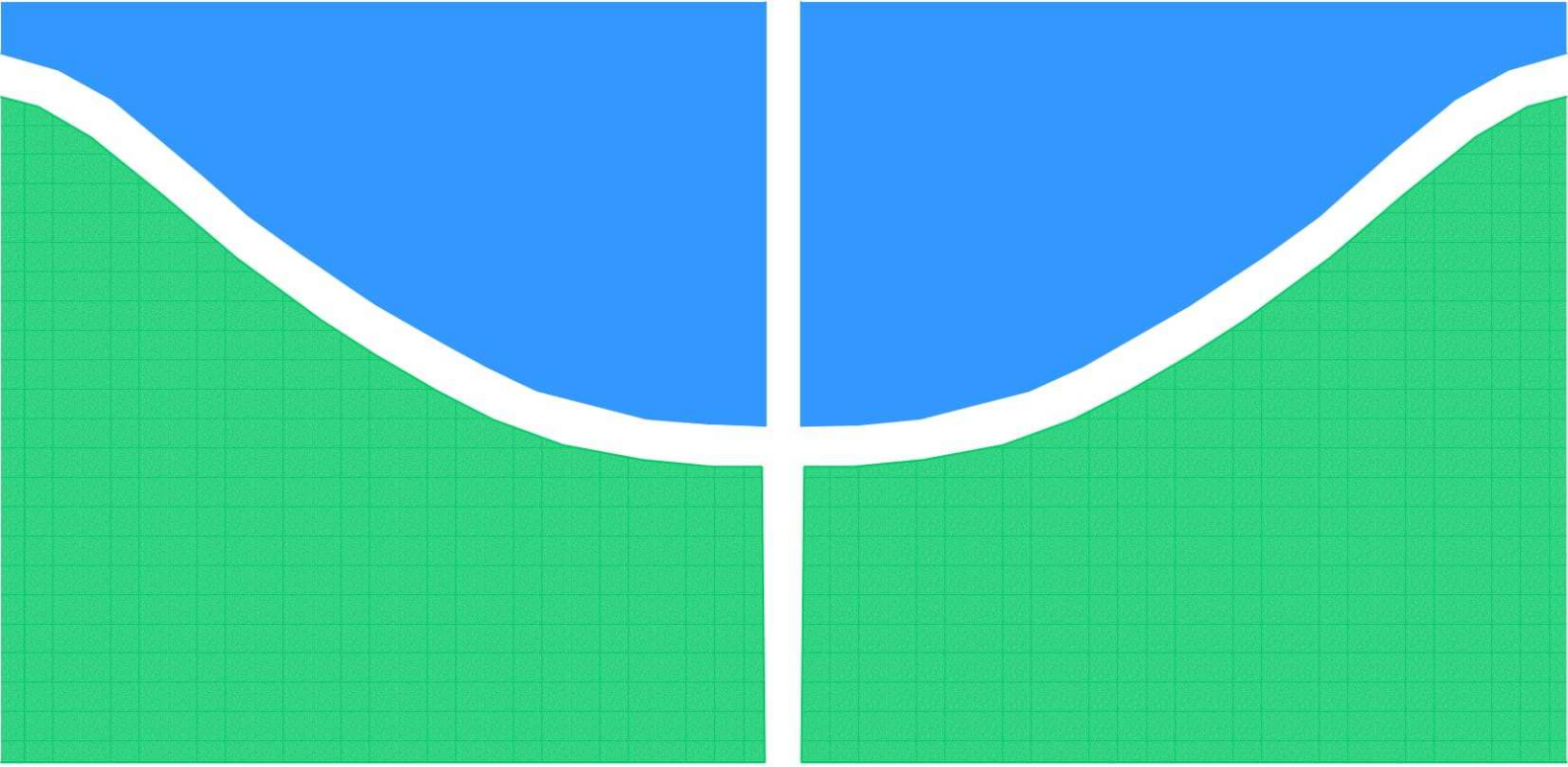

Universidade de Brasília - UnB

Faculdade do Gama - FGA

Engenharia Eletrônica

Implementação em FPGA de Geradores de Síndromes para Decodificação BCH DVB-S2X

Autor: David da Silva Ferreira

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2018

David da Silva Ferreira
Implementação em FPGA de Geradores de Síndromes para Decodificação BCH DVB-S2X/ David da Silva Ferreira. – Brasília, DF, 2018.
61 p. : il. (algumas color.); 30 cm.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade do Gama - FGA , 2018.

1. BCH. 2. DVB-S2X. I. Prof. Dr. Daniel Mauricio Muñoz Arboleda. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação em FPGA de Geradores de Síndromes para Decodificação BCH DVB-S2X

CDU 02:141:005.6

David da Silva Ferreira

Implementação em FPGA de Geradores de Síndromes para Decodificação BCH DVB-S2X

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, DF, 10 de dezembro de 2018.

Prof. Dr. Daniel Mauricio Muñoz Arboleda
Orientador

Prof. Dr. Leonardo Aguayo
Convidado 1

Prof. Dr. José Camargo da Costa
Convidado 2

Brasília, DF
2018

Resumo

A implementação de decodificadores BCH DVB-S2X enfrenta o desafio de determinar algoritmos que possam ser executados com os recursos de *hardware* disponíveis em um tempo compatível com as taxas de transmissão do sistema de comunicação. O padrão DVB-S2X determina tamanhos de *frames* entre 3240 e 55440 bits para o código BCH. O processamento paralelo de todos os bits durante a decodificação requer muitos recursos computacionais. E o processamento bit a bit torna o tempo de decodificação extensivo, limitando a taxa de recepção do terminal. O uso de FPGAs permite explorar o paralelismo intrínseco dos algoritmos, promovendo o aumento do desempenho da decodificação. Além disso, as linguagens de descrição de *hardware* permitem uma prototipagem rápida e de baixo custo.

Este trabalho visa a implementação em FPGA de arquiteturas de geradores de síndromes. Isto exigiu a implementação de um multiplicador de Galois parametrizável, de acordo com as ordens de campo determinadas pelo padrão DVB-S2X. Após a implementação do multiplicador de Galois, foi implementada uma arquitetura de entrada serial, na qual uma multiplicação é realizada para cada bit do *frame* para computar uma síndrome. Para melhoria de desempenho, uma arquitetura de entrada paralela foi implementada, em que os bits do *frame* são processados em pacotes de 8 bits, reduzindo o número de multiplicações por um fator de 8.

A plataforma de hardware usada para implementar as arquiteturas é um Xilinx Zynq Z -7010 XC7Z010-1CLG225C4334. A partir dos resultados de síntese, mapeamento e roteamento em FPGA, obteve-se informações sobre a frequência máxima de operação, consumo de recursos e energia. O gerador de síndromes de entrada serial atingiu a frequência máxima de operação de 200 MHz, com taxa de processamento de 11,11 Mbps e consumo de energia de 0,166W. O gerador de entrada paralela, com ordem 8 de paralelismo, atingiu a frequência máxima de 196 MHz, com taxa de processamento de 87,3 Mbps e consumo de energia de 0,184W. A arquitetura de entrada paralela apresentou melhor desempenho, com um pequeno aumento no consumo de *Look Up Tables* do FPGA.

Palavras-chave: Códigos corretores; BCH; Multiplicador de Galois; FPGA.

Abstract

The implementation of BCH DVB-S2X decoders faces the challenge of determining algorithms that can be run with the available hardware resources in a time compatible with the transmission rates of the communication system. The DVB-S2X standard determines frame sizes between 3240 and 55440 bits for BCH code. Parallel processing of all bits during decoding requires a lot of computational resources. And bitwise processing makes the decoding time extensive, limiting the reception rate of the terminal. The use of FPGAs allows to explore the intrinsic parallelism of the algorithms, promoting the increase of decoding performance. In addition, hardware description languages enable rapid, low-cost development.

This work aims at the implementation in FPGA of syndrome generator architectures. It required the implementation of a parameterizable Galois multiplier, according to the field orders determined by the DVB-S2X standard. After Galois multipier implementation, a serial input architecture was implemented, in which one multiplying is performed for each bit of frame to compute one syndrome. For performance improvement, a parallel input architecture was implemented, in which bits of frame are processed 8-by-8 in parallel, it reduces the multiplying executions by a factor of 8.

The hardware platform used to implement the architectures is a Xilinx Zynq Z-7010 XC7Z010-1CLG225C4334. From results of synthesis, mapping and routing in FPGA, it was obtained information about the maximum frequency of operation, resource consumption and energy. The serial input syndromes generator achieved maximum operating frequency of 200 MHz, with a processing rate of 11.11 Mbps and power consumption of 0.166W. The parallel input generator, with order 8 of parallelism, achieved maximum frequency of 196 MHz, with a processing rate of 87.3 Mbps and power consumption of 0.184W. The parallel input architecture presented better performance, with a small increase in FPGA Look Up Tables consumption.

Key-words: Correction codes; BCH; Galois multiplier; FPGA.

Lista de ilustrações

Lista de abreviaturas e siglas

ACM	*Adaptive Coding and Modulation*
ARM	*Advanced RISC Machine*
BCH	*Bose–Chaudhuri–Hocquenghem*
BEC	*Backward Error Correction*
CLB	*Configurable Logic Block*
DVB-S	*Digital Video Broadcasting - Satellite*
DVB-S2	*Digital Video Broadcasting - Satellite Second Generation*
DVB-S2X	*Digital Video Broadcasting - Satellite Second Generation Extension*
DTH	*Direct To Home*
FEC	*Forward Error Correction*
FPGA	*Field Programmable Gate Array*
IOB	*Input/Output Block*
LDPC	*Low-Density Parity Check*
MBD	*Model-Based Design*
MIL	*Model In the Loop*
PL	*Physical Layer*
PLD	*Programmable Logic Device*
RAM	*Read Only Memory*
RISC	*Reduced Instruction Set Computer*
SoC	*System on Chip*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very-High Speed Integrated Circuits*
VL-SNR	*Very-Low Signal to Noise Ratio*

Sumário

1 Introdução

Um sistema de comunicação genérico consiste de elementos trocando sinais, que carregam informações, por um meio físico denominado de canal, seja este um cabo ou um espaço onde se propagam ondas de altas frequências. O sinal pode ser alterado por distorções e ruídos durante a transmissão pelo canal. Distorções correspondem a operações aplicadas ao sinal que podem ser corrigidas, desde que se conheça a operação inversa. Ruídos são perturbações estatísticas imprevisíveis que nem sempre podem ser completamente removidas, exigindo do receptor alguma capacidade para decifrar a informação inserida no sinal ruidoso (SHANNON, 1948).

Em comunicações digitais, a transmissão é dividida em intervalos de tempo, nos quais o sinal transmitido é restrito a um número finito de formas de ondas. Assim, o receptor pode, com base nas propriedades estatísticas do ruído, estimar qual das formas de ondas possíveis tem maior probabilidade de ter sido transmitida. Códigos de correção de erros são utilizados para aumentar a robustez a ruídos de sistemas de comunicação digital. Esta técnica consiste em codificar a mensagem para adicionar redundâncias, seguindo um critério. A codificação auxilia o receptor a estimar a mensagem enviada. Algoritmos de decodificação são utilizados para identificar e corrigir erros na mensagem recebida (JR.; CAIN, 1981).

Existem padrões de transmissões que especificam as formas de onda do sinal transmitido e os processamentos aplicados à mensagem para diminuir a taxa de erro no recebimento. Para sistemas de comunicação de satelital, caracterizados por uma *hub* central se comunicando com terminais, o padrão DVB-S2X define como deve ser o sinal transmitido. A codificação do canal, segundo o padrão, inclui os códigos corretores de erros *Low-Density Parity Check* (LDPC) e *Bose–Chaudhuri–Hocquenghem*(BCH).

O LDPC adiciona redundâncias para que um conjunto extenso de equações de paridades sejam satisfeitas pelos bits do *frame* codificado. A decodificação do LDPC consiste em estimar o *frame* considerando as equações de paridades que devem ser satisfeitas. Este é um processo determinístico que não garante a ausência de erros no *frame* estimado. Por este motivo o BCH é utilizado para identificar e corrigir até 12 bits do *frame* estimado pelo LDPC. Na codificação do BCH, são adicionados bits de redundância para que o *frame* resultante, interpretado como um polinômio, satisfaça equações polinomiais em campos de Galois. A decodificação do BCH é um processo determinístico, dividido em três etapas: calcular as síndromes, solucionar a equação-chave e encontrar a localização dos erros.

1.1 Descrição do Problema

O padrão DVB-S2X determina tamanhos de *frame* codificados entre 3240 e 55440 bits para o BCH. A primeira etapa do processo de decodificação do BCH consiste em determinar valores de síndromes, o que possibilita a identificação e correção de erros. O cálculo das síndromes depende da análise de todos os bits do *frame* recebido. A paralelização do processamento dos bits é uma abordagem que possibilita o aumento da performance do decodificador. Em contrapartida, um alto grau de paralelismo pode implicar em um aumento significativo do consumo de recursos, o que pode inviabilizar o projeto devido ao aumento dos custos.

Uma análise comparativa entre arquiteturas de geradores de síndromes que processam os bits de forma serial e de forma paralela pode ser realizada a fim de determinar o custo-benefício da paralelização. No entanto, isto implicaria no desenvolvimento de um *hardware* dedicado, o que demandaria tempo e geraria mais gastos. Portanto, torna-se necessário a utilização de abordagens de prototipagem de *hardware* que sejam rápidas e de baixo custo.

Este trabalho está inserido como parte do desenvolvimento de um rádio definido por *software* (RDS), em que são recebidos *frames* codificados segundo o padrão DVB-S2X. A Figura 1 apresenta um diagrama de blocos com as etapas de processamento que devem ser realizados pelo terminal para decodificação dos códigos corretores de erro. O foco deste trabalho é o decodificador BCH apresentado na Figura 2, especificamente na implementação de geradores de síndromes.

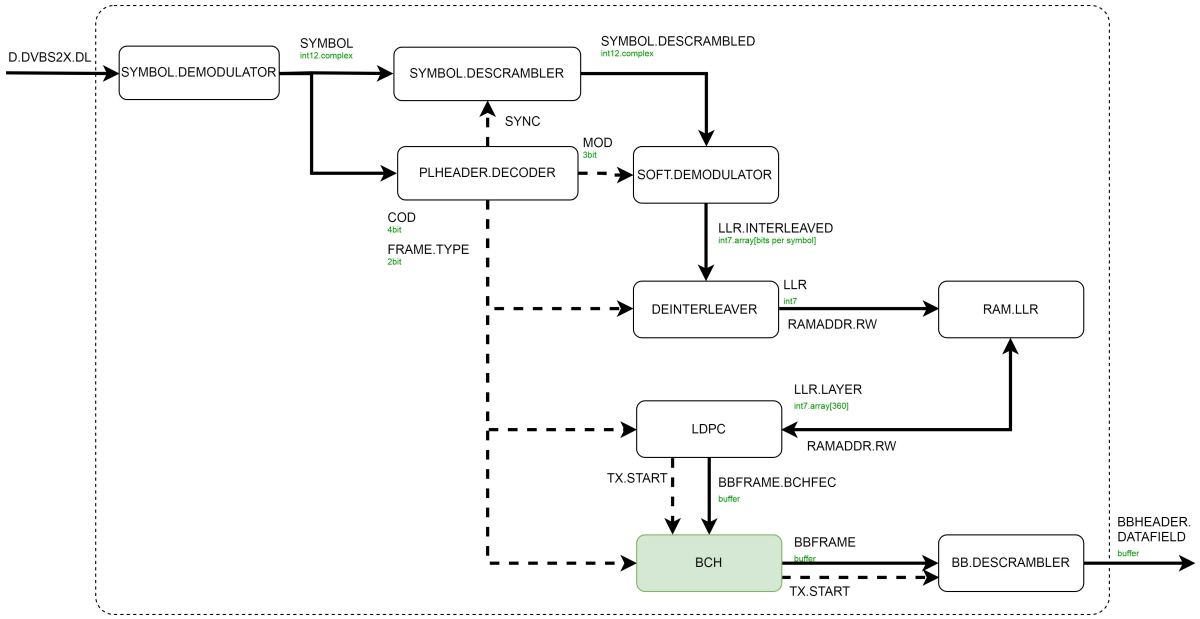

Figura 1 – Diagrama de blocos com as etapas de processamento realizados pelo terminal para decodificação no padrão DVB-S2X(ETSI, 2009).

