

Arley Henrique Salvador

IMPLEMENTAÇÃO EM VHDL DE UMA ARQUITETURA
PARALELA DE UM CÓDIGO DE REED-SOLOMON
APLICADO A REDES OTN

Campinas
2015

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Arley Henrique Salvador

IMPLEMENTAÇÃO EM VHDL DE UMA ARQUITETURA
PARALELA DE UM CÓDIGO DE REED-SOLOMON
APLICADO A REDES OTN

Dissertação de mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Telecomunicações e Telemática.

Orientador: Dalton Soares Arantes.
Co-orientador: Júlio César R. F. de Oliveira.

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Arley Henrique Salvador, e orientada pelo Prof. Dr. Dalton Soares Arantes

Campinas
2015

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Elizangela Aparecida dos Santos Souza - CRB 8/8098

Sa38i Salvador, Arley Henrique, 1979-
Implementação em VHDL de uma arquitetura paralela de um código de Reed-Solomon aplicado a Redes OTN / Arley Henrique Salvador. – Campinas, SP : [s.n.], 2015.

Orientador: Dalton Soares Arantes.
Coorientador: Júlio César Rodrigues Fernandes de Oliveira.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Reed-Solomon, Códigos de. 2. Códigos corretores de erros (Teoria da informação). 3. Comunicações ópticas. 4. Algoritmos paralelos. I. Arantes, Dalton Soares, 1946-. II. Oliveira, Júlio César Rodrigues de. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: VHDL implementation of parallel architecture of the Reed-Solomon code for OTN networks

Palavras-chave em inglês:

Reed-Solomon codes

Error correcting codes (Information theory)

Optical communications

Parallel algorithms

Área de concentração: Telecomunicações e Telemática

Titulação: Mestre em Engenharia Elétrica

Banca examinadora:

Dalton Soares Arantes [Orientador]

José Santo Guistafre Panaro

Renato Baldini Filho

Data de defesa: 28-05-2015

Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Arley Henrique Salvador

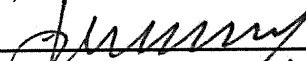
Data da Defesa: 28 de maio de 2015

Título da Tese: "Implementação em VHDL de uma Arquitetura Paralela de um Código de Reed-Solomon Aplicado a Redes OTN"

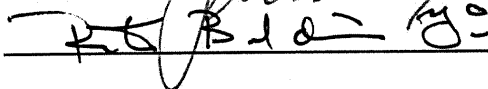
Prof. Dr. Dalton Soares Arantes (Presidente):



Prof. Dr. José Santo Guiscafré Panaro:



Prof. Dr. Renato Baldini Filho:



Resumo

Este trabalho apresenta a implementação de uma arquitetura paralela de um código corretor de erros para aplicações em redes ópticas que utilizam a técnica *Forward Error Correction* (FEC). O algoritmo FEC especificamente tratado neste trabalho é o *Reed-Solomon*, que é destinado principalmente a sistemas que sofrem influência causada por erros em rajadas somados ao sinal durante a transmissão, o que o torna adequado para transmissões ópticas. São expostas estruturas seriais de codificador e decodificador FEC e as etapas para convertê-las para uma estrutura paralela. Após descrever as etapas de conversão de uma estrutura serial para paralela é apresentada a estrutura do codificador/decodificador FEC Reed-Solomon RS(255,239) com estrutura paralela para operar em redes ópticas a uma taxa de 100 Gbit/s. Esta descrição exemplificativa é o objetivo principal deste trabalho. A implementação paralela do FEC oferece como vantagem a capacidade de processar os dados de forma rápida, permitindo o emprego desta solução em sistemas com altas taxas de dados. Foi elaborado um ambiente de testes com uma aplicação em redes de transporte óptico, ou *Optical Transport Network* (OTN). Esta funcionalidade consiste de um Transponder, que tem a função de mapear um cliente de 100 Gigabits Ethernet dentro de uma estrutura de quadro destinado a transmissão de dados em redes ópticas. Deste modo, pôde-se comprovar os resultados e o desempenho da estrutura proposta.

Palavras-chaves: Correção de Erro. Reed-Solomon. Redes de Transporte Óptico. OTN. Estrutura Paralela. Aplicações OTN em 100 Gbit/s.

Abstract

This paper presents the implementation of a parallel architecture for an error-correcting code for optical applications that uses the Forward Error Correction (FEC) technique. The FEC algorithm specifically addressed in this work is applied to the Reed-Solomon, which is mainly intended for systems that are harmed by burst errors, which makes it suitable for optical transmissions. A serial FEC encoder/decoder structure and the steps to convert it to a parallel approach are addressed in this work. An example of method that generates an encoder/decoder for a RS(255,239) Reed-Solomon code with parallel structure, able to operate at 100 Gbit/s data rate in optical networks, is also presented. The parallel implementation offers higher FEC processing speeds to handle higher throughputs. A test environment was designed with an application in optical transport networks (OTN). This feature consists of a transponder which maps a 100 Gigabit Ethernet client inside an OTN frame structure. With this setup the expected results for the proposed FEC circuitry could be experimentally verified.

Key-words: Forward Error Correction. Reed-Solomon. Optical Transport Network. OTN. Parallel Structure. OTN Applications for 100 Gbit/s.

Conteúdo

Introdução Geral	1
1 Introdução	2
1.1 Visão Geral	2
1.2 Motivação	4
1.3 Contribuição	4
1.4 Estado da Arte	5
1.5 Organização da Dissertação	6
2 Redes Ópticas de Transporte	8
2.1 Histórico Sobre Redes de Transporte	8
2.2 Hierarquia da OTN	9
2.3 Taxas de Transmissão	10
2.4 Estrutura do Quadros OTN	11
2.5 <i>Forward Error Correction</i> (FEC)	13
3 Álgebra de Campos Finitos	17
3.1 Conceitos Básicos	17
3.1.1 Grupo	17
3.1.2 Anel	18
3.1.3 Campo	18
3.2 Campos Finitos	19
3.2.1 Definição de Campo Finito	19
3.2.2 Propriedades de um Campo de Finito	19
3.2.3 Campo Finito Binário $GF(2)$	20
3.2.4 Campo Binário Estendido $GF(2^m)$	21
4 Teoria Básica Sobre Códigos Corretores de Erro	24
4.1 Conceitos Fundamentais de Códigos Corretores de Erro	25
4.1.1 Códigos de Bloco	25
4.1.2 Códigos Cíclicos	26
4.1.3 Matriz Geradora	27
4.1.4 Código Sistemático	28
4.1.5 Matriz Verificação de Paridade	28
4.1.6 Taxa de Código	29

4.1.7	Distância e Peso de Hamming	29
4.1.8	Capacidade de Detecção e Correção de Erros	30
4.1.9	Ganho de Codificação	31
4.2	Códigos Detectores de Erro	33
4.2.1	Códigos de Paridade	33
4.2.2	Códigos de Checksum	33
4.2.3	Códigos de Redundância Cíclica	34
4.3	Códigos Corretores de Erro	35
4.3.1	Código Bose-Chaudhuri-Hocquenqhem	35
4.3.2	Código Hamming	35
4.3.3	Código Reed-Solomon	36
5	Arquitetura Paralela do Código RS(255,239) para 100 Gbit/s	45
5.1	Operações Básicas em Campo Finito	46
5.1.1	Soma e Subtração	46
5.1.2	Multiplicação	46
5.1.3	Divisão e Inversão	48
5.2	Codificador Reed-Solomon	49
5.2.1	Estrutura Serial do Codificador FEC RS(255,239)	49
5.2.2	Estrutura Paralela do Codificador Reed-Solomon	51
5.3	Decodificador Reed-Solomon	57
5.3.1	Síndrome	57
5.3.2	Algoritmo Berlekamp Massey	61
5.3.3	Algoritmo Chien Search	64
5.3.4	Algoritmo de Forney	73
6	Resultados e Desempenho do Circuito	76
6.1	Simulação do Circuito	76
6.2	Ocupação de Recursos do FPGA	77
6.3	Arranjo Experimental	78
6.3.1	Cenário de Teste - Meio Regenerador OTU2	79
6.3.2	Cenário de Teste - Transponder 100GbE	81
7	Conclusões	87
	Bibliografia	89

DEDICO ESTE TRABALHO AOS
MEUS PAIS, CÂNDIDO SALVADOR
FILHO E MARTA ROSÁLIA LEITE
SALVADOR, PELO INCENTIVO E
APOIO A TODAS AS MINHAS ESCO-
LHAS.

Agradecimentos

Agradeço,
a Deus, pela realização deste trabalho.

Ao Professor Dalton Soares Arantes pela valiosa orientação e pela oportunidade de trabalhar ao seu lado.

Ao co-orientador, Dr Júlio César Rodrigues Fernandes de Oliveira, pelo seu apoio na realização deste trabalho.

A todos da gerência de tecnologias ópticas da Fundação CPqD, pelo tempo que fui dispensado para a realização deste trabalho. Aos colegas de trabalho, em especial Cleber Akira Nakandakare, Daniele Renata da Silva, Diego Brito de Carvalho, Eduardo Mobilon, Luis Renato Monte e Rodrigo Bernardo, agradeço pelo o apoio e parceria nas publicações.

Agradeço aos meus pais, Cândido Salvador Filho e Marta Rosália Leite Salvador pelo apoio e exemplo que me deram durante toda a minha vida. Gostaria de registrar aqui o meu amor e admiração por vocês.

Finalmente agradeço a Unicamp pela ensino de qualidade oferecido aos alunos de pós-graduação. Meu sincero reconhecimento pelo trabalho do governo estadual de São Paulo, que mantém esta instituição, o que permite que muitas pessoas possam ter acesso ao ensino superior gratuito. Esta oportunidade permitiu-me a realização de um sonho.

Estar decidido, acima de qualquer coisa, é o segredo do êxito.

Henry Ford

Lista de Figuras

1.1	Sistema de Comunicação	2
2.1	Hierarquia OTN.	10
2.2	Estrutura de Quadros OTN.	12
2.3	Entrelaçamento de Dados.	14
2.4	Entrelaçamento de Dados no Processamento Serial.	15
2.5	Entrelaçamento de Dados no Processamento Paralelo.	15
4.1	Curva Típica de Ganho de Codificação. Fonte: Codificação de Canal: Códigos de Bloco Lineares.[1]	32
4.2	Diagrama em Blocos do Decodificador FEC Reed Solomon.	39
5.1	Somador em Campo Finito.	46
5.2	Multiplicador em Campo de Finito.	48
5.3	Divisor em Campo Finito.	49
5.4	Codificador Reed-Solomon.	50
5.5	Codificador Paralelo RS(255,239).	52
5.6	Codificador Paralelo RS(255,239) e M=5.	54
5.7	Estágios de Codificação de Paridade.	56
5.8	Diagrama em Blocos do Decodificador FEC Reed Solomon.	57
5.9	Processamento da Síndrome de Forma Serial.	58
5.10	Processamento da Síndrome de Forma Paralela.	60
5.11	Fluxograma do Algoritmo Berlekamp-Massey.	62
5.12	Arquitetura Serial do Chien Search Locator.	66
5.13	Arquitetura Paralela do Chien Search Locator.	67
5.14	Arquitetura Paralela Exemplo do Chien Search Locator.	69
5.15	Arquitetura Serial do <i>Chien Search Evaluator</i>	70
5.16	Arquitetura Paralela do Chien Search Locator.	71
5.17	Arquitetura Paralela Exemplo do <i>Chien Search Evaluator</i>	73
5.18	Arquitetura Serial de Implementação Serial do Algoritmo de Forney.	74
5.19	Arquitetura Paralela do Algoritmo de Forney.	74
5.20	Arquitetura Paralela Exemplo do Algoritmo de <i>Forney</i>	75

6.1	Ambiente Teste do Codificador e Decodificador FEC RS(255,239).	79
6.2	Diagrama em Blocos do Meio Regenerador	80
6.3	Teste 1	81
6.4	Ganho de Codificação	82
6.5	Diagrama em Blocos do Transponder 100 Gigabit-Ethernet	82
6.6	Teste de Stress do FEC	85
6.7	Gráfico BER Entrada x BER Saída Experimental	85

Lista de Tabelas

2.1	Taxa de Dados Para os Quadros OPuk Padronizados pela G.709	11
2.2	Taxa de Dados Para os Quadros ODUk Padronizados pela G.709	11
2.3	Taxa de Dados Para os Quadros OTUk Padronizados pela G.709	11
3.1	Resultado da Adição Entre os Elementos de um GF(2)	20
3.2	Resultado da Multiplicação Entre os Elementos de um GF(2)	20
3.3	Lista de Polinômios Primitivos	22
3.4	Resultado da Adição Entre os Elementos de um GF(2 ³)	22
3.5	Resultado da Multiplicação Entre os Elementos de um GF(2 ³)	23
3.6	Representações dos Elementos em GF(2 ³)	23
4.1	Palavras Código para C(6,3)	26
5.1	Variáveis Utilizadas no Algoritmo Berlekamp Massey	63
5.2	Sequência de Verificação das Raízes do Polinômio Localizador Processados Serialmente	65
5.3	Sequência de Verificação das Raízes do Polinômio Localizador Processados Paralelamente (5 Coeficientes a Cada Ciclo de Clock)	68
5.4	Sequência de Verificação das Raízes do Polinômio Localizador Processados Serialmente	70
5.5	Sequência de Verificação das Raízes do Polinômio Avaliador Processados Paralelamente (5 Coeficientes a Cada Ciclo de Clock)	72
6.1	Medidas de Desempenho do codificador/Decodificador FEC	78

Lista de Acrônimos e Notação

ARQ	<i>Automatic Repeat Request</i>
ASK	<i>Amplitude-shift keying</i>
ASIC	<i>Application Specific Integration Circuit</i>
BCH	Código Bose-Chaudhuri-Hocquenghem
BER	Taxa de Erros de Bit
BM	<i>Berlekamp-Massey</i>
CRC	Código de Redundância Cíclica
CSE	<i>Chien Search Evaluator</i>
CSL	<i>Chien search Locator</i>
DUT	<i>Design Under Test</i>
FAS	<i>Frame Alignment Signal</i>
FEC	<i>Forward Error Correction</i>
FPGA	<i>Field Programmable Gate Array</i>
FSK	<i>Frequency Shift Keying</i>
GF	<i>Galois Field</i>
GTH	<i>Transceiver Serializador-Deserializador</i>
IP	<i>Internet Protocol</i>
ITU-T	<i>International Telecommunication Unit</i>
JC	Controle de Justificação
LUT	<i>Lookup Table</i>
OC	<i>Optical Channel</i>
ODU _k	<i>Optical Data Unit k</i>
OH	<i>Overhead</i>
OMS	<i>Optical Multiplex Section</i>
OOK	<i>On Off Keying</i>
OPU	<i>Optical Payload Unit</i>
OTN	<i>Optical Transport Network</i>
OTU _k	<i>Optical Transport Unit k</i>
OTS	<i>Optical Transport Section</i>
OSNR	<i>Optical Signal to Noise Relation</i>
PB	Peta Bytes
PDH	<i>Plesiochronous Digital Hierarchy</i>
PMD	Dispersão por Modo de Polarização
PSI	Identificador de Estrutura do Payload
PSK	<i>Phase Shift Keying</i>

PSTN	<i>Public Switched Telephone Network</i>
SDH	<i>Synchronous Digital Hierarchy</i>
RS	<i>Reed-Solomon</i>
SRAM	Memória RAM Estática
TCM	Monitoração de controle <i>Tandem</i>
TDM	<i>Time Division Multiplexing</i>
SONET	<i>Synchronous Optical Networking</i>
VHDL	Linguagem de Descrição de <i>Hardware</i>
WDM	Multiplexação por Divisão de Comprimento de Onda
\forall	notação que significa para qualquer
\in	notação que significa pertence ao conjunto
A	notação para matrizes (letras maiúsculas do alfabeto latino)
U_k	notação para espaço vetorial
(G, \cdot)	Conjunto de elementos com a operação \cdot definida entre eles
A^T	$(^t)$, notação que indica a operação de transposto da matriz A
$V(X)$	notação para representar um polinômio
$C(n, k)$	notação para representar os parâmetros de um código corretor de erros
\bar{u}	notação para representar um vetor
\mathbb{R}	conjunto dos números reais
\mathbb{Z}	conjunto dos números inteiros
\mathbb{Z}_+	conjunto dos números inteiros não negativos
\mathbb{N}	conjunto dos números naturais (incluindo o zero)
α	especialmente utilizada para representar um elemento do campo de Galois na forma polinomial

Introdução Geral

O crescente aumento do tráfego de dados através de redes de telecomunicações tem impulsionado a evolução dos sistemas de transmissão já existentes e a criação de novas soluções que atendam a esta demanda. A principal tecnologia responsável por este crescimento é a tecnologia fotônica. Desde a introdução dos primeiros sistemas de transmissão óptica, a capacidade de transmissão têm aumentado e o custo por bit transmitido diminuído gradualmente [2]. Ao longo do tempo surgiram, ou foram agregadas, novas tecnologias aos sistemas de transmissões ópticas que permitiram esta evolução, como por exemplo a tecnologia de transmissão óptica WDM (*Wavelength Division Multiplexing*), a transmissão coerente e os algoritmos corretores de erros.

O presente trabalho dedica-se exclusivamente ao tema de códigos corretores de erros, em especial ao código Reed-Solomon. Este tipo de algoritmo é um tema que já foi bastante explorado por trabalhos anteriores. Entretanto, diante da demanda de sistemas de comunicação que operem a taxas cada vez mais elevadas, este trabalho apresenta uma arquitetura paralela do código corretor de erros Reed-Solomon RS(255,239) que seja capaz de processar dados a 100 Gbit/s.

Introdução

1.1 Visão Geral

Um **sistema de comunicação** digital tem como objetivo transportar dados de uma fonte de informação para um determinado destino. A Figura 1.1 apresenta os principais blocos funcionais de um sistema de comunicação de dados [3].

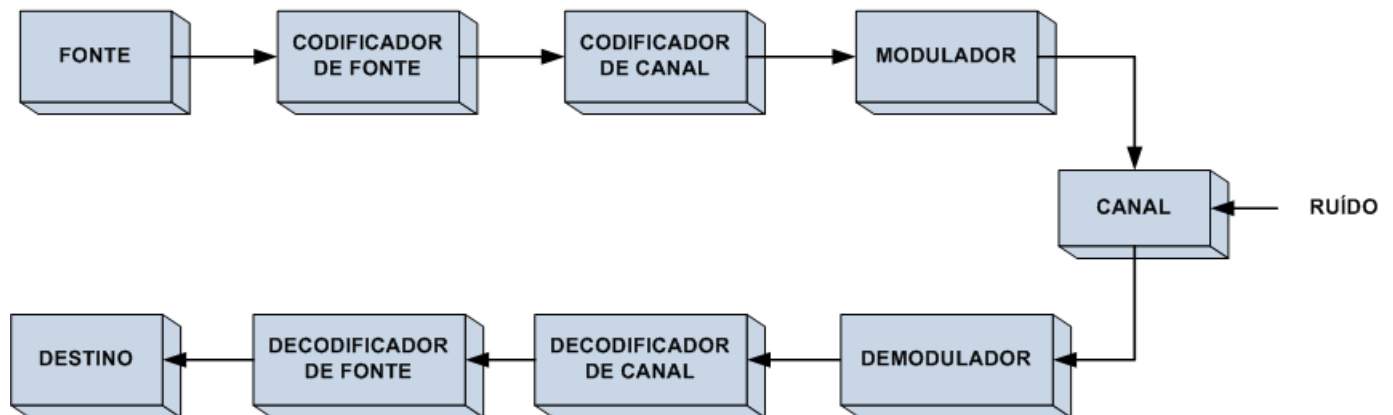


Figura 1.1: Sistema de Comunicação

A **fonte** de informação gera os dados que devem ser transmitidos. Estes dados pertencem a um alfabeto finito e geralmente são representados por símbolos de base binária. A informação gerada pode ter diversas naturezas, como por exemplo: voz, música ou vídeo.

Em muitos casos, as informações geradas podem ter conteúdos redundantes. O bloco **codificador de fonte** associa a cada mensagem um símbolo adequado para a transmissão pelo canal de comunicação, com o objetivo de reduzir possíveis redundâncias. Este mapeamento do sinal da fonte em outro sinal antes da transmissão gera uma representação mais eficiente da informação e tem como resultado imediato uma economia de banda do canal de comunicação [4].

O **codificador de canal** é utilizado para melhorar a confiabilidade da transmissão de dados. A transmissão de dados por um canal de comunicação pode sofrer influência e ser distorcido por consequência do **ruído** aditivo ao sinal. A intensidade desta interferência depende de alguns fatores, como por exemplo: a potência de transmissão, a potência do ruído e a taxa de transmissão. Existem alguns meios de se reduzir esta interferência, porém, se o nível de perturbação for muito elevado, a correção deste efeito torna-se impraticável. Na codificação de canal são adicionados novos símbolos à mensagem que será transmitida, chamados símbolos de paridade, para que o receptor tenha a capacidade de detectar e corrigir eventuais erros. Esta função é implementada por intermédio dos **códigos corretores de erro**. O uso dos códigos corretores de erro em redes de transporte óptico de dados tornou-se imprescindível por aumentar a confiabilidade da transmissão de dados através do canal. Em transmissões com taxas iguais ou superiores a 100 Gbit/s, a recomendação ITU-T G.709 define que o emprego de técnicas de correção de erros é mandatório.

O **modulador** prepara a informação para a transmissão pelo canal de comunicação. Várias técnicas de modulação digital podem ser empregadas, como por exemplo *amplitude shift keying* (ASK), *frequency-shift keying* (FSK), *phase shift keying* (PSK), *differential phase shift keying* (DPSK) e *quadrature amplitude modulation* (QAM).

Na recepção, todo o processo é revertido até que a informação originalmente produzida pela fonte alcance seu **destino**.

O processo de decodificação normalmente não assegura a recuperação completa da mensagem original, devido ao ruído e distorções causadas por efeito de propagação da luz na fibra óptica. Consequentemente, o desempenho do sistema de comunicação representado na Figura 1.1 está diretamente relacionado à eficiência com que a informação gerada pela fonte é representada e, ainda, com a taxa na qual esta informação é transmitida através do canal de comunicação, sob efeito do ruído.

Este trabalho apresenta um método de implementação de um algoritmo de codificação de canal para sistemas de comunicações ópticas que operam com taxas na ordem de 100 Gbit/s.

1.2 Motivação

O rápido crescimento da utilização de aplicações em rede de dados exige a evolução na mesma proporção dos sistemas ópticos para longas distâncias. Assim sendo, a motivação deste trabalho foi definir, projetar, implementar e verificar experimentalmente uma estrutura lógica de um código corretor de erros FEC com alto grau de paralelismos, que fosse capaz de operar com altas taxas de dados. Além disso, buscou-se uma arquitetura que respeitasse os requisitos da norma ITU-T G.709. Neste projeto específico do codificador/decodificador de erros, visou-se encontrar uma alternativa eficiente de implementar o algoritmo Reed-Solomon RS(255, 239) para operar com taxa de 100 Gbit/s.

1.3 Contribuição

O principal objetivo de contribuição deste trabalho é apresentar uma arquitetura paralela de FEC Reed-Solomon que seja capaz de operar em 100 Gbit/s e respeitar a norma ITU-T G.709. Além disso, buscou-se uma implementação que oferecesse uma eficiência em área e de recursos superior a uma concatenação de 10 circuitos de 10 Gbit/s, tanto para o codificador quanto para o decodificador. Na seção 6.2 é apresentada uma tabela com os recursos ocupados em FPGA para implementar um codificador/decodificador FEC de 10 Gbit/s e de 100 Gbit/s. Com essas informações pode-se comparar a quantidade de *slices* ocupados em um FPGA entre um decodificador FEC de 100 Gbit/s e um de 10 Gbit/s e observar que essa relação é de 3,5 vezes. Analogamente, se compararmos os codificadores de 100 Gbit/s e 10 Gbit/s, esta mesma relação é de 7,1 vezes.

A partir dos resultados obtidos neste trabalho foram publicados e apresentados três artigos em conferências internacionais. O primeiro artigo, intitulado “*Design and FPGA Implementation of a 100 Gbit/s Optical Transport Network Processor*”, foi publicado na conferência FPL2013 (*Field-Programmable Logic and Applications*), realizada em Porto, Portugal [5]. O segundo artigo, com o título “*100 Gbit/s FEC for OTN Protocol: Design Architecture and Implementation Results*” foi publicado no Simpósio ITS2014 em São Paulo, Brasil [6]. O terceiro artigo publicado tem o título “*111.18 Gbit/s G-FEC Design for OTN Applications: Theory and Implementation Techniques*” e foi publicado no congresso DesignCon 2014 em Santa Clara, Estados Unidos [7]. Este último trabalho foi indicado como finalista para o prêmio de “*Best Paper Awards 2014*” na

categoria de trabalhos “*High Speed Design*”.

Além dos artigos mencionados também foi depositado um pedido de patente com o título “*Método de Codificação FEC para Sistemas de Transmissão OTN com Altas Taxas de Transmissão*” e número de processo BR 10 2013009315 7 [8] no INPI (Instituto Nacional de Propriedade Intelectual).

Outra contribuição bastante relevante deste trabalho foi a geração de capacitação na área de teoria de códigos e implementação de circuitos de códigos corretores de erros para sistemas que operem em altas taxas de transmissão.

1.4 Estado da Arte

Algumas publicações e dissertações recentes [9][10][11][12][13] tratam deste mesmo assunto apresentando arquiteturas e descrições dos passos de implementação do codificador/decodificador FEC Reed-Solomon. Entretanto, estes trabalhos apresentam somente soluções para implementação serial deste circuito.

O artigo com o título “*VHDL Design and FPGA Implementation of Parallel Reed-Solomon (15, K, D) Encoder/Decoder*”[14], propõe uma contribuição semelhante à descrita neste trabalho, mas não descreve como o circuito foi implementado, apresentando somente os resultados de simulação.

Além dos artigos mencionados, foi realizada uma pesquisa por anterioridades de patentes depositadas. Uma arquitetura de processamento paralelo do codificador Reed-Solomon para utilização em altas taxas está descrita no pedido de patente EP0913949 (*Tezuka*) intitulado “*Device and Method for Carrying out Reed-Solomon Encoding*”[15]. Nesta invenção, considerando um algoritmo Reed-Solomon com número de coeficientes da palavra código igual a N , o número de coeficientes de mensagem igual a K e o número de elementos a serem processados paralelamente igual a M , para se obter o circuito do codificador paralelo pode se realizar o método descrito na patente. Contudo, este método vale somente para circuitos onde o número de coeficientes de paridade ($N-K$) seja divisível por M .

No documento de patente US6859905 (*Seki*) intitulado “*Parallel Processing Reed-Solomon Encoding Circuit and Method*”[16] é descrito um método que possibilita a implementação de um codificador paralelo para qualquer comprimento de palavra código N e número de processamen-

tos paralelos M . O método citado é bastante abrangente, mas não apresenta solução quando a mensagem (K) não é divisível por M .

A implementação do circuito codificador e decodificador FEC Reed-Solomon tratada nos capítulos seguintes busca apresentar a manipulação algébrica empregada para converter a estrutura serial para a paralela do FEC e a implementação do circuito Reed-Solomon independente do número de N , K e M .

1.5 Organização da Dissertação

No Capítulo 2 são introduzidas algumas características das redes de transporte óptico que estão diretamente relacionadas com a implementação do código corretor de erros descrita neste trabalho. As informações deste capítulo foram extraídas da recomendação ITU-T G.709 [17], que define as características e funcionalidades de uma rede de transporte óptico. Mais especificamente, são expostos um breve histórico das redes de transporte, as taxas empregadas em redes de transporte óptico, o esquema de entrelaçamento de dados para aumentar a eficiência de correção de erros e alguns requisitos impostos pela norma para a implementação do FEC.

No Capítulo 3, importantes conceitos sobre a aritmética em campo de Galois são apresentados. As operações matemáticas de adição, subtração, divisão e multiplicação baseadas nesta aritmética são detalhadas tendo sempre em vista a posterior implementação destes elementos na codificação e decodificação do algoritmo Reed-Solomon RS(255,239).

No Capítulo 4 é apresentada a teoria básica sobre códigos corretores de erros. De uma forma geral estes códigos são subdivididos em duas classes: ARQ e FEC. Além da descrição do FEC, são incluídos exemplos de códigos detectores de erros e códigos corretores de erros. Dentre alguns exemplos de algoritmos matemáticos de correção de erros, é descrito o conceito matemático do algoritmo Reed-Solomon.

O Capítulo 5 detalha o principal tema deste trabalho, que é a implementação de uma estrutura paralela do Codificador e do Decodificador FEC RS(255,239). Inicialmente é apresentado a estrutura convencional de cada módulo do FEC e as etapas para se convertê-las para uma arquitetura paralela. Além disto, é apresentado um exemplo de implementação de um codificador/decodificador FEC Reed-Solomon RS(255, 239) que processa 5 símbolos a cada ciclo de relógio.

O Capítulo 6 descreve como o circuito do FEC foi simulado e validado em laboratório. Os testes visaram comprovar o funcionamento do circuito. Cabe ressaltar que não é escopo deste trabalho fazer uma caracterização detalhada do FEC. Haja vista que trata-se de um algoritmo padronizado, cujo desempenho de ganho não depende de sua implementação. Ainda assim, foram realizados testes para comprovar os benefícios da utilização do FEC.

O Capítulo 7 descreve as conclusões do autor e propõe trabalhos futuros.

Redes Ópticas de Transporte

2.1 Histórico Sobre Redes de Transporte

Uma rede de transporte é uma plataforma tecnológica que permite a transferência fiável e transparente da informação à distância, permitindo suportar diferentes serviços. Além disso, uma rede de transporte deve garantir diferentes funcionalidades, como por exemplo transmissão, multiplexação, encaminhamento, proteção, supervisão e aprovisionamento de capacidade.

As primeiras redes de transporte de dados eram analógicas e utilizavam cabos coaxiais para transmissão. Na década de 70, surgiram os sistemas de transmissão digital utilizando a técnica **TDM** (*Time Division Multiplexing*), que proporcionou uma maior eficiência espectral. A técnica TDM foi concebida para o transporte de voz nas redes **PSTN** (*Public Switched Telephone Network*), utilizando a tecnologia **PDH** (*Plesyochronous Digital Hierarchy*). O padrão PDH de transmissão de sinais foi concebido para uma arquitetura de multiplexação assíncrona. Cada canal multiplexado opera de forma plesiócrona, ou seja, com um relógio que não é sincronizado com os relógios dos outros canais apesar de ser nominalmente idêntico, dentro de limites estabelecido por normas. [18].

