Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Códigos Corretores de Erros em Hardware para Sistemas de Telecomando e Telemetria em Aplicações Espaciais

Gabriel Marchesan Almeida

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Eduardo Augusto Bezerra

Porto Alegre, Março de 2007

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Códigos Corretores de Erros em Hardware para Sistemas de Telecomando e Telemetria em Aplicações Espaciais***", apresentada por Gabriel Marchesan Almeida, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sstemas Embarcados e Sistemas Digitais, aprovada em 12/01/2007 pela Comissão Examinadora:

Prof. Dr. Eduardo Augusto Bezerra – PPGCC/PUCRS
Orientador (a)

Prof. Dr. Dalcidio Moraes Claudio – PPGCC/PUCRS

Prof. Dr. Rubem Dutra R. Fagundes – FENG/PUCRS

Homologada em 02/04/2007, conforme Ata No. 008 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 16 – sala 106 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

Dados Internacionais de Catalogação na Publicação (CIP)

A447c Almeida, Gabriel Marchesan
Códigos corretores de erros em hardware para sistemas de telecomando e telemetria em aplicações espaciais / Gabriel Marchesan Almeida. – Porto Alegre, 2007.
96 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS
Orientador: Prof. Dr. Eduardo Augusto Bezerra

1. Informática. 2. Códigos Corretores de Erros.
3. Telecomando. 4. Telemetria. 5. Sistemas Espaciais.
6. VHLD I. Título.

CDD 004.22

**Ficha Catalográfica elaborada pelo
Setor de Processamento Técnico da BC-PUCRS**

PUCRS
Campus Central
Av. Ipiranga, 6681 – prédio 16 – CEP 90619-900
Porto Alegre – RS – Brasil
Fone: +55 (51) 3320-3544 – Fax: +55 (51) 3320-3548
Email: bceadm@pucrs.br
www.pucrs.br/biblioteca

À minha mãezinha Eleniza, que além de prover condições para meus estudos, soube me auxiliar e ser meu ponto de apoio em todos os momentos. Ao meu irmão Rafael que sempre me serviu de exemplo e incentivo para seguir na área. À minha esposa Bruna, sempre companheira, amorosa e carinhosa em todas as horas, meu fruto de inspiração. Aos meus familiares que sempre estiveram juntos em oração e pensamento e em especial à jóia preciosa da família, meu filho João Gabriel.

Agradecimentos

Obrigado meu bom Deus pelo dom da vida, pela minha saúde e pela inteligência que me deste. Obrigado por estar ao meu lado nas horas difíceis e por iluminar e guiar os meus caminhos a todo instante. Agradeço também a todos os meus anjos terrestres, iniciando com quem me concebeu a vida, minha mãezinha querida Eleniza, a qual tenho muita admiração e amo muito. Ela que me deu todo o suporte emocional, afetivo, financeiro e principalmente seu amor de mãe. Obrigado por estar sempre ao meu lado, guiando meus passos e torcendo a cada vitória, cada conquista, cada etapa alcançada. Agradeço também ao meu irmão Rafael, que sempre me incentivou e sempre me ajudou nas tomadas importantes de decisões que surgiram no decorrer da minha vida.

Agradeço à minha esposa Bruna, sempre companheira, amiga, que soube me entender quando não pude estar presente devido aos compromissos do mestrado e principalmente por me fazer feliz e completo a cada dia. Um obrigado muito especial também ao meu filho João Gabriel, que nasceu para ajudar com seus chorinhos na escrita da minha dissertação, que me deu dicas importantes e que me ensina a cada dia ser um pai cada vez mais babão. Agradeço a todos os meus familiares que me apoiaram e sempre estiveram ao meu lado durante essa fase da minha vida. Agradeço também aos meus amigos, pessoas especiais as quais pude sempre contar quando precisei de alguma ajuda. Obrigado aos meus colegas de mestrado, em especial às pessoas do Grupo de Sistemas Embarcados (GSE), pela amizade, espírito de companherismo e por me aturarem durante esse período. Faço um agradecimento especial ao meu amigo Luis Vitório Cargnini que me ajudou durante as etapas de estudo dos algoritmos utilizados no trabalho e por sempre estar disponível para ajudar quando precisei. Agradeço ao meu orientador Bezerra, que sempre soube ser paciente, incentivador e o mais importante, amigo em todas as horas, além é claro de ser meu padrinho de casamento. Valeu Bezerra por tudo o que fizeste por mim e minha família, valeu padrinho!!! Vê se na minha próxima cirurgia aparece pra me fazer a visita e levar a maçã quando eu ainda estiver no hospital. =)

Também agradeço aqueles ao qual me deram suporte acadêmico e financeiro para que esse trabalho pudesse ser concluído. Inicialmente agradeço a PUC por acreditar no meu potencial como aluno e por me prover o suporte acadêmico necessário para contribuir com a minha formação. Agradeço também à todas as pessoas do PPGCC que sempre me ajudaram quando precisei. Obrigado ao CNPq por me prover o suporte financeiro necessário para o desenvolvimento do trabalho durante esse período. Obrigado também à AEB (Agência Espacial Brasileira) pela oportunidade que me deu em participar de um projeto em âmbito nacional e pelo suporte financeiro alocado para minha viagem de estágio no exterior. Obrigado ao INPE (Instituto Nacional de Pesquisas Espaciais) pela oportunidade de participar do programa UNIESPAÇO como parceiro de projeto, em especial ao pesquisador Fernando Pessota, que acompanhou de perto o desenvolvimento do mesmo durante esses 2 anos. Um obrigado também ao professor Rubem Dutra Ribeiro Fagundes, que me ajudou com a parte matemática dos algoritmos de Reed-Solomon e BCH, principal foco desse trabalho.

Obrigado à *University of Sussex/UK*, em especial às pessoas do *Space Science Centre*, que me acolheram e me receberam de braços abertos, tornando possível a conclusão das minhas pesquisas. Obrigado ao professor Paul Gough, coordenador do grupo de pesquisas espaciais da universidade e ao professor Andy Buckley, por terem me acolhido e me dado todo o suporte que precisei. Agradeço também aos meus amigos Ana Vitória Hulshof e Eduardo Hulshof, que me receberam como um irmão em sua casa durante o período em que estive em Brighton/UK. Agradeço ainda aos meus amigos Gibson e Mônica, que também me acolheram e me chamaram para as sessões de degustação de cerveja e vinho durante o período em que precisei de boas idéias.

Resumo

Esse trabalho apresenta uma pesquisa acadêmica no escopo de códigos corretores de erros empregados em sistemas espaciais. O principal objetivo desse trabalho contempla o projeto, implementação e validação de circuitos corretores de erros para dados de telemetria e telecomando, seguindo o padrão CCSDS (*Consultative Committee for Space Data Systems*). Ambos os módulos de telemetria e telecomando são descritos em linguagem VHDL e implementam, respectivamente, os algoritmos de correção de erros Reed-Solomon e BCH (Bose, Chaudhuri and Hocquenghem), os quais possuem alta capacidade de correção de erros ocorridos durante o processo de transferência de dados entre o veículo espacial e a base terrestre.

Palavras-chave: Reed-Solomon, BCH, Códigos Corretores de Erros, FPGA, VHDL, CCSDS, SoC, Telemetria, Telecomando.

Abstract

This work investigates the hardware implementation of error correcting codes algorithms for space applications. The goal is the design, implementation and validation, of a basic telecommand and telemetry system, following the CCSDS (Consultative Committee for Space Date System) standard. The whole system is conceived targeting configurable computing technology. Both telemetry and telecommand modules are written in VHDL language employing, respectively, Reed-Solomon (RS) and Bose, Chaudhuri and Hocquenghem (BCH) algorithms for error correcting. These algorithms present high error correcting capabilities, which is important when considering the noise channel link for data transference between a spacecraft and a ground station.

Keywords: Reed-Solomon, BCH, Error Correcting Codes, FPGA, VHDL, CCSDS, SoC, Telemetry, Telecommand.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Símbolos e Abreviaturas

AEB Agência Espacial Brasileira

AMBA *Advanced Microcontroller Bus Architecture*

ASIC *Application Specific Integrated Circuits*

BCH *Bose, Chaudhuri and Hocquenghem*

BSC *Binary Symmetric Channel*

CAD *Computer Aided Design*

CAN *Controller Area Network*

CCSDS *Consultative Committee for Space Data Systems*

CD *Compact Disc*

CI Circuitos Integrados

CLB *Configurable Logic Block*

CLTU *Command Link Transmission Unit*

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico

CORDIC *Coordinate Rotation Digital Computer*

CPDU *Command Pulse Distribution Unit*

CRC *Cyclic Redundancy Check*

DVB *Digital Video Broadcast*

DVD *Digital Video Disc*

DSP *Digital Signal Processor*

ECC *Error Correcting Code*

EDAC *Error Detection-and-Correction*

ESA *European Space Agency*

ESTO *Earth Science Technology Office*

FPGA *Field Programmable Gate Array*

GF *Galois Field*

HDLC *High Level Data Link Control*

IDE *Integrated Drive Electronics*

IEEE *Institute of Electrical and Eletronic Engineers*

IOB *Input/Output Block*

IP *Intellectual Property*

IPv6 *Internet Protocol version 6*

LFSR *Linear Feedback Shift Register*

LI Linearmente Independentes

LUT *Look-up Table*

MEA *Modified Euclidean Algorithm*

MPEG *Moving Picture Experts Group*

MSB *Most Significant Bit*

NASA *National Aeronautics and Space Administration*

OBC *On-Board Computer*

PLD *Programmable Logic Device*

PUCRS Pontifícia Universidade Católica do Rio Grande do Sul

RAID *Redundant Arrays of Independent Disks*

RAM *Random Access Memory*

RS *Reed-Solomon*

RTOS *Real-Time Operating Systems*

SCPS *Space Communications Protocol Standards*

SIMD *Single Instruction Multiple Data*

SoC *System on a Chip*

SoCs OBC *System on a Chip On-board Computer*

SRAM *Static Random Access Memory*

TC Telecomando

TM Telemetria

VHDL *VHSIC Hardware Description Language*

VHSIC *Very High Speed Integrated Circuit*

VLSI *Very Large Scale Integration*

Capítulo 1

Introdução

Sistemas computacionais da atualidade têm utilizado, com freqüência, mecanismos capazes de garantir a integridade dos dados processados. Algumas aplicações como por exemplo CD (*Compact Disc*), TV digital e, em especial aplicações no contexto espacial, necessitam garantir que os dados transmitidos através de um canal ruidoso, propenso a erros, sejam entregues da mesma maneira com que foram enviados. Um canal é definido como o meio por onde os dados trafegam, onde os mesmos estão sujeitos a ruídos e interferência, podendo sofrer alterações durante o processo de transmissão. A Figura 1.1 apresenta o processo de envio de dados, representado por d. Antes do envio, os dados são codificados, representados por c. A mensagem c é então enviada e devido à interferências no processo de transmissão, essa mensagem é recebida alterada, representada por x. O destinatário então decodifica a mensagem e o dado é recebido, representado por d'. Para garantir a corretude dos dados, existem diferentes classes de códigos corretores de erros (ECC[1]) disponíveis atualmente. Esses códigos funcionam através da aplicação de algoritmos para codificação e decodificação de forma tal que a mensagem recebida seja recuperada ao sofrer interferências durante o processo de transmissão.