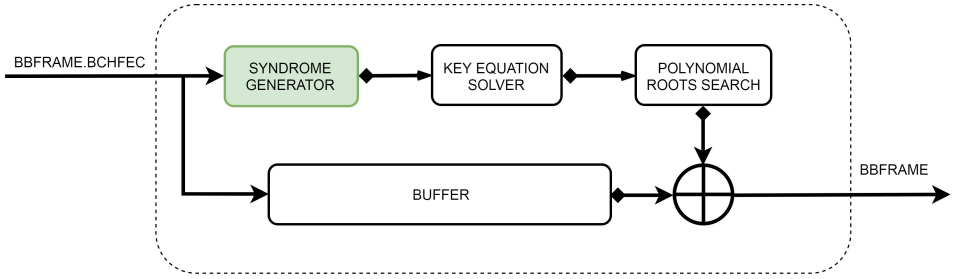

Figura 2 – Diagrama de blocos do decodificador BCH. (CHAVES; LIMA; MERTES, 2014)

1.2 Justificativa

Além da complexidade matemática, o desenvolvimento de decodificadores DVB-S2X enfrenta o desafio de determinar algoritmos de decodificação que possam ser executados com os recursos de *hardware* disponíveis, em um tempo compatível com as taxas de transmissão do sistema de comunicação. Devido aos possíveis tamanhos para os *frames* do BCH no padrão DVB-S2X, o processamento paralelo de todos bits durante a decodificação exige muitos recursos computacionais. Enquanto que o processamento bit a bit torna o tempo de decodificação extenso, limitando a taxa de recepção do terminal. Muitos trabalhos têm sido desenvolvidos nos últimos em busca de soluções para este problema (VIRGOVIč, 2018) (MARCHAND; BOUTILLON, 2015) (ATTAR, 2015) (ALVES et al., 2014) (LIMA et al., 2014).

FPGA é uma tecnologia considerada para o desenvolvimento de receptores DVB-S2X, pois as linguagens de descrição de *hardware* permitem um desenvolvimento rápido e de baixo custo. A implementação de um *hardware* dedicado não limita o potencial dos algoritmos. A possibilidade de realizar teste em *hardware*, sem a necessidade do desenvolvimento de uma PCB (do inglês, *Printed Circuit Board*), diminui os custos do projeto e torna o processo mais rápido e criativo, viabilizando que mais propostas de soluções sejam testadas. Além disso, a implementação em FPGA permite explorar o paralelismo intrínseco dos algoritmos, visando o aumento do desempenho.

1.3 Objetivos

O presente trabalho visa o estudo e a implementação em linguagem de descrição de hardware (VHDL) de geradores de síndromes para decodificadores BCH, segundo o padrão DVB-S2X. Os objetivos específicos são:

- Desenvolvimento de um modelo de referência em Matlab de um decodificador BCH que inclua a descrição de um multiplicador de Galois com ordem de campo parametrizável e de geradores de síndromes de entrada serial e paralela.

- Implementação em FPGA de uma arquitetura de *hardware* de um multiplicador de Galois parametrizável para os campos que o padrão DVB-S2X determina.
- Implementação em FPGA de uma arquitetura de *hardware* de um gerador de síndromes de entrada serial que atenda o padrão DVB-S2X.
- Implementação em FPGA de uma arquitetura de *hardware* de um gerador de síndromes de entrada paralela que atenda o padrão DVB-S2X.

1.4 Contribuições do Trabalho

Este trabalho, como parte do projeto de um RDS, busca disponibilizar um modelo de referência em alto nível da implementação dos algoritmos de geradores de síndromes de entrada serial e paralela, utilizados no processo de decodificação do BCH. Além disso, visa a realização de um estudo comparativo das arquiteturas em termos da máxima frequência de operação, do consumo de energia e do consumo de recursos, tais como *Look Up Tables*, *Flip Flops*, DSPs, blocos de RAM e multiplexadores. Os resultados obtidos constituem parte do processo de desenvolvimento do rádio definido por *software* no padrão DVB-S2X.

1.5 Organização do Trabalho

Este trabalho está organizado em cinco capítulos. O primeiro capítulo, Introdução, busca contextualizar o cenário de problematização e apresentar uma proposta de solução. O segundo capítulo, Fundamentação Teórica, visa a definição de conceitos básicos que são necessários para a compreensão do desenvolvimento e objetivos do trabalho. O terceiro capítulo, Metodologia, explica os métodos aplicados para garantia de qualidade e as ferramentas utilizadas no trabalho, assim como o desenvolvimento do mesmo. O quarto capítulo, Resultados, apresenta os dados obtidos durante a implementação do trabalho e a execução de testes. Por fim, o quinto capítulo, Conclusões, busca realizar discussões dos assuntos tratados, bem como uma análise dos resultados obtidos e a projeção para trabalhos futuros.

2 Fundamentação Teórica

2.1 Sistemas de Comunicação Digital

A teoria de comunicação trata a informação como uma quantidade física, que pode ser mensurada, transformada, armazenada e transmitida de um local a outro. Tal informação pode ser medida pela quantidade de incerteza envolvida e descrita matematicamente por estimativas probabilísticas (MOON, 2005).

O conceito de comunicações digitais engloba a transmissão de informação em formato digital de uma fonte geradora a um ou mais destinatários. As características físicas do canal pelo qual a informação é transportada são de suma importância durante a análise e o desenvolvimento de um sistema de comunicação (PROAKIS; SALEHI, 2007).

Um diagrama genérico composto por elementos básicos de um sistema de comunicação digital é apresentado na Figura 3.

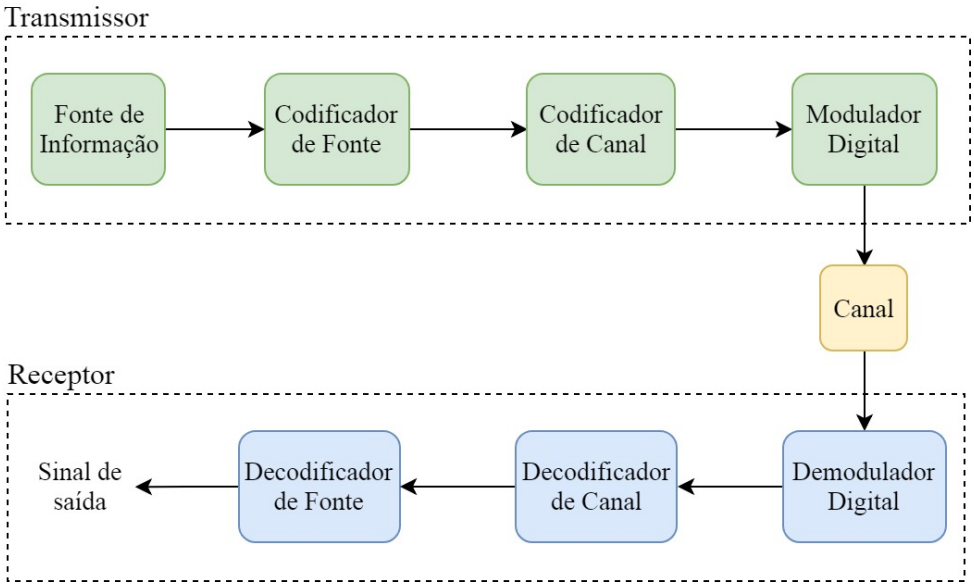

Figura 3 – Diagrama genérico de um sistema de comunicação digital. (PROAKIS; SALEHI, 2007)

A mensagem produzida pela fonte é representada, de forma eficiente, em uma sequência de dígitos binários, denominada sequência de informação. O bloco codificador de fonte é o responsável por realizar o processo de conversão do sinal em um sequência binária, denominada codificação ou compressão de dados.

O codificador do canal é responsável por introduzir controladamente redundâncias na sequência de informação. Os dados redundantes podem ser utilizados no receptor para

corrigir possíveis erros causados por ruídos e interferências do canal de transmissão.

O modulador digital recebe a palavra-código e realiza a interface com o canal de transmissão, mapeando a sequência de informação binária em formas de onda. O canal de comunicação é o meio físico de transmissão que é utilizado para enviar a informação do transmissor para o receptor.

No receptor, o demodulador digital processa a forma de onda corrompida pelo canal e a reduz em uma sequência de números que representam estimativas dos símbolos transmitidos. A partir do conhecimento do código utilizado pelo codificador de canal e a redundância contida na mensagem recebida, o decodificador de canal utiliza essa sequência de números para reconstruir a sequência de informação original.

Em seguida, o decodificador de fonte realiza a reconstrução da mensagem original da fonte, tendo conhecimento do método utilizado no processo de codificação da fonte. Devido aos possíveis erros de decodificação e distorções introduzidas pelo codificador, a saída do sinal decodificado pode se tratar de uma aproximação da informação original.

2.2 Codificação do Canal

Ao transmitir a informação digital, interferências e ruídos intrínsecos do canal de transmissão podem degradar o sinal, adicionando erros. O módulo responsável por manter as taxas de erros dentro do limite máximo aceitável é o codificador de canal.

O uso de codificadores para o controle eficiente das taxas de erros de um sistema de comunicação digital foi demonstrado por Claude Elwood Shannon. Considerando que o único fator degradante de um canal de transmissão seja um Ruído Gaussiano Branco Aditivo (AWGN, do inglês *Additive White Gaussian Noise*), a capacidade do canal, também conhecida como limite de Shannon, é dada pela Equação 2.1. Onde P representa a potência média do sinal transmitido, N_0 é a densidade espectral de potência do ruído e W é a largura de banda (PROAKIS; SALEHI, 2007).

$$C = W \cdot log_2 \left(1 + \frac{P}{N_0 \cdot W}\right) [bits/s] \tag{2.1}$$

Segundo Shannon, se a taxa de transmissão da informação for menor ou igual à capacidade do canal, existe um código corretor de erros que possibilita a comunicação pelo canal com uma probabilidade de erro tão baixa quanto desejável. Porém, se a taxa de transmissão é maior do que a capacidade do canal, códigos corretores não poderão alcançar um desempenho confiável (SHANNON, 1948).

Entre os métodos mais utilizados para o processo codificação e decodificação de canal, estão o *Backward Error Correction* (BEC) e o *Forward Error Correction* (FEC). Um sistema de decodificação do tipo BEC realiza apenas a detecção do erro e solicita ao

transmissor o reenvio da mensagem. Apesar de simples, o método exige uma comunicação do tipo *duplex* e gera um atraso na transmissão. Um sistema de decodificação do tipo FEC possui a capacidade de identificar e corrigir uma determinada quantidade de erros, sem a necessidade de requisitar a retransmissão da mensagem (MENGARDA, 2016).

Nos sistemas do tipo FEC, o processo de codificação do canal consiste em utilizar uma mensagem composta por um conjunto de k bits de informação e mapeá-los em uma sequência binária de n bits, denominada palavra-código, conforme apresentado na Figura 4. A razão entre a quantidade de bits da palavra codificada e a mensagem original k/n é denominada taxa de código. Os n-k bits acrescidos são denominados bits de paridade (JAMRO, 1997).

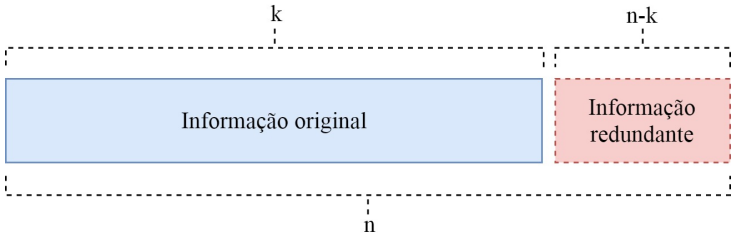

Figura 4 – Representação de uma palavra-código de n bits, composta por uma mensagem de k bits concatenados com n-k bits de paridade.

Os códigos corretores de erro (ECC, do inglês *Error Correcting Codes*) são blocos de extrema importância nos sistemas modernos de comunicação. Em diversas aplicações, grande parte do processamento de sinais em banda base é dedicada aos ECC. Entre as aplicações dos código corretores, estão inclusas as comunicações espaciais e satelitais, as comunicações móveis, a transmissão e o armazenamento de dados.

2.3 Padrão DVB-S2X

O Projeto DVB, fundado em 1993, é uma associação liderada pela indústria de emissoras, operadores de rede, desenvolvedores de *software*, órgãos reguladores, entre outras pessoas comprometidas com o desenvolvimento de padrões globais para a prestação de serviços de dados e televisão digital. O *Digital Video Broadcasting - Satellite* (DVB-S) foi a primeira versão de especificação desse padrão e tinha como principal funcionalidade oferecer serviços de televisão *Direct-To-Home* (DTH) (ETSI, 1997).

A segunda versão *Digital Video Broadcasting - Satellite Second Generation* (DVB-S2) foi desenvolvida visando a melhoria da eficiência nos serviços de comunicação via satélite. Alguns recursos foram incrementados, tais como codificações e modulações adaptativas e variáveis. Também foi incorporada a codificação do tipo FEC, concatenando um código *Low-Density Parity Check* (LDPC) com um código Bose–Chaudhuri–Hocquenghem (BCH). Essa técnica possui alta eficiência, latência relativamente reduzida e pode atingir

taxas de transmissão próximas à capacidade do canal (limite de Shannon) (MORELLO; MIGNONE, 2006).

O padrão DVB-S2 possui uma interface adaptativa, permitindo o uso de diferentes combinações de modulação e codificação. A utilização desses recursos ocorre quando o sistema está operando no modo *Adaptive Coding and Modulation* (ACM). Neste modo, o sistema é capaz de se adaptar às condições do canal de transmissão. Para que a adaptação do sistema ocorra, torna-se necessário que sejam disponibilizadas informações a respeito das condições físicas do canal de transmissão. O envio destas informações é realizado pelo canal de retorno do sistema de comunicação (ETSI, 2009).

A arquitetura do sistema DVB-S2 é apresentada na Figura 5. Os blocos pontilhados se referem aos sub-sistemas que são relevantes apenas para aplicações com múltiplas entradas de dados.

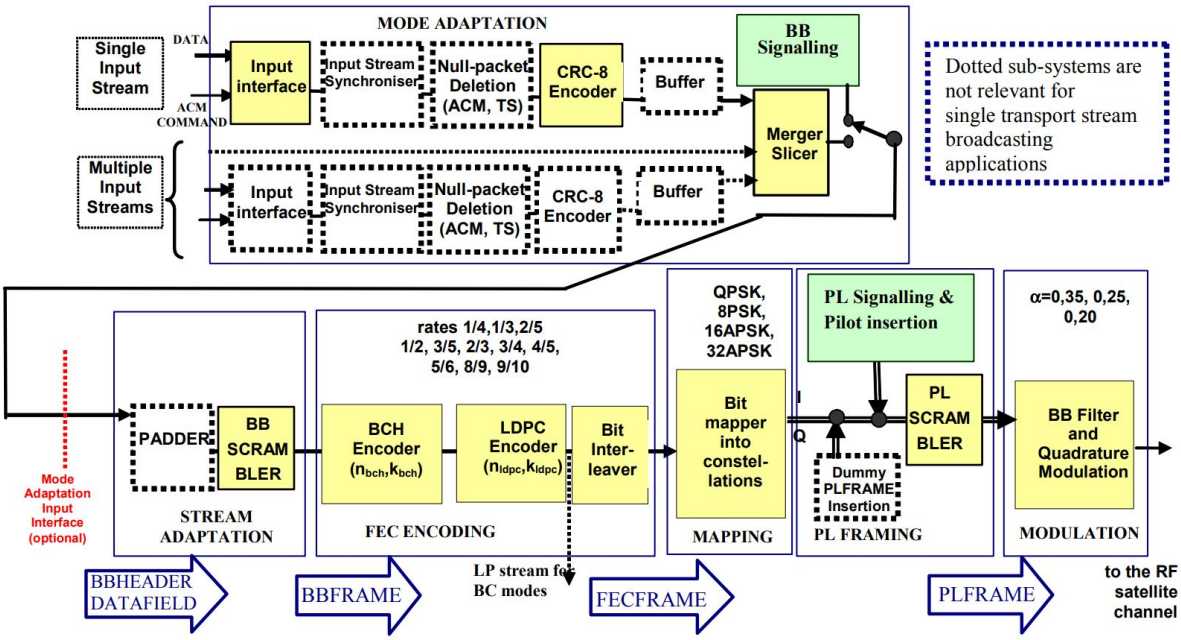

Figura 5 – Diagrama de blocos funcional do sistema DVB-S2 (ETSI, 2009).

A seguir há uma breve descrição dos principais blocos do sistema DVB-S2: *Mode Adaptation*, *Stream Adaptation*, *FEC encoding*, *Mapping*, *PL Framing* e *Modulation*.

- *Mode Adaptation*: A interface de entrada interpreta e realiza a conversão dos sinais de entrada em sinais digitais, considerando o primeiro bit como o mais significativo (MSB, do inglês *Most Significant Bit*). A entrada de sinalização *ACM Command* permite a configuração dos parâmetros de transmissão a serem adotados na modulação.

