c_{d_b-1}$, onde d_b é o grau do nodo de bit, que corresponde ao número de linhas conectadas a esse nodo de bit, então os locais dos elementos não-zero das colunas seguintes são dados pelo deslocamento descendente de $c_0, c_1, \cdots, c_{d_b-1}$ por p.

Mapeando corretamente a RAM, as células de uma linha da *top* RAM correspondem aos respectivos elementos não-zero em cada coluna de um grupo de nodos de bit. Ou seja, se a célula 0 da Figura 4.9 corresponde ao elemento não-zero da linha, ou local c_0 e coluna 0 da submatriz $\boldsymbol{A}$, então a célula pq, que está na mesma linha que a célula 0 na *top* RAM, corresponde ao elemento não-zero na linha $(c_0 + 1p) \bmod (N-K)$ e coluna 1 da submatriz $\boldsymbol{A}$. Do mesmo modo, a célula $2pq$ corresponde ao elemento não-zero na linha $(c_0 + 2p) \bmod (N-K)$ e coluna 2 da submatriz $\boldsymbol{A}$ e assim por diante.

Além disso, uma vez que 0, pq e $2pq$ estão na linha 0 da *top* RAM, o valor 0 deve estar armazenado em ROM. Similarmente, os índices das linhas das células na *top* RAM que correspondem às linhas $c_1, c_2, \cdots, c_{d_b-1}$ e coluna 0 da

submatriz $\boldsymbol{A}$ também devem estar armazenadas em ROM porque esses índice das linhas são dependentes das taxas de código.

Na submatriz $\boldsymbol{B}$ os grupos dos nodos de bit são organizados de um modo diferente. Os grupos de nodos de bit são as colunas $i, i+p, i+2p, \cdots, i+(M-1)p$ da submatriz $\boldsymbol{B}$, onde $i = 0, 1, 2, \cdots, p-1$. Uma vez que a submatriz $\boldsymbol{B}$ tem dois elementos não-zero em sequência, com exceção da coluna mais à direita, a correspondência da submatriz $\boldsymbol{B}$ na *bottom* RAM durante a Atualização dos Nodos de Bit é menos complexa. Uma vez que a célula Mpq na Figura 4.10 corresponde ao primeiro elemento não-zero da coluna 0 da submatriz $\boldsymbol{B}$, então a célula $Mpq+1$ corresponde ao outro elemento não-zero da coluna 0. As células $Mpq+2p, Mpq+4p, \cdots, Mpq+2(M-1)p$ correspondem ao primeiro elemento não-zero das colunas $p, 2p, \cdots, (M-1)p$, respectivamente.

Como pode ser visto na Figura 4.9 e na Figura 4.10, o tamanho da *top* RAM é $pq \times M$ e o tamanho da *bottom* RAM é $2p \times M$. Para que o decodificador seja capaz de suportar todas as 21 combinações de tamanho de bloco e taxas de código, o tamanho da RAM deve ser o maior valor de $(pq+2p) \times 360$ para o conjunto de taxas de código previsto no padrão. De acordo com a Tabela 4.4, a situação operacional onde o tamanho de bloco $N = 64800$ *bits* e taxa de código $3/5$ é a que exige maior RAM. Para esta situação, o tamanho da RAM é de 792×360 células de memória. Pode-se notar que algumas taxas de códigos não possuem o valor de q inteiro. Essas taxas de código serão discutidas a seguir.

Eroz et. al (EROZ, SUN, & LEE, 2006) fazem um mapeamento de cada elemento não-zero da matriz $\boldsymbol{H}$ para cada um dos locais na RAM. O artigo apresenta um exemplo de como, depois da matriz $\boldsymbol{H}$ estar corretamente mapeada, a leitura dos valores da RAM a serem processados durante o passo de Atualização dos Nodos de Validação ser sequencial e a leitura dos valores da RAM a serem processados durante o passo de Atualização dos Nodos de Bit ser indexada. No mesmo trabalho é apresentado um exemplo de mapeamento utilizando a mesma matriz $\boldsymbol{H}$ mostrada na Equação 3.4. De acordo com o autor, a matriz $\boldsymbol{H}$ deve ser mapeada da seguinte maneira:

$$\mathrm{H} = \underbrace{\left[\begin{array}{cccccccccccc|cccccc}
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right]}_{\mathrm{A}\qquad\qquad\qquad\qquad\mathrm{B}}
\begin{array}{l}
\Rightarrow e_0 \rightarrow e_5, e_{36} \\
\Rightarrow e_6 \rightarrow e_{11}, e_{37} \rightarrow e_{38} \\
\Rightarrow e_{12} \rightarrow e_{17}, e_{39} \rightarrow e_{40} \\
\Rightarrow e_{18} \rightarrow e_{23}, e_{41} \rightarrow e_{42} \\
\Rightarrow e_{24} \rightarrow e_{29}, e_{43} \rightarrow e_{44} \\
\Rightarrow e_{30} \rightarrow e_{35}, e_{45} \rightarrow e_{46}
\end{array} \qquad (4.2)$$

As células na *top* RAM e na *bottom* RAM são representadas por e_i e correspondem às linhas no gráfico de Tanner, ou simplesmente, aos elementos não-zero da matriz $\boldsymbol{H}$. Como mostrado na Equação 4.2, os seis elementos não-zero na linha 0 da submatriz $\boldsymbol{A}$ são mapeados em $e_0 \rightarrow e_5$, os seis elementos na linha 1 são mapeados em $e_6 \rightarrow e_{11}$ e assim por diante. Entretanto, para a realização do passo de Atualização dos Nodos de Bit, é necessário saber exatamente qual elemento não-zero equivale a qual linha. Já na submatriz $\boldsymbol{B}$, o elemento da linha 0 é mapeado em e_{36}, os dois elementos da linha 1 são mapeados em $e_{37} \rightarrow e_{38}$ e assim por diante.

De acordo com o mapeamento proposto, a *top* RAM e *bottom* RAM da matriz $\boldsymbol{H}$ serão conforme

$$top\ RAM = \begin{bmatrix}
e_0 & e_{12} & e_{24} \\
e_1 & e_{13} & e_{25} \\
e_2 & e_{14} & e_{26} \\
e_3 & e_{15} & e_{27} \\
e_4 & e_{16} & e_{28} \\
e_5 & e_{17} & e_{29} \\
e_6 & e_{18} & e_{30} \\
e_7 & e_{19} & e_{31} \\
e_8 & e_{20} & e_{32} \\
e_9 & e_{21} & e_{33} \\
e_{10} & e_{22} & e_{34} \\
e_{11} & e_{23} & e_{35}
\end{bmatrix} \qquad bottom\ RAM = \begin{bmatrix}
x & e_{39} & e_{43} \\
e_{36} & e_{40} & e_{44} \\
e_{37} & e_{41} & e_{45} \\
e_{38} & e_{42} & e_{46}
\end{bmatrix}. \qquad (4.3)$$

Eroz et. al (EROZ, SUN, & LEE, 2006) mostra que a relação entre a matriz $\boldsymbol{H}$ e a RAM será, então, conforme

$$\boldsymbol{H} = \left[\begin{array}{cccccccccccc|cccccc} e_0 & 0 & e_1 & 0 & e_2 & 0 & 0 & 0 & e_3 & e_4 & e_5 & 0 & e_{36} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e_6 & e_7 & e_8 & 0 & e_9 & e_{10} & 0 & 0 & 0 & e_{11} & e_{37} & e_{38} & 0 & 0 & 0 & 0 \\ e_{13} & e_{12} & 0 & 0 & 0 & e_{14} & e_{15} & 0 & 0 & 0 & e_{16} & e_{17} & 0 & e_{39} & e_{40} & 0 & 0 & 0 \\ e_{18} & 0 & 0 & 0 & e_{19} & e_{20} & 0 & e_{21} & e_{22} & e_{23} & 0 & 0 & 0 & 0 & e_{41} & e_{42} & 0 & 0 \\ 0 & e_{25} & e_{24} & e_{26} & 0 & 0 & 0 & e_{27} & 0 & e_{28} & 0 & e_{29} & 0 & 0 & 0 & e_{43} & e_{44} & 0 \\ 0 & e_{30} & 0 & e_{32} & 0 & e_{31} & e_{34} & 0 & e_{33} & 0 & e_{35} & 0 & 0 & 0 & 0 & 0 & e_{45} & e_{46} \end{array}\right] \quad (4.4)$$

Quando é feito o mapeamento da submatriz $\boldsymbol{B}$ para a *bottom* RAM, a primeira célula da *bottom* RAM não é utilizada. Na sequência, os elementos não-zero da submatriz $\boldsymbol{B}$ são mapeados conforme

$$\boldsymbol{B} = \left[\begin{array}{ccccccccc} 1 & & & & & & & & \\ \downarrow & & & & & & & & \\ 1 & \rightarrow & 1 & & & & 0 & & \\ & & \downarrow & & & & & & \\ & & 1 & \rightarrow & & & & & \\ & & & & \ddots & & & & \\ & & & & & \rightarrow & 1 & & \\ & & 0 & & & & \downarrow & & \\ & & & & & & 1 & \rightarrow & 1 \end{array}\right]. \quad (4.5)$$

Eroz et. al (EROZ, SUN, & LEE, 2006), no entanto, apenas apresenta o método de mapeamento da submatriz $\boldsymbol{B}$ nas respectivas posições da bottom RAM, não mostrando um modo de mapear a submatriz $\boldsymbol{A}$ nas respectivas posições da top RAM. Uma possibilidade de mapeamento da submatriz $\boldsymbol{A}$, seria a expansão dos valores do Apêndice A para a matriz $\boldsymbol{H}$, conforme a Equação 3.3 e buscar cada elemento não-zero em cada linha e, ao encontrá-lo, designar a primeira posição livre da RAM. Este método, no entanto, não seria eficiente, uma vez que as matrizes do padrão DVB-S2 são de ordem muito grande. Para mapear corretamente os elementos da submatriz $\boldsymbol{A}$ na top RAM foi utilizada a técnica proposta por Loi em (LOI, 2010).

4.3.2.2 Definição dos Coeficientes da ROM

Para mapear a submatriz $\boldsymbol{A}$ na top RAM é necessário apenas os valores dados no Apêndice A. É possível converter esses valores em coeficientes que são armazenados na ROM. Mais especificamente, são dois coeficientes obtidos a partir dos valores contidos no Apêndice A: coeficiente row e coeficiente shift.

O coeficiente row é utilizado para indicar qual linha da RAM a FU terá que acessar durante o passo de Atualização de Nodos de Bit. Já o coeficiente shift,

será utilizado pelo Shuffle Network para reposicionar as colunas da RAM também durante o passo de Atualização dos Nodos de Bit.

Como exemplo, é utilizada a matriz da Equação 3.4, reproduzida na Equação 4.6, a qual apresenta as mesmas características das matrizes do padrão DVB-S2.

$$\boldsymbol{H} = \left[\begin{array}{cccccccccccc|cccccc} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \tag{4.6}$$

Os parâmetros dessa matriz são os seguintes: tamanho de palavra-código $N = 64800$, tamanho de mensagem $K = 12$, periodicidade $M = 3$ e parâmetros $p = 2$ e $q = 6$.

Se a matriz da Equação 4.6 estivesse no padrão DVB-S2, os valores que apareceriam no Apêndice A seriam os seguintes:

$$\begin{array}{ccc} 0 & 2 & 3 \\ 1 & 4 & 5 \\ 1 & 2 & 5 \\ 0 & 3 & 4 \end{array} \tag{4.7}$$

Esses valores são obtidos através da matriz $\boldsymbol{H}$ na Equação 4.6. A primeira coluna do primeiro grupo de nodos de bit possui elementos não-zero nas linhas 0, 2 e 3. As outras colunas (coluna 2 e coluna 3) desse grupo são a repetição dessa primeira coluna, mas deslocada para baixo pelo parâmetro $p = 2$. Por isso, o primeiro conjunto de coeficientes é 0, 2 e 3, que são os valores contidos na primeira linha da Equação 4.7. A primeira coluna do segundo grupo de nodos de bit (coluna 4) da Equação possui elementos não-zero nas linhas 1, 4 e 5. As outras colunas desse grupo (coluna 5 e coluna 6) são a repetição da coluna 4, deslocada por $p = 2$. O segundo conjunto de valores de coeficientes são 1, 4 e 5, que são os valores contidos na segunda linha da Equação 4.7.

A partir da Equação 4.7 é possível encontrar os valores do coeficiente row e do coeficiente shift para qualquer taxa de código do padrão DVB-S2 de acordo com o algoritmo a seguir apresentado (Algoritmo 4.2).

Algoritmo 4.2 Algoritmo de Definição dos Coeficientes da ROM para qualquer taxa de código (exceto para as taxas de códigos especiais)

1. Ler os valores do Apêndice A para uma determinada taxa de código
2. Inicializar um vetor $LUT = [0, q, 2q, \cdots, (p-1)q]$

Para cada valor (g) lido do Apêndice A **fazer**

3a. $index = g \bmod p$

3b. obter $row = LUT(index)$

3c. $LUT(index) = LUT(index) + 1$

3d. obter $shift = g \div p$

Fim para

Retornar cada valor *row* e *shift* obtidos

Aplicando o Algoritmo 4.2 à matriz da Equação 4.7, foram obtidos os seguintes coeficientes:

$$\begin{aligned} row &= [0, 1, 6, 7, 2, 8, 9, 3, 10, 4, 11, 5] \\ shift &= [0, 1, 1, 0, 2, 2, 0, 1, 2, 0, 1, 2] \end{aligned} \tag{4.8}$$

Os valores de *shift* e *row* devem ser obtidos *off-line* e devem ser armazenados no bloco ROM. O algoritmo 4.2 serve para obter os coeficientes para qualquer código LDPC que apresente as mesmas estruturas dos códigos definidos no padrão DVB-S2.

4.3.2.3 Casos Especiais de Taxas de Código

Na Tabela 4.4, algumas taxas de código estão marcadas como casos especiais de taxas de código, apesar desses códigos não serem marcados como especiais no padrão. A razão para que essas taxas de códigos estejam marcadas é que o peso das linhas da submatriz $\boldsymbol{A}$ não é sempre constante, como acontece nas demais taxas de código. Além disso, essas taxas de códigos possuem linhas da submatriz $\boldsymbol{A}$ com pesos que variam entre dois e cinco valores diferentes, conforme Tabela 4.5.

Tabela 4.5 - Peso das Linhas da Submatriz A das Taxas de Códigos Especiais

Taxa	Peso da Linha	Índice dos grupos de nodos de validação
$1/4$	1	6, 13, 14, 15, 20, 26, 27, 28, 30
	2	0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 16, 17, 18, 19, 21, 22, 23, 24, 25, 29, 31, 32, 33, 34, 35
$4/9$	2	4, 6, 16, 19
	3	0, 5, 7, 9, 10, 15, 18, 21, 23
	4	1, 3, 8, 11, 12, 13, 14, 17, 20, 22
	5	2, 24
$11/15$	7	3
	8	0, 4, 9
	9	2, 6, 8, 10
	10	1, 7, 11
	11	5
$7/9$	9	1
	10	0, 3, 4
	11	2, 5, 6, 7, 8, 9
$37/45$	14	0, 1, 2, 3
	15	7
	16	5
	17	4,6

Fonte: Loi, Kung (2010).

Os pesos variados das linhas dessas taxas de código afetam o passo de Atualização dos Nodos de Validação, porque as FUs não acessam mais um número constante q de linhas da RAM, ainda mais que, em alguns casos, o valor de q nem mesmo é um valor inteiro. As características especiais dessas taxas de código também afetam o mapeamento dos elementos não-zero da submatriz $\boldsymbol{A}$ para a top RAM.

Apesar disso, a periodicidade $M = 360$ ainda existe, o que significa que se a linha i na matriz $\boldsymbol{A}$, para $0 \leq i < p$, tem um peso de linha particular, então as linhas $i + p, i + 2p, i + 3p, \cdots, i + (M - 1)p$, que pertencem ao mesmo grupo de nodos de validação, todas têm o mesmo peso de linha.

Na Tabela 4.5, os índices dos grupos dos nodos de validação identificam os grupos de nodos de validação em uma determinada taxa de código que possui um particular peso de linha. Utilizando a propriedade da periodicidade dos pesos

das linhas, durante o passo de Atualização dos Nodos de Validação, as FUs leem o número de linha da top RAM de acordo com o valor do peso da linha e do índice dos grupos de nodos de validação dados na Tabela 4.5 ao invés de ler sempre q linhas da top RAM, que é o caso das demais taxas de código.

Por exemplo, no taxa de código $37/45$, onde $p = 8$, as FUs leem as primeiras 14 linhas da top RAM para processar o grupo de nodos de validação com índice 0, que corresponde a processar as linhas 0, 8, 16, ⋯, 2872 da submatriz $\boldsymbol{A}$. Na sequência, as FUs leem as próximas 14 linhas para processar o grupo de nodos de validação de índice 1, que correspondem às linhas 1, 9, 17, ⋯, 2873 da submatriz $\boldsymbol{A}$. Após isso, as FUs leem as próximas 14 linhas para o grupo de nodos de validação de índice 2 e outras 14 linhas para o grupo de nodos de validação de índice 3. Então, as FUs leem 17 linhas da top RAM para o grupo 4 dos nodos de validação, 16 linhas para o grupo 5 dos nodos de validação, 17 linhas para o grupo 6 dos nodos de validação e 15 linhas para o grupo 7 dos nodos de validação.

Algoritmo 4.3 Algoritmo de Definição dos Coeficientes da ROM para as taxas de códigos especiais

3. Ler os valores do Apêndice A para uma determinada taxa de código
4. Inicializar um vetor $LUT = [0, rw_0, rw_0 + rw_1, rw_0 + rw_1 + rw_2, \cdots]$

Para cada valor (g) lido do Apêndice A **fazer**

3a. $index = g \bmod p$

3b. obter $row = LUT(index)$

3c. $LUT(index) = LUT(index) + 1$

3d. obter $shift = g \div p$

Fim para

Retornar cada valor *row* e *shift* obtidos

Para gerar os coeficientes row e os coeficientes shift dessas taxas de código, é preciso fazer uma pequena modificação no Algoritmo 4.2, mais especificamente na inicialização do vetor LUT. Ao invés de inicializar o vetor $LUT = [0, q, 2q, \cdots, (p-1)q]$, é preciso inicializar o vetor $LUT = [0, rw_0, rw_0 +$

$rw_1, rw_0 + rw_1 + rw_2, \cdots]$, onde rw_i é o peso das linhas do grupo de nodos de validação de índice i. O procedimento para gerar os coeficientes row e shift para os casos especiais, então, segue de acordo com o Algoritmo 4.3.

4.3.3 Arquitetura do *Shuffle Network*

Os coeficientes *shift* discutidos nas seções anteriores, como dito, são armazenados na ROM. Esses coeficientes são utilizados pelo bloco *Shuffle Network* para realizar deslocamentos cíclicos nas saídas das Unidades de Cálculo (FUs) antes que sejam armazenadas na RAM. As saídas das FUs precisam ser deslocadas por causa do esquema de mapeamento de memória discutido na Seção 4.3.2.1. Cada FU acessa e processa uma coluna na *Top* RAM e uma coluna na *bottom* RAM.

Considerando como exemplo a matriz $\boldsymbol{H}$ na Equação 4.4, a *top* RAM e a *bottom* RAM na Equação 4.2, $\boldsymbol{H}$ possui periodicidade $M = 3$, então, é necessário a implementação de 3 FUs, denotadas FU_0, FU_1 *e* FU_2, as quais são responsáveis, respectivamente, pelas colunas 0, 1 e 2. Considere-se o acesso à *top* RAM para a FU_0. Durante o passo de Atualização dos Nodos de Validação, a FU_0 acessa as células e_0, e_1, e_2, e_3, e_4 e e_5 da *top* RAM porque todas estão na linha 0 da submatriz $\boldsymbol{A}$ na Equação 4.3. Uma vez que essas células estão todas na coluna 0, a FU_0 simplesmente precisa acessar os conteúdos das células da RAM sequencialmente. Da mesma forma, o acesso à RAM da FU_1 *e* FU_2 também é sequencial e o acesso à RAM para os outros grupos de nodos de validação são realizados similarmente.

Ao final do passo de Atualização dos Nodos de Validação, assume-se que os novos valores da saída da FU_0 são escritos de volta nos mesmo locais da *top* RAM em que foram lidos no início do processo. Durante o passo de Atualização dos Nodos de Bit, a FU_0 é responsável pelo processamento da primeira coluna de cada grupo de nodos de bit. Por isso, a FU_0 precisa processar os conteúdos das células e_0, e_{13} e e_{18} para a primeira coluna do primeiro grupo de nodos de bit, de acordo com a Equação 4.4. Na *top* RAM, o conteúdo dessas células estão na linha 0, coluna 0; linha 1, coluna 1; e linha 6, coluna 1 da *top* RAM,

respectivamente, que são os coeficientes *row* e *shift* armazenados na ROM para o primeiro grupo de nodos de bit, conforme mostrado na Equação 4.7. Para que a FU_0 tenha acesso aos conteúdos corretos, é preciso que os valores das linhas 0, 1 e 6 sejam deslocados para a esquerda ciclicamente por 0, 1 e 1, respectivamente, antes de entrarem na FU_0. Além disso, ao final do passo de Atualização de Nodos de Bit, as saídas da FU_0 necessitam ser deslocadas para a direita ciclicamente de volta por 0, 1 e 1 antes de serem armazenadas na RAM para que a FU_0 seja capaz de acessar os conteúdos das células de forma apropriada no passo de Atualização de Nodos de Validação da próxima iteração. Por isso são necessários dois módulos de deslocamento cíclico, chamados de *Shuffle Network*. Um entre as saídas da RAM e as entradas das FUs e outro entre as entradas da RAM e as saídas das FUs.

Para reduzir a utilização dos recursos de *hardware* da FPGA, apenas um *Shuffle Network* é implementado entre as saídas das FUs e as entradas da RAM, conforme mostra o diagrama de blocos da Figura 4.7. Essa opção é possível se for implementada tanto a operação de deslocamento cíclico para a direita quanto a operação de deslocamento cíclico para a esquerda no mesmo *Shuffle Network*.

Além disso, dois conjuntos de coeficientes *shift* são armazenados para um acesso mais eficiente da ROM. Considere-se o exemplo onde os valores de *row* e *shift* são mostrados na Equação 4.8. Durante o passo de Atualização dos Nodos de Bit, os valores de *shift* estão na mesma ordem como mostrado na Equação 4.7. Desse jeito, o acesso à ROM para os coeficientes *shift* é sequencial durante o passo de Atualização dos Nodos de Bit, uma vez que a linha da *top* RAM que é acessada é indexada pelo coeficiente *row*, que está no mesmo endereço da ROM que o coeficiente *shift.* Entretanto, durante o passo de atualização dos Nodos de Validação, o acesso à *top* RAM é sequencial. Então, para evitar a necessidade de procurar na ROM o endereço dos coeficientes *shift*, um outro conjunto de coeficientes *shifts* são armazenados na ROM, conjunto este chamado *ishift*. Além disso, como um outro conjunto de coeficientes *shift* é armazenado na ROM, a arquitetura do *Shuffle Network* pode ser ainda mais simplificada se implementar somente a operação de deslocamento cíclico para a direita e armazenar os valores $ishift = M - shift$ na ROM. Resumindo, para se obter os coeficientes *ishift* e seus respectivos endereços na ROM, é preciso

ordenar a lista dos coeficientes *ishift* em ordem crescente dos seus respectivos coeficientes *row* e gerar os valores de *ishift* utilizando $ishift = M - shift$. Os valores resultantes são mostrados na Tabela 4.6.