Em decorrência de problemas operacionais do sistema PDH, um sistema mais flexível, com maior capacidade de gerenciamento e compatível com esta tecnologia, foi padronizado pelo ITU-T na década de 1980. Esse sistema foi denominado hierarquia digital síncrona, ou **SDH** (*Synchronous Digital Hierarchy*) [19].

Os primeiros esforços de desenvolvimento dessa nova hierarquia de transmissão foram feitos no Bellcore (*Bell Communications Research*) por meio do sistema denominado **SONET** (*Synchro-*

nous Optical Network). Com o surgimento e a popularização das redes de pacotes, em especial a **Ethernet**, os sistemas SDH foram sendo adaptados para realizar o transporte também desse tipo de tráfego. Consequentemente, surgiram novos tipos de equipamentos SDH, denominados SDH-NG (Next Generation).

Paralelamente ao surgimento dessas tecnologias, ocorreu a evolução das tecnologias ópticas e o aumento significativo do tráfego de dados, surgindo a necessidade de integração entre as redes de dados e as redes ópticas. Assim, iniciaram-se os estudos do padrão **OTN** (*Optical Transport Network*), que visava maximizar a eficiência dos sistemas de transmissão e proporcionar a integração IP/WDM. A primeira versão do padrão OTN foi publicada na década de 90 pelo ITU-T (G.709). Neste momento foi especificado um envelope digital para encapsular diversos tipos de protocolos, como, por exemplo, SDH, ATM e Ethernet, e transportá-los na OTN. No entanto, com o crescente desenvolvimento da tecnologia Ethernet, especialmente as tecnologias de 40 e 100 Gigabit Ethernet (GbE), os estudos em OTN foram reiniciados e uma nova versão da norma G.709 foi aprovada em dezembro de 2009 [17]. Nessa nova versão, são propostas novas capacidades e o suporte para altas taxas da Ethernet [20].

Uma das principais vantagens da utilização da OTN é possibilitar o uso eficiente dos comprimentos de onda dos sistemas **WDM** (*Wavelength Division Multiplexing*), efetuando a agregação de diversos tipos de tráfego.

Finalmente, a OTN possibilita o transporte de serviços de dados com maior granularidade. Com isso, é possível criar rotas fim a fim, que se originam na rede de dados, passam pela rede óptica e terminam na rede de dados.

2.2 Hierarquia da OTN

A OTN é estruturada de forma hierárquica em três camadas de rede: **OCh** (*Optical Channel*), **OMS** (*Optical Multiplex Section*) e **OTS** (*Optical Transmission Section*), conforme Figura 2.1.

A camada OCh fornece conectividade fim a fim para os canais ópticos, para então transmitir, de forma transparente, as informações do cliente em vários formatos, como, por exemplo, STM-N, PDH, Ethernet, etc. A camada OMS fornece funcionalidade de rede de um sinal óptico de múltiplos comprimentos de onda (cor), assegurando a integridade dos diversos comprimentos

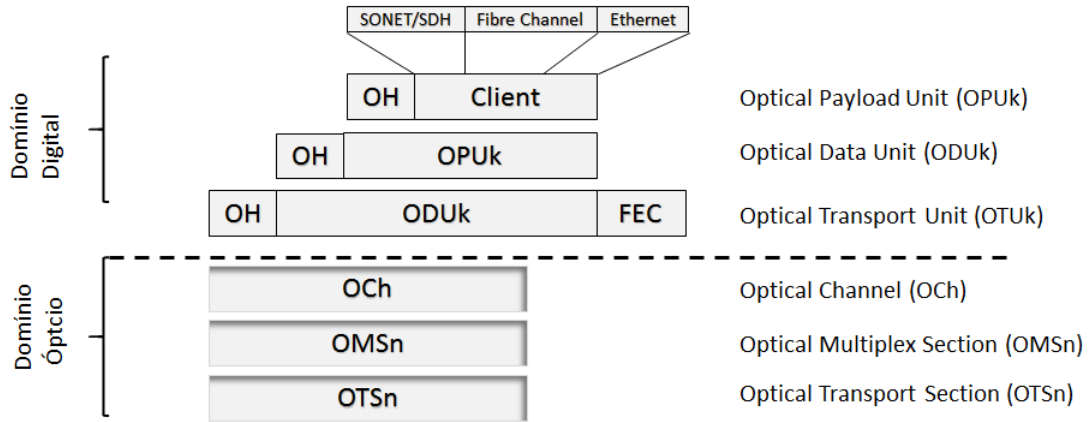


Figura 2.1: Hierarquia OTN.

de onda das informações da OMS. Já a camada OTS fornece funcionalidade para transmissão de sinais ópticos em meios ópticos de diferentes tipos (por exemplo, fibras específicas). Dessa forma, diversos canais podem ser mapeados dentro da OMS e, então, transportados via camada OTS.

Embora essas três camadas da parte óptica realizem boa parte das funcionalidades de transporte da OTN, segundo a recomendação ITU-T G.872, nem todos os requerimentos de gerenciamento podem ser satisfeitos pelos mecanismos providos na camada óptica como, por exemplo, uma avaliação quantitativa da qualidade do sinal.

Por esta razão, a recomendação ITU-T G.709 [17] define também um conjunto de subcamadas na hierarquia OTN, conforme Figura 2.2. As subcamadas **OPU_k** (*Optical Payload Unit*), **ODU_k** (*Optical Data Unit*) e **OTU_k** (*Optical Transport Unit*) – onde $k = 1, 2, 3$ e 4 indica a taxa de operação, também identificado como quadros – estão no domínio elétrico. A função de cada subcamada é descrita na Seção 2.4.

2.3 Taxas de Transmissão

Conforme mencionado, a norma OTN suporta o transporte de clientes com diferentes taxas de transmissão. As taxas padronizadas para os OPU's, de acordo com a G.709, estão indicadas na Tabela 2.1, com a inclusão de algumas variantes [21]:

- (a) **OPU2e**, onde “e” significa extended ou over-clocked, que opera com a taxa base do padrão 10 GbE;

- (b) **OPU0**, que é utilizado para mapeamento de quadros 1 GbE; e
- (c) **ODUflex**, que pode ser utilizado para mapear outros tipos de cliente.

Tabela 2.1: Taxa de Dados Para os Quadros OPUk Padronizados pela G.709

Tipo	Taxa Nominal (kb/s)	Tolerância
OPU0	$(238/239) \cdot 1.244,16$	+/-20 ppm
OPU1	2.488,32	+/-20 ppm
OPU2	$(238/237) \cdot 9.953,28$	+/-20 ppm
OPU3	$(238/236) \cdot 39.813,12$	+/-20 ppm
OPU4	$(238/227) \cdot 99.532,80$	+/-20 ppm
OPU2e	$(238/237) \cdot 10.312,50$	+/-100 ppm
OPUflex	taxa do cliente	cliente

A Tabela 2.2 apresenta as taxas de transmissão dos quadros ODUk.

Tabela 2.2: Taxa de Dados Para os Quadros ODUk Padronizados pela G.709

Tipo	Taxa Nominal (kb/s)	Tolerância
ODU0	1.244,16	+/-20 ppm
ODU1	$(239/238) \times 2.488,32$	+/-20 ppm
ODU2	$(239/237) \cdot 9.953,28$	+/-20 ppm
ODU3	$(239/236) \cdot 39.813,12$	+/-20 ppm
ODU4	$(239/227) \cdot 99.532,80$	+/-20 ppm
ODU2e	$(239/237) \cdot 10.312,50$	+/-100 ppm
ODUflex	taxa do cliente	cliente

A Tabela 2.3 apresenta as diferentes taxas e a tolerância do quadro OTUk.

Tabela 2.3: Taxa de Dados Para os Quadros OTUk Padronizados pela G.709

Tipo	Taxa Nominal (kb/s)	Tolerância
OTU1	$(255/238) \times 2.488,32$	+/-20 ppm
OTU2	$(255/237) \cdot 9.953,28$	+/-20 ppm
OTU3	$(255/236) \cdot 39.813,12$	+/-20 ppm
OTU4	$(255/227) \cdot 99.532,80$	+/-20 ppm

2.4 Estrutura do Quadros OTN

A estrutura do quadro OTN é muito similar à utilizada no quadro SONET e possui 6 campos distintos, conforme apresentado na Figura 2.2. A seguir são listados e descritos cada item do quadro.

Unidade de carga do canal óptico (Optical Payload Unit - OPU). A estrutura de quadros OPU contém a carga de sinais do cliente (OPUk) e a sobrecarga (OPUk OH) necessária para o mapeamento de qualquer sinal do cliente na OPU. O mapeamento dos sinais do cliente pode incluir a adaptação de taxa de sinal de cliente para um para um sinal de taxa de *bits* constante. Os exemplos de sinais comuns são Ethernet, Fibre Channel e SONET/SDH. Conforme mencionado anteriormente, as 4 taxas de carga associadas a OPU são de 2,5 Gbit/s, 10 Gbit/s, 40 Gbit/s e 100 Gbit/s.

Unidade de dados do canal óptico (Optical Data Unit - ODU). A ODU é a unidade utilizada para transportar a OPU. A ODU é formada pela OPU e o cabeçalho de ODU (ODU OH). Este cabeçalho fornece funções de monitoramento, manutenção e operação dos canais ópticos.

Unidade de transporte do canal óptico (Optical Transport Unit - OTU). A OTU contém a estrutura ODU, a sobrecarga OTU (OTU OH) e o campo com os dados de paridade para a correção de erro adiante (FEC). A OTU fornece funções de monitoramento de conexão e de correção e detecção de erros.

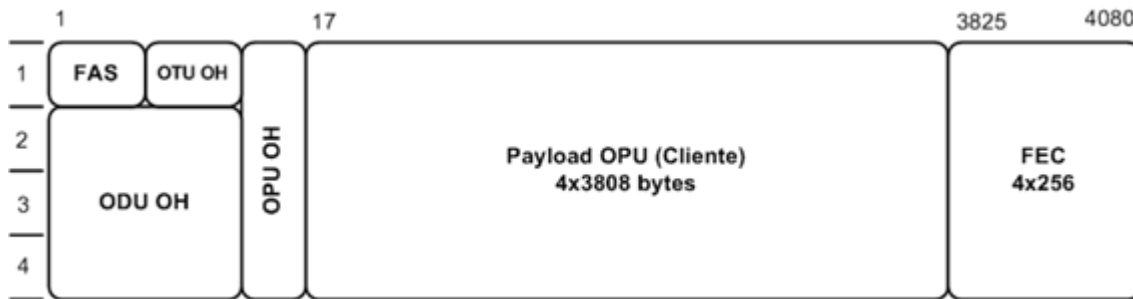


Figura 2.2: Estrutura de Quadros OTN.

Além dos campos funcionais descritos no parágrafo anterior, a OTU também possui um campo chamado de cabeçalho de alinhamento, **FAS** (*Frame Alignment Signal*). Esta sobrecarga é sub-dividida em dois campos. O primeiro campo (FAS) é formado por seis *bytes* com valores fixos, F6F6F6282828 em hexadecimal, e é utilizado na recepção no processo de alinhamento do quadro. O segundo campo é o alinhador de multiquadro **MFAS** (*Multi-Frame Alignment*), que é um *byte* com valor incremental entre 0 e 255. A cada novo quadro o valor de MFAS é incrementado de 1 e uma sequência de 256 quadros formam um multiquadro.

2.5 *Forward Error Correction* (FEC)

Um código corretor de erro consiste de uma técnica para reduzir a taxa de erro de bit no canal de comunicação. O método envolve a transmissão de informação adicional, chamados de redundância, anexada a mensagem que se deseja transmitir. O termo *Forward Error Correction* pode ser traduzido como correção de erros adiante, o que significa que a decodificação dos erros é realizada após a transmissão dos dados.

Conforme mencionado na Seção 2.1, a primeira versão do padrão definido para redes ópticas foi publicada na década de 90 pelo ITU-T. Neste mesmo período o Grupo de Estudos 15 da ITU-T também elaborou a recomendação para o uso do FEC em sistemas ópticos submarinos, de onde originou-se a Recomendação ITU-T G.975. Este documento especifica o algoritmo Reed-Solomon RS(255,239) como um código adequado para aplicações em redes ópticas. Este código corretor é classificado como um código de bloco e possui um ganho de correção de 6,3 dB (NEGC - Net Electrical Coding-Gain), sob uma condição de taxa de erro (BER) de 10^{-15} . Posteriormente uma nova versão da Recomendação ITU-T G.975.1 foi publicada com 9 novos esquemas de codificação FEC com ganhos superiores ao RS(255,239), os chamados super-FEC. O esquema apresentado no Anexo I.9 possui o melhor desempenho entre as 8 estruturas propostas. Este código consiste de dois códigos BCH(1020,988) entrelaçados e com um mesmo número de *bits* por palavra-código. O ganho de codificação deste tipo de estrutura de FEC altera-se de acordo com o número de iterações. Conforme apresentado em [22], após 10 iterações e submetido a uma BER de 10^{-13} , o ganho de codificação do E-FEC I.9 é melhor do que 8.5 dB. Por esta razão, a classe de códigos corretores super-FEC tornou-se uma alternativa de melhoria do desempenho dos sistemas de transmissões ópticas em resposta ao crescimento da demanda de consumo.

O presente trabalho descreve uma estrutura paralela para o código corretor de erros RS(255,239) para operar com taxas de 100 Gbit/s em redes ópticas.

Para aumentar o desempenho da correção de erros, a norma OTN prevê a da técnica de entrelaçamento de dados (*Interleaving*). Esta funcionalidade redistribui os erros de maneira mais uniforme entre os blocos codificados e aumenta a robustez a erro em rajada. Em aplicações OTN o entrelaçamento é implementado distribuindo os *bytes* de cada linha do quadro OTN entre 16 sublinhas, ou 16 palavras código. Cada sublinha do quadro é tratada de forma independente em 16 codificadores/decodificadores FEC. Isto pode evitar que a ocorrência de erros em rajada

sobrecarregue um único decodificador. Assim, o entrelaçamento faz com que na recepção os erros em rajada sejam distribuídos uniformemente entre os decodificadores FEC.

A Figura 2.3 ilustra o esquema de entrelaçamento dos dados de acordo com a recomendação ITU-T G.709.

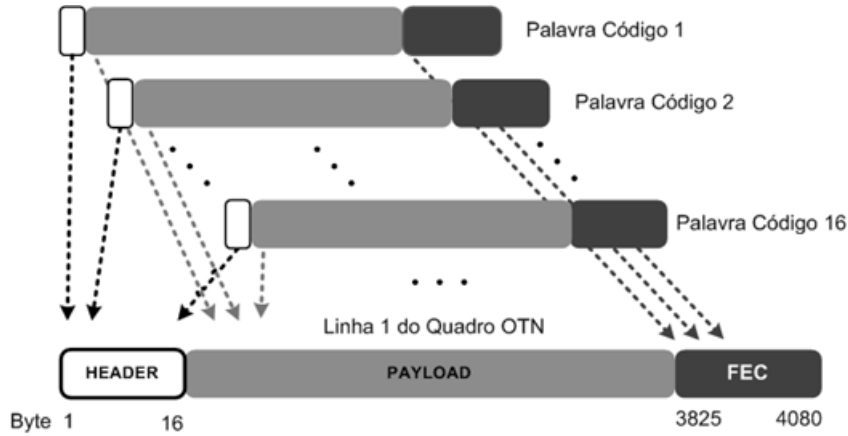


Figura 2.3: Entrelaçamento de Dados.

A equação

$$Sub_Linha[sl][p] = Bytes_Linha_OTN[sl + (16 \times p)], \quad (2.1)$$

descreve como os *bytes* de uma linha do quadro OTN são distribuídos para cada uma das 16 sublinhas do quadro. Onde sl indica a sublinha do quadro OTN, portanto ($sl = 1, 2, 3, \dots, 16$); e p indica a posição em que o *byte* será alocado na sublinha, portanto ($p = 0, 1, 2, \dots, 254$). Por exemplo, a sublinha 1 ($sb = 1$) receberá os *bytes* das posições (1, 17, 33, \dots , 4065) de uma linha do quadro OTN.

Além de aumentar a capacidade de correção de erros em rajada, o entrelaçamento também permite o aumento da taxa de processamento do algoritmo. Isto acontece porque 16 processadores FEC funcionam de forma paralela.

A Figura 2.4 apresenta como os dados são distribuídos para os 16 codificadores em um esquema de processamento serial. Inicialmente os *bytes* de uma linha do quadro OTN são recebidas de forma serial e cada símbolo é encaminhado para um codificador específico para implementar o entrelaçamento.

A Figura 2.5 apresenta como os dados são distribuídos para os 16 codificadores em um esquema de processamento paralelo, onde cada codificador FEC necessita processar N *bytes* por ciclo de relógio para possibilitar o fluxo de dados do sistema. Neste exemplo, são processados

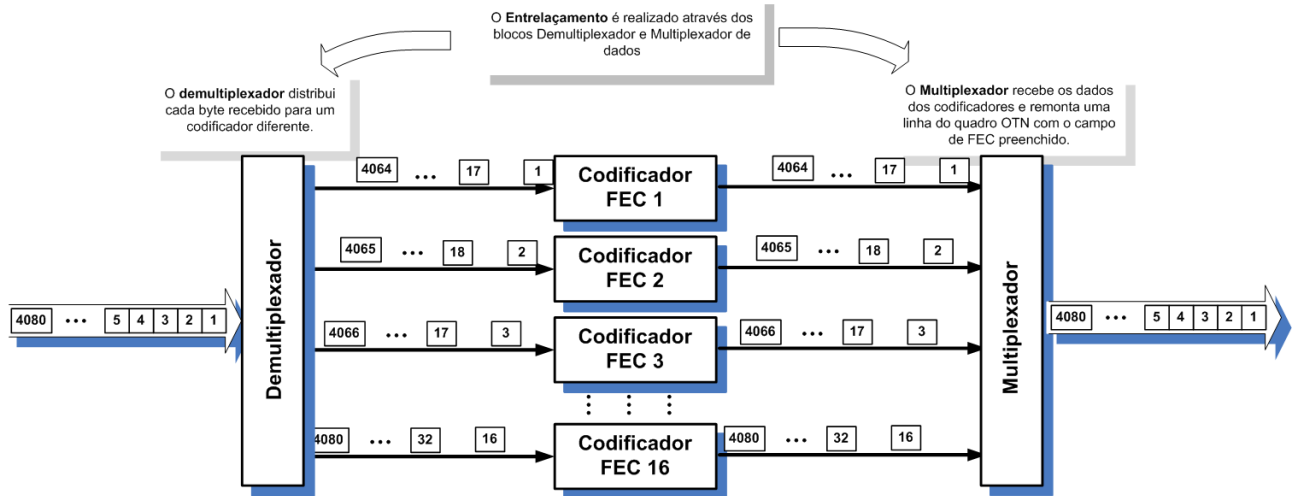


Figura 2.4: Entrelaçamento de Dados no Processamento Serial.

5 bytes por ciclo de relógio, portanto N é igual a 5. Para o decodificador FEC, o esquema de entrelaçamento é exatamente igual ao do codificador.

Nesta arquitetura, os dados são distribuídos através de demultiplexadores e todos os 16 codificadores possuem o mesmo circuito. Portanto, o Capítulo 5 fornece informações somente de como um codificador é implementado para processar os bytes de forma serial e paralela.

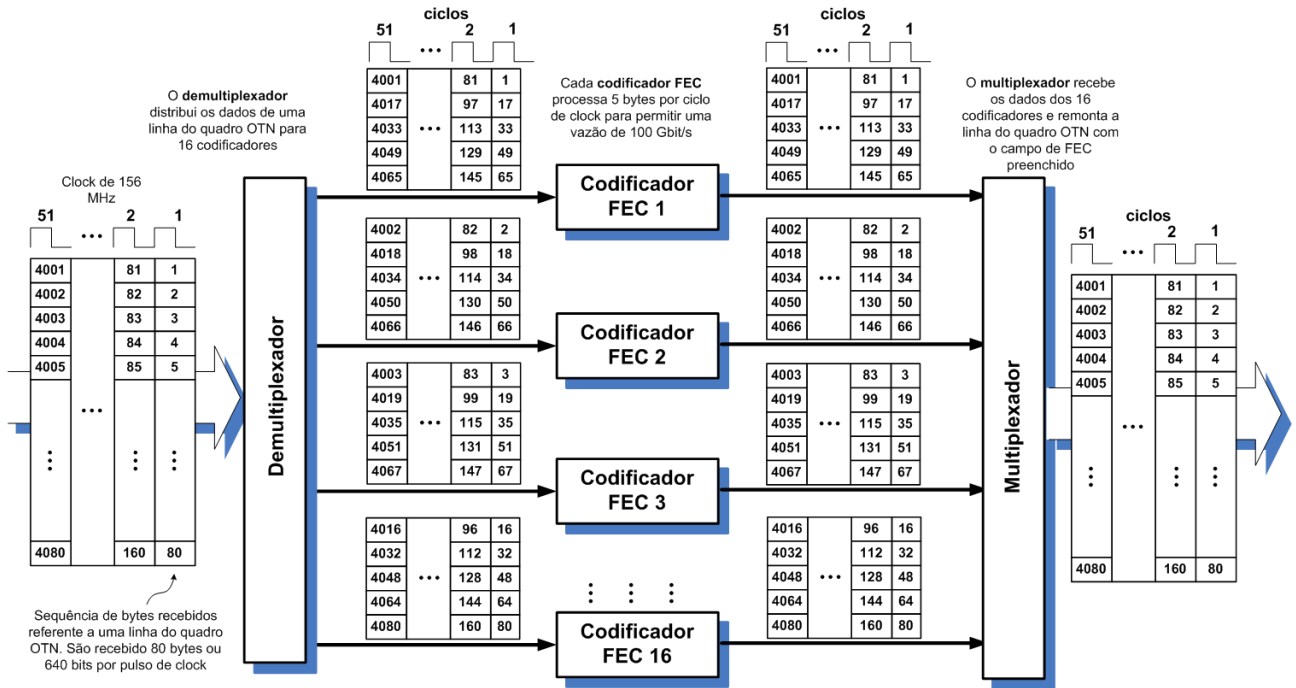


Figura 2.5: Entrelaçamento de Dados no Processamento Paralelo.

Como exemplo do benefício gerado pelo entrelaçamento, considere que 64 palavras código

com capacidade de correção de 8 *bytes* cada sejam entrelaçadas. O resultado é que o grupo de dados entrelaçados possibilita a correção de qualquer combinação de erros dentro de uma faixa de 512 *bytes* [23].

Álgebra de Campos Finitos

Para entender os princípios de codificação e decodificação de códigos não-binários, é necessária uma introdução aos **campos finitos**, também conhecidos como **campos de Galois** (*Galois Fields*, GF). Esta estrutura algébrica é denotada por $GF(q)$ ou \mathbb{F}_q , onde q é um número primo p ou a potência de um número primo p . Um campo $GF(q)$ pode ser definido como um conjunto com n elementos, fechado com relação às operações de adição e multiplicação, e para qual todo elemento tem seu inverso aditivo e multiplicativo, com a exceção do número 0, que não tem inverso multiplicativo [24]. Antes da definição formal de campos de finitos, serão apresentados os conceitos e as propriedades das estruturas Grupo, Anel e Campo.

3.1 Conceitos Básicos

3.1.1 Grupo

Um **Grupo** (G, \cdot) consiste de um conjunto G com uma operação binária (\cdot) , em que G satisfaça as seguintes condições:

- (i) Propriedade associatividade: $a \cdot (b \cdot c) = (a \cdot b) \cdot c, \forall a, \forall b \text{ e } \forall c \in G$.
- (ii) Propriedade identidade: $\exists e \in G: a \cdot e = e \cdot a = a, \forall a \in G$.
- (iii) Propriedade inversa: $a \cdot a^{-1} = a^{-1} \cdot a = e, \forall a \in G, \forall a^{-1} \in G$.

Os grupos podem ser de ordem infinita ou finita. A ordem de um grupo G é determinada por $ord(G)$. Entretanto neste trabalho será considerado somente grupos de ordem finita.

3.1.2 Anel

Anel pode ser definido como um conjunto não vazio A e um par de operações, respectivamente uma adição $(x,y) \rightarrow x+y$ e uma multiplicação $(x,y) \rightarrow x \cdot y$, é chamado de anel se as seguintes condições forem satisfeitas:

- (i) Propriedade associativa da adição: $a \cdot (b + c) = a \cdot b + a \cdot c, \forall a, \forall b, \forall c \in A$.
- (ii) Propriedade comutativa da adição: $a + b = b + a, \forall a, \forall b \in A$.
- (iii) Propriedade identidade da adição: $\exists e \in A \mid a + e = e + a = a, \forall a \in A$.
- (iv) Propriedade inversa da adição: $\forall a \in A, \exists -a \in A \mid a + (-a) = (-a) + a = 0$.
- (v) Propriedade distributiva: $\forall a, b, c \in A$, pode-se definir que $a \cdot (b + c) = a \cdot b + a \cdot c$, e que $(b + c) \cdot a = b \cdot a + c \cdot a$.
- (vi) Propriedade associativa da multiplicação: $\forall a, b, c \in A$, pode-se definir que $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

3.1.3 Campo

A partir da definição de anel, pode-se definir **Campo** se forem verificadas as seguintes condições:

- (i) $(F, +, \cdot)$ é um anel.
- (ii) Propriedade comutativa da multiplicação: $\forall a, b \in F, a \cdot b = b \cdot a$.
- (iii) Propriedade identidade da multiplicação: $\exists e \in A \mid a + e = e + a = a, \forall a \in F$.
- (iv) Propriedade inversa da multiplicação: Se $a \in F$ e $a \neq 0, \exists b \in F \mid a \cdot b = b \cdot a = 1$.

A definição de Campo é bastante parecida com a definição de Anel, apresentada anteriormente. A definição de campo pode ser escrita da seguinte forma: **“Um Campo é um anel comutativo com identidade, no qual cada elemento tem um inverso multiplicativo.”**

3.2 Campos Finitos

3.2.1 Definição de Campo Finito

De uma forma geral, **Campo Finito** ou **Campo de Galois** é um conjunto de elementos com os quais pode-se aplicar operações de soma, subtração, multiplicação e divisão, sem sair deste conjunto. A ordem de um campo finito define o número de elementos contidos neste campo, que é sempre um número primo ou a potência de um número primo. Para todo número primo p , existe um campo finito de p elementos. Na verdade, para qualquer número inteiro positivo m , é possível estender o campo primo $GF(p)$, para um campo estendido com p^m elementos, denotado por $GF(p^m)$. Contudo, códigos com símbolos binários gerados a partir de um campo binário $GF(2)$ e de sua extensão $GF(2^m)$ são amplamente utilizados em sistemas de transmissão digital [25]. Por esta razão, nas seções seguintes serão tratados somente campos dessa natureza.

3.2.2 Propriedades de um Campo de Finito

As principais propriedades de um campo finito são [26]:

- (i) A todos os elementos do GF são definidas duas operações, chamadas adição (+) e multiplicação (\times).
- (ii) O resultado da adição ou multiplicação entre dois elementos de um GF de ser um elemento do GF .
- (iii) A identidade da adição de “zero” deve existir, de tal modo que $a + 0 = a$ para qualquer elemento a dentro do campo.
- (iv) A identidade da multiplicação de “um” deve existir, de tal modo que $a \times 1 = a$ para qualquer elemento a dentro do campo.
- (v) Para todo elemento a do campo, existe um complemento da adição do elemento, de tal modo que $a + b = 0$. Isto permite a operação de subtração seja definida como a adição de complemento.
- (vi) Para qualquer elemento b do campo GF diferente de zero, existe uma inversa da multiplicação do elemento b^{-1} , de tal modo que $bb^{-1} = 1$. Isto permite que a operação de divisão seja definida como a multiplicação por uma inversa.

- (vii) Ambas as operações de adição e multiplicação devem satisfazer às leis comutativas, distributivas e associativas.

3.2.3 Campo Finito Binário $GF(2)$

O campo de Galois mais simples que se pode ter é definido por $GF(2)$. Os elementos deste conjunto são $\{0,1\}$ e por esta razão são chamados de **campo de Galois binário**. Neste caso o campo é de ordem 2. As possíveis operações entre esses elementos são apresentadas nas tabelas 3.1 e 3.2.

A adição é implementada com uma simples operação ou-exclusivo entre os dois elementos do campo ($i + j = r$), cujo operador também é representado por \oplus . A operação de subtração produz o mesmo efeito de uma soma e também é implementada com lógica ou-exclusivo.

O resultado da soma entre dois elementos em um campo de Galois binário pode ser observado na Tabela 3.1.

Tabela 3.1: Resultado da Adição Entre os Elementos de um $GF(2)$

+	0	1
0	0	1
1	1	0

A multiplicação em campo de Galois entre dois elementos i e j é definida como ($i \times j = r$), cujo operador também é representado por \otimes . A operação módulo é também utilizada na multiplicação para garantir que o resultado da operação esteja contido no campo finito. A equação de multiplicação em campo finito é denotada por

$$r = i \times j \mod p(X).$$

Para o grupo binário com os elementos $\{0,1\}$ o resultado r da multiplicação entre os elementos estão apresentados na Tabela 3.2

Tabela 3.2: Resultado da Multiplicação Entre os Elementos de um $GF(2)$

\times	0	1
0	0	0
1	0	1

Outra operação importante em um campo finito é a inversão. Lembrando que a inversa de um elemento $a \neq 0 \in \text{GF}(p)$, é denotado por $a^{-1} \bmod p$, ou simplesmente a^{-1} . Assim, conforme definido pela propriedade (vi) de campos finitos, a inversa de um elemento a pertencente ao campo finito $\text{GF}(q)$ corresponde a um elemento que multiplicado por a resulta em 1 ($a \cdot a^{-1} = 1$). Portanto, esta operação não é definida para o elemento 0 do campo. A inversão é importante porque permite que a operação de divisão seja definida como a multiplicação por uma inversa. Estes elementos binários de um campo finito também podem ser representados na forma de polinômios. Normalmente, um polinômio de grau k pode ser representado da seguinte forma

$$f(x) = f_k x^k + f_{k-1} x^{k-1} + \dots + f_2 x^2 + f_1 x^1 + f_0, \quad (3.1)$$

onde os coeficientes f_0, f_1, \dots, f_k , correspondem aos elementos do campo. Para um campo binário, estes coeficientes podem assumir os valores 1 ou 0. Um número binário de $(k+1)$ *bits* pode ser representado através de um polinômio de grau k .