O bloco sincronizador de fluxo de entrada fornece meios adequados para garantir uma taxa constante de bits (CBR, do inglês *Constant Bit Rate*) e um atraso constante de transmissão para fluxos de entrada de pacotes. Os pacotes nulos são identificados e removidos para reduzir a taxa de informações e aumentar a proteção contra erros no modulador. O processo é realizado de forma que os pacotes nulos removidos possam ser reinseridos no receptor onde estavam originalmente.

O codificador CRC-8 realiza a detecção dos erros em nível de pacotes recebidos. O *slicer* realiza a leitura de um campo de dados (*DATAFIELD*) de comprimento DFL (do inglês *Data Field Length*). O *merger* concatena, em uma saída, diferentes campos de dados lidos e fatiados de uma de suas entradas. Caso haja apenas um fluxo, somente a funcionalidade de fatiamento é aplicada.

A saída do bloco *Mode Adaptation* é composta por uma sequência de 80 bits que informam o campo *BBHEADER*, seguido do campo de dados. O *BBHEADER* é o cabeçalho de banda base inserido pelo bloco *BB Signalling*. Este cabeçalho identifica o formato que estão os dados no pacote.

- *Stream Adaptation*:

  O processo de adaptação do conjunto de informações (*BBHEADER* e *DATAFIELD*) é realizado para conformar o tamanho do *frame* para K_{bch} bits. Este é o tamanho padrão utilizado para o *frame* de entrada do código BCH, que varia de acordo com a taxa de código e o tipo de *frame* escolhido pelo sistema de adaptação. Portanto, torna-se necessária a inserção de um conjunto de zeros ao final do pacote de dados, denominado *PADDING*.

  O bloco *BB Scrambler* é responsável por embaralhar os dados do pacote, incluindo *BBHEADER*, *DATAFIELD* e *PADDING*, utilizando um polinômio gerador de sequência pseudo-aleatória (PRBS, do inglês *Pseudo Random Binary Sequence*).

- *FEC encoder*:

  O sistema FEC, responsável pela codificação do canal, é composto por um BCH concatenado com um LPDC, seguidos de um intercalador de bits. Ambos os códigos possuem diferentes taxas de código, que são adaptadas de acordo com as características do canal. A combinação dos códigos BCH e LPDC permite uma correção efetiva e robusta da informação transmitida. O funcionamento do código BCH será descrito com maiores detalhes na seção 2.4. Após o LDPC, a informação é encaminhada ao bloco que realiza a intercalação dos bits do *frame*, distribuindo os bits serialmente em colunas e realizando a leitura por linhas.

- *Mapping* e *PL Framing*:

  De acordo com as características do canal, os bits são mapeados em uma das constelações estipuladas pelo padrão. Na camada física, os dados são representados em

conjuntos de blocos, contendo 90 símbolos cada. Estes símbolos são a representação do dado binário na constelação em que foram mapeados. Aplica-se um cabeçalho com modulação fixa em $\pi/2$ BPSK, contendo as informações dos processos de codificação e modulação, para que o receptor seja capaz de demodular e decodificar a mensagem recebida.

- *PL Scrambler* e *Modulation*:

  Antes de realizar a transmissão do sinal, cada *PLFRAME*, excluindo os cabeçalhos, deve ser randomizado pelo *PL Scrambler*. Este processo consiste em multiplicar cada amostra por uma sequência de randomização. Após esse processo, os sinais são filtrados pela raiz quadrada de um cosseno levantado. De acordo com os requisitos da aplicação, o fator de *roll-off* é ajustado entre 0.35, 0.25 e 0.2.

  Após a filtragem em banda base, a modulação em quadratura será realizada a partir da multiplicação das amostras em fase e em quadratura por $sin(2\pi f_0 t)$ e por $cos(2\pi f_0 t)$, respectivamente, onde f_0 se refere à frequência da portadora. Os dois sinais resultantes devem ser adicionados para obter o sinal de saída do modulador.

O padrão *Digital Video Broadcasting - Satellite Second Generation Extension* (DVB-S2X) é uma extensão das especificações técnicas do DVB-S2. A especificação DVB-S2X reutiliza a arquitetura do DVB-S2, apresentada na Figura 5, adicionando passos de modulação e codificação mais refinados, filtragem mais precisa com fatores de *roll-off* de até 0.05.

Além das principais áreas de aplicação do DVB-S2 (transmissão digital de vídeo, *forward link* para serviços interativos utilizando ACM, aquisição eletrônica de notícias via satélite, etc), o padrão DVB-S2X possui como alvo as novas áreas de aplicação que exigem relação sinal-ruído muito baixa (VL-SNR, do inglês *Very-Low Signal to Noise Ratio*) (ETSI, 2014).

2.4 Código BCH

As operações matemáticas utilizadas nos processos de codificação e decodificação dos códigos BCH pertencem ao domínio dos campos finitos, ou campos de Galois. Portanto, a compreensão dos processos realizados pelo código BCH requer o entendimento dos conceitos básicos da teoria destes campos.

2.4.1 Campos de Galois

Um campo consiste de um conjunto de elementos dos quais é possível realizar operações aritméticas e obter elementos do mesmo. Seja F um campo composto por

um conjunto de elementos, com duas operações denominadas adição e multiplicação e denotadas por '$\oplus$' e '$\otimes$', respectivamente (JAMRO, 1997).

Considerando λ e φ elementos do campo F, tal que $\gamma = \lambda \oplus \varphi$ e $\delta = \lambda \otimes \varphi$, onde γ e δ também pertencem ao campo. Para que F seja considerado de fato um campo, as seguintes propriedades precisam ser satisfeitas:

- Associativa: para todo λ, φ, $\gamma \in F$

$$\begin{aligned} \lambda \oplus (\varphi \oplus \gamma) &= (\lambda \oplus \varphi) \oplus \gamma \\ \lambda \otimes (\varphi \otimes \gamma) &= (\lambda \otimes \varphi) \otimes \gamma \end{aligned} \tag{2.2}$$

- Identidade: existe um elemento identidade para a adição e um para a multiplicação tal que, para todo $\lambda \in F$, as seguintes operações são satisfeitas:

$$\begin{aligned} 0 \oplus \lambda &= \lambda \oplus 0 = \lambda \\ \lambda \otimes 1 &= 1 \otimes \lambda = \lambda \end{aligned} \tag{2.3}$$

- Inversa: se $\lambda \in F$, existem elementos φ e $\gamma \in F$, tais que:

$$\begin{aligned} \lambda \oplus \varphi &= 0 \\ \lambda \otimes \gamma &= 1 \end{aligned} \tag{2.4}$$

  O elemento φ é denominado inverso aditivo de λ, visto que $\varphi = (\text{-}\lambda)$. O elemento γ é denominado inverso multiplicativo de λ, dado que $\gamma = \lambda^{-1}$ $(\lambda \neq 0)$.

- Comutativa: para todo $\lambda \in F$, as equações a seguir são satisfeitas.

$$\begin{aligned} \lambda \oplus \varphi &= \varphi \oplus \lambda \\ \lambda \otimes \varphi &= \varphi \otimes \lambda \end{aligned} \tag{2.5}$$

- Distributiva: para todo λ, φ e γ pertencentes ao campo F, as seguintes equações são satisfeitas.

$$(\lambda \oplus \varphi) \otimes \gamma = (\lambda \otimes \gamma) \oplus (\varphi \otimes \gamma) \tag{2.6}$$

A existência de um elemento multiplicativo inverso possibilita a operação de divisão no campo. Para $\lambda, \varphi, \gamma \in F$, $\gamma = \varphi/\lambda$ é definido como $\gamma = \varphi \otimes \lambda^{-1}$. De forma similar, a existência de um elemento aditivo inverso possibilita a operação de subtração. Para $\lambda, \varphi, \gamma \in F$, $\gamma = \varphi - \lambda$ é definido como $\gamma = \varphi \oplus (-\lambda)$.

É possível mostrar que o conjunto de inteiros {0, 1, 2, ..., p-1}, onde p é primo, forma um campo finito se suas operações são realizadas em módulo p e as propriedades mostradas pelas equações 2.2, 2.3, 2.4, 2.5 e 2.6 são satisfeitas. Este campo é denominado

campo finito de ordem p, ou GF(p) em homenagem a Evariste Galois (MACWILLIAMS; SLOANE, 1977).

Neste trabalho, será considerada apenas a aritmética onde p é igual a 2^m, pois a partir do GF(2) a representação dos elementos do campo finito é mapeada dentro do domínio digital, composto por elementos binários. A aritmética em GF(2) é definida como módulo 2 e, a partir dela, a extensão do campo GF(2^m) é gerada. Antes de introduzir o campo GF(2^m), algumas definições são necessárias.

Os polinômios sobre o campo GF(2) podem ser adicionados, subtraídos, multiplicados e divididos entre si. Considere *p(x)* um polinômio de grau m sobre o campo GF(2) que possui a forma apresentada pela Equação 2.7, em que os coeficientes p_i são elementos de GF(2) = {0, 1}. O polinômio *p(x)* de grau m sobre GF(2) é dito irredutível se este não for divisível por outro polinômio do mesmo campo de grau menor do que m e maior do que zero.

$$p(x) = p_0 + p_1 x + ... + p_m x^m \tag{2.7}$$

Para gerar o campo GF(2^m), é escolhido um polinômio mônico irredutível *p(x)* de grau m, sobre GF(2). Considere um conjunto, composto por 2^m polinômios de grau menor que m, sobre GF(2), formados a partir de *p(x)*. Pode-se provar que, se as operações realizadas com esses elementos possuem módulo *p(x)*, o conjunto forma um campo de 2^m elementos, denotado GF(2^m).

Um conjunto de m elementos linearmente independentes $\beta = \{\beta_0, \beta_1, ..., \beta_{m-1}\}$ de GF(2^m) é denominado base para GF(2^m). Portanto, qualquer elemento do campo pode ser representado como a soma ponderada das bases do campo, conforme descrito pela Equação 2.8. O elemento λ também pode ser denotado pelo vetor $(\lambda_0, \lambda_1, ..., \lambda_{m-1})$.

$$\lambda = \lambda_0\beta_0 + \lambda_1\beta_1 + ... + \lambda_{m-1}\beta_{m-1}, \qquad \lambda_i \in GF(2) \tag{2.8}$$

Considerando *p(x)* como polinômio irredutível de GF(2^m) e α como a raiz deste polinômio, então A = $\{1, \alpha, ..., \alpha^{m-1}\}$ é uma base polinomial para GF(2^m). Um polinômio irredutível de grau m é denominado polinômio primitivo se o menor inteiro positivo n para o qual *p(x)* é divisível por $x^n - 1$ for $n = 2^m - 1$. Se α é uma raiz de *p(x)*, onde esse polinômio não é apenas irredutível mas também primitivo, então GF(2^m) pode ser representado pelo conjunto de elementos GF(2^m) = $\{0, 1, \alpha, \alpha^2, ..., \alpha^{n-1}\}$, onde n=$2^m - 1$. Nesse caso, α é denominado elemento primitivo e $\alpha^n = 1$.

Nos campos GF(2^m), considerando que o elemento primitivo α é representado por 2, é possível satisfazer as propriedades apresentadas nas Equações 2.2, 2.3, 2.4, 2.5 e 2.6 definindo a adição como uma operação OU exclusivo (XOR, do inglês *Exclusive OR*) com os elementos do campo. Além disso, a operação de adição é equivalente à subtração dentro do campo. A multiplicação é realizada em módulo *p(x)*, que representa o polinômio

primitivo do campo. Considerando dois elementos do campo $\lambda(x) = (\lambda_{m-1}x^{m-1}+..+\lambda_2x^2+\lambda_1x+\lambda_0)$ e $\varphi(x) = (\varphi_{m-1}x^{m-1} + ... + \varphi_2x^2 + \varphi_1x + \varphi_0)$, o produto entre os polinômios é apresentado na Equação 2.9 (STEWART, 2003).

$$\delta(x) = \lambda(x) \otimes \varphi(x) = [\lambda(x) \times \varphi(x)] \quad mod \quad p(x) \tag{2.9}$$

A Equação 2.10 exemplifica o produto bit a bit entre os polinômios $\lambda(x)$ e $\varphi(x)$ de um campo GF(2^4).

$$\begin{aligned} \lambda(x) \times \varphi(x) =& (\varphi_3x^3 + \varphi_2x^2 + \varphi_1x^1 + \varphi_0)\lambda_0 \quad + \\ & (\varphi_3x^3 + \varphi_2x^2 + \varphi_1x^1 + \varphi_0)\lambda_1x^1 + \\ & (\varphi_3x^3 + \varphi_2x^2 + \varphi_1x^1 + \varphi_0)\lambda_2x^2 + \\ & (\varphi_3x^3 + \varphi_2x^2 + \varphi_1x^1 + \varphi_0)\lambda_3x^3 \end{aligned} \tag{2.10}$$

Assumindo que o polinômio primitivo deste campo é $p(x) = x^4 + x + 1$ e que os coeficientes dos polinômios são dados por $\lambda(x) = x^3 + x^2 + 1$ e $\varphi(x) = x^3 + 1$, o produto bit a bit é realizado da seguinte forma:

$$\begin{array}{cccccccc} & & & & 1 & 1 & 0 & 1 \\ & & & \times & 1 & 0 & 0 & 1 \\ \hline & & & & 1 & 1 & 0 & 1 \\ & & & 0 & 0 & 0 & 0 & \\ & & 0 & 0 & 0 & 0 & & \\ \oplus & 1 & 1 & 0 & 1 & & & \\ \hline & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{array} = x^6 + x^5 + x^2 + 1$$

O resultado do produto bit a bit é dividido pelo polinômio primitivo, de acordo com a Equação 2.11.

$$\frac{x^6 + x^5 + x^2 + 1}{x^4 + x + 1} = x^2 + x + \frac{-x^3 - x^2 - x + 1}{x^4 + x + 1} \tag{2.11}$$

O processo de multiplicação nos campos de Galois ocorre em duas etapas. Primeiramente, é realizada a multiplicação clássica com operações lógicas AND bit a bit, seguidas de operações XOR com os resultados parciais deslocados à esquerda. O produto entre os polinômios é dividido pelo polinômio primitivo. No exemplo apresentado acima, o resultado da divisão foi $x^2 + x$ e o resto $-x^3 - x^2 - x + 1$. Tendo em vista a equivalência das operações de adição e subtração nos campos de Galois, o polinômio resultante é da multiplicação é $\delta(x)$=$x^3 + x^2 + x + 1$, na representação polinomial, e $(1111)_2$, em binário.

2.4.2 Processo de codificação

Os códigos BCH são caracterizados pelos seguintes parâmetros: tamanho da palavra-código(n), tamanho da mensagem(k) e capacidade de correção de erros(t). Considere uma

mensagem $m = \{m_{k-1}, m_{k-2}, ..., m_1, m_0\}$. A codificação se baseia em gerar n-k bits de paridade e concatená-los com os bits da mensagem, gerando uma palavra código $c = (m_{k-1}, m_{k-2}, ..., m_1, m_0, d_{n-k-1}, d_{n-k-2}, ..., d_1, d_0)$.

O processo de codificação é inicializado com a multiplicação da mensagem $m(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + ... + m_1x, m_0$, representada como polinômio, por x^{n-k}. Esse procedimento é equivalente a realizar o deslocamento à esquerda de n-k bits. O passo seguinte consiste em dividir $x^{n-k}m(x)$ pelo polinômio gerador $g(x)$. O resto da divisão corresponde aos bits de paridade, denotados por $d(x) = d_{n-k-1}x^{n-k-1} + d_{n-k-2}x^{n-k-2} + ... + d_1x + d_0$. Então, a palavra-código é representada por $c(x) = x^{n-k}m(x) + d(x)$ (GAUTAM; TANDEL, 2010).