Tabela 4.6 - Coeficientes row, shift e ishift na ROM do exemplo

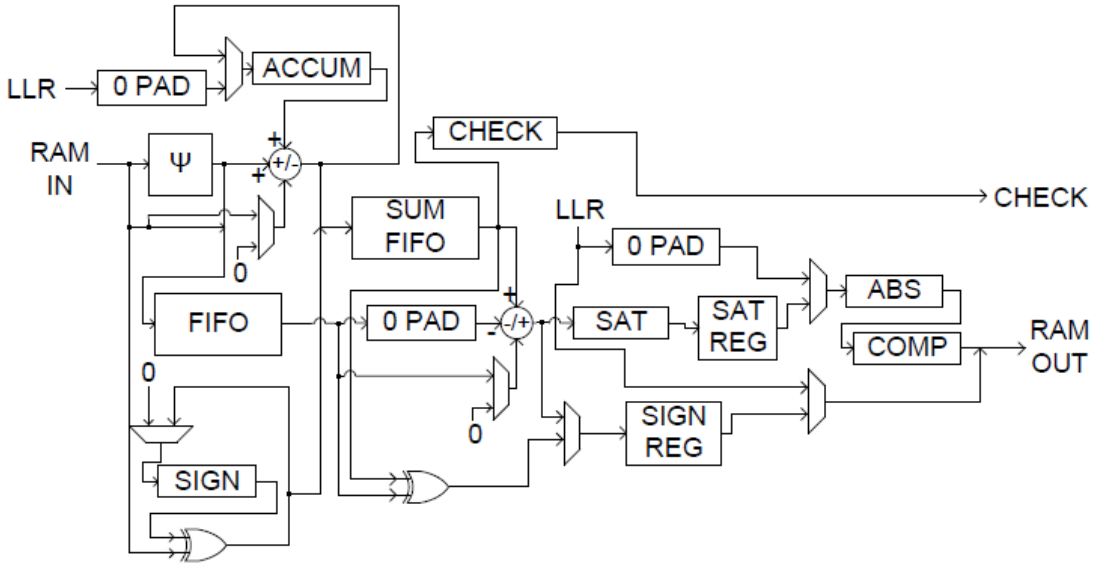
Local na ROM	*row*	*shift*	*ishift*
0	0	0	3
1	1	1	2
2	6	1	1
3	7	0	2
4	2	2	3
5	8	2	1
6	9	0	2
7	3	1	3
8	10	2	1
9	4	0	3
10	11	1	1
11	5	2	2

Fonte: Loi, Kung (2010).

4.3.4 Arquitetura das Unidades de Cálculo (FUs)

As Unidades de Cálculo (FUs) são utilizadas para computar a Equação 3.21 e a Equação 3.24. A FU implementada é baseada em (LOI, 2010). O diagrama de blocos da FU é mostrado na Figura 4.11. Cada FU é uma estrutura híbrida que é capaz de realizar a Equação 3.21 e a Equação 3.24. Como os dois cálculos não são realizados no mesmo momento no algoritmo SPA, conforme descrito no Capítulo 3, as duas operações são combinadas em um único módulo, visando reduzir os recursos de *hardware*. As operações realizadas pelo módulo da FU são descritas a seguir, utilizando o diagrama de blocos da Figura 4.11 como referência.

Figura 4.11 - Diagrama de Blocos da Unidade de Cálculo



Fonte: adaptado de Loi, Kung (2010).

4.3.4.1 O Passo de Inicialização

Durante o passo de Inicialização do algoritmo SPA, os valores LLR são inseridos no decodificador no *Buffer* LLR. Na sequência, esses valores são armazenados na RAM quando se atribui cada valor LLR a um nodo de *bit* da caixa de mensagem. Uma vez que os valores da RAM são comprimidos utilizando o bloco COMP dentro da FU, os valores LLR também precisam ser comprimidos antes de serem armazenados na RAM, durante o passo de Inicialização. Para evitar a implementação de um outro conjunto de blocos COMP fora da FU, os valores LLR são inseridos nas FUs durante o passo de Inicialização através da entrada LLR, conforme ilustra o diagrama de blocos da Figura 4.11. O multiplexador seleciona o valor LLR no lugar do valor contido no registrador SAT REG para ser utilizado pelos blocos ABS e COMP. O *bit* de sinal do LLR é selecionado pelo multiplexador para ser combinado com a saída do bloco COMP, de modo a formar o valor de saída RAM OUT.

4.3.4.2 O Passo de Atualização dos Nodos de Validação

A FU recebe os valores vindos da RAM pela entrada RAM IN. Do Capítulo 3, é visto que durante o passo de Atualização dos Nodos de Validação,

q linhas da *top* RAM e 2 linhas da *bottom* RAM são processadas para cada grupo de nodos de validação. Uma vez que q varia para cada taxa de código, a FU adota um sistema de entrada serial onde uma linha da RAM é acessada por ciclo de *clock*, o que significa que o conteúdo de uma célula na linha da RAM é enviado para a entrada RAM IN de uma FU a cada ciclo de *clock*.

Durante o passo de Atualização dos Nodos de Validação, quando cada um dos valores da RAM é inserido na FU, o *bit* de sinal é enviado à porta lógica XOR no canto esquerdo da Figura 4.11, onde os *bits* de sinal de cada valor da RAM são acumulados no registrador SIGN. O registrador SIGN é iniciado com o valor 0, através do multiplexador conectado a sua entrada, antes de cada conjunto de entrada $q + 2$ valores de RAM. A magnitude de cada valor de RAM é processado pelo bloco Ψ, que realiza a função Ψ, conforme a Equação 3.22, e sua implementação é mostrada na Seção 4.3.4.1. A saída da função Ψ é somada com o valor presente no registrador ACCUM e armazenado de volta no registrador ACCUM para realizar o somatório na Equação 3.21. O registrador ACCUM também é inicializado com o valor 0, através do multiplexador na sua entrada, colocando a entrada LLR em 0. A terceira entrada do somador/subtrator seleciona se o somador/subtrator realizada a operação de soma ou a operação de subtração. Durante o passo de Atualização dos Nodos de Validação, o multiplexador à direita do bloco Ψ, na Figura 4.11, seleciona o valor 0 para a soma, uma vez que as saídas do bloco Ψ são somadas juntas. As saídas do bloco Ψ são concatenadas com o bit de sinal dos valores da RAM e armazenados na FIFO, que é uma fila do tipo “primeiro que entra é o primeiro que sai”. Uma vez que os registradores ACCUM e SIGN acumularam $q + 2$ valores, seus conteúdos são concatenados e armazenados na SUM FIFO. A saída da SUM FIFO é separada novamente em bit de sinal e parte de magnitude. A parte da magnitude é utilizada pelo subtrator/somador, onde a saída do bloco Ψ, que está armazenada na FIFO, é lida para ser subtraída da soma acumulada. A terceira entrada do subtrator/somador é usado para selecionar entre a subtração ou a adição e é controlada por um multiplexador que tem como saída o valor 0, o que representa que o subtrator/somador realizará a operação de subtração durante o passo de Atualização dos Nodos de Bit. A saída da subtração é saturada pelo bloco SAT. O registrador SAT REG armazena temporariamente o valor de saída

do bloco SAT por motivos de sincronização. Por fim, o valor de SAT REG é comprimido pelo bloco COMP. Essas operações são discutidas adiante. O bit de sinal da saída da SUM FIFO é utilizado como entrada da porta lógica XOR antes do registrador SIGN REG. A outra entrada da porta lógica XOR é o bit de sinal do valor de RAM que foi armazenado na FIFO. A porta lógica XOR realiza a "subtração" do sinal na segunda parte da Equação (3.9). A saída da porta lógica XOR é selecionada pelo multiplexador para ser temporariamente armazenada no registrador SIGN REG por motivos de sincronização. A saída do registrador SIGN REG é selecionada pelo próximo multiplexador para ser concatenada com a saída do bloco COMP. O valor combinado é o valor de saída RAM OUT que é enviado ao *Shuffle Network* e subsequentemente armazenados novamente na RAM.

4.3.4.3 O Passo de Atualização dos Nodos de *Bit*

A FU recebe os valores vindos da RAM pela entrada RAM IN. Do Capítulo 3, é visto que durante o passo de Atualização dos Nodos de *Bit*, para cada grupo de nodos de *bit*, o número de linhas da *top* RAM que são processadas depende do grau de nodo de *bit* d_b do referido grupo de nodo de bit. Uma vez que o grau de nodo de *bit* d_b varia para cada taxa de código, a FU adota um sistema de entrada serial onde uma linha da RAM é acessada por ciclo de *clock*, o que significa que o conteúdo de uma célula na linha da RAM é enviado para a entrada RAM IN de uma FU a cada ciclo de *clock*.

Durante o passo de Atualização dos Nodos de Bit, d_b valores são inseridos nas FUs pela entrada RAM IN. Uma vez que os cálculos dos sinais estão incluídos na soma, e não separados como no passo de Atualização dos Nodos de Validação, a porta lógica XOR e o registrador SIGN não são utilizados no Passo de Atualização dos Nodos de Bit. O bit de sinal simplesmente é separado da parte da magnitude. A parte da magnitude é utilizada como entrada do bloco Ψ e a saída é enviada ao somador/subtrator, onde a soma é acumulada no registrador ACCUM. Durante o passo de Atualização dos Nodos de Bit, o multiplexador na entrada seletora do somador/subtrator tem como saída o bit de sinal da entrada RAM IN. Por isso, caso o valor de RAM IN seja positivo, então o bit de sinal é 0 e o somador/subtrator realiza a soma dos valores, caso o valor

de RAM IN seja negativo, então o bit de sinal é 1 e o somador/subtrator realiza a subtração dos valores. O registrador ACCUM é inicializado com o valor da entrada LLR, que é lido do *Buffer* LLR, para adicionar o termo λ_j da Equação 3.24 ao somatório. Similar ao passo de Atualização dos Nodos de Validação, a saída da bloco Ψ e o bit de sinal da entrada RAM IN são concatenados e armazenados na FIFO. Uma vez que d_b valores foram acumulados, a soma resultante no registrador ACCUM é armazenada na SUM FIFO. Assim como no passo de Atualização dos Nodos de Validação, a saída da SUM FIFO é aplicada à entrada do subtrator/somador, onde cada saída do bloco Ψ armazenada na FIFO é usada para ser subtraída da soma. Uma vez que os valores do sinal e da magnitude são utilizados juntos no cálculo do passo de Atualização dos Nodos de Bit, o bit de sinal da FIFO é selecionado pelo multiplexador para ser utilizado como entrada seletora do subtrator/somador, de modo que, se o sinal do valor armazenado na FIFO for positivo, o subtrator/somador realiza a operação de subtração, se o sinal do valor armazenado na FIFO for negativo, o subtrator/somador realiza a operação de adição. O valor armazenado na SUM FIFO é, também, o valor S_j na Equação 3.25 do passo de Tomada de Decisão. Então, o bit de sinal do valor armazenado na SUM FIFO, que é z_j na Equação 3.26, é temporariamente armazenado no registrador CHECK por motivos de sincronização, e é utilizado como a saída CHECK da FU, o qual é enviado ao Módulo de Validação de Paridades (PCM) e ao *Buffer* de Mensagem Decodificada. A saída do subtrator/somador é separada em bit de sinal e parte de magnitude. A parte de magnitude é saturada pelo bloco SAT, armazenada temporariamente pelo bloco SAT REG por motivos de sincronização, e comprimida pelo bloco COMP. O bit de sinal é selecionado pelo multiplexador para ser armazenado temporariamente no registrador SIGN REG e, na sequência, ser selecionado pelo próximo multiplexador para ser concatenado com a saída do bloco COMP e formar a saída RAM OUT da FU.

De acordo com o esquema de mapeamento de memória visto na Seção 4.3.2.1, são necessárias 360 FUs para a implementação do decodificador. Essas 360 FUs operam em paralelo e independentes uma da outra, de modo que todas as linhas e colunas de um mesmo grupo de nodos de validação ou um mesmo grupo de nodos de bit são processados simultaneamente. Coletivamente, todas

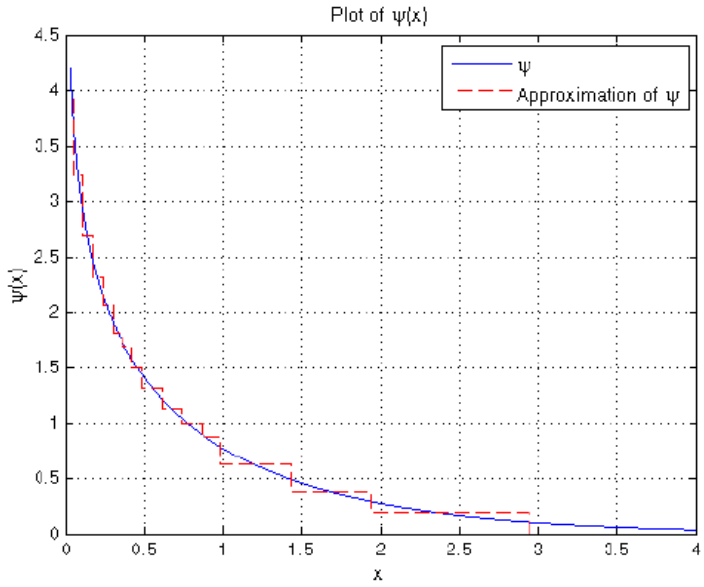
as FUs começam os cálculos de um grupo de nodos de validação ou grupo de nodos de bit juntas. Além disso, as 360 saídas das 360 FUs a serem reescritas na RAM pertencem sempre à mesma linha da RAM, assim como os 360 valores que foram lidos da RAM para serem entradas das 360 FUs.

Uma vez que 360 FUs são necessárias para a implementação do decodificador LDPC, qualquer redução dos recursos de *hardware* de uma FU, através de otimização, resulta em uma redução de 360 vezes na utilização dos recursos de *hardware*, uma vez que essa otimização é realizada em 360 FUs.

4.3.4.4 Implementação da Função ψ

O bloco Ψ na Figura 4.11 representa a implementação da função Ψ na Equação. O gráfico da função $\Psi(x)$ comparado ao resultado obtido pela implementação do bloco $\Psi(x)$ é mostrado na Figura 4.12.

Figura 4.12 - Gráfico da função Ψ e sua aproximação.



Fonte: Loi, Kung (2010).

É possível notar na Figura 4.12 que para grandes valores de entrada, x, os valores de saída da função $\Psi(x)$ se tornam muito pequenos. Devido à não linearidade e inclinação crescente constante, ZHANG, WANG, & PARHI sugeriram um esquema de quantização variável (ZHANG, WANG, & PARHI, 2001). Porém, para utilizar essa quantização tanto para as entradas do bloco ψ quanto para as saídas do bloco ψ, seria necessário converter o valor da saída

do formato de variável quantizada para o formato de complemento de dois antes de ser utilizada pelo somador/subtrator e acumulada no registrador ACCUM, representado na Figura 4.11, o que aumenta a utilização de *hardware*.

Tabela 4.7 - Esquema de Quantização da Função ψ

Entrada		Saída			Entrada		Saída		
Dec.	Bin.	Dec.	Bin.	Comp.	Dec.	Bin.	Dec.	Bin.	Comp.
0.00000	0.00000	Inf	11.1111	11.1111	4.000	100.000	0.0366	00.0001	00.0000
0.03125	0.00001	4.1590	11.1111		4.125	100.001	0.0323	00.0001	
0.06250	0.00010	3.4661	11.0111	11.0100	4.250	100.010	0.0285	00.0000	
0.09375	0.00011	3.0610	11.0001		4.375	100.011	0.0252	00.0000	
0.12500	0.00100	2.7739	10.1100	10.1011	4.500	100.100	0.0222	00.0000	
0.15625	0.00101	2.5515	10.1001		4.625	100.101	0.0196	00.0000	
0.18750	0.00110	2.3701	10.0110	10.0101	4.750	100.110	0.0173	00.0000	
0.21875	0.00111	2.2170	10.0011		4.875	100.111	0.0153	00.0000	
0.25000	0.01000	2.0846	10.0001	10.0001	1.000	101.000	0.7719	00.1100	00.1010
0.28125	0.01001	1.9682	01.1111		1.125	101.001	0.6737	00.1011	
0.31250	0.01010	1.8644	01.1110	01.1101	1.250	101.010	0.5895	00.1001	
0.34375	0.01011	1.7708	01.1100		1.375	101.011	0.5169	00.1000	
0.37500	0.01100	1.6856	01.1011	01.1011	1.500	101.100	0.4539	00.0111	00.0110
0.40625	0.01101	1.6076	01.1010		1.625	101.101	0.3990	00.0110	
0.43750	0.01110	1.5356	01.1001	01.1000	1.750	101.110	0.3511	00.0110	
0.46875	0.01111	1.4689	01.1000		1.875	101.111	0.3092	00.0101	
0.50000	0.10000	1.4063	01.0111	01.0101	2.000	110.000	0.2723	00.0100	00.0011
0.53125	0.10001	1.3488	01.0110		2.125	110.001	0.2400	00.0100	
0.56250	0.10010	1.2944	01.0101		2.250	110.010	0.2116	00.0011	
0.59375	0.10011	1.2432	01.0100		2.375	110.011	0.1866	00.0011	
0.62500	0.10100	1.1950	01.0011	01.0010	2.500	110.100	0.1645	00.0011	
0.65625	0.10101	1.1494	01.0010		2.625	110.101	0.1451	00.0010	
0.68750	0.10110	1.1062	01.0010		2.750	110.110	0.1280	00.0010	
0.71875	0.10111	1.0652	01.0001		2.875	110.111	0.1130	00.0010	
0.75000	0.11000	1.0262	01.0000	01.0000	3.000	111.000	0.0997	00.0010	00.0000
0.78125	0.11001	0.9891	01.0000		3.125	111.001	0.0879	00.0001	
0.81250	0.11010	0.9538	00.1111		3.250	111.010	0.0776	00.0001	
0.84375	0.11011	0.9200	00.1111		3.375	111.011	0.0685	00.0001	
0.87500	0.11100	0.8878	00.1110	00.1110	3.500	111.100	0.0604	00.0001	
0.90625	0.11101	0.8569	00.1110		3.625	111.101	0.0533	00.0001	
0.93750	0.11110	0.8274	00.1101		3.750	111.110	0.0470	00.0001	
0.96875	0.11111	0.7991	00.1101		3.875	111.111	0.0415	00.0001	

Fonte: Zhang; Wang; Pathi (2001)

Por isso, a FU utiliza o esquema de precisão variável proposto por (ZHANG, WANG, & PARHI, 2001) apenas para as entradas do bloco ψ. As saídas do bloco ψ são valores de ponto fixo de 6 *bits* no formato 2.4, onde os dois *bits* mais significativos representam a parte inteira do valor e os quatro *bits* menos significativos representam a parte fracionária do valor. Utilizando esse esquema de quantização variável, a Tabela 4.7 é gerada. A Tabela 4.7 apresenta o valor de entrada da função $\Psi(x)$ na forma decimal e a sua respectiva quantização variável no formato de ponto fixo de 6 *bits*. A Tabela 4.7 também apresenta a saída da função $\Psi(x)$ na forma decimal e a sua respectiva quantização em ponto fixo de 6 *bits* no formato 2.4.

A implementação direta da Tabela 4.7 requer uma função *Booleana* com entrada de 6 *bits* e saída de 6 *bits*, o que implica em uma *Look-Up Table* (LUT) com dimensões $2^6 \times 6$ *bits*. OH & PARHI sugeriram um método de redução do tamanho da LUT (OH & PARHI, 2006). Com o método proposto é criada a coluna "Comp." na Tabela 4.7. Como resultado, a função ψ pode ser implementada como uma LUT de dimensões $2^4 \times 6$ ao invés de uma LUT de dimensões $2^6 \times 6$ *bits*.

A LUT resultante para aproximar a função ψ é apresentada na Tabela 4.8, e o correspondente gráfico é mostrado em linhas tracejadas na Figura 4.12.

Tabela 4.8 - LUT da Função ψ

Entrada	Saída	Entrada	Saída
0000	11.1111	1000	01.0101
0001	11.0100	1001	01.0010
0010	10.1011	1010	01.0000
0011	10.0101	1011	00.1110
0100	10.0001	1100	00.0000
0101	01.1101	1101	00.1010
0110	01.1011	1110	00.0110
0111	01.1000	1111	00.0011

Fonte: o autor.

(OH & PARHI, 2006) também propõem uma função de compressão (COMP) que comprime os valores de saída RAM OUT. De acordo com a Figura 4.11, as magnitudes das entradas de RAM IN entram no bloco ψ

imediatamente após serem inseridas na FU. Uma vez que a LUT da função ψ tem como entrada um valor de tamanho 4 *bits*, mesmo que os valores de entrada RAM IN possuíssem mais de 4 *bits*, seria necessário o arredondamento ou o truncamento dos valores para ser utilizado pelo bloco ψ. Por isso, o resultado do subtrator/somador é comprimido pelo bloco COMP para reduzir o valor em complemento de dois para um valor de 4 *bits*. O valor de 4 *bits* gerado corresponde ao valor de entrada de 4 *bits* do bloco ψ. A LUT resultante é mostrada na Tabela 4.9. Fazendo isso, os valores da RAM possuem 4 *bits* de magnitude, além do bit de sinal, resultando em um tamanho de 5 *bits*. Além disso, observa-se na Tabela 4.9, que os valores de entrada do bloco COMP têm tamanho de 6 *bits* e são interpretados no formato 2.4. Qualquer valor maior que 11.1111 em binário, que é 3.9375 em decimal, é saturado para 6 *bits* porque, da Tabela 4.7, qualquer valor de entrada da função ψ maior que o valor decimal 3.0 tem como saída o valor 00.0000. Por isso o bloco SAT é utilizado antes do bloco COMP para saturar os valores no tamanho de 6 *bits*.

Tabela 4.9 - LUT da Função de Compressão

Entrada	Saída	Entrada	Saída
00.0000	0000	00.100x	1000
00.0001	0001	00.101x	1001
00.0010	0010	00.110x	1010
00.0011	0011	00.111x	1011
00.0100	0100	01.0xxx	1101
00.0101	0101	01.1xxx	1110
00.0110	0110	10.xxxx	1111
00.0111	0111	11.xxxx	1100

Fonte: o autor.

4.3.4.5 Projeto e Utilização da SUM FIFO

O uso da SUM FIFO melhora o controle de fluxo da FU, o qual é dificultado pela variação do número de valores de RAM IN que são inseridos na FU. Considere-se a taxa de código $R = 1/4$ para o tamanho de *frame* $N = 64800$ no padrão DVB-S2. A submatriz $\boldsymbol{A}$ desse código possui colunas com doze, três, dois e um elementos não-zero, ou seja, o grau dos nodos de bit d_b dessa taxa de código pode ser qualquer um desses valores.

Durante o passo de Atualização dos Nodos de *Bit*, a FU começa processando conjuntos com doze valores de RAM IN. Quando o último conjunto de doze valores de RAM IN é inserido, começam a ser inseridos conjuntos de três valores de RAM IN. Quando o primeiro conjunto de três valores RAM IN termina de ser inserido, a FU ainda está calculando os valores de RAM OUT utilizando o valor da soma do último conjunto de doze valores de RAM IN e subtraindo os valores da FIFO. Se fosse utilizado um registrador de soma no lugar da SUM FIFO, a inserção do conjunto de três valores de RAM IN deveria ser suspensa até todos os doze valores do último conjunto terem sido processados e enviados à RAM pela saída RAM OUT, momento em que o registrador estaria disponível para ser sobrescrito. Utilizando a SUM FIFO, no entanto, o conjunto de três valores não necessita ser interrompido e o valor da soma é enfileirado na SUM FIFO, enquanto os resultados RAM OUT do conjunto anterior são computados.

Utilizando uma SUM FIFO no lugar de um registrador de soma, o controle de fluxo é melhorado porque o fluxo de valores de RAM IN que entra na FU não precisa ser interrompido em nenhum momento durante a decodificação. O lado negativo, no entanto, é que a implementação de uma SUM FIFO, e não de um registrador de soma, representa aumento nos recursos de *hardware* utilizado pelo decodificador. Apesar disso, o tamanho da SUM FIFO pode ser reduzido observando o comportamento dos valores de RAM IN que são inseridos na FU.

O registrado ACCUM e o somador/subtrator na Figura 4.11 realizam a parte do somatório da Equação 3.21 e da Equação 3.24. Dadas as taxas de código do padrão DVB-S2, especificados na Tabela 4.4, o número máximo de valores de RAM que são somados na Equação 3.21 é 30 e o número máximo de valores que são somados na Equação 3.24 é 13. Como a FU computa as duas equações, o número máximo de valores acumulados é 30. Com isso, a quantidade de *bits* que o registrador ACCUM e o somador/subtrator devem ter é igual ao número de *bits* de cada valor que é somado, que é 1 bit de sinal + 6 *bits* da saída do bloco Ψ, além do limite máximo de log_2 da quantidade de valores que são somados. Por isso, o tamanho do registrador ACCUM é de $7 + \lceil log_2(30) \rceil = 7 + 5 = 12$. Contudo, uma vez que a soma armazenada no registrador ACCUM é logo em seguida armazenada na SUM FIFO, onde é

subtraída pelo valor da saída do bloco Ψ, saturada e então comprimida para gerar os valores de saída da FU, a largura da SUM FIFO pode ser reduzida.