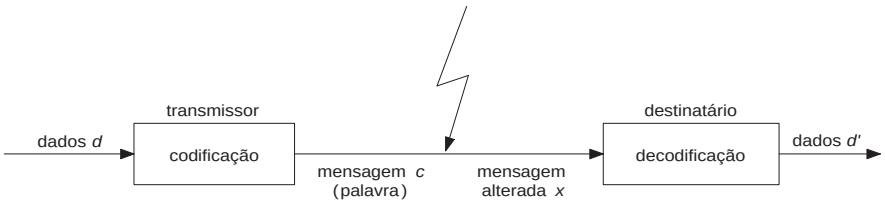

Figura 1.1: Interferência ocorrida durante o processo de transmissão de dados

Alguns códigos apenas possuem a capacidade de detectar erros e funcionam a partir da identificação da ocorrência de erros durante o processo de transmissão. Ao identificar um erro o módulo receptor pode simplesmente descartar a mensagem corrompida, ou enviar um pedido de retransmissão da mensagem para o emissor. A Figura 1.2 apresenta o fluxo de pedido de retransmissão da mensagem corrompida.

O processo emissor envia a mensagem através de um canal propenso a erros, a mensagem então é corrompida e o processo receptor ao detectar o erro, envia um pacote de solicitação de retransmissão da mensagem. A mensagem é então retransmitida, e ao não detectar-se novos erros, a mesma é passada para a aplicação de destino.

[1] do inglês *Error Correcting Codes*

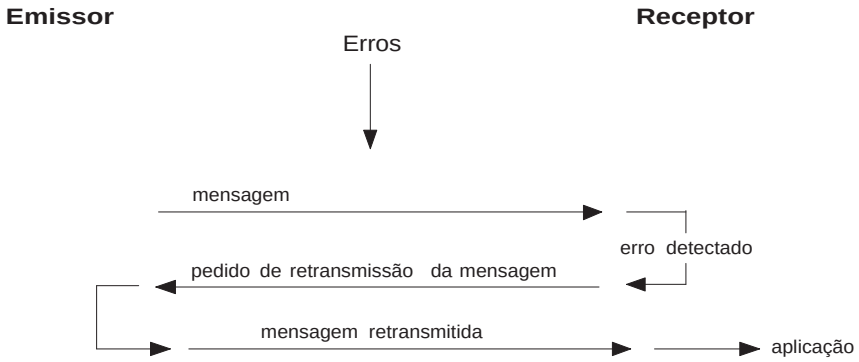

Figura 1.2: Representação de um pedido de re-envio da mensagem corrompida

Nesse trabalho são investigadas as classes de códigos corretores de erros *Reed-Solomon* (RS) [1] e *Bose-Chaudhuri-Hocquenghem* (BCH) [2, 3]. Essas classes de códigos utilizam o conceito de corpos finitos e os algoritmos funcionam basicamente através de cálculos sobre polinômios. A teoria sobre corpos finitos compreende um campo da álgebra abstrata que está diretamente ligada à teoria de Galois, utilizada nos dois algoritmos citados anteriormente. No contexto de aplicações espaciais, os códigos RS e BCH são os algoritmos mais utilizados atualmente [4, 5] e servem para assegurar que pacotes transmitidos de um veículo espacial para uma estação terrestre e vice-versa sejam facilmente recuperados quando corrompidos durante o processo de transmissão de dados.

Tradicionalmente códigos RS têm sido empregados na codificação de canal devido às suas excelentes propriedades de codificação e decodificação. Chama-se de codificação de canal a codificação de sinais de informação com o objetivo de diminuir a taxa de erro de símbolo e/ou de bit durante a transmissão dos mesmos através de um canal de comunicação. Códigos Reed-Solomon são blocos de códigos lineares não binários capazes de recuperar erros aleatórios tão bem quanto erros em rajada[2] [6]. Algumas de suas propriedades incluem, não somente sistemas de transmissão de dados, mas também comunicação *wireless* confiável e sistemas de armazenamento de dados como RAID (*Redundant Array of Independent Disks*). Códigos RS também têm sido utilizados durante missões de exploração planetária da NASA (*National Aeronautics and Space Administration*) e ESA (*European Space Agency*) [7]. Os códigos BCH são encontrados especialmente no contexto espacial e são utilizados para correção de dados corrompidos no envio dos mesmos provenientes da estação terrestre para o veículo espacial.

A NASA tem utilizado diferentes códigos corretores de erros em seus projetos. Para as missões entre 1969 e 1977 o satélite *Mariner* [8][9] usou o código Reed-Muller. A interferência que esse satélite estava sujeito era bem aproximada a denominada "bell-curve" (distribuição normal) [10], assim sendo, o código de Reed-Muller foi bem satisfatório para essa situação. Logo em seguida, os veículos espaciais *Voyager 1* e *Voyager 2*, que transmitiam imagens coloridas de Júpiter e Saturno no final da década de 70, mais precisamente em 1979 e 1980 fazia com que a transmissão de imagens coloridas necessitasse o envio de 3 vezes o tamanho de dados, representando as três cores, o código de correção de erros utilizado foi o de Golay (24,12,8) [11][12]. O código de Golay permite correção de apenas 3 erros por *codeword*. O *Voyager 2* enviado para Urano e Netuno, teve seu código alterado para um Código Convolucional concatenado ao Código de Reed-Solomon por possuir maior capacidade de correção de erros.

Os códigos Reed-Solomon e BCH foram escolhidos para o padrão CCSDS (*Consultative Com-*

[2]Erros em rajada são formados por uma seqüência de erros ocorridos um após o outro

mittee for Space Data Systems) [13] devido à sua alta capacidade de detecção e correção de erros. O código RS permite ser configurado de acordo com o número de símbolos de paridade e capacidade de correção.

Em sistemas espaciais, basicamente, uma estação terrestre envia telecomandos (TC) a serem consumidos por módulos computacionais presentes nos veículos espaciais, recebendo telemetria (TM) proveniente dos mesmos [14]. A Figura 1.3 ilustra o processo de envio de telecomando e recebimento de telemetria.

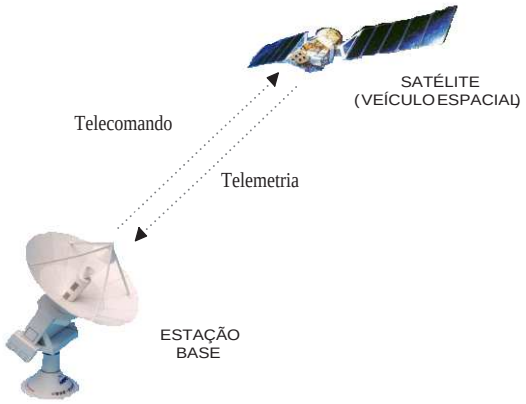

Figura 1.3: Envio de telecomando e recebimento de telemetria em uma base terrestre

Os pacotes de telecomando são compostos de comandos enviados pela estação terrestre para serem executados nos módulos presentes no veículo espacial. Entre eles estão: posicionamento de instrumentos que capturam imagens através de câmera digital, requisição de temperatura externa ao veículo ou até mesmo instruções para navegação do satélite, entre outros. Os dados resultantes do processamento das requisições, bem como dados sobre a "saúde" do veículo espacial, são encapsulados dentro de pacotes de telemetria, os quais são enviados para a estação terrestre. O envio de telemetria pode ser dado de duas formas: a primeira delas se dá através do envio da telemetria em resposta a uma requisição recebida através de telecomando, a segunda forma é o envio constante de informações de telemetria, como por exemplo velocidade do veículo espacial, localização e temperatura.

1.1 Objetivo Principal

Esse trabalho tem por objetivo principal investigar técnicas para o projeto de hardware para codificação de telemetria utilizando o algoritmo de Reed-Solomon e decodificação de telecomando utilizando o algoritmo BCH. Foi dado ênfase apenas na codificação de TM e decodificação de TC, uma vez que essas atividades são realizadas *on-board* e a pesqusia vislumbra aspectos de sistemas embarcados para veículos espaciais. Os pacotes de TC e TM estão baseados no padrão CCSDS, padrão esse reconhecido mundialmente e utilizado pelas principais agências espaciais, entre elas NASA, ESA e Agência Espacial Russa.

1.2 Estratégia para Desenvolvimento do Trabalho

O trabalho foi composto basicamente de seis etapas: a primeira definiu o levantamento bibliográfico e investigação dos algoritmos que melhor se adaptam a projetos de hardware para aplicações espaciais. Nessa etapa foram escolhidos os algoritmos de RS para codificação/decodificação de telemetria e o algoritmo BCH para codificação/decodificação de telecomando. Após a realização do estudo e investigação dos códigos levantados, partiu-se para a segunda etapa, que contemplou a implementação e validação em software do algoritmo RS, que possibilitou a validação da funcionalidade do algoritmo RS. A partir de sua validação, partiu-se para a terceira etapa, a qual contemplou o projeto, implementação e validação do algoritmo de RS em hardware, descrito em linguagem VHDL e sintetizado em um dispositivo FPGA. Na quarta etapa, partiu-se para a implementação em hardware das camadas de empacotamento e transferência propostas pelo padrão CCSDS para TM e TC. Nessa etapa foi realizado um estudo sobre as camadas que compõe o modelo de serviço de telemetria e telecomando. Na quinta etapa foi realizado o projeto, implementação e validação do algoritmo BCH, o qual foi descrito em linguagem VHDL e sintetizado em um dispositivo FPGA da mesma forma como feito com o algoritmo RS. Na última etapa, dados de simulação e síntese foram levantados, analisados e discutidos.

1.3 Organização do Texto

Esse documento está organizado da seguinte forma. O Capítulo 2 apresenta alguns trabalhos realizados no contexo de hardware utilizado para correção de erros. No Capítulo 3 são apresentadas definições, notações e teoremas aplicados a álgebra presente nos algoritmos de correção de erros BCH e Reed-Solomon. O Capítulo 4 apresenta conceitos básicos sobre códigos corretores de erros enquanto o Capítulo 5 descreve os processos de codificação e decodificação do algoritmo BCH, utilizado para correção de erros em pacotes de telecomandos. Da mesma forma, no Capítulo 6 o algoritmo Reed-Solomon é descrito com mais detalhes e os passos de codificação e decodificação são apresentados. Esse algoritmo é utilizado para correção de possíveis erros ocasionados no envio de pacotes de telemetria provenientes do veículo espacial para a estação terrestre. O Capítulo 7 apresenta a fundamentação teórica presente no escopo espacial, em especial sistemas de telecomando e telemetria, apresentando as camadas que compõe o modelo serviço para TC e TM, o qual foi objeto de estudo no presente trabalho. No Capítulo 8 é apresentada a estratégia utilizada para projeto e implementação em software e hardware dos algoritmos BCH e RS. Nele, são apresentadas cada etapa de projeto e passos intermediários que serviram como validação dos algoritmos levantados. O Capítulo 9 apresenta os resultados obtidos no desenvolvimento desse trabalho. Por fim, o Capítulo 10 apresenta as conclusões obtidas e sugestões de trabalhos futuros para a continuação do mesmo.

Capítulo 2

Trabalhos Relacionados

A implementação de códigos RS e BCH em hardware reconfigurável é atraente por várias razões. Primeiramente por permitir descrição de códigos parametrizáveis, na qual a capacidade de correção de erros pode ser alterada facilmente. Em segundo lugar, por permitir rápida prototipação, o que para muitas aplicações comerciais é um dos principais fatores.