2.4.3 Processo de decodificação

O processo de decodificação pode ser dividido em três processos principais: calcular as síndromes, solucionar a equação-chave e encontrar a localização dos erros. Considere que o decodificador tenha recebido a informação $r(x)$, tal que:

$$r(x) = c(x) + e(x) \tag{2.12}$$

Onde $c(x)$ representa a palavra-código enviada e $e(x)$ representa os erros inerentes do canal, que foram adicionados à informação transmitida. A representação polinomial destes elementos é apresentada na Equação 2.13.

$$\begin{aligned} r(x) &= r_0 + r_1x + r_2x^2 + ... + r_{n-1}x^{n-1} \\ c(x) &= c_0 + c_1x + c_2x^2 + ... + c_{n-1}x^{n-1} \\ e(x) &= e_0 + e_1x + e_2x^2 + ... + e_{n-1}x^{n-1} \end{aligned} \tag{2.13}$$

O primeiro passo no processo de decodificação é armazenar os primeiros k bits da informação recebida $r(x)$ em um registrador (*buffer*) e iniciar o cálculo das síndromes. Para uma capacidade de correção de t erros do código BCH, são geradas $2t$ síndromes. Considere a definição das síndromes apresentada na Equação 2.14.

$$S_j = \sum_{i=0}^{n-1} r_i\alpha^{ij} = r_0 + r_1\alpha^j + r_2\alpha^{2j} + ... + r_{n-1}\alpha^{(n-1)j} \tag{2.14}$$

Dado que $r_i = c_i + e_i$, para $0 \leq i \leq (n-1)$, as síndromes podem ser representadas, conforme a Equação 2.15.

$$S_j = \sum_{i=0}^{n-1}(c_i + e_i)\alpha^{ij} = \sum_{i=0}^{n-1} c_i\alpha^{ij} + \sum_{i=0}^{n-1} e_i\alpha^{ij} \tag{2.15}$$

Pela definição dos códigos BCH (JR.; CAIN, 1981), tem-se que:

$$\sum_{i=0}^{n-1} c_i\alpha^{ij} = 0 \tag{2.16}$$

Logo, a Equação 2.15 é reduzida à Equação 2.17.

$$S_j = \sum_{i=0}^{n-1} e_i \alpha^{ij} \tag{2.17}$$

Portanto, o cálculo das síndromes depende apenas dos erros inseridos *e(x)* e não da informação transmitida *c(x)*. Se não houver erros na transmissão, as síndromes terão valor igual a zero.

O segundo passo do processo de decodificação consiste em determinar os coeficientes do polinômio localizador de erros $\sigma(\mathrm{x}) = \sigma_0 + \sigma_1 x + ... + \sigma_t x^t$, a partir das síndromes calculadas no processo anterior. A relação entre as síndromes e os valores de σ_j é dado pela Equação 2.18. As posições dos erros na informação recebida *r(x)* são determinadas pelas raízes de $\sigma(\mathrm{x})$.

$$\sum_{j=0}^{t} S_{t+i-j}\sigma_j = 0 \qquad para \quad 1 \leq i \leq t \tag{2.18}$$

Existem algoritmos que foram elaborados para determinar os coeficientes de $\sigma(\mathrm{x})$. Alguns exemplos são: algoritmo de Peterson-Gorenstein-Zieler (PETERSON; JR, 1972), Euclidiano (SUGIYAMA et al., 1975) e Berlekamp-Massey Algorithm (BERLEKAMP, 1973).

O passo seguinte da decodificação consiste em determinar a localização dos erros, ou seja, calcular as raízes do polinômio localizador de erros. Esse processo é realizado a partir da substituição dos valores que representam as potências do elemento primitivo do campo $(1, \alpha, \alpha^2, ..., \alpha^{n-1})$ no polinômio $\sigma(\mathrm{x})$.

Chien apresenta uma solução para essa etapa utilizando substituição sequencial (CHIEN, 1964). No algoritmo de Chien, a soma $\sigma_0 + \sigma_1\alpha^j + \sigma_2\alpha^{2j} + ... + \sigma_t\alpha^{tj}$, para $0 \leq$ j $\leq$ (k-1), é verificada a cada iteração. Se $\sigma(\alpha^j)$=0, significa que o bit r_{n-j-1} da informação recebida está corrompido e deve ser corrigido. Finalmente, o processo de correção consiste em realizar a operação XOR da saída deste bloco com a mensagem armazenada no *buffer*.

2.5 BCH no padrão DVB-S2X

O DVB-S2X especifica os polinômios primitivos utilizados nos processos de codificação e decodificação do BCH, a capacidade de correção de erros e os tamanhos possíveis para a mensagem e para a palavra código, de acordo com a taxa de código e o tipo de *frame* (ETSI, 2014). Para cada tamanho de mensagem, existe um tamanho de palavra-código correspondente. A palavra-código é composta pela mensagem (*BBFRAME*) concatenada com os bits de paridade (BCHFEC), conforme ilustrado na Figura 6.

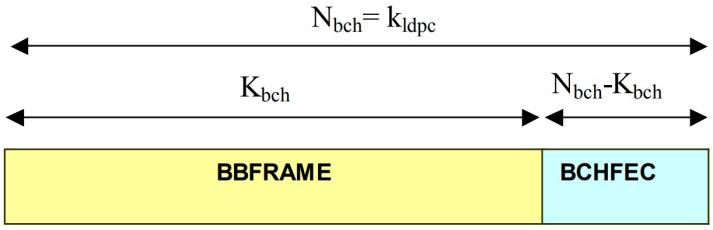

Figura 6 – Disposição dos dados após o processo de codificação do BCH(ETSI, 2009).

O código BCH, segundo o padrão DVB-S2X, possui capacidade máxima de correção de 12 bits. Este bloco opera com três tipos de frame: *short*, *medium* e *normal*. Os *short frames* operam no campo GF(2^{14}), os *medium frames* operam no campo GF(2^{15}) e os *normal frames* operam no campo GF(2^{16}). Para cada um dos tipos de *frame*, de acordo com a taxa de código, existe um tamanho específico para a palavra-código (N_{bch}) e um tamanho para a mensagem (K_{bch}).

Os polinômios primitivos utilizados nos processos de codificação e decodificação para *short, medium* e *normal frames* estão apresentados no Apêndice A. Os tamanhos N_{bch} e K_{bch}, de acordo com os tipos de *frames* e com as taxas de códigos descritas pelo padrão, estão apresentados no Apêndice B.

Como os tamanhos dos *frames* são menores do que o tamanho padrão (n=$2^m - 1$), onde m representa a ordem do campo de Galois, os códigos BCH que seguem o padrão DVB-S2X são considerados códigos encurtados (do inglês, *shortned BCH codes*) (CHAVES; LIMA; MERTES, 2014).

2.6 Hardware Reconfigurável

Os dispositivos de lógica programável (PLD, do inglês *Programmable Logic Device*) geralmente são utilizados para fins de prototipagem rápida, permitindo a modelagem de um circuito digital a partir de uma linguagem de descrição de *hardware*. As camadas que compõem o circuito integrado dos PLDs existem desde sua fabricação, o processo de descrição realiza a conexão entre os blocos lógicos existentes.

Um tipo complexo de PLD é o FPGA (do inglês *Field Programmable Gate Array*), que oferece uma conectividade genérica entre os blocos lógicos. Devido a flexibilidade e a possibilidade de projetar um sistema com paralelismo de *hardware*, os FPGAs são ideais para executar tarefas de processamento de sinais em alta performance e de múltiplas entradas (RAJ, 2018).

A Figura 7 apresenta um exemplo de arquitetura de FPGAs da família Spartan-3E da Xilinx. Como pode ser observado, esse sistema é composto por diversos blocos de lógica configurável (CLBs, do inglês *Configurable Logic Blocks*), barramentos de entradas

e saídas de blocos (IOBs, do inglês *Input/Output Blocks*), blocos de multiplicação e blocos de RAM.

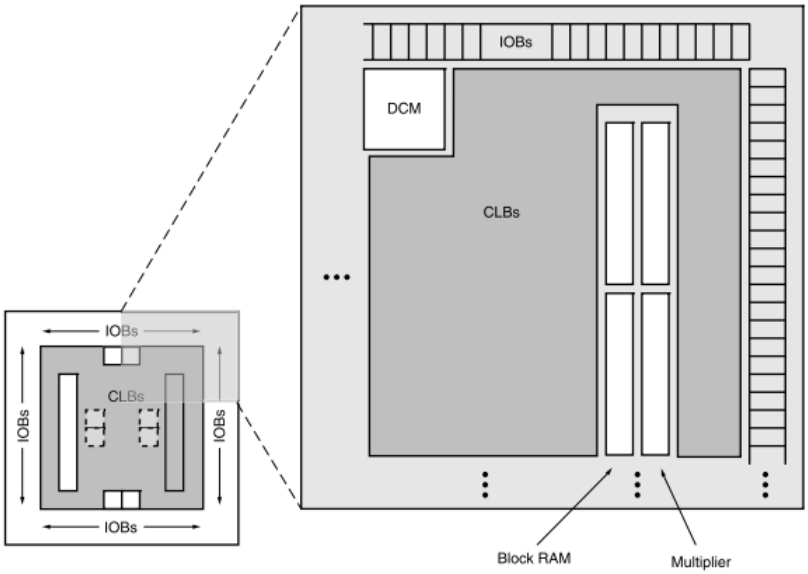

Figura 7 – Arquitetura da família Spartan-3E. (XILINX, 2013).

O projeto do rádio definido por *software* no padrão DVB-S2X, que contém como um dos blocos principais o decodificador BCH, será realizado na plataforma Adalm Pluto, apresentada na Figura 8. Essa plataforma se baseia da junção de um transceptor de radiofrequência da *Analog Devices AD96363* com o Zynq Z-7010 XC7Z010-1CLG225C4334 da Xilinx (DEVICES, 2017). Zynq é um sistema em chip (SoC, do inglês *System on Chip*) programável que combina um processador ARM Cortex-A9 (*Advanced RISC Machine*) com um *Field Programmable Gate Array* (FPGA). O ARM Cortex-A9 é um processador capaz de executar um sistema operacional. A FPGA é baseada na série de arquiteturas Xilinx 7 (CROCKETT et al., 2014).

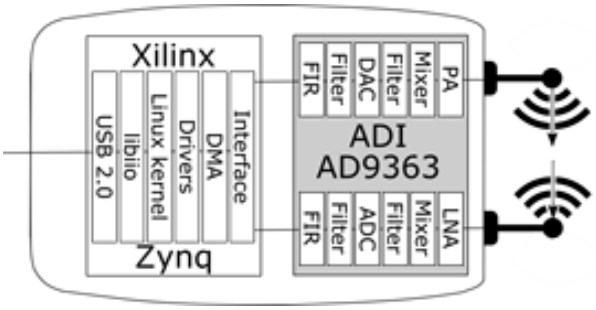

Figura 8 – Arquitetura da plataforma Adalm Pluto (DEVICES, 2017).

2.7 Estado da Arte

Esta seção visa a apresentação de resultados obtidos em trabalhos que tiveram como tema a implementação em FPGA dos algoritmos utilizados no processo de decodificação do BCH. Entretanto, estão apresentadas implementações para outras aplicações, além da que foi detalhada para sistemas de comunicação satelital no padrão DVB-S2X. Os resultados referentes aos decodificadores BCH estão dispostos na Tabela 1.

O trabalho *"A method to handle BCH(n,k,t) algorithm over large GF(n) in pratical hardware implementations"*, realizado por Alexandra Stanciu, Titus Iulian Ciocoiu e Florica Moldoveanu, apresenta uma abordagem para utilização de elementos dos campos de Galois aplicada à otimização de um código BCH. A implementação em FPGA foi realizada buscando o mínimo custo de área. Os testes da arquitetura, implementada em um dispositivo da família Virtex 4-XC4VSX35, resultaram em um consumo de 4320 *Flip Flops* (FFs), 8551 *Look Up Tables* (LUTs) e 12 blocos de memória RAM (BRAMs) (STANCIU; CIOCOIU; MOLDOVEANU, 2015).

Em *"A Synthesizable BCH Decoder for DVB-S2 Satellite Communications"*, desenvolvido por Cesar G. Chaves, Eduardo R. de Lima e Jacqueline G. Mertes, é apresentado o projeto de um decodificador BCH para comunicações digitais via satélite, segundo o padrão DVB-S2. O protótipo do projeto foi implementado no dispositivo Stratix IV GX da Altera, com o qual pôde-se obter uma frequência máxima de operação de 100,93 MHz. Os recursos utilizados foram 11368 LUTs e 1250 FFs (CHAVES; LIMA; MERTES, 2014).

Botao Zhang, Dongpei Liu, Shixian Wang, Xucan Chen e Hengzhu Liu desenvolveram o trabalho *"Design and Implementation of Area-Efficient DVB-S2 BCH Decoder"* que visa a implementação de um decodificador BCH no padrão DVB-S2. O projeto foi avaliado em plataforma FPGA e em biblioteca ASIC. Os resultados obtidos mostraram que a área lógica do BCH implementado é 13% menor do que a ocupada por outros decodificadores existentes. A implementação foi realizada na plataforma de *hardware* Spartan-3 XC3S1500 da Xilinx, com a qual pôde-se obter uma frequência máxima de operação de 104 MHz e consumo de 2517 FFs, 4489 LUTs e 5 BRAMs (ZHANG et al., 2010).

O projeto *"High Speed DVB-S2 BCH code decoder & encoder VHDL source code overview"*, desenvolvido pela empresa ComBlock, apresenta a implementação em FPGA de um código BCH no padrão DVB-S2 para duas plataformas de *hardware*, uma da família Spartan 3E da Xilinx e outra da Virtex 5. A implementação do codificador com a Spartan 3E resultou em uma frequência máxima de operação de 83 MHz e um consumo de 258 FFs e 1268 LUTs. Para o decodificador, foi obtida uma frequência máxima de 73 MHz e um consumo de 5002 FFs, 7910 LUTs e 9 BRAMs. A implementação do codificador com a Virtex 5 teve como resultado uma frequência máxima de operação de 131 MHz e um consumo de 265 FFs e 964 LUTs. Para o decodificador, foi obtida uma frequência máxima

de 166 MHz e um consumo de 4995 FFs, 6919 LUTs e 5 BRAMs (COMBLOCK, 2009).

O trabalho *"FPGA implementation of a new BCH decoder used in Digital Video Broadcasting - Satellite - Second Generation (DVB-S2)"*, desenvolvido por El Habti Idrissi Anas, El Gouri Rachid, Ahmed Lichioui e Hlou Laamari, realiza a implementação em FPGA de um decodificador BCH para aplicações de comunicação satelital, segundo o padrão DVB-S2. É proposto um algoritmo simplificado para decodificação BCH, visando a redução da complexidade da implementação. Este algoritmo reduz o consumo de recursos e de energia, com uma porcentagem que pode atingir 32% do consumo observado no algoritmo básico. A implementação de um decodificador BCH(3240, 3072, 12), em um FPGA Spartan 3E XC3S500E-5FG320 da Xilinx resultou em um consumo de 296 FFs e 389 LUTs (ANAS et al., 2005).

A tese de mestrado *Study and Design of Architecture for BCH code encoder and decoder*, realizada por Anuradha A. Gautam e Komal B. Tandel, realiza o projeto de um código BCH(15,5). O projeto do codificador é baseado em *Liner-Feedback Shift Register* (LFSR) para divisão polinomial e o decodificador é baseado no algoritmo IBM. O FPGA utilizado para a implementação foi um Spartan 3E - XC3S1IN500. A implementação do codificador resultou em um consumo de 39 FFs e 29 LUTs. O decodificador obteve um consumo de 218 FFs e 560 LUTs (GAUTAM; TANDEL, 2010).

Tabela 1 – Resultados de implementações de algoritmos de decodificação BCH

Autor	**Ano**	**Tecnologia**	**Algoritmo**	**Consumo de Recursos**
A. Stanciu et al.	2015	Virtex 4	BCH(n,k,t)	FFs: 4320 LUTs: 8551 BRAMs: 12
C.G. Chave et al.	2014	Stratix IV	BCH DVB-S2	FFs: 1250 LUTs: 11368 *Clock*: 100,93 MHz
B. Zhang et al.	2010	Spartan 3E	BCH DVB-S2	FFs: 2517 LUTs: 4489 BRAMs: 5 *Clock*: 104 MHz
A. A. Gautam et al.	2010	Spartan 3E	BCH(15,5)	FFs: 218 LUTs: 560
ComBlock	2009	Spartan 3E	BCH DVB-S2	FFs: 5002 LUTs: 7910 BRAMs: 9 *Clock*: 73 MHz
ComBlock	2009	Virtex 5	BCH DVB-S2	FFs: 4995 LUTs: 6919 BRAMs: 5 *Clock*: 166 MHz
E.H.I. Anas et al.	2005	Spartan 3E	BCH DVB-S2	FFs: 296 LUTs: 389

3 Metodologia

O presente trabalho realiza o estudo comparativo entre duas arquiteturas de geradores de síndromes para decodificadores BCH. As implementações realizadas compõem parte de um sistema FEC do projeto de um rádio definido por *software* no padrão DVB-S2X. Para a organização e gerenciamento das tarefas relacionadas ao desenvolvimento do projeto, foi adotada a metodologia ágil SCRUM com *Sprints* semanais. Os primeiros *Sprints* foram dedicadas ao desenvolvimento do sistema FEC. No começo de cada *Sprint*, são realizadas reuniões para revisar os resultados do *Sprint* anterior e definir os itens que farão parte do *Sprint Backlog* (PRIES; QUIGLEY, 2010).