De acordo com a Tabela 4.8, o valor máximo da saída do bloco ψ é 11.1111 em binário, que é 011.1111 em complemento de dois binário e 3.9375 em decimal. Por isso, o valor máximo armazenado no registrador ACCUM é $3{,}9375 \times 30 = 118.125$, que é 01110110.0010 em complemento de dois binário de tamanho 12 *bits*. Na sequência, essa soma deve ser subtraída por um valor da FIFO por vez, cujo valor máximo é o valor máximo da função ψ, que corresponde a 3.9375 em decimal. A diferença é então comprimida pela LUT dada na Tabela 4.9. De acordo com a Tabela 4.9, no entanto, qualquer valor maior que 11.0000 em binário, ou 3.0 em decimal, é comprimido em 1100 em binário. Por isso, se a soma armazenada no registrado ACCUM for maior que $3.0 + 3.9375 = 6.9375$ em decimal ou 0110.1111 em complemento de dois binário, independentemente do valor da FIFO que é subtraído da soma, o resultado vai ser sempre maior ou igual a 3.0 em decimal ou 011.0000 em complemento de dois binário. Esse valor é enviado ao bloco ABS, que converte o valor recebido em um número maior ou igual a 11.0000 em binário. Por fim, qualquer valor maior ou igual a 11.0000 em binário é comprimido para 1100 através do bloco COMP, de acordo com a Tabela 4.9. Desta forma, o valor no registrado ACCUM pode ser reduzido de 12 *bits* para 8 *bits* pois, se o resultado da soma armazenado na SUM FIFO for maior ou igual a 0110.1111 em binário, a saída RAM OUT é sempre 1100 independente dos valores armazenados na FIFO.

4.3.5 Arquitetura do Módulo de Verificação de Paridades

O Módulo de Verificação de Paridades (PCM) é utilizado para verificar as equações de paridade durante o passo de Tomada de Decisão no algoritmo SPA. As equações de paridade, apresentadas na Equação 3.5, são utilizadas durante a codificação dos códigos LDPC DVB-S2 e, também pelo decodificador, durante o processo de decodificação. No decodificador, são expressas da seguinte forma:

$$
\begin{aligned}
0 &= a_{0,0}x_0 \oplus a_{0,1}x_1 \oplus \cdots \oplus a_{0,K-1}x_{K-1} \oplus p_0 \\
0 &= a_{1,0}x_0 \oplus a_{1,1}x_1 \oplus \cdots \oplus a_{1,K-1}x_{K-1} \oplus p_0 \oplus p_1 \\
0 &= a_{2,0}x_0 \oplus a_{2,1}x \oplus \cdots \oplus a_{2,K-1}x_{K-1} \oplus p_1 \oplus p_2 \\
&\vdots \\
0 &= a_{N-K-1,0}x_0 \oplus a_{N-K-1,1}x_1 \oplus \cdots \oplus a_{1,K-1}x_{K-1} \oplus p_{N-K-2} \oplus p_{N-K-1}.
\end{aligned}
\tag{4.9}
$$

A diferença entre as equações de paridade utilizadas no codificador e as utilizadas no decodificador é que os bits de paridade são movidos para o lado direito da equação. No processo de verificação das equações de paridade, os N bits decodificados na forma de hard-decision, z_j, gerados na saída CHECK das FUs são inseridos nas equações e, caso todas as equações sejam verdadeiras, ou seja, se a parte da direita da equação resultar no valor 0, as equações de paridades são validadas.

Uma vez que cada uma das vinte e umas diferentes combinações de tamanhos de bloco e taxas de código tem conjuntos de equações de paridades diferentes, um PCM que verifique cada conjunto separadamente é necessário. Se cada um dos conjuntos de equações de paridade fosse implementado separadamente em uma FPGA, seriam necessários muitos recursos de hardware. A arquitetura do PCM apresentada nessa seção possui a habilidade de verificar as equações de paridade de cada um dos códigos LDPC no padrão DVB-S2.

O projeto do PCM é similar ao projeto do codificador LDPC DVB-S2 apresentado na Seção 4.2. Nos dois casos, as equações de paridade são implementadas, com exceção que o codificador utiliza os bits da mensagem para gerar os bits de paridade e o PCM utiliza tanto os bits de mensagem quanto os bits de paridade para verificar se as equações de paridades são satisfeitas ou não. Por isso, a arquitetura do PCM é baseada na arquitetura do codificador apresentado na seção 4.2 deste trabalho.

Considere a matriz $\boldsymbol{S}$ binária a seguir:

$$\boldsymbol{S} = \begin{bmatrix} S_0 & S_p & S_{2p} & \cdots & S_{(M-1)p} \\ S_1 & S_{p+1} & S_{2p+1} & \cdots & S_{(M-1)p+1} \\ S_2 & S_{p+2} & S_{2p+2} & \cdots & S_{(M-1)p+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{p-1} & S_{2p-1} & S_{3p-1} & \cdots & S_{N-K-1} \end{bmatrix} \tag{4.10}$$

onde $M = 360$ e p é o parâmetro dependente da taxa de código conforme apresenta a Tabela 3.1.

O algoritmo do PCM é mostrado conforme segue (Algoritmo 4.3).

Algoritmo 4.3 Algoritmo do Módulo de Verificação de Paridades

---

Iniciar todas as posições de $\boldsymbol{S}$ em 0

Pegar o primeiro $\underline{v}$ das FUs

Para cada uma das linhas no apêndice A **fazer**

Para $r =$ cada elemento em cada linha **fazer**

$\boldsymbol{S}(r \bmod p, :) = \boldsymbol{S}(r \bmod p, :) \oplus rot_{r \div p}(\underline{v})$

Fim para

$\underline{v} =$ próximo $\underline{v}$ das FUs

Fim para

Pegar o primeiro $\underline{p}$ das FUs

Para $i = 0$ até $p - 1$ **fazer**

Para $j = 0$ até 1 **fazer**

Se $i = p - 1$ (último índice das linhas de $\boldsymbol{S}$) **então**

$\boldsymbol{S}(r \bmod p, :) = \boldsymbol{S}(r \bmod p, :) \oplus rot_1(\underline{p})$

Senão

$\boldsymbol{S}(r \bmod p, :) = \boldsymbol{S}(r \bmod p, :) \oplus \underline{p}$

Fim se

Fim para

$\underline{p} =$ próximo $\underline{p}$ das FUs

Fim para

Se número de valores em $\boldsymbol{S} \leq 0$ **então**

PASSA VERIFICAÇÃO DAS PARIDADES

Senão

FALHA VERIFICAÇÂO DAS PARIDADES

Fim se

Durante as operações do PCM, cada elemento em $\boldsymbol{S}$ é inicializado em 0. Na Seção 4.3.4 foi mostrado que as 360 FUs geram 360 valores de 1 bit simultaneamente, que são entradas do PCM. Esses valores representam uma porção de 360 valores de z_j na Equação 3.26. Se esses 360 *bits* que foram inseridos no PCM correspondem à parte da mensagem da palavra-código, ou seja, correspondem a z_j onde $0 \leq j \leq K-1$, então eles formam um vetor $\underline{v}$, que é utilizado para atualizar a matriz $\boldsymbol{S}$ assim que são gerados seguindo a equação

$$\boldsymbol{S}(r \bmod p, :) = \boldsymbol{S}(r \bmod p, :) \oplus rot_{r \div p}(\underline{v}), \qquad (4.11)$$

onde $\boldsymbol{S}(x, :)$ é o vetor de 360 *bits* pela linha x da matriz $\boldsymbol{S}$, $rot_y(\underline{v})$ é o vetor $\underline{v}$ deslocado para a direita ciclicamente em y posições, r é um valor obtido do Apêndice A para uma dada taxa de código e $\oplus$ é o operador de soma de módulo dois *bit* a *bit*. Por último, o vetor $\underline{p}$ representa o vetor de 360 *bits* formado pelos valores vindos das FUs que correspondem a z_j, onde $K \leq j \leq N-1$. Com essas informações é possível explicar o Algoritmo 4.3.

O algoritmo começa com a matriz $\boldsymbol{S}$ tendo todos os seus valores iniciais em zero. No momento em que as porções de 360 bits são enviados pelas FUs para o PCM, a matriz $\boldsymbol{S}$ é atualizada de acordo com a Equação 4.11. Uma vez que as FUs primeiro geram a parte da mensagem da palavra-código candidata, esses bits formam os vetores $\underline{v}$ e o primeiro laço para é executado. De acordo com o Algoritmo 4.3, cada vetor $\underline{v}$ atualiza o mesmo número de linhas da matriz $\boldsymbol{S}$ que o número de elementos nas linhas do Apêndice A, que é o grau de nodos de bit d_b de cada grupo de nodos de bit. Quando todos os valores no Apêndice A tiverem sido utilizados, significa que todos os bits de validação vindos das FUs que compõem a parte da mensagem da palavra-código candidata também foram todos utilizados. O segundo laço para começa e os 360 bits vindos das FUs correspondem à parte redundante da palavra-código candidata e formam o vetor

$\underline{p}$. Durante o segundo laço para, duas linhas consecutivas da matriz $\boldsymbol{S}$ são atualizadas a cada vetor $\underline{p}$ utilizando a equação

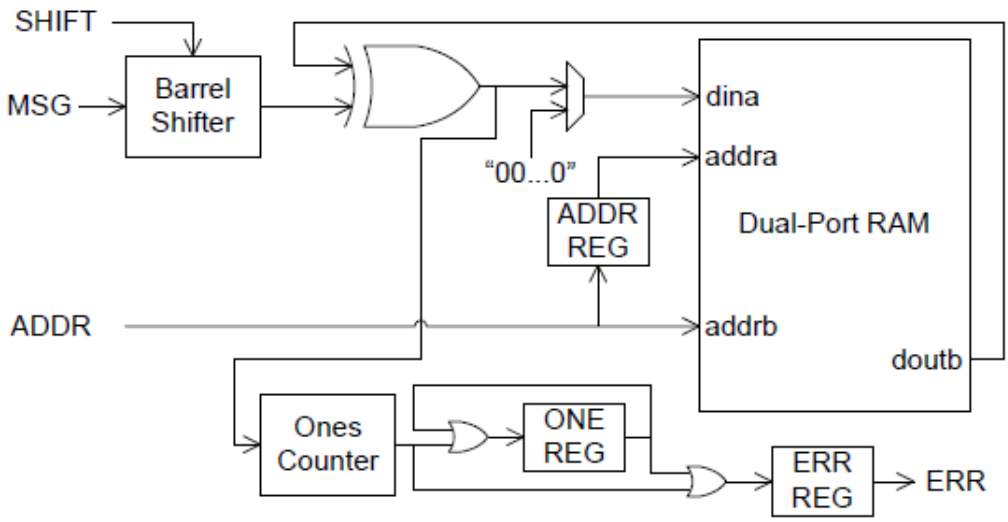
$$\boldsymbol{S}((i+j) \bmod p, :) = \boldsymbol{S}((i+j) \bmod p, :) \oplus (\underline{p}). \tag{4.12}$$

É possível notar que a Equação 4.12 é similar à Equação 4.11, com exceção que o índice das linhas da matriz $\boldsymbol{S}$ são diferentes e que não há a operação de deslocamento cíclico para a direita no vetor $\underline{p}$, a não ser quando o último vetor $\underline{p}$ é inserido no PCM, caso em que é deslocado ciclicamente para a direita em uma posição. Após o término do segundo laço **para**, todos os últimos 360 *bits* de validação vindos das FUs já foram processados pelo PCM e a matriz $\boldsymbol{S}$ resultante contém o resultado de todas as equações de paridade. Cada elemento em $\boldsymbol{S}$ corresponde a um resultado de uma equação na Equação 4.9. Por isso, se todos os elementos na matriz $\boldsymbol{S}$ forem iguais a zero, então a palavra-código candidata passa pela verificação das equações de paridade e se torna a palavra-código decodificada. Caso algum elemento da matriz $\boldsymbol{S}$ seja diferente de zero, a palavra-código falha a verificação das equações de paridade. Neste caso, a decodificação não é completada e o decodificador retorna ao passo de Atualização dos Nodos de Validação e itera mais uma vez.

É importante salientar que o Algoritmo 4.3 só funciona com o controle de fluxo do decodificador conforme especificado nesse capítulo, devido à ordem que os vetores $\underline{v}$ e $\underline{p}$ são gerados. No decodificador apresentado, quando as FUs geram a parte referente à mensagem da palavra-código candidata, os primeiros 360 *bits* de validação são gerados em ordem sequencial, ou seja, os *bits* $0 \rightarrow 359$ da palavra-código são gerados, seguidos dos *bits* $360 \rightarrow 719$ e assim por diante. Quando a parte referente à redundância da palavra-código candidata é gerada, o conjunto dos 360 *bits* são gerados da seguinte forma: são gerados os *bits* $0, p, 2p, \cdots, 359p$ da parte redundante da palavra-código, seguido pelos *bits* $1, p+1, 2p+1, \cdots, 359p+1$ e assim por diante. De acordo com essa ordem de geração dos *bits*, após o primeiro laço **para** no Algoritmo 4.3, a matriz $\boldsymbol{S}$ contém o acúmulo da operação $\oplus$ de todos $a_{x,y}x_x$ da Equação 4.9.O segundo laço **para** completa as equações de paridade aplicando $\oplus\, p_z$ aos valores já contidos na matriz $\boldsymbol{S}$.

O diagrama de blocos do módulo de validação de paridades é apresentado na Figura 4.13. Os componentes principais do PCM são o *Barrel Shifter* (BS), a RAM *dual-port* e o *Ones Counter* (OC).

Figura 4.13 - Diagrama de Blocos do Módulo de Validação de Paridades



Fonte: Loi, Kung (2010).

Os 360 *bits* de validação vindos das FUs entram no PCM através da entrada MSG. A entrada MSG é deslocada pelo *Barrel Shifter* pelo número de posições selecionados pela entrada SHIFT. Os valores que entram em SHIFT são os mesmo valores *shift* gerados pelo Algoritmo 4.2. O BS realiza as operações $rot_{r \div p}(\underline{v})$ e $rot_1(\underline{p})$ na Equação 4.11 e na Equação 4.12, respectivamente.

A saída do BS é enviada a uma das entradas da porta lógica XOR, que realiza a operação $\oplus$. A outra entrada da porta lógica XOR é a saída $doutb$ da RAM *dual-port*, onde a matriz $\boldsymbol{S}$ está armazenada. A saída $doutb$ é selecionada pela entrada *addrb*, a qual é controlada pela entrada ADDR. Os valores da entrada ADDR são, na verdade, os valores de $r \bmod p$, que estão armazenados na ROM juntamente com os valores *row*, *shift* e também um contador, que tem como saída os valores $(i + j) \bmod p$. A saída da porta lógica XOR é selecionada pelo multiplexador para ser armazenada novamente na RAM através da entrada *dina* da RAM, completando a atribuição das linhas da matriz $\boldsymbol{S}$ na Equação 4.11 e na Equação 4.12. A entrada *dina* da RAM escreve no endereço da RAM

definido pela entrada *addra*, o qual representa o valor de ADDR atrasado (pelo registrador ADDR REG) de um ciclo de *clock*.

Uma vez que a RAM *dual-port* armazena a matriz $\boldsymbol{S}$ de acordo com a Equação 4.10, suas dimensões são $p \times M$ *bits*. Para suportar todas as taxas de código do padrão DVB-S2, onde o valor máximo de p é 135, a RAM deve ter dimensão de 135×360 *bits*. Uma RAM *dual-port* é utilizada porque os valores da RAM são lidos (para realimentar a porta lógica XOR e realizar a operação $\oplus$ com os valores vindos do *Barrel Shifter)* e armazenados novamente na RAM através da outra porta.

É possível notar, no Algoritmo 4.3, que durante o segundo laço **para**, as linhas 0 e 1 da matriz $\boldsymbol{S}$ são atualizadas primeiro, seguido pelas linhas 1 e 2. Após o PCM atualizar as linhas 1 e 2, durante o segundo laço **para**, a linha 1 nunca mais será atualizada. Por isso, após a atualização das linhas 1 e 2, o conteúdo contido na linha 1 é o resultado das equações de paridade dessa linha. A linha 1, então, é resetada para um vetor de zeros através do multiplexador através da seleção da entrada *dina,* e o conteúdo da linha 1, que é o resultado da saída da porta lógica XOR, é enviado ao bloco OC. Na sequência, as linhas 2 e 3 são atualizadas e o conteúdo da linha 2 é enviada ao bloco OC. Ao atualizar as linhas 3 e 4, o conteúdo da linha 3 é enviado ao bloco OC. Essa dinâmica segue até as linhas p e 0 serem atualizadas, quando ambas são enviadas ao bloco OC e a RAM é completamente resetada com valores 0 e está pronta para a próxima iteração.

No bloco OC, é efetuada a contagem da quantidade de elementos não-zero contidos em cada vetor de 360 *bits* que chegam. Após realizada esta contagem em cada vetor, o resultado no formato de 1 bit é enviado ao registrador ONE REG. O bloco OC não necessita identificar a quantidade exata de elementos não-zero contidos em cada vetor, mas simplesmente precisa determinar se existe pelo menos um elemento não-zero contido no vetor, já que o decodificador não precisa saber o número exato de equações de paridade que não são satisfeitas, visto que uma equação errada já invalida a decodificação. Por isso, caso não existam elementos não-zero contidos em um vetor de 360 *bits*, a saída do bloco OC é 0, caso contrário, a saída do bloco OC é 1.

Após o processamento das linhas p e 1 da matriz $\boldsymbol{S}$, a saída do OC não é enviada ao registrador ONE REG, mas sim combinada com a saída do registrador ONE REG e armazenada no registrador ERR REG. Assim, quando todo o conteúdo de todas as linhas da matriz $\boldsymbol{S}$ é processado pelo bloco OC, o registrador ERR REG armazena o valor que indica se existe algum valor não-zero nas equações de paridade e o valor do registrador ERR REG é enviado pelo PCM, através da saída ERR, para o Controlador do decodificador. Se a saída ERR é 0, então todas as equações de paridade foram satisfeitas, caso contrário, a saída ERR é 1, indicando que uma ou mais equações de paridade não foram satisfeitas. Utilizando o controle de fluxo descrito anteriormente, a saída ERR é gerada 3 ciclos de *clock* após o último conjunto de 360 *bits* do último grupo de nodos de bit ser processado.

4.3.6 Arquitetura do *Buffer* LLR e do *Buffer* de Mensagem Decodificada

O *Buffer* LLR e o *Buffer* de Mensagem Decodificada são conversores serial/paralelo e paralelo/serial, respectivamente.

O *Buffer* LLR é utilizado para armazenar os valores LLR que chegam ao decodificador. Uma vez que são possíveis dois tamanhos de blocos distintos ($N = 64800$ *bits* e $N = 16200$ *bits)*, o *Buffer* LLR deve ser capaz de armazenar o maior dos dois casos, ou seja, 64800 valores. Além disso, uma vez que as FUs acessam os valores LLR em grupos de 360, já que elas operam simultaneamente, a RAM do *Buffer* LLR possui 360 valores de largura.

Os valores armazenados no *Buffer* LLR possuem tamanho igual a 7 *bits* no formato de complemento de dois a fim de facilitar as somas que acontecem dentro das FUs. Os valores LLR de 7 *bits* usam o formato 3.4 em ponto fixo, onde o *bit* mais significativo é o *bit* do sinal seguido por dois *bits* que formam a parte inteira do valor, e os quatro últimos *bits* formam a parte fracionária do valor.

O *Buffer* de Mensagem Decodificada é utilizado para armazenar a possível mensagem decodificada a cada iteração do decodificador. Se o PCM afirma que todas as equações de paridade são satisfeitas, então os valores

contidos no *Buffer* de Mensagem Decodificada são considerados a saída do decodificador e saem de forma serial.

Além disso, uma vez que os códigos LDPC no padrão DVB-S2 são sistemáticos, a saída do decodificador apresenta apenas a parte relativa à mensagem da palavra-código. Por isso, o *Buffer* de Mensagem Decodificada precisa apenas armazenar a parte da mensagem da palavra-código candidata para qualquer uma das taxas de código. O maior número de *bits* na parte relativa à mensagem de um código LDPC no padrão DVB-S2 é quando a taxa de código é $9/10$ com o *frame* normal, onde $K = 58320$ *bits*. Além disso, 360 *bits* são gerados simultaneamente pelas FUs e enviados ao *Buffer* de Mensagem Decodificada, então o tamanho da RAM do *Buffer* é de 162×360 *bits*.

5 Resultados

Neste capítulo são apresentados os resultados obtidos a partir da implementação em *software* e em *hardware* do *core* do codificador de canal LDPC em conformidade com o padrão DVB-S2.

A metodologia adotada para desenvolvimento do *core* em lógica programável consiste nos passos apresentados nos próximos parágrafos.

Os algoritmos de codificação e decodificação LDPC, apresentados no Capítulo 3, foram inicialmente implementados em linguagem de programação C, utilizando o *software* CodeBlocks. Simulações em linguagem C foram realizadas tanto para fins de prova de conceito quanto para serem utilizados como base para a implementação em VHDL.

Posteriormente, os algoritmos de codificação e decodificação LDPC foram descritos em VHDL, de acordo com a arquitetura apresentada no Capítulo 4, utilizando o *software* Xilinx ISE 14.2. Para a simulação VHDL em tempo real, utilizou-se o simulador iSim (ISIM, 2016). Os códigos VHDL foram sintetizados e utilizados para configurar a plataforma de *hardware* Xilinx Kintex 7.

Simulações em VHDL foram utilizadas para validar o funcionamento do código e para verificar a eficiência dos algoritmos implementados. Para atingir eficiência considerada satisfatória foram utilizadas três diferentes representações da função ψ: inicialmente foram utilizados 4 *bits*; na sequência, o número de *bits* foi aumentado para 5 e, posteriormente, para 6 *bits*.

Os resultados de síntese apresentam a utilização de recursos de *hardware* na plataforma de desenvolvimento escolhida. Com base nos recursos utilizados pela síntese do codificador e do decodificador, é calculada a máxima taxa de transferência dos sistemas, a partir da frequência máxima de operação obtida.

5.1 Resultados de Simulação

Esta seção apresenta os resultados de simulação em linguagem de programação C e em VHDL.

5.1.1 Simulação em C

As simulações em linguagem de programação C dos algoritmos de codificação e decodificação LDPC visam:

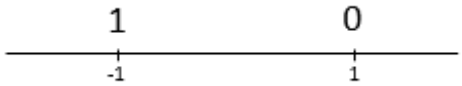
A. Validar o algoritmo do codificador e do decodificador;
B. Verificar o desempenho dos algoritmos quando submetidos a ruído branco Gaussiano.

A. Validação dos algoritmos de codificação e decodificação

As simulações para validação dos algoritmos de codificação e decodificação visam verificar tanto o processo de inserção dos *bits* de paridade como o processo de decodificação da palavra-código em mensagem. Para tal, são geradas mensagens aleatórias, as quais são codificadas nas respectivas palavras-código, moduladas, e aplicadas diretamente à entrada do decodificador. No processo de decodificação, observa-se a saída do decodificador ao final da primeira iteração: se todas as equações de verificação de paridade são satisfeitas, infere-se que tanto o algoritmo de codificação quanto o de decodificação estão corretos. Caso contrário, ou seja, se alguma das equações de verificação de paridade não é satisfeita ao final da primeira iteração, infere-se que existe algum erro de lógica no codificador ou no decodificador.

Para gerar aleatoriamente as mensagens, foi implementada a função **RandomInput**, a qual gera mensagens aleatórias de tamanho $\boldsymbol{K}$, dependendo da taxa de código. A função **BPSKModulation** implementa a modulação BPSK. Na modulação BPSK um bit de valor 0 é representado pelo valor inteiro 1 e o bit de valor 1 é representado pelo valor inteiro -1. *A* Figura 5.1 apresenta a constelação da modulação BPSK e a Tabela 5.1 apresenta a representação dos valores binários em simbolos.

Figura 5.1 - Constelação da Modulação BPSK



Fonte: o autor.

Tabela 5.1 - Mapeamento da Modulação BPSK

Bit	0	1
Símbolo	1	-1

Fonte: o autor.