Deste modo, pode-se representar sequência binária $(1, 0, 1, 1, 1, 0, 1)$ através do polinômio

$$f(x) = x^6 + x^4 + x^3 + x^2 + 1. \quad (3.2)$$

3.2.4 Campo Binário Estendido $\text{GF}(2^m)$

O campo de Galois $\text{GF}(2^m)$ é gerado a partir da extensão do campo $\text{GF}(2)$ e construído a partir de um polinômio primitivo de grau m . O campo $\text{GF}(2^m)$ é definido como um campo com 2^m elementos com m *bits* cada um. Partindo desta definição, os m *bits* de um dado binário podem ser agrupados e referidos como um elemento do campo de Galois $\text{GF}(2^m)$.

Para se criar um campo de Galois, primeiramente é necessário escolher um polinômio primitivo com a mesma ordem do campo que se deseja. Um polinômio irreduzível $p(X)$ de grau m é dito primitivo se o menor valor positivo n para o qual $p(X)$ divide $X^n + 1$ é $n = 2^m - 1$. Por exemplo, pode-se verificar que o polinômio $p(X) = X^4 + X + 1$ divide $X^{15} + 1$, mas não divide outro $X^n + 1$ para $1 \leq n < 15$. Portanto, $X^4 + X + 1$ é um polinômio primitivo.

Não é trivial reconhecer um polinômio primitivo. Para facilitar a geração de um campo de ordem 2^m , existem tabelas que apresentam uma lista de polinômios primitivos de grau m . A Tabela 3.3 apresenta os polinômios primitivos utilizados na geração de campos de Galois [27].

Tabela 3.3: Lista de Polinômios Primitivos

m	Polinômio Primitivo	m	Polinômio Primitivo
3	$X^3 + X + 1$	14	$X^{14} + X^{10} + X^6 + X + 1$
4	$X^4 + X + 1$	15	$X^{15} + X + 1$
5	$X^5 + X^2 + 1$	16	$X^{16} + X^{12} + X^3 + X + 1$
6	$X^6 + X + 1$	17	$X^{17} + X^3 + 1$
7	$X^7 + X^3 + 1$	18	$X^{18} + X^7 + 1$
8	$X^8 + X^4 + X^3 + X^2 + 1$	19	$X^{19} + X^5 + X^2 + X + 1$
9	$X^9 + X^4 + 1$	20	$X^{20} + X^3 + 1$
10	$X^{10} + X^3 + 1$	21	$X^{21} + X^2 + 1$
11	$X^{11}X^2 + 1$	22	$X^{22} + X + 1$
12	$X^{12} + X^6 + X^4 + X + 1$	23	$X^{23} + X^5 + 1$
13	$X^{15} + X^4 + X^3 + X + 1$	24	$X^{24} + X^7 + X^2 + X + 1$

Como exemplo, será construído um campo de Galois com m igual a 3, portanto $GF(8)$. Neste caso, o campo possui os elementos 0, 1, 2, 3, 4, 5, 6, 7. Da Tabela 3.3, o polinômio primitivo escolhido para gerar o campo finito é dado por

$$p(X) = X^3 + X + 1. \quad (3.3)$$

O campo $GF(2^3)$ pode ser tratado como um campo estendido de $GF(2)$. Assim como no exemplo para o campo de Galois $GF(2)$, podemos inicialmente gerar uma tabela como o resultados das operações de soma e subtração entre os elementos deste campo. Conforme já mencionado, uma soma pode ser facilmente implementada em campo finito através de uma operação ou-exclusivo entre os coeficientes binários dos elementos do cálculo.

Tabela 3.4: Resultado da Adição Entre os Elementos de um $GF(2^3)$

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Tabela 3.5: Resultado da Multiplicação Entre os Elementos de um $GF(2^3)$

X	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Qualquer elemento do campo pode ser expresso por

$$a_i(\alpha) = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + \dots + a_{i(m-1)}\alpha^{i(m-1)}. \quad (3.4)$$

Portanto, todos os elementos deste campo podem ser gerados como uma potência de α . Esta representação é então, utilizada para processamento matemático que utiliza aritmética de campos Finitos.

A tabela seguinte mostra três formas diferentes de representar os elementos no $GF(2^3)$.

Tabela 3.6: Representações dos Elementos em $GF(2^3)$

Forma Potência	Forma Polinomial	Forma Binária $\alpha^0, \alpha^1, \alpha^2$
0	0	000
1	1	100
α^1	α	010
α^2	α^2	001
α^3	$1+\alpha$	110
α^4	$\alpha+\alpha^2$	011
α^5	$1+\alpha+\alpha^2$	111
α^6	$1+\alpha^2$	101

A primeira coluna da Tabela 3.6 representa a potência de α . A segunda coluna apresenta a representação polinomial dos elementos do campo. Esta representação é obtida a partir da equação 3.3. Por fim, a terceira coluna é a representação binária dos elementos do campo.

Teoria Básica Sobre Códigos Corretores de Erro

Neste capítulo são apresentados conceitos fundamentais de Teoria de Códigos, que permitirão a posterior introdução dos algoritmos mais comuns de correção de erros. A primeira etapa para recuperar a informação original transmitida consiste em detectar a presença de erro na mensagem recebida. Uma vez encontrada, a recuperação da mensagem pode ser realizada de duas formas:

1. O receptor solicita ao transmissor um novo envio da mensagem. Esta técnica é também conhecida como **Automatic Repeat Request** (ARQ).
2. Utilizando uma estrutura de correção de erros diretamente no receptor. Esta técnica é conhecida como **Forward Error Correction** (FEC), ou correção de erros adiante.

A simples detecção de erro requer uma complexidade menor no processamento da redundância a ser adicionada a mensagem. Por outro lado, algoritmos de FEC necessitam de um número maior de dados redundantes e, por conseguinte, uma taxa de transmissão mais elevada. Existem duas classes de códigos FEC: **Códigos de Bloco e Códigos Convolucionais**.

O FEC recomendado pela norma ITU-T G.709 pertence à classe dos Códigos de Bloco. Por esta razão, a seguir serão apresentados conceitos e exemplos de códigos detectores e corretores de erro pertencentes somente a este grupo.

4.1 Conceitos Fundamentais de Códigos Corretores de Erro

As subseções seguintes apresentam alguns conceitos importantes sobre a Teoria de Códigos Corretores de Erro pertinentes ao conjunto de códigos de bloco.

4.1.1 Códigos de Bloco

Em um **Código de Bloco**, o codificador divide a mensagem em segmentos de dados de k *bits* consecutivos de informação. Quando este segmento da mensagem é representado por um conjunto de *bits*, o código é chamado de **Código Binário**, onde cada símbolo pode assumir os valores lógicos 1 ou 0. Diferentemente, se cada símbolo deste segmento de mensagem for representado por uma palavra binária, o código é chamado de **Código Não-Binário**.

Para mensagens representadas por blocos de k *bits*, existirão 2^k mensagens possíveis. Nas seções seguintes um vetor com os *bits* de mensagem são representados por vetores do tipo $\bar{u} = (u_0, u_1, \dots, u_{k-1})$ pertencentes a um espaço vetorial U_k , onde $u_i=1$ ou 0. A função do codificador é transformar um bloco de k *bits* de mensagem em um novo vetor de n *bits*. Assim, existem 2^k palavras codificadas, chamadas de **Palavras Código** correspondentes. Este conjunto de 2^k palavras código é representado por vetores do tipo $\bar{v} = (v_0, v_1, \dots, v_{n-1})$, pertencentes a um espaço vetorial V_n , onde $v_i=0$ ou 1. Assim, U_k é um subconjunto do espaço vetorial V_n .

A Figura ?? ilustra a formação de uma palavra código.

Genericamente um código de bloco pode ser resumido por um mnemônico do tipo $C(n,k)$, onde C contém a(s) letra(s) que definem o algoritmo empregado, n é o número de símbolos da palavra código e k é o número de símbolos do bloco de mensagem.

Assim, ainda que uma palavra código tenha um número maior de *bits* do que a mensagem, um código de bloco é capaz de representar somente 2^k símbolos diferentes de palavras codificadas.

Um código de bloco de comprimento n e 2^k palavras código é considerado um **Código de Bloco Linear** se, e somente se, as suas 2^k palavras código formarem um subespaço dimensional k de um espaço vetorial de todos os 2^n palavras sobre o campo $GF(2)$.

Portanto, os 2^k símbolos são um subconjunto dos 2^n símbolos possíveis do espaço vetorial V_n . Em teoria de códigos diz-se que as 2^k palavras código pertencem a um subespaço do espaço vetorial V_n . Ainda se pode afirmar que, para ser considerado um código de bloco linear, qualquer

operação matemática realizada entre duas palavras código devem resultar em uma palavra código pertencente ao espaço vetorial V_n .

Por exemplo, para um codificador hipotético de bloco linear $C(6,3)$, o subespaço vetorial de 2^k símbolos está apresentado na Tabela 4.1.

Tabela 4.1: Palavras Código para $C(6,3)$

Mensagem (vetores \bar{u})	Palavra Código (vetores \bar{v})
000	000 000
001	001 101
010	010 011
011	011 001
100	100 110
101	101 011
110	110 101
111	111 000

4.1.2 Códigos Cíclicos

Um código é considerado **Código Cíclico** se, ao se efetuar um deslocamento cíclico, ou rotação, de uma palavra código para a direita, por exemplo, o resultado também pertencer ao espaço vetorial V_n .

Esta classe de código pode ser eficientemente codificada e decodificada usando registradores de deslocamento e lógica combinacional.

Para facilitar a manipulação e a descrição dos códigos, normalmente os vetores de dados são representados na forma polinomial

$$V(X) = b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + \dots + b_2X^2 + b_1X + b_0, \quad (4.1)$$

onde os coeficientes $b_{n-1}, b_{n-2}, \dots, b_1, b_0$ correspondem ao valor binário (1 ou 0) de cada elemento do vetor de dados. Por exemplo, considerando a palavra código representada pelo vetor $\bar{v} = (110101)$, seu polinômio correspondente resulta em

$$V(X) = X^5 + X^4 + X^2 + 1. \quad (4.2)$$

O polinômio 4.2, por possuir sempre coeficientes com valor 1 ou 0, é considerado como um código binário. Por outro lado, se o código for não-binário, sua representação polinomial poderia ser, por exemplo, da seguinte forma

$$V(x) = 10X^5 + 7X^3 + 3X^4 + 2X^2 + 3X + 1, \quad (4.3)$$

onde cada coeficiente do polinômio $V(x)$ corresponde a um símbolo binário representado com m -bits. Como muitos códigos utilizam divisão polinomial no seu processamento, este artifício torna-se bastante útil para o cálculo das palavras códigos.

4.1.3 Matriz Geradora

Os símbolos pertencentes a um código de bloco $C(n, k)$ são gerados a partir da multiplicação entre o vetor mensagem \bar{u} e uma matriz geradora G . A matriz geradora G tem uma dimensão $k \times n$, conforme a expressão

$$G = \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \vdots \\ \bar{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & g_{0,2} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & g_{1,2} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{bmatrix},$$

onde $\bar{g}_i = (g_{i,0}, g_{i,1}, g_{i,2}, \dots, g_{i,n-1})$, para $0 \leq i \leq k-1$.

O cálculo de uma palavra código \bar{v} de C , pode ser generalizada de acordo com a equação

$$\bar{v} = u_0 \cdot \bar{g}_0 + u_1 \cdot \bar{g}_1 + u_2 \cdot \bar{g}_2 + \cdots + u_{k-1} \cdot \bar{g}_{k-1}, \quad (4.4)$$

onde $u_i = 1$ ou 0 , para $0 \leq i \leq k-1$.

Se $\bar{u}_i = (u_0, u_1, u_2, \dots, u_{k-1})$, é o vetor mensagem a ser codificada, a correspondente palavra código é obtida da seguinte forma:

$$\bar{v} = \bar{u} \cdot G = [u_0, u_1, \dots, u_{k-1}] \cdot \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \vdots \\ \bar{g}_{k-1} \end{bmatrix}$$

A multiplicação entre os vetores mensagens e a matriz geradora G gera 2^k vetores independentes. Uma outra forma de determinar o número máximo de palavras códigos, ou vetores independentes, que podem ser obtidos a partir da matriz geradora é a partir da operação $\text{rank}(G)$. A matriz geradora pode ser então expressa na forma

$$G = [I_k | P]. \quad (4.5)$$

Essa matriz geradora G possui duas submatrizes. A primeira, I_k , corresponde a uma matriz identidade $k \times k$ e a segunda submatriz, $P_{k \times (n-k)}$ corresponde a uma matriz de paridade. Portanto, para o exemplo apresentado na Tabela 4.1, a matriz geradora é dada por

$$G = \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \bar{g}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Para a mensagem $\bar{u} = (101)$, a palavra código é dada por:

$$\bar{v} = \bar{u} \cdot G = \bar{u} \cdot \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \bar{g}_2 \end{bmatrix} = 1.(100110) + 0.(010011) + 1.(001101).$$

Portanto, $\bar{v} = 101011$.

4.1.4 Código Sistemático

Um código é dito **Sistemático** quando a paridade é adicionada à palavra código sem alterar a mensagem original. O código da Tabela 3-1 pode ser considerado como um código sistemático, pois mantém seus k *bits* de dados inalterado.

Esta característica é garantida porque a matriz geradora possui o termo I_k com uma matriz identidade.

4.1.5 Matriz Verificação de Paridade

Para cada matriz geradora existe uma outra matriz associada chamada **Matriz de Verificação de Paridade**. Esta matriz tem a função de verificar a integridade de uma palavra

código hipotética.

Para uma matriz \mathbf{G} de dimensão $k \times n$, com k linhas linearmente independentes, existe uma matriz \mathbf{H} de dimensão $(n - k) \times n$, com $n - k$ linhas linearmente independentes. As linhas de \mathbf{H} são geradas de tal modo que, qualquer vetor em uma linha de \mathbf{G} é ortogonal a uma linha de \mathbf{H} e, qualquer vetor que é ortogonal às linhas de \mathbf{H} é uma linha do espaço vetorial \mathbf{G} . Portanto, pode-se descrever o código de bloco linear (n, k) gerado por uma matriz \mathbf{G} da seguinte forma: Um vetor \mathbf{v} , é uma palavra código gerada por \mathbf{G} se e somente se, $\bar{\mathbf{v}} \cdot \mathbf{H}^T = 0$. As 2^{n-k} combinações de linhas da matriz \mathbf{H} formam um código linear $C_d(n, n - k)$. Este código é um espaço nulo do código linear \mathbf{C} gerado pela matriz \mathbf{G} . Ou seja, para qualquer $v \in C$ e qualquer $w \in C_d, v \times w = 0$, C_d é chamado código dual de \mathbf{C} . Portanto, a matriz de verificação de paridade para o código linear \mathbf{C} é a matriz geradora para o seu código dual C_d . Se a matriz geradora de um código linear (n, k) está na forma sistemática, a matriz de verificação de paridade é dada por

$$H = \left[P^T | I_{n-k} \right] = \left[\begin{array}{cccc|cccc} p_{0,0} & p_{1,0} & \dots & p_{k-1,0} & 1 & 0 & \dots & 0 \\ p_{0,1} & p_{1,1} & \dots & p_{k-1,1} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{0,n-k-1} & p_{1,n-k-1} & \dots & p_{k-1,n-k-1} & 0 & 0 & \dots & 1 \end{array} \right],$$

onde P^T é a transposta da submatriz de paridades.

4.1.6 Taxa de Código

A relação entre k e n ($R=k/n$) determina um parâmetro de desempenho do código, chamado **Taxa de Código**. Este parâmetro define a proporção de informação do código que corresponde à mensagem.

4.1.7 Distância e Peso de Hamming

A **Distância de Hamming** é um parâmetro definido entre duas palavras código que, em termos práticos mede a verossimilhança entre as palavras. Considerando dois vetores (\bar{v}_1 e \bar{v}_2) com uma mesma quantidade de símbolos binários, a distância de Hamming entre esses dois vetores é igual ao número de elementos que se diferem entre \bar{v}_1 e \bar{v}_2 . Portanto, a distância de Hamming entre os vetores \bar{v}_1 e \bar{v}_2 pode ser denotada por $d_h(\bar{v}_1 \text{ e } \bar{v}_2)$ e calculada conforme a

equação

$$d_h(\bar{v}_1, \bar{v}_2) = \sum_{i=0}^{n-1} \bar{v}_{1i} \oplus \bar{v}_{2i}, \quad (4.6)$$

onde n é o número de elementos dos vetores e \oplus representa uma operação ou-exclusivo.

Para um código C , a distância mínima de Hamming, que pode ser denotada por d_{min} , é definida como a menor distância entre quaisquer vetores distintos do código C .

O **Peso de Hamming** de um vetor \bar{v}_1 , denotado por $w(\bar{v}_1)$, corresponde ao número de *bits* diferentes de 0 deste vetor \bar{v}_1 . Por exemplo, considerando o vetor $\bar{v}_1 = (0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1)$, o peso de Hamming do vetor \bar{v}_1 é 3.

4.1.8 Capacidade de Detecção e Correção de Erros

Para as considerações referentes à capacidade de correção de um código linear são necessárias algumas definições, apresentadas a seguir.

Seja $\bar{v} = (v_1, v_2, \dots, v_n)$ o vetor código (palavra-código) transmitido por um canal, que utiliza um código corretor de erros aleatórios. Devido à influência do ruído presente no canal, o vetor recebido $\bar{r} = (r_1, r_2, \dots, r_n)$ pode ser qualquer uma das 2^n ênuplas. A diferença entre \bar{r} e \bar{v} é o chamado **Padrão de Erros** ou **Vetor de Erros**, que é causado pelo distúrbio no canal e representado pelo vetor

$$\begin{aligned} \bar{e} &= (e_1, e_2, \dots, e_n), \\ \bar{e} &= \bar{v} + \bar{r}. \end{aligned}$$

Quando $e_i = v_i + r_i = 1$, há um erro na posição i do vetor recebido \bar{r} . No receptor, a função do decodificador de canal é a de identificar o vetor código \bar{v} a partir do vetor recebido \bar{r} . Em um canal de transmissão, a decodificação de máxima verossimilhança (também denominada decodificação de máxima probabilidade, ou *maximum-likelihood decoding*) irá identificar \bar{v} como sendo o vetor código mais próximo do vetor recebido \bar{r} com relação à distância de Hamming, isto é, aquele para o qual $d(\bar{v}, \bar{r})$ é mínimo. Pode-se demonstrar que um código com distância mínima d_{min} pode corrigir t erros, sendo

$$t = \frac{d_{min} - 1}{2}. \quad (4.7)$$

4.1.9 Ganho de Codificação

O benefício obtido através da utilização do FEC em um sistema de comunicação pode ser medido através de um parâmetro chamado **Ganho de Codificação**. O ganho de codificação é definido como sendo a relação entre E_b/N_0 (Relação sinal ruído) do sinal não codificado pelo E_b/N_0 do sinal codificado, para uma **BER** (*Bit Error Rate*). Onde E_b e N_0 correspondem à energia de bit e a densidade espectral de ruído com codificação respectivamente. Além da relação E_b/N_0 , pode-se também medir a relação sinal ruído através do fator de qualidade Q ou a **OSNR** (*Optical Signal-to-Noise Ratio*). Soluções para FEC são tipicamente descritas em termos de **NEGC** (*Net Electrical Coding Gain*). Neste caso o desempenho do FEC é medido em função do melhoramento da relação **ESNR** (*Electrical Signal-to-Noise Ratio*) quando o receptor emprega o FEC [28]. Para calcular o valor em dB do ganho de codificação, pode-se utilizar a equação [29][30]

$$G = 10 \log \left(\frac{E_b}{N_0} \right)_{nc} - 10 \log \left(\frac{E_b}{N_0} \right)_c, \quad (4.8)$$

onde $(E_b/N_0)_{nc}$ é a relação entre a energia de *bit* e a densidade espectral de ruído sem codificação e $(E_b/N_0)_c$ é a relação entre a energia de *bit* e a densidade espectral de ruído com codificação. As curvas características de BER em função de E_b/N_0 , para um esquema de transmissão codificado e não codificado, são apresentadas na Figura 4.1 (Fonte: [1]).

Um cuidado deve ser tomado para a correta determinação de BER em função de E_b/N_0 com a codificação: o valor de E_b refere-se à energia por *bit* de informação, ou seja, considerando-se que a energia total gasta para a transmissão seja a mesma para os dois casos, então a energia por *bit* de informação com a codificação é menor do que a energia de *bit* sem a codificação, devido à inserção dos *bits* de redundância. Assim sendo, a relação entre $(E_b/N_0)_c$ e $(E_b/N_0)_{nc}$ fica afetada pela taxa de codificação de acordo com a relação

$$\left(\frac{E_b}{N_0} \right)_c = R \left(\frac{E_b}{N_0} \right)_{nc}. \quad (4.9)$$

Por esta razão o ganho de codificação de rede, ou *Net Coding Gain* (**NCG**), leva em vigor o aumento da taxa devido a inclusão do campo de paridade à mensagem. Isto significa, por exemplo, que no codificador FEC RS(255,239) existe um aumento de 7% da taxa de dados para transmitir a mensagem e a paridade.

Ainda com relação a Figura 4.1, os valores de taxa de erro e de E_b/N_0 apresentados referem-se

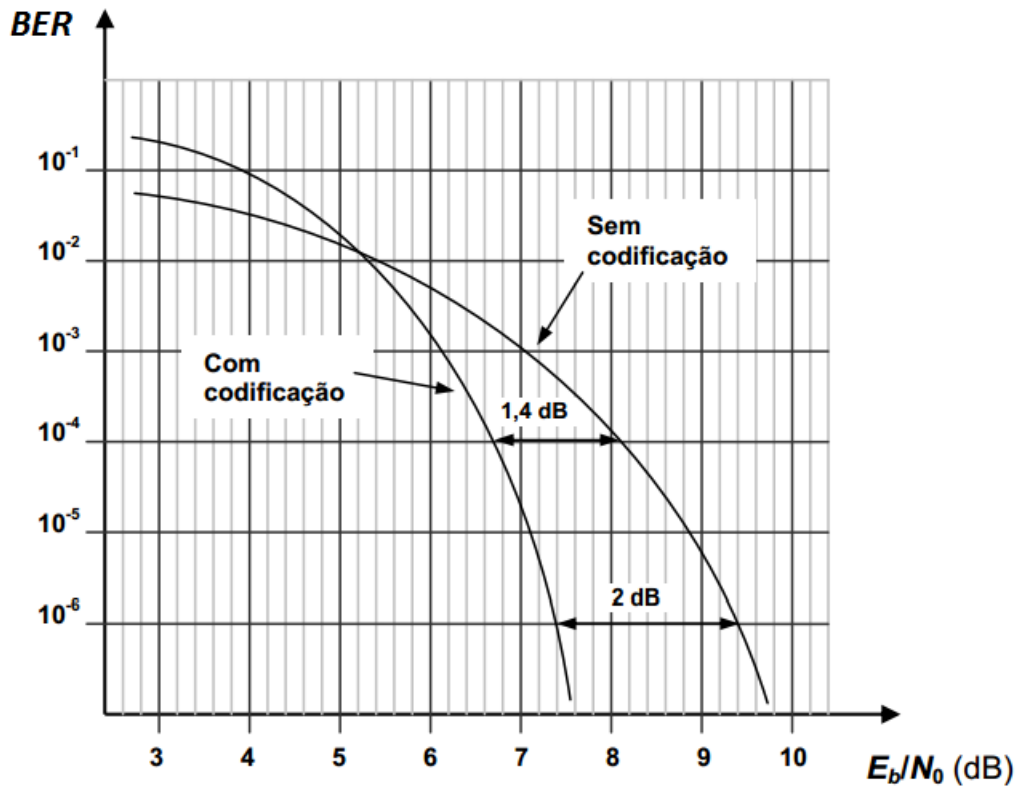


Figura 4.1: Curva Típica de Ganho de Codificação. Fonte: Codificação de Canal: Códigos de Bloco Lineares.[1]

a um esquema hipotético de correção de erros e permitem generalizar as seguintes conclusões:

- (1) Para valores reduzidos de E_b/N_0 a codificação não apresenta nenhum benefício, podendo o ganho de codificação ser até negativo, para o valor de E_b/N_0 determinado pela aproximação das duas curvas.
- (2) Para diferentes valores de BER obtém-se diferentes ganhos de codificação. Por exemplo, para $BER = 10^{-4}$ o ganho de codificação é igual o a 1,4 dB, enquanto para $BER = 10^{-6}$ o ganho sobe para 2 dB. Isso demonstra a dependência do ganho de codificação com BER.
- (3) A codificação permite obter redução de BER com a mesma energia (E_b/N_0 constante) em relação ao sinal sem codificação. Por exemplo, para $E_b/N_0 = 7$ dB, a BER cai de 10^{-3} para um pouco mais que 10^{-5} .
- (4) Considerando-se a expansão de largura de faixa provocada pela codificação, pode-se concluir ainda que a diminuição de BER e/ou E_b/N_0 decorrente da codificação produz uma expansão na largura de faixa.

- (5) Considerando-se um mesmo algoritmo de correção, a elevação da taxa de dados resulta em um ganho de codificação diferente para uma mesma BER.

4.2 Códigos Detectores de Erro

Para detectar os eventuais erros gerados durante a transmissão de dados, é verificado na recepção se algumas características da palavra código foram preservadas após a transmissão. Caso ocorra alguma violação, o erro é então detectado.

A detecção de erros pode ser realizada por diversos métodos. Este trabalho descreve três tipos de códigos detectores de erro como exemplo: Códigos de Paridade, Checksum e CRC.

4.2.1 Códigos de Paridade

O método da verificação de paridade é bastante simples, porém não é tão eficiente. A ideia é adicionar 1 *bit* antes ou depois da palavra a ser transmitida com o propósito de fazer com que o número de *bits* com valor 1 da palavra código tenha uma paridade pré-definida. Por exemplo, se fosse escolhido uma codificação com paridade par e a mensagem a ser transmitida fosse o *byte* $\bar{u} = (01100001)$, a palavra codificada seria $\bar{v} = (011000011)$. Deste modo, o número de *bits* com valor 1 da palavra código passa a ser par.

O problema para este tipo de código é que, se um número par de *bits* for corrompido, o receptor não será capaz de identificar o erro.

O código de paridade tem distância de Hamming de 2 *bits*. Portanto, $d = 2 - 1 = 1$, e somente a variação de 1 *bit* pode ser detectada.

4.2.2 Códigos de Checksum

Em Códigos de Checksum a ideia é transmitir uma sequência de símbolos (ou *bytes*) de mensagem e posteriormente adicionar um novo símbolo com o resultado do somatório dos *bytes* de informação. Na recepção a soma dos símbolos é recalculada e o resultado é comparado com o símbolo anexado à mensagem.

4.2.3 Códigos de Redundância Cíclica

Ao contrário dos algoritmos anteriores, o Código de Redundância Cíclica (CRC) possui uma complexidade razoável. O CRC é um código de bloco e portanto, a mensagem é dividida em blocos de k símbolos e é codificada em blocos de n símbolos. Neste caso, os símbolos podem ser representados por *bits*, ou *bytes*. Este código é implementado através de um divisor polinomial. Assim, o bloco de mensagem $M(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \dots + m_1x + m_0$, é dividido por um polinômio $G(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + g_{n-k-2}x^{n-k-2} + \dots + g_1x + g_0$, de ordem $n - k$, gerando um segundo polinômio $P(x) = p_{n-k-1}x^{n-k-1} + p_{n-k-2}x^{n-k-2} + \dots + p_1x + p_0$, correspondente ao resto da divisão. Para montar a palavra código, o polinômio de paridade é concatenado com o polinômio de mensagem e pode ser representado como $C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_1x + c_0$. Por exemplo, considerando um bloco de mensagem com 8 *bits* com o valor binário $\bar{u} = (10011101)$, os valores dos coeficientes do polinômio $M(x)$ de ordem 8 ($k=8$) são dados por $(1, 0, 0, 1, 1, 1, 0, 1)$. O polinômio mensagem está representado na equação $M(x) = x^7 + x^4 + x^3 + x^2 + 1$. O polinômio gerador possui ordem 4 pois $(n - k = 4)$ é dado por $G(x) = x^3 + x + 1$ será utilizado para cálculo da paridade. O cálculo da divisão pode ser ilustrado da seguinte forma

$$\begin{array}{r}
 \begin{array}{cccccc}
 & & & +x^4 & & +x^2 & +1 \\
 \hline
 x^3 + x + 1 &) & +x^7 & & +x^4 & +x^3 & +x^2 & +1 \\
 & & +x^7 & & +x^5 & +x^4 & & \\
 \hline
 & & & & +x^5 & & +x^3 & +x^2 & \\
 & & & & +x^5 & & +x^3 & +x^2 & \\
 \hline
 & & & & & & & & +1
 \end{array}
 \end{array}
 .$$

O resto da divisão pode ser representado pelo polinômio $P(x) = 1$. Conforme mencionado anteriormente, a palavra código pode ser obtida através da concatenação entre o polinômio de mensagem e a paridade. Portanto, a palavra código possui os valores $\bar{v} = (100111010001)$ e pode ser representado na forma polinomial

$$C(x) = x^{11} + x^8 + x^7 + x^6 + x^4 + 1. \quad (4.10)$$

Na recepção a palavra código é novamente dividida por $G(x)$ e, caso a mensagem tenha sofrido interferência do canal, o resto da divisão não será zero. Ademais, se o resto da divisão for zero, isto significa que não ocorreu erro na mensagem recebida.

4.3 Códigos Corretores de Erro

Os códigos corretores de erro são capazes de recuperar a mensagem original partir de um código com erros. Consistem na identificação de padrões que representam uma condição inválida do código, mas que mantenha o valor original da mensagem. Por exemplo, o parâmetro mais básico a ser identificado em uma palavra código é a distância de Hamming. Se uma palavra código que sofreu alteração estiver mais próxima do código válido do que de qualquer outro, a correção modifica os dados para este novo valor, corrigindo o erro.

4.3.1 Código Bose-Chaudhuri-Hocquenqhem

Código Bose-Chaudhuri-Hocquenqhem, ou simplesmente códigos BCH, é uma classe importante de código polinomial, linear, cíclico e sistemático, capazes de corrigir um número variável de *bits* de erros. Os códigos BCH mais comuns são classificados com os parâmetros a seguir. Para qualquer inteiro positivo $m \geq 3$ e $t \geq (2m - 1)/2$, existe um código binário com os seguintes parâmetros:

- Tamanho da Palavra Código : $n = 2^m - 1$.
- Tamanho do Bloco de Mensagem : $k \geq n - m \cdot t$.
- Distância de Hamming Mínima : $d_{min} \geq 2t + 1$.