Inicialmente, foi realizado um planejamento para identificação dos processos realizados com o sinal recebido pelo decodificador FEC. Em seguida, foi definida a ordem de implementações dos subsistemas que realizam as etapas do processamento do sinal, de acordo com o grau de complexidade. O decodificador BCH foi especificado como um dos primeiros subsistemas a serem desenvolvidos. Para cada subsistema, foram definidas as tarefas necessárias para disponibilizá-lo validado.

O processo de desenvolvimento do trabalho teve como base o *Model-Based Design* (MBD). O MBD é uma abordagem centrada em modelos, em que são definidos modelos matemáticos para todos os componentes necessários ao funcionamento do sistema. Estes modelos são utilizados como especificação funcional, com a vantagem de não possuir ambiguidade na representação matemática. Outro benefício é o potencial de utilização destes modelos em metodologias de garantia da qualidade, mediante simulações, tal como a metodologia *Model In the Loop* (MIL), utilizada no desenvolvimento deste trabalho.

O MIL consiste em simular, em ambiente computacional, todos os componentes do sistema, representados por seus modelos matemáticos. Essa abordagem permite testar cada componente, antes de implementá-lo em *hardware*. Isto possibilita a verificação antecipada da sua conformidade com os requisitos do projeto, evitando retrabalhos durante a implementação em VHDL (AARENSTRUP, 2015).

Os modelos matemáticos dos componentes do sistema foram codificados com a utilização do *software* Matlab, cuja a linguagem permite a implementação rápida de algoritmos. Entretanto, não foram utilizados todos os recursos matemáticos disponibilizados pelo *software*. Buscou-se uma aproximação da implementação em linguagem de descrição de *hardware* para que a simulação considerasse fatores como quantidade de registradores e ciclos de *clock* exigidos pelos algoritmos. A adequação do código em alto nível também teve como objetivo servir de base para a implementação em VHDL. Portanto, os parâmetros, as operações matemáticas e as interfaces do blocos foram implementadas

considerando registradores, representados por vetores binários.

Após a implementação dos algoritmos, foi realizada uma sequência de simulações MIL para diversos casos de teste. Além de validar o algoritmo, a simulação fornece um arquivo de texto contendo os valores das entradas e as respectivas respostas do sistema. Esses arquivos são utilizados na etapa de verificação comportamental das implementações em linguagem de descrição de *hardware*.

Após a aprovação do algoritmo nas simulações MIL, foi realizada a codificação em VHDL. As arquiteturas implementadas foram submetidas a uma análise comportamental por meio de simulações, visando a verificação de conformidade. Essa análise consiste em instanciar as entidades como componentes em um arquivo de simulação. Neste arquivo, são criados sinais que são responsáveis por gerar estímulos nas entradas do sistema. Visando aprimorar a análise, os valores recebidos pelos sinais foram lidos do arquivo de texto gerado durante as simulações em alto nível. As respostas referente às entradas do circuitos proposto foram salvas em arquivos de texto para comparação com as saídas da implementação em alto nível.

As implementações dos códigos de simulação e VHDL, assim como o acompanhamento da evolução do trabalho, foram realizadas com o uso da ferramenta Git para versionamento de projetos. O Git é um sistema que armazena as mudanças feitas em arquivos ao longo de um período, visando a recuperação de versões específicas e o monitoramento das modificações ao longo do ciclo de vida do projeto (CHACON, 2009).

3.1 Requisitos

Após a realização de um estudo para a elaboração de estratégias de sincronização entre os subsistemas do decodificador FEC - DVB-S2X, foram definidos requisitos de interface e funcionamento dos sistemas. Os requisitos do decodificador BCH estão descritos a seguir.

- Reset assíncrono
  O decodificador deve retomar as condições inicias e mantê-las enquanto o sinal de reset estiver habilitado. Quando o sinal de reset assíncrono *a_reset_in* for acionado, o BCH deve permanecer no estado ocioso em que aguarda o início do recebimento do próximo *frame* para decodificação.

- Reconhecimento do início do *frame* e dos parâmetros de codificação
  O BCH deve iniciar o processo de decodificação quando um novo *frame* for recebido. Tendo em vista que o processo de decodificação depende do tamanho do *frame* e da taxa de código, quando o decodificador estiver ocioso aguardando o próximo *frame* e houver uma borda de subida no sinal *sync_in*, então o BCH deve ler as entradas

frame_type_in e *code_rate_in* para identificar a taxa de código e o tipo do frame que está sendo recebido. Deve-se assumir o estado em que aguarda que o *frame* esteja disponível para ser processado

- Aguardar disponibilidade dos bits
  As duas arquiteturas propostas para o gerador de síndrome se diferenciam pela forma em que os dados do *frame* são recebidos e processados pelo decodificador. O gerador de síndromes de entrada serial implica o recebimento do *frame* bit a bit. Enquanto o gerador de síndromes de entrada paralela requer que o BCH processe os bits em pacotes de 8 bits.

  Por segurança, a leitura da entrada de recebimento do *frame* deve ser feita apenas quando os sinais de entrada, provenientes do decodificador LDPC, estejam estáveis. Quando uma leitura for realizada, o decodificador deve aguardar que a entrada *ldpc_data_ready_in* seja acionada.

- Confirmação de leitura dos bits do *frame* recebido
  O decodificador LDPC realiza a atualização dos bits fornecidos ao BCH somente após a confirmação de recebimento dos bits previamente transmitidos. Quando o BCH realizar a leitura dos bits, um pulso no sinal de saída *read_ack_out* deve ser gerado para confirmar a leitura.

3.2 Modelo de Simulação

O modelo de simulação consiste da implementação em alto nível do sistema para verificação do funcionamento. A codificação foi baseada na modelagem matemática descrita pelas equações 2.9 e 2.14. Essa implementação poderia ser realizada em poucas linhas de código, utilizando funções do Matlab e operações matriciais. O Código 3.1 apresenta um exemplo de implementação em alto nível.

```
1 function [y] = gf_multiplier( x1, x2, order_field, prim_poly )
2    x1_gf = gf(x1, order_field, prim_poly);
3    x2_gf = gf(x2, order_field, prim_poly);
4    y_gf  = x1_gf .* x2_gf;
5    y = uint16(y_gf.x);
6 end
```

Código 3.1 – Exemplo de código em alto nível que realiza a multiplicação de polinômios nos campos de Galois.

No entanto, esse tipo de implementação não fornece uma base para a codificação em linguagem de descrição de *hardware*. Portanto, foi realizada uma adequação para que a implementação ficasse próxima do código de mais baixo nível, no qual são realizadas operações lógicas com registradores. O Código 3.2 exemplifica a adequação realizada aos códigos.

```
function [y] = gf_multiplier( x1, x2, prim_poly )
m = size(prim_poly,2) - 1;
sim_required_clock_count = 1 + m;
   for sim_clock_count = 1:sim_required_clock_count
      if sim_clock_count == 1
         x1_reg = x1(1:m);
         x2_reg = x2(1:m);
         prim_poly_reg = prim_poly;
         y = zeros(1,m);
      else
         shift_remainder  =   xor(left_shift(y),prim_poly_reg(2:end)&y(1));
         y = xor(x1_reg & x2_reg(1) , shift_remainder);
         x2_reg = left_shift(x2_reg);
      end
   end
end
```

Código 3.2 – Trecho de código em alto nível com adequação para se aproximar da implementação VHDL da multiplicação de polinômios nos campos de Galois.

Devido às características das operações realizadas nos campos de Galois, a implementação em alto nível do multiplicador e do gerador de síndromes não considerou o uso de funções para cálculo de produto de polinômios e resto de divisão. A implementação foi baseada em operações lógicas AND, XOR e deslocamento de bits de um vetor binário. O Código 3.3 apresenta um trecho do código em VHDL que realiza a mesma implementação dos Códigos 3.1 e 3.2.

```
when CALCULATING=>
   shift_product_v := polynomial_product( M_MAX_DVBS2X - 2 downto 0 ) & '0';

   if polynomial_product( gf_length - 1 ) = '1' then
      shift_product_v( LSB_PRIM_POLY_LENGTH - 1 downto 0 ) := shift_product_v(
          LSB_PRIM_POLY_LENGTH - 1 downto 0 ) xor lsb_prim_poly;
   end if;

   shift_remainder_v := shift_product_v and mask_polynomial;

   if second_polynomial( gf_length - 1 ) = '0' then
      polynomial_product <= shift_remainder_v;
   else
      polynomial_product <= shift_remainder_v xor first_polynomial;
   end if;

   second_polynomial <= second_polynomial( M_MAX_DVBS2X - 2 downto 0 ) & '0';
   counter <= counter - 1;
```

Código 3.3 – Trecho de código VHDL que realiza a multiplicação de polinômios nos campos de Galois.

3.3 Arquitetura do Multiplicador de Galois

O multiplicador de Galois é responsável por realizar o produto entre dois polinômios de um campo de Galois, segundo a equação 2.9. A ordem e o polinômio gerador do campo são inferidos pelo multiplicador em função do tamanho do *frame*, segundo o padrão DVB-S2X (ETSI, 2014), e apresentado no Apêndice A. A entidade do multiplicador é apresentada na Figura 9. A Tabela 2 descreve o significado de cada entrada e saída da entidade do multiplicador de Galois.

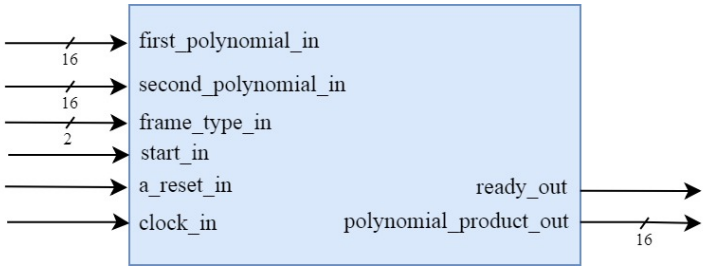

Figura 9 – Entidade do multiplicador de Galois para o padrão DVB-S2X.

Os processos do multiplicador são sincronizados pelo sinal *clock_in*. A cada subida de borda deste sinal, o bloco verifica se a entrada *start_in* está em nível lógico alto para inicializar do processo de multiplicação. Em seguida, o multiplicador realiza a leitura das entradas *first_polynomial_in*, *second_polynomial_in* e *frame_type_in*. A partir dessa leitura, o tipo do *frame* é verificado e, consequentemente, a ordem do campo de Galois.

De acordo com a ordem do campo, o multiplicador seleciona o respectivo polinômio primitivo e configura o contador que armazena o número de iterações. Ao final da operação, o sinal *ready_out* é configurado com nível lógico alto e o resultado é disponibilizado na saída *polynomial_product_out*.

Tabela 2 – Descrição das entradas e saídas do multiplicador de Galois

$N°$ **de bits**	**Nome**	**Descrição**
1	clock_in	Sinal de *clock*.
1	a_reset_in	Reset assíncrono.
1	start_in	Entrada para solicitação de início de multiplicação.
2	frame_type_in	Tipo do *frame*.
16	first_polynomial_in	Primeiro fator da multiplicação.
16	second_polynomial_in	Segundo fator da multiplicação.
16	polynomial_product_out	Resultado da multiplicação.
1	ready_out	Sinal emitido para indicar o fim da multiplicação.

A multiplicação em um campo de Galois se baseia no produto bit a bit dos polinômios, seguido do cálculo do resto da divisão do produto pelo polinômio primitivo. Após realizar a leitura das entradas e configurar o polinômio primitivo para o campo, o multiplicador inicia as operações de acordo com a subida de borda do sinal *clock_in*.

Primeiramente, é realizada a leitura do bit mais significativo do registrador que armazena o segundo fator da multiplicação. Caso o bit seja igual a um, ele contribuirá para o resultado total da multiplicação, logo, a operação XOR é realizada entre o resultado parcial, inicializado com zero, e os bits do primeiro fator.

Em seguida, o bit mais significativo do resultado parcial da multiplicação é lido. Caso o bit seja igual a um, a operação XOR do resultado parcial com o polinômio primitivo é realizada, visto que o resultado de um deslocamento à esquerda resultaria em um *overflow*, ultrapassando o número máximo de bits de representação. Por fim, é realizado um deslocamento à esquerda do registrador do segundo fator para que na próxima iteração seja verificado o próximo bit.

Dessa forma, o multiplicador realiza o cálculo do resto da divisão do produto pelo polinômio primitivo, utilizando os resultados parciais da multiplicação. As operações de XOR realizadas representam a soma dos resultados parciais, que é acumulada em um registrador que disponibilizará o resultado final. A Figura 10 apresenta a arquitetura proposta para o multiplicador de Galois.

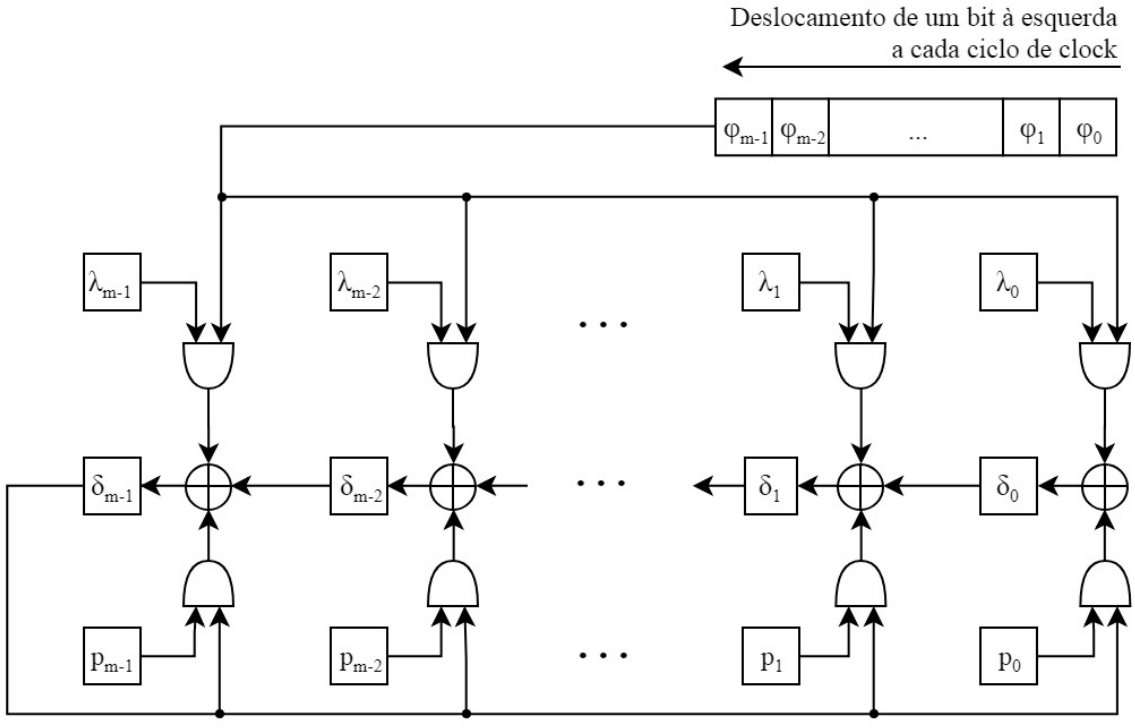

Figura 10 – Arquitetura do multiplicador de Galois para os polinômios $\lambda(x)$ e $\varphi(x)$ a partir do polinômio primitivo p(x), segundo o padrão DVB-S2X.

Onde λ representa o primeiro fator da multiplicação, φ o segundo fator, p o polinômio primitivo e δ o registrador que acumula os resultados parciais e disponibiliza o resultado final após m iterações. A quantidade de iterações m é definida pela ordem do campo GF(2^m).

3.4 Arquiteturas dos Geradores de Síndromes

O gerador de síndromes é responsável por realizar o cálculo de 24 síndromes, dado que o padrão DVB-S2X determina a capacidade de correção de 12 bits para o BCH. O tamanho da palavra-código é inferido em função da taxa de código e do tipo do *frame*, conforme apresentado no Apêndice B.

As arquiteturas implementadas se diferenciam pela forma que são processados os bits do *frame*. Esta abordagem impacta a latência, a taxa de processamento do circuito e o consumo de recursos do FPGA. Estes são uns dos principais fatores analisados no projeto do decodificador BCH. Uma das arquiteturas recebe o *frame* bit a bit, enquanto a outra arquitetura recebe p bits em paralelo a cada iteração.

3.4.1 Gerador de Síndromes de Entrada Serial

A entidade do gerador de síndromes de entrada serial é apresentada na Figura 11. A Tabela 3 descreve o significado de cada entrada e saída deste bloco.