Como exemplo de modulação BPSK, considere-se que, após a codificação LDPC a palavra código seja formada por uma sequência binária de 8 *bits*, por exemplo $\underline{c} = 10011100$. A função **BPSKModulation** vai transformar esta sequência binária na sequência $\underline{x_{BPSK}} = (-1,1,1,-1,-1,-1,1,1)$. Desta forma, as mensagens são codificadas em palavras-código, moduladas, utilizando modulação BPSK, e enviadas à entrada do decodificador, ou ao canal.

O decodificador LDPC espera receber na entrada valores de LLR. Nos testes de validação dos algoritmos, a saída do modulador BPSK é assumida como entrada LLR do decodificador.

A função **DecodVerification** verifica se a mensagem transmitida é corretamente decodificada após a primeira iteração. A função ainda contabiliza os erros em caso de erro na mensagem decodificada.

Testes foram realizados para todas as vinte e uma taxas de código previstas no padrão DVB-S2. Para cada taxa de código, foram geradas 100 mensagens aleatórias. O decodificador foi programado para realizar no máximo 50 iterações do algoritmo. Os resultados dos testes demostraram sucesso na decodificação das palavras-código, validando os algoritmos de codificação e decodificação implementados em linguagem de programação C. Para todos os casos, o sinal de erro do decodificador apresentou sinal lógico ‘0’, indicando que não há erros na decodificação da mensagem.

B. Verificação do desempenho dos algoritmos quando submetidos a ruído branco Gaussiano

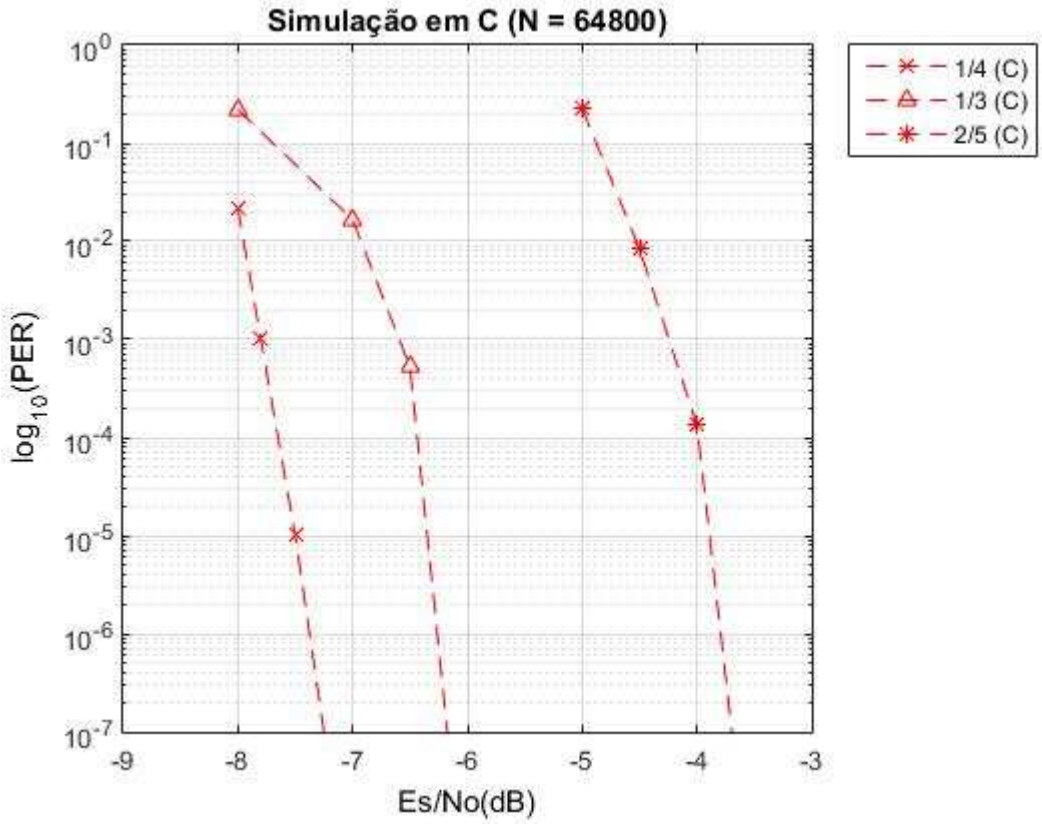
Para verificar o desempenho dos algoritmos quando submetidos a ruído branco Gaussiano, foi implementada a função **AWGN**, que adiciona ruído AWGN na sequência de palavras-código moduladas. A sequência de símbolos modulados, após a adição de ruído, é aplicada à entrada do decodificador e representa os valores de LLR.

A avaliação de desempenho é realizada através do gráfico da taxa de erro de pacote (PER) pela relação E_s/N_o. E_s/N_o é a relação entre a energia do símbolo e a densidade de potência espectral. Cada *frame* LDPC é dividido para formar múltiplos pacotes no formato MPEG (*Moving Picture Expert Group*), padrão de compressão e transmissão de áudio e vídeo (ETSI TR 102 376 V 1.1.1, 2005). De acordo com o padrão, cada pacote MPEG possui 188 bytes, ou 1504 *bits*. Por isso, a avaliação do desempenho é realizada através da PER.

Foram simulados *frames* de tamanho normal, com $N = 64800$ *bits*, para três taxas de código diferentes: $R = 1/4$, $R = 1/3$ e $R = 2/5$. Para cada uma das taxas avaliadas, foram codificadas 100 mensagens contendo informação e transmitidas em canais cujo ruído branco varia de $-8\ dB$ à $-3\ dB$.

A Figura 5.2 apresenta o gráfico da $PER \times E_s/N_o$ obtido.

Figura 5.2 - Resultado da simulação em C



Fonte: o autor.

A análise da Figura 5.2 permite constatar que, para a taxa $R = 1/4$, o sistema consegue decodificar corretamente todos os *bits* transmitidos a partir de um canal com ruído de $-7\ dB$. Para a taxa $R = 1/3$, o sistema consegue

decodificar corretamente todos os *bits* transmitidos a partir de um canal com ruído de $-6\,dB$. Para a taxa $R = 2/5$, o sistema consegue decodificar corretamente todos os bits transmitidos a partir de um canal com ruído de -3 dB.

5.1.2 Simulação em VHDL

As simulações em linguagem de descrição de *hardware* VHDL dos algoritmos de codificação e decodificação LDPC visam:

A. Validar o algoritmo do codificador e do decodificador;
B. Verificar o desempenho dos algoritmos quando submetidos a ruído branco Gaussiano;
C. Avaliar o compromisso entre o desempenho do LDPC e a ocupação de recursos da FPGA.

Os resultados de simulação em linguagem de programação C são utilizados como balizamento para os resultados de simulação em VHDL.

A. Validação dos algoritmos de codificação e decodificação

As simulações para validação dos algoritmos de codificação e decodificação visam verificar tanto o processo de inserção dos *bits* de paridade como o processo de decodificação da palavra-código em mensagem.

Quatro *testbenches* foram descritos para implementar o teste de validação dos algoritmos:

1. **RandomInput**: gera mensagens aleatórias de tamanho $\boldsymbol{K}$ *bits* e envia as mensagens para o codificador. O codificador, então, gera as palavras-código associadas às mensagens originais e as envia ao *testbench* **BPSKModulation**.
2. **BPSKModulation**: recebe as palavras-código do codificador e implementa a modulação BPSK para cada bit da palavra-código gerada. Para o teste de validação dos algoritmos de codificação e decodificação, a sequência de sinal modulado é aplicada diretamente à entrada do decodificador, ou seja,

considera-se que o sinal transmitido não é corrompido por nenhum tipo de interferência no canal de transmissão.

3. **BPSKtoFixedPoint**: converte a sequência modulada recebida através do canal em uma sequência de valores LLR, representada em ponto fixo no formato 3.4. Esta conversão é necessária uma vez que os valores aplicados à entrada do decodificador são valores LLR.
4. **DecodVerification**: desenvolvido em Matlab, este *testbench* verifica se a mensagem transmitida originalmente corresponde à mensagem decodificada pelo decodificador. Além disso, eventuais erros de decodificação são contabilizados.

Assim como nos testes de simulação em C, observa-se a saída do decodificador ao final da primeira iteração: se todas as equações de verificação de paridade são satisfeitas, infere-se que tanto o algoritmo de codificação quanto o de decodificação estão corretos.

Testes foram realizados para todas as vinte e uma taxas de código previstas no padrão DVB-S2. Para cada taxa de código avaliada, foram geradas 100 mensagens aleatórias. O decodificador foi programado para realizar no máximo 50 iterações do algoritmo. Para todos os casos, o sinal de erro do decodificador apresentou sinal lógico ‘0’, demostrando sucesso na decodificação das palavras-código, validando os algoritmos de codificação e decodificação implementados em linguagem de descrição de *hardware* VHDL.

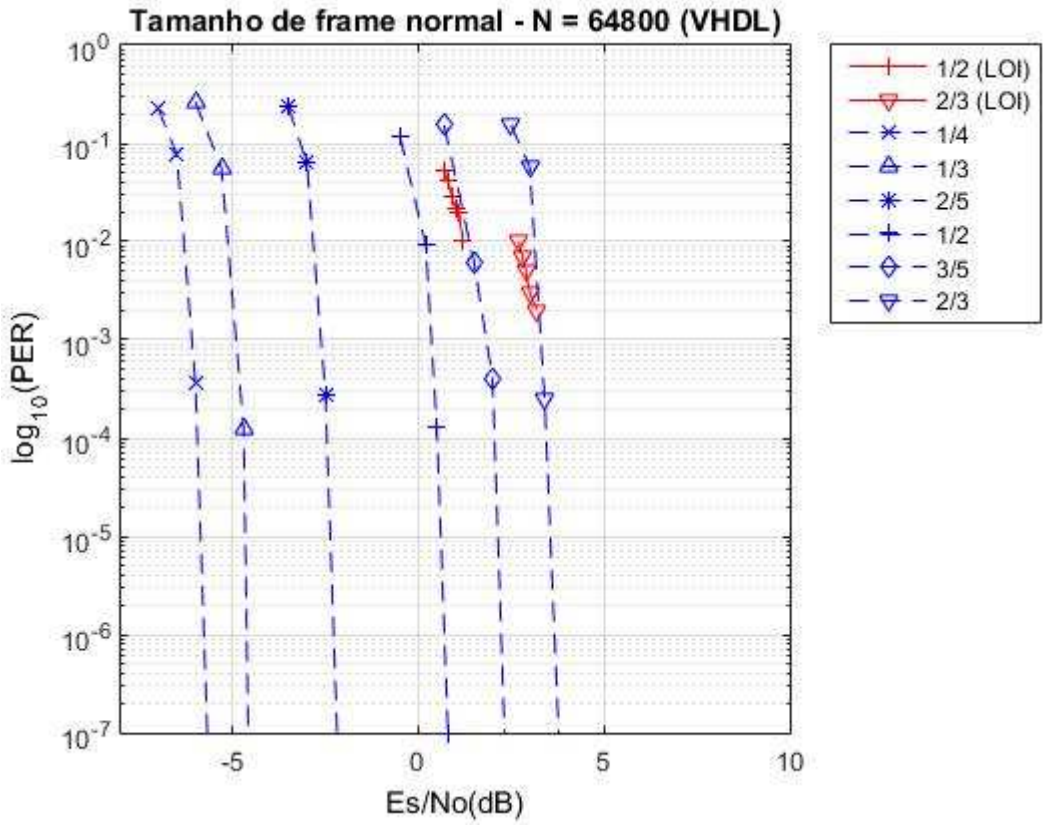
B. Verificação do desempenho dos algoritmos quando submetidos a ruído branco Gaussiano

Para verificar o desempenho do codec LDPC, quando o sinal transmitido é submetido a ruído branco Gaussiano no canal de comunicação, foi desenvolvido em Matlab o tesbench AWGN. Este testbench adiciona ruído AWGN na sequência que representa as palavras-código moduladas, obtidas através do BPSKModulation. A sequência com o ruído adicionado é enviada ao BPSKtoFixedPoint, o qual transforma esta sequência em valores LLR em ponto fixo, e a envia ao decodificador.

Foram avaliadas seis taxas de código para os *frames* normais ($1/4, 1/3, 2/5, 1/2, 3/5, 2/3$) e seis taxas de código para os *frames* pequenos ($1/5, 1/3, 2/5, 4/9, 3/5, 2/3$).

O padrão DVB-S2 (ETSI TR 102 376 V 1.1.1, 2005) define como alvo de desempenho a relação E_s/N_o para a performance *Quasi-Error-Free* (QEF) em cada modo de operação (tamanho do *frame* e taxa de código). O bloco de FEC definido no padrão DVB-S2 é composto pela concatenação de um codificador LDPC e um codificador BCH. No entanto, não faz parte do escopo deste trabalho o codificador BCH. Logo, é esperado que o desempenho individual do codificador LDPC implementado neste trabalho não atenda plenamente o alvo estabelecido pelo padrão. O padrão define a performance QEF como Taxa de Erro de Pacote (PER) de 10^{-7}.

Figura 5.3 - Gráfico PER x Es/No da Simulação em VHDL de algumas taxas de códigos dos frames normais



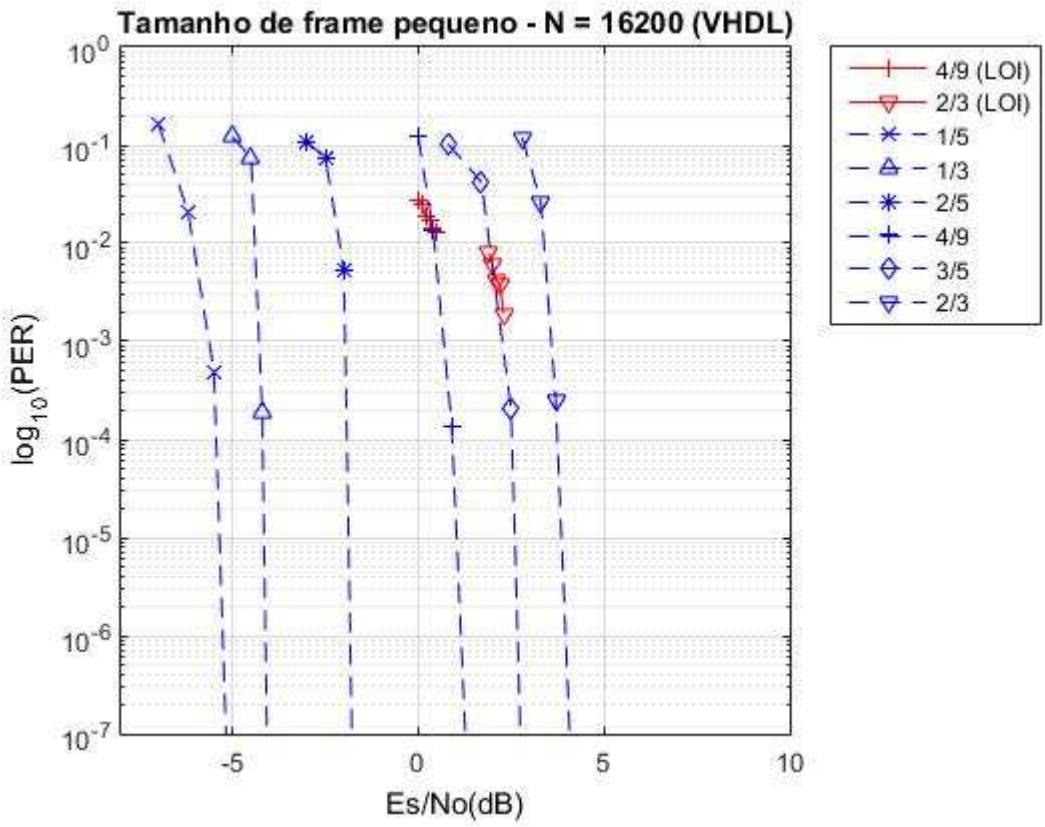
Fonte: o autor.

A Figura 5.3 mostra o gráfico $PER \times E_s/N_o$ das taxas de códigos testadas para os *frames* normais ($N = 64800$). As linhas tracejadas em azul na Figura 5.3

representam a performance de cada uma das taxas de código simuladas nesse trabalho e as linhas vermelhas do gráfico da Figura 5.3 representam as duas taxas de códigos ($R = 1/2$ e $R = 2/3$) simuladas por Loi em (LOI, 2010).

É possível observar que para a taxa $R = 1/2$, os resultados obtidos neste trabalho apresentam um desempenho melhor do que obtido no trabalho de Loi (em torno de 1,0 dB). Já para a taxa $R = 2/3$, no entanto, o resultado obtido nesse trabalho apresenta um desempenho levemente inferior ao obtido por Loi (em torno de 0,2 dB). Para as demais taxas, não é possível comparar o desempenho devido à inexistência de resultados de comparação em literatura.

Figura 5.4 - Gráfico PER x Es/No da Simulação em VHDL de algumas taxas de códigos dos frames pequenos

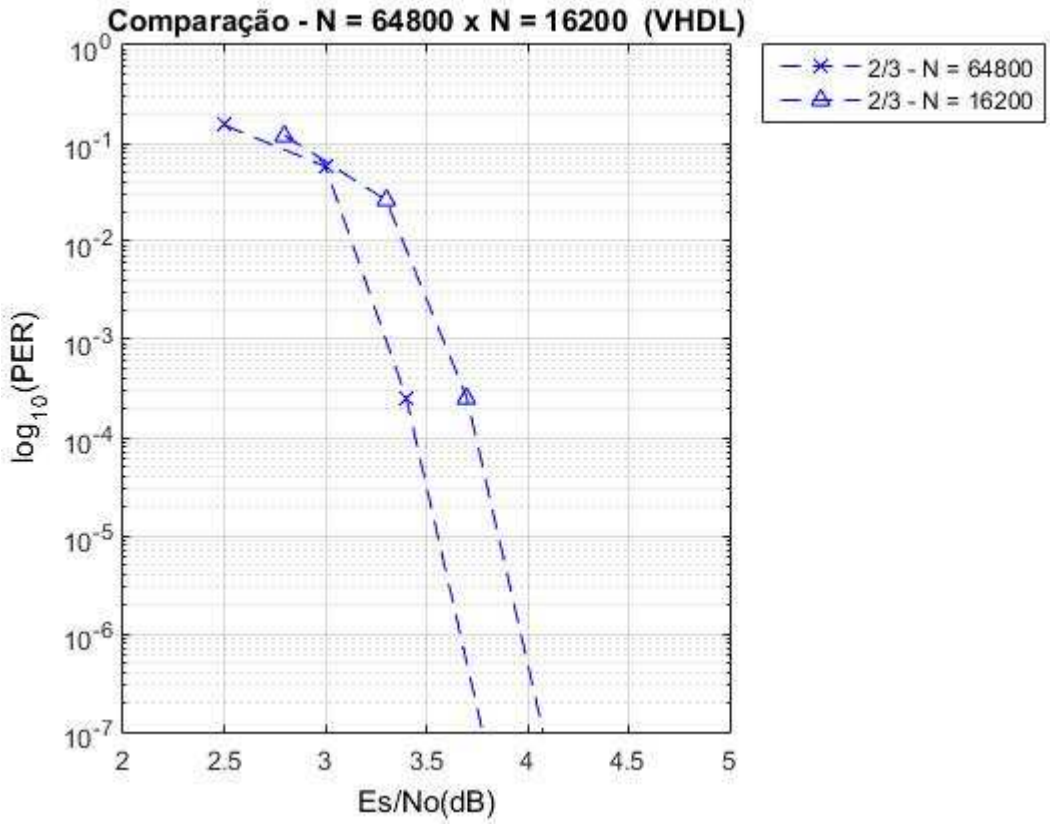


Fonte: o autor.

A Figura 5.4 mostra o gráfico $PER \times E_s/N_o$ das taxas de códigos testadas para os *frames* pequenos ($N = 16200$). As linhas tracejadas em azul representam a performance de cada uma das taxas de código simuladas nesse trabalho e as linhas vermelhas do gráfico representam as duas taxas de códigos ($R = 4/9$ e $R = 2/3$) simuladas por Loi em (LOI, 2010).

É possível observar que para a taxa $R = 1/2$, os resultados obtidos neste trabalho apresentam um desempenho semelhante aos resultados obtidos no trabalho de Loi. Já para a taxa $R = 2/3$, no entanto, os resultados obtidos neste trabalho apresentam um resultado inferior (em torno de 0,7 dB). Para as demais taxas, não é possível comparar o desempenho devido à inexistência de resultados de comparação em literatura.

Figura 5.5 - Comparação entre taxas de tamanhos de frames



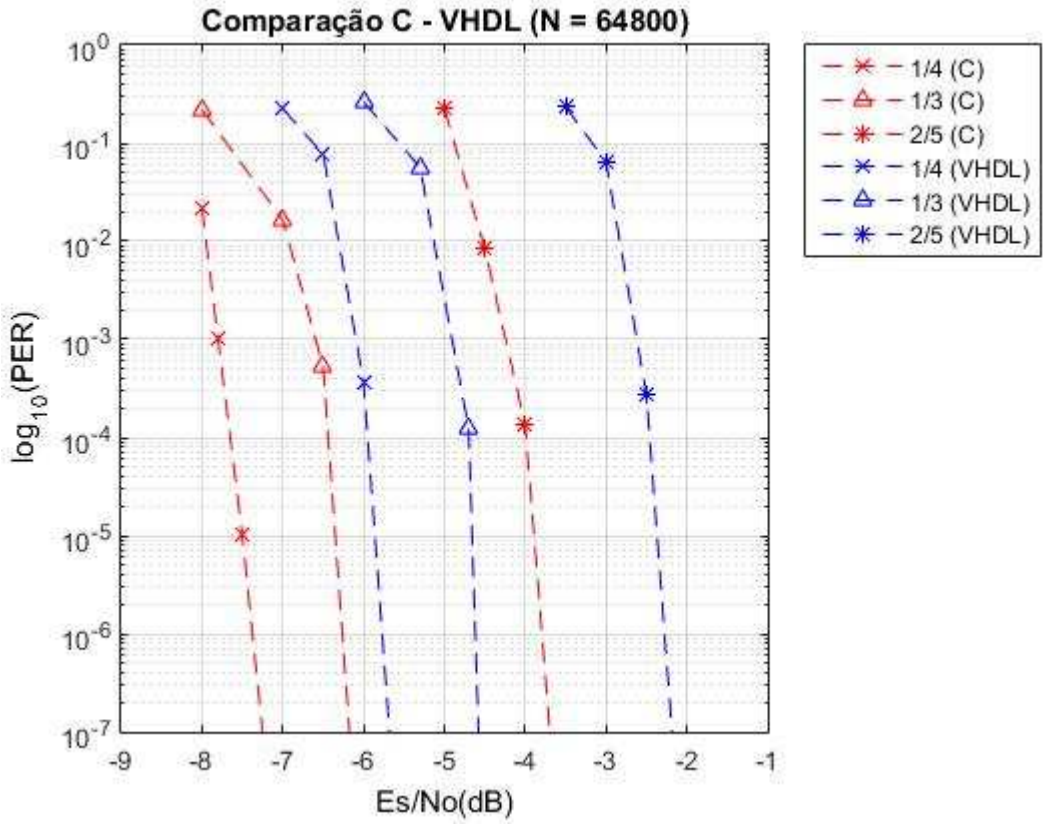
Fonte: o autor.

De acordo com o Eroz et al em (EROZ, SUN, & LEE, 2004), as taxas de código dos *frames* normais apresentam eficiência em torno de 0.25 dB e 0,3 dB superior comparado com as taxas de código dos *frames* pequenos. Xiao e Kim, em (XIAO & KIM, 2008), apresentam um código LDPC alternativo para o padrão DVB-S2. Nesse código, o autor apresenta uma matriz $\boldsymbol{H}$ de tamanho menor que as definidas pelo padrão DVB-S2 para, em um momento seguinte, aumentar o tamanho da mensagem, aumentando o desempenho do decodificador. Esses dois trabalhos mostram que quanto maior o tamanho da mensagem, melhor o desempenho do decodificador. Portanto, para uma mesma taxa, a taxa

equivalente aos *frames* normais ($N = 64800$ *bits*) deve apresentar um desempenho superior aos *frames* pequenos ($N = 16200$ *bits*). A Figura 5.5 apresenta a comparação para a taxa de $R = 2/3$ simulada com frames normais e pequenos. Como é possível observar, a taxa $R = 2/3$ correspondente ao *frame* normal ($N = 64800$) possui uma eficiência em torno de 0,3 dB superior a mesma taxa $R = 2/3$ correspondente ao *frame* pequeno ($N = 16200$), conforme esperado.