Onde m é o número de *bits* de paridade e t é o número de erros que pode ser corrigido. O código BCH é capaz de corrigir até t *bits* errados por palavra código. O código BCH oferece uma flexibilidade na escolha dos parâmetros n (tamanho da palavra código) e k (tamanho do bloco de mensagem).

4.3.2 Código Hamming

O **Código de Hamming** recebeu este nome em homenagem ao seu criador, Richard Hamming. Este código representa uma primitiva e um esquema de correção de erros que é trivial, o

qual é chave para o entendimento de esquemas mais complexos. Aqui os códigos têm a distância de Hamming mínima, isto é, $d_{min} = 3$, portanto um erro pode ser corrigido e dois erros detectados. Para possibilitar a correção de erros, além da detecção de erros, a localização do erro deve também ser identificada. Assim, para a correção de 1 *bit* em um quadro de n *bits*, se existir um erro fora da posição dos n -*bits*, este deve ser identificado, caso contrário deve ser considerado que não existe erro. Uma vez localizado, a correção é trivial: o valor do *bit* é invertido.

O código Hamming(n, k) é um código de correção de erros linear que codifica k *bits* de dados para gerar uma palavra código de n *bits*. Em termos matemáticos, para qualquer inteiro positivo $m \geq 2$, existe um código binário com os seguintes parâmetros:

- Tamanho da Palavra Código : $n = 2^m - 1$.
- Tamanho do Bloco de Mensagem : $2^m - m - 1$.
- Taxa de Código : $\frac{k}{n} = 1 - \frac{m}{2^m - 1}$.

4.3.3 Código Reed-Solomon

O **Código Reed-Solomon** é amplamente conhecido, relativamente fácil de implementar e fornece uma boa eficiência na correção de erros em rajadas, tornando-se compatível com sistemas de transmissão de dados digitais.

O código Reed-Solomon pertence a família dos códigos de bloco, lineares, não binários, sistemáticos e cíclicos. Esses conceitos foram introduzidos na seção 3.1. Códigos não binários consistem de códigos com símbolos sendo representados por conjunto de *bits*. Por exemplo, para um código Reed-Solomon com n igual a 10, codificado em um campo $GF(2^8)$, isto significa que a palavra código possui 10 símbolos com 8 *bits* cada. A seguir, os símbolos do código Reed-Solomon serão referenciados também como *bytes*. Códigos RS são normalmente representados como RS(n, k), com símbolos de m *bits*. Para o código utilizado em redes de transporte óptico, RS(255,239), obtém-se:

- Tamanho da Palavra-Código : $n = 2^m - 1 = 255$.
- Tamanho do Bloco de Mensagem : $k = 2^m - 1 - 2t = 239$.
- Dígitos de Paridade : $n - k = 2t = 16$.

- Distância de Hamming Mínima : $2t + 1 = 17$.
- Números de Símbolos Corrigíveis : $t = (n - k)/2 = 8$.

A quantidade de símbolos que podem ser corrigidos, $(n-k)/2$, é representada pela variável t . Cada símbolo é um elemento e do campo $GF(2^m)$, deste modo, para quaisquer t símbolos da palavra código com erros, o decodificador será capaz de recuperar a mensagem original, ainda que os erros sejam em um ou em todos os *bits* destes *bytes*. Por exemplo, para o código RS(255, 239), no máximo 8 símbolos podem ser corrigidos, com erros em um *bit* ou em múltiplos *bits*.

4.3.3.1 Codificador Reed-Solomon

O processo de codificação Reed-Solomon consiste em converter um bloco de k símbolos de mensagem em uma palavra código com n símbolos. Para tal, considere o polinômio de mensagem $M(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \dots + m_1x + m_0$, a palavra código representada pelo polinômio $C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_1x + c_0$ e o polinômio gerador $G(x) = g_{n-k-1}x^{n-k-1} + g_{n-k-2}x^{n-k-2} + \dots + g_1x + g_0$. O polinômio gerador é formado pela multiplicação de $n-k$ polinômios de grau mínimo que tem 16 raízes com potências consecutivas de α , conforme descrito em

$$G(x) = \prod_{i=b}^{b+2t-1} (x + \alpha^i) = (x + \alpha^b)(x + \alpha^{b+1})(x + \alpha^{b+2}) \dots (x + \alpha^{b+2t-1}). \quad (4.11)$$

Se α é um elemento primitivo de $GF(2^m)$, o gerador polinomial $G(x)$ de um código Reed-Solomon com os símbolos pertencentes a $GF(2^m)$ tem como raízes os elementos $\alpha, \alpha^2, \alpha^3, \dots, \alpha^{2t}$, para b igual a 1. O coeficiente b no produto 4.11, define a faixa de raízes de $G(x)$. Como o código Reed-Solomon definido pela norma G.709, RS(255,239), é capaz de corrigir no máximo 8 símbolos da palavra código ($t=8$), o polinômio gerador fica como:

$$G(x) = x^{16} + 59x^{15} + 13x^{14} + 104x^{13} + 189x^{12} + 68x^{11} + 209x^{10} + 30x^9 + 8x^8 + 163x^7 + 65x^6 + 41x^5 + 226x^4 + 98x^3 + 50x^2 + 36x + 59. \quad (4.12)$$

Uma importante propriedade da palavra código é a de ser divisível pelo polinômio gerador. Considerando que o quociente e o resto da divisão entre uma palavra código e o polinômio

gerador são representados respectivamente pelos polinômios $Q(x)$ e $P(x)$, pode-se afirmar que a relação

$$C(x) = M(x)x^{n-k} + P(x) = Q(x)G(x) \quad (4.13)$$

é verdadeira. Os coeficientes da palavra código são representados também na forma polinomial em $C(x)$. O polinômio $C(x)$ de grau $n-1$ é pertencente a $GF(2^m)$ se e somente se $C(x)$ for divisível por $G(x)$.

Conforme mencionado anteriormente, para calcular o polinômio de paridade $P(x)$, o polinômio de mensagem $M(x)$ é deslocado $n - k$ posições e dividido por $G(x)$. O resto desta divisão é concatenado à mensagem para formar a palavra código. Adicionalmente, conforme pode ser observado em

$$C(x) = x^{n-k}M(x) + M(x) \mod G(x), \quad (4.14)$$

o polinômio $C(x)$ possui essencialmente dois termos. O primeiro corresponde ao polinômio mensagem, que é deslocado $n-k$ posições para esquerda e o segundo termo, corresponde ao resultado da paridade. Deste modo, a palavra código será um múltiplo inteiro do polinômio gerador.

4.3.3.2 Decodificador Reed-Solomon

Depois de transmitir uma palavra código em um canal de transmissão, os dados podem degradar-se. Deste modo, o polinômio que representa os símbolos da palavra código $C(x)$ modificados pelos erros $E(x)$ somados durante a transmissão, será referenciado a seguir como, $R(x) = C(x) + E(x)$. Este polinômio recebido pode ser escrito como, $R(x) = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + \dots + r_1x + r_0$ onde $E(x)$ é o polinômio correspondente aos símbolos de erro.

O processo de decodificação Reed-Solomon consiste em encontrar o polinômio $E(x)$. Para isto, é necessário que duas informações sejam extraídas do polinômio $R(x)$: a localização dos erros e suas correspondentes magnitudes. Estas localizações e magnitudes constituem o polinômio de erro $E(x)$.

Um processo típico de decodificação Reed-Solomon, compreende as seguintes etapas:

1. Síndrome: Cálculo do polinômio de síndrome, $S(x)$.

2. Berlekamp Massey: Cálculo dos coeficientes do polinômio localizador de erros $\Lambda(x)$, e cálculo dos coeficientes do polinômio avaliador de erros $\Omega(x)$.
3. Chien Search: Encontra a posição dos erros calculando as raízes do polinômio localizador de erros, $\Lambda(\alpha^i)$. Gera dois parâmetros intermediários para o cálculo do valor dos erros, $\Lambda'(\alpha^i)$ e $\Omega(\alpha^i)$.
4. Forney Algorithm: Encontra o valor dos coeficientes do polinômio de erro $E(x)$ a partir dos parâmetros intermediários, $\Lambda'(\alpha^i)$ e $\Omega(\alpha^i)$, calculados no bloco Chien Search.
5. Correção: Corrige os erros do polinômio $C(x)$ através do polinômio de erro $E(x)$.

A Figura 4.2 apresenta o diagrama em blocos do decodificador Reed-Solomon.

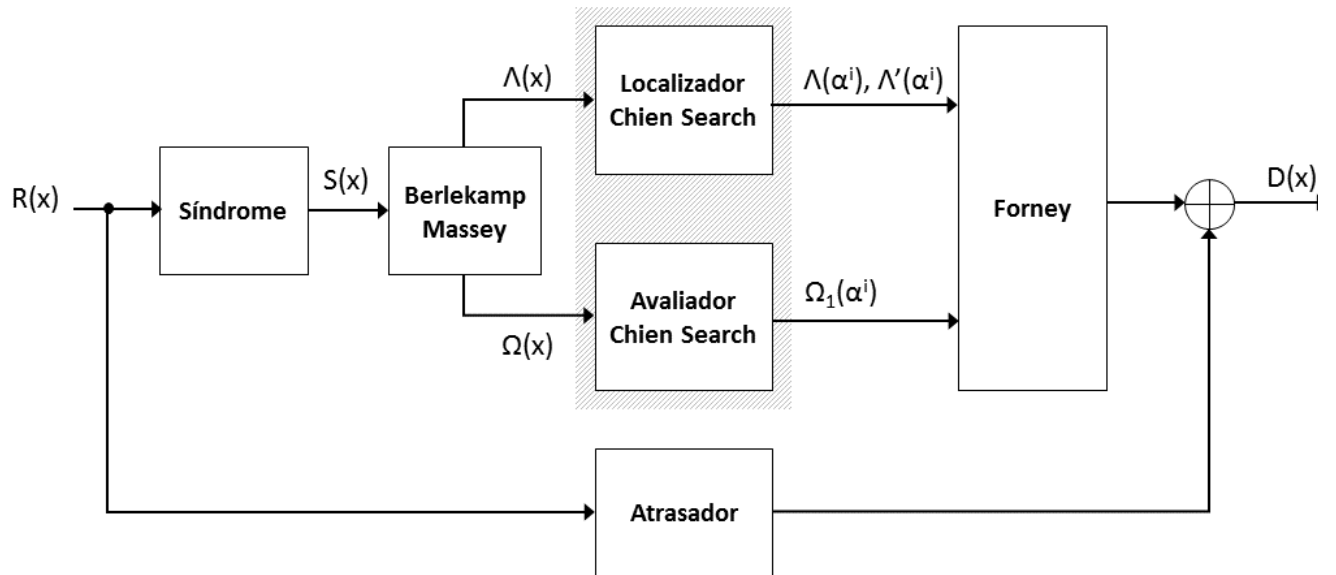


Figura 4.2: Diagrama em Blocos do Decodificador FEC Reed Solomon.

As subseções seguintes descrevem cada um dos blocos do decodificador RS.

Síndrome

Caso a palavra código tenha sido modificada por alguma interferência de ruído aditivo durante a transmissão, a mesma não será divisível pelo polinômio gerador $G(x)$. Esta verificação é chamada de síndrome e é realizada verificando se as 16 raízes consecutivas do polinômio gerador $G(x)$ 4.12, também são raízes do polinômio de dados recebidos $R(x)$.

O resultado da síndrome é representado com um polinômio do tipo

$$S(x) = s_{2t}x^{2t} + s_{2t-1}x^{2t-1} + \dots + s_2x + s_1, \quad (4.15)$$

onde cada um dos coeficientes do polinômio de síndrome são calculados da seguinte forma [4], $s_i = R(\alpha^i), i = 1, 2, 3, \dots, 2t$. Considerando que $R(x) = C(x) + E(x)$, os coeficientes de síndrome podem ser definidos como

$$s_i = C(\alpha^i) + E(\alpha^i), i = 1, 2, 3, \dots, n - 1. \quad (4.16)$$

Como $C(x)$ possui os coeficientes α^j como raízes, ou seja, $C(\alpha^j)$ é sempre 0, a equação de s_i pode ser simplificada e redefinida como

$$s_i = E(\alpha^i) = \sum_{j=1}^t e_j \alpha^{i_j}, i = 1, 2, 3, \dots, n - 1, \quad (4.17)$$

onde e_i corresponde ao valor do erro do j -ésimo erro encontrado na i -ésima posição da palavra código e α^{i_j} corresponde ao j -ésimo erro encontrado na i -ésima posição da palavra código.

O padrão de erro pode ser encontrado a partir do polinômio de síndrome em 4.15. Seja μ a quantidade de erros, $0 \leq \mu \leq t$, que estão nas posições i_1, i_2, \dots, i_μ . Deste modo, pode-se definir um novo polinômio de erros que pode ser escrito como

$$E(x) = e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \dots + e_{i_\mu}\alpha^{i_\mu}. \quad (4.18)$$

Definindo $\alpha^{i_j} = \beta_j$, a equação 4.18 pode ser utilizada para gerar $2t$ novas equações a partir dos coeficientes de síndrome diferentes de 0. A seguir são representadas as i , para $0 < i_j \leq 2t$, equações geradas a partir de 4.18, da seguinte forma

$$\begin{aligned} s_1 &= R(\alpha) = E(\alpha) = e_{i_1}\beta_1 + e_{i_2}\beta_2 + \dots + e_{i_\mu}\beta_\mu \\ s_2 &= R(\alpha^2) = E(\alpha^2) = e_{i_1}\beta_1^2 + e_{i_2}\beta_2^2 + \dots + e_{i_\mu}\beta_\mu^2 \\ &\vdots \\ s_{2t} &= R(\alpha^{2t}) = E(\alpha^{2t}) = e_{i_1}\beta_1^{2t} + e_{i_2}\beta_2^{2t} + \dots + e_{i_\mu}\beta_\mu^{2t}. \end{aligned} \quad (4.19)$$

Portanto, na decodificação existirão no máximo $2t$ equações com $2t$ incógnitas: t valores de erros e t localizações de erros. Entretanto, as $2t$ equações não podem ser resolvidas de modo usual pois são não-lienares (algumas incógnitas possuem expoentes). As técnicas que resolvem

este sistema de equações são conhecidas como algoritmos de decodificação Reed-Solomon.

A solução dessas equações é bastante complexa, exigindo que polinômios intermediários sejam computados. Assim, conforme já mencionado, dois novos polinômios são determinados antes que as posições e valores dos erros possam ser encontrados.

O polinômio localizador de erros é definido como

$$\Lambda(x) = \lambda_0 + \lambda_1 x + \lambda_2 x^2 + \dots + \lambda_\mu x^\mu, \quad (4.20)$$

e o polinômio avaliador de erros é definido como

$$\Omega(x) = 1 + \omega_1 x + \omega_2 x^2 + \dots + \omega_\mu x^\mu. \quad (4.21)$$

Existem dois métodos tipicamente utilizados para obter estes dois novos polinômios a partir do polinômio de síndrome:

- Berlekamp Massey
- Euclidiano

Ambos os métodos possuem uma complexidade similar. Entretanto, o algoritmo Berlekamp-Massey apresenta uma eficiência um pouco maior nas operações realizadas [31] e por esta razão foi a solução escolhida neste trabalho.

Algoritmo Berlekamp-Massey

Considerando a ocorrência de um erro em um símbolo de uma palavra código, o valor do erro não afeta a sua posição. Portanto, o polinômio localizador de erro pode ser definido como:

$$\Lambda(x) = (1 - \beta_1 x)(1 - \beta_2 x)(1 - \beta_3 x) \dots (1 - \beta_\mu x). \quad (4.22)$$

Portanto, as raízes do polinômio $\Lambda(x)$ são $\beta_1^{-1}, \beta_2^{-1}, \dots, \beta_\mu^{-1}$, correspondem ao inverso do número com a localização do erro. Utilizando as identidades de Newton, podemos expressar os coeficientes deste polinômio de acordo com as seguintes equações

$$\begin{aligned}
\lambda_0 &= 1 \\
\lambda_1 &= \beta_1 + \beta_2 + \dots + \beta_\mu \\
\lambda_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \dots + \beta_\mu\beta_{\mu-1} \\
&\vdots \\
\lambda_\mu &= \beta_1\beta_2\beta_3 \dots \beta_\mu.
\end{aligned}$$

É possível de se obter a relação entre os coeficientes de $\Lambda(x)$ e $S(x)$:

$$\begin{aligned}
s_1 + \lambda_1 &= 0 \\
s_2 + \lambda_1 s_1 &= 0 \\
s_3 + \lambda_1 s_2 + \lambda_2 s_1 &= 0 \\
&\vdots \\
s_\mu + \lambda_1 s_{\mu-1} + \lambda_2 s_{\mu-2} + \dots + \lambda_{\mu-1} s_2 + \lambda_\mu s_1 &= 0.
\end{aligned}$$

Relacionando os dois conjuntos de equações λ e s_i , obtém-se o seguinte resultado, que é conhecido como identidades de Newton,

$$\begin{aligned}
s_1 + \lambda_1 &= \underbrace{(\beta_1 + \beta_2 + \dots + \beta_\mu)}_{\text{sindrome 1}} + \underbrace{(\beta_1 + \beta_2 + \dots + \beta_\mu)}_{\text{lambda 1}} = 0 \\
s_2 + \lambda_1 s_1 &= \underbrace{(\beta_1)^2 + (\beta_2)^2 + \dots + (\beta_\mu)^2}_{\text{sindrome 2}} + \underbrace{(\beta_1 + \beta_2 + \dots + \beta_\mu)}_{\text{lambda 1}} \underbrace{(\beta_1 + \beta_2 + \dots + \beta_\mu)}_{\text{sindrome 1}} = 0 \\
&\vdots
\end{aligned}$$

O objetivo do algoritmo Berlekamp-Massey é encontrar o mínimo grau do polinômio $\Lambda(x)$ cujos coeficientes satisfazem estas identidades de Newton. O algoritmo procede como segue:

1. A primeira fase do algoritmo BM visa determinar o grau mínimo do polinômio $\Lambda_{BM}^{(1)}(x)$ que satisfaz a primeira identidade de Newton.
2. A segunda identidade de Newton é testada. Se o polinômio $\Lambda_{BM}^{(1)}(x)$ satisfaz a segunda identidade de Newton em (32), então $\Lambda_{BM}^{(2)}(x) = \Lambda_{BM}^{(1)}(x)$. Caso contrário, o algoritmo adiciona um termo de correção em $\Lambda_{BM}^{(1)}(x)$ para o polinômio $\Lambda_{BM}^{(2)}(x)$ seja capaz de satisfazer as duas primeiras identidades de Newton.
3. Na k-ésima iteração, o polinômio de grau mínimo será:

$$\Lambda_{BM}^{(k)}(x) = 1 + \lambda_1^{(k)} x + \lambda_2^{(k)} x^2 + \dots + \lambda_{l_k}^{(k)} x^{l_k}. \quad (4.23)$$

Onde l_k define a ordem do polinômio e $1 \leq l_k \leq k$, para os quais os coeficientes satisfaçam as seguintes identidades:

4. O próximo passo consiste em definir um novo polinômio de grau mínimo:

$$\begin{aligned} s_1 + \lambda_1^{(k+1)} &= 0 \\ s_2 + \lambda_1^{(k+1)} s_1 &= 0 \\ s_3 + \lambda_1^{(k+1)} s_2 + \lambda_2^{(k+1)} s_1 &= 0 \\ &\vdots \\ s_{l_{k+1}} + \lambda_1^{(k+1)} s_{l_k} + \lambda_2^{(k+1)} s_{l_{k-1}} + \dots + \lambda_{l_k}^{(k+1)} s_2 + \lambda_{l_{k+1}}^{(k+1)} s_1 &= 0. \end{aligned}$$

Uma vez que o algoritmo BM alcança o pass $2t$, o polinômio $\Lambda_{BM}^{(2t)}(x)$ é chamado como polinômio localizador de erro $\Lambda(x)$.

Para encontrar o polinômio avaliador de erro, uma vez encontrado o polinômio localizador de erros, utiliza-se uma equação que relaciona os polinômios $S(x)$, $\Lambda(x)$ e $\Omega(x)$. Esta equação é chamada de equação chave e é definida como

$$\Lambda(x)S(x) = \Omega(x) \mod x^{2t+1}. \quad (4.24)$$

Como os polinômios $S(x)$ e $\Lambda(x)$ já foram determinados através da primeira etapa do algoritmo Berlekamp Massey, a equação chave pode ser rearranjada na forma

$$\Omega(x) = \Lambda(x)S(x) \mod x^{2t+1}. \quad (4.25)$$

Algoritmo Chien Search

Depois de obter os dois polinômios de localização e avaliação dos erros, precisa-se determinar as raízes do polinômio localizador de erros, as quais representam o inverso da posição dos erros. Portanto, para todos os elementos de $GF(2^m)$ deve-se calcular o valor de $\lambda(\alpha^i)$, onde $i = 1, 2, 3, \dots, 2^{m-1}$.

Se algum elemento de GF se revelar raiz deste polinômio $\Lambda(\alpha^i) = 0$, um erro ocorreu na posição inversa de i , isto é, em $(n - i)$. Consequentemente, se esta condição não for satisfeita, significa que não existe erro na palavra código.

Algoritmo de Forney

O algoritmo Forney é usado para encontrar o valor dos erros. Para tal, o polinômio $\Omega(x)$ possibilita o cálculo do valor do erro através da equação

$$e_{j_i} = \frac{\Omega(\beta_l^{-i})}{\Lambda'(\beta^{-i})}. \quad (4.26)$$

Portanto, para um erro na posição α^i , a equação 4.26 fica

$$e_{j_i} = \frac{\Omega(\alpha_l^{-i})}{\Lambda'(\alpha^{-i})}. \quad (4.27)$$

O polinômio $\Lambda'(x)$ corresponde a derivada do polinômio $\Lambda(x)$ em um ponto x . A derivada de uma função polinomial $f(x)$ pertencente a $GF(2^m)$ é uma função polinomial $f'(x)$ com os seus coeficientes ímpares, conforme observado nas equações:

$$f(x) = f_1 + f_2x^2 + f_3x^3 + f_4x^4 + \dots + f_{n-1}x^{n-1} \quad (4.28)$$

e

$$f'(x) = f_1 + f_3x^3 + f_5x^5 + \dots + f_{n-1}x^{n-1}. \quad (4.29)$$

Arquitetura Paralela do Código RS(255,239) para 100 Gbit/s

Este Capítulo visa expor o assunto principal deste trabalho, a arquitetura paralelo do código corretor de erros Reed-Solomon RS(255,239). Para facilitar este entendimento, é apresentada a estrutura convencional de implementação do FEC serial e os passos para converter esta arquitetura para paralela.

Iniciamente a especificação matemática formal utilizada nas implementações seguintes foi exposta na seção 4.3.3.

A partir da definição teórica e matemática do FEC, o segundo passo do desenvolvimento é a implementação desses blocos, sendo que o resultado desta etapa é apresentado neste capítulo. A metodologia utilizada para abordar esta implementação baseia-se na divisão em quatro etapas, a saber:

1. Operações básicas em Campo Finito: São expostos os circuitos utilizados para implementar as operações básicas em Campo Finito: multiplicação, soma e inversão.
2. Estrutura serial: Para cada um dos blocos envolvidos no codificador e decodificador é apresentada a estrutura serial dos circuitos.
3. Passos para conversão da estrutura serial para paralela: Para cada um dos blocos envolvidos no codificador e decodificador são apresentadas as etapas de conversão da abordagem serial para a estrutura paralela dos circuitos.
4. Circuito FEC paralelo RS(255, 239) que atenda as recomendações ITU-T G.709: Para

cada um dos blocos envolvidos no codificador e decodificador é apresentado o circuito final para a abordagem paralela.

5.1 Operações Básicas em Campo Finito

Conforme descrito no Capítulo 3, corpos finitos são estruturas nas quais as operações de adição e multiplicação estão definidas. Portanto, como o escopo deste trabalho são códigos definidos em um corpo de extensão composto por m *bits* estendido do corpo finito binário, a implementação destas operações deve estar definida para tais elementos.

5.1.1 Soma e Subtração

A operação de adição e subtração em corpos finitos binários pode ser implementada em *hardware* utilizando portas ou-exclusivo (XOR), uma vez que nesta operação não existe o “vai um” como na soma de inteiros. Outro ponto importante de ser mencionado é que, como em corpos binários existem somente dois elementos possíveis, as operações de adição e subtração são equivalentes. O somador em Campo Finito está representado na Figura 5.1.

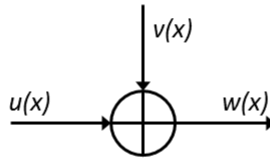


Figura 5.1: Somador em Campo Finito.

5.1.2 Multiplicação

A implementação do multiplicador entre dois elementos u e v em Campo Finito equivale multiplicar os dois valores inteiros e reduzir o resultado através da operação módulo p . Como em uma implementação física todos os dados são representados na forma binária, cada elemento de entrada do multiplicador pode ser tratado como um polinômio de grau máximo $m - 1$. Assim, o produto é representado na forma polinomial

$$w(x) = u(x)v(x) \mod p(x). \quad (5.1)$$

A ideia neste caso é multiplicar os elementos $u(x)$ e $v(x)$, dividir o resultado pelo polinômio $p(x)$ e atribuir para $w(x)$ o resto desta operação. Se a implementação em circuito seguisse esta mesma sequência de cálculos, o desempenho seria baixo e impossibilitaria e sua utilização em sistemas que operem em altas taxas. Neste caso, como o polinômio $p(x)$ é conhecido, é possível realizar a operação de multiplicação e divisão de forma paralela, aumentando assim o desempenho do bloco.

Dado que $u(x)$ e $v(x)$ são representados na forma

$$u(x) = u_{m-1}x^{m-1} + u_{m-2}x^{m-2} + \dots + u_1x + u_0 \quad (5.2)$$

e

$$v(x) = v_{m-1}x^{m-1} + v_{m-2}x^{m-2} + \dots + v_1x + v_0, \quad (5.3)$$

substituindo 5.2 e 5.3 em 5.1, tem-se

$$\begin{aligned} w(x) = & (d_0 + d_1x + d_2x^2 + \dots + d_{m-1}x^{m-1} + \\ & d_mx^m + d_{m+1}x^{m+1} + \dots + d_{2m-2}x^{2m-2}) \mod p(x) \\ w(x) = & d_0 + d_1x + d_2x^2 + \dots + d_{m-1}x^{m-1} + \\ & d_m[x^m \mod p(x)] + \dots + d_{2m-2}[x^{2m-2} \mod p(x)] \end{aligned} \quad (5.4)$$

onde

$$d_k = \begin{cases} \sum_{i=0}^{m-1} u_i v_{k-i} \\ \sum_{i=m}^{2m-2} u_{k-i+(m-1)} v_{i-(m-1)} \end{cases}.$$

Devido ao fato de que $p(x)$ é conhecido, os termos com grau acima de m podem ser pré-computados. Sendo assim, tem-se

$$r_i(x) = r_{i,0} + r_{i,1}x + \dots + r_{i,(m-1)}x^{m-1} = x_i \mod p(x), \quad (5.5)$$

onde $m \leq i \leq 2m - 1$. Deste modo, pode-se obter os coeficientes do polinômio $w(x)$, que representam o resultado do produto entre $u(x)$ e $v(x)$, na forma

$$w_j = d_j + \sum_{i=m}^{2m-2} d_i \cdot r_{i,j}, \quad (5.6)$$

onde $j = 0, 1, 2, \dots, m-1$. Esta arquitetura pode ser implementada utilizando somente funções lógicas XOR e AND.

O multiplicador sobre campos finitos é sempre representado conforme apresentado na Figura 5.2.

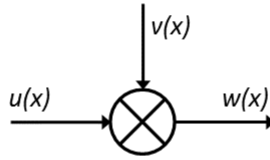


Figura 5.2: Multiplicador em Campo de Finito.

5.1.3 Divisão e Inversão

A operação de divisão é necessária no processo de decodificação, mais especificamente, na etapa de correção da palavra recebida. Porém, como a operação de divisão não é tão facilmente implementada, opta-se pela operação de inversão seguida por uma operação de multiplicação. A inversão de um elemento do campo tende a ser uma operação mais complexa de ser implementada do que a multiplicação. Atualmente existem diversos algoritmos que podem ser empregados nessa operação [32]. O artigo [33] propõe método baseado no algoritmo de Euclides estendido e é apresenta a seguir.

Como pode ser observado, o algoritmo de Euclides estendido realiza n iterações para o cálculo da inversão, onde n corresponde ao grau do polinômio gerador. Porém, porque o FEC que será apresentado posteriormente é destinado a aplicações com altas taxas de dados este método não é possível de ser aplicado.

A solução encontrada neste caso foi o emprego de uma tabela em memória no modelo CAM (*Content Addressable Memory*) [34], onde o elemento a ser invertido é utilizado como o endereço desta tabela, e o elemento invertido correspondente é o conteúdo deste endereço. Assim, a operação de inversão nada mais é do que uma tabela fixa em *hardware* composta por n elementos. A Figura 5.3 contém um diagrama representativo da implementação de um divisor polinomial.

Algorithm 1 Algoritmo de Euclides Estendido

```

procedure INVERSAO( $G(x), A(x)$ )
 $S(x) \leftarrow G(x); V(x) \leftarrow 0;$ 
 $R(x) \leftarrow A(x); U(x) \leftarrow 1;$ 
  while  $\deg(R(x)) \neq 0$  do
     $\delta \leftarrow \deg(R(x)) - \deg(S(x))$ 
    if  $\delta < 0$  then
       $temp \leftarrow S(x); S(x) \leftarrow R(x); R(x) \leftarrow temp;$ 
       $temp \leftarrow V(x); V(x) \leftarrow U(x); U(x) \leftarrow temp;$ 
       $\delta \leftarrow -\delta$ 
    end if
     $S(x) \leftarrow S(x) - x^\delta \times R(x)$ 
     $V(x) \leftarrow V(x) - x^\delta \times U(x)$ 
  end while
end procedure

```

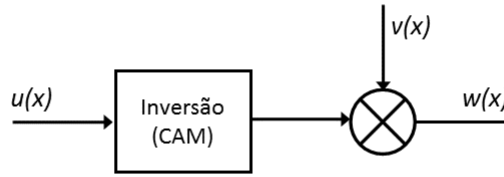


Figura 5.3: Divisor em Campo Finito.

5.2 Codificador Reed-Solomon

O Codificador FEC RS(255,239) tem a função de gerar uma palavra código $C(x)$ com 255 *bytes* a partir de um polinômio de mensagem $M(x)$ de 239 *bytes*. As próximas seções descrevem as estruturas serial e paralela para implementação do Codificador FEC RS(255,239).