Esse capítulo apresenta alguns trabalhos relevantes na área de códigos corretores de erros em hardware, dando ênfase aos algoritmos de Reed-Solomon e BCH, utilizados para atividades de correção de possíveis erros ocasionados no envio de telemetria para a estação terrestre e telecomando para o veículo espacial.

Em [15] é proposta uma implementação de um computador de bordo de um satélite com a lógica programável de um único chip integrando o desenvolvimento de um sistema de comunicação em software para o mesmo. O sistema de comunicação está baseado no padrão CCSDS e a integração de núcleos de propriedade intelectual forma o principal subsistema de um*system-on-a-chip*.

O Centro Espacial de Surrey (SSC[1]) desenvolve pequenos satélites utilizando tecnologias de acordo com o padrão CCSDS, possui entre as suas principais pesquisas, o projeto chamado *ChipSat*, cujo objetivo consiste na aplicação de tecnologias para projeto de satélites de pequeno porte. Como parte do *ChipSat*, um computador de bordo de um satélite de pequeno porte é implementado na forma de um dispositivo SoC. Alguns núcleos de propriedade intelectual escritos em linguagem de descrição de hardware VHDL são usados para construir o sistema do computador de bordo. Os principais blocos do SoC são compostos de: microprocessador (Leon Sparc), unidade de detecção e correção de erros (EDAC[2]) em memória, carregador de *bootstrap*, controlador HDLC (*High Level Data Link Control*), interface CAN (*controller area network*), interface de rede, interface IDE (*Integrated Drive Electronics*), co-processador matemático e interface de barramento de periféricos. Um FPGA Xilinx Virtex é usado para prototipação do SoC.

A Figura 2.1 apresenta o diagrama de blocos do SoC proposto em [15]. O subsistema resultante LEON+CAN+EDAC (indicado em vermelho na Figura 2.1, foi implementado em um FPGA Xilinx Vitex XCV800 usando a placa de prototipação XESS XSV-800. As ferramentas de CAD (*Computer Aided Design*) usadas foram: Simulador VHDL ModelSim, ferramenta de síntese Synplify 6.0+ e ferramentas da *Xilinx Foundation*. versões 2.1i e 3.1i. Como resultado do projeto do SoC, levantou-se dados referentes à ocupação de área em FPGA dos módulos IPs utilizados.

[1] do inglês *Surrey Space Centre*
[2] do inglês *error-detection-and-correction*

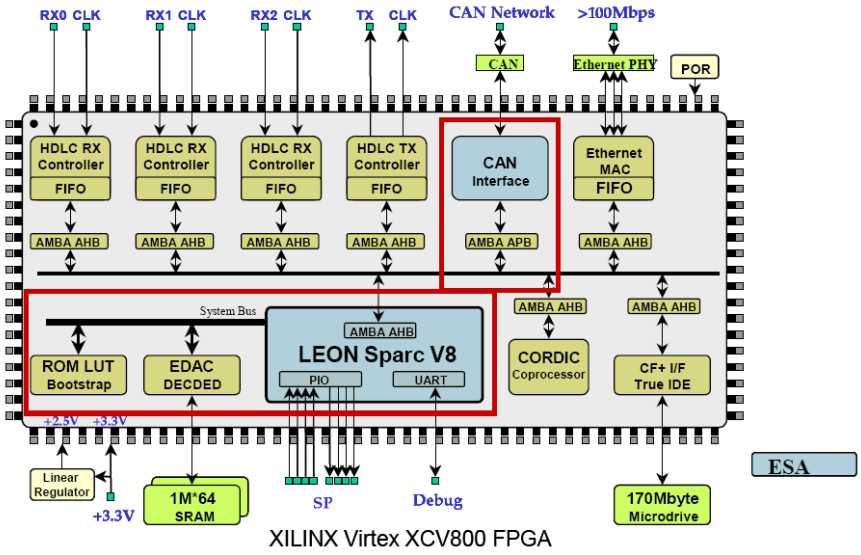

Figura 2.1: Diagrama de blocos do SoC de Surrey

A Tabela 2.1 apresenta os módulos SoC e dados referentes a ocupação em termos de *slices* CLB (*Configurable Logic Block*), BlockRAMs, IOBs (*Input/Output Block*), freqüência (MHz) predita por uma ferramenta de *Place & Route* e freqüência obtida através de simulação.

É possível observar, através da Tabela 2.1, que o subsistema implementado na forma de um SoC (LEON+CAN+EDAC) necessita cerca de 50% da capacidade do FPGA utilizado Virtex XCV800. O arquivo *bitstream* do subsistema implementado possui um tamanho de 576KB. Estimativa de área de um SoC-OBC (sem co-processador) indica que ele ocupa cerca de três quartos (3/4) de um chip [15]. O co-processador matemático de ponto flutuante de 32 bits está baseado no algoritmo CORDIC (*Coordinate Rotation Digital Computer*) e é capaz de processar 17 operações de ponto flutuante, como adição, multiplicação, divisão, raiz quadrada, funções trigonométricas e hiperbólicas assim como *log* e *exp*. O co-processador ocupa aproximadamente

Tabela 2.1: Resultados de implementação do trabalho de Surrey

Módulos SoC	**CLB Slices**	**BlockRAMs**	**IOBs**	**MHz (P & R)**	**MHz (Sim)**
LEON 1-2.2.2	2572 de 9408 (27%)	14/28 (50%)	61 de 166 (36%)	23.226	26.67
LEON 1-2.4.0	3640 de 9408 (38%)	14/28 (50%)	58 de 166 (34%)	24.79	26.67
LEON 2-1.0.2a	3754 de 9408 (39%)	14/28 (50%)	60 de 166 (36%)	24.03	26.67
LEON 1-2.4.0 + CAN + EDAC	4694 de 9408 (49%)	14/28 (50%)	100 de 166 (60%)	25.22	-
LEON 2-1.0.2a + CAN + EDAC	4749 de 9408 (50%)	14/28 (50%)	99 de 166 (59%)	25.186	-

a metade de um chip XCV800. Sendo assim, todo o sistema OBC (*On-Board Computer*) não cabe em uma Virtex XCV800. Entretanto, um FPGA Virtex maior, por exemplo, XCV200E, é capaz de suportar e executar o sistema completo.

É importante observar que o sistema de telemetria e telecomando é executado em um processador embarcado, ou seja, o sistema está implementado totalmente em software do ponto de vista da arquitetura interna. Do ponto de vista externo ao sistema, o que pode ser visto é um hardware FPGA operando as funcionalidades do padrão CCSDS. Dessa forma, é possível dizer que tem-se um hardware dedicado para a execução dessas operações. A principal diferença entre o trabalho proposto em Surrey e o trabalho proposto nesse documento se dá na forma com que os sistemas foram implementados. Na metodologia proposta no trabalho de Surrey, existe um processador embarcado no FPGA que executa operações de telecomando e telemetria em software. Na metodologia adotada no presente trabalho, ambos os sistemas foram projetados e implementados diretamente em hardware, sendo descritos na linguagem VHDL. Assim sendo, no lugar de um processador embarcado, como feito em Surrey, existem circuitos cujo objetivo é realizar tarefas propostas pelo padrão CCSDS, tais como empacotamento, transferência e codificação dos pacotes de telemetria e telecomando a serem processados. Os demais trabalhos apresentados nesse capítulo se referem ao projeto de hardware para o algoritmo de correção de erros Reed-Solomon, o qual é um dos principais focos do presente trabalho.

Em [16] é proposta uma técnica para otimização de códigos RS a fim de obter uma melhoria em relação à tolerância à falhas em memórias. A principal parte no processo de codificação e decodificação do algoritmo RS, conforme será apresentado no Capítulo 6, está baseada em operações envolvendo multiplicações de constantes. Conseqüentemente o esforço para diminuir o *overhead* em relação à área e tempo de execução no circuito como um todo, resulta no uso de multiplicadores. Há dois possíveis métodos para otimização no processo de multiplicação: o primeiro diz respeito à escolha do gerador polinomial mais apropriado para a aplicação, o qual é responsável por gerar todos os padrões de erros possíveis de acordo com a capacidade de correção do algoritmo. O segundo refere-se à escolha das constantes mais adequadas para as multiplicações [16]. O trabalho apresenta a proposta de uma ferramenta para escolher o gerador polinomial mais adequado de acordo com a necessidade do usuário/aplicação.

A ferramenta é capaz de procurar, entre todos os polinômios possíveis, o polinômio mais adequado para tal aplicação, avaliando o circuito em relação ao número de multiplicadores criados em termos de área e desempenho. Por exemplo, para o primeiro polinômio, as constantes multiplicativas são 00101, 01010, 10100 e 01101, para o segundo polinômio as constantes são 11101, 00111, 01110 e 11100. A primeira opção totaliza 11 constantes zeros, a segunda somente 7. Devido ao fato do primeiro polinômio conter mais constantes zeros em relação ao segundo, o mesmo resultará em uma melhor simplificação do circuito multiplicador pois ele eliminará mais portas AND.

O circuito proposto por [16] foi sintetizado sobre o dispositivo FPGA VirtexE 600 da Xilinx e foram levantadas medidas relacionadas à ocupação de área do circuito em termos de número de LUTs, menor unidade lógica presente em um FPGA. A Tabela 2.2 apresenta dados levantados no trabalho proposto em [16], com comparação de desempenho entre geradores de códigos manuais e automáticos. O trabalho avalia os resultados em termos de número de LUTs de 4 entradas e o desempenho é avaliado em termos de atraso e o número de *slices* em um caminho crítico.

Em [18], Mamidi apresenta um novo algoritmo para executar a codificação Reed-Solomon. Nele são propostas quatro novas instruções para a aritmética envolvida sobre Corpos de Galois, principal teoria envolvida sobre códigos corretores de erros, apresentada com mais detalhe no Capítulo 3. O artigo apresenta que ao utilizar o algoritmo proposto, pode-se aumentar em um fator de 12 vezes a velocidade de decodificação do algoritmo de Reed-Solomon comparado com uma implementação puramente em software. O artigo direciona o estudo do algoritmo de Reed-

Tabela 2.2: Comparação de desempenho e área entre geradores de códigos manuais e automáticos

-	NEU03[17]		Manualmente Otimizado		RS-OPGE	
Parâmetro	**Codif.**	**Decodif.**	**Codif.**	**Decodif.**	**Codif.**	**Decodif.**
Número de LUT4s	215	538	140	402	134	392
Comparação com NEU03(%)	NA	NA	-35	-25	-37	-27
Atraso (ns)	7.25	47.6	7	30.5	7	30.5
Número de *slices* em cascata	9	33	8	22	8	22

Solomon ao padrão DVB (*Digital Video Broadcast*), requisito para a transmissão de pacotes MPEG2 (*Moving Picture Experts Group*) [19]. O padrão DVB utiliza o padrão RS$(204, 188)$, com capacidade de correção de 8 erros por pacote ($t = 8$). Para isso, são necessários 16 símbolos de verificação anexados à mensagem a ser codificada, resultando em uma *codeword* de 204 bytes ($n = 204$). Um maior detalhamento do algoritmo RS é apresentado adiante no Capítulo 6. No artigo, um novo algoritmo para codificação Reed-Solomon em arquiteturas SIMD (*Single Instruction Multiple Data*) é apresentado. Esse algoritmo não necessita de divisão de polinômio sobre Corpos de Galois para cálculo de símbolos de paridade.