Por fim, a Figura 5.6 compara os resultados de simulação em C com os resultados de simulação em VHDL para as taxas $R = 1/4$, $R = 1/3$ e $R = 2/5$, considerando *frames* normais. Os resultados de simulação em VHDL consideram a representação de 4 *bits* para a implementação da função ψ. Cabe salientar que, a definição da representação da função ψ em 4 *bits* se deve ao fato de que, os resultados apresentados por Loi, utilizados como comparação neste trabalho, utilizam esta mesma representação.

Figura 5.6 - Comparação da simulação em C com a simulação em VHDL



Fonte: o autor.

Como é possível observar na Figura 5.6, a simulação em C apresenta resultados melhores porque utiliza ponto flutuante nos cálculos da decodificação enquanto a simulação em VHDL utiliza ponto fixo.

Devido às diferenças de aproximadamente 1,7 dB entre os resultados obtidos na simulação em C e os resultados obtidos na simulação em VHDL, a arquitetura da função ψ, apresentada na Seção 4.3.4.1, foi modificada, aumentando-se a sua representação. Os resultados provenientes desta alteração são apresentação na próxima subseção.

C. Avaliar o compromisso entre o desempenho do LDPC e a ocupação de recursos da FPGA.

A avaliação do compromisso entre o desempenho do *core* LDPC desenvolvido e a ocupação dos recursos da FPGA foi realizada através do aumento da quantidade de *bits* para representação da função ψ. Três representações da função ψ foram implementadas e avaliadas, sendo elas com 4 *bits* (resultados apresentados na subseção anterior), 5 *bits* e 6 *bits*. A implementação partiu de 4 *bits* por ser a quantidade de *bits* utilizada por Loi, implementação esta utilizada para fins de comparação de resultados.

Tabela 5.2 - LUT da função ψ representada por 5 bits

Entrada	Saída	Entrada	Saída
00000	11.1111	01101	00.1111
00001	11.0100	01110	00.1110
00010	10.1011	01111	00.1101
00011	10.0101	10000	00.1011
00100	10.0101	10001	00.1000
00101	10.0001	10010	00.0110
00110	01.1101	10011	00.0101
00111	01.1000	10100	00.0100
01000	01.0110	10101	00.0011
01001	01.0100	10110	00.0010
01010	01.0010	10111	00.0001
01011	01.0001	11000	00.0000
01100	01.0000	-	-

Fonte: o autor.

A Tabela 5.2 apresenta a LUT da função ψ para a representação utilizando 5 *bits*.

Por causa da adição de 1 bit na representação original da função ψ, a arquitetura do bloco COMP, também mostrada na Seção 4.3.2.1 na Tabela 4.9, foi modificada, com o intuito de ser equivalente à LUT da função ψ. A Tabela 5.3 mostra a nova LUT da função de Compressão para a representação da função ψ utilizando 5 *bits*.

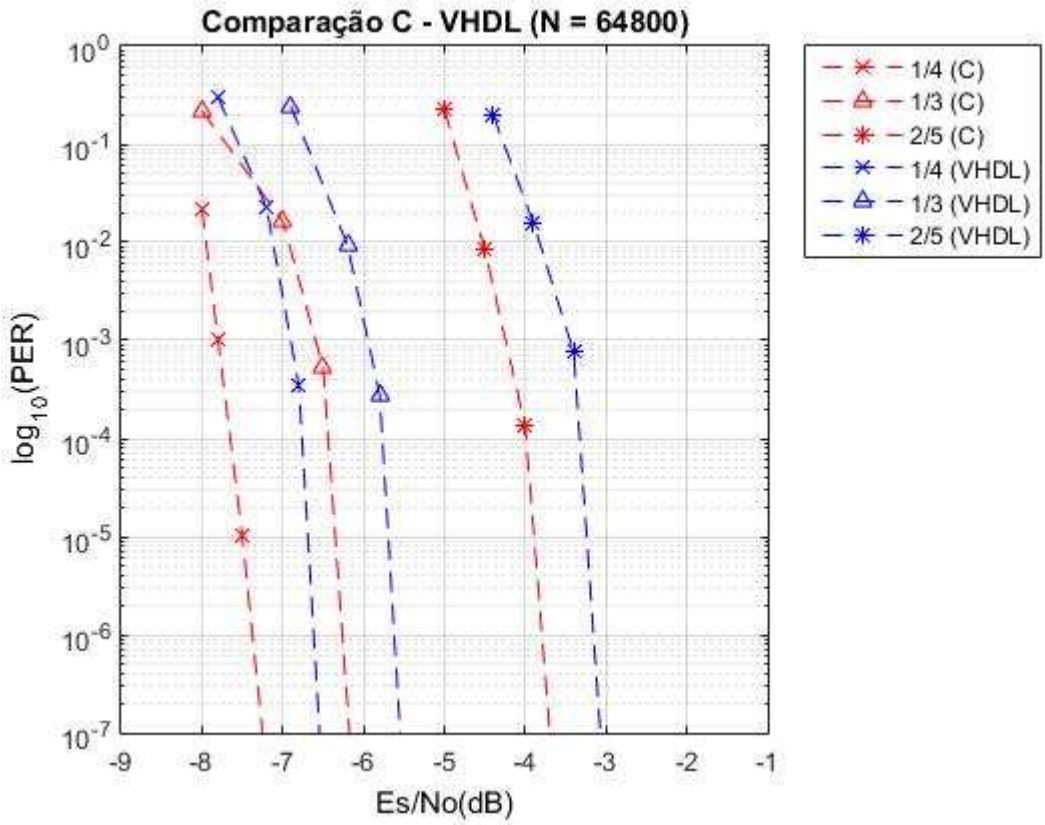
Tabela 5.3 - LUT da Função de Compressão representada por 5 bits

Entrada	Saída	Entrada	Saída
00.0000	00000	00.1101	01101
00.0001	00001	00.1110	01110
00.0010	00010	00.1111	01111
00.0011	00011	01.00xx	10000
00.0100	00100	01.01xx	10001
00.0101	00101	01.10xx	10010
00.0110	00110	01.11xx	10011
00.0111	00111	10.00xx	10100
00.1000	01000	10.01xx	10101
00.1001	01001	10.1xxx	10110
00.1010	01010	11.0xxx	10111
00.1011	01011	11.1xxx	11000
00.1100	01100	-	-

Fonte: o autor.

A Figura 5.7 apresenta a comparação entre os resultados obtidos na simulação em C e os resultados obtidos em VHDL, utilizando a representação da função ψ em uma resolução de 5 *bits*, conforme mostrado na Tabela 5.2 e na Tabela 5.3, para as taxas $R = 1/4$, $R = 1/3$ e $R = 2/5$, considerando tamanho de *frame* normal ($N = 64800$ *bits*). É possível observar que, utilizando 5 *bits* para quantizar a função ψ ao invés dos 4 *bits* originalmente proposto, os resultados da simulação em VHDL estão mais próximos aos resultados obtidos na simulação em C. A diferença entre as curvas foi reduzida em aproximadamente 1 dB.

Figura 5.7 - Comparação da simulação em C com a simulação em VHDL para a função ψ representada por 5 bits



Fonte: o autor.

Com o objetivo de reduzir ainda mais esta diferença de 1 dB, melhorando consequentemente o desempenho do FEC, uma nova representação da função ψ foi implementada. Mais um bit foi adicionado à representação, fazendo com que a resolução seja de 6 *bits*. A Tabela 5.4 apresenta a nova LUT da função ψ.

Tabela 5.4 - LUT da função ψ representada por 6 bits

Entrada	Saída	Entrada	Saída
000000	011.1111	100000	000.1100
000001	011.1111	100001	000.1011
000010	011.0111	100010	000.1001
000011	011.0001	100011	000.1000
000100	010.1100	100100	000.0111
000101	011.1001	100101	000.0110
000110	010.0110	100110	000.0110
000111	010.0011	100111	000.0101
001000	010.0001	101000	000.0100
001001	001.1111	101001	000.0100

001010	001.1110	101010	000.0011
001011	001.1100	101011	000.0011
001100	001.1011	101100	000.0011
001101	001.1010	101101	000.0010
001110	001.1001	101110	000.0010
001111	001.1000	101111	000.0010
010000	001.0111	110000	000.0010
010001	001.0110	110001	000.0001
010010	001.0101	110010	000.0001
010011	001.0100	110011	000.0001
010100	001.0011	110100	000.0001
010101	001.0010	110101	000.0001
010110	001.0010	110110	000.0001
010111	001.0001	110111	000.0001
011000	001.0000	111000	000.0001
011001	001.0000	111001	000.0001
011010	000.1111	111010	000.0000
011011	000.1111	111011	000.0000
011100	000.1110	111100	000.0000
011101	000.1110	111101	000.0000
011110	000.1101	111110	000.0000
011111	000.1101	111111	000.0000

Fonte: o autor.

Por causa da adição de 2 *bits* com relação à representação original da função ψ (de 4 para 6 *bits*), a arquitetura do bloco COMP, apresentado na Seção 4.3.2.1 na Tabela 4.9, foi modificada, afim de se tornar equivalente à LUT da função ψ. A Tabela 5.5 mostra a nova LUT da função de Compressão.

Tabela 5.5 - LUT da Função de Compressão representada por 6 bits

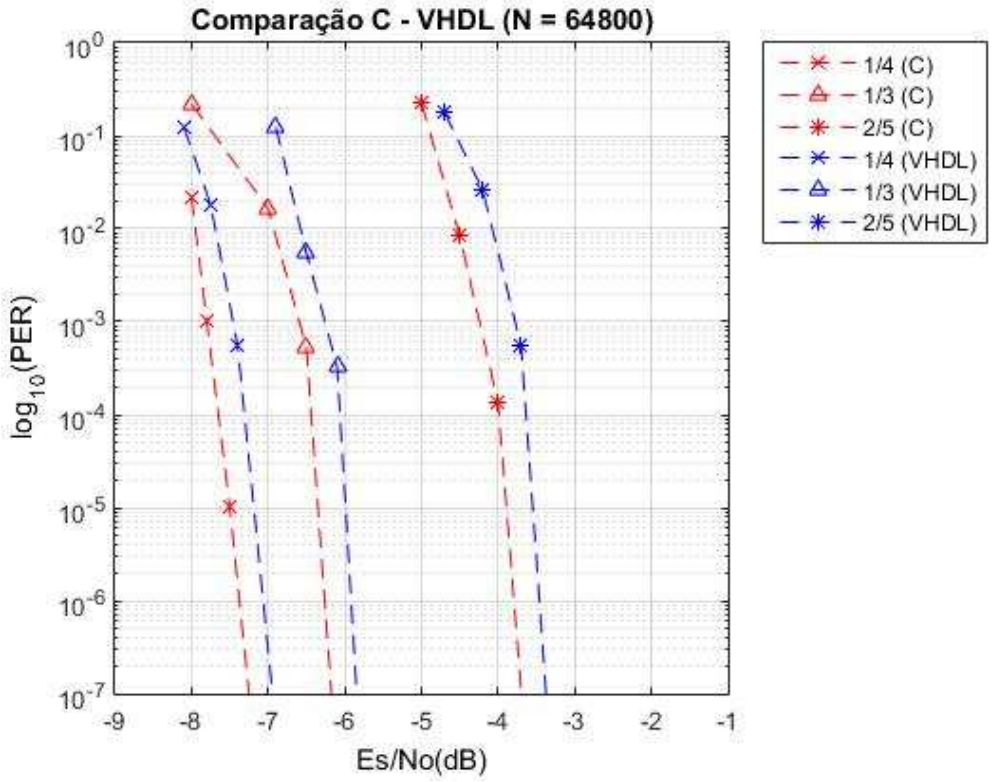
Entrada	Saída	Entrada	Saída
000.00000	000000	001.000xx	100000
000.00001	000001	001.001xx	100001
000.00010	000010	001.010xx	100010
000.00011	000011	001.011xx	100011
000.00100	000100	001.100xx	100100
000.00101	000101	001.101xx	100101
000.00110	000110	001.110xx	100110
000.00111	000111	001.111xx	100111

000.01000	001000	010.000xx	101000
000.01001	001001	010.001xx	101001
000.01010	001010	010.010xx	101010
000.01011	001011	010.011xx	101011
000.01100	001100	010.100xx	101100
000.01101	001101	010.101xx	101101
000.01110	001110	010.110xx	101110
000.01111	001111	010.111xx	101111
000.10000	010000	011.000xx	110000
000.10001	010001	011.001xx	110001
000.10010	010010	011.010xx	110010
000.10011	010011	011.011xx	110011
000.10100	010100	011.100xx	110100
000.10101	010101	011.101xx	110101
000.10110	010110	011.110xx	110110
000.10111	010111	011.111xx	110111
000.11000	011000	100.000xx	111000
000.11001	011001	100.001xx	111001
000.11010	011010	100.010xx	111010
000.11011	011011	100.011xx	111011
000.11100	011100	100.100xx	111100
000.11101	011101	100.101xx	111101
000.11110	011110	100.110xx	111110
000.11111	011111	100.111xx	111111

Fonte: o autor.

A Figura 5.8 apresenta a comparação entre os resultados obtidos na simulação em C e os resultados obtidos em VHDL, utilizando a representação da função ψ em uma resolução de 6 *bits*, conforme mostrado na Tabela 5.4 e na Tabela 5.5, para as taxas $R = 1/4$, $R = 1/3$ e $R = 2/5$, considerando tamanho de *frame* normal ($N = 64800$ *bits*). É possível observar que, utilizando 6 *bits* de resolução para a função ψ ao invés dos 4 *bits* originalmente propostos e dos 5 *bits* anteriores, é possível obter um resultado em VHDL bastante próximo ao obtido pela simulação em C. Este resultado é considerado plenamente satisfatório, dado que a implementação em C utiliza ponto flutuante e a implementação em VHDL representação em ponto fixo.

Figura 5.8 - Comparação da simulação em C com a simulação em VHDL para a função ψ representada por 6 bits

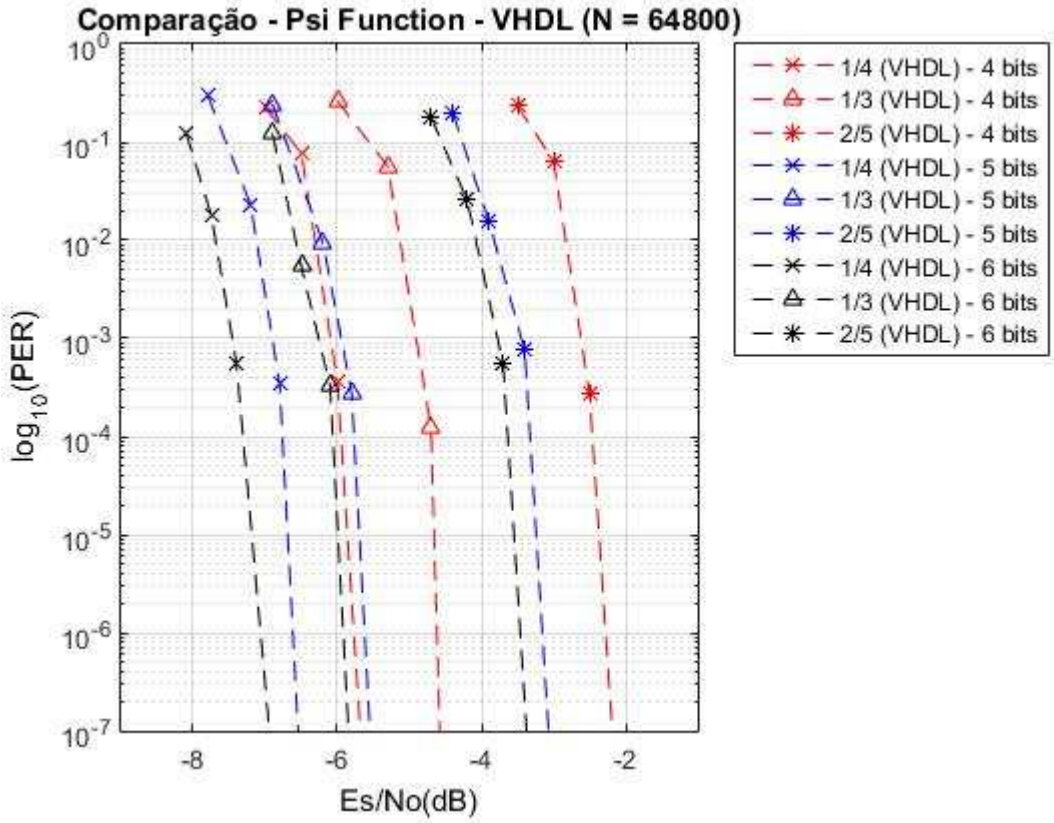


Fonte: o autor.

A Figura 5.9 apresenta a comparação entre os resultados obtidos para as três implementações da função ψ, para as taxas avaliadas. É possível observar o aumento no desempenho do decodificador à medida que é aumentado o tamanho da representação da função ψ, conforme esperado. Também é possível notar que a melhora no desempenho de 4 para 5 *bits* é consideravelmente superior à melhora obtida na alteração de 5 para 6 *bits*. Considerando os resultados de implementação com resolução de 4 e 6 *bits*, observa-se que, para a taxa de $R = 1/4$, obteve-se um desempenho superior em aproximadamente 1,3 dB; para a taxa de $R = 1/3$, obteve-se um desempenho superior em aproximadamente 1,2 dB e para a taxa de $R = 2/5$, obteve-se um desempenho superior em aproximadamente 1,2 dB.

Cabe salientar que esta melhora de desempenho se dá ao custo de uma maior ocupação dos recursos de *hardware* da FPGA, como será apresentado na próxima Seção.

Figura 5.9 - Comparação entre as diferentes resoluções da função ψ

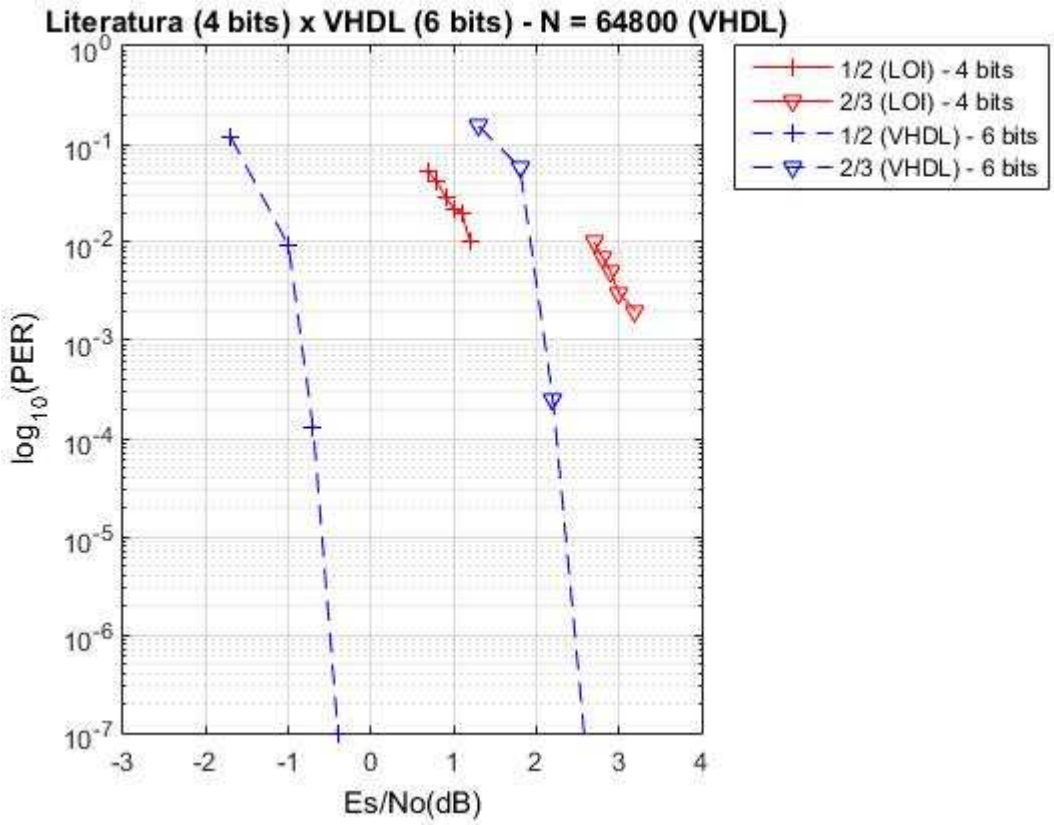


Fonte: o autor.

Como a melhora do desempenho de 5 para 6 *bits* é de apenas cerca 0,3 dB, um possível aumento na representação da função ψ para 7 *bits*, acarretaria em uma pequena melhora no desempenho e um aumento significativo na utilização dos recursos de *hardware*. Além disso, em Telecomunicações, uma melhora abaixo de 0,5 dB no desempenho de qualquer sistema não justifica um aumento na utilização dos recursos de *hardware*.

A Figura 5.10 apresenta a comparação entre o resultado apresentado por Loi, utilizando 4 *bits* de resolução para a função ψ, com os resultados obtidos neste trabalho, utilizando uma resolução de 6 *bits* para a função ψ. Como é possível observar, a resolução de 6 *bits* apresenta um aumento considerável no desempenho do decodificador. Para a taxa de $R = 1/2$, obteve-se um desempenho superior em aproximadamente 2,2 dB e para a taxa de $R = 2/3$, obteve-se um desempenho superior em aproximadamente 1,2 dB.

Figura 5.10 - Comparação entre a Literatura de 4 bits *com o teste de 6 bits*



Fonte: o autor.

É importante salientar que a exigência mínima dos requisitos pelo padrão DVB-S2 inclui um *Encoder* BCH e um *Decoder* BCH juntamente com o LDPC, os quais não foram implementados nesse trabalho, o que aumenta a performance do bloco FEC. Também é importante salientar que o padrão utiliza modulações menos robustas que a BPSK, fator que piora a performance dos códigos.

5.2 Resultado de Síntese em Dispositivo Xilinx Kintex 7

O codificador LDPC DVB-S2 e o decodificador DVB-S2 foram sintetizados utilizando o pacote de *software* Xilinx ISE 14.2 para o uso na FPGA Xilinx Kintex 7 XC7K160. Não foram encontrados na literatura trabalhos semelhantes que tenham utilizado a mesma FPGA no desenvolvimento, razão pela qual os resultados de síntese foram comparados com resultados de trabalhos que utilizam outras FPGAs. A Tabela 5.6 apresenta os recursos mais relevantes para a implementação do codificador LDPC desenvolvido neste

trabalho, disponíveis na FPGA Kintex. Explicações mais detalhadas sobre a FPGA são apresentadas na Seção 4.1.

Tabela 5.6 - Recursos da FPGA Kintex 7

Células lógicas	Blocos Lógicos Configuráveis (CLBs)		*Slices* de DSP	Blocos de Blocos de RAM		
	Slices	Max RAM Distribuída (Kb)		18 Kb	36 Kb	Max (Kb)
162,240	25,350	2,188	600	650	325	11,700

Fonte: o autor.

A Tabela 5.7 apresenta os recursos disponíveis na FPGA Virtex 6 XC6VLX240T e a Tabela 5.8 apresenta os recursos disponíveis na FPGA Virtex 2 Pro XC2VP30. Trabalhos desenvolvidos nestas FPGAs são utilizados para fins de comparação com os resultados obtidos neste trabalho.

Tabela 5.7 - Recursos da FPGA Virtex 6 XC6VLX240T

Células lógicas	Blocos Lógicos Configuráveis (CLBs)		*Slices* de DSP	Blocos de Blocos de RAM		
	Slices	Max RAM Distribuída (Kb)		18 Kb	36 Kb	Max (Kb)
241,152	37,680	3,650	768	832	416	14,976

Fonte: o autor.

Tabela 5.8 - Recursos da FPGA Virtex 2 Pro XC2VP30

Células lógicas	Blocos Lógicos Configuráveis (CLBs)		*Slices* de DSP	Blocos de Blocos de RAM		
	Slices	Max RAM Distribuída (Kb)		18 Kb	36 Kb	Max (Kb)
30,816	13,696	428	-	136	-	2,448

Fonte: o autor.