5.2.1 Estrutura Serial do Codificador FEC RS(255,239)

Para implementar o Codificador RS(255,239) de forma serial, o circuito deve realizar uma operação módulo entre os polinômios de Mensagem $M(x)$ e $G(x)$, conforme já descrito na equação

$$C(x) = x^{n-k}M(X) + M(x) \mod G(x). \quad (5.7)$$

A abordagem utilizada neste projeto foi a de um circuito do tipo *Linear Feedback Shift Register* (LFSR). A Figura 5.4 apresenta o circuito convencional de codificador FEC Reed-Solomon [12]. Este circuito é implementado com registradores, multiplicadores por constantes em campo de

Galois, somadores em campo de Galois e dois multiplexadores. A mensagem $M(x)$ é fornecida para o codificador símbolo a símbolo, portanto a entrada de dados é um barramento de 8 *bits*.

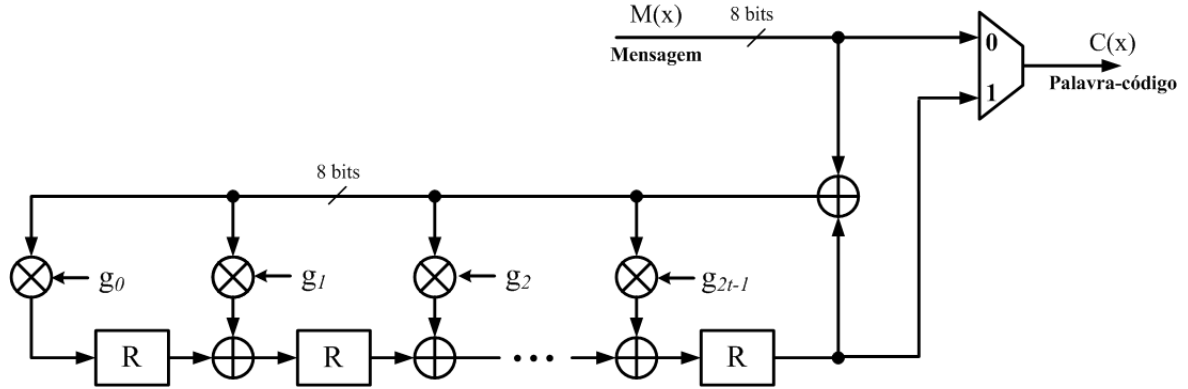


Figura 5.4: Codificador Reed-Solomon.

Os coeficientes $g_0, g_1, \dots, g_{2t-1}$ são obtidos diretamente do polinômio gerador $G(x)$, apresentado em 5.8. Portanto, no caso do codificador RS(255,239) os coeficientes g_n do codificador são $G = \{59, 36, 50, 98, 226, 41, 65, 163, 8, 30, 209, 68, 189, 104, 13, 59\}$ e podem ser representados através do polinômio

$$G(x) = x^{16} + 59x^{15} + 13x^{14} + 104x^{13} + 189x^{12} + 68x^{11} + 209x^{10} + 30x^9 + 8x^8 + 163x^7 + 65x^6 + 41x^5 + 226x^4 + 98x^3 + 50x^2 + 36x + 59. \quad (5.8)$$

O processo de codificação serial é realizado em dois estágios:

- (a) Na primeira etapa, o primeiro termo de 5.7 é implementado. Neste momento, os k símbolos de mensagem ($M(x)$) são transferidos para a saída $C(x)$. O multiplexador seleciona a sua entrada 0 para a saída.

Simultaneamente os *bytes* da mensagem são transferidos para a rede de realimentação para que a divisão polinomial seja processada.

- (b) Na segunda etapa, o segundo termo de 5.7 (Símbolos de paridade - $P(x)$) já está processada. O multiplexador seleciona a entrada 1 e os *bytes* de paridade presentes nos registradores são transmitidos para a saída $C(x)$ do codificador em $n - k$ ciclos de *clock*.

5.2.2 Estrutura Paralela do Codificador Reed-Solomon

Para descrever a estrutura paralela do FEC Reed-Solomon, considera-se o a estrutura genérica de processamento paralelo, exposto na Figura 5.5. Nesta figura são utilizados alguns parâmetros e elementos que estão definidos a seguir.

- (1) M : Número de símbolos paralelos que serão processados.
- (2) N : Número de símbolos da palavra código $C(x)$.
- (3) K : Número de símbolos da mensagem $M(x)$.
- (4) MP_i : Multiplicador paralelo de índice i . Na estrutura de processamento paralelo os multiplicadores de Campo em Galois observados na arquitetura serial são substituídos por multiplicadores paralelos em Campo de Galois. Cada MP possui M multiplicadores por constantes em Campo de Galois. Portanto, é necessário calcular quais são essas constantes a serem utilizadas em cada multiplicador. O algoritmo 2 fornece uma rotina para obter os coeficientes dos multiplicadores paralelos. Este algoritmo recebe como entrada o polinômio gerador do código $G(x)$, o valor de M e fornece como saída uma matriz $H_{2t \times M}$ com as constantes dos multiplicadores paralelos.

Algorithm 2 Gerador de Coeficientes de Multiplicador Paralelo

```

1: procedure COEFGEN( $G(x), M$ )
2:   for  $j \leftarrow 0, M - 1$  do
3:     for  $n \leftarrow 0, 2t - 1$  do
4:       if  $j = 0$  then
5:          $H_{i,j} \leftarrow g_i$ 
6:       else if  $n = 0$  then
7:          $H_{n,j} \leftarrow H_{15,j-1} \cdot H_{n,0}$ 
8:       else
9:          $H_{n,j} \leftarrow H_{15,j-1} \cdot H_{n,0} + H_{n-1,j-1}$ 
10:      end if
11:    end for
12:  end for
13: end procedure

```

- (5) CPP : Célula de processamento paralelo. Esta célula possui um conjunto de M registradores que recebem os resultados parciais dos multiplicadores paralelo, registram e transferem os resultados para somadores em Campo de Galois (CG). Estes somadores CG somam

os valores recebidos dos registradores com resultado calculado pelo próximo conjunto de multiplicadores paralelos. O número de CPP's a serem utilizados no codificador pode ser calculado dividindo $(N - K)/M$.

- (6) *CPF*: Célula de processamento final. A CPF é empregada sempre que o número de símbolos de paridade $(N - K)$ não for divisível por M . O número de CPF pode ser obtido através do cálculo $(N - K) \bmod M$.
- (7) *CRP*: Célula de roteamento parcial. A CRP é empregada sempre que o número de símbolos de paridade $(N - K)$ não for divisível por M . O número de CRF pode ser obtido através do cálculo $(N - K) \bmod M$.

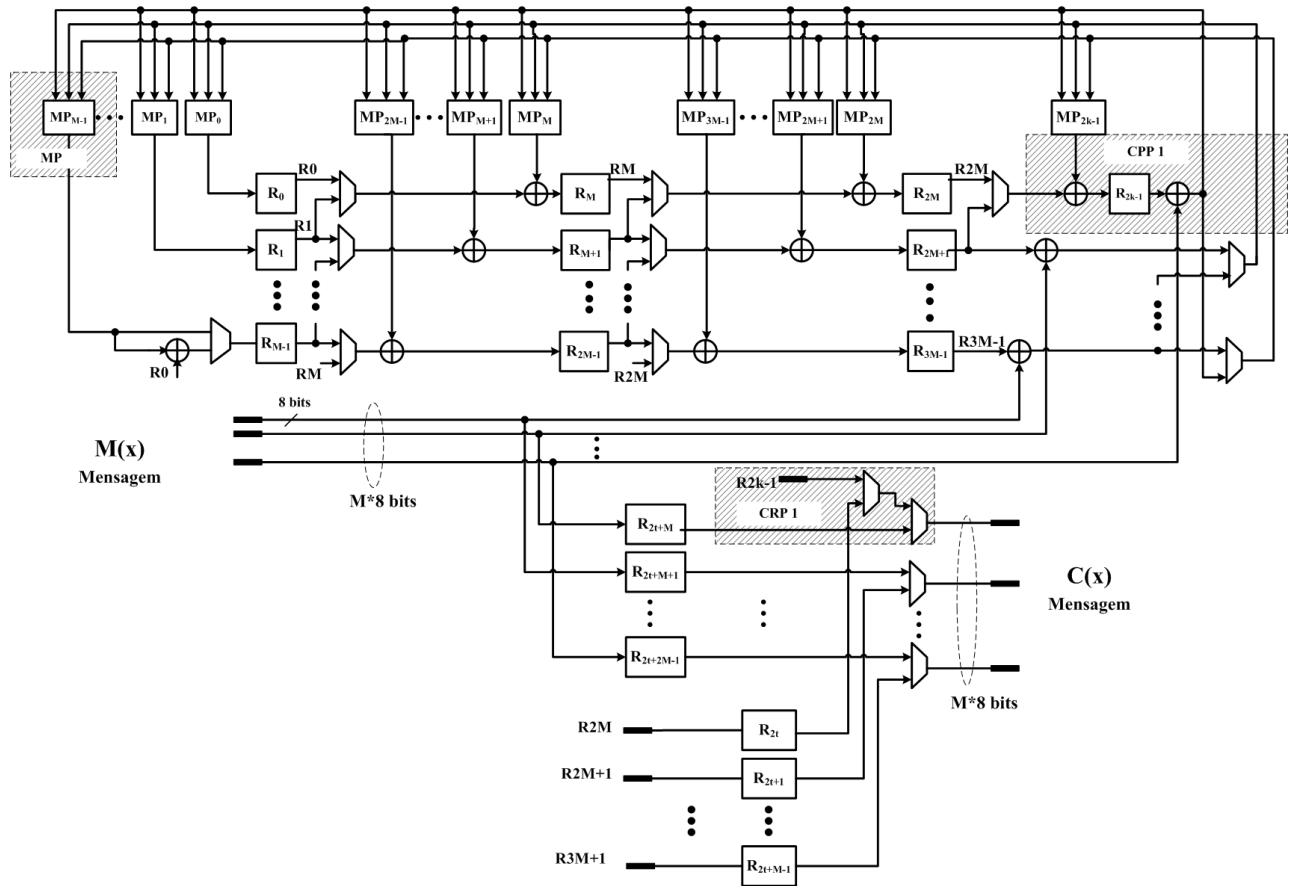


Figura 5.5: Codificador Paralelo RS(255,239).

Descrição do Codificador FEC RS(255,239) Paralelo (ITU-T G.709)

A seguir é apresentado o circuito do codificador FEC Reed-Solomon.

- (1) $M = 5$.
- (2) $N = 255$.
- (3) $K = 239$.
- (4) MP_i : A matriz $H_{2t \times M}$ é apresentada a seguir.

$$H = \begin{bmatrix} \overline{h_0} \\ \overline{h_1} \\ \overline{h_2} \\ \overline{h_3} \\ \overline{h_4} \\ \overline{h_5} \\ \overline{h_6} \\ \overline{h_7} \\ \overline{h_8} \\ \overline{h_9} \\ \overline{h_{10}} \\ \overline{h_{11}} \\ \overline{h_{12}} \\ \overline{h_{13}} \\ \overline{h_{14}} \\ \overline{h_{15}} \end{bmatrix} = \begin{bmatrix} 205 & 180 & 8 & 44 & 59 \\ 89 & 59 & 252 & 228 & 36 \\ 138 & 141 & 216 & 246 & 50 \\ 4 & 11 & 72 & 17 & 98 \\ 100 & 103 & 5 & 44 & 229 \\ 82 & 38 & 76 & 40 & 41 \\ 157 & 239 & 161 & 116 & 65 \\ 238 & 110 & 14 & 243 & 163 \\ 59 & 185 & 230 & 102 & 8 \\ 176 & 19 & 159 & 192 & 30 \\ 59 & 240 & 46 & 24 & 209 \\ 198 & 42 & 52 & 91 & 68 \\ 189 & 62 & 216 & 14 & 40 \\ 182 & 88 & 215 & 45 & 104 \\ 128 & 199 & 157 & 122 & 13 \\ 10 & 41 & 114 & 33 & 59 \end{bmatrix}$$

- (5) $CPP = (N - K)/M = 3$.
- (6) $CPF = (N - K) \bmod M = 1$.
- (7) $CRP = (N - K) \bmod M = 1$.

A Figura 5.6 apresenta o circuito para a especificação proposta.

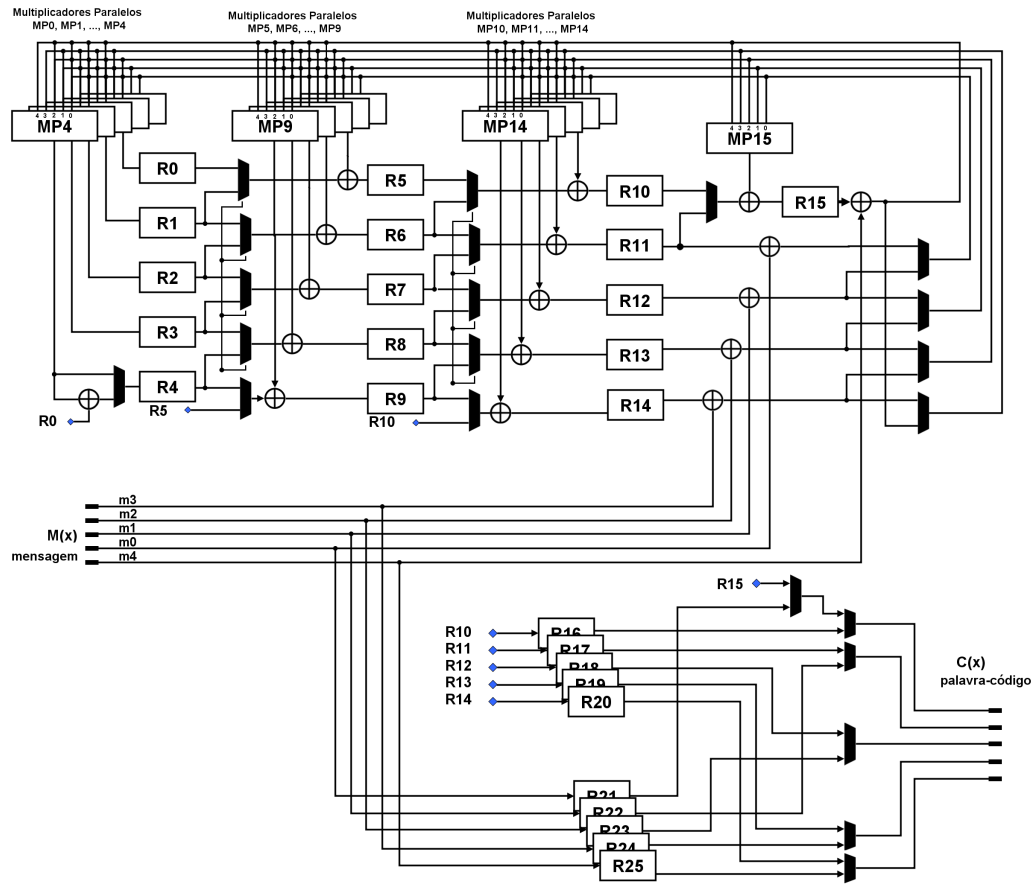


Figura 5.6: Codificador Paralelo RS(255,239) e $M=5$.

Cada palavra código no algoritmo RS(255,239) possui 255 *bytes*, que são gerados a partir do processamento de 239 *bytes* de mensagem. Portanto, são necessários 51 ciclos de relógio (255 *bytes* da palavra código / 5 *bytes* processados por período de *clock*).

Conforme exposto na Figura 5.6, a mensagem é recebida através de do barramento $M(x)$, o qual transfere 5 *bytes* a cada ciclo de relógio para o codificador Reed-Solomon. A redundância é calculada com uma estrutura de registradores de deslocamento com realimentação. Nesta arquitetura, os símbolos realimentados são inseridos em 16 conjuntos de multiplicadores paralelos em campo de Galois, MP_i , onde $i = 0, 1, 2, \dots, 15$. Cada sub-bloco MP multiplica 5 símbolos de mensagem por 5 constantes $H = (h_{i,0}, h_{i,1}, h_{i,2}, h_{i,3}, h_{i,4})$. Cada um dos 16 conjuntos de constantes foram calculados na etapa (4) da implementação.

Durante o processamento do campo de paridade, cada MP_n multiplica os *bytes* de mensagem recebidos por seu respectivo conjunto de constantes. O multiplicador paralelo MP_0 multiplica os símbolos recebidos através das entradas $m[0], m[1], m[2], m[3]$ e $m[4]$ pelas constantes definidas no vetor $(h_0) = (20518084459)$.

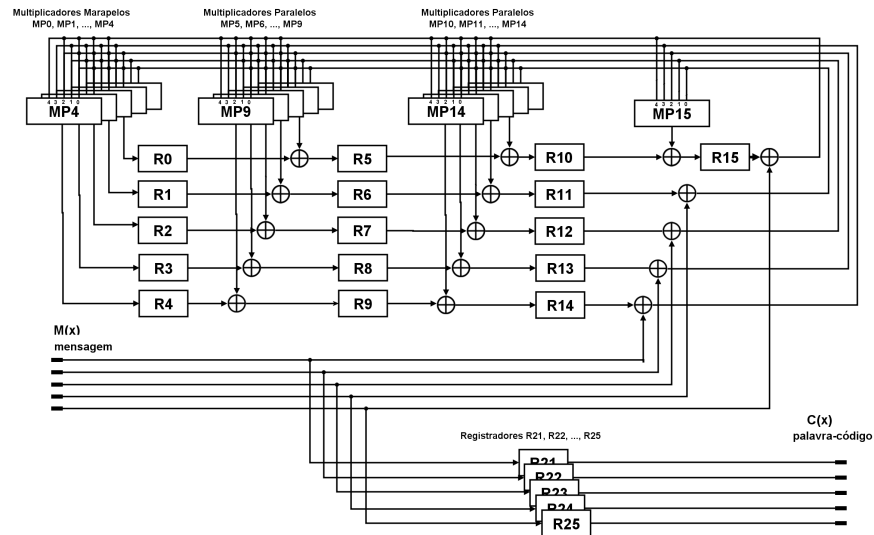
Todo o processo de decodificação é feito em três estágios, os quais são controlados pelo bloco de controle que define a configuração do multiplexadores.

Estágios de Codificação

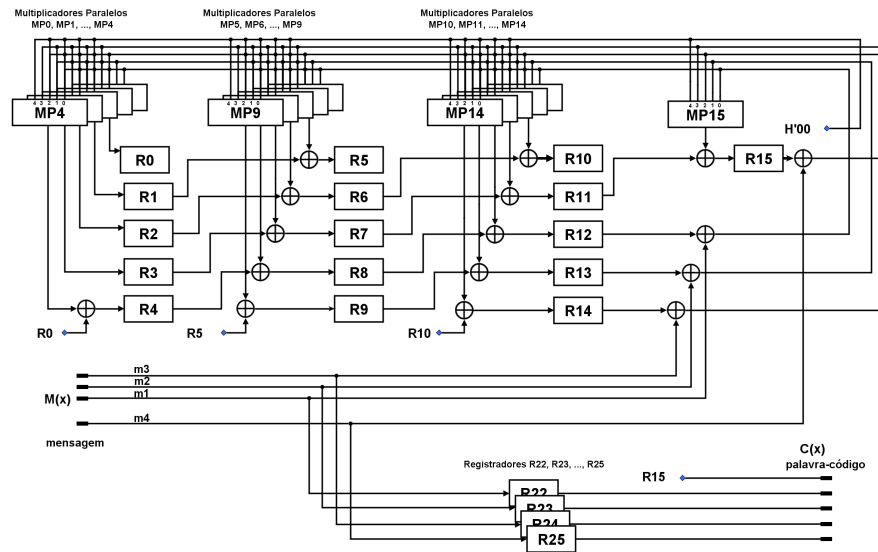
No primeiro estágio os primeiros 235 símbolos de mensagem ($m_{238}, m_{237}, m_{236}, \dots, m_5, m_4$) são transferidos para o codificador de forma paralela, ou seja, 5 *bytes* a cada ciclo de relógio. Para processar estes dados, os multiplexadores de processamento conectam os sinais recebidos dos registradores (R_n , onde $n = 0, 1, 2, \dots, 15$) para as suas respectivas saídas. Concomitantemente, os multiplexadores de saída conectam para as suas saídas os registradores (R_n , onde $n = 21, 22, \dots, 25$). O circuito simplificado do primeiro estágio de processamentos de paridade está apresentado na Figura 5.7(a).

No Segundo estágio de codificação, os últimos 4 símbolos de mensagem (m_3, m_2, m_1 e m_0) são transferidos para o codificador. Para processar estes dados, os multiplexadores de processamento conectam para as suas saídas os sinais recebidos dos registradores (R_n , onde $n = 1, 2, 3, \dots, 11$), respectivamente. Os multiplexadores de realimentação encaminham para as suas saídas os sinais dos registradores (R_n , onde $n = 12, 13, 14$ e 15) somados com os últimos 4 símbolos recebidos no barramento $M(x)$. O multiplexador da célula de processamento final transfere o sinal do somador GF para a sua saída. Os multiplexadores de saída enviam para o barramento de saída $C(x)$ os símbolos da entrada armazenados nos registradores (R_n , onde $n = 22, 23, 24$ e 25). Os multiplexadores da célula de roteamento final transferem para a saída o primeiro símbolo de paridade, que está disponível em R_{15} . O circuito simplificados do segundo estágio de processamentos de paridade está apresentado na Figura 5.7(b).

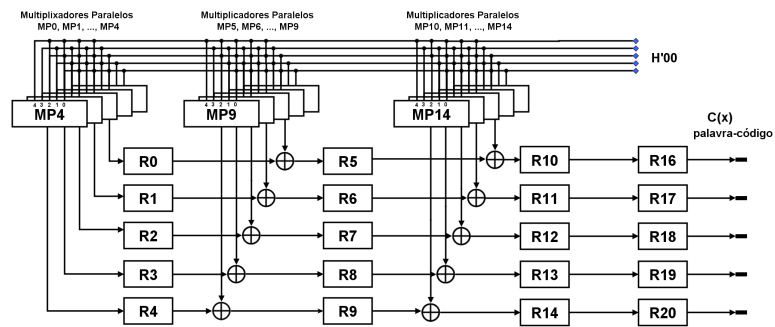
Finalmente, no último estágio os 15 símbolos de paridade que faltam para completar a palavra código estão disponíveis nos registradores (R_n , onde $n = 0, 1, 2, \dots, 14$). Para transferir estes *bytes* para a saída, os multiplexadores do divisor polinomial selecionam os dados dos registradores (R_n , onde $n = 0, 1, 2, \dots, 11$), respectivamente. Os multiplexadores que realimentam o divisor polinomial inserem o valor zero na entrada dos multiplexadores paralelos (PM_n , onde $n = 0, 1, 2, \dots, 14$). No próximos 3 ciclos de *clock*, os símbolos de paridade são registrados e transferidos para a saída $C(X)$. O circuito simplificados do terceiro estágio de processamentos de paridade está apresentado na Figura 5.7(c).



(a) Primeiro Estágio de Codificação



(b) Segundo Estágio de Codificação



(c) Terceiro Estágio de Codificação

Figura 5.7: Estágios de Codificação de Paridade.

5.3 Decodificador Reed-Solomon

Conforme Figura 4.2, o decodificador possui 5 blocos funcionais: Síndrome, algoritmo Berlekamp-Massey, Localizador Chien Search, Avaliador Chien Search e Algoritmo de Forney. A implementação serial e paralela de cada um desses blocos é apresentada nas subseções seguintes. A estrutura do decodificador paralelo é semelhante à serial e é apresentada na Figura 5.8.

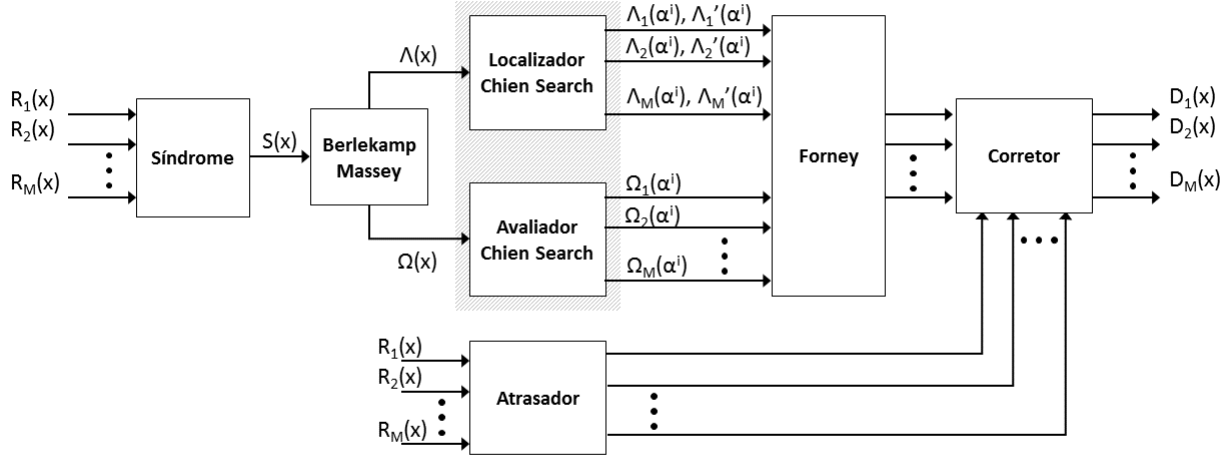


Figura 5.8: Diagrama em Blocos do Decodificador FEC Reed Solomon.

5.3.1 Síndrome

O circuito de síndrome tem a função de checar se as $2t$ raízes do polinômio gerador também são raízes do polinômio de dados recebidos $R(x)$, e é computada conforme descrito na equação

$$s_i = R(\alpha^i) = r_{n-1}(\alpha^i)^{n-1} + r_{n-2}(\alpha^i)^{n-2} + \dots + r_1(\alpha^i) + r_0. \quad (5.9)$$

Portanto, os coeficientes de síndrome são definidos por

$$\begin{aligned} s_0 &= R(\alpha^0) \\ s_1 &= R(\alpha^1) \\ &\vdots \\ s_{2t-1} &= R(\alpha^{2t-1}). \end{aligned} \quad (5.10)$$

Após o cálculo dos coeficientes de síndrome, obtém-se

$$S(x) = s_{2t-1}x^{2t-1} + s_{2t-2}x^{2t-2} + \dots + s_1x + s_0. \quad (5.11)$$

As estruturas serial e paralela do circuito de síndrome são apresentadas a seguir.

5.3.1.1 Arquitetura Serial do Processamento de Síndrome

Para facilitar o processamento da síndrome, primeiramente a equação 5.9 é modificada para a forma

$$s_i = (((\dots((r_{n-1}\alpha^i + r_{n-2})\alpha^i + r_{n-3})\alpha^i + \dots)\alpha^i + r_1)\alpha^i + r_0. \quad (5.12)$$

Assim, a equação 5.12 começa a ser determinada a partir do termo mais significativo da palavra recebida (r_{n-1}), e cada novo símbolo é usado, paralelamente, para efetuar a multiplicação por todas as $2t$ raízes do polinômio gerador. Em *hardware*, esta operação é realizada com uma porta ou-exclusivo (XOR) para cada *bit*, um multiplicador polinomial e um registrador onde, a cada ciclo de relógio, um novo símbolo recebido é somado em módulo 2 com o resultado do ciclo anterior. O circuito resultante segue o modelo constante na Figura 5.9. As saídas deste módulo consistem em cada um dos coeficientes de $S(x)$ em paralelo representados em 5.10.

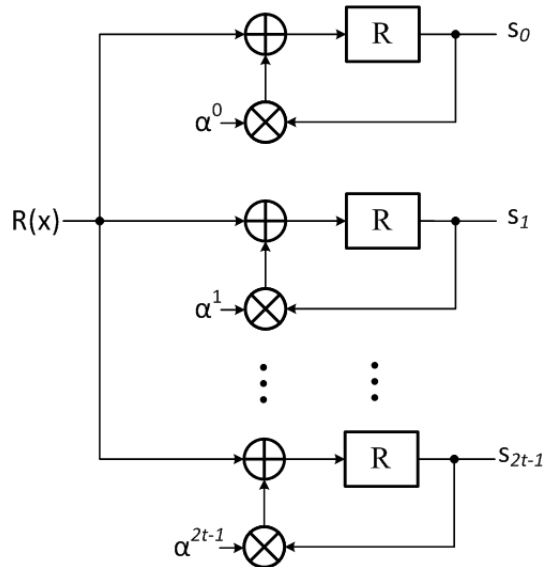


Figura 5.9: Processamento da Síndrome de Forma Serial.

Neste circuito de síndrome, cada coeficiente do polinômio $R(x)$ é processado de forma serial por uma célula de processamento serial de síndrome (CPSS). Portanto, são necessários n ciclos

de *clock* para que a palavra código seja processada e os $n - k$ coeficientes do polinômio de síndrome estejam prontos.