Em [20], Atieno apresenta um decodificador de erros Reed-Solomon adaptado. A principal inovação do algoritmo é o projeto de um sistema de decodificação do canal o qual contém múltiplos componentes decodificadores. Para que a recuperação de dados seja executada de forma correta, somente um dos pares gerador/decodificador deve produzir um resultado correto. Os parâmetros de entrada do decodificador foram gerados via simulação. A chave para a melhoria do desempenho é o uso de reconfiguração dinâmica baseada em amostragem periódica de condições de ruído do canal. Através de experimentos, foi apresentado que 14% de melhoria no desempenho pode ser atingido através do uso de reconfiguração do decodificador em tempo de execução em comparação a implementação estática de maior complexidade, maior poder de decodificação. O decodificador proposto no artigo [20] foi validado em hardware utilizando a placa de desenvolvimento NIOS da Altera, com um FPGA Stratix.

Embora o processo de codificação Reed-Solomon adaptável seja bastante explorado [21], implementações parametrizáveis em hardware são limitadas. Em Lee [8] um sistema de decodificação que fornece múltiplos decodificadores para múltiplos canais é descrito. Desde que a célula básica MEA (*Modified Euclid Algorithm*) para o projeto seja replicada, os blocos MEA consomem cerca de 80% de área do decodificador. Em uma implementação de canal único, específico para DVD (*Digital Video Disc*), múltiplos blocos MEA são usados para melhorar a decodificação utilizando um dispositivo Altera Flex10K200. O trabalho proposto em [20] utiliza uma única célula de processamento MEA a qual é usada de forma recursiva. Uma técnica de decodificação RS adaptável é apresentada em [22] e permite variação nos parâmetros de n e k em tempo de execução, desde que reconfiguração dinâmica não esteja sendo usada.

Para decodificação Reed-Solomon, várias implementações em hardware encontram-se disponíveis [23]. A maioria das implementações são baseadas no uso de LFSR (*Linear Feedback Shift Register*), o qual fornece um meio eficiente para executar o cálculo sobre divisão polinomial [24]. Um resumo de implementações em software e hardware para codificação Reed-Solomon é apresentado em [25]. Os autores de [26] introduzem uma proposta de hardware-software *codesign*

para processamento de codificação e decodificação Reed-Solomon. Eles propõe duas novas instruções para processadores de propósito geral que possibilitam codificação e decodificação RS com baixo consumo de potência.

Por fim, muitos trabalhos têm sido propostos direcionados para diversos tipos de aplicações utilizando os algoritmos de Reed-Solomon e BCH. Esse capítulo apresentou alguns desses trabalhos objetivando fazer um levantamento dos diferentes meios de aplicação desses algoritmos. O Capítulo 3 apresenta os conceitos e definições matemáticas referentes aos elementos e operações envolvidos nos processos de codificação e decodificação dos algoritmos Reed-Solomon e BCH.

Capítulo 3

Álgebra Abstrata

3.1 Histórico

Desde as antigas civilizações, como a egípcia ou a persa, há referências a métodos de cálculo e fórmulas polinomiais. Os hindus, em 600 AC (Antes de Cristo) já sabiam resolver equações quadráticas, e os babilônios possuíam alguma maquinaria de manipulação algébrica que usavam casos especiais da fórmula quadrática [27].

A álgebra simbólica, surgiu com os árabes, entre 600 e 1000 DC (Depois de Cristo). Aqui surge já a fórmula cúbica, e destaca-se o trabalho de Al-Quarizimi (séc.IX, nome que deu origem à palavra algarismo). No séc. XVII a geometria analítica já estava bem compreendida, e já se usava a álgebra para resolver problemas geométricos, e vice-versa.

O progresso continuou com vários matemáticos notáveis nos séculos seguintes, nomeadamente como Euler e Lagrange (séc. XVIII), que se preocuparam, entre outras coisas, em encontrar fórmulas resolventes para polinômios de grau cinco, e Gauss (séc. XIX), a quem se deve aquela que é por vezes considerada a primeira demonstração do teorema fundamental da álgebra, ou teorema de d'Alembert (ainda que essa demonstração não satisfaça os critérios contemporâneos de clareza) [27]. No mesmo século, Abel e Galois, usando a teoria de grupos, provaram a inexistência de fórmulas resolventes gerais com radicais para graus maiores que quatro. Foi surgindo também interesse nos polinômios enquanto fórmulas que descrevem superfícies e curvas, dando origem ao que se chama hoje geometria algébrica.

No início do século XX, surgiram as estruturas algébricas abstratas, os teoremas não construtivos de Hilbert e os desenvolvimentos da geometria algébrica, que constituiram um novo olhar sobre os polinômios.

Ainda hoje os polinômios são tema de investigação, tanto no campo computacional como no campo teórico. O último teorema de Fermat, por exemplo, formulado em 1641, e demonstrado finalmente em 1994 por Andrew Wiles e Richard Taylor, trata da resolução de uma equação diofantina polinomial. As bases de Gröbner, importantes em termos computacionais na geometria algébrica, foram também devidamente estudadas no fim do século passado.

Esse capítulo tem por objetivo apresentar os conceitos matemáticos dentro do escopo de álgebra abstrata. A finalidade não é de esgotar o assunto, mas sim, de apresentar conceitos, definições e teoremas para melhor entendimento do trabalho aqui proposto.

Na Seção 3.2 são apresentados os conceitos relacionados às estruturas algébricas, entre elas teoria de conjunto, grupo, anel e corpo. A Seção 3.3 introduz a teoria de Corpo de Galois, apresenta o processo de construção e ordem de um corpo. Em continuação, a Seção 3.4 apresenta as propriedades no escopo de Corpos de Galois, entre elas estão ordem do corpo, ordem de um

elemento, elementos primitivos e características de Corpos de Galois. Por fim, na Seção 3.5 são apresentadas as propriedades fundamentais de polinômios e suas raízes como irredutibilidade, polinômio primitivo, raízes de um polinômio primitivo e exemplo de construção para o Corpo de Galois GF(2^3).

3.2 Estruturas Algébricas

As estruturas algébricas formam um ramo da álgebra moderna que estende o conceito da teoria dos conjuntos, efetuando uma análise sobre os elementos constituintes de uma dada estrutura e respectivas operações entre si.

Nessa seção introduzem-se os conceitos relacionados com as estruturas algébricas, de uma forma progressiva, salientando os pontos mais importantes que estão relacionados com a construção dos códigos utilizados na codificação de canal. A análise é efetuada partindo da estrutura mais simples denominada por grupóide até chegar ao corpo e posteriormente ao objeto de estudo, o Corpo de Galois.

3.2.1 Conjunto e Grupóide

Um conjunto pode ser definido como um agrupamento de elementos sem operações definidas entre si [28][29]. O número de elementos que pertencem a um dado conjunto determina a sua dimensão, eventualmente infinita, o que define uma característica muito importante designada por cardinalidade.

Se num dado conjunto E for definida uma operação binária '.' tal que a sua aplicação a dois quaisquer elementos de E, resulte num terceiro elemento pertencente à E (não necessariamente diferente), então esta operação designa-se por lei de composição interna em$E$, e tem-se a estrutura algébrica mais simples, o grupóide.

$$(E,.) \text{ é grupóide} <=> \begin{cases} \text{A operação '.' é uma lei de composição interna em } E \\ \text{O conjunto } E \text{ é um espaço fechado} \end{cases}$$

O grupóide resume-se a um espaço fechado com uma operação entre dois dos seus elementos, designados por operandos. O resultado da aplicação da operação resulta em um terceiro elemento pertencente ao grupóide, e que pode coincidir com um dos operandos.

3.2.2 Semigrupo, Grupo, Sub-grupo e Respectiva Ordem

Se a operação '.', além de ser uma lei de composição interna em$E$, possuir ainda a propriedade associativa, então a estrutura algébrica adquire uma nova propriedade e passa a designar-se por semigrupo.

$$(E,.) \text{ é semigrupo} <=> \begin{cases} \text{(E,.) é grupóide} \\ (a.b).c = a.(b.c), \forall a, \forall b, \forall c \in E \end{cases}$$

Se os elementos do semigrupo E apresentarem as seguintes propriedades:

1. Existência de elemento identidade (neutro):

   $\exists\, e \in E : a.e = e.a \;\forall\; a \in E$

2. Existência de elemento inverso para cada elemento de E:
   $a.a^{-1} = a^{-1}.a = e, \forall a \in E, \forall a^{-1} \in E, a^{-1}$ único

   Então pode-se dizer que existe uma estrutura algébrica mais elaborada, denominada por grupo. No caso da operação '.' ser também comutativa, tem-se um grupo comutativo ou abeliano.

3. A operação '.' é comutativa quando se verifica a seguinte condição:
   $a.b = b.a, \forall a, \forall b \in E$

Em resumo, a definição do grupo é a seguinte:

$$(E,.) \text{ é grupo} <=> \begin{cases} \text{(E,.) é semigrupo} \\ \text{Existência de elemento identidade em E} \\ \text{Existência de elemento inverso único para cada elemento de E} \end{cases}$$

Os grupos podem ter ordem (por ex. cardinalidade) infinita. A ordem de um grupo G denomina-se por $ord(G)$. No entanto para os objetivos do presente estudo apenas se consideram estruturas algébricas de ordem finita. Para simplificar a análise de um determinado grupo este pode ser subdividido em vários sub-grupos com menor cardinalidade que o grupo que lhes deu origem. Um conjunto G é subgrupo de E se respeitar a seguinte relação:

$$(G,.) \text{ é um subgrupo de (E,.)} \begin{cases} \text{(G,.) é um grupo} \\ G \in E \end{cases}$$

O Teorema de Lagrange enuncia uma propriedade importante que relaciona a ordem do grupo G com a ordem de um dos possíveis subgrupos denominados por S.

Teorema A. *Sendo S um subgrupo de G então $ord(G)$ é múltipla de $ord(S)$*

3.2.3 Ordem de um Elemento e suas Propriedades

No ponto anterior foi apresentado que a ordem de um grupo define-se como a cardinalidade do grupo. No entanto, associado a cada elemento g do grupo G também existe uma ordem, denominada por $ord(g)$, que se define como o menor número inteiro a que se tem de elevar q para obter o elemento identidade do grupo [27][29], designado por e. De uma forma mais formal tem-se :

$$g^{ord(g)} = e, \forall g \in G \tag{3.1}$$

Tendo como exemplo um grupo de ordem 4, com a operação '.' definida como a multiplicação módulo 5, obtêm-se os resultados apresentados na Tabela 3.1, para as várias aplicações da operação:

Tabela 3.1: Grupo de ordem 4 com multiplicação módulo 5

.	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Como se pode verificar o elemento identidade da operação é o número 1. Cada um dos elementos tem um inverso único. A ordem dos elementos é calculada a partir da definição, e os resultados são apresentados na Tabela 3.2.

Tabela 3.2: Ordem dos elementos do grupo de ordem 4 com multiplicação módulo 5

g	ord(g)
1	1
2	4
3	4
4	1

Existe uma propriedade importante que relaciona a ordem de um elemento com a ordem do grupo, e que se pode inferir pela Tabela 3.2. Enuncia-se a forma seguinte: *"A ordem de um elemento é sub-múltipla da ordem do grupo."*

O exemplo anterior serve também para mostrar um Teorema que envolve os grupos e os números primos. O enunciado do Teorema é o seguinte:

Teorema B. *Os elementos* $S = \{1, 2, 3, \ldots, p-1\}$ *formam um grupo comutativo de ordem* $p-1$ *para a multiplicação módulo* p *se e somente se* p *for um número primo.*

3.2.4 Anel

Na base da formação da estrutura grupo está a aplicação de uma operação binária sobre um dado conjunto de elementos que obedecem a um dado conjunto de regras.