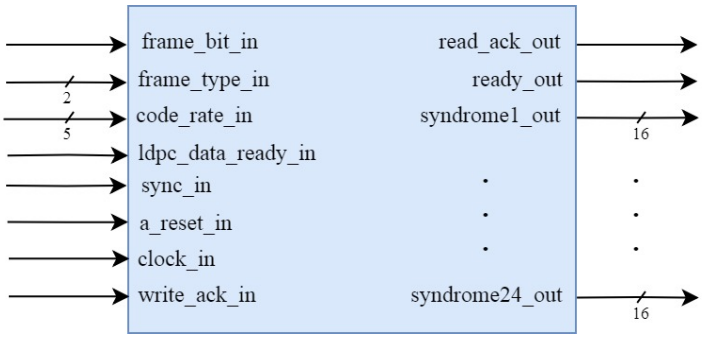

Figura 11 – Entidade do gerador de síndromes de entrada serial.

Os processos do gerador de síndrome de entrada serial são sincronizados pelo sinal *clock_in*. A cada subida de borda deste sinal, o bloco verifica se a entrada *sync_in* está em nível lógico alto para inicializar do processo cálculo das síndromes. Em seguida, o gerador realiza a leitura das entradas *frame_bit_in*, *frame_type_in*, *code_rate_in* e *ldpc_data_ready_in*. A partir dessa leitura, o tipo do *frame* e a taxa de código são verificadas. De acordo com esses parâmetros, o gerador configura o tamanho do *frame* e o conjunto de k elementos α^k para cada síndrome.

O tamanho do *frame* N_{BCH} é utilizado para registrar o número de vezes que a entrada *frame_bit_in* precisa ser lida. A cada iteração do cálculo das síndromes, é realizada a leitura do sinal de entrada *ldpc_data_ready_in* que o decodificador LDPC aciona quando um bit estável é disponibilizado na entrada *frame_bit_in* do BCH. Ao realizar a leitura do bit, a saída *read_ack_out* é acionada em nível lógico alto para informar ao LDPC que o bit foi lido e o próximo pode ser disponibilizado.

Com o bit do *frame* lido, é realizada a operação XOR entre o bit e a saída dos k multiplicadores de Galois, conforme apresentado na Figura 12. Os resultados parciais são armazenados em registradores e utilizados como fatores para as próximas multiplicações. O segundo fator do multiplicador de Galois de cada síndrome é α^k, onde α representa a raiz do polinômio primitivo. Após N_{BCH} iterações, o sinal de saída *ready_out* é colocado em nível lógico alto para informar ao próximo bloco do decodificador BCH, gerador dos coeficientes do polinômio localizador de erro, que os resultados estão disponíveis nas k saídas $syndrome_k$*_out* do gerador.

A entrada *write_ack_in* é utilizada para que o gerador dos coeficientes do polinômio localizador de erro informe ao gerador de síndromes que as saídas foram lidas e que o bloco pode retornar ao estado ocioso. No estado ocioso, o gerador de síndromes é reconfigurado para o estado inicial e aguarda o próximo sinal *sync_in* para processar o próximo *frame*.

Tabela 3 – Descrição das entradas e saídas do gerador de síndromes de entrada serial

N° de bits	**Nome**	**Descrição**
1	clock_in	Sinal de *clock*.
1	a_reset_in	Reset assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
2	frame_type_in	Tipo do *frame*.
5	code_rate_in	Taxa de código.
1	frame_bit_in	Entrada de bits do frame.
1	ldpc_data_ready_in	Entrada que sinaliza que o bit está disponível.
16	$syndrome_k$_out	Saída que disponibiliza a $síndrome_k$.
1	read_ack_out	Sinal emitido para indicar que o bit foi lido.
1	ready_out	Sinal emitido para indicar o fim do cálculo.

A Figura 12 apresenta a arquitetura proposta para o gerador de síndromes de entrada serial.

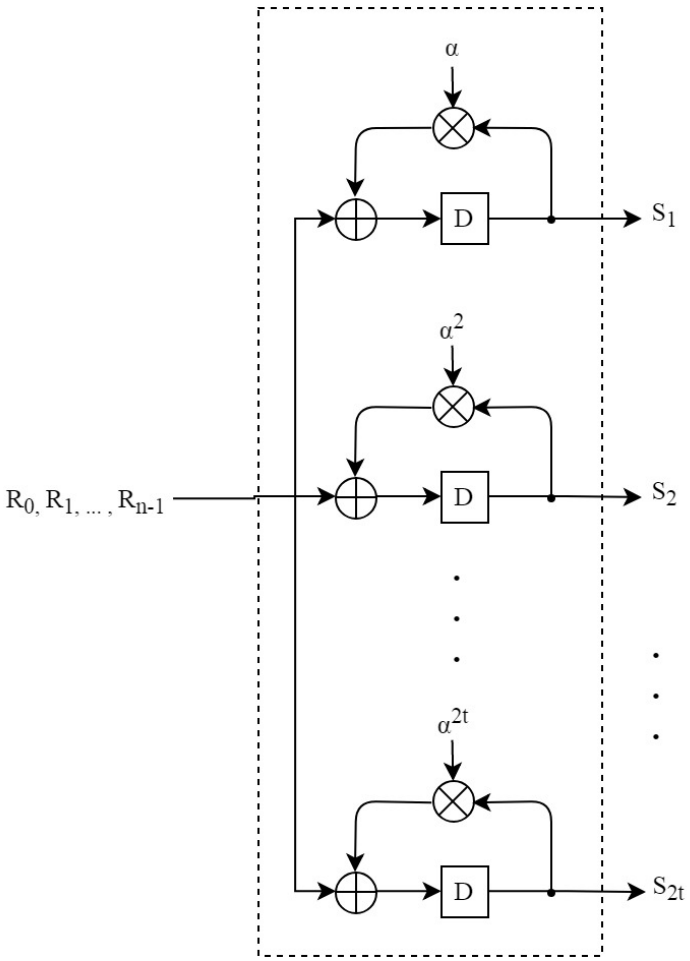

Figura 12 – Arquitetura do gerador de síndromes de entrada serial.

3.4.2 Gerador de Síndromes de Entrada Paralela

A entidade do gerador de síndromes de entrada paralela é apresentada na Figura 13. A Tabela 4 descreve o significado de cada entrada e saída deste bloco.

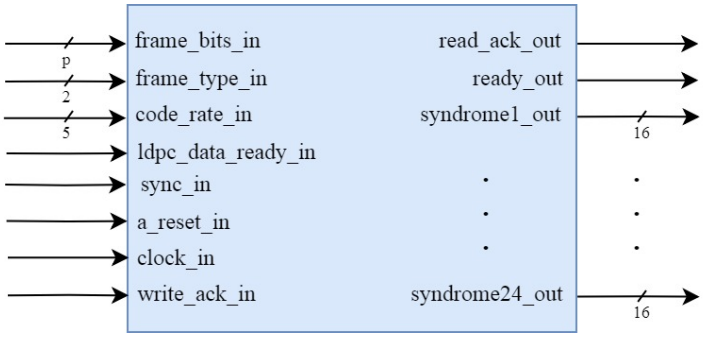

Figura 13 – Interface do gerador de síndromes de entrada paralela.

O gerador de síndromes de entrada paralela possui a interface similar ao gerador de síndromes de entrada serial. A leitura dos sinais segue o mesmo procedimento de sincronização, se diferenciando pela quantidade de bits de entrada. A cada iteração do cálculo das síndromes, é realizada a leitura do sinal de entrada *ldpc_data_ready_in* que o decodificador LDPC aciona quando os bits estão estáveis e disponíveis nas p entradas *frame_bits_in* do BCH. Ao realizar a leitura dos bits, a saída *read_ack_out* é acionada em nível lógico alto para informar ao LDPC que os próximos p bits do *frame* podem ser disponibilizados para a próxima iteração.

Com os bits do *frame* lidos, é realizada a multiplicação entre os bits e os *p-1* elementos $\alpha^{k(p-i-1)}$ do campo, onde k representa o número da síndrome, p o grau de paralelismo dos bits do *frame* e i o índice dos p bits de cada iteração. São realizadas operações XOR com as saídas dos multiplicadores. Como cada multiplicação é realizada entre um elemento do campo e um bit, cada par de multiplicadores da entrada foi substituído por multiplexador (MUX) com um par de bits do *frame* como seletores. De acordo com os valores dos bits, a saída do MUX pode ser zero, um valor de $\alpha^{k(p-i-1)}$ respectivo ao bit ou a XOR entre os $\alpha^{k(p-i-1)}$ respectivos a cada bit. A operação XOR é realizada com as saídas dos MUXes.

Em seguida, é realizada a operação XOR entre o resultado dessa operação e a saída do multiplicador de Galois no estágio final de cada iteração. Esse multiplicador tem como fatores a realimentação do registrador que armazena o resultado parcial do cálculo da síndrome$_k$ e o respectivo α^{kp}. Após N_{BCH}/p iterações, o sinal de saída *ready_out* é colocado em nível lógico alto e são disponibilizadas as k saídas *syndrome$_k$_out*.

Tabela 4 – Descrição das entradas e saídas do gerador de síndromes de entrada paralela

N° **de bits**	**Nome**	**Descrição**
1	clock_in	Sinal de *clock*.
1	a_reset_in	Reset assíncrono.
1	sync_in	Entrada que sinaliza o início de um *frame*.
2	frame_type_in	Tipo do *frame*.
5	code_rate_in	Taxa de código.
p	frame_bits_in	Entradas de p bits do frame.
1	ldpc_data_ready_in	Entrada que sinaliza que os bits estão disponíveis.
16	syndrome$_k$_out	Saída que disponibiliza a síndrome$_k$.
1	read_ack_out	Sinal emitido para indicar que os bits foram lidos.
1	ready_out	Sinal emitido para indicar o fim do cálculo.

A Figura 14 apresenta a arquitetura proposta para o gerador de síndromes de entrada paralela.

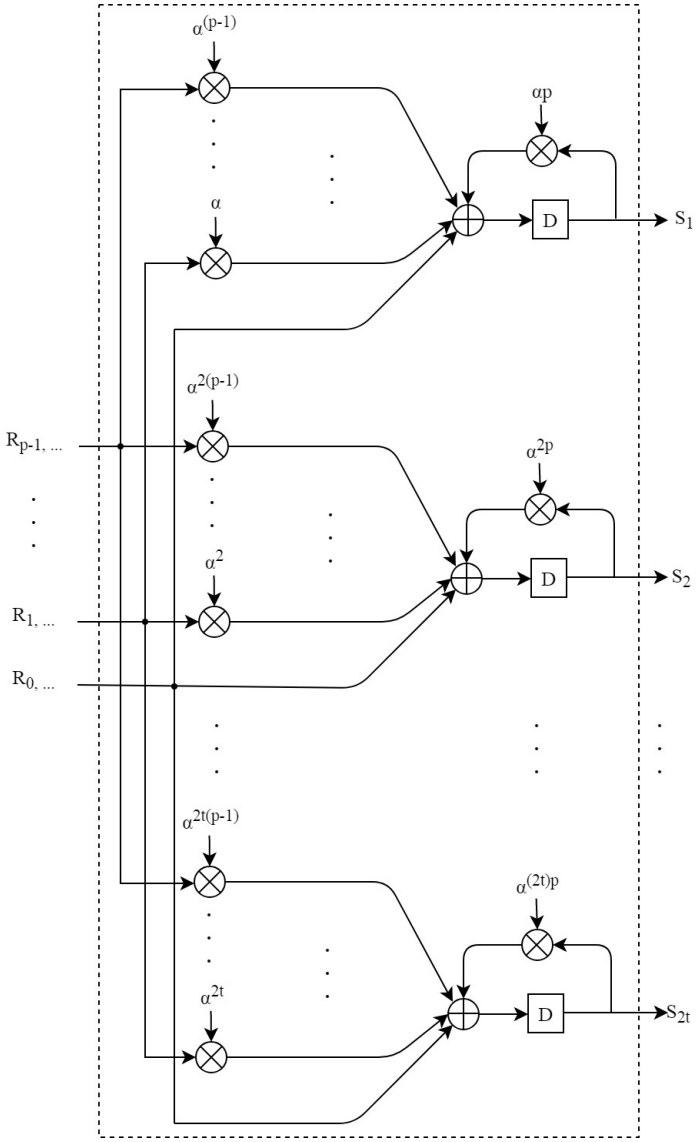

Figura 14 – Arquitetura do gerador de síndromes de entrada paralela.

3.5 Modelo de Simulação das Arquiteturas

O modelo de simulação das arquiteturas foi baseado em uma análise comportamental. As entidades dos geradores de síndromes foram instanciadas como componentes em arquivos de simulação. A análise consistiu em utilizar arquivos de textos com dados, gerados pela simulação em Matlab, como fontes de dados para os estímulos de testes das entradas das arquiteturas. As respostas dos sistemas foram salvas em arquivos de texto para comparação com as saídas da implementação em alto nível.

4 Resultados

4.1 Resultados de Simulação

As arquiteturas implementadas em VHDL foram submetidas a uma análise comportamental por meio de simulações. Foram utilizados arquivos de texto, extraídos da simulação em alto nível, como estímulos nas entradas dos sistemas. As Figuras 15 e 16 apresentam os resultados de simulação para as arquiteturas de geradores de síndromes de entrada serial e de entrada paralela. Foi considerado um *short frame* com taxa de código de 1/4, o que corresponde a um tamanho de *frame* de 3240 bits, segundo o padrão DVB-S2X.

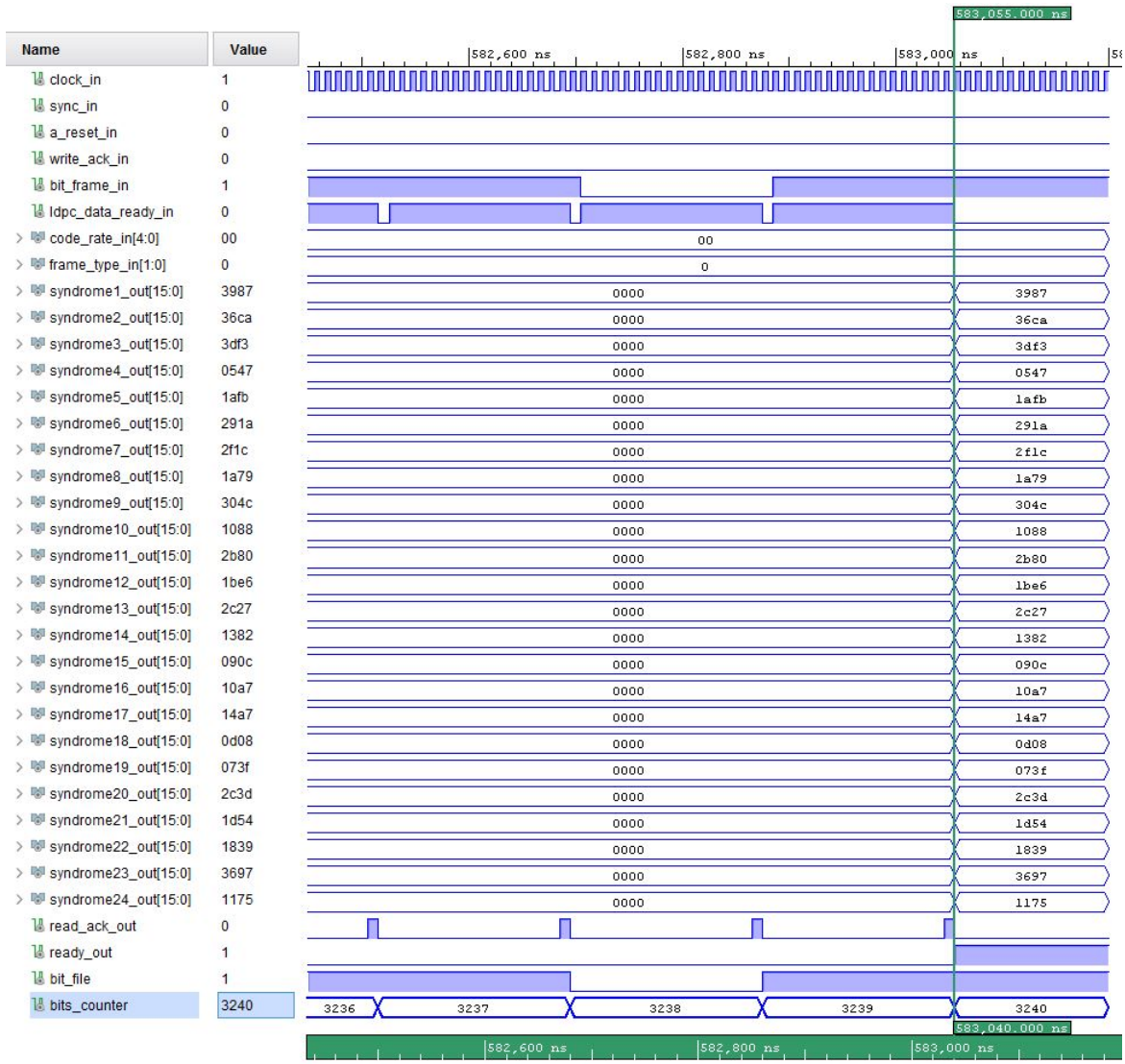

Figura 15 – Simulação do gerador de síndromes de entrada serial utilizando entradas geradas pela simulação de alto nível como sinais de estímulos.