5.3.1.2 Arquitetura Paralela do Processamento de Síndrome

Para se descrever a estrutura paralela do circuito de síndrome, podem-se seguir os seguintes passos:

- (1) M : Definição do número de símbolos que serão processados paralelamente.
- (2) $R_i(x)$: Polinômios de recepção parcial. Para representar a recepção dos dados paralelos, o polinômio $R(x) = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + r_{n-3}x^{n-3} + \dots + r_1x + r_0$ é dividido em M polinômios, $R(x) = R_1(x) + R_2(x) + \dots + R_M(x)$, onde cada parcela $R_i(x)$ é definida por

$$\begin{aligned}
 R_1(x) &= \sum_{i=0}^{(n/M)-1} r_{n-1-(i \cdot M)} x^{n-1-(i \cdot M)} = r_{n-1}x^{n-1} + r_{n-1-M}x^{n-1-M} + \dots + r_{M-1}x^{M-1} \\
 R_2(x) &= \sum_{i=0}^{(n/M)-1} r_{n-2-(i \cdot M)} x^{n-2-(i \cdot M)} = r_{n-2}x^{n-2} + r_{n-2-M}x^{n-2-M} + \dots + r_{M-2}x^{M-2} \\
 &\vdots \\
 R_M(x) &= \sum_{i=0}^{(n/M)-1} r_{n-M-(i \cdot M)} x^{n-M-(i \cdot M)} = r_{n-M}x^{n-M} + r_{n-2M}x^{n-2M} + \dots + r_{M-M}.
 \end{aligned}$$

- (3) Modificação das Equações $R_i(x)$: Assim como na implementação serial, as equações parciais são modificadas para tornar a definição do *hardware*.
- (4) $CPPS_i$: Célula de Processamento Paralelo de Síndrome. Assim como a célula de processamento serial de síndrome (CPSS), este elemento é utilizado para verificar se a i -ésima raiz do polinômio gerador $G(x)$ também é raiz do polinômio $R(x)$. Cada célula CPPS checa uma única raiz de $R(x)$. Portanto são necessárias $2t$ células CPPS. O circuito empregado em cada CPPS, está representado na Figura 5.10. A sequência de processamento de uma célula de cálculo de síndrome paralela é semelhante ao serial, exceto no último ciclo de relógio. Durante os $n/M - 1$ primeiros ciclos, os símbolos recebidos na entrada são multiplicados por x^M ou α^{iM} . No último ciclo de processamento, o multiplexador chaveia para o multiplicador o coeficiente x^{M-i} , onde i corresponde ao índice do polinômio parcial de recepção

$R_i(x)$. Os M polinômios parciais são alterados para permitir a implementação dos circuitos de processamento de síndrome, da seguinte forma

$$\begin{aligned} R_1(x) &= (\dots ((r_{n-1}x^M + r_{n-2M-1})x^M + r_{n-3M-1})x^M + \dots + r_{M-1})x^{M-1} \\ R_2(x) &= (\dots ((r_{n-2}x^M + r_{n-2M-2})x^M + r_{n-3M-2})x^M + \dots + r_{M-2})x^{M-2} \\ &\vdots \\ R_M(x) &= (\dots ((r_{n-M}x^M + r_{n-2M-M})x^M + r_{n-3M-M})x^M + \dots + r_{M-M})x^{M-M}. \end{aligned}$$

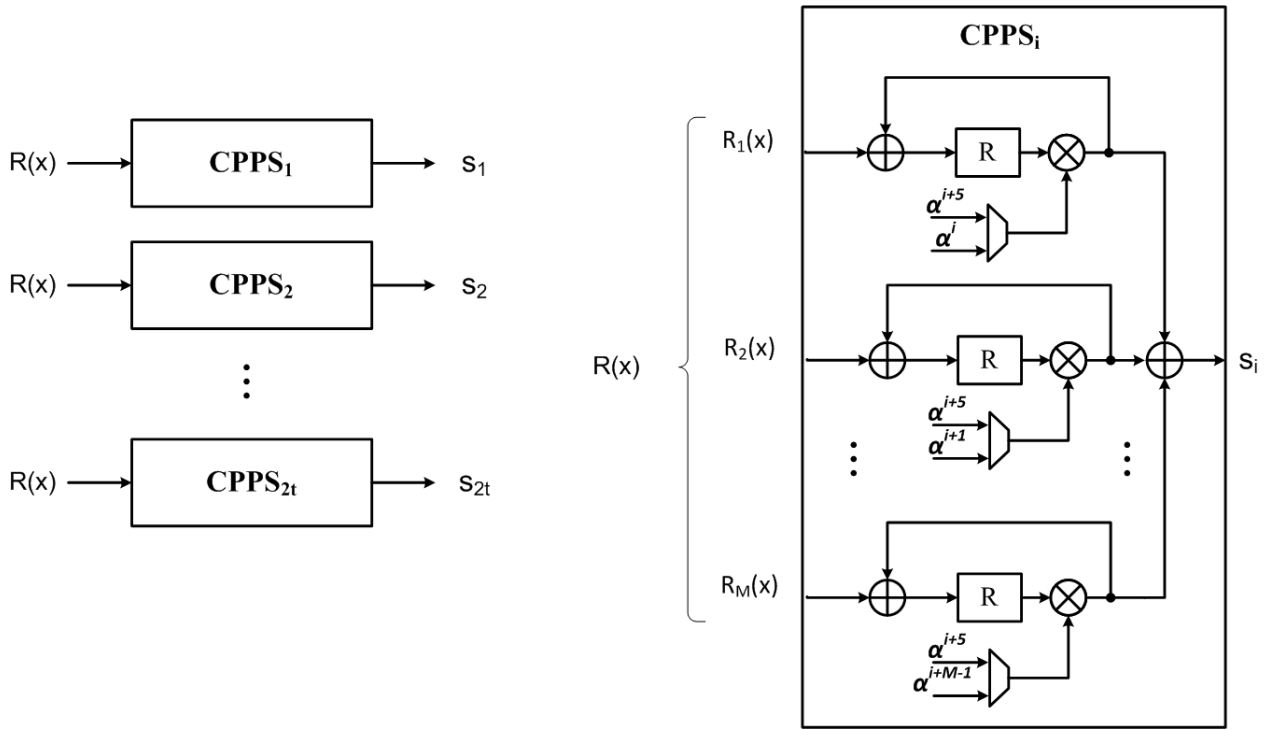


Figura 5.10: Processamento da Síndrome de Forma Paralela.

Descrição do Codificador FEC RS(255,239) Paralelo (ITU-T G.709)

Descrição do Circuito de Síndrome do Decodificador FEC RS(255,239) Paralelo (ITU-T G.709)

- (1) $M = 5$.
- (2) $R_i(x)$: Os polinômios parciais obtidos a partir de $R(x)$ ficam:

$$\begin{aligned}
R_1(x) &= r_{254}x^{254} + r_{249}x^{249} + r_{244}x^{244} + \dots + r_4x^4 \\
R_2(x) &= r_{253}x^{253} + r_{248}x^{248} + r_{243}x^{243} + \dots + r_3x^3 \\
R_3(x) &= r_{252}x^{252} + r_{247}x^{247} + r_{242}x^{242} + \dots + r_2x^2 \\
R_4(x) &= r_{251}x^{251} + r_{246}x^{246} + r_{241}x^{241} + \dots + r_1x \\
R_5(x) &= r_{250}x^{250} + r_{245}x^{245} + r_{240}x^{240} + \dots + r_0.
\end{aligned}$$

Para facilitar a elaboração de uma estrutura de hardware capaz de processar cada um dos polinômios $R_i(x)$, as equações são transformadas para

$$\begin{aligned}
R_1(x) &= ((((((r_{254}x^5 + r_{249})x^5 + r_{244})x^5 + r_{239})x^5 + \dots + r_9)x^5 + r_4)x^4 \\
R_2(x) &= ((((((r_{253}x^5 + r_{248})x^5 + r_{243})x^5 + r_{238})x^5 + \dots + r_8)x^5 + r_3)x^3 \\
R_3(x) &= ((((((r_{252}x^5 + r_{247})x^5 + r_{242})x^5 + r_{237})x^5 + \dots + r_7)x^5 + r_2)x^2 \\
R_4(x) &= ((((((r_{251}x^5 + r_{246})x^5 + r_{241})x^5 + r_{236})x^5 + \dots + r_6)x^5 + r_1)x \\
R_5(x) &= ((((((r_{250}x^5 + r_{245})x^5 + r_{240})x^5 + r_{235})x^5 + \dots + r_5)x^5 + r_0).
\end{aligned}$$

- (3) *CPPS*: São necessárias 16 células de processamento paralelo de síndrome, que podem ser construídas baseadas na estrutura da Figura 5.10.

5.3.2 Algoritmo Berlekamp Massey

5.3.2.1 Arquitetura para o Algoritmo Berlekamp Massey (BM)

O fluxograma da Figura 5.11 representa o algoritmo Berlekamp Massey, bem como o índice de cada etapa de sua implementação. Conforme apresentado na seção 4.3.3.2, o algoritmo Berlekamp Massey tem como fim o cálculo dos polinômios localizador de erros $\Lambda(x)$ e avaliador de erros $\Omega(x)$. O modo de implementação do algoritmo BM descrito neste trabalho segue uma versão proposta no trabalho [35], que elimina uma operação de inversão existente nas versões anteriores. A vantagem neste caso é a redução do comprimento de caminho crítico, aumentando assim o desempenho em frequência desta arquitetura. Este circuito pode ser utilizado tanto na

abordagem serial quanto paralela do decodificador. A implementação do algoritmo Berlekamp Massey é realizada em 9 etapas, conforme apresentado no fluxograma na Figura 5.11.

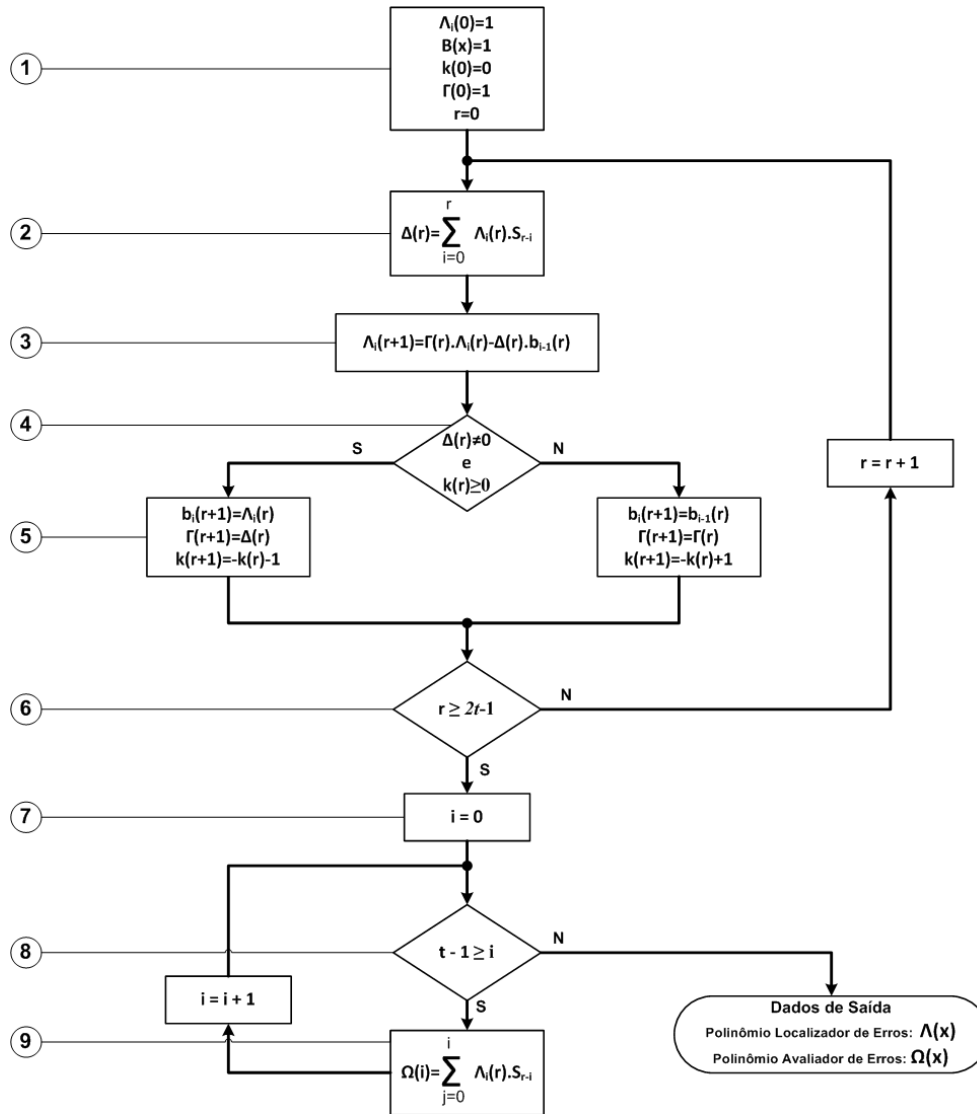


Figura 5.11: Fluxograma do Algoritmo Berlekamp-Massey.

(1) Etapa 1: Inicialização de variáveis.

Durante a execução dos cálculos precisam ser utilizadas diversas variáveis intermediárias. Esta etapa consiste em inicializar estas variáveis. A Tabela 5.1 apresenta uma descrição de cada variáveis, bem como os seus valores iniciais.

Tabela 5.1: Variáveis Utilizadas no Algoritmo Berlekamp Massey

Variável	Descrição	Valor Inicial
$\Lambda_i(x)$	Matriz de conexão que define o polinômio localizador em diferentes estágios do algoritmo, onde i é o índice do estágio.	$\Lambda_i(0) = 0$
$\Omega(x)$	Polinômio avaliador de erros	X
$B(x)$	Polinômio de conexão antes da última alteração.	1
$\Gamma(x)$	Discrepância antes da última alteração.	1
Δ	Discrepância da iteração atual.	X
l_k	Contador de iterações com discrepância igual a zero.	0
$S(x)$	Vetor de síndrome. É utilizado como dado de entrada no algoritmo para o cálculo do polinômio de localização de erros.	X

(2) Etapa 2: Polinômio de grau mínimo.

Nesta etapa o algoritmo entra em um *loop* destacados como 2, 3, 4, 5 e 6 na Figura 5.11. Este laço visa encontrar os coeficientes do polinômio localizador de erros passando por i estágios de $\Lambda(x)$, definindo l_k como o tamanho do polinômio $\Lambda(x)$ na k -ésima iteração. A equação que descreve estas iterações é definida por

$$\Lambda_{BM}^{(k)}(x) = 1 + \lambda_1^{(k)}x + \lambda_2^{(k)}x^2 + \dots + \lambda_{l_k}^{(k)}x^{l_k}. \quad (5.13)$$

A segunda fase do algoritmo BM visa determinar o grau mínimo do polinômio $\Lambda_{BM}^{(1)}(x)$ que satisfaz a primeira identidade de Newton em $S_1 + \Lambda_1 = 0$.

(3) Etapa 3: Testa a próxima identidade de Newton.

A segunda identidade de Newton é testada. Se o polinômio $\Lambda_{BM}^{(1)}(x)$ satisfaz a segunda identidade de Newton, então $\Lambda_{BM}^{(2)}(x) = \Lambda_{BM}^{(1)}(x)$. Caso contrário, o algoritmo adiciona um termo de correção Δ em $\Lambda_{BM}^{(1)}(x)$ para que o polinômio $\Lambda_{BM}^{(2)}(x)$ seja capaz de satisfazer as duas primeiras identidades de Newton $S_1 + \Lambda_1 = 0$ e $S_2 + \Lambda_1 S_1 = 0$.

(4) Etapa 4: Atualiza as variáveis do processamento.

Se a discrepância (Δ) for diferente de zero as variáveis são atualizadas com o termo de correção, caso contrário os valores do laço atual são mantidos.

(5) Etapa 5 : Atualiza as variáveis do processamento

Verifica se o algoritmo atingiu o número máximo de iterações possíveis. Caso $k \geq 2t - 1$ o processamento do polinômio localizador de erros está terminado. Caso contrário um novo laço de processamento é executado.

(6) Etapas 6, 7 e 8: Polinômio Avaliador de Erros

Nas etapas 6, 7 e 8 o polinômio localizador de erros e o polinômio de síndrome são relacionados utilizando a equação chave, dada por

$$\Lambda(x)S(x) = \Omega(x) \mod x^{2t+1}, \quad (5.14)$$

para se determinar o polinômio avaliador de erros $\Omega(x)$.

5.3.3 Algoritmo Chien Search

O Chien Search é utilizado para calcular as raízes do polinômio de localização e avaliação de erros. Estes cálculos são implementados através de duas estruturas chamadas de *Chien Search Locator* (CSL) e *Chien Search Evaluator* (CSE).

O circuito CSL calcula o resultado de

$$\Lambda(\alpha^{-i}) = 1 + \lambda_0 + \lambda_1(\alpha^{-i})^1 + \lambda_2(\alpha^{-i})^2 + \dots + \lambda_\mu(\alpha^{-i})^\mu \quad (5.15)$$

e o circuito CSE o resultado de

$$\Omega(\alpha^{-i}) = \omega_0 + \omega_1(\alpha^{-i})^1 + \omega_2(\alpha^{-i})^2 + \dots + \omega_\mu(\alpha^{-i})^\mu. \quad (5.16)$$

As seções seguintes apresentam os circuitos seriais paralelo de CSL e CSE.

5.3.3.1 Arquitetura Serial do *Chien Search Locator*

Na arquitetura serial do *Chien Search Locator*, após receber um novo polinômio localizador do algoritmo Berlekamp Massey, uma raiz é verificada a cada período de relógio.

Pode-se representar os coeficientes GF processados a cada ciclo de *clock* através da equação, $\alpha^{(k-1)}$, onde k representa o índice do ciclo de relógio de processamento.

O circuito de processamento serial das raízes do polinômio localizador está exposto na Figura 5.15. Pode-se observar que o circuito possui uma estrutura regular com $\mu + 1$ células que calculam valores intermediários de cada raiz. Inicialmente os registradores recebem os coeficientes do polinômio localizador de erros, produzindo como resultado o valor de $\Lambda(\alpha^0)$. Este valor é acumulado para que a próxima raiz $\Lambda(\alpha^1)$ seja calculada e assim sucessivamente.

Conforme apresentado na Seção 4.3.3.2, a derivada do polinômio localizador é utilizado para calcular o valor de um possível erro encontrado. Este resultado pode ser obtido somando os resultados dos coeficientes ímpares, conforme representado na Figura 5.15 na saída $\Lambda'(\alpha^{k-1})$.

Tabela 5.2: Sequência de Verificação das Raízes do Polinômio Localizador Processados Serialmente

Ciclo k	Coefficiente Verificado	Verificação da Raíz
1 ^o	α^0	$\Lambda(\alpha^0) = 1 + \lambda_0 + \lambda_1(\alpha^0)^1 + \lambda_2(\alpha^0)^2 + \dots + \lambda_\mu(\alpha^0)^\mu$
2 ^o	α^1	$\Lambda(\alpha^1) = 1 + \lambda_0 + \lambda_1(\alpha^1)^1 + \lambda_2(\alpha^1)^2 + \dots + \lambda_\mu(\alpha^1)^\mu$
3 ^o	α^2	$\Lambda(\alpha^2) = 1 + \lambda_0 + \lambda_1(\alpha^2)^1 + \lambda_2(\alpha^2)^2 + \dots + \lambda_\mu(\alpha^2)^\mu$
\vdots	\vdots	\vdots
n ^o	α^{n-1}	$\Lambda(\alpha^{n-1}) = 1 + \lambda_0 + \lambda_1(\alpha^{n-1})^1 + \lambda_2(\alpha^{n-1})^2 + \dots + \lambda_\mu(\alpha^{n-1})^\mu$

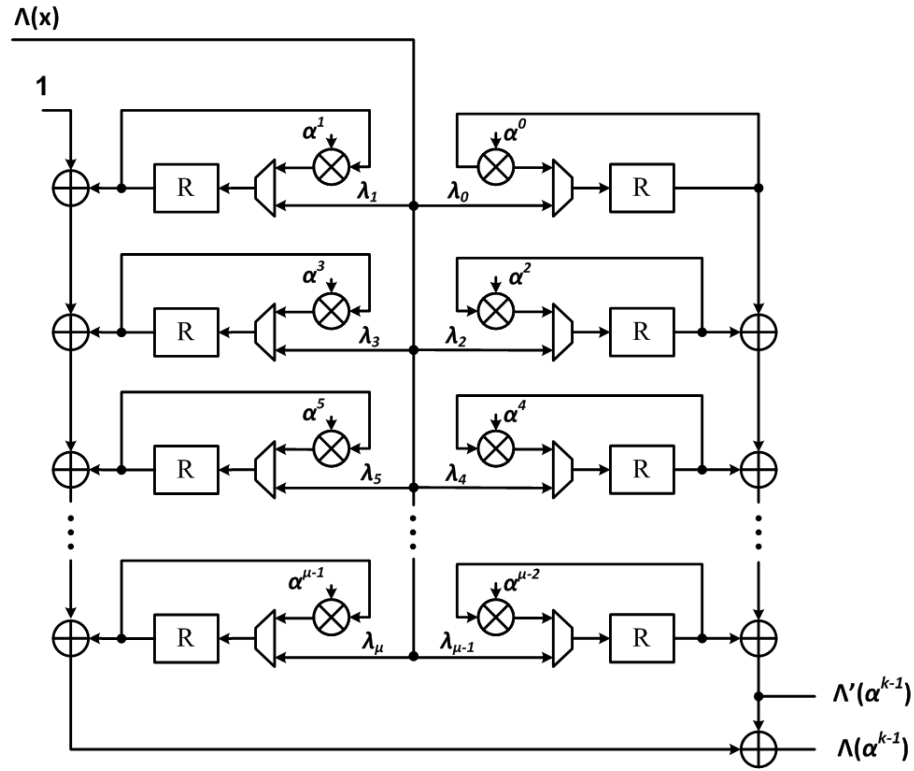


Figura 5.12: Arquitetura Serial do Chien Search Locator.

5.3.3.2 Arquitetura Paralela do *Chien Search Locator*

Para se implementar uma arquitetura paralela do Chien Search Locator (CSL), seguem-se os seguintes passos.

- (1) M: Define-se o número de símbolos paralelos a serem processados.
- (2) $CSLP_i$: Célula de processamento paralelo *Chien Search Locator*. Nesta etapa são geradas M células de processamento. No processamento paralelo o polinômio localizador $\Lambda(x)$ de erros é dividido em M polinômios representados por $\Lambda(x) = \Lambda_1(x) + \Lambda_2(x) + \dots + \Lambda_M(x)$, onde cada parcela $\Lambda_i(x)$ verifica uma raiz diferente, possibilitando que o circuito processe M raízes paralelamente. As equações seguintes descrevem quais as raízes que cada polinômio parcial $\Lambda_i(x)$ verifica:

$$\begin{aligned}
\Lambda_1(x) &= \Lambda(\alpha^{0+5i}) \\
\Lambda_2(x) &= \Lambda(\alpha^{1+5i}) \\
&\vdots \\
\Lambda_M(x) &= \Lambda(\alpha^{M-1+5i}),
\end{aligned}$$

onde a variável i corresponde ao ciclo de *clock* de processamento das raízes, que varia entre $i = 1, 2, 3, \dots, n/M$.

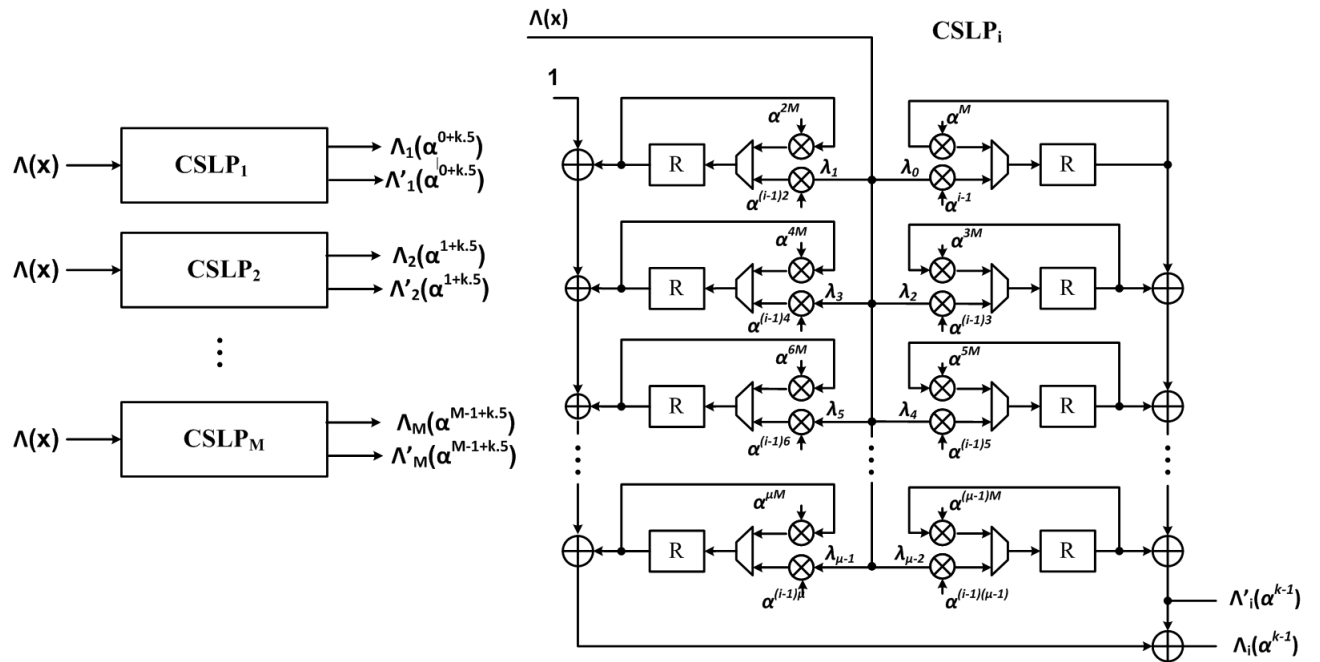


Figura 5.13: Arquitetura Paralela do Chien Search Locator.

Esta sequência de verificação é apresentada na Tabela 5.3.

Tabela 5.3: Sequência de Verificação das Raízes do Polinômio Localizador Processados Paralelamente (5 Coeficientes a Cada Ciclo de Clock)

Ciclo k	Coeficiente Verificado $i = 0, 1, 2, \dots, M - 1$	Verificação da Raíz
1 ^o	$\alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{M-1}$	$\Lambda_1(\alpha^0) = 1 + \lambda_0 + \lambda_1(\alpha^0)^1 + \dots + \lambda_\mu(\alpha^0)^\mu$ $\Lambda_2(\alpha^1) = 1 + \lambda_0 + \lambda_1(\alpha^1)^1 + \dots + \lambda_\mu(\alpha^1)^\mu$ \vdots $\Lambda_M(\alpha^{M-1}) = 1 + \lambda_0 + \lambda_1(\alpha^{M-1})^1 + \dots + \lambda_\mu(\alpha^{M-1})^\mu$
2 ^o	$\alpha^M, \alpha^{M+1}, \dots, \alpha^{2M-1}$	$\Lambda_1(\alpha^M) = 1 + \lambda_0 + \lambda_1(\alpha^M)^1 + \dots + \lambda_\mu(\alpha^M)^\mu$ $\Lambda_2(\alpha^{M+1}) = 1 + \lambda_0 + \lambda_1(\alpha^{M+1})^1 + \dots + \lambda_\mu(\alpha^{M+1})^\mu$ \vdots $\Lambda_M(\alpha^{2M-1}) = 1 + \lambda_0 + \lambda_1(\alpha^{2M-1})^1 + \dots + \lambda_\mu(\alpha^{2M-1})^\mu$
3 ^o	$\alpha^{2M}, \alpha^{2M+1}, \dots, \alpha^{3M-1}$	$\Lambda_1(\alpha^{2M}) = 1 + \lambda_0 + \lambda_1(\alpha^{2M})^1 + \dots + \lambda_\mu(\alpha^{2M})^\mu$ $\Lambda_2(\alpha^{2M+1}) = 1 + \lambda_0 + \lambda_1(\alpha^{2M+1})^1 + \dots + \lambda_\mu(\alpha^{2M+1})^\mu$ \vdots $\Lambda_M(\alpha^{3M-1}) = 1 + \lambda_0 + \lambda_1(\alpha^{3M-1})^1 + \dots + \lambda_\mu(\alpha^{3M-1})^\mu$
\vdots	\vdots	\vdots
$(n/M)^o$	$\alpha^{n-M}, \alpha^{n-M+1}, \dots, \alpha^{n-1}$	$\Lambda_1(\alpha^{n-M}) = 1 + \lambda_0 + \lambda_1(\alpha^{n-M})^1 + \dots + \lambda_\mu(\alpha^{n-M})^\mu$ $\Lambda_2(\alpha^{n-M+1}) = 1 + \lambda_0 + \lambda_1(\alpha^{n-M+1})^1 + \dots + \lambda_\mu(\alpha^{n-M+1})^\mu$ \vdots $\Lambda_M(\alpha^{n-1}) = 1 + \lambda_0 + \lambda_1(\alpha^{n-1})^1 + \dots + \lambda_\mu(\alpha^{n-1})^\mu$

Descrição do Circuito de *Chien Search Locator* do Decodificador FEC RS(255,239) Paralelo (ITU-T G.709)

(1) $M = 5$.

(2) *CSLP*: Neste exemplo são utilizadas 5 células de processamento *Chien Search Locator* paralela. Cada célula processa uma parcela do polinômio, possibilitando que 5 raízes sejam

verificadas por ciclo de *clock*. Cada parcela $\Lambda_i(x)$ é representada pelas equações

$$\Lambda_1(x) = \Lambda(\alpha^{0+5i})$$

$$\Lambda_2(x) = \Lambda(\alpha^{1+5i})$$

$$\Lambda_3(x) = \Lambda(\alpha^{2+5i})$$

$$\Lambda_4(x) = \Lambda(\alpha^{3+5i})$$

$$\Lambda_5(x) = \Lambda(\alpha^{4+5i}).$$

A Figura 5.17 apresenta o circuito utilizado para processar cada resultado paralelo das 5 raízes do polinômio $\Lambda(x)$.

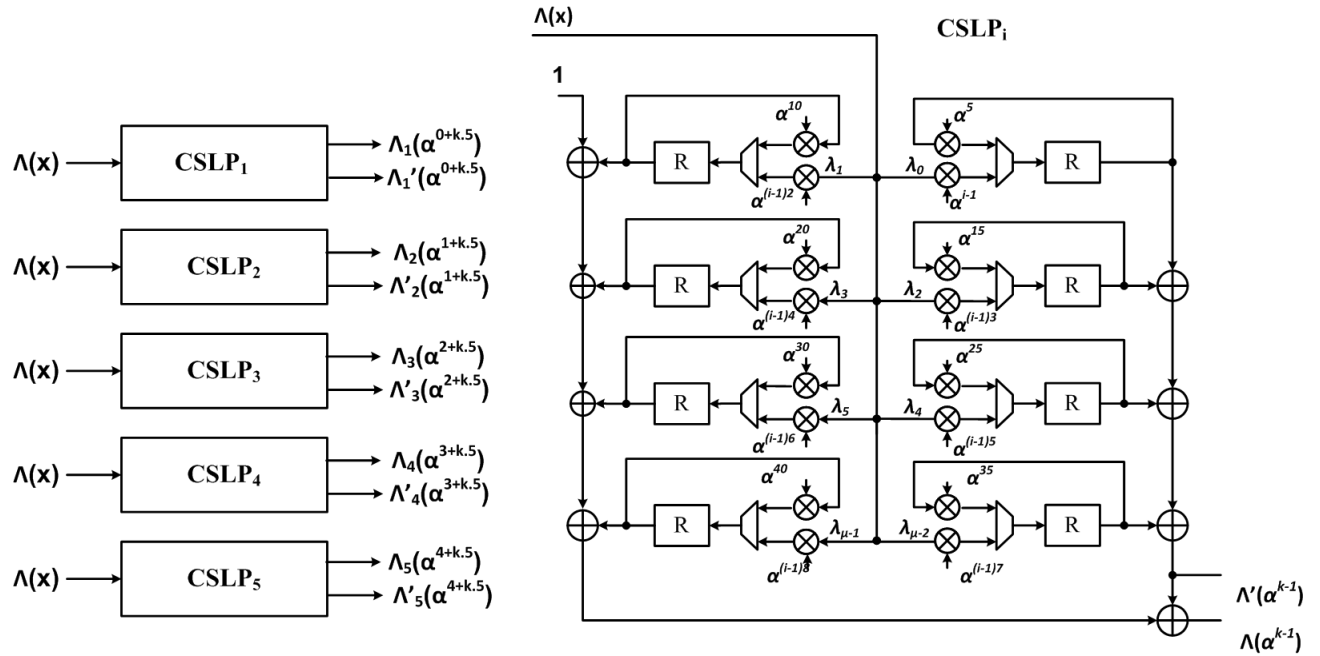


Figura 5.14: Arquitetura Paralela Exemplo do Chien Search Locator.