É possível agregar a um grupo mais uma operação binária denominada por '+' de forma que se estabeleça um conjunto de relações que obedecem a certas regras de forma a obter a estrutura algébrica Anel.

$$(E,.) \text{ é anel } <=> \begin{cases} \text{(E,+) é grupo comutativo} \\ \text{(E,.) é semigrupo} \\ \text{'.' distribui sobre '+' : } a.(b+c) = a.b + a.c, \forall a, \forall b, \forall c \in E \end{cases}$$

Esta é a definição base de anel. No entanto, esta estrutura ainda pode ter variantes. No caso da operação '.' ser comutativa, tem-se um anel comutativo. Se a operação '.' tiver um elemento identidade, então tem-se um anel com identidade. Na situação em que se verificam ambos os casos anteriores, tem-se um anel comutativo com identidade [29].

3.2.5 Corpo

Partindo da definição anterior sobre a constituição de um Anel, quando verificadas determinadas condições, obtém-se uma nova estrutura algébrica denominada por Corpo.

$$(E,.) \text{ é corpo } <=> \begin{cases} \text{(E,+) é grupo comutativo} \\ \text{(E - \{0\},.) é grupo comutativo} \\ \text{'.' distribui sobre '+' : } a.(b+c) = a.b + a.c, \forall a, \forall b, \forall c \in E \end{cases}$$

A definição de Corpo é por isso semelhante à definição de Anel, apresentada anteriormente e pode ser escrita da seguinte forma:

"Um corpo é um anel comutativo com identidade, no qual cada elemento tem um inverso multiplicativo."

Ou ainda de outra forma mais simples:

"Um corpo é constituído por dois grupos. Todos os elementos formam um grupo comutativo aditivo. Os elementos diferentes de {0} formam um grupo comutativo multiplicativo."

Torna-se necessário dar ênfase nesta definição pois é sobre a estrutura algébrica Corpo que o presente estudo vai ser direcionado a partir dessa seção.

3.3 Corpo de Galois

Um Corpo de Galois define-se como sendo um corpo de ordem finita. Significa que tem uma cardinalidade perfeitamente conhecida, a qual o caracteriza completamente [30]. Genericamente um Corpo de Galois de ordem p é representado por $GF(p)$.

3.3.1 Construção

Para obter um Corpo de Galois de ordem p, inteiro primo, consideram-se todos os inteiros positivos $S = \{0, 1, 2, \ldots, p-1\}$. Desta forma respeitam-se as duas condições de construção de um grupo devido às seguintes propriedades:

- O conjunto de inteiros $\{0, 1, 2, \ldots, p-1\}$, forma um grupo aditivo comutativo para a soma módulo p.
- O conjunto de inteiros $\{0, 1, 2, \ldots, p-1\}$, com p sendo um número primo positivo, forma um grupo multiplicativo comutativo para a multiplicação módulo$p$.

A forma mais simples que se pode ter, consiste no Corpo de Galois de ordem 2, representado por $GF(2)$. É possível obter Corpos de ordem superior q, gerados a partir da ordem p [30], tal que se verifiquem as seguintes condições:

- p é um número primo;
- $m > 1$;
- $q = p^m$;

3.3.2 Ordem de um Corpo

A partir dos dados do ponto anterior é possível concluir que os Corpos de Galois de ordem prima p com valores baixos são fáceis de construir. O seguinte Teorema enuncia a construção de um Corpo de Galois de ordem p:

Teorema C. *Os inteiros positivos $S = \{0, 1, 2, \ldots, p-1\}$, sendo p um número primo, constituem o $GF(p)$ para adição e multiplicação módulo p.*

Um Corpo de Galois de ordem p^m pode ser obtido como um espaço vetorial sobre um outro corpo de ordem p. Portanto os corpos de ordem p servem de base geradora para a construção de outros corpos de ordem superior.

3.4 Propriedades dos Corpos de Galois

Nesta seção é realizada uma análise detalhada das principais características de um Corpo de Galois e dos respectivos elementos que o constitui. Inicialmente são analisadas as propriedades dos corpos de uma forma geral, em seguida passa-se para a análise dos elementos que o constituem, descrevendo um conjunto de propriedades úteis, que servem de fundamento à aplicação de polinômios sobre o corpo e à fatorização do polinômio $x^n + 1$, que representa a chave para a criação de um código cíclico.

Um código cíclico é aquele que, a partir de um vetor U pertencente ao subespaço vetorial do código cíclico, é possível gerar todos os demais códigos através do deslocamento sucessivo de U. O vetor U pode ser descrito como: $U = (\mu_0\ \mu_1\ \mu_2\ ...\ \mu_{n-1})$.

3.4.1 Ordem do Corpo

Sendo um corpo de ordem finita e com todas as suas propriedades bem conhecidas, é possível afirmar que um corpo é completamente caracterizado pela sua ordem. A notação $GF(p)$ representa um Corpo de Galois de ordem p. No caso de uma notação $GF(p)[x]$, significa que tem-se uma aplicação de polinômios sobre os Corpos de Galois, e que os coeficientes dos polinômios são constituídos de valores entre 0 e $p-1$. O grau dos polinômios designa-se por n e é independente dos coeficientes dos mesmos.

Exemplo:

- Corpos de Galois de ordem 2: GF(2) = $\{0, 1\}$;
- Corpos de Galois de ordem 3: GF(3) = $\{0, 1, 2\}$;

3.4.2 Ordem de um Elemento

Existe semelhança entre a ordem de um elemento e a ordem de um Grupo, também existe o conceito de ordem de um elemento quando se trata de um Corpo, sabendo que um corpo é constituído por duas estruturas algébricas do tipo Grupo. A ordem de um elemento é definida como o menor número inteiro positivo tal que esse elemento precisa de um operador, usando a operação de caráter multiplicativo do corpo, para obter o seu elemento identidade designado por '1'. De uma forma mais formal tem-se:

Seja $\beta \in GF(p)$. A ordem de β, representada por $ord(\beta)$, é o menor inteiro positivo m tal que $\beta^m = 1$.

3.5 Propriedades dos Polinômios e suas Raízes

A notação $GF[q](x)$ é utilizada para indicar um Corpo de Galois que tem ordem q, sobre o qual são aplicados polinômios de grau n com coeficientes com valores compreendidos entre 0 e $q-1$. Um exemplo de um polinômio $p(x)$ pode ser apresentado na Equação 3.2.

$$p(x) = a_5.x^5 + a_4.x^4 + a_x + 1 \tag{3.2}$$

Os coeficientes de um polinômio a_i pertencem obrigatoriamente ao $GF(p)$, ao passo que o grau do polinômio n, pode ter um valor qualquer.

3.5.1 Irredutibilidade

Um polinômio designa-se por irredutível quando não é possível efetuar a sua fatorização em pelo menos dois polinômios de grau inferior, usando coeficientes até uma dada ordem.

Exemplo:
Dado o $GF[2](x)$, e $p(x) = x^2 + x + 1$ um polinômio cujos coeficientes estão compreendidos entre 0 e 1, inclusive. Este polinômio é irredutível em $GF(2)$. Mas no caso de considerar $GF(4)$ (coeficientes a_i tomam valores entre 0 e 3), logo o polinômio deixa de ser irredutível.

3.5.2 Polinômio Primitivo

Um polinômio $p(x)$ de grau m é primitivo em $GF(p)[x]$ se for irredutível e atender às seguintes condições apresentadas nas Equações 3.3 e 3.4:

$$rem\left[\frac{x^n - 1}{p(x)}\right] = 0; \tag{3.3}$$

$$n = p^m - 1; \tag{3.4}$$

Para ser um polinômio primitivo, $p(x)$ deve ser irredutível. No entanto nem todos os polinômios irredutíveis são primitivos. O caso contrário verifica-se sempre, todos os polinômios primitivos são irredutíveis.

3.5.3 Raízes de um Polinômio Primitivo

As raízes de um polinômio primitivo possuem propriedades bastantes interessantes no estudo dos Corpos de Galois, e que servem posteriormente para a geração de Corpos de$q$ a partir de subcorpos de ordem p. Cada Corpo de Galois de ordem q, possui um subcorpo de ordem prima p.

"As raízes do polinômio primitivo, de grau m, $p(x) \in GF(p)[x]$, são elementos primitivos em $GF(q)$, com $q = p^m$."

Isto significa que um $GF(q)$, pode ser construído através de um subcorpo de ordem prima $GF(p)$, e de seu polinômio primitivo, ou seja, $GF(q)$ é um espaço vetorial sobre $GF(p)$. Quando $GF(p)$ tem ordem prima, $GF(q)$ pode designar-se como a sua extensão. Assim como os grupos contêm subgrupos, os Corpos de Galois também contêm subcorpos, para além do subcorpo base de ordem prima.

3.5.4 Exemplo de Construção para $GF(8)$

$GF(8)$ pode ser visto como espaço vetorial sobre $GF(2)$. O polinômio $p(x) = x^3 + x + 1$ é primitivo em $GF(2)$. As suas raízes são elementos primitivos de $GF(q)$ com $q = p^m = 2^3 = 8$. Assim como apresentado nas propriedades dos elementos primitivos, prova-se que todos os elementos de $GF(q)$ podem ser obtidos através de $(q-1)$ produtos dos elementos primitivos.

Seja α uma raiz do polinômio primitivo $p(x)$, é possível obter uma representação vetorial de $GF(8)$ tal como apresentado na Tabela 3.3, através do conjunto base $\{1, \alpha, \alpha^2\}$.

Ao conseguir efetuar a construção de$GF(8)$ desta forma, ficando com a representação vetorial, as operações efetuadas sobre este Corpo ficam bastante facilitadas na medida que se pretende

Tabela 3.3: Representação de $GF(8)$ no formato vetorial

0	(0,0,0)
α	(0,1,0)
α^2	(0,0,1)
$\alpha^3 = \alpha + 1$	(1,1,0)
$\alpha^4 = \alpha^2 + \alpha$	(0,1,1)
$\alpha^5 = \alpha^2 + \alpha + 1$	(1,1,1)
$\alpha^6 = \alpha^2 + 1$	(1,0,1)
$\alpha^7 = 1$	(1,0,0)

efetuar adição em $GF(8)$, basta efetuar adição de vetores sobre $GF(2)$. Este é um aspecto importante a levar em conta quando se pretende efetuar a implementação de um $GF(q)$, com $q = p^m$, tanto em hardware quanto em software, pois basta ter os vetores da base e efetuar combinações entre eles para obter um determinado elemento. Por exemplo, para obter o elemento α^5 tem-se:

$$\alpha^5 = \alpha^2 + \alpha + 1 \tag{3.5}$$

A soma dos vetores correpondentes é dada por:

$$(0, 0, 1) + (0, 1, 0) + (1, 0, 0) = (1, 1, 1). \tag{3.6}$$

Dessa forma fica apresentada a importância das propriedades dos polinômios primitivos e das suas raízes na construção de Corpos de Galois. Esse capítulo apresentou alguns conceitos importantes para a formação do embasamento matemático necessário para o entendimento dos algoritmos BCH e RS apresentados nos Capítulos 5 e 6 respectivamente. O Capítulo 4 introduz o conceito de códigos corretores de erros apresentando definições importantes como por exemplo *Distância de Hamming*, *Código de Bloco Linear* e *Códigos Convolucionais*.