As arquiteturas implementadas se diferenciam pela forma em que o *frame* é recebido e processado pelo decodificador. O gerador de síndromes de entrada serial realiza o recebimento do *frame* bit a bit, enquanto o gerador de síndromes de entrada paralela processa o *frame* em pacotes de 8bits, conforme apresentado na Figura 14.

Os resultados de simulação apresentam o funcionamento da sincronização do BCH com o LDPC. Visto que o bloco gerador de síndromes é responsável por realizar a primeira etapa no processo de decodificação, este possui uma interface direta com o bloco anterior do sistema FEC, o LDPC. As etapas de sincronização dos blocos geradores de síndromes estão de acordo com os requisitos apresentados na Seção 3.1.

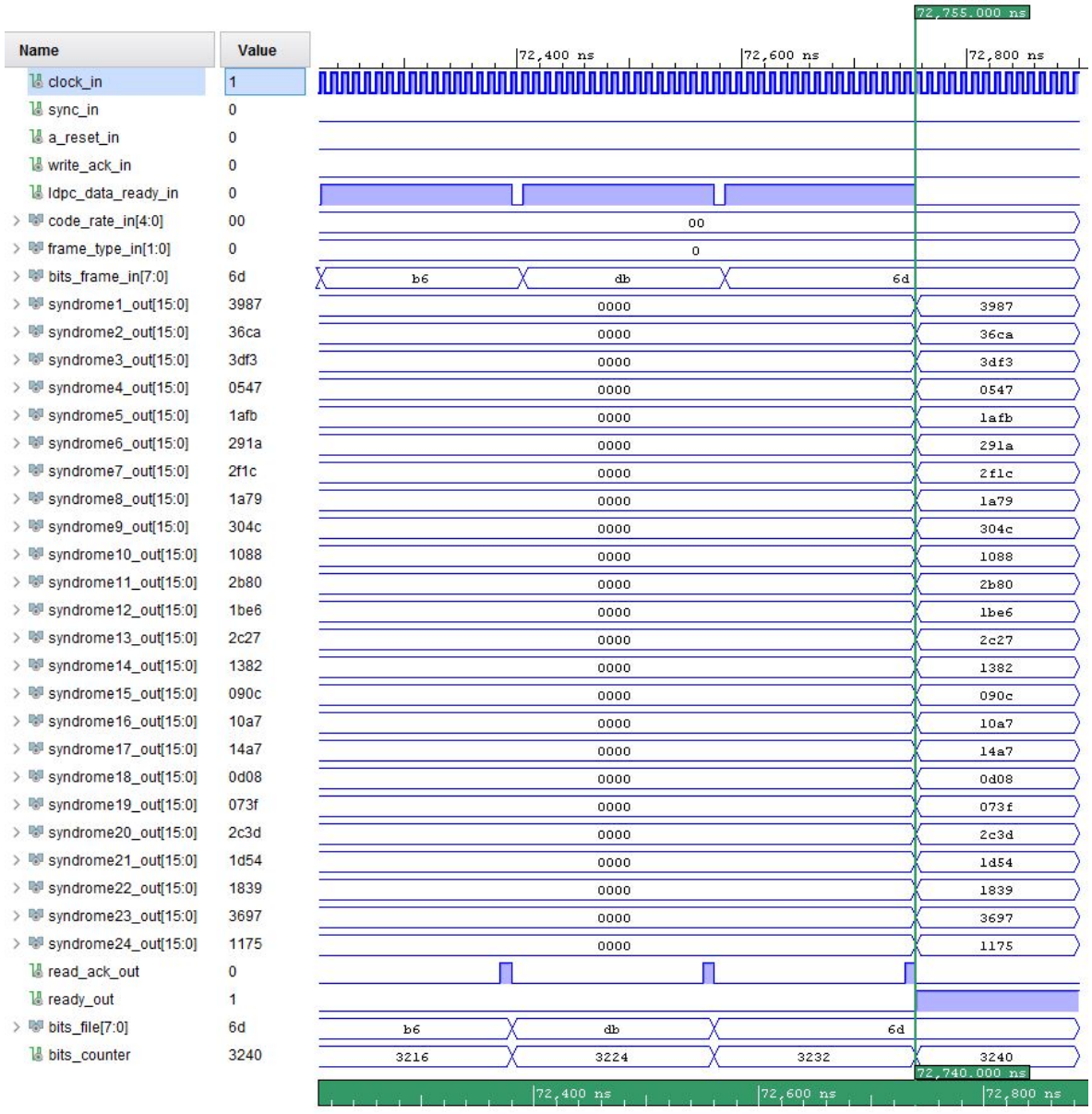

Figura 16 – Simulação do gerador de síndromes de entrada paralela utilizando entradas geradas pela simulação de alto nível como sinais de estímulos.

Devido a quantidade de ciclos de *clock* necessários para o processamento de todos os bits do *frame*, os resultados de simulação ilustram apenas os últimas iteração do cálculo, onde são disponibilizadas nas saídas dos blocos as 24 síndromes geradas. A arquitetura de entrada serial realizou o cálculo das síndromes em 58304 ciclos de *clock*, enquanto a de entrada paralela realizou o cálculo em 7274 ciclos.

O Apêndice C apresenta os valores das síndromes, em representação binária, obtidos nas simulações das arquiteturas implementadas para os geradores de síndrome, apresentados nas Figuras 15 e 16. Estes valores estão em conformidade com os resultados obtidos em simulação com a implementação em alto nível.

4.2 Caracterização dos circuitos

Após o processo de análise de síntese, mapeamento dos pinos de entrada e saída das arquiteturas e o roteamento de conexão entre os componentes, foram obtidas informações referentes ao consumo de recursos de cada sistema. A Tabela 5 apresenta a quantidade de *Look Up Tables*(LUTs), *Flip Flops* (FFs), multiplexadores (MUXes), entradas e saídas (IOs), blocos de memória RAM (BRAMs) e DSPs utilizados em cada implementação.

Os multiplicadores de Galois e os blocos de cálculo de uma síndrome foram instanciadas como componentes dentro de arquiteturas maiores. Portanto, estes componentes não tiveram consumo de IOs reportados, pois sua comunicação com os demais módulos é realizado por meio de sinais internos. A ferramenta de implementação em *hardware* realiza um a otimização dos recursos utilizados, portanto diferentes blocos de multiplicadores de Galois utilizados em blocos distintos podem ter seu consumo reduzido, portanto foi considerado o maior valor observado para o consumo desses componentes.

Dos recursos utilizados pelo gerador de síndromes de entrada serial, 14 LUTs desempenharam o papel de memória para registradores utilizados no processo de deslocamento de bits realizado na operação de multiplicação nos campos de Galois. O gerador de síndromes de entrada paralela utilizou apenas 2 LUTs para esse propósito.

Tabela 5 – Consumo de recursos das arquiteturas implementadas em FPGA

Arquitetura	**LUTs**	**FFs**	**MUXes**	**IOs**	**BRAMs**	**DSPs**
Multiplicador de Galois	62	77	0	0	0	0
Síndrome - Serial	84	124	0	0	0	0
Gerador de Síndromes - Serial	1652	2972	49	35	0	0
Síndrome - Paralela	116	124	32	0	0	0
Gerador de Síndromes - Paralela	2476	2990	49	42	0	0

A Figura 17 apresenta uma comparação do consumo de área do FPGA para as arquiteturas de geradores de síndromes. Os multiplicadores de Galois foram destacados com a cor vermelho. Os *slices* apresentados em cor verde foram utilizados para a implementação dos demais componentes dos geradores. Pode-se observar que os multiplicadores compõem grande parte do consumo de área de ambas arquiteturas.

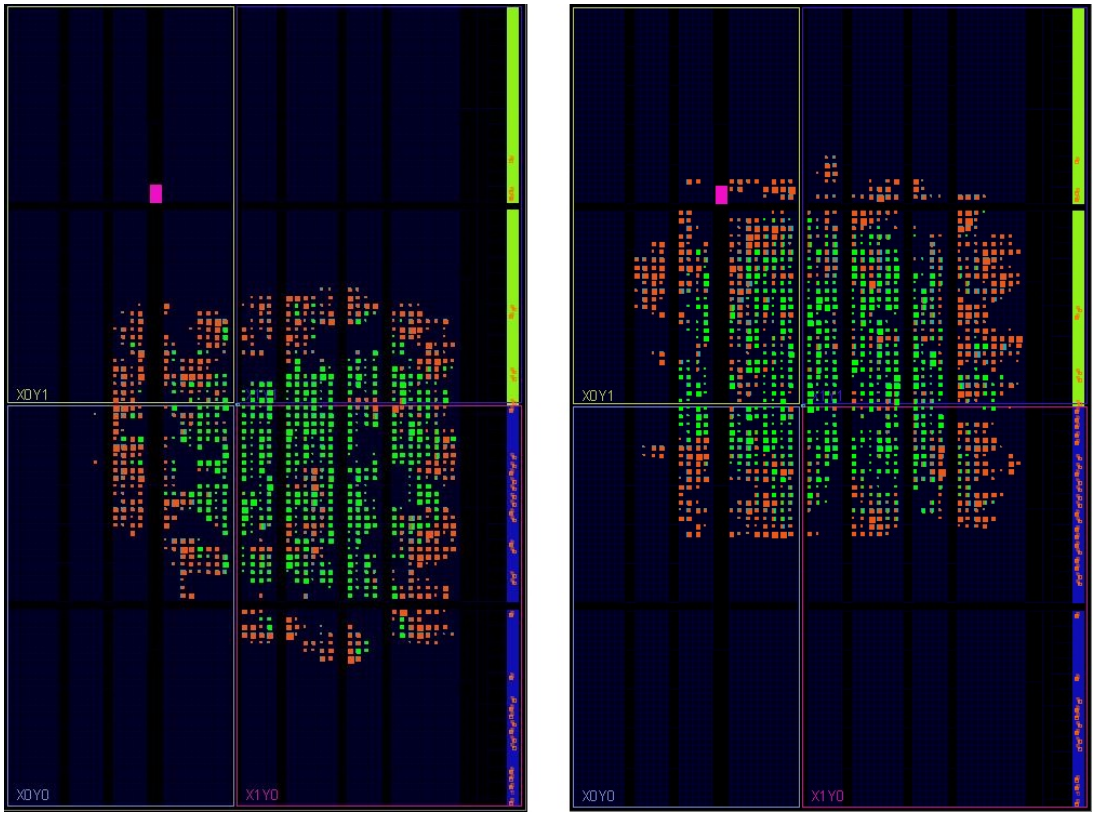

Figura 17 – Comparação entre o consumo de área do FPGA para as arquiteturas de entrada serial (esquerda) e paralela (direita). Os recursos utilizados pelos multiplicadores de Galois estão destacados de vermelho.

O gerador de síndrome de entrada serial, implementado em FPGA, possui frequência máxima de *clock* de 200MHz. Para esta frequência de operação a taxa de processamento do circuito é de aproximadamente 11,11 Mbps. O consumo de energia reportado foi de 0,166 W, sendo que 0,094 W (57%) corresponde a parte estática do dispositivo e 0,072 W (43%) à parte dinâmica, que abrange o consumo de energia por *clocks*, sinais, lógica implementada, entradas e saídas.

O gerador de síndrome de entrada paralela, com grau de paralelismo 8, possui frequência máxima de *clock* de 196 MHz. Para esta frequência, a taxa de processamento é de aproximadamente 87,3 Mbps. O consumo de energia reportado foi de 0,184 W, sendo que 0,094 W (57%) corresponde a parte estática e 0,090 W (43%) à parte dinâmica implementada pela arquitetura.

5 Conclusões

O foco deste trabalho foi a implementação em FPGA de duas arquiteturas de geradores de síndromes para decodificadores BCH no padrão DVB-S2X, visando a realização de um estudo comparativo. As arquiteturas implementadas se diferenciam pelo modo que os bits da palavra-código (*frame*) são recebidos e processados pelo decodificador. O gerador de síndromes de entrada serial recebe o *frame* bit a bit, enquanto que o gerador de síndromes de entrada paralela recebe e processa o *frame* em pacotes de 8 bits.

A principal base para o desenvolvimento do projeto em linguagem de descrição de *hardware* foi o modelo de simulação em alto nível, baseado no método *Model In the Loop*. A adequação dos algoritmos, descritos em linguagem Matlab, para um contexto próximo do que é realizado a nível de registradores possibilitou a verificação antecipada de características da implementação em *hardware*, tais como a quantidade de quantos ciclos de *clock* para cada iteração, número de registradores e fluxo de informação. Essa abordagem permitiu uma análise prévia da lógica dos circuitos combinacionais e sequenciais, o que tornou a implementação em VHDL mais prática.

O uso de arquivos de texto, gerados em simulação de alto nível, para análise comportamental do circuitos implementados em VHDL, proporcionou a validação mais rápida do funcionamento do sistema e tornou o processo de simulação das arquiteturas menos trabalhoso. Esta técnica cria uma interface entre a simulação em alto nível e a implementação em *hardware*, permitindo ao desenvolvedor uma certificação da confiabilidade do sistema.

Os resultados de simulação apresentaram o funcionamento da lógica do sistema, possibilitando a verificação do tempo necessário, em ciclos de *clock* para que o sistema disponibilizasse uma saída válida, após ser inicializado. Observou-se que a razão entre os tempos necessários para se ter uma saída válida, foi de aproximadamente 8, o que corresponde ao grau de paralelismo de uma das arquiteturas. Foi possível realizar a verificação da técnica de sincronização adotada no sistema FEC, baseado no reconhecimento de entrada e confirmação de recebimento.

A análise de síntese, seguida do mapeamento e roteamento em FPGA, permitiu a análise de parâmetros mais específicos dos circuitos. O gerador síndromes de entrada paralela proporcionou o aumento da taxa de processamento em 8 vezes, em comparação ao gerador de entrada serial. O consumo de recursos utilizados pela arquitetura paralelizada foi maior, com uma pequena diferença de consumo de área em FPGA e de consumo de energia. A arquitetura utilizada para gerador de entrada serial garantiu uma frequência máxima de operação de 200MHz, enquanto que a paralelizada foi limitada a 196 MHz.

A demanda de recursos para realizar operações de multiplicação nos campos de Galois foi um dos fatores que foi evidenciado pela análise do consumo de área em FPGA das topologias implementadas. O bloco multiplicador é o centro das operações realizadas pelo algoritmo de cálculo de síndromes. Portanto, uma otimização do multiplicador de Galois se torna necessário para a garantia de maior eficiência do sistema e baixo custo de operação. Outra característica relacionada à arquitetura do multiplicador implementado é o tempo necessário para a disponibilização de dados válidos, que varia de acordo com a ordem do campo.

Uma abordagem que poderia ser utilizada para reduzir o tempo gasto para a realização da multiplicação de elementos dos campos de Galois é o armazenamento em memória do conjunto das potências do elemento primitivo α dos campos de Galois, integrado à uma lógica de busca de acordo com os índices dos polinômios. No entanto, essa solução exigiria grande espaço de armazenamento, tendo em vista que o padrão DVB-S2X determina polinômios que estão contidos em campos de Galois de ordem 14, 15 e 16. O cálculo das síndromes utiliza os valores das potências de α, mas de maneira constante durante todo o processo de decodificação. A geração de 24 síndromes com entrada serial necessitou o armazenamento dos valores das 24 primeiras potências de α dos três campos determinados pelo DVB-S2X. Enquanto que o gerador de síndromes com grau de paralelismo 8 necessitou o armazenamento de 192 valores de potências de α para cada um dos três campos.

5.1 Trabalhos Futuros

O uso de algoritmos de decodificação de alta performance para sistemas FEC permite que um sistema de comunicação tenha maior desempenho no processo de correção de erros de mensagens degradadas pelo canal de transmissão. O aumento do desempenho, por meio de uma abordagem com maior esforço computacional, gera o aumento do consumo de recursos. Portanto, é de extrema importância a aplicação de técnicas que visem a otimização de algoritmos decodificadores que garantam altas taxas de processamento e baixo consumo. Muitos trabalhos têm sido realizados com a proposta de soluções para a implementação dos algoritmos do BCH com o mínimo de ocupação de área em FPGA (LEE; YOO; PARK, 2012) (CHEN; PARHI, 2004b) (ZHANG et al., 2010) (YOO; LEE; PARK, 2011) (CHEN; PARHI, 2004a). Os trabalhos futuros consistirão em realizar a implementação dos demais algoritmos utilizados no processo de decodificação do BCH, visando alta performance e baixo consumo de recursos, promovendo um custo mínimo de área em FPGA.