5.3.3.3 Arquitetura Serial do Chien Search Evaluator

Na arquitetura serial do *Chien Search Evaluator*, após receber um novo polinômio avaliador do algoritmo Berlekamp Massey, uma raiz é verificada a cada período de *clock*.

Pode-se representar os coeficientes GF processados a cada ciclo de *clock* através do termo, $\alpha^{(k-1)}$, onde k representa o índice do ciclo de *clock* de processamento.

O circuito de processamento serial das raízes do polinômio localizador está exposto na Figura 5.15. Pode-se observar que o circuito possui uma estrutura regular com μ células que calculam

valores intermediários de cada raiz. Inicialmente os registradores recebem os coeficientes do polinômio avaliador de erros, produzindo como resultado o valor de $\Omega(\alpha^0)$. Este valor é acumulado para que a próxima raiz $\Omega(\alpha^1)$ seja calculada e assim sucessivamente.

Tabela 5.4: Sequência de Verificação das Raízes do Polinômio Localizador Processados Serialmente

Ciclo k	Coefficiente Verificado	Verificação da Raíz
1 ^o	α^0	$\Omega(\alpha^0) = \omega_0 + \omega_1(\alpha^0)^1 + \omega_2(\alpha^0)^2 + \dots + \omega_\mu(\alpha^0)^\mu$
2 ^o	α^1	$\Omega(\alpha^1) = \omega_0 + \omega_1(\alpha^1)^1 + \omega_2(\alpha^1)^2 + \dots + \omega_\mu(\alpha^1)^\mu$
3 ^o	α^2	$\Omega(\alpha^2) = \omega_0 + \omega_1(\alpha^2)^1 + \omega_2(\alpha^2)^2 + \dots + \omega_\mu(\alpha^2)^\mu$
\vdots	\vdots	\vdots
n ^o	α^{n-1}	$\Omega(\alpha^{n-1}) = \omega_0 + \omega_1(\alpha^{n-1})^1 + \omega_2(\alpha^{n-1})^2 + \dots + \omega_\mu(\alpha^{n-1})^\mu$

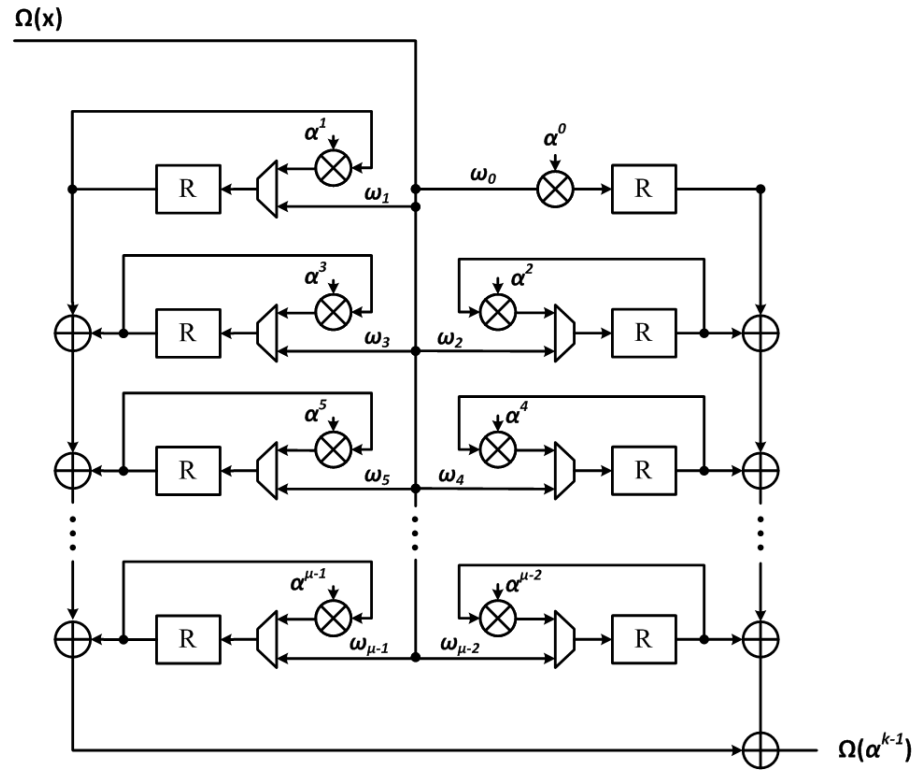


Figura 5.15: Arquitetura Serial do *Chien Search Evaluator*.

5.3.3.4 Arquitetura Paralela do Chien Search Evaluator

Para se implementar uma arquitetura *Chien Search Locator* paralelo, seguem-se os seguintes passos:

- (1) M : Define-se o número de símbolos paralelos a serem processados.
- (2) $CSEP_i$: Célula de processamento paralelo *Chien Search Evaluator*. Nesta etapa são geradas M células de processamento. Cada célula está representada na Figura 5.16. Cada uma das M células verifica uma raiz do polinômio avaliador por ciclo de *clock*. Esta sequência de verificação é apresentada na Tabela 5.3.

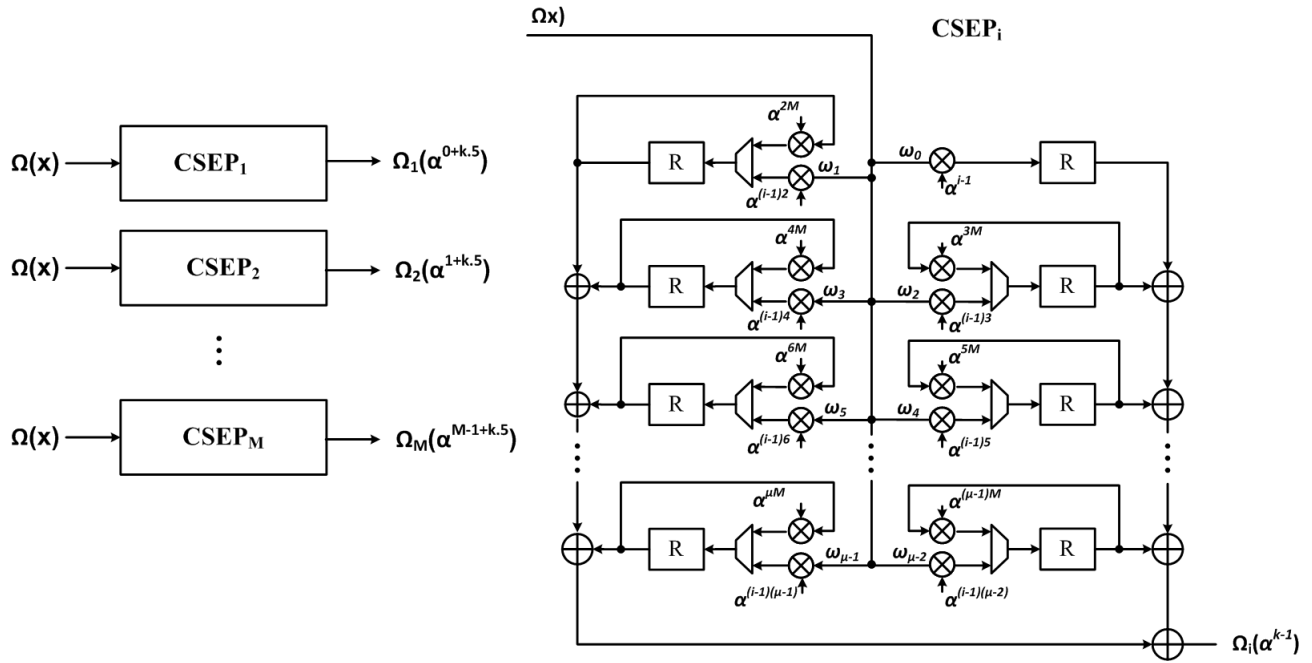


Figura 5.16: Arquitetura Paralela do Chien Search Locator.

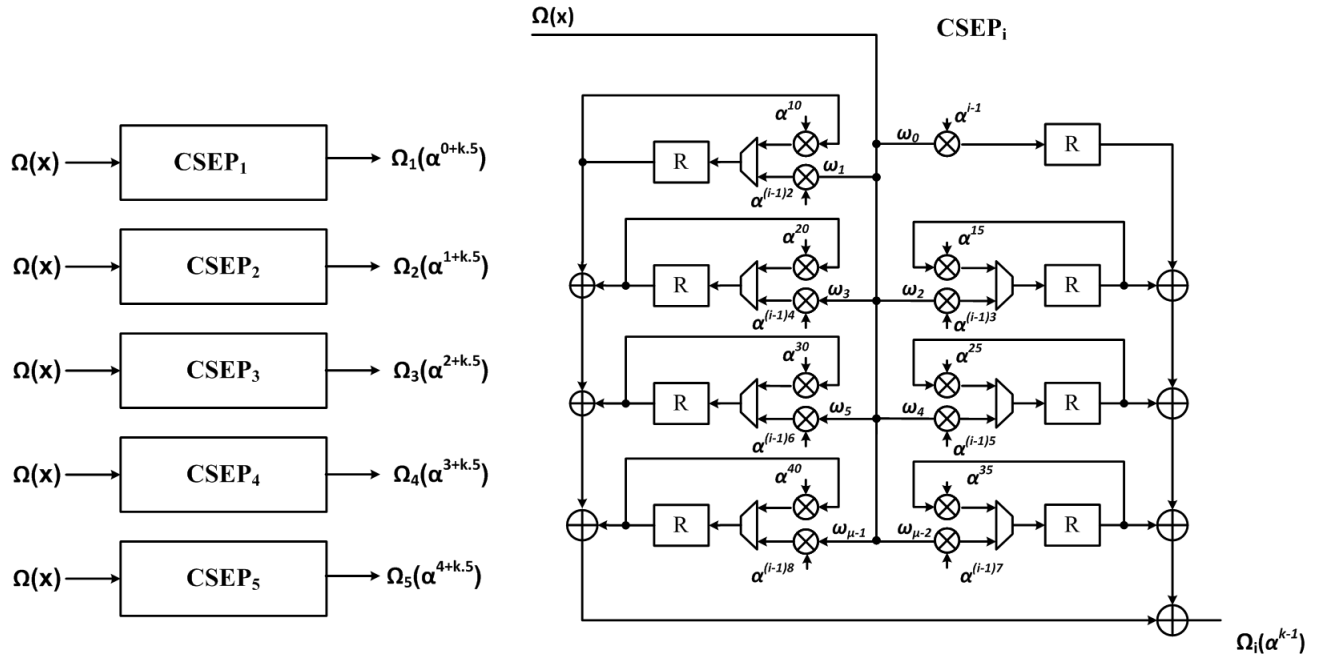
Tabela 5.5: Sequência de Verificação das Raízes do Polinômio Avaliador Processados Paralelamente (5 Coeficientes a Cada Ciclo de Clock)

Ciclo k	Coeficiente Verificado $i = 0, 1, 2, \dots, M - 1$	Verificação da Raíz
1 ^o	$\alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{M-1}$	$\Omega(\alpha^0) = \omega_0 + \omega_1(\alpha^0)^1 + \dots + \omega_\mu(\alpha^0)^\mu$ $\Omega(\alpha^1) = \omega_0 + \omega_1(\alpha^1)^1 + \dots + \omega_\mu(\alpha^1)^\mu$ \vdots $\Omega(\alpha^{M-1}) = \omega_0 + \omega_1(\alpha^{M-1})^1 + \dots + \omega_\mu(\alpha^{M-1})^\mu$
2 ^o	$\alpha^M, \alpha^{M+1}, \dots, \alpha^{2M-1}$	$\Omega(\alpha^M) = \omega_0 + \omega_1(\alpha^M)^1 + \dots + \omega_\mu(\alpha^M)^\mu$ $\Omega(\alpha^{M+1}) = \omega_0 + \omega_1(\alpha^{M+1})^1 + \dots + \omega_\mu(\alpha^{M+1})^\mu$ \vdots $\Omega(\alpha^{2M-1}) = \omega_0 + \omega_1(\alpha^{2M-1})^1 + \dots + \omega_\mu(\alpha^{2M-1})^\mu$
3 ^o	$\alpha^{2M}, \alpha^{2M+1}, \dots, \alpha^{3M-1}$	$\Omega(\alpha^{2M}) = \omega_0 + \omega_1(\alpha^{2M})^1 + \dots + \omega_\mu(\alpha^{2M})^\mu$ $\Omega(\alpha^{2M+1}) = \omega_0 + \omega_1(\alpha^{2M+1})^1 + \dots + \omega_\mu(\alpha^{2M+1})^\mu$ \vdots $\Omega(\alpha^{3M-1}) = \omega_0 + \omega_1(\alpha^{3M-1})^1 + \dots + \omega_\mu(\alpha^{3M-1})^\mu$
\vdots	\vdots	\vdots
$(n/M)^o$	$\alpha^{n-M}, \alpha^{n-M+1}, \dots, \alpha^{n-1}$	$\Omega(\alpha^{n-M}) = \omega_0 + \omega_1(\alpha^{n-M})^1 + \dots + \omega_\mu(\alpha^{n-M})^\mu$ $\Omega(\alpha^{n-M+1}) = \omega_0 + \omega_1(\alpha^{n-M+1})^1 + \dots + \omega_\mu(\alpha^{n-M+1})^\mu$ \vdots $\Omega(\alpha^{n-1}) = \omega_0 + \omega_1(\alpha^{n-1})^1 + \dots + \omega_\mu(\alpha^{n-1})^\mu$

Descrição do Circuito de *Chien Search Evaluator* do Decodificador FEC RS(255,239) Paralelo (ITU-T G.709)

(1) $M = 5$.

(2) *CSEP*: Neste exemplo são utilizadas 5 células de processamento *Chien Search Locator* paralela. A Figura 5.17

Figura 5.17: Arquitetura Paralela Exemplo do *Chien Search Evaluator*

5.3.4 Algoritmo de Forney

A última etapa de decodificação consiste em relacionar os resultados recebidos do bloco *Chien* e obter os valores dos erros na sua posição exata para que se possa corrigi-los. A equação utilizada para a determinação do erro é dada por

$$e_i = \frac{\Omega(\alpha^{-i})}{\Lambda'(\alpha^{-i})}. \quad (5.17)$$

Quando o resultado de $\Lambda(\alpha^{-i}) = 0$, significa que neste instante foi detectado um erro no dado recebido e portanto o valor de e_i deve combinado com os dados recebidos $R(x)$, devidamente atrasados para compensar o atraso de processamento do decodificador, para se obter os dados originais transmitidos $C(x)$.

Por outro lado, quando $\Lambda(\alpha^{-i}) \neq 0$, o sistema deve apenas transferir o dado recebido e atrasado para a saída.

5.3.4.1 Arquitetura de Implementação Serial do Algoritmo de Forney

O circuito do processamento serial do valor dos erros a serem corrigidos está exposto na Figura 5.18. O bloco *inv* implementa uma inversão em campo finito nos dados de entrada para

que a divisão apresentada em 5.17 seja calculada. O multiplexador de saída transfere o valor 0 para a saída sempre que um erro não for detectado, ou seja, $\Lambda(\alpha_{-i}) \neq 0$. Caso contrário o valor do e_i é diferente de zero e deve ser utilizado para corrigir os dados de entrada. A correção dos dados é realizada através de uma operação ou-exclusivo, conforme apresentado na Figura 5.18.

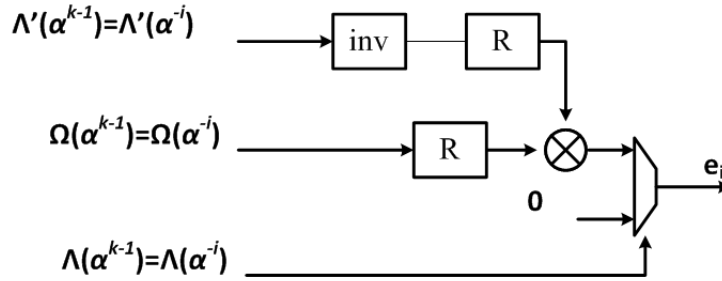


Figura 5.18: Arquitetura Serial de Implementação Serial do Algoritmo de Forney.

5.3.4.2 Arquitetura de Implementação Paralela do Algoritmo de Forney

Para se implementar uma arquitetura *Forney* paralelo, segue-se os seguintes passos.

- (1) M: Defini-se o número de símbolos paralelos a serem processados.
- (2) $CPSF_i$: Célula de processamento serial *Forney*. As células $CPSF_i$ são exatamente iguais às utilizadas no processamento paralelo do algoritmo de *Forney*. A única diferença é que no processamento paralelo estas células são replicadas M vezes para relacionar as raízes dos M polinômios parciais ($\Lambda'_i(x)$, $\Lambda_i(x)$ e $\Omega_i(x)$) e calcular o valor do erro simbólico da posição i do barramento, sendo que $i = 1, 2, \dots, M$. A Figura 5.19 apresenta a estrutura deste bloco.

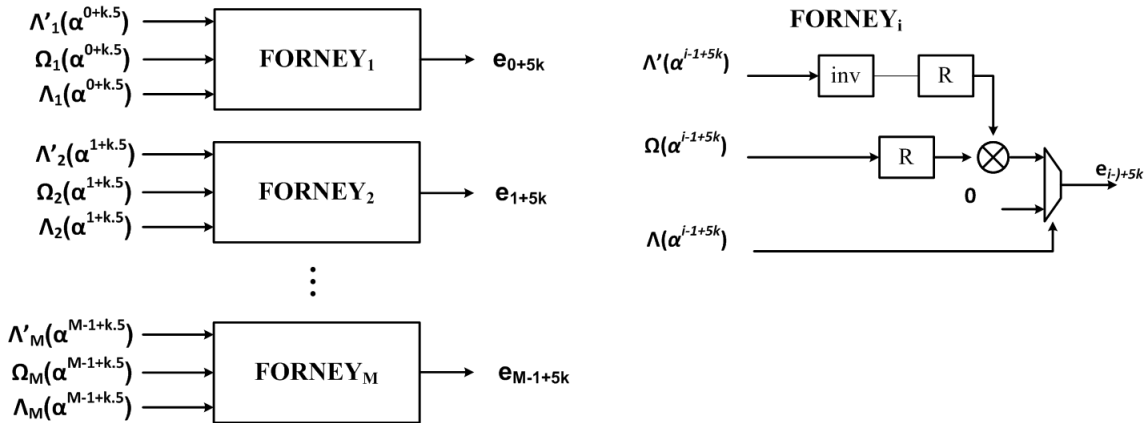


Figura 5.19: Arquitetura Paralela do Algoritmo de Forney.

Descrição do Circuito do Algoritmo de *Forney* do Decodificador FEC RS(255,239) Paralelo (ITU-T G.709)

Para o exemplo de implementação deste algoritmo quando $M=5$, são necessários 5 blocos de processamento Forney, conforme apresentado na Figura 5.20.

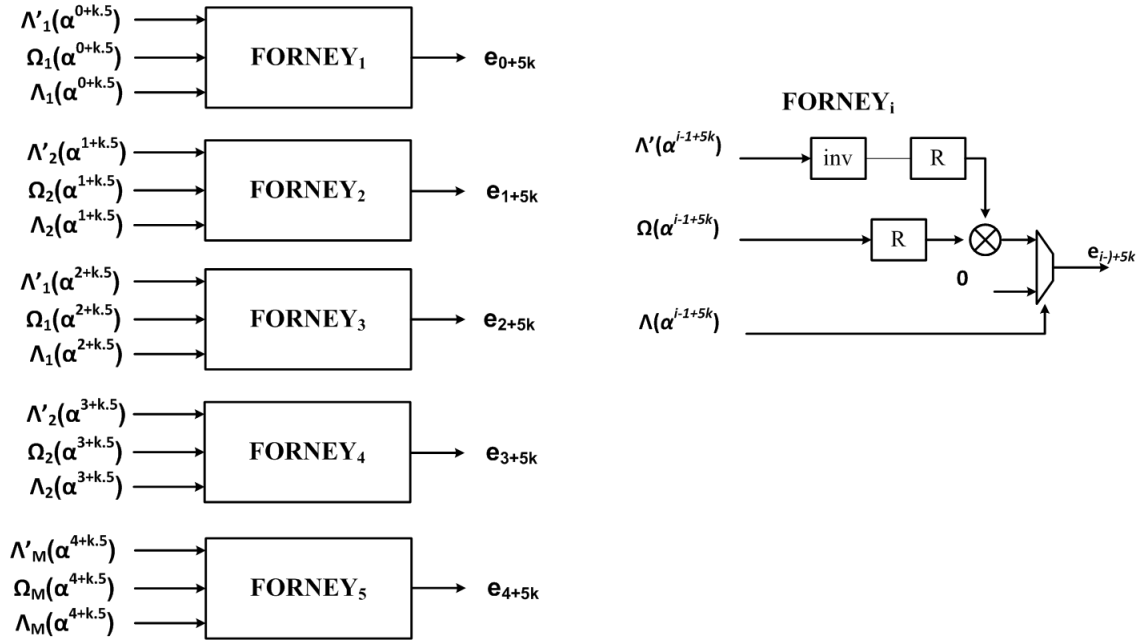


Figura 5.20: Arquitetura Paralela Exemplo do Algoritmo de *Forney*

Resultados e Desempenho do Circuito

Este capítulo visa descrever como o circuito do FEC foi simulado e validado em laboratório. Com relação aos testes físicos do circuito, é importante ressaltar que o algoritmo Reed-Solomon descrito é um código padronizado e conhecido no ambiente acadêmico e a implementação apresentada neste trabalho não altera seu desempenho de correção. Por esta razão, não é objetivo deste trabalho realizar uma caracterização detalhada do FEC. Os testes buscaram somente comprovar o correto funcionamento de todas as características do circuito.

6.1 Simulação do Circuito

O projeto do Codificador/Decodificador FEC foi validado em um dispositivo lógico programável do tipo **FPGA** (*Field Programmable Gate Array*). Este componente pode ser descrito, de forma muito simplificada, como um grande arranjo (*array*) de portas lógicas capazes de serem programadas e reconfiguradas para desempenhar funções lógicas específicas.

Existem diversos tipos de linguagens de descrição de *hardware* que são utilizadas para programar um FPGA, tais como: Verilog, VHDL, System Verilog, entre outras. O circuito do FEC foi descrito utilizando a linguagem VHDL.

Antes de programar o dispositivo com a lógica descrita, o circuito foi simulado utilizando uma abordagem baseada em *testbench*. Todos os testes foram implementados com a linguagem VHDL e com verificação baseada em *assertions*. Um *testbench* é essencialmente um código VHDL que instancia o projeto como um DUT (*Design Under Test*), gera rotinas de estímulos nas portas de entrada de um circuito e compara os resultados gerados nas portas de saída deste mesmo

DUT com os resultados de um modelo comportamental do bloco (*Golden Model*).

Foi realizado um plano de testes para o Codificador e para o Decodificador que visou testar todas as funcionalidades do bloco. Em simulação foram realizadas as seguintes avaliações: comportamento durante o estado de *reset*, validação campo de paridade do FEC, validação da decodificação sem que haja a inserção de erros, validação da decodificação inserindo erros em até 8 *bytes* da palavra código, monitoração de comportamento do bloco quando são inseridos erros em mais de 8 *bytes* da palavra código, teste em modo de transferência direta de dados da entrada para a saída (*bypass*), avaliação do bloco desabilitado, validação da inicialização da codificação e decodificação quando um novo quadro é iniciado e a validação da codificação quando os dados de entrada são alternados entre dados válidos e inválidos.

6.2 Ocupação de Recursos do FPGA

Um FPGA é projetado com blocos lógicos configuráveis (CLB), que são interconectados através de uma matriz de roteamento. O modelo de FPGA utilizado foi um Virtex-6 do fabricante Xilinx. Neste dispositivo, um CLB é subdividido em dois *slices*, os quais possuem 8 geradores de funções lógicas ou *look-up-tables* (**LUT**), 8 registradores, multiplexadores e lógica de *carry* [36]. Deste modo, qualquer arquitetura de circuito digital poderia ser implementada através da configuração destes blocos lógicos. Os recursos do FPGA são utilizados na elaboração de um circuito digital através de um processo de síntese.

A ferramenta de síntese disponibilizada pela Xilinx, o ISE, gera um relatório de desempenho em frequência, utilização de recursos e potência. Estas informações são disponibilizadas na Tabela 6.1.

A frequência de operação é o valor da frequência de trabalho utilizada nas aplicações de teste. No codificador paralelo, por exemplo, utilizou-se uma frequência de operação de 173 MHz. Considerando que são processados 5 *bytes* a cada ciclo de *clock* e que a ordem de entrelaçamento é de 16, $R_b = 173 \cdot 10^6 \times 16 \times 40 = 111,18 \text{ Gb/s}$ pode-se chegar na taxa de *bits* apresentada para este bloco.

Tabela 6.1: Medidas de Desempenho do codificador/Decodificador FEC

Parâmetro		FEC Serial (OTU2)		FEC Paralelo (OTU4)	
		<i>Codif.</i>	<i>Decodif.</i>	<i>Codif.</i>	<i>Decodif.</i>
Recursos	Slices	3.176	35.879	22.534	125.739
	Registradores	2.872	16.509	5.894	60.253
	LUTs	2.848	27.729	19.520	112.293
Desempenho	Frequência Máxima	546 MHz	200 MHz	372 MHz	253 MHz
	Frequência Operação	173 MHz	173 MHz	173 MHz	173 MHz
	Taxa	10,7 Gbit/s	10,7 Gbit/s	111,18 Gbit/s	111,18 Gbit/s
	Consumo	1,29 W	7,812 W	7,244 W	10,574 W

6.3 Arranjo Experimental

O teste físico visou avaliar o funcionamento do Codificador e Decodificador FEC de 10 Gbit/s e de 100 Gbit/s em laboratório. Foram utilizados dois ambientes de testes: O Meio Regenerador OTU2 e um Transponder 100 Gigabit-Ethernet. Nos dois casos o ambiente foi montado com os itens listados a seguir. Na descrição de cada item existe uma referência numérica que também é indicada na Figura 6.1.

1. Plataforma Xilinx ML630 [37] contendo dois FPGA's Virtex-6 do fabricante Xilinx [36] (Itens 3 e 4 da Figura 6.1). A placa ML630 possui 24 interfaces elétricas padrão GTH que podem operar a taxas de 10 Gbit/s.
2. Equipamento analisador de protocolo e gerador de tráfego *Optical Network Tester* tesxtbfONT-512 (Item 1 da Figura 6.1) fabricado pela empresa JDSU [38].
3. Placa extensora para módulo **CFP** e módulo transceptor CFP (Itens 2 e 5 da Figura 6.1).
4. Dispositivo *dongle de loopback* óptico (Item 6 da Figura 6.1). Este módulo converte 10 linhas de 10 Gbit/s ópticas para 10 linhas de 10 Gbit/s no domínio elétrico.

Para controlar e monitorar a aplicação, foi utilizado o *software* Chipscope, que é um conjunto de ferramentas da Xilinx que permite a monitoração de sinais internos do FPGA em tempo real. O equipamento *Optical Network Tester* (**ONT-512**) é um analisador de protocolo e gerador de tráfego que foi configurado para transmitir tráfego de dados nos protocolos OTU2, OTU4 ou 100 Gigabit-Ethernet. Os dados do gerador são transmitidos e recebidos simultaneamente através

As duas aplicações utilizadas para testar o codificador e decodificador FEC foram elaboradas no ambiente apresentado na Figura 6.1.

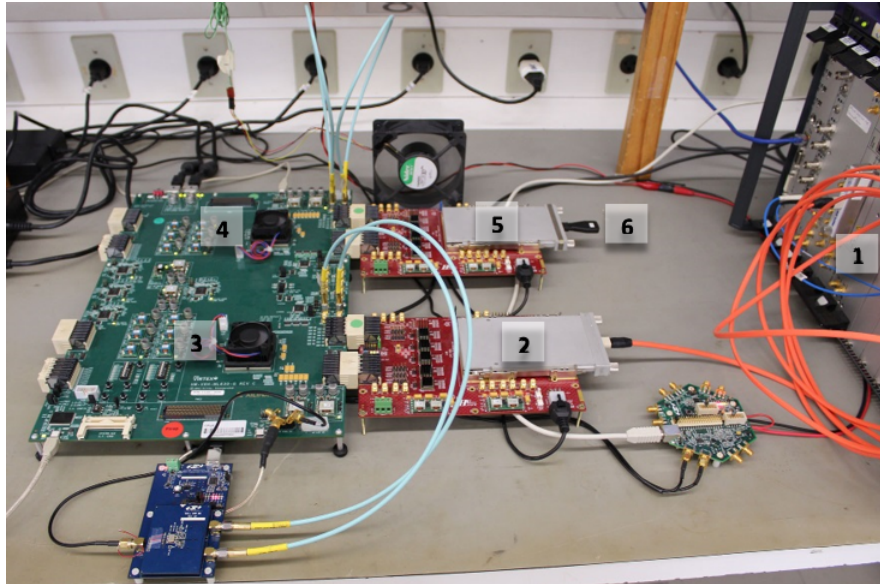


Figura 6.1: Ambiente Teste do Codificador e Decodificador FEC RS(255,239).

Este ambiente foi elaborado pensando somente no teste do FEC, não se tratando portanto de uma aplicação comercial para redes de transporte óptico. A aplicação do Meio Regenerador em questão corrige os erros de um quadro OTU2 somente em um sentido de tráfego, sendo portanto uma aplicação *simplex* e não *full-duplex* como normalmente se utiliza na prática. O diagrama em blocos da arquitetura do Meio Regenerador está apresentado na Figura 6.3.