Capítulo 4

Códigos Corretores de Erros

Por definição, códigos corretores de erros são códigos que possuem a capacidade de detecção e correção de erros ocorridos sobre um determinado conjunto de dados. Geralmente esses dados são transmitidos através de um canal ruidoso, propenso a erros. Chama-se codificação do canal a codificação de sinais de informação com o objetivo de diminuir a taxa de erro de símbolo e/ou de bit durante a transmissão dos mesmos através de um canal de comunicação.

Geralmente os códigos corretores de erros estão baseados no mesmo princípio: dados de redundância são adicionados à informação a fim de corrigir possíveis erros que podem ocorrer nos processos de armazenamento e transmissão de dados. Na forma mais usual, símbolos de redundância são anexados aos símbolos de informação para obter uma seqüência de código denominada de *codeword*. Para fins de ilustração, uma *codeword* obtida através da codificação utilizando um código de bloco é mostrada na Figura 4.1.

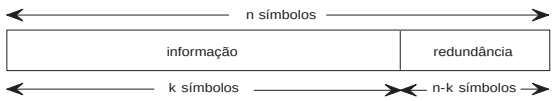

Figura 4.1: Codificação sistemática para correção de erros

4.1 Códigos de Bloco e Códigos Convolucionais

De acordo com o modo com que a redundância é adicionada às mensagens, os códigos corretores de erros podem ser divididos em duas classes: código de bloco e código convolucional. Ambos os tipos de esquema de codificação são encontrados em aplicações práticas. Códigos de blocos, basicamente, processam a informação bloco por bloco, tratando cada bloco de bits de informação de forma independente com relação a outro bloco. Ao contrário de códigos de blocos, a saída de um codificador convolucional depende não somente da informação de entrada, mas também das entradas e saídas anteriores da forma bloco a bloco ou bit a bit. Códigos convolucionais são usados geralmente para melhorar o desempenho da comunicação via rádio e satélites.

4.2 Distância de *Hamming*, Campo de *Hamming* e Capacidade de Correção de Erros

Considere um código corretor de erros C com elementos binários. Objetivando explorar as capacidades de correção de erros, nem todos os 2^n vetores binários possíveis de largura n são permitidos para serem transmitidos. C é um subconjunto de vetores binários de dimensão n pertencente ao espaço $V_2 = 0, 1^n$, onde os elementos estão o mais distante possível um do outro.

Em um espaço vetorial V_2, a distância é definida como o número de inteiros diferentes entre dois vetores. Sendo $\overline{x}_1 = (x_{1,0}, x_{1,1}, \ldots, x_{1,n-1})$ e $\overline{x}_2 = (x_{2,0}, x_{2,1}, \ldots, x_{2,n-1})$ dois vetores pertencentes a V_2. A distância de *Hamming* entre $\overline{x}_1$ e $\overline{x}_2$, denotada por $d_H = (\overline{x}_1, \overline{x}_2)$ é definida pela Equação 4.1.

$$d_H(\overline{x}_1, \overline{x}_2) = |i : x_{1,i} \neq x_{2,i}, 0 \leq i < n|, \tag{4.1}$$

onde $|A|$ corresponde ao número de elementos (ou a cardinalidade) de um conjunto A.

Dado um código C, a distância mínima de *Hamming*, d_{min}, é definida como a distância mínima entre todos os pares distintos de *codewords* em C, representado pela Equação 4.2.

$$d_{min} = \min_{\overline{v}_1, \overline{v}_2 \in C} \{d_H(\overline{v}_1, \overline{v}_2) | \overline{v}_1 \neq \overline{v}_2\}. \tag{4.2}$$

Para fins de notação, no decorrer do capítulo são adotados os seguintes símbolos: (n, k, d_{min}) é usado para descrever os parâmetros de um código de blocos de comprimento n, o qual codifica mensagens de tamanho k e possui uma distância mínima de *Hamming* d_{min}. O tamanho do código é dado por $|C| = 2^k$.

Exemplo 1 O mais simples código corretor de erros é composto de repetições binárias de comprimento 3. Ele repete cada bit três vezes, sendo '0' codificado em um vetor de (000) e '1' em um vetor de (111). Sendo que esses vetores são diferentes em três posições, a distância mínima de *Hamming* é igual a três. A Figura 4.2 é uma representação vetorial desse código. O espaço vetorial corresponde a um conjunto de $2^3 = 8$ vértices de um cubo tridimensional. A distância de *Hamming* entre as *codewords* (000) e (111) é igual ao número de bordas dispostas entre eles. Isso equivale ao número de coordenadas necessárias para converter (000) para (111), ou vice-versa. Então $d_H((000), (111)) = 3$. Como existem somente duas *codewords*, $d_{min} = 3$.

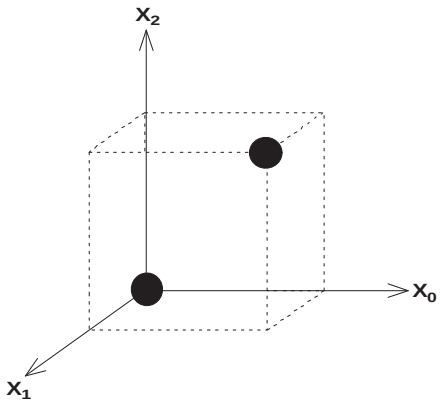

Figura 4.2: Um código de repetição (3,1,3) em um espaço vetorial binário tridimensional

O espaço vetorial binário V_2 é também conhecido como *espaço de Hamming*. Sendo $\overline{v}$ uma *codeword* de um código corretor de erros C. Uma *esfera de Hamming* $S_t(\overline{v})$, de raio t e centralizada em $\overline{v}$, é composta de um conjunto de vetores em V_2 que possui distância menor ou igual a t a partir do centro de $\overline{v}$.

$$S_t(\overline{v}) = \{\overline{x} \in V_2 | d_H(\overline{x}, \overline{v}) \leq t\} \tag{4.3}$$

O tamanho de um (ou o número de *codewords* em) $S_t(\overline{v})$ é dado pela Equação 4.4.

$$|S_t(\overline{v})| = \sum_{i=0}^{t} \binom{n}{i} \tag{4.4}$$

Exemplo 2 A Figura 4.3 apresenta a *esfera de Hamming* de raio $t = 1$ sobre *codewords* de código de repetição binária (3,1,3).

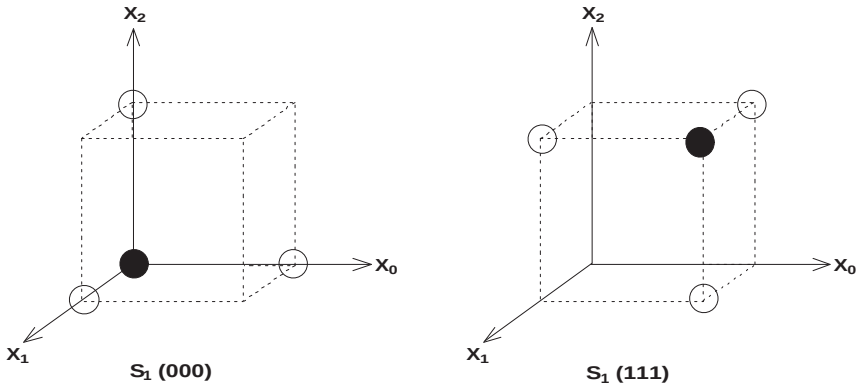

Figura 4.3: *Esfera de Hamming* de raio $t = 1$ sobre *codewords* de código de repetição binária (3,1,3)

É possível observar que as *esferas de Hamming* são disjuntas, ou seja, não existe vetor em V_2 pertencente a ambos $S_1(000)$ e $S_1(111)$. Como resultado, se houver uma troca em uma posição qualquer de uma *codeword* $\overline{v}$, o vetor resultante ainda se encontrará dentro da *esfera de Hamming* centralizada em $\overline{v}$. Esse conceito é a base para o entendimento e definição da capacidade de correção de erros de um código C.

A capacidade de correção de erros (t) de um código C é dada pelo maior raio da *esfera de Hamming* $S_t(\overline{v})$ sobre todas as *codewords* $\overline{v} \in C$, no qual, para todos os diferentes pares pertencentes a $\overline{v}_i$, $\overline{v}_j \in C$, as *esferas de Hamming* correspondentes são disjuntas, por ex.,

$$t = \max_{\overline{v}_i, \overline{v}_j \in C} \{\ell | S_\ell(\overline{v}_i) \cap S_\ell(\overline{v}_j) = \emptyset,\ \overline{v}_i \neq \overline{v}_j\} \tag{4.5}$$

Em termos de distância mínima de C, d_{min}, uma definição equivalente e mais utilizada é dada na Equação 4.6.

$$t = \lfloor (d_{min} - 1)/2 \rfloor, \tag{4.6}$$

onde $\lfloor x \rfloor$ denota o maior inteiro menor ou igual a x.

Para calcular a distância mínima d_{min} de um código C, de acordo com a Equação 4.2, um total de (no máximo) $2^k(2^k - 1)$ distâncias entre pares distintos de *codewords* são necessárias. Isso é

praticamente impossível para códigos de tamanho maiores, como$k = 50$. Uma das vantagens de códigos de blocos lineares é que o cálculo ded_{min} necessita saber apenas o tamanho de*Hamming* de todas $2^k - 1$ *codewords* diferentes de zero.

4.3 Códigos de Bloco Linear

Como mencionado na seção anterior, encontrar um bom código significa encontrar um subconjunto deV_2 com elementos o mais distante possível um do outro, o que é muito difícil. Além disso, sempre que um conjunto é encontrado, existe ainda o problema de como atribuir as*codewords* para mensagens de informação.

Códigos Lineares são subespaços vetoriais deV_2. Isso significa que a codificação pode ser realizada através de multiplicação de matrizes. Na área de circuitos digitais, simples codificadores podem ser construídos usando operações de XOR's, AND's e flip-flops do tipo D. Nessa seção, as operações de soma e multiplicação no espaço vetorial binário estão representadas pela saída de portas lógicas XOR (adição módulo 2) e AND, respectivamente. As Tabelas de adição e multiplicação para elementos binários são apresentadas pela Tabela 4.1.

Tabela 4.1: Representação das operações de soma e multiplicação módulo 2

a	b	a+b	a.b
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

4.3.1 Matrizes Geradora e de Verificação de Paridade

Seja C um código linear binário (n, k, d_{min}). Sendo C um subespaço vetorial k-dimensional, existe uma base $\{\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_{k-1}\}$, no qual qualquer *codeword* $\overline{v} \in C$ pode ser representada como uma combinação linear de elementos na base:

$$\overline{v} = u_0\overline{v}_0 + u_1\overline{v}_1 + \ldots + u_{k-1}\overline{v}_{k-1}, \tag{4.7}$$

onde $u_i \in \{0, 1\}, 1 \leq i < k$. A Equação 4.7 pode ser escrita em termos de matriz geradora$G$ e um vetor de mensagem, $\overline{u} = (u_0, u_1, \ldots, u_{k-1})$ como segue:

$$\overline{v} = \overline{u}G, \tag{4.8}$$

onde

$$G = \begin{pmatrix} \overline{v}_0 \\ \overline{v}_1 \\ \vdots \\ \overline{v}_{k-1} \end{pmatrix} = \begin{pmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,n-1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k-1,0} & v_{k-1,1} & \cdots & v_{k-1,n-1} \end{pmatrix} \tag{4.9}$$

Seja C um espaço vetorial em V_2 de k dimensões, existe um espaço dual C^T de $(n-k)$ dimensões, gerado pelas linhas de uma matriz H, denominada de *matriz de verificação de paridade*, apresentada na Equação 4.10, onde H^T representa a matriz transposta de H. Além disso, é possível observar que para qualquer *codeword* $\overline{v} \in C$,

$$GH^T = 0 \tag{4.10}$$

$$\overline{v}H^T = \overline{0} \tag{4.11}$$

A Equação 4.11 é de fundamental importância na decodificação de códigos lineares. Um código linear $C^{\perp}$ gerado por H é um código linear binário $(n, n-k, d^{\perp}_{min})$, chamado de código dual de C.