5.2.1 Resultado de Síntese do *Encoder*

A arquitetura proposta, como dito anteriormente, foi sintetizada na FPGA Xilinx XC7K160. Como comparação foi utilizado o trabalho de (GOMES, GONÇALVES, SILVA, FALCAO, & MAIA, 2007) sintetizado na FPGA Xilinx XC2VP30. A Tabela 5.5 apresenta o resultado da síntese das duas FPGAs. (GOMES, GONÇALVES, SILVA, FALCAO, & MAIA, 2007) apresenta os resultados de síntese em termos de percentual de utilização dos recursos. Para fins de comparação, estes percentuais foram convertidos em valores quantitativos aproximados, de acordo com os percentuais apresentados, e com os dados de recursos disponíveis no *datasheet* da FPGA (apresentados na Tabela 5.8).

Tabela 5.9 - Comparação dos resultados de síntese

Recursos utilizados	XC2VP30 - GOMES -	XC7K160 - Este trabalho -
Número de *Slices de Flip-Flops*	3%	1%
	aproximadamente 1000 *slices*	1839 *slices*
Número de BRAMs	20%	6%
	28 blocos	22 blocos
Frequência Máxima	131,7 MHz	289,918 MHz

Fonte: o autor.

De acordo com a Tabela 5.9, este trabalho utiliza um número maior de *slices* de *Flip-Flops* do que ao utilizado por (GOMES, GONÇALVES, SILVA, FALCAO, & MAIA, 2007). Por outro lado, o número de blocos de memória BRAM utilizados neste trabalho é menor do que o utilizado por Gomes et. Al.

Analisando o esquemático da arquitetura proposto por Gomes et. Al, é possível observar que ele diz utilizar apenas uma porta lógica XOR para a Equação 3.12, para fazer a operação simultânea da operação XOR para 360 *bits*, enquanto este trabalho utilizou $M = 360$ portas lógicas XOR para representar a mesma Equação 3.12, utilizando uma porta lógica para a operação XOR de 2 *bits*. A utilização das $M = 360$ portas lógicas foi devido à impossibilidade do programa de desenvolvimento de utilizar apenas uma porta lógica.

Com base na Tabela 5.9, é possível operar o sistema a uma frequência máxima de *clock* de $f_{clk\_max} = 289{,}918MHz$. De fato, para um dado ciclo de *clock* ($(f_{operação} > f_{clk\_max})$, a taxa de transferência atingível é uma função do parâmetro do código W, o qual representa o número de elemento da matriz $\boldsymbol{H}$ na forma compacta, dado pelo Apêndice A deste trabalho e por p. Na arquitetura utilizada neste trabalho, o Algoritmo 4.1 leva dois ciclos de *clock* para cada iteração do primeiro laço de '*para'* e outros dois ciclos de *clock* para cada iteração do segundo laço de '*para*'. Desta forma, a taxa de transferência do codificador é dada por

$$Taxa_{tranferencia} = \frac{tamanho_{frame} \times f_{operação}}{(W + p) \times 2} \tag{5.1}$$

A Tabela 5.10 mostra a taxa de transferência do *Encoder* atingível para cada uma das vinte e uma taxas de código dos códigos DVB-S2, quando operando na maior frequência de *clock* da arquitetura, ou seja, $f_{clk\_max} = 289{,}918MHz$.

Tabela 5.10 - Máxima Taxa de Transferência do Codificador

Tamanho de *frame* N = 64800				**Tamanho de *frame* N = 16200**			
Taxa	**p**	**W**	**Taxa Transferência (Gb/s)**	**Taxa**	**p**	**W**	**Taxa Transferência (Gb/s)**
1/4	135	270	23,193	1/5	36	63	23,720
1/3	120	360	19,569	1/3	30	90	19,569
2/5	108	432	17,395	2/5	27	108	17,395
1/2	90	450	17,395	4/9	25	85	21,348
3/5	72	648	13,046	3/5	18	162	13,046
2/3	60	480	17,395	2/3	15	120	17,395
3/4	45	540	16,057	11/15	12	108	19,569
4/5	36	576	15,348	7/9	10	105	20,420
5/6	30	600	14,910	37/45	8	121	18,204
8/9	20	500	18,064	8/9	5	125	18,064
9/10	18	504	17,994	---	---	---	---

Fonte: o autor.

5.2.2 Resultado de Síntese do *Decoder*

A arquitetura proposta neste trabalho, utilizando 4, 5 e 6 *bits* de resolução para a função ψ, foi sintetizada na FPGA Xilinx XC7K160. Como comparação, foi utilizado o trabalho de (LOI, 2010) que utiliza uma FPGA Xilinx XC6VLX240T. Assim como (GOMES, GONÇALVES, SILVA, FALCAO, & MAIA, 2007), (LOI, 2010) apresenta os resultados de síntese em termos de percentual de utilização dos recursos. Para fins de comparação, estes percentuais foram convertidos em valores quantitativos aproximados, de acordo com os percentuais apresentados, e com os dados de recursos disponíveis no *datasheet* da FPGA (apresentados na Tabela 5.7). A Tabela 5.11 mostra a comparação dos resultados de síntese das duas FPGAs, considerando as três diferentes resoluções da função ψ implementadas neste trabalho.

Tabela 5.11 - Comparação dos resultados de síntese

Recursos utilizados	**XC6VLX240T** - LOI -	**XC7K160** - Este trabalho -	**XC7K160** - Este trabalho -	**XC7K160** - Este trabalho -
Resolução Função ψ	4 *bits*	4 *bits*	5 *bits*	6 *bits*
Número de *Slices* de *Flip-Flops*	17%	22,3%	22,3%	30,2%
	aprox. 51244 *slices*	45183 *slices*	45183 *slices*	61189 *slices*
Número de *Slices* LUT	60%	78,4%	78,4%	83,7%
	aprox. 90432 *slices*	79524 *slices*	79524 *slices*	84899 *slices*
Número de BRAMs	31%	51,4%	62,3%	79,7%
	aprox. 258 blocos	334 blocos	405 blocos	518 blocos
Frequência Máxima	214,5 MHz	100,058 MHz	100,058 MHz	100,058 MHz

Fonte: o autor.

Comparando a utilização de recursos da arquitetura proposta neste trabalho utilizando a função ψ com resolução de 4 *bits* com a de Loi, é possível constatar que a arquitetura implementada neste trabalho requer a utilização de menos *slices* de *Flip-Flop* e menos *slices* de LUT, mas apresenta um número

maior de blocos de BRAM. Os resultados mostram que, utilizando um número menor de *slices* de *Flip-Flops*, um número menor de *slices* de LUT e um número maior de blocos de RAM, o sistema apresenta um desempenho semelhante e até, em alguns casos, melhor que o decodificador comparado, como apresentado na Seção 5.1.2.

Não é possível identificar os fatores que levam à utilização de um menor número de *slices* de *Flip-Flops* e de *slices* LUT, uma vez que algumas informações sobre a arquitetura do decodificador implementado por Loi não estão disponívies. É possível que o trabalho de comparação tenha utilizado mais *bits* de representação de valores para a realização dos cálculos dentro das Unidades de Cálculo (FUs), principal módulo do sistema, que utiliza o maior número de recursos da FPGA. O aumento da utilização de blocos de BRAMs é devido ao aumento de 1 *bit* na representação dos valores LLR recebidos pelo decodificador. Os valores LLR são representados com 7 *bits* em ponto fixo no formato 3.4, ao invés de 6 *bits* em ponto fixo no formato 3.3 como no trabalho comparado. Neste trabalho optou-se pelo uso do formato 3.4 com o objetivo de obter melhores resultados de desempenho do bloco FEC. O uso do formato 3.4 acarretou no aumento da utilização de memória BRAM, já que é necessário aumentar um bit por célula no *Buffer* LLR.

Também foram sintetizadas as arquiteturas apresentando as diferentes resoluções da função ψ. Como é possível observar na Tabela 5.11, para a síntese da arquitetura utilizando uma resolução de 5 *bits* para a função ψ, há apenas um aumento da utilização de blocos de memória BRAM. Esse aumento é devido ao aumento de 1 *bit* em cada uma das células de armazenamento da memória RAM, o que acarreta no aumento de memória para armazenamento da LUT que representa a função ψ e da LUT da função de Compressão.

Já na síntese para a resolução da função ψ de 6 *bits*, além do aumento da utilização de blocos de memória BRAM, devido ao aumento do tamanho da LUT da função ψ e do aumento da LUT da função de compressão, há um aumento no número de *slices* de *Flip-Flop* e também no número de *slices* LUT. Esse acréscimo é devido a um aumento necessário na representação da parte fracionária do valor em ponto fixo, como é possível observar na Tabela 5.5, onde a função de Compressão utiliza 5 *bits* na representação da parte fracionária em

ponto fixo. Além disso, é necessário aumentar a representação dos valores LLR em ponto fixo de 3.4 para 3.5, de modo a tornar compatível com a função de Compressão, conforme ilustra a Tabela 5.5. Esse aumento na representação também aumenta o número de blocos de memória BRAM, já que os valores LLR também são armazenados em uma memória RAM dentro do sistema.

Através dos resultados de desempenho apresentados na Figura 5.9 e de síntese apresentados na Tabela 5.11, observa-se que com aumento da representação da função ψ, de 4 para 5 e 6 *bits*, obteve-se um desempenho dos algoritmos significativamente superior. Com a representação de 6 *bits*, os resultados apresentados pela implementação em VHDL ficaram em torno de 0,4 dB inferiores aos resultados obtidos em linguagem C, utilizando ponto flutuante, resultado considerado plenamente satisfatório. No entanto, o aumento da performance está condicionado ao aumento do utilização dos recursos de *hardware*. Os resultados apresentados neste capítulo demostram o compromisso entre desempenho e ocupação de *hardware*. A implementação mais adequada depende dos recursos de *hardware* disponíveis e do desempenho mínimo desejado do FEC.

A taxa de transferência do *Decoder* é dada pelo tamanho do *frame*, N, dividido pelo número máximo de ciclos de *clock* gastos para decodificar um *frame* multiplicado pela máxima frequência de operação. O número máximo de ciclos de *clock* da decodificação é dado pelo número máximo de iterações multiplicado pelo número de ciclos de *clock* por iteração. O número de ciclos de *clock* por iteração é igual ao número de ciclos de *clock* para processar a Atualização dos Nodos de Validação mais o número de ciclos de *clock* para processar a Atualização dos Nodos de Bit mais o atraso das FUs mais o atraso do PCM. O número de ciclos de *clock* para processar o passo de Atualização dos Nodos de Validação e o passo de Atualização dos Nodos de Bit é igual ao número de linhas na *top* RAM e na *bottom* RAM, que é dado por $pq + 2p$. O atraso das FUs é $w_j + 3$, onde w_j é o grau de nodo de bit diferente de 3, e o atraso do PCM é 3 conforme visto na Seção 4.3.5. O número de ciclos de *clock* para a inserção dos valores LLR e para a saída do sinal decodificado não são considerados. A taxa de transferência, então, é dada por

$$Taxa_{tranferencia} = \frac{tamanho_{frame} \times f_{operação}}{(2 \times (pq + 2p) + w_j + 6) \times max_{iter}} \quad (5.2)$$

A Tabela 5.12 mostra a taxa de transferência atingível para casa uma das vinte e uma taxas de código dos códigos DVB-S2 quando operando na maior frequência de *clock* da arquitetura $f_{clk\_max} = 100{,}058\ MHz$.

Tabela 5.12 - Máxima Taxa de Transferência do Decodificador

Tamanho de *frame* N = 64800					**Tamanho de *frame* N = 16200**				
Taxa	**p**	**q**	w_j	**Taxa Transfe-rência (Mb/s)**	**Taxa**	**p**	**q**	w_j	**Taxa Transfe-rência (Gb/s)**
1/4	135	2	12	393,670	1/5	36	1.75	12	375,217
1/3	120	3	12	354,885	1/3	30	3	12	339,819
2/5	108	4	12	328,958	2/5	27	4	12	315,972
1/2	90	5	8	339,286	4/9	25	3.4	8	380,502
3/5	72	9	12	269,819	3/5	18	9	12	261,020
2/3	60	8	13	354,594	2/3	15	8	13	338,754
3/4	45	12	12	338,224	11/15	12	9	12	383,200
4/5	36	16	11	329,208	7/9	10	10.5	3	417,230
5/6	30	20	13	322,815	37/45	8	15.125	13	368,814
8/9	20	25	4	396,560	8/9	5	25	4	385,938
9/10	18	28	4	396,560	---	---	---	---	---

Fonte: o autor.

6 Conclusões e Trabalhos Futuros

Esta dissertação de mestrado foi viabilizada através da implementação de uma Bolsa na Modalidade Mestrado, concedida através do ao Edital MCTI/CNPq Nº 20/2013 - PNM (GM e GD), relativo ao Programa Nacional de Microeletrônica. O projeto contemplado, intitulado "*Desenvolvimento de IP core para correção de erro no standard DVB-S2 (Digital Video Broadcasting - Satellite Generation 2)*", enquadra-se no contexto das demandas por autonomia tecnológica nacional nas áreas de defesa e comunicações.

O bloco de correção de erros do padrão DVB-S2 é composto por um codificador de canal LDPC, um codificador BCH e um *interleaver*, sendo o codec LDPC o bloco principal e de maior complexidade. Neste contexto, este trabalho apresenta o desenvolvimento de um *core* do codificador de canal LDPC, em conformidade com o padrão DVB-S2, desenvolvido em lógica programável.

O codec LDPC foi desenvolvido, inicialmente, em linguagem de programação C para fins de prova de conceito. Os resultados de simulação em C são utilizados para balizamento do desenvolvimento do *core* em lógica programável.

A arquitetura do codificador e do decodificador LDPC, implementados em lógica programável, foi definida de modo a viabilizar a implementação em FPGA. Os códigos LDPC utilizados no padrão DVB-S2 possuem dois tamanhos de palavra-código, que, no contexto de codificação LDPC, são equivalentemente denominadas *frame*. Os *frames* normais possuem tamanho de bloco $N = 64800$ *bits* e os *frames* pequenos possuem tamanho de bloco $N = 16200$ *bits*. Onze taxas de códigos são especificadas para os *frames* normais ($1/4$, $1/3$, $2/5$, $1/2$, $3/5$, $2/3$, $3/4$, $4/5$, $5/6$, $8/9$ e $9/10$) e dez para os *frames* pequenos ($1/5$, $1/3$, $2/5$, $4/9$, $3/5$, $2/3$, $11/15$, $7/9$, $37/49$, $8/9$). Para cada uma das taxas, uma matriz de paridade $\boldsymbol{H}$ é necessária para a codificação e para a decodificação. No entanto, pelo tamanho dos *frames* do padrão DVB-S2, as matrizes dos *frames* normais possuiriam 64800 colunas e um número variável de linhas e as matrizes dos *frames* pequenos possuiriam 16200 colunas e um número variável de linhas. Armazenar todas essas matrizes em uma FPGA se torna impraticável devido ao

limitado número de recursos de *hardware* disponível. As matrizes de paridade do padrão DVB-S2, contudo, apresentam uma periodicidade $M = 360$ (EROZ, SUN, & LEE, 2004). Aproveitando o fator de periodicidade das matrizes, foi possível reduzir drasticamente os requisitos de memória e armazenar todas as matrizes de paridade das taxas de código.

A definição da representação numérica inicial das variáveis envolvidas no sistema foi definida de acordo com trabalhos existentes em literatura, os quais são utilizados para fins de comparação de resultados. Em uma etapa posterior, foi alterada a representação numérica das variáveis, aumentando a resolução, com o intuito de avaliar o compromisso entre o desempenho do codec LDPC e a ocupação de recursos da FPGA,

Esta dissertação apresenta o projeto detalhado em FPGA de um codificador e de um decodificador LDPC para o padrão DVB-S2. O codificador e o decodificador são capazes de codificar e decodificar todas as onze taxas para os *frames* normais e todas as dez taxas para os *frames* pequenos.

Os resultados de simulação do *core* VHDL possibilitaram validar os algoritmos de codificação e decodificação, verificar o desempenho dos algoritmos quando submetidos a ruído branco Gaussiano e avaliar o compromisso entre o desempenho do codec LDPC e a ocupação dos recursos da FPGA.

Na etapa de validação dos algoritmos, todas as vinte e uma taxas foram avaliadas. Para cada uma das taxas foram geradas mensagens aleatórias, as quais foram codificadas e aplicadas à entrada do decodificador. Para cada palavra-código aplicada à entrada do decodificador, foi verificada a saída do decodificador ao final da primeira iteração, de modo a assegurar que todas as equações de verificação de paridade fossem satisfeitas.

Na etapa de verificação do desempenho do codec LDPC quando submetido a ruído branco Gaussiano, foram avaliadas seis taxas de código para os *frames* normais e seis taxas de código para os *frames* pequenos. Considerando a mesma representação numérica de 4 *bits* da função ψ proposta e implementada por Loi (LOI, 2010), os resultados obtidos neste trabalho, considerando *frames* normais, apresentaram uma melhora de aproximadamente

1,0 dB para a taxa $R = 1/2$ e um desempenho levemente inferior (em torno de 0,2 dB) para a taxa $R = 2/3$. Para os *frames* pequenos, os resultados obtidos neste trabalho são semelhantes aos obtidos por Loi para a taxa taxa $R = 1/2$ e em torno de 0,7 dB inferior para a taxa $R = 2/3$.

Com relação à comparação das simulações do *core* VHDL (considerando a representação numérica de 4 *bits* da função ψ) com as simulações em linguagem C (considerando ponto flutuante), o código implementado em VHDL apresentou desempenho inferior em aproximadamente 1,7 dB.

Na etapa de avaliação do compromisso entre o desempenho do codec LDPC e a ocupação dos recursos da FPGA, alterou-se a resolução da representação numérica da variável que representa a função ψ, de quatro para cinco e, posteriormente, para 6 *bits*. Os resultados de comparação entre a simulação do *core* VHDL e a simulação em C demostram que a diferença entre as curvas foi reduzida em aproximadamente 1 dB, ou seja, o desempenho do *core* aumentou em aproximadamente 1 dB com o aumento de 1 *bit* na resolução numérica da função ψ. Para a resolução de 6 *bits* da função ψ, obteve-se uma diferença entre os resultados do core VHDL e da simulação em C entre 0,3 e 0,4 dB, ou seja, o desempenho do *core* aumentou entre 1,3 e 1,4 dB com o aumento de 2 *bits* na resolução numérica da função ψ.

Comparando os resultados obtidos neste trabalho, utilizando 6 *bits* para a representação numérica da função ψ, com os resultados obtidos por Loi (LOI, 2010), obteve-se desempenho superior em aproximadamente 2,2 dB para a taxa $R = 1/2$ e 1,2 dB para a taxa $R = 2/3$.

A melhora no desempenho obtida através do aumento da representação numérica da função ψ acarreta em maior ocupação dos recursos de *hardware* da FPGA. As três implementações do codec LDPC (com 4, 5 e 6 *bits*) foram sintetizadas e os resultados demonstram que, para a representação de 4 *bits*, a implementação proposta neste trabalho apresentou uma redução na utilização de *slices* de flip-flop e de *slices* LUT, e um aumento no uso de blocos BRAM, quando comparada à implementação proposta por Loi (LOI, 2010). Neste trabalho foram utilizados 45183 *slices* de flip-flop, 79524 *slices* LUT e 334 blocos

BRAM, enquanto que a implementação de Loi utilizou 51244 *slices* de flip-flop, 90432 *slices* LUT e 258 blocos BRAM.

A síntese da implementação do *core* utilizando 5 *bits* na representação da função ψ apresentou o mesmo uso de *slices* de flip-flop e *slices* LUT, tendo aumentado somente o número de blocos BRAM (de 334 para 405 blocos). Cabe salientar que esta representação acarretou em uma melhora no desempenho de aproximadamente 1 dB, ao custo do aumento de 71 blocos BRAM.

A síntese da implementação do *core* utilizando 6 *bits* na representação da função ψ apresentou aumento no uso de *slices* de flip-flop (de 45183 para 61189 *slices*), *slices* LUT (de 79524 para 84899 *slices*) e número de blocos BRAM (de 405 para 518 blocos).

Em síntese, como principais contribuições deste trabalho pode-se destacar:

- A definição de uma arquitetura para o codec LDPC com viabilidade de implementação em FPGA;
- A implementação de um *core* LDPC funcional, desenvolvido em lógica programável, em conformidade com o padrão DVB-S2;
- A avaliação entre desempenho do codec LDPC e a ocupação de recursos da FPGA;
- O projeto e a implementação do *core* em FPGA de um codificador e de um decodificador LDPC para o padrão DVB-S2, disponível para o desenvolvimento de produtos tecnológicos de alto valor agregado, no âmbito da demanda pela conquista de autonomia tecnológica nacional nas áreas de defesa e comunicações.

6.1 Trabalhos Futuros

Para trabalhos futuros, outras representações numéricas da função ψ poderão ser avaliadas, assim como a representação de outras variáveis utilizadas no sistema, tais como os valores LLR e os valores da RAM.

Algoritmos alternativos apresentados por (PAPAHRALABOS, et al., 2008) podem ser implementados nas FUs para melhorar a performance de erro

do decodificador LDPC na FPGA, na expectativa de aproximar a performance do sistema aos requisitos de performance de erro definidos pelo padrão DVB-S2.

O *core* implementado pode ser acrescido de um codec BCH e um *Interleaver,* de modo a contemplar todas as funcionalidades do bloco FEC previstas no padrão DVB-S2. O padrão também prevê a adoção de modulações mais densas, como o QPSK, o 8PSK, 16APSK e 32APSK, para as quais o desempenho do *core* deve ser avaliado.

Uma potencial plataforma para o desenvolvimento do codificador LDPC DVB-S2 e do decodificador DVB-S2 é a implementação utilizando unidades de processamento gráfico (GPU). GPUs modernas são conhecidas pelo seu grande número de processadores dedicados para cálculos aritméticos (PATTERSON & HENESSY, 2009). Uma vez que a decodificação dos códigos LDPC DVB-S2 é altamente paralela e é necessário um grande número de unidades de cálculo, a plataforma GPU é bastante adequada na implementação dos decodificadores LDPC. Como a GPU é programada em ambiente de *software*, tem uma flexibilidade similar à flexibilidade das FPGAs porque os programas de GPU podem ser facilmente recompilados para o uso em uma diferente GPU. Além disso, as GPUs possuem unidades aritméticas em ponto flutuante embutidas em cada processador, podendo então processar os cálculos dos passos de Atualização dos Nodos de Validação e de Atualização dos Nodos de Bit com uma precisão maior que nos projetos com FPGA, o que melhoraria a *performance* do erro do decodificador.

Referências

ALBERTY, E.; DEFEVER, S.; MOREAU, C.; DE GAUNDENZI, R.; GINESI, A.; RINALDO, R.; VERNUCCI, A. **Adaptive Coding and Modulation for the DVB-S2 Standard Interactive Applications:** Capacity Assessment and Key System Issues. IEEE Wireless Communications, 2007 pp. 61-69.

ASH, R. **Information Theory***.* Interscience. John Wiley & Sons, 1967.

BERROU, C. G.; THITIMAJSHIMA, P. **Near Shannon limit error-correction coding and decoding:** Turbo-codes. IEEE International Conference on Communications, 1993, pp. 1064-1070.

CARLSON, A. B. **Communication Systems***.* McGraw-Hill, 1965.

CHEN, C.-T. **Linear System Theory and Design***.* Harcourt Brace Publishers, 1984.

CLARK JR, G. C.; CAIN, J. B. **Error-Correction Coding for Digital Communication.** Plenum Press, 1988.

DE CASTRO, F.; DE CASTRO, M. C. (2001). **Comunicação Digital**. Disponível em: <http://www.feng.pucrs.br/~decastro/pdf/cd4.pdf>, 2001. Acesso em 15 abr. 2016.

EROZ, M.; SUN, F.-W.; LEE, L. **DVB-S2 Low Density Parity Check Codes with Near Shannon Limit Performance**. International Journal of Satellite Communications and Networking, pp. 269-279, jun. 2004.

EROZ, M.; SUN, F.-W.; LEE, L.-N. **An Innovative Low-Density Parity-Check Code Design With near-Shannon-limit Performance and Simple Implementation**. IEEE Transactions on Communications, v. 54, pp. 13-17, jan. 2006.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. **ETSI EN 302 307 V 1.2.1**: Digital Video Broadcasting (DVB)**:** Second Generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications, 2009. Disponível em: <http://www.etsi.org/deliver/etsi_en/302300_302399/302307/01.02.01_60/en_302307v010201p.pdf>. Acesso em: 15 abr. 2016.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. **ETSI TR 102 376 V 1.1.1**. Digital Video Broadcasting (DVB): User guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications, 2005. Disponível em: < http://www.etsi.org/deliver/etsi_tr/102300_102399/102376/01.01.01_60/tr_102376v010101p.pdf>. Acesso em: 15 abr. 2016.