Neste ambiente foram utilizados cinco blocos funcionais: Alinhador de quadros, Embaralhador e Desembaralhador (*Scrambler*), Codificador e Decodificador FEC. O blocos Alinhador de quadros e *Scrambler* são descritos a seguir.

O *Alinhador de Bit e Quadro* tem a função de encontrar o início de quadro através do cabeçalho de alinhamento (FA OH). Após identificar o início de quadro, o Alinhador posiciona os fluxos de dados de forma que os próximos recebam sempre o cabeçalho em uma posição conhecida no barramento de dados.

O *scrambler* é utilizado em sistemas de telecomunicações para evitar que uma sequência longa de *bits* 1's ou 0's sejam transmitidos. Caso isto ocorra, podem ocorrer problemas de sincronismo

na recepção [39]. Os dados do quadro OTN, exceto o campo de alinhamento (FA OH), devem ser embaralhados antes de serem transmitidos. O *scrambler* possui internamente um gerador de sequência pseudo-aleatória construído a partir do polinômio gerador ($x^{16} + x^{12} + x^3 + x + 1$). Esta sequência é combinada através de uma operação OU-exclusivo com os dados do quadro OTN.

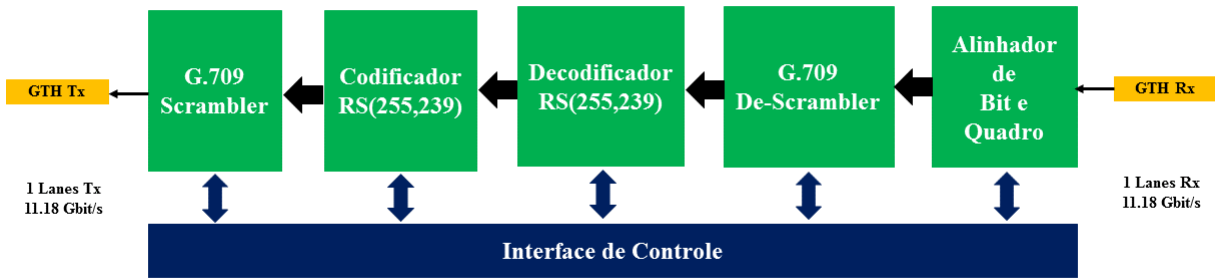


Figura 6.2: Diagrama em Blocos do Meio Regenerador

Procedimento Experimental 1 - Ganho de Codificação

Conforme mencionado na Seção 4.1.9, o ganho de codificação é um importante parâmetro para medir o desempenho de um circuito de correção de erros. Esta medida corresponde a diferença entre a SNR da saída de um decodificador e a SNR da mensagem codificada para uma mesma taxa de BER. Esta medida é tipicamente realizada relacionando SNR (E_b/N_0 , Q ou OSNR) versus BER. Porém, conforme a norma ITU-T G.975 esta medida pode também ser realizada variando o nível de potência óptica de entrada. Para o teste descrito no documento citado, o ganho de potência óptica esperada para uma BER de E-11 é entre 4 e 5 dB [22]. O teste realizado no *hardware* do, FEC descrito neste trabalho utilizou esta mesma estratégia, porém visou fazer somente uma medida do benefício do FEC no cenário de teste proposto e não uma caracterização do FEC.

Para medir este parâmetro, o equipamento ONT-512 foi configurado para gerar quadros OTU2. Estes dados foram transmitidos para o Meio Regenerador através de uma fibra óptica. Em paralelo com a linha foi inserido um medidor de potência óptica e um atenuador.

Para avaliar a BER na saída do decodificador, o codificador foi desabilitado através do módulo de controle Chipscope. O sinal decodificado retorna para o ONT-512 e sua BER é então avaliada, conforme ilustrado na Figura 6.3.

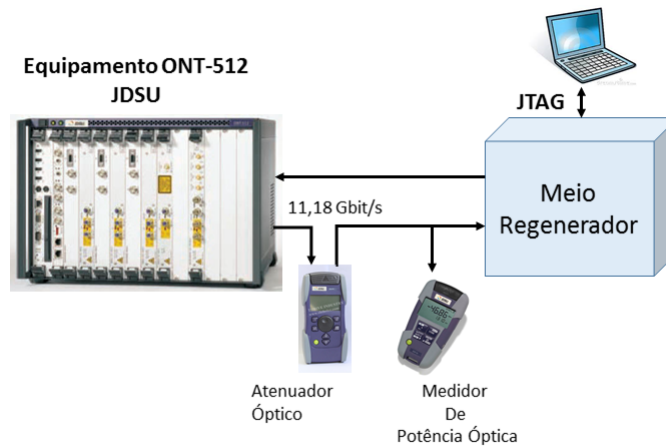


Figura 6.3: Teste 1

Deste modo, conhecendo-se a potência do sinal óptico inserido no decodificador FEC e a BER do sinal decodificado, foi possível levantar a curva do ganho do decodificador FEC RS(255,239), conforme apresentado na Figura 6.4. O ganho de nível de potência óptica de entrada observado na curva foi de 4.1 dB para uma BER de E-10. Conforme descrito nos estudos realizados por um grupo de estudos do IEEE [40], testes realizados com valor de BER muito baixos tornam-se não práticos para um dado tempo. Por exemplo, a uma taxa de 10 Gbit/s seriam necessários 5,8 dias para a ocorrência de 50 erros. Portanto, por consequência do tempo disponível dos equipamentos para os testes, fixou-se o valor mínimo de BER em E-12.

Vale dizer que o ganho de 4.1 dB de potência óptica medido neste teste é válido somente para este ambiente específico. Qualquer alteração de elementos desta estrutura poderia variar parâmetros do sistema (perdas de conexões, sensibilidade do CFP, a precisão do atenuador, a precisão do medidor, etc.), modificando assim o valor da medida.

6.3.2 Cenário de Teste - Transponder 100GbE

Neste ambiente foi utilizada uma aplicação OTN para avaliar o funcionamento do FEC dentro de um contexto real de trabalho. O Transponder 100GbE ilustrado na Figura 6.5, é uma aplicação que transmite e encapsula os dados de um quadro *Ethernet* em um quadro OTN a 111,18 Gbit/s. A padronização do formato e tamanho de quadros 100GbE é definida pela norma IEEE 802.3ba [41]. O Transponder OTN 100GbE possui duas interfaces: cliente e linha. Os pacotes Ethernet, recebidos a uma taxa de 103.25 Gbit/s, são transmitidos do equipamento Ethernet do cliente para o lado cliente do Transponder. No Transponder, os pacotes do cliente

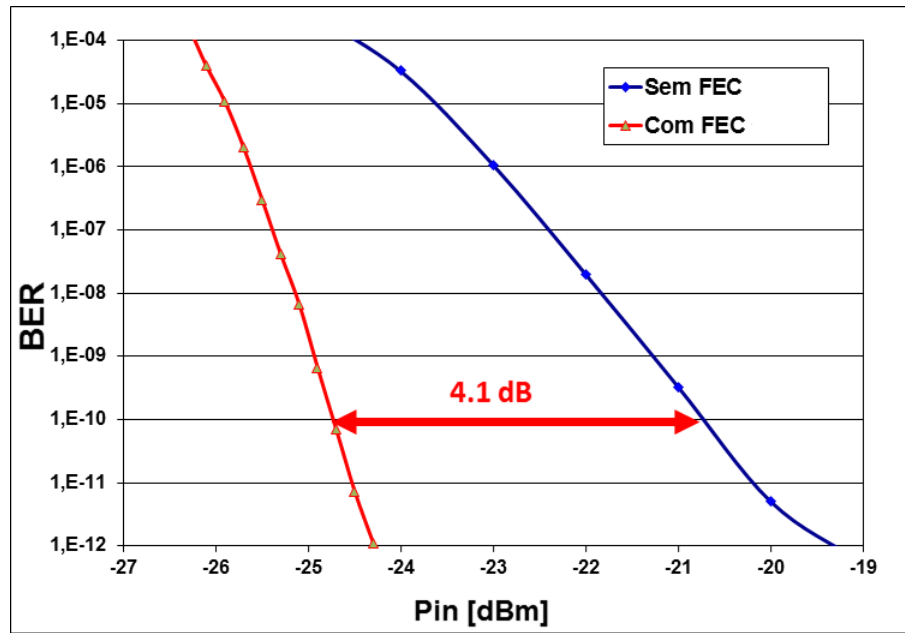


Figura 6.4: Ganho de Codificação

são encapsulados em quadros OTN e transmitidos para a interface de linha. A interface linha envia os pacotes OTN para a rede óptica. Na direção oposta a interface linha recebe os pacotes OTN, os erros são corrigidos pelo decodificador FEC, o cabeçalho OTN é processado e os pacotes Ethernet são desencapsulados e transmitidos para o seu destino.

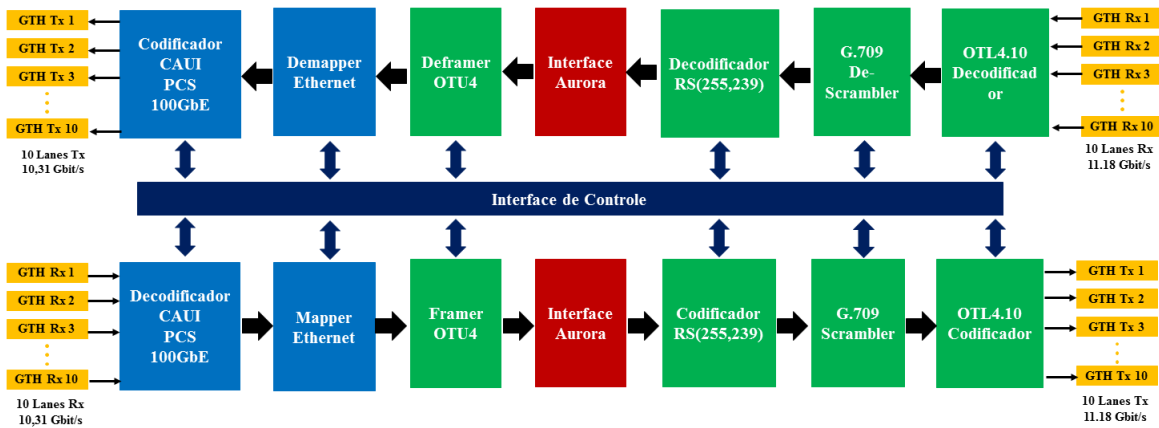


Figura 6.5: Diagrama em Blocos do Transponder 100 Gigabit-Ethernet

O projeto do Transponder 100GbE está dividido em dois grupos de blocos funcionais: blocos do Lado Linha, que implementam as funcionalidades do protocolo OTUk, definido na norma ITU-T G.709, e o conjunto de blocos do Lado Cliente, que implementam as funcionalidades do protocolo Gigabit-Ethernet, definido pela norma IEEE 802.3ba. Os blocos em verde imple-

mentam as funcionalidades do lado linha, que incluem: Codificador/Decodificador OTL4.10, *Scrambler/De-Scrambler*, Codificador/Decodificador G-FEC e *Framer/Deframer* OTU4. Os blocos em azul implementam as funcionalidades do lado cliente, que incluem: Codificador/Decodificador CAUI PCS e Mapper/Demapper Ethernet. O projeto do transponder distribuídos em dois FPGA's (Itens 2 e 5 da Figura 6.1). A linha de dados foi conectada entre os dois FPGA's através de uma interface proprietária do fabricante Xilinx, chamada de Interface Aurora. Esta interface está representada através do bloco vermelho.

As subseções seguintes descrevem cada um dos blocos do Transponder.

O Decodificador OTL4.10 tem como principal objetivo remontar o quadro OTU4 que é recebido através de 10 linhas de 11,18 Gbit/s. Inicialmente é realizado um alinhamento de bit em cada linha de dado. Na segunda fase, o atraso entre dados é compensado, as linhas são reordenadas e finalmente o quadro é remontado. O Codificador OTL4.10 recebe os dados do quadro OTU4 através de um barramento de dados com 640 *bits* proveniente do Scrambler G.709. O codificador tem a função de distribuir os dados do quadro para os 10 transceivers. Porém, anteriormente cada *lane* recebe um cabeçalho de dados para permitir que o decodificador seja capaz de alinhar e de compensar os atrasos entre as linhas.

O *scrambler*, conforme mencionado na descrição do ambiente de teste 1, é utilizado para evitar uma sequência muito longa de valores lógicos, ou níveis de tensão, repetidos na linhas de transmissão.

O *Framer* OTU4 é responsável por preencher os cabeçalhos de OTU e ODU. As informações dos cabeçalhos são fornecidas pela Interface de Controle e as ações consequentes são processadas dentro do bloco *Framer* OTU4. O *Deframer* OTU4 monitora os cabeçalhos OTU e ODU para avaliar as condições do canal de transmissão. O alinhamento de multiquadro é realizado e são gerados sinais de alarme quando alguma falha ocorre.

O *Mapper Ethernet* recebe blocos 64B/66B da Interface CAUI PCS, agrupa-os em conjuntos de 80 *bytes* e os encapsula na área de *payload* OPU4 usando o procedimento de mapeamento genérico, **GMP** (*Generic Mapping Procedure*). A distribuição é baseada em um algoritmo sigma/delta controlada por um através de um circuito de controle realimentado. A acomodação destes dados dentro do campo de OPU é realizada através do preenchimento dos campos vazias com *bytes* de *stuff*, que possuem valor zero.

A Interface CAUI realiza as funções descritas na subcamada **PCS** (*Physical Coding Sublayer*

do protocolo. O protocolo 100GbE opera com 20 *lanes* lógicos de 32 *bits* multiplexados em 10 *lanes* físicos. Esta multiplexação é realizada em um bloco chamado **PMA** (*Physical Medium Attachment*). Na recepção cada *lane* lógico é alinhado e sincronizado utilizando uma marcação chamada *Alignment Marker*. Assim como na interface OTL4.10, a interface CAUI também precisa fazer uma compensação dos atrasos entre os *lanes*. Posteriormente ao processo de *deskew*, os dados do quadro *Ethernet* são reordenados e remontados.

Procedimento Experimental 2 - Avaliação Funcional do Decodificador FEC

Para avaliar o decodificador FEC, foi utilizado um recurso específico para teste de FEC em quadro OTN disponível no equipamento ONT-512. Esta aplicação trata-se de um insersor de erros que possui três modos de operação:

- (1) **FEC Correctable**: Insere uma quantidade de erros de modo que o decodificador seja capaz de corrigir, ou seja, no máximo 8 bytes errados por palavra código.
- (2) **FEC Uncorrectable**: Insere uma quantidade de erros de modo que o decodificador não seja capaz de corrigir, ou seja, mais de 8 bytes errados por palavra código.
- (3) **FEC Stress**: Insere o maior número de erros que o dispositivo FEC é capaz de corrigir no menor tempo possível. Afeta todas as posições de *bit* em menos de 2 segundos.

Este recurso de inserção de erros foi utilizado para testar o decodificador FEC e sua operação se manteve estável em todas as condições de trabalho.

Para que o teste pudesse ser realizado, o módulo OTL4.10, Scrambler e Codificador/Decodificador FEC foram gravados no FPGA 1. Posteriormente foi implementado um *loopback* do Decodificador FEC para o Codificador FEC. O codificador foi desabilitado através do módulo de controle Chipscope e o equipamento ONT-512 foi configurado para capturar um histórico de dados não corrigidos. Desta forma, pôde-se validar módulo de correção com os múltiplos padrões de erros inseridos.

Procedimento Experimental 3 - Curva Característica de BER

Outro teste que mede a capacidade de correção do FEC é relacionar a taxa de erros de *bit* (BER - *Bit Error Rate*) de entrada do decodificador com a BER de saída através de um gráfico de BER de Entrada *versus* BER de Saída. Neste teste o insersor de erros do ONT-512

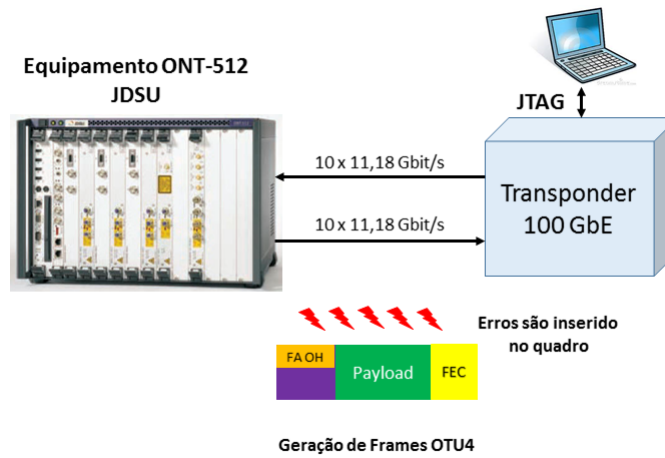


Figura 6.6: Teste de Stress do FEC

foi configurado para injetar um erro randômico com BER fixa. Foram gerados sinais com BER entre $1\text{E-}4$ e $8\text{E-}4$ e o sinal corrigido foi avaliado pelo próprio equipamento ONT-512. Deste modo, pôde-se montar o gráfico apresentado na Figura 6.7.

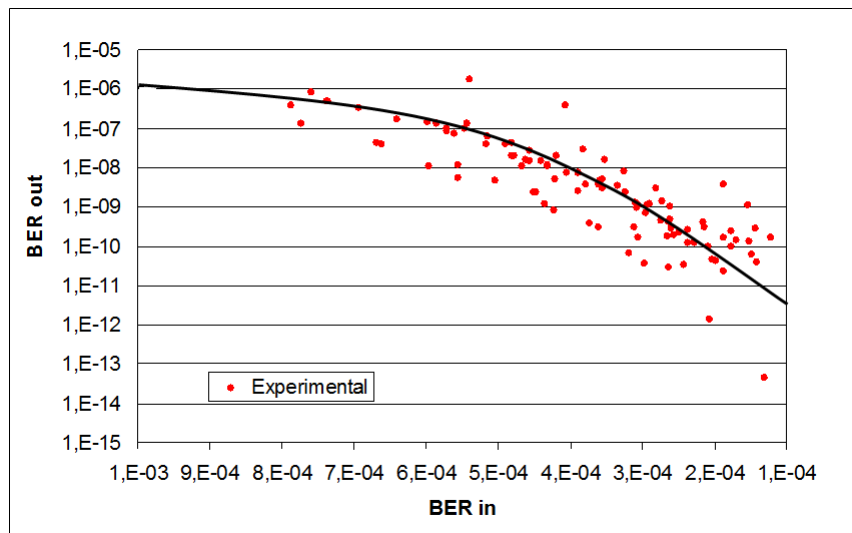


Figura 6.7: Gráfico BER Entrada x BER Saída Experimental

Procedimento Experimental 4 - Avaliação Funcional do Codificador FEC

Para avaliar o codificador FEC, nos dois ambientes de teste os decodificadores foram configurados para preencher o campo de FEC com valores iguais a zero. Os codificadores foram habilitados através da interface Chipscope. O equipamento ONT-512 foi habilitado para decodificar os campos de FEC dos quadros recebidos e avaliar se os dados estão corretos.

Este teste foi realizado por longos períodos e os codificadores de 10 Gbit/s e 100 Gbit/s foram validados.

Conclusões

O foco deste trabalho foi apresentar uma arquitetura paralela de FEC Reed-Solomon RS(255, 239) destinado a aplicações em redes ópticas que operem com taxas de dados elevadas. Além desta arquitetura foi apresentado também um método de conversão de um circuito convencional de processamento de FEC RS(n,k) em um estrutura de processamento paralelo.

Uma revisão teórica dos conceitos fundamentais que sustentam a teoria dos Campos Finitos e codificação, bem como da evolução dos protocolos de transporte óptico até os padrões atuais, foi apresentada nos Capítulos 2, 3 e 4.

No Capítulo 5, todas as fases para converter um circuito convencional de FEC em uma arquitetura paralela são descritas. Conforme proposto no tema do trabalho, o método foi aplicado para gerar um circuito seguindo as recomendações definidas na na recomendação ITU-T G.975. No Capítulo 6, a etapa experimental iniciou-se a partir do desenvolvimento do circuito codificador e decodificador FEC RS(255,239) em VHDL para as taxas de 10 Gbit/s e 100 Gbit/s. A versão para 10 Gbit/s utiliza a estrutura convencional. Por outro lado, a versão para 100 Gbit/s utiliza a estrutura paralela proposta. Os circuitos FEC foram incluídos em duas aplicações OTN: o Meio Regenerador OTU2 e um Transponder 100GbE.

As duas primeiras avaliações realizadas buscaram verificar o correto comportamento do FEC. No teste do codificador uma grande variedade de mensagens foram codificadas e o campo de paridade foi avaliado pelo equipamento ONT-512. No teste do decodificador todas as possibilidades de inserção de erros oferecidas pelo equipamento ONT-512 foram exploradas.

Na opinião do autor, estes testes foram conclusivos na validação do FEC. Entretanto, ainda foram levantadas duas curvas para ilustrar o benefício do FEC.

O primeiro gráfico foi a curva de ganho de codificação. De acordo com a norma ITU-T G.975 [22], o ganho de codificação do FEC apresentado está dentro do esperado. O segundo gráfico foi utilizado para comprovar a capacidade de correção do FEC, relacionando a BER de saída com a BER de entrada do FEC. Como o FEC RS(255,239) é um algoritmo padronizado, a implementação apresentada neste trabalho não ofereceu melhorias em suas curvas de desempenho quando comparadas com as publicadas pela norma ITU-T G.975.

Com relação às perspectivas futuras de novos trabalhos, o aumento na demanda por banda larga aponta para a necessidade de se ultrapassar o limite dos 100 Gbit/s. Para as novas gerações de OTN, a nova hierarquia de OTU5 será necessária para a acomodação transparente das próximas gerações Ethernet [42]. O autor acompanha os estudos e propostas realizadas pelo grupo de estudos 15 (*Student Group 15, SG15*), que busca a definição do novo padrão OTN para as novas gerações de transmissão em fibra com taxas iguais e/ou superiores a 400 Gbit/s. Com o surgimento de padrões para 400 Gbit (400 GbE e OTU5) e as atuais discussões sobre qual a melhor forma de ultrapassar o limite de 1 Tbit/s indicam uma necessidade grande de esforço na expansão e otimização das tecnologias atuais. O FEC como sendo uma tecnologia já incorporada ao protocolo de redes de transporte óptico deverá também sofrer grandes melhorias. Este será o grande desafio: não somente melhorar a velocidade de processamento dos dados do FEC, mas também utilizar novos esquemas com maior eficiência.

Bibliografia

- [1] Geraldo Gil R. Gomes, “Codificação de Canal: Códigos de Bloco Lineares,” 2011. Disponível em http://www.cesarkallas.net/arquivos/faculdade-pos/TP301-codificacao-fonte/2_Bloco_V2011_Rev2.pdf, Acesso em 10 fev. 2015.
- [2] H. A. E. A. Elsaid, “Robust optical transmission systems, modulation and equalization,” Master’s thesis, Technische Universiteit Eindhoven, Eindhoven, Holanda, September 2008.
- [3] C. E. Shannon, *A Mathematical Theory of Communication*. The Bell System Technical Journal, vol. 27, pp. 379–423, 623–656 ed., July 1948.
- [4] J. Castiñeira and P. G. Farrel, *Essentials of Error-Control Coding*. John Wiley & Sons, 1 ed., September 2006.
- [5] R. Bernardo, A. H. Salvador, E. Mobilon, and L. R. Monte, “Design and FPGA Implementation of a 100 Gbit/s Optical Transport Network Processor,” *International Conference on Field Programmable Logic and Applications - FPL2013*. Porto, Portugal, August 2013.
- [6] A. H. Salvador, C. A. Nakandakare, D. S. Arantes, D. B. de Carvalho, E. Mobilon, and J. C. R. F. de Oliveira, “100 Gbit/s G-FEC Design for OTN Protocol: Design Architecture and Implementation Results,” *International Telecommunications Symposium - ITS2014*. São Paulo, Brasil, August 2014.
- [7] A. H. Salvador, C. A. Nakandakare, D. B. de Carvalho, and E. Mobilon, “111.18 Gbit/s G-FEC for OTN Applications: Theory and Implementation Techniques,” *Design Conference - DesignCon2014*. Santa Clara, USA, January 2014.

- [8] A. H. Salvador, “Método de Codificação FEC para Sistemas de Transmissão OTN com Alta Taxa de Transmissão.” Patente INPI BR 10 2013009315 7, Fevereiro 2013.
- [9] H. A. E. A. Elsaid, “Design and implementation of reed solomon decoder using decomposed inversionless berlekamp-massey algorithm,” Master’s thesis, Faculty of Engineering at Cairo University, Cairo, Egypt, September 2010.
- [10] P. Dayal and R. K. Patial, “FPGA Implementation of Reed-Solomon Encoder and Decoder for Wireless Network 802.16,” *International Journal of Computer Applications*. Punjab, India, April 2013.
- [11] T. C. Barbosa, “Implementação em fpga de uma arquitetura reed-solomon para uso em comunicações Ópticas,” Master’s thesis, Universidade Federal de Itajubá, Itajubá, Brasil, Agosto 2010.
- [12] T. Grimm, “Desenvolvimento em linguagem de descrição de hardware de codificador e decodificador reed-solomon,” Master’s thesis, Universidade Federal de Santa Catarina, Florianópolis, Brasil, Fevereiro 2014.
- [13] S. Das, “Vhdl implementation of reed-solomon coding,” Master’s thesis, National Institute of Technology Rourkela, Rourkela, India, May 2011.
- [14] M. Elharaussi, A. Hamyani, and M. B. E. Rabat, “VHDL Design and FPGA Implementation of a Parallel Reed-Solomon(15, K, D) Encoder/Decoder,” *International Journal of Advanced Computer Science and Applications*. West Yorkshire, United Kingdom, January 2013.
- [15] H. Tezuka, “Device and method for carrying out reed-solomon encoding.” Available in <http://www.google.com/patents/US6219816>. Patent Number US 6219816 B1, February 2001.
- [16] K. Seki, “Parallel processing reed-solomon encoding circuit and method.” Available in <http://www.google.com.ar/patents/US6859905>. Patent Number US 6859905 B2, March 2005.
- [17] ITU-T G.709, “Interfaces for Optical Transport Network (OTN),” Recommendation Y.1331, International Telecommunication Union, Geneva, Switzerland, February 2012.

- [18] R. Bernardo, T. R. Tronco, C. L. A. Lessa, and E. Mobilon, “OTN: A Evolução das Redes de Transporte,” *Caderno Técnico CPqD*, 2011. Campinas, Brasil, Dezembro 2011.
- [19] T. R. Tronco, *Redes de Nova Geração*. Editora Érica, 2006.
- [20] *Mapping and Transport Standard for OTU4*, vol. 5805, IEEE Communications Society. USA, 2010.
- [21] M. Abou-Shaban, “Over-Clocked Optical Transport Network,” *Expertise Reaching Out - EXFO*. Quebec, Canada, Mach 2009.
- [22] ITU-T G.975, “Forward Error Correction for Submarine Systems,” Recommendation Y.1331, International Telecommunication Union, Geneva, Switzerland, February 2000.
- [23] ITU-T G.709 Tutorial, “Optical Transport Network (OTN) Tutorial,” Recommendation Y.1331, International Telecommunication Union, Geneva, Switzerland, February 2010.
- [24] G. L. Mullen and D. Panario, *Handbook of Finite Fields*. CRC Press, 2 ed., 2013.
- [25] H. S. Madeira, “Introdução a teoria de galois: Uma perspectiva histórica.,” Master’s thesis, Universidade Católica de Brasília, Brasília, Brasil, Fevereiro 2008.
- [26] R. E. Blahut, *Theory and practice of error-control codes*. Addison-Wesley, 2 ed., 1983.
- [27] D. J. C. Shu Lin, *Error Control Coding: Fundamentals and Application*. Prentice-Hall, 2 ed., 2004.
- [28] Aliathon, “FEC Overview,” 2013. Disponível em http://www.aliathon.com/our_products/otn/fec, Acesso em 10 fev. 2015.
- [29] R. E. Ziemer and R. W. Peterson, *Introduction Digital Communications*. Upper Saddle River: Prentice Hall, 2 ed., 2001.
- [30] E. R. Berlekamp, *Algebraic Coding Theory*. McGraw-Hill, 2 ed., 1968.
- [31] Y. Jiang, *A Practical Guide to Error-Control Coding Using MATLAB*, vol. 4 of 10. Norwood, USA: Artech House, 2 ed., May 2011.
- [32] F. R. Henríquez, *Cryptographic Algorithms on Reconfigurable Hardware*. Springer, 2 ed., 2007.

- [33] K. Kobayashi, N. Takagi, and K. Takagi, “An Algorithm for Inversion in $GF(2^m)$ Suitable for Implementation Using a Polynomial Multiply Instruction on $GF(2)$,” *Conference: Computer Arithmetic*. Nagoya, Japan, October 2007.
- [34] T. Kohonen, *Content-Addressable Memories*. Springer, 2 ed., 1989.
- [35] D. Sarwate and N. Shanbhag, “High-Speed Architectures for Reed-Solomon Decoders,” *Very Large Scale Integration*, Outubro 2001.
- [36] Xilinx, “Virtex-6 FPGA GTH Transceivers, User Guide UG371,” 2011. Disponível em <http://www.xilinx.com/support/>, Acesso em 20 jan. 2015.
- [37] Xilinx, “ML630 Virtex-6 HXT FPGA Optical Transmission Network Evaluation Board,” 2011. Disponível em http://www.xilinx.com/support/documentation/boards_and_kits/ug828_ML630_eval_bd.pdf, Acesso em 10 dez. 2014.
- [38] JDSU, “ONT-512 Optical Network Tester,” 2012. Disponível em <http://www.jdsu.com>, Acesso em 15 out. 2014.
- [39] A. H. Salvador and V. Corso, “100 Gbit/s Scrambler Architectures for OTN Protocol: FPGA Implementation and Result Comparison,” *Midwest Symposium on Circuits and systems - MWSCAS 2012*. Idaho, USA, August 2012.
- [40] Michael Fleischer Reumann, “BER Measurements for 100GbE. IEEE 802.3 Higher Speed Study Group,” May 2008. Disponível em http://www.ieee802.org/3/hssg/public/jan07/fleischer_reumann_01_0107.pdf, Acesso em 10 fev. 2015.
- [41] IEEE 802.3ba, “CSMA/CD (Ethernet),” Recommendation Y.1331, International Telecommunication Union, Geneva, Switzerland, February 2010.
- [42] G. L. Cesar, “Fundamentos do padrão OTN,” Master’s thesis, Escola Politécnica da USP, São Paulo, Brasil, Setembro 2014.