4.4 Codificação e Decodificação de Códigos de Bloco Linear

4.4.1 Codificando com G e H

A Equação 4.8 apresenta uma regra de codificação para códigos de bloco linear que pode ser implementada de forma direta. Se a codificação for sistemática, então o gerador da matriz$G$ de um código de bloco linear C (n, k, d_{min}) pode ser apresentada por G_{sys}, através de operações AND/OR e permutação de linhas e colunas. G_{sys} é composta de duas sub-matrizes: A matriz identidade (k, k), denotada por I_k, e a sub-matriz de paridade P de dimensão $(k, n-k)$, na qual

$$G_{sys} = (I_k|P), \tag{4.12}$$

onde

$$P = \begin{pmatrix} P_{0,0} & P_{0,1} & \cdots & P_{0,n-k-1} \\ P_{1,0} & P_{1,1} & \cdots & P_{1,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ P_{k-1,0} & P_{k-1,1} & \cdots & P_{k-1,n-k-1} \end{pmatrix} \tag{4.13}$$

Sendo $GH^T = 0$, na forma sistemática H_{sys} da matriz de paridade é

$$H_{sys} = (P^T|I_{n-k}) \tag{4.14}$$

Exemplo 3 Considere um código de bloco linear $(4, 2, 2)$ com a seguinte matriz geradora:

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Para transformar G para a forma sistemática, troca-se a segunda e a quarta coluna e obtém-se:

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Então a sub-matriz de paridade é dada por:

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

É possível notar que nesse caso a relação $P = P^T$ é referenciada como seu próprio código dual. A partir da Equação 4.14 tem-se a matriz de paridades na forma sistemática apresentada como segue:

$$H_{sys} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{4.15}$$

O vetor $\overline{u} = (u_0, u_1, \ldots, u_{k-1})$ representa a mensagem a ser codificada enquanto o vetor $\overline{v} = (v_0, v_1, \ldots, v_{n-1})$ representa a *codeword* correspondente em C.

Se os parâmetros de C tal que $k < (n-k)$, ou equivalentemente a taxa do código $k/n < 1/2$, a codificação com a matriz geradora é mais econômica. O custo considerado aqui é em termos de número de operações binárias. Nesse caso:

$$\overline{v} = \overline{u}G_{sys} = (\overline{u}, \overline{v}_p), \tag{4.16}$$

onde $\overline{v}_p = \overline{u}P = (v_k, v_{k+1}, \ldots, v_{n-1})$ representa a parte de verificação de paridade da *codeword*.

Entretanto, se $k > (n-k)$ ou $k/n > 1/2$, então a codificação alternativa com a matriz de paridades H necessita de menor número de cálculos. Nesse caso, tem-se a codificação baseada na Equação 4.11, $(\overline{u}, \overline{v}_p)H^T = 0$, na qual $(n-k)$ posições de paridades $v_k, v_{k+1}, \ldots, v_{n-1}$ são obtidas como segue:

$$v_j = u_0 p_{0,j} + u_1 p_{1,j} + \ldots + u_{k-1} p_{k-1,j}, \quad k \leq j < n \tag{4.17}$$

Exemplo 4 Considerando o código linear binário (4,2,2) do Exemplo 3. Sendo a mensagem e as *codewords* representadas por $\overline{u} = (u_0, u_1)$ e $\overline{v} = (v_0, v_1, v_2, v_3)$, respectivamente. Da Equação 4.17 tem-se que:

$$\begin{aligned} v_2 &= u_0 + u_1 \\ v_3 &= u_0 \end{aligned} \tag{4.18}$$

A relação entre as $2^2 = 4$ mensagens de 2 bits e as *codewords* correspondentes é dada por:

$$\begin{aligned} (00) &\mapsto (0000) \\ (01) &\mapsto (0110) \\ (10) &\mapsto (1011) \\ (11) &\mapsto (1101) \end{aligned} \tag{4.19}$$

4.4.2 Decodificação do Vetor Padrão

Nessa subseção o processo de codificação é apresentado e objetiva encontrar a *codeword* $\overline{v}$ mais apropriada para a palavra recebida $\overline{r} = \overline{v} + \overline{e}$.

Um vetor padrão para um código de bloco linear C é uma tabela de todos os possíveis vetores recebidos $\overline{r}$ dispostos em um meio na qual a *codeword* $\overline{v}$ pode ser lida para $\overline{r}$. A Tabela 4.2 apresenta o vetor padrão de um código de bloco linear binário.

Tabela 4.2: Vetor padrão de um código de bloco linear binário

$\overline{s}$	$\overline{u}_0 = \overline{0}$	$\overline{u}_2$	$\cdots$	$\overline{u}_{k-1}$
$\overline{0}$	$\overline{v}_0 = \overline{0}$	$\overline{v}_1$	$\cdots$	$\overline{v}_{2^k-1}$
$\overline{s}_1$	$\overline{e}_1$	$\overline{e}_1 + \overline{v}_1$	$\cdots$	$\overline{e}_1 + \overline{v}_{2^k-1}$
$\overline{s}_2$	$\overline{e}_2$	$\overline{e}_2 + \overline{v}_1$	$\cdots$	$\overline{e}_2 + \overline{v}_{2^k-1}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\overline{s}_{2^{k-1}-1}$	$\overline{e}_{2^{k-1}-1}$	$\overline{e}_{2^{k-1}-1} + \overline{v}_1$	$\cdots$	$\overline{e}_{2^{k-1}-1} + \overline{v}_{2^k-1}$

O vetor padrão contém 2^{n-k} linhas e $2^k + 1$ colunas. A entrada das 2^k colunas mais a direita do vetor contém todos os vetores em $V_2 = \{0,1\}^n$. Para descrever o processo de decodificação, se faz necessário introduzir o conceito de *síndrome*. A síndrome de uma palavra em V_2 é definida na Equação 4.20.

$$\overline{s} = rH^T, \tag{4.20}$$

onde H representa a matriz de paridade de C. Sendo $\overline{s}$ um conjunto de sintomas que indicam erros, supondo que a *codeword* $\overline{v} \in C$ é transmitida sobre um BSC ([1]) e recebida como $\overline{r} = \overline{v} + \overline{e}$. A síndrome de $\overline{r}$ é

$$\overline{s} = \overline{r}H^T = (\overline{v} + \overline{e})H^T = \overline{e}H^T, \tag{4.21}$$

onde a igualdade é deduzida da Equação 4.11.

4.4.2.1 Procedimento de Construção do Vetor Padrão

1. Na primeira linha, em cada posição correspondente as 2^k colunas mais a direita, coloca-se todas as *codewords* de C, começando com a *codeword* formada de zeros na posição mais à esquerda. Na posição correspondente a primeira coluna, coloca-se a síndrome composta de zeros. Faça $j = 0$;

2. Faça $j = j+1$. Encontre a palavra com menor distância de *Hamming* $\overline{e}_j$ em V_2, não em C, e não incluída nas linhas anteriores. A síndrome correspondente $\overline{s}_j = \overline{e}_j H^T$ é a primeira (mais à direita) entrada da linha. As 2^k entradas restantes na linha são obtidas pela adição de $\overline{e}_j$ a todas as entradas da primeira linha (as *codewords* de C).

3. Repita o passos anterior até que todos os vetores em V_2 sejam incluídos no vetor. De forma equivalente, $j = j + 1$. Se $j < 2^{n-k}$, então repita o passo anterior, senão pare.

Exemplo 5 O vetor padrão de um código linear binário $(4, 2, 2)$ é apresentado na Tabela 4.3:

Tabela 4.3: Representação do vetor padrão de um código binário $(4, 2, 2)$

$\overline{s}$	00	01	10	11
00	0000	0110	1011	1101
11	1000	1110	0011	0101
10	0100	0010	1111	1001
01	0001	0111	1010	1100

O processo de decodificação seguindo os procedimentos do vetor padrão se dá como segue. Seja $\overline{r}$ a palavra recebida, conforme apresentada na Equação 4.22. A síndrome de um elemento é calculada pela Equação 4.23.

$$\overline{r} = \overline{v} + \overline{e} \tag{4.22}$$

$$\overline{s}_i = (\overline{e}_i + \overline{v})H^T = \overline{e}_i H^T, \tag{4.23}$$

que é independente da escolha particular de $\overline{v} \in C$. O processo de decodificação simplificado é: calcular a síndrome da palavra recebida $\overline{r} = \overline{e}_{i'} + \overline{v}$,

[1] do inglês *Binary Symmetric Channel*

$$\overline{s}_{i'} = (\overline{e}_{i'} + \overline{v})H^T = \overline{e}_{i'}H^T, \tag{4.24}$$

e encontrar $\overline{s}_{i'}$ na coluna mais à esquerda do vetor padrão. Então deve-se ler saída do valor $\overline{e}_{i'}$, da segunda coluna, adiciona-se ele a palavra recebida para obter a *codeword* $\overline{v'} \in C$ mais aproximada em $\overline{r}$. Entretando, ao invés de usar $n \times 2^n$ bits, a decodificação do vetor padrão pode ser implementada com um vetor de $n \times 2^{n-k}$ bits.

Exemplo 6 Considere um código linear binário $(4, 2, 2)$ do Exemplo 3. Suponha que a *codeword* $\overline{v} = (0110)$ é transmitida e que $\overline{r} = (0010)$ é recebida. Então a síndrome é

$$\overline{s} = \overline{r}H^T = (0010)\begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = (1 \quad 0).$$

A partir do vetor padrão de um código, o vetor $e' = (0100)$ é encontrado (Tabela 4.3) e então a *codeword* estimada é $v' = r + e' = (0010) + (0100) = (0110)$, logo um erro foi corrigido.

Este Capítulo apresentou a definição de códigos corretores de erros bem como o embasamento matemático nos processos de codificação e decodificação de códigos de bloco linear. O capítulo apresenta exemplos de formação do vetor padrão e procedimento de cálculo da síndrome e recuperação da *codeword* corrompida. O Capítulo 5 introduz o algoritmo BCH, utilizado para atividades de codificação e decodificação de telecomando no contexto espacial. Nele são apresentados os passos necessários para codificação e decodificação de *codewords* no padrão definido pelo CCSDS.

Capítulo 5

Algoritmo BCH

O principal objetivo desse capítulo é introduzir um conjunto mínimo de conceitos necessários para o entendimento de códigos cíclicos binários e para implementações eficientes dos processos de codificação e decodificação. Também nesse capítulo é apresentada uma importante família de códigos cíclicos binários, chamada de códigos BCH, nome esse dado em homenagem a seus criadores *Bose-Chaudhuri-Hocquenghem*. Códigos binários BCH com distância mínima 3, conhecidos como *Códigos de Hamming*, têm sido bastante utilizados em redes de computadores e em memórias, devido à simplicidade e velocidade nos processos de codificação e decodificação.