Referências

AARENSTRUP, R. *Managing Model-Based Design*. United States: The MathWorks Inc, 2015. ISBN 13: 978-1512036138. Citado na página 33.

ALVES, D. C. et al. *FPGA implementation of a FEC decoding subsystem for a DVB-S2 receiver*. [S.l.: s.n.], 2014. 1-6 p. Citado na página 15.

ANAS, E. H. I. et al. *FPGA implementation of a new BCH decoder used in Digital Video Broadcasting - Satellite - Second Generation (DVB-S2)*. [S.l.: s.n.], 2005. ISSN 1992-8645. Citado na página 31.

ATTAR, I. S. S. *Efficient decoding of block codes in DVB-S2 standard*. 2015. 1-6 p. Citado na página 15.

BERLEKAMP, E. R. *A Survey of Algebraic Coding Theory: Lectures Held at the Department of Automation and Information, July 1970*. [S.l.]: Springer, 1973. ISBN 0387810889,9780387810881. Citado na página 27.

CHACON, S. *Pro Git*. Xxii, 265 p. [S.l.]: Apress, 2009. ISBN 1430218339,9781430218333. Citado na página 34.

CHAVES, C.; LIMA, E. D.; MERTES, J. *A Synthesizable BCH Decoder for DVB-S2 Satellite Communications*. [S.l.: s.n.], 2014. Citado 4 vezes nas páginas 7, 15, 28 e 30.

CHEN, Y.; PARHI, K. K. *Area efficient parallel decoder architecture for long BCH codes*. [S.l.: s.n.], 2004. v. 5. V-73 p. Citado na página 50.

CHEN, Y.; PARHI, K. K. *Small area parallel Chien search architectures for long BCH codes*. [S.l.: s.n.], 2004. v. 12. 545-549 p. ISSN 1063-8210. Citado na página 50.

CHIEN, R. T. *Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes*. [S.l.: s.n.], 1964. v. 10. 357 - 363 p. Citado na página 27.

COMBLOCK. *High Speed DVB-S2 BCH code decoder & encoder VHDL source code overview*. Gaithersburg, Maryland: [s.n.], 2009. Citado na página 31.

CROCKETT, L. H. et al. *The Zynq Book Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. University of Strathclyde, Glasgow, Scotland, UK: Strathclyde Academic Media, 2014. Accessado em 13 de agosto de 2018. Disponível em: <https://is.muni.cz/el/1433/jaro2015/PV191/um/The_Zynq_Book_ebook.pdf>. Citado na página 29.

DEVICES, A. *ADALM-PLUTO - SDR Active Learning Module*. 2017. Accessado em 13 de agosto de 2018. Disponível em: <https://www.analog.com/media/en/news-marketing-collateral/product-highlight/ADALM-PLUTO-Product-Highlight.pdf>. Citado 2 vezes nas páginas 7 e 29.

ETSI, E. T. S. I. *Digital Video Broadcasting (DVB); A guideline for the use of DVB specifications and standards*. [S.l.], 1997. Disponível em: <https://www.etsi.org/deliver/etsi_tr/101200_101299/101200/01.01.01_60/tr_101200v010101p.pdf>. Citado na página 19.

ETSI, E. T. S. I. *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*. [S.l.], 2009. Disponível em: <https://www.etsi.org/deliver/etsi_en/302300_302399/302307/01.02.01_60/en_302307v010201p.pdf>. Citado 4 vezes nas páginas 7, 14, 20 e 28.

ETSI, E. T. S. I. *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications. Part II: S2-Extensions (DVB-S2X)*. [S.l.], 2014. Disponível em: <https://www.etsi.org/deliver/etsi_en/302300_302399/30230702/01.01.01_60/en_30230702v010101p.pdf>. Citado 3 vezes nas páginas 22, 27 e 37.

GAUTAM, A. A.; TANDEL, K. B. *Study and design of architecture for BCH code encoder and decoder*. [S.l.: s.n.], 2010. Citado 2 vezes nas páginas 26 e 31.

JAMRO, E. *The design of a VHDL based synthesis tool for BCH codecs*. [s.n.], 1997. Disponível em: <http://home.agh.edu.pl/~jamro/bch_thesis/bch_thesis.html>. Citado 2 vezes nas páginas 19 e 23.

JR., G. C. C.; CAIN, J. B. *Error-Correction Coding for Digital Communications*. 1. ed. [S.l.]: Springer US, 1981. (Applications of Communications Theory). ISBN 978-1-4899-2176-5,978-1-4899-2174-1. Citado 2 vezes nas páginas 13 e 26.

LEE, Y.; YOO, H.; PARK, I. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2012. 1609-1612 p. ISSN 2379-190X. Citado na página 50.

LIMA, E. R. de et al. *A detailed DVB-S2 receiver implementation: FPGA prototyping and preliminary ASIC resource estimation*. [S.l.: s.n.], 2014. 1-6 p. ISSN 2330-989X. Citado na página 15.

MACWILLIAMS, F. J.; SLOANE, N. J. A. *The theory of error correcting codes*. Amsterdam: Elsevier, 1977. ISBN 0444850102. Citado na página 24.

MARCHAND, C.; BOUTILLON, E. *LDPC decoder architecture for DVB-S2 and DVB-S2X standards*. [S.l.: s.n.], 2015. 1-5 p. Citado na página 15.

MENGARDA, A. C. *Core LDPC para o padrão DVB-S2 - Digital Video Broadcasting - Satellite Generation 2*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2016. Disponível em: <http://tede2.pucrs.br/tede2/handle/tede/7061>. Citado na página 19.

MOON, T. K. *Error Correction Coding: Mathematical Methods and Algorithms*. [S.l.]: Wiley-Interscience, 2005. ISBN 9780471648000,0471648000. Citado na página 17.

MORELLO, A.; MIGNONE, V. *DVB-S2: The Second Generation Standard for Satellite Broad-Band Services*. [S.l.: s.n.], 2006. v. 94. 210-227 p. ISSN 0018-9219. Citado na página 20.

PETERSON, W. W.; JR, E. J. W. *Error-Correcting Codes, Second Edition*. [S.l.]: The MIT Press, 1972. (The MIT Press). ISBN 978-0262527316. Citado na página 27.

PRIES, K. H.; QUIGLEY, J. M. *Scrum Project Management*. [S.l.]: CRC Press, 2010. ISBN 978-1-4398-2517-4. Citado na página 33.

PROAKIS, J.; SALEHI, M. *Digital Communications, 5th Edition*. 5th. ed. [S.l.]: McGraw-Hill Science Engineering Math, 2007. ISBN 0072957166,9780072957167. Citado 3 vezes nas páginas 7, 17 e 18.

RAJ, A. A. B. *FPGA-based Embedded System Developer's Guide*. [S.l.]: CRC, 2018. ISBN 9781498796750. Citado na página 28.

SHANNON, C. E. *A Mathematical Theory of Communication*. [S.l.]: Bell system Technical Journal, 1948. Citado 2 vezes nas páginas 13 e 18.

STANCIU, A.; CIOCOIU, T. I.; MOLDOVEANU, F. *A method to handle BCH(n,k,t) algorithm over large GF(n) in pratical hardware implementations*. [S.l.: s.n.], 2015. Citado na página 30.

STEWART, I. N. *Galois theory*. 3. ed. [S.l.]: Chapman and Hall/CRC, 2003. (Chapman and Hall/CRC mathematics). ISBN 1584883936,9781584883937. Citado na página 25.

SUGIYAMA, Y. et al. A method for solving key equation for decoding goppa codes. *Information and Control*, v. 27, n. 1, p. 87 – 99, 1975. ISSN 0019-9958. Citado na página 27.

VIRGOVIč, M. *Forward error correction for storage applications*. Masaryk University - Faculty of Informatics: [s.n.], 2018. Citado na página 15.

XILINX. *Spartan-3E FPGA Family Data Sheet*. 2013. Accessado em 23 de junho de 2018. Disponível em: <https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf>. Citado 2 vezes nas páginas 7 e 29.

YOO, H.; LEE, Y.; PARK, I. . *Area-efficient syndrome calculation for strong BCH decoding*. [S.l.: s.n.], 2011. v. 47. 107-108 p. ISSN 0013-5194. Citado na página 50.

ZHANG, B. et al. *Design and implementation of area-efficient DVB-S2 BCH decoder*. [S.l.: s.n.], 2010. v. 3. V3-179-V3-184 p. Citado 2 vezes nas páginas 30 e 50.

Apêndices

APÊNDICE A – Polinômios primitivos para o BCH - DVB-S2X

As Tabelas 6, 7 e 8 apresentam os polinômios primitivos para *short, medium* e *normal frames* estipulados pelo padrão DVB-S2X.

Tabela 6 – Polinômios primitivos para *short frames*

$g_1(x)$	$1 + x + x^3 + x^5 + x^{14}$
$g_2(x)$	$1 + x^6 + x^8 + x^{11} + x^{14}$
$g_3(x)$	$1 + x + x^2 + x^6 + x^9 + x^{10} + x^{14}$
$g_4(x)$	$1 + x^4 + x^7 + x^8 + x^{10} + x^{12} + x^{14}$
$g_5(x)$	$1 + x^2 + x^4 + x^6 + x^8 + x^9 + x^{11} + x^{13} + x^{14}$
$g_6(x)$	$1 + x^3 + x^7 + x^8 + x^9 + x^{13} + x^{14}$
$g_7(x)$	$1 + x^2 + x^5 + x^6 + x^7 + x^{10} + x^{11} + x^{13} + x^{14}$
$g_8(x)$	$1 + x^5 + x^8 + x^9 + x^{10} + x^{11} + x^{14}$
$g_9(x)$	$1 + x + x^2 + x^3 + x^9 + x^{10} + x^{14}$
$g_{10}(x)$	$1 + x^3 + x^6 + x^9 + x^{11} + x^{12} + x^{14}$
$g_{11}(x)$	$1 + x^4 + x^{11} + x^{12} + x^{14}$
$g_{12}(x)$	$1 + x + x^2 + x^3 + x^5 + x^6 + x^7 + x^8 + x^{10} + x^{13} + x^{14}$

Tabela 7 – Polinômios primitivos para *medium frames*

$g_1(x)$	$1 + x^2 + x^3 + x^5 + x^{15}$
$g_2(x)$	$1 + x + x^4 + x^7 + x^{10} + x^{11} + x^{15}$
$g_3(x)$	$1 + x^2 + x^4 + x^6 + x^8 + x^{10} + x^{12} + x^{13} + x^{15}$
$g_4(x)$	$1 + x^2 + x^3 + x^5 + x^6 + x^8 + x^{10} + x^{11} + x^{15}$
$g_5(x)$	$1 + x + x^2 + x^4 + x^6 + x^7 + x^{10} + x^{12} + x^{15}$
$g_6(x)$	$1 + x^4 + x^6 + x^7 + x^{12} + x^{13} + x^{15}$
$g_7(x)$	$1 + x^2 + x^4 + x^5 + x^7 + x^{11} + x^{12} + x^{14} + x^{15}$
$g_8(x)$	$1 + x^2 + x^4 + x^6 + x^8 + x^9 + x^{11} + x^{14} + x^{15}$
$g_9(x)$	$1 + x + x^2 + x^4 + x^5 + x^7 + x^9 + x^{11} + x^{12} + x^{13} + x^{15}$
$g_{10}(x)$	$1 + x + x^2 + x^3 + x^4 + x^7 + x^{10} + x^{11} + x^{12} + x^{13} + x^{15}$
$g_{11}(x)$	$1 + x + x^2 + x^4 + x^9 + x^{11} + x^{15}$
$g_{12}(x)$	$1 + x^2 + x^4 + x^8 + x^{10} + x^{11} + x^{13} + x^{14} + x^{15}$

Tabela 8 – Polinômios primitivos para *normal frames*

$g_1(x)$	$1 + x^2 + x^3 + x^5 + x^{16}$
$g_2(x)$	$1 + x + x^4 + x^5 + x^6 + x^8 + x^{16}$
$g_3(x)$	$1 + x^2 + x^3 + x^4 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{16}$
$g_4(x)$	$1 + x^2 + x^4 + x^6 + x^9 + x^{11} + x^{12} + x^{14} + x^{16}$
$g_5(x)$	$1 + x + x^2 + x^3 + x^5 + x^8 + x^9 + x^{10} + x^{11} + x^{12} + x^{16}$
$g_6(x)$	$1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{12} + x^{13} + x^{14} + x^{15} + x^{16}$
$g_7(x)$	$1 + x^2 + x^5 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{15} + x^{16}$
$g_8(x)$	$1 + x + x^2 + x^5 + x^6 + x^8 + x^9 + x^{12} + x^{13} + x^{14} + x^{16}$
$g_9(x)$	$1 + x^5 + x^7 + x^9 + x^{10} + x^{11} + x^{16}$
$g_{10}(x)$	$1 + x + x^2 + x^5 + x^7 + x^8 + x^{10} + x^{12} + x^{13} + x^{14} + x^{16}$
$g_{11}(x)$	$1 + x^2 + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{13} + x^{16}$
$g_{12}(x)$	$1 + x + x^5 + x^6 + x^7 + x^9 + x^{11} + x^{12} + x^{16}$

APÊNDICE B – Tamanhos de mensagem e palavra-código para o BCH - DVB-S2X

As Tabelas 9, 10 e 11 apresentam os tamanhos N_{BCH} e K_{BCH}, de acordo com os tipos de *frames* e com as taxas de códigos descritas pelo padrão DVB-S2X.

Tabela 9 – Tamanhos de palavra-código e de mensagem para *short frames*

Taxa de Código	N_{BCH} **(bits)**	K_{BCH} **(bits)**
1/4	3240	3072
11/45	3960	3792
4/15	4320	4152
14/45	5040	4872
1/3	5400	5232
2/5	6480	6312
1/2	7200	7032
7/15	7560	7392
8/15	8640	8472
26/45	9360	9192
3/5	9720	9552
2/3	10800	10632
3/4	11880	11712
32/45	11520	11352
4/5	12600	12432
5/6	13320	13152
8/9	14400	14232

Tabela 10 – Tamanhos de palavra-código e de mensagem para *medium frames*

Taxa de Código	N_{BCH} **(bits)**	K_{BCH} **(bits)**
1/5	5840	5660
11/45	7920	7740
1/3	10800	10620

Tabela 11 – Tamanhos de palavra-código e de mensagem para *normal frames*

Taxa de Código	N_{BCH} **(bits)**	K_{BCH} **(bits)**
1/4	16200	16008
1/3	21600	21408
2/5	25920	25728
1/2	32400	32208
3/5	38880	38688
2/3	43200	43040
3/4	48600	48408
4/5	51840	51648
5/6	54000	53840
8/9	57600	57472
9/10	58320	58192
2/9	14400	14208
13/45	18720	18528
9/20	29160	28968
90/180	32400	32208
96/180	34560	34368
11/20	35640	35448
100/180	36000	35808
104/180 e 26/45	37440	37248
18/30	38880	38688
28/45	40320	40128
23/36	41400	41208
116/180	41760	41568
20/30	43200	43008
124/180	44640	44448
25/36	45000	44808
128/180	46080	45888
13/18	46800	46608
132/180 e 22/30	47520	47328
135/180	48600	48408
140/180 e 7/9	50400	50208
154/180	55440	55248

APÊNDICE C – Síndromes obtidas nas simulações das arquiteturas implementadas

A Tabela 12 apresenta os valores das síndromes, em representação binária, obtidos nas simulações das arquiteturas implementadas para os geradores de síndrome, apresentados nas Figuras 15 e 16.

Tabela 12 – Representação binária dos resultados de simulação das arquiteturas implementadas

Síndrome	**Valor**
S_1	0011100110000111
S_2	0011011011001010
S_3	0011110111110011
S_4	0000010101000111
S_5	0001101011111011
S_6	0010100100011010
S_7	0010111100011100
S_8	0001101001111001
S_9	0011000001001100
S_{10}	0001000010001000
S_{11}	0010101110000000
S_{12}	0001101111100110
S_{13}	0010110000100111
S_{14}	0001001110000010
S_{15}	0000100100001100
S_{16}	0001000010100111
S_{17}	0001010010100111
S_{18}	0000110100001000
S_{19}	0000011100111111
S_{20}	0010110000111101
S_{21}	0001110101010100
S_{22}	0001100000111001
S_{23}	0011011010010111
S_{24}	0001000101110101