GALLAGER, R. G. **Low-Density Parity-Check Codes**. Cambridge, MA: MIT, 1963.

GOMES, F.; GONÇALVES, J.; SILVA, V.; FALCAO, M.; MAIA, P. **High throughput encoder architecture for DVB-S2 LDPC-IRA codes**. International Conference on Microelectronic. pp. 271-274, 2007.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE 1076**. IEEE Standard VHDL Language Reference Manual. 2000. Disponível em: <https://standards.ieee.org/findstds/standard/1076-2000.html>. Acesso em: 15 abr. 2016.

JIN, H.; KAHNDEKAR, A.; MCELIECE, R. **Irregular repeat-accumulate codes**. 2nd International Symposium on Turbo Codes and Related Topics, 2000.

LIN, S.; COSTELLO, D. J. **Error Control Coding**. Englewood Cliff: Prentice-Hall, 1983.

LIN, S.; COSTELLO, D. J. **Error Control Coding**. 2nd Ed. Upper Saddle River, NJ: Prentice Hall, 2004.

LOI, K. C. **Field-Programmable Gate-Array (FPGA) Implementation of Low-Density Parity-Check (LDPC) Decoder in Digital Video Broadcasting - Second Generation (DVB-S2)**. Saskatoon, Saskatchewan: University of Saskatchewa,. 2010.

MASERA, G.; QUAGLIO, F.; VACCA, F. **Finite precision implementation of LDPC decoders**. IEEE Procedings - Communications, pp. 1098-1102, dez. 2005.

OH, D.; PARHI, K. K. **Low Complexity Implementations of Sum-Product Algorithm**. IEEE Workshop on Signal Processing Systems Design and Implementation, 2006, pp. 262-267, out. 2006.

PAPAHRALABOS, S.; PAPALEO, M.; MATHIOPOULOS, P. T.; NERI, M., VANELLI-CORALLI, A.; CORAZZA, G. (Março de 2008). **DVB-S2 LDPC**

Decoding Using Robust Check Node Update Approximations. IEEE Transactions on Broadcasting, v. 54, pp. 120-126, mar. 2008.

PATTERSON, D. A.; HENESSY, J. L. **Computer Organization and Design:** The hardware/software Interface, 4th ed. Burlington: Morgan Kaufmann, 2009.

PETERSON, W. W.; WELDON JR, E. J. **Error-Correction Codes***.* MIT Press, 1990.

PROAKIS, J. G. **Digital Communications***.* McGraw-Hill, 1995.
SHANNON, C. E. **A Mathematical Theory of Communications**. Bell Systems Technical Journal, pp. 379-423, 623-656, 1948.

TANNER, R. M. **Recursive Approach to Low Complexity Codes**. IEEE Transactions on Information Theory, v. *27* n. 05, pp. 533-547, set. 1981.

TAUB, H.; SCHILLING, D. L. **Principles of Communications Systems**. McGraw-Hill, 1986.

XIAO, Y.; KIM, K. **Alternative Good LDPC Code for DVB-S2**. ICSP 2008 Proceedings, pp. 1959-1962, 2008.

XILINX. **7-SERIES FPGA OVERVIEW**. S.d. Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf>. Acesso em 15 abr. 2016.

XILINX. **ISE**. S.d. Disponível em: < www.xilinx.com >. Acesso em: 15 abr. 2016

XILINX. **ISE Simulator**. 2016. Disponível em <http://www.xilinx.com/products/design-tools/isim.html >: Acesso em: 15 abr. 2016.

XILINX, **KINTEX-7 FPGAs DATA SHEET**. s.d. Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds182_Kintex_7_Data_Sheet.pdf>. Acesso em: 15 abr. 2016,

ZHANG, T.; WANG, Z.; PARHI, K. K. **On Finite Precision Implementation of Low Density Parity Check Codes Decoder**. IEEE International Symposium on Circuits and Systems, 2001, v. 4, pp. 202-205, mai. 2001.

Apêndice A

Valores do Anexo B e C do Padrão DVB-S2

Nesse apêndice, os valores do anexo B e C do Padrão DVB-S2 (ETSI EN 302 307 V 1.2.1, 2009) são reproduzidos. Os valores dos *frames* normais são mostrados primeiro, seguido pelos valores dos *frames* pequenos.

Tabela A.1 - N = 64800, Taxa de Código = 1/4

23606 36098 1140 28859 18148 18510 6226 540 42014 20879 23802 47088	36046 32914 11836
16419 24928 16609 17248 7693 24997 42587 16858 34921 21042 37024 20692	7304 39782 33721
1874 40094 18704 14474 14004 11519 13106 28826 38669 22363 30255 31105	16905 29962 12980
22254 40564 22645 22532 6134 9176 39998 23892 8937 15608 16854 31009	11171 23709 22460
8037 40401 13550 19526 41902 28782 13304 32796 24679 27140 45980 10021	34541 9937 44500
40540 44498 13911 22435 32701 18405 39929 25521 12497 9851 39223 34823	14035 47316 8815
15233 45333 5041 44979 45710 42150 19416 1892 23121 15860 8832 10308	15057 45482 24461
10468 44296 3611 1480 37581 32254 13817 6883 32892 40258 46538 11940	30518 36877 879
6705 21634 28150 43757 895 6547 20970 28914 30117 25736 41734 11392	7583 13364 24332
22002 5739 27210 27828 34192 37992 10915 6998 3824 42130 4494 35739	448 27056 4682
8515 1191 13642 30950 25943 12673 16726 34261 31828 3340 8747 39225	12083 31378 21670
18979 17058 43130 4246 4793 44030 19454 29511 47929 15174 24333 19354	1159 18031 2221
16694 8381 29642 46516 32224 26344 9405 18292 12437 27316 35466 41992	17028 38715 9350
15642 5871 46489 26723 23396 7257 8974 3156 37420 44823 35423 13541	17343 24530 29574
42858 32008 41282 38773 26570 2702 27260 46974 1469 20887 27426 38553	46128 31039 32818
22152 24261 8297	20373 36967 18345
19347 9978 27802	46685 20622 32806
34991 6354 33561	
29782 30875 29523	
9278 48512 14349	
38061 4165 43878	
8548 33172 34410	
22535 28811 23950	
20439 4027 24186	
38618 8187 30947	
35538 43880 21459	
7091 45616 15063	
5505 9315 21908	

Fonte: ETSI (2007).

Tabela A.2 - N = 64800, Taxa de Código = 1/3

34903 20927 32093 1052 25611 16093 16454 5520 506 37399 18518 21120	29094 5357 19224
11636 14594 22158 14763 15333 6838 22222 37856 14985 31041 18704 32910	9562 24436 28637
17449 1665 35639 16624 12867 12449 10241 11650 25622 34372 19878 26894	40177 2326 13504
29235 19780 36056 20129 20029 5457 8157 35554 21237 7943 13873 14980	6834 21583 42516
9912 7143 35911 12043 17360 37253 25588 11827 29152 21936 24125 40870	40651 42810 25709
40701 36035 39556 12366 19946 29072 16365 35495 22686 11106 8756 34863	31557 32138 38142
19165 15702 13536 40238 4465 40034 40590 37540 17162 1712 20577 14138	18624 41867 39296
31338 19342 9301 39375 3211 1316 33409 28670 12282 6118 29236 35787	37560 14295 16245
11504 30506 19558 5100 24188 24738 30397 33775 9699 6215 3397 37451	6821 21679 31570
34689 23126 7571 1058 12127 27518 23064 11265 14867 30451 28289 2966	25339 25083 22081
11660 15334 16867 15160 38343 3778 4265 39139 17293 26229 42604 13486	8047 697 35268
31497 1365 14828 7453 26350 41346 28643 23421 8354 16255 11055 24279	9884 17073 19995
15687 12467 13906 5215 41328 23755 20800 6447 7970 2803 33262 39843	26848 35245 8390
5363 22469 38091 28457 36696 34471 23619 2404 24229 41754 1297 18563	18658 16134 14807
3673 39070 14480 30279 37483 7580 29519 30519 39831 20252 18132 20010	12201 32944 5035
34386 7252 27526 12950 6875 43020 31566 39069 18985 15541 40020 16715	25236 1216 38986
1721 37332 39953 17430 32134 29162 10490 12971 28581 29331 6489 35383	42994 24782 8681
736 7022 42349 8783 6767 11871 21675 10325 11548 25978 431 24085	28321 4932 34249
1925 10602 28585 12170 15156 34404 8351 13273 20208 5800 15367 21764	4107 29382 32124
16279 37832 34792 21250 34192 7406 41488 18346 29227 26127 25493 7048	22157 2624 14468
39948 28229 24899	38788 27081 7936
17408 14274 38993	4368 26148 10578
38774 15968 28459	25353 4122 39751
41404 27249 27425	
41229 6082 43114	
13957 4979 40654	
3093 3438 34992	
34082 6172 28760	
42210 34141 41021	
14705 17783 10134	
41755 39884 22773	
14615 15593 1642	
29111 37061 39860	
9579 33552 633	
12951 21137 39608	
38244 27361 29417	
2939 10172 36479	

Fonte: ETSI (2007).

Tabela A.3 - N = 64800, Taxa de Código = 2/5

31413 18834 28884 947 23050 14484 14809 4968 455 33659 16666 19008	25796 31795 12152	28229 31684 30160
13172 19939 13354 13719 6132 20086 34040 13442 27958 16813 29619 16553	12184 35088 31226	15293 8483 28002
1499 32075 14962 11578 11204 9217 10485 23062 30936 17892 24204 24885	38263 33386 24892	14880 13334 12584
32490 18086 18007 4957 7285 32073 19038 7152 12486 13483 24808 21759	23114 37995 29796	28646 2558 19687
32321 10839 15620 33521 23030 10646 26236 19744 21713 36784 8016 12869	34336 10551 36245	6259 4499 26336
35597 11129 17948 26160 14729 31943 20416 10000 7882 31380 27858 33356	35407 175 7203	11952 28386 8405
14125 12131 36199 4058 35992 36594 33698 15475 1566 18498 12725 7067	14654 38201 22605	10609 961 7582
17406 8372 35437 2888 1184 30068 25802 11056 5507 26313 32205 37232	28404 6595 1018	10423 13191 26818
15254 5365 17308 22519 35009 718 5240 16778 23131 24092 20587 33385	19932 3524 29305	15922 36654 21450
27455 17602 4590 21767 22266 27357 30400 8732 5596 3060 33703 3596	31749 20247 8128	10492 1532 1205
6882 873 10997 24738 20770 10067 13379 27409 25463 2673 6998 31378	18026 36357 26735	30551 36482 22153
15181 13645 34501 3393 3840 35227 15562 23615 38342 12139 19471 15483	7543 29767 13588	5156 11330 34243
13350 6707 23709 37204 25778 21082 7511 14588 10010 21854 28375 33591	13333 25965 8463	28616 35369 13322
12514 4695 37190 21379 18723 5802 7182 2529 29936 35860 28338 10835	14504 36796 19710	8962 1485 21186
34283 25610 33026 31017 21259 2165 21807 37578 1175 16710 21939 30841	4528 25299 7318	23541 17445 35561
27292 33730 6836 26476 27539 35784 18245 16394 17939 23094 19216 17432	35091 25550 14798	33133 11593 19895
11655 6183 38708 28408 35157 17089 13998 36029 15052 16617 5638 36464	7824 215 1248	33917 7863 33651
15693 28923 26245 9432 11675 25720 26405 5838 31851 26898 8090 37037	30848 5362 17291	20063 28331 10702
24418 27583 7959 35562 37771 17784 11382 11156 37855 7073 21685 34515	28932 30249 27073	13195 21107 21859
10977 13633 30969 7516 11943 18199 5231 13825 19589 23661 11150 35602	13062 2103 16206	4364 31137 4804
19124 30774 6670 37344 16510 26317 23518 22957 6348 34069 8845 20175	7129 32062 19612	5585 2037 4830
34985 14441 25668 4116 3019 21049 37308 24551 24727 20104 24850 12114	9512 21936 38833	30672 16927 14800
38187 28527 13108 13985 1425 21477 30807 8613 26241 33368 35913 32477	35849 33754 23450	
5903 34390 24641 26556 23007 27305 38247 2621 9122 32806 21554 18685	18705 28656 18111	
17287 27292 19033	22749 27456 32187	

Fonte: ETSI (2007).

Tabela A.4 - N = 64800, Taxa de Código = 1/2

54 9318 14392 27561 26909 10219 2534 8597	36 26165 11241
55 7263 4635 2530 28130 3033 23830 3651	37 7666 26962
56 24731 23583 26036 17299 5750 792 9169	38 16290 8480
57 5811 26154 18653 11551 15447 13685 16264	39 11774 10120
58 12610 11347 28768 2792 3174 29371 12997	40 30051 30426
59 16789 16018 21449 6165 21202 15850 3186	41 1335 15424
60 31016 21449 17618 6213 12166 8334 18212	42 6865 17742
61 22836 14213 11327 5896 718 11727 9308	43 31779 12489
62 2091 24941 29966 23634 9013 15587 5444	44 32120 21001
63 22207 3983 16904 28534 21415 27524 25912	45 14508 6996
64 25687 4501 22193 14665 14798 16158 5491	46 979 25024
65 4520 17094 23397 4264 22370 16941 21526	47 4554 21896
66 10490 6182 32370 9597 30841 25954 2762	48 7989 21777
67 22120 22865 29870 15147 13668 14955 19235	49 4972 20661
68 6689 18408 18346 9918 25746 5443 20645	50 6612 2730
69 29982 12529 13858 4746 30370 10023 24828	51 12742 4418
70 1262 28032 29888 13063 24033 21951 7863	52 29194 595
71 6594 29642 31451 14831 9509 9335 31552	53 19267 20113
72 1358 6454 16633 20354 24598 624 5265	
73 19529 295 18011 3080 13364 8032 15323	
74 11981 1510 7960 21462 9129 11370 25741	
75 9276 29656 4543 30699 20646 21921 28050	
76 15975 25634 5520 31119 13715 21949 19605	
77 18688 4608 31755 30165 13103 10706 29224	
78 21514 23117 12245 26035 31656 25631 30699	
79 9674 24966 31285 29908 17042 24588 31857	
80 21856 27777 29919 27000 14897 11409 7122	
81 29773 23310 263 4877 28622 20545 22092	
82 15605 5651 21864 3967 14419 22757 15896	
83 30145 1759 10139 29223 26086 10556 5098	
84 18815 16575 2936 24457 26738 6030 505	
85 30326 22298 27562 20131 26390 6247 24791	
86 928 29246 21246 12400 15311 32309 18608	
87 20314 6025 26689 16302 2296 3244 19613	
88 6237 11943 22851 15642 23857 15112 20947	
89 26403 25168 19038 18384 8882 12719 7093	
0 14567 24965	
1 3908 100	
2 10279 240	
3 24102 764	
4 12383 4173	
5 13861 15918	
6 21327 1046	
7 5288 14579	
8 28158 8069	
9 16583 11098	
10 16681 28363	
11 13980 24725	
12 32169 17989	
13 10907 2767	
14 21557 3818	
15 26676 12422	
16 7676 8754	
17 14905 20232	
18 15719 24646	
19 31942 8589	
20 19978 27197	
21 27060 15071	
22 6071 26649	
23 10393 11176	
24 9597 13370	
25 7081 17677	
26 1433 19513	
27 26925 9014	
28 19202 8900	
29 18152 30647	
30 20803 1737	
31 11804 25221	
32 31683 17783	
33 29694 9345	
34 12280 26611	
35 6526 26122	

Fonte: ETSI (2007).

Tabela A.5 - N = 64800, Taxa de Código = 3/5

22422 10282 11626 19997 11161 2922 3122 99 5625 17064 8270 179	25 6393 3725
25087 16218 17015 828 20041 25656 4186 11629 22599 17305 22515 6463	26 597 19968
11049 22853 25706 14388 5500 19245 8732 2177 13555 11346 17265 3069	27 5743 8084
16581 22225 12563 19717 23577 11555 25496 6853 25403 5218 15925 21766	28 6770 9548
16529 14487 7643 10715 17442 11119 5679 14155 24213 21000 1116 15620	29 4285 17542
5340 8636 16693 1434 5635 6516 9482 20189 1066 15013 25361 14243	30 13568 22599
18506 22236 20912 8952 5421 15691 6126 21595 500 6904 13059 6802	31 1786 4617
8433 4694 5524 14216 3685 19721 25420 9937 23813 9047 25651 16826	32 23238 11648
21500 24814 6344 17382 7064 13929 4004 16552 12818 8720 5286 2206	33 19627 2030
22517 2429 19065 2921 21611 1873 7507 5661 23006 23128 20543 19777	34 13601 13458
1770 4636 20900 14931 9247 12340 11008 12966 4471 2731 16445 791	35 13740 17328
6635 14556 18865 22421 22124 12697 9803 25485 7744 18254 11313 9004	36 25012 13944
19982 23963 18912 7206 12500 4382 20067 6177 21007 1195 23547 24837	37 22513 6687
756 11158 14646 20534 3647 17728 11676 11843 12937 4402 8261 22944	38 4934 12587
9306 24009 10012 11081 3746 24325 8060 19826 842 8836 2898 5019	39 21197 5133
7575 7455 25244 4736 14400 22981 5543 8006 24203 13053 1120 5128	40 22705 6938
3482 9270 13059 15825 7453 23747 3656 24585 16542 17507 22462 14670	41 7534 24633
15627 15290 4198 22748 5842 13395 23918 16985 14929 3726 25350 24157	42 24400 12797
24896 16365 16423 13461 16615 8107 24741 3604 25904 8716 9604 20365	43 21911 25712
3729 17245 18448 9862 20831 25326 20517 24618 13282 5099 14183 8804	44 12039 1140
16455 17646 15376 18194 25528 1777 6066 21855 14372 12517 4488 17490	45 24306 1021
1400 8135 23375 20879 8476 4084 12936 25536 22309 16582 6402 24360	46 14012 20747
25119 23586 128 4761 10443 22536 8607 9752 25446 15053 1856 4040	47 11265 15219
377 21160 13474 5451 17170 5938 10256 11972 24210 17833 22047 16108	48 4670 15531
13075 9648 24546 13150 23867 7309 19798 2988 16858 4825 23950 15125	49 9417 14359
20526 3553 11525 23366 2452 17626 19265 20172 18060 24593 13255 1552	50 2415 6504
18839 21132 20119 15214 14705 7096 10174 5663 18651 19700 12524 14033	51 24964 24690
4127 2971 17499 16287 22368 21463 7943 18880 5567 8047 23363 6797	52 14443 8816
10651 24471 14325 4081 7258 4949 7044 1078 797 22910 20474 4318	53 6926 1291
21374 13231 22985 5056 3821 23718 14178 9978 19030 23594 8895 25358	54 6209 20806
6199 22056 7749 13310 3999 23697 16445 22636 5225 22437 24153 9442	55 13915 4079
7978 12177 2893 20778 3175 8645 11863 24623 10311 25767 17057 3691	56 24410 13196
20473 11294 9914 22815 2574 8439 3699 5431 24840 21908 16088 18244	57 13505 6117
8208 5755 19059 8541 24924 6454 11234 10492 16406 10831 11436 9649	58 9869 8220
16264 11275 24953 2347 12667 19190 7257 7174 24819 2938 2522 11749	59 1570 6044
3627 5969 13862 1538 23176 6353 2855 17720 2472 7428 573 15036	60 25780 17387
0 18539 18661	61 20671 24913
1 10502 3002	62 24558 20591
2 9368 10761	63 12402 3702
3 12299 7828	64 8314 1357
4 15048 13362	65 20071 14616
5 18444 24640	66 17014 3688
6 20775 19175	67 19837 946
7 18970 10971	68 15195 12136
8 5329 19982	69 7758 22808
9 11296 18655	70 3564 2925
10 15046 20659	71 3434 7769
11 7300 22140	
12 22029 14477	
13 11129 742	
14 13254 13813	
15 19234 13273	
16 6079 21122	
17 22782 5828	
18 19775 4247	
19 1660 19413	
20 4403 3649	
21 13371 25851	
22 22770 21784	
23 10757 14131	
24 16071 21617	

Fonte: ETSI (2007).

Tabela A.6 - N = 64800, Taxa de Código = 2/3

0 10491 16043 506 12826 8065 8226 2767 240 18673 9279 10579 20928	4 9161 15642
1 17819 8313 6433 6224 5120 5824 12812 17187 9940 13447 13825 18483	5 10714 10153
2 17957 6024 8681 18628 12794 5915 14576 10970 12064 20437 4455 7151	6 11585 9078
3 19777 6183 9972 14536 8182 17749 11341 5556 4379 17434 15477 18532	7 5359 9418
4 4651 19689 1608 659 16707 14335 6143 3058 14618 17894 20684 5306	8 9024 9515
5 9778 2552 12096 12369 15198 16890 4851 3109 1700 18725 1997 15882	9 1206 16354
6 486 6111 13743 11537 5591 7433 15227 14145 1483 3887 17431 12430	10 14994 1102
7 20647 14311 11734 4180 8110 5525 12141 15761 18661 18441 10569 8192	11 9375 20796
8 3791 14759 15264 19918 10132 9062 10010 12786 10675 9682 19246 5454	12 15964 6027
9 19525 9485 7777 19999 8378 9209 3163 20232 6690 16518 716 7353	13 14789 6452
10 4588 6709 20202 10905 915 4317 11073 13576 16433 368 3508 21171	14 8002 18591
11 14072 4033 19959 12608 631 19494 14160 8249 10223 21504 12395 4322	15 14742 14089
12 13800 14161	16 253 3045
13 2948 9647	17 1274 19286
14 14693 16027	18 14777 2044
15 20506 11082	19 13920 9900
16 1143 9020	20 452 7374
17 13501 4014	21 18206 9921
18 1548 2190	22 6131 5414
19 12216 21556	23 10077 9726
20 2095 19897	24 12045 5479
21 4189 7958	25 4322 7990
22 15940 10048	26 15616 5550
23 515 12614	27 15561 10661
24 8501 8450	28 20718 7387
25 17595 16784	29 2518 18804
26 5913 8495	30 8984 2600
27 16394 10423	31 6516 17909
28 7409 6981	32 11148 98
29 6678 15939	33 20559 3704
30 20344 12987	34 7510 1569
31 2510 14588	35 16000 11692
32 17918 6655	36 9147 10303
33 6703 19451	37 16650 191
34 496 4217	38 15577 18685
35 7290 5766	39 17167 20917
36 10521 8925	40 4256 3391
37 20379 11905	41 20092 17219
38 4090 5838	42 9218 5056
39 19082 17040	43 18429 8472
40 20233 12352	44 12093 20753
41 19365 19546	45 16345 12748
42 6249 19030	46 16023 11095
43 11037 19193	47 5048 17595
44 19760 11772	48 18995 4817
45 19644 7428	49 16483 3536
46 16076 3521	50 1439 16148
47 11779 21062	51 3661 3039
48 13062 9682	52 19010 18121
49 8934 5217	53 8968 11793
50 11087 3319	54 13427 18003
51 18892 4356	55 5303 3083
52 7894 3898	56 531 16668
53 5963 4360	57 4771 6722
54 7346 11726	58 5695 7960
55 5182 5609	59 3589 14630
56 2412 17295	
57 9845 20494	
58 6687 1864	
59 20564 5216	
0 18226 17207	
1 9380 8266	
2 7073 3065	
3 18252 13437	

Fonte: ETSI (2007).

Tabela A.7 - N = 64800, Taxa de Código = 3/4

0 6385 7901 14611 13389 11200 3252 5243 2504 2722 821 7374	24 2655 14957
1 11359 2698 357 13824 12772 7244 6752 15310 852 2001 11417	25 5565 6332
2 7862 7977 6321 13612 12197 14449 15137 13860 1708 6399 13444	26 4303 12631
3 1560 11804 6975 13292 3646 3812 8772 7306 5795 14327 7866	27 11653 12236
4 7626 11407 14599 9689 1628 2113 10809 9283 1230 15241 4870	28 16025 7632
5 1610 5699 15876 9446 12515 1400 6303 5411 14181 13925 7358	29 4655 14128
6 4059 8836 3405 7853 7992 15336 5970 10368 10278 9675 4651	30 9584 13123
7 4441 3963 9153 2109 12683 7459 12030 12221 629 15212 406	31 13987 9597
8 6007 8411 5771 3497 543 14202 875 9186 6235 13908 3563	32 15409 12110
9 3232 6625 4795 546 9781 2071 7312 3399 7250 4932 12652	33 8754 15490
10 8820 10088 11090 7069 6585 13134 10158 7183 488 7455 9238	34 7416 15325
11 1903 10818 119 215 7558 11046 10615 11545 14784 7961 15619	35 2909 15549
12 3655 8736 4917 15874 5129 2134 15944 14768 7150 2692 1469	36 2995 8257
13 8316 3820 505 8923 6757 806 7957 4216 15589 13244 2622	37 9406 4791
14 14463 4852 15733 3041 11193 12860 13673 8152 6551 15108 8758	38 11111 4854
15 3149 11981	39 2812 8521
16 13416 6906	40 8476 14717
17 13098 13352	41 7820 15360
18 2009 14460	42 1179 7939
19 7207 4314	43 2357 8678
20 3312 3945	44 7703 6216
21 4418 6248	0 3477 7067
22 2669 13975	1 3931 13845
23 7571 9023	2 7675 12899
24 14172 2967	3 1754 8187
25 7271 7138	4 7785 1400
26 6135 13670	5 9213 5891
27 7490 14559	6 2494 7703
28 8657 2466	7 2576 7902
29 8599 12834	8 4821 15682
30 3470 3152	9 10426 11935
31 13917 4365	10 1810 904
32 6024 13730	11 11332 9264
33 10973 14182	12 11312 3570
34 2464 13167	13 14916 2650
35 5281 15049	14 7679 7842
36 1103 1849	15 6089 13084
37 2058 1069	16 3938 2751
38 9654 6095	17 8509 4648
39 14311 7667	18 12204 8917
40 15617 8146	19 5749 12443
41 4588 11218	20 12613 4431
42 13660 6243	21 1344 4014
43 8578 7874	22 8488 13850
44 11741 2686	23 1730 14896
0 1022 1264	24 14942 7126
1 12604 9965	25 14983 8863
2 8217 2707	26 6578 8564
3 3156 11793	27 4947 396
4 354 1514	28 297 12805
5 6978 14058	29 13878 6692
6 7922 16079	30 11857 11186
7 15087 12138	31 14395 11493
8 5053 6470	32 16145 12251
9 12687 14932	33 13462 7428
10 15458 1763	34 14526 13119
11 8121 1721	35 2535 11243
12 12431 549	36 6465 12690
13 4129 7091	37 6872 9334
14 1426 8415	38 15371 14023
15 9783 7604	39 8101 10187
16 6295 11329	40 11963 4848
17 1409 12061	41 15125 6119
18 8065 9087	42 8051 14465
19 2918 8438	43 11139 5167
20 1293 14115	44 2883 14521
21 3922 13851	
22 3851 4000	
23 5865 1768	

Fonte: ETSI (2007).

Tabela A.8 - N = 64800, Taxa de Código = 4/5

0 149 11212 5575 6360 12559 8108 8505 408 10026 12828	0 5647 4935
1 5237 490 10677 4998 3869 3734 3092 3509 7703 10305	1 4219 1870
2 8742 5553 2820 7085 12116 10485 564 7795 2972 2157	2 10968 8054
3 2699 4304 8350 712 2841 3250 4731 10105 517 7516	3 6970 5447
4 12067 1351 11992 12191 11267 5161 537 6166 4246 2363	4 3217 5638
5 6828 7107 2127 3724 5743 11040 10756 4073 1011 3422	5 8972 669
6 11259 1216 9526 1466 10816 940 3744 2815 11506 11573	6 5618 12472
7 4549 11507 1118 1274 11751 5207 7854 12803 4047 6484	7 1457 1280
8 8430 4115 9440 413 4455 2262 7915 12402 8579 7052	8 8868 3883
9 3885 9126 5665 4505 2343 253 4707 3742 4166 1556	9 8866 1224
10 1704 8936 6775 8639 8179 7954 8234 7850 8883 8713	10 8371 5972
11 11716 4344 9087 11264 2274 8832 9147 11930 6054 5455	11 266 4405
12 7323 3970 10329 2170 8262 3854 2087 12899 9497 11700	12 3706 3244
13 4418 1467 2490 5841 817 11453 533 11217 11962 5251	13 6039 5844
14 1541 4525 7976 3457 9536 7725 3788 2982 6307 5997	14 7200 3283
15 11484 2739 4023 12107 6516 551 2572 6628 8150 9852	15 1502 11282
16 6070 1761 4627 6534 7913 3730 11866 1813 12306 8249	16 12318 2202
17 12441 5489 8748 7837 7660 2102 11341 2936 6712 11977	17 4523 965
18 10155 4210	18 9587 7011
19 1010 10483	19 2552 2051
20 8900 10250	20 12045 10306
21 10243 12278	21 11070 5104
22 7070 4397	22 6627 6906
23 12271 3887	23 9889 2121
24 11980 6836	24 829 9701
25 9514 4356	25 2201 1819
26 7137 10281	26 6689 12925
27 11881 2526	27 2139 8757
28 1969 11477	28 12004 5948
29 3044 10921	29 8704 3191
30 2236 8724	30 8171 10933
31 9104 6340	31 6297 7116
32 7342 8582	32 616 7146
33 11675 10405	33 5142 9761
34 6467 12775	34 10377 8138
35 3186 12198	35 7616 5811
0 9621 11445	0 7285 9863
1 7486 5611	1 7764 10867
2 4319 4879	2 12343 9019
3 2196 344	3 4414 8331
4 7527 6650	4 3464 642
5 10693 2440	5 6960 2039
6 6755 2706	6 786 3021
7 5144 5998	7 710 2086
8 11043 8033	8 7423 5601
9 4846 4435	9 8120 4885
10 4157 9228	10 12385 11990
11 12270 6562	11 9739 10034
12 11954 7592	12 424 10162
13 7420 2592	13 1347 7597
14 8810 9636	14 1450 112
15 689 5430	15 7965 8478
16 920 1304	16 8945 7397
17 1253 11934	17 6590 8316
18 9559 6016	18 6838 9011
19 312 7589	19 6174 9410
20 4439 4197	20 255 113
21 4002 9555	21 6197 5835
22 12232 7779	22 12902 3844
23 1494 8782	23 4377 3505
24 10749 3969	24 5478 8672
25 4368 3479	25 4453 2132
26 6316 5342	26 9724 1380
27 2455 3493	27 12131 11526
28 12157 7405	28 12323 9511
29 6598 11495	29 8231 1752
30 11805 4455	30 497 9022
31 9625 2090	31 9288 3080
32 4731 2321	32 2481 7515
33 3578 2608	33 2696 268
34 8504 1849	34 4023 12341
35 4027 1151	35 7108 5553

Fonte: ETSI (2007).

Tabela A.9 - N = 64800, Taxa de Código = 5/9

0 4362 416 8909 4156 3216 3112 2560 2912 6405 8593 4969 6723	0 1464 3559	0 1769 7837
1 2479 1786 8978 3011 4339 9313 6397 2957 7288 5484 6031 10217	1 3376 4214	1 3801 1689
2 10175 9009 9889 3091 4985 7267 4092 8874 5671 2777 2189 8716	2 7238 67	2 10070 2359
3 9052 4795 3924 3370 10058 1128 9996 10165 9360 4297 434 5138	3 10595 8831	3 3667 9918
4 2379 7834 4835 2327 9843 804 329 8353 7167 3070 1528 7311	4 1221 6513	4 1914 6920
5 3435 7871 348 3693 1876 6585 10340 7144 5870 2084 4052 2780	5 5300 4652	5 4244 5669
6 3917 3111 3476 1304 10331 5939 5199 1611 1991 699 8316 9960	6 1429 9749	6 10245 7821
7 6883 3237 1717 10752 7891 9764 4745 3888 10009 4176 4614 1567	7 7878 5131	7 7648 3944
8 10587 2195 1689 2968 5420 2580 2883 6496 111 6023 1024 4449	8 4435 10284	8 3310 5488
9 3786 8593 2074 3321 5057 1450 3840 5444 6572 3094 9892 1512	9 6331 5507	9 6346 9666
10 8548 1848 10372 4585 7313 6536 6379 1766 9462 2456 5606 9975	10 6662 4941	10 7088 6122
11 8204 10593 7935 3636 3882 394 5968 8561 2395 7289 9267 9978	11 9614 10238	11 1291 7827
12 7795 74 1633 9542 6867 7352 6417 7568 10623 725 2531 9115	12 8400 8025	12 10592 8945
13 7151 2482 4260 5003 10105 7419 9203 6691 8798 2092 8263 3755	13 9156 5630	13 3609 7120
14 3600 570 4527 200 9718 6771 1995 8902 5446 768 1103 6520	14 7067 8878	14 9168 9112
15 6304 7621	15 9027 3415	15 6203 8052
16 6498 9209	16 1690 3866	16 3330 2895
17 7293 6786	17 2854 8469	17 4264 10563
18 5950 1708	18 6206 630	18 10556 6496
19 8521 1793	19 363 5453	19 8807 7645
20 6174 7854	20 4125 7008	20 1999 4530
21 9773 1190	21 1612 6702	21 9202 6818
22 9517 10268	22 9069 9226	22 3403 1734
23 2181 9349	23 5767 4060	23 2106 9023
24 1949 5560	24 3743 9237	24 6881 3883
25 1556 555	25 7018 5572	25 3895 2171
26 8600 3827	26 8892 4536	26 4062 6424
27 5072 1057	27 853 6064	27 3755 9536
28 7928 3542	28 8069 5893	
29 3226 3762	29 2051 2885	
0 7045 2420	0 10691 3153	
1 9645 2641	1 3602 4055	
2 2774 2452	2 328 1717	
3 5331 2031	3 2219 9299	
4 9400 7503	4 1939 7898	
5 1850 2338	5 617 206	
6 10456 9774	6 8544 1374	
7 1692 9276	7 10676 3240	
8 10037 4038	8 6672 9489	
9 3964 338	9 3170 7457	
10 2640 5087	10 7868 5731	
11 858 3473	11 6121 10732	
12 5582 5683	12 4843 9132	
13 9523 916	13 580 9591	
14 4107 1559	14 6267 9290	
15 4506 3491	15 3009 2268	
16 8191 4182	16 195 2419	
17 10192 6157	17 8016 1557	
18 5668 3305	18 1516 9195	
19 3449 1540	19 8062 9064	
20 4766 2697	20 2095 8968	
21 4069 6675	21 753 7326	
22 1117 1016	22 6291 3833	
23 5619 3085	23 2614 7844	
24 8483 8400	24 2303 646	
25 8255 394	25 2075 611	
26 6338 5042	26 4687 362	
27 6174 5119	27 8684 9940	
28 7203 1989	28 4830 2065	
29 1781 5174	29 7038 1363	

Fonte: ETSI (2007).

Tabela A.10 - N = 64800, Taxa de Código = 8/9

0 6235 2848 3222	13 1969 3869	6 5821 4932	19 5736 1399	12 2644 5073
1 5800 3492 5348	14 3571 2420	7 6356 4756	0 970 2572	13 4212 5088
2 2757 927 90	15 4632 981	8 3930 418	1 2062 6599	14 3463 3889
3 6961 4516 4739	16 3215 4163	9 211 3094	2 4597 4870	15 5306 478
4 1172 3237 6264	17 973 3117	10 1007 4928	3 1228 6913	16 4320 6121
5 1927 2425 3683	18 3802 6198	11 3584 1235	4 4159 1037	17 3961 1125
6 3714 6309 2495	19 3794 3948	12 6982 2869	5 2916 2362	18 5699 1195
7 3070 6342 7154	0 3196 6126	13 1612 1013	6 395 1226	19 6511 792
8 2428 613 3761	1 573 1909	14 953 4964	7 6911 4548	0 3934 2778
9 2906 264 5927	2 850 4034	15 4555 4410	8 4618 2241	1 3238 6587
10 1716 1950 4273	3 5622 1601	16 4925 4842	9 4120 4280	2 1111 6596
11 4613 6179 3491	4 6005 524	17 5778 600	10 5825 474	3 1457 6226
12 4865 3286 6005	5 5251 5783	18 6509 2417	11 2154 5558	4 1446 3885
13 1343 5923 3529	6 172 2032	19 1260 4903	12 3793 5471	5 3907 4043
14 4589 4035 2132	7 1875 2475	0 3369 3031	13 5707 1595	6 6839 2873
15 1579 3920 6737	8 497 1291	1 3557 3224	14 1403 325	7 1733 5615
16 1644 1191 5998	9 2566 3430	2 3028 583	15 6601 5183	8 5202 4269
17 1482 2381 4620	10 1249 740	3 3258 440	16 6369 4569	9 3024 4722
18 6791 6014 6596	11 2944 1948	4 6226 6655	17 4846 896	10 5445 6372
19 2738 5918 3786	12 6528 2899	5 4895 1094	18 7092 6184	11 370 1828
0 5156 6166	13 2243 3616	6 1481 6847	19 6764 7127	12 4695 1600
1 1504 4356	14 867 3733	7 4433 1932	0 6358 1951	13 680 2074
2 130 1904	15 1374 4702	8 2107 1649	1 3117 6960	14 1801 6690
3 6027 3187	16 4698 2285	9 2119 2065	2 2710 7062	15 2669 1377
4 6718 759	17 4760 3917	10 4003 6388	3 1133 3604	16 2463 1681
5 6240 2870	18 1859 4058	11 6720 3622	4 3694 657	17 5972 5171
6 2343 1311	19 6141 3527	12 3694 4521	5 1355 110	18 5728 4284
7 1039 5465	0 2148 5066	13 1164 7050	6 3329 6736	19 1696 1459
8 6617 2513	1 1306 145	14 1965 3613	7 2505 3407	
9 1588 5222	2 2319 871	15 4331 66	8 2462 4806	
10 6561 535	3 3463 1061	16 2970 1796	9 4216 214	
11 4765 2054	4 5554 6647	17 4652 3218	10 5348 5619	
12 5966 6892	5 5837 339	18 1762 4777	11 6627 6243	

Fonte: ETSI (2007).

Tabela A.11 - N = 64800, Taxa de Código = 8/9

0 5611 2563 2900	16 1775 3476	14 906 4432	12 6146 3323	10 5161 2293
1 5220 3143 4813	17 3216 2178	15 3225 1111	13 1939 5002	11 4682 3845
2 2481 834 81	0 4165 884	16 6296 2583	14 5140 1437	12 3045 643
3 6265 4064 4265	1 2896 3744	17 1457 903	15 1263 293	13 2818 2616
4 1055 2914 5638	2 874 2801	0 855 4475	16 5949 4665	14 3267 649
5 1734 2182 3315	3 3423 5579	1 4097 3970	17 4548 6380	15 6236 593
6 3342 5678 2246	4 3404 3552	2 4433 4361	0 3171 4690	16 646 2948
7 2185 552 3385	5 2876 5515	3 5198 541	1 5204 2114	17 4213 1442
8 2615 236 5334	6 516 1719	4 1146 4426	2 6384 5565	0 5779 1596
9 1546 1755 3846	7 765 3631	5 3202 2902	3 5722 1757	1 2403 1237
10 4154 5561 3142	8 5059 1441	6 2724 525	4 2805 6264	2 2217 1514
11 4382 2957 5400	9 5629 598	7 1083 4124	5 1202 2616	3 5609 716
12 1209 5329 3179	10 5405 473	8 2326 6003	6 1018 3244	4 5155 3858
13 1421 3528 6063	11 4724 5210	9 5605 5990	7 4018 5289	5 1517 1312
14 1480 1072 5398	12 155 1832	10 4376 1579	8 2257 3067	6 2554 3158
15 3843 1777 4369	13 1689 2229	11 4407 984	9 2483 3073	7 5280 2643
16 1334 2145 4163	14 449 1164	12 1332 6163	10 1196 5329	8 4990 1353
17 2368 5055 260	15 2308 3088	13 5359 3975	11 649 3918	9 5648 1170
0 6118 5405	16 1122 669	14 1907 1854	12 3791 4581	10 1152 4366
1 2994 4370	17 2268 5758	15 3601 5748	13 5028 3803	11 3561 5368
2 3405 1669	0 5878 2609	16 6056 3266	14 3119 3506	12 3581 1411
3 4640 5550	1 782 3359	17 3322 4085	15 4779 431	13 5647 4661
4 1354 3921	2 1231 4231	0 1768 3244	16 3888 5510	14 1542 5401
5 117 1713	3 4225 2052	1 2149 144	17 4387 4084	15 5078 2687
6 5425 2866	4 4286 3517	2 1589 4291	0 5836 1692	16 316 1755
7 6047 683	5 5531 3184	3 5154 1252	1 5126 1078	17 3392 1991
8 5616 2582	6 1935 4560	4 1855 5939	2 5721 6165	
9 2108 1179	7 1174 131	5 4820 2706	3 3540 2499	
10 933 4921	8 3115 956	6 1475 3360	4 2225 6348	
11 5953 2261	9 3129 1088	7 4266 693	5 1044 1484	
12 1430 4699	10 5238 4440	8 4156 2018	6 6323 4042	
13 5905 480	11 5722 4280	9 2103 752	7 1313 5603	
14 4289 1846	12 3540 375	10 3710 3853	8 1303 3496	
15 5374 6208	13 191 2782	11 5123 931	9 3516 3639	

Fonte: ETSI (2007).

Tabela A.12 - N = 16200, Taxa de Código = 1/5

6295 9626 304 7695 4839 4936 1660 144 11203 5567 6347 12557
10691 4988 3859 3734 3071 3494 7687 10313 5964 8069 8296 11090
10774 3613 5208 11177 7676 3549 8746 6583 7239 12265 2674 4292
11869 3708 5981 8718 4908 10650 6805 3334 2627 10461 9285 11120
7844 3079 10773
3385 10854 5747
1360 12010 12202
6189 4241 2343
9840 12726 4977

Fonte: ETSI (2007).

Tabela A.13 - N = 16200, Taxa de Código = 1/3

416 8909 4156 3216 3112 2560 2912 6405 8593 4969 6723 6912
8978 3011 4339 9312 6396 2957 7288 5485 6031 10218 2226 3575
3383 10059 1114 10008 10147 9384 4290 434 5139 3536 1965 2291
2797 3693 7615 7077 743 1941 8716 6215 3840 5140 4582 5420
6110 8551 1515 7404 4879 4946 5383 1831 3441 9569 10472 4306
1505 5682 7778
7172 6830 6623
7281 3941 3505
10270 8669 914
3622 7563 9388
9930 5058 4554
4844 9609 2707
6883 3237 1714
4768 3878 10017
10127 3334 8267

Fonte: ETSI (2007).

Tabela A.14 - N = 16200, Taxa de Código = 2/5

5650 4143 8750 583 6720 8071 635 1767 1344 6922 738 6658
5696 1685 3207 415 7019 5023 5608 2605 857 6915 1770 8016
3992 771 2190 7258 8970 7792 1802 1866 6137 8841 886 1931
4108 3781 7577 6810 9322 8226 5396 5867 4428 8827 7766 2254
4247 888 4367 8821 9660 324 5864 4774 227 7889 6405 8963
9693 500 2520 2227 1811 9330 1928 5140 4030 4824 806 3134
1652 8171 1435
3366 6543 3745
9286 8509 4645
7397 5790 8972
6597 4422 1799
9276 4041 3847
8683 7378 4946
5348 1993 9186
6724 9015 5646
4502 4439 8474
5107 7342 9442
1387 8910 2660

Fonte: ETSI (2007).

Tabela A.15 - N = 16200, Taxa de Código = 4/9

20 712 2386 6354 4061 1062 5045 5158	11 8935 4996
21 2543 5748 4822 2348 3089 6328 5876	12 3028 764
22 926 5701 269 3693 2438 3190 3507	13 5988 1057
23 2802 4520 3577 5324 1091 4667 4449	14 7411 3450
24 5140 2003 1263 4742 6497 1185 6202	
0 4046 6934	
1 2855 66	
2 6694 212	
3 3439 1158	
4 3850 4422	
5 5924 290	
6 1467 4049	
7 7820 2242	
8 4606 3080	
9 4633 7877	
10 3884 6868	

Fonte: ETSI (2007).

Tabela A.16 - N = 16200, Taxa de Código = 3/5

2765 5713 6426 3596 1374 4811 2182 544 3394 2840 4310 771	5 1733 6028
4951 211 2208 723 1246 2928 398 5739 265 5601 5993 2615	6 3786 1936
210 4730 5777 3096 4282 6238 4939 1119 6463 5298 6320 4016	7 4292 956
4167 2063 4757 3157 5664 3956 6045 563 4284 2441 3412 6334	8 5692 3417
4201 2428 4474 59 1721 736 2997 428 3807 1513 4732 6195	9 266 4878
2670 3081 5139 3736 1999 5889 4362 3806 4534 5409 6384 5809	10 4913 3247
5516 1622 2906 3285 1257 5797 3816 817 875 2311 3543 1205	11 4763 3937
4244 2184 5415 1705 5642 4886 2333 287 1848 1121 3595 6022	12 3590 2903
2142 2830 4069 5654 1295 2951 3919 1356 884 1786 396 4738	13 2566 4215
0 2161 2653	14 5208 4707
1 1380 1461	15 3940 3388
2 2502 3707	16 5109 4556
3 3971 1057	17 4908 4177
4 5985 6062	

Fonte: ETSI (2007).

Tabela A.17 - N = 16200, Taxa de Código = 2/3

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

Fonte: ETSI (2007).

Tabela A.18 - N = 16200, Taxa de Código = 11/15

3 3198 478 4207 1481 1009 2616 1924 3437 554 683 1801	8 1015 1945
4 2681 2135	9 1948 412
5 3107 4027	10 995 2238
6 2637 3373	11 4141 1907
7 3830 3449	0 2480 3079
8 4129 2060	1 3021 1088
9 4184 2742	2 713 1379
10 3946 1070	3 997 3903
11 2239 984	4 2323 3361
0 1458 3031	5 1110 986
1 3003 1328	6 2532 142
2 1137 1716	7 1690 2405
3 132 3725	8 1298 1881
4 1817 638	9 615 174
5 1774 3447	10 1648 3112
6 3632 1257	11 1415 2808
7 542 3694	

Fonte: ETSI (2007).

Tabela A.19 - N = 16200, Taxa de Código = 7/9

5 896 1565	7 951 2068	9 2116 1855
6 2493 184	8 3108 3542	0 722 1584
7 212 3210	9 307 1421	1 2767 1881
8 727 1339	0 2272 1197	2 2701 1610
9 3428 612	1 1800 3280	3 3283 1732
0 2663 1947	2 331 2308	4 168 1099
1 230 2695	3 465 2552	5 3074 243
2 2025 2794	4 1038 2479	6 3460 945
3 3039 283	5 1383 343	7 2049 1746
4 862 2889	6 94 236	8 566 1427
5 376 2110	7 2619 121	9 3545 1168
6 2034 2286	8 1497 2774	

Fonte: ETSI (2007).

Tabela A.20 - N = 16200, Taxa de Código = 37/49

3 2409 499 1481 908 559 716 1270 333 2508 2264 1702 2805	6 497 2228
4 2447 1926	7 2326 1579
5 414 1224	0 2482 256
6 2114 842	1 1117 1261
7 212 573	2 1257 1658
0 2383 2112	3 1478 1225
1 2286 2348	4 2511 980
2 545 819	5 2320 2675
3 1264 143	6 435 1278
4 1701 2258	7 228 503
5 964 166	0 1885 2369
6 114 2413	1 57 483
7 2243 81	2 838 1050
0 1245 1581	3 1231 1990
1 775 169	4 1738 68
2 1696 1104	5 2392 951
3 1914 2831	6 163 645
4 532 1450	7 2644 1704
5 91 974	

Fonte: ETSI (2007).

Tabela A.21 - N = 16200, Taxa de Código = 8/9

0 1558 712 805	4 1496 502	3 544 1190
1 1450 873 1337	0 1006 1701	4 1472 1246
2 1741 1129 1184	1 1155 97	0 508 630
3 294 806 1566	2 657 1403	1 421 1704
4 482 605 923	3 1453 624	2 284 898
0 926 1578	4 429 1495	3 392 577
1 777 1374	0 809 385	4 1155 556
2 608 151	1 367 151	0 631 1000
3 1195 210	2 1323 202	1 732 1368
4 1484 692	3 960 318	2 1328 329
0 427 488	4 1451 1039	3 1515 506
1 828 1124	0 1098 1722	4 1104 1172
2 874 1366	1 1015 1428	
3 1500 835	2 1261 1564	

Fonte: ETSI (2007).