5.1 Códigos Cíclicos Binários

Códigos cíclicos são uma classe de códigos corretores de erros que são eficientemente codificados e decodificados utilizando simples registradores de deslocamentos e elementos de lógica combinacional, sendo sua representação baseada em polinômios.

5.1.1 Polinômio Gerador e Polinômio de Verificação de Paridade

Sendo C representado por um código de bloco linear (n, k). Seja $\overline{u}$ e $\overline{v}$ as representações de uma mensagem e de sua *codeword* correspondente em C, respectivamente. Códigos cíclicos são códigos lineares com propriedades que tornam possível sua implementação em hardware. Para toda *codeword* $\overline{v}$ existe um polinômio $\overline{v}(x)$ associado,

$$\overline{v} = (v_0, v_1, \cdots, v_{n-1}) \mapsto \overline{v}(x) = v_0 + v_1 x + \cdots + v_{n-1} x^{n+1} \tag{5.1}$$

A variável x serve para indicar a posição relativa de um elemento v_i de $\overline{v}$ como um termo $v_i x^i$ de $\overline{v}(x), 0 \leq i < n$. Um código de bloco linear C é cíclico se somente se todo o ciclo de deslocamento de uma *codeword* é outra *codeword*,

$$\overline{v} = (v_0, v_1, \cdots, v_{n-1}) \in C \Longleftrightarrow \overline{v}^{(1)} = (v_{n-1}, v_0, \cdots, v_{n-2}) \in C. \tag{5.2}$$

Em linguagem de polinômios, um deslocamento cíclico de uma posição, representado por $\overline{v}^{(1)}(x)$, é realizado pela multiplicação por x módulo $(x^n - 1)$,

$$\overline{v}(x) \in C \Longleftrightarrow \overline{v}^{(1)}(x) = x\overline{v}(x) \ mod \ (x^n - 1) \in C. \tag{5.3}$$

Registradores de deslocamento podem ser usados para este propósito, como ilustrado na Figura 5.1.

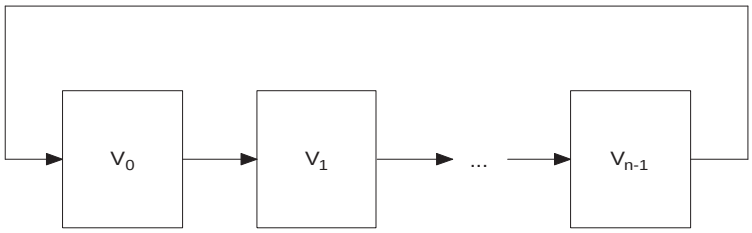

Figura 5.1: Representação de um registrador de deslocamento cíclico

Exemplo 1 Seja $n = 7$, um deslocamento cíclico de uma posição de um vetor $\overline{v} = (0101011)$ é igual a $\overline{v}^{(1)} = (1010101)$. Em termos de polinômios, $\overline{v}(x) = x + x^3 + x^5 + x^6$ e

$$\begin{aligned} \overline{v}^{(1)}(x) = x\overline{v}(x) &= x^2 + x^4 + x^6 + x^7 \ mod\ (x^7 + 1) \\ &= x^2 + x^4 + x^6 + x^7 + (x^7 + 1) + 1 \\ &= 1 + x^2 + x^4 + x^6 \end{aligned} \tag{5.4}$$

5.1.2 O Gerador Polinomial

Uma propriedade importante de códigos cíclicos é que todos os polinômios do código $\overline{v}(x)$ são múltiplos de um único polinômio, $\overline{g}(x)$, chamado de *gerador polinomial* do código. Esse polinômio é especificado por suas raízes, chamadas de *zeros do código*. É possível mostrar que o gerador polinomial $\overline{g}(x)$ divide $(x^n - 1)$. Assim como inteiros, "$a(x)$ divide $b(x)$" sempre que $b(x) = q(x)a(x)$. Entretanto, para encontrar o gerador polinomial, o polinômio $(x^n - 1)$ deve ser fatorado em fatores irredutíveis, $\phi_j(x), j = 1, 2, \cdots, \ell$,

$$(x^n - 1) = \phi_1(x)\phi_2(x)\cdots\phi_\ell(x). \tag{5.5}$$

Uma importante observação é que em operações binárias de $a - b$ e $a + b$ (módulo 2) resulta no mesmo valor. Conforme a Equação 5.5, o polinômio $g(x)$ é dado por:

$$\overline{g}(x) = \prod_{j \in J \subset \{1,2,\cdots,\ell\}} \phi_j(x).$$

Exemplo 2 Com coeficientes sobre $Z_2 = \{0, 1\}$, o polinômio $x^7 + 1$ é fatorado como

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1). \tag{5.6}$$

Alguns exemplos de códigos cíclicos de tamanho 7 são:

- Um código cíclico binário de Hamming (7,4,3) é gerado pelo polinômio

$$\overline{g}(x) = x^3 + x + 1.$$

- Um código cíclico de paridade é geardo por

$$\overline{g}(x) = (x^3 + x + 1)(x^3 + x^2 + 1).$$

- Um código seqüencial de tamanho máximo é gerado por

$$\overline{g}(x) = (x+1)(x^3+x+1).$$

5.1.3 Codificação e Decodificação de Códigos Cíclicos Binários

A dimensão de um código cíclico binário (n,k) é dada por

$$k = n - deg[\overline{g}(x)],$$

onde deg[.] representa o grau do argumento. Desde que um código cíclico$C$ seja também linear, qualquer conjunto de$k$ vetores linearmente independentes (LI) podem ser selecionados com uma matriz geradora. Em particular, os vetores binários associados à$\overline{g}(x), x\overline{g}(x), \cdots, x^{k-1}\overline{g}(x)$ são LI. Esses vetores podem ser usados como linhas de uma matriz geradora de$C$. Nesse caso, uma regra de codificação sistemática é obtida, fazendo com que os bits da mensagem não apareçam explicitamente em qualquer posição das *codewords*.

Exemplo 3 Considere o código cíclico de Hamming (7,4,3) com o gerador polinomial$\overline{g}(x) = x^3+x+1 \Longleftrightarrow \overline{g} = (1101)$. Uma matriz geradora para esse código é

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}. \tag{5.7}$$

Como alternativa, a sub-matriz de paridade da matriz geradora de um código cíclico pode ser construída com os vetores associados com os seguintes polinômios:

$$\begin{gathered} x^{n-1} \ mod \ \overline{g}(x), \\ \vdots \\ x^{n-k+1} \ mod \ \overline{g}(x), \\ x^{n-k} \ mod \ \overline{g}(x), \end{gathered}$$

e uma codificação sistemática é obtida, como ilustrado no exemplo anterior.

Exemplo 4 Considere um código cíclico de Hamming$C$ $(7,4,3)$. Logo, $\overline{g}(x) = x^3+x+1$, e

$$\begin{aligned} x^6 \ mod \ (x^3+x+1) &= x^2+1, \\ x^5 \ mod \ (x^3+x+1) &= x^2+x+1, \\ x^4 \ mod \ (x^3+x+1) &= x^2+x, \\ x^3 \ mod \ (x^3+x+1) &= x+1, \end{aligned}$$

Seguindo a matriz geradora sistemática de$C$ tem-se

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}. \tag{5.8}$$

Seja $\overline{u}(x)$ a mensagem a ser codificada. A codificação de*codewords* de códigos cíclicos binários pode ser sistemática ou não-sistemática, dependendo do meio com que a mensagem é processada:

- Codificação não-sistemática:

$$\overline{v}(x) = \overline{u}(x)\overline{g}(x). \tag{5.9}$$

- Codificação sistemática:

$$\overline{v}(x) = x^{n-k}\overline{u}(x) + [x^{n-k}\overline{u}(x) \ mod \ \overline{g}(x)]. \tag{5.10}$$

5.1.4 Polinômio de Verificação de Paridade

Outro polinôminio, $\overline{h}(x)$, chamado de polinômio de paridade, pode ser associado com a matriz de verificação de paridade. O gerador polinomial e o polinômio de verificação de paridade são dados por:

$$\overline{g}(x)\overline{h}(x) = x^n + 1. \tag{5.11}$$

O polinômio de verificação de paridade pode ser calculado a partir do gerador polinomial com $\overline{h}(x) = (x^n + 1)/\overline{g}(x) = h_0 + h_1 x + \cdots + h_k x^k$. Então, a matriz de verificação de paridade para C é dada pelas linhas do vetor binário associadas com os primeiros$n-k-1$ deslocamentos cíclicos diferentes de zero $\overline{h}^{(j)} = x^j\overline{h}(x) \ mod \ (x^n - 1), j = 0, 1, \cdots, n-k-1$.

$$H = \begin{pmatrix} h_0 & h_1 & \cdots & & h_k & 0 & 0 & \cdots & 0 \\ 0 & h_0 & h_1 & \cdots & \cdots & h_k & 0 & \cdots & 0 \\ 0 & 0 & h_0 & h_1 & \cdots & \cdots & h_k & \cdots & 0 \\ 0 & 0 & 0 & \ddots & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & h_k \end{pmatrix}. \tag{5.12}$$

Exemplo 5 O polinômio de verificação de paridade para o código cíclico de Hamming (7,4,3), com o gerador polinomial $\overline{g}(x) = x^3 + x + 1$, é $\overline{h}(x) = (x^7 + 1)/(x^3 + x + 1) = x^4 + x^2 + x + 1$. A matriz de verificação de paridade para esse código é dada por

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}. \tag{5.13}$$

5.2 Propriedades sobre $GF(2^m)$

O corpo $GF(2^m)$ é isomórfico (com relação a "+") para o espaço linear$0, 1^m$. Em outras palavras, para cada elemento $\beta \in GF(2^m)$. existe um único vetor binário m-dimensional $\overline{v}_\beta \in 0, 1^m$.

Existe um elemento primitivo $\alpha \in GF(2^m)$, na qual todo elemento β em $GF(2^m)$ pode ser representado como $\beta = \alpha^j, 0 \leq j \leq 2^m - 2$. Este elemento α é a raíz de um polinômio irredutível, chamado de polinômio primitivo, $p(x)$ sobre $0, 1$, por ex. $p(a) = 0$. Um elemento primitivo do corpo $GF(2^m)$ satisfaz a equação $\alpha^{2^m-1} = 1$ e $n = 2^m - 1$ é o menor inteiro positivo na qual $a^n = 1$.

Exemplo 6 Seja $p(x) = x^3 + x + 1$ um polinômio primitivo de $GF(2^3)$. Seja α um elemento primitivo na qual $p(\alpha) = a^3 + a + 1 = 0$ e $a^7 = 1$. A Tabela 5.1 apresenta três formas diferentes de representação dos elementos de $GF(2^3)$.

Quando adicionados elementos a $GF(2^m)$, a representação vetorial é a mais usada, pois uma simples operação de OU EXCLUSIVO é necessária. Entretanto, quando elementos são multiplicados, a representação de potência é mais eficiente. Usando representação de potência,

Tabela 5.1: Diferentes formas de representação dos elementos de $GF(2^3)$

Potência	Polinômio	Vetor
-	0	000
1	1	001
α	α	010
α^2	α^2	100
α^3	$1+\alpha$	011
α^4	$\alpha+\alpha^2$	110
α^5	$1+\alpha+\alpha^2$	111
α^6	$1+\alpha^2$	101

uma multiplicação torna-se simplesmente uma adição módulo2^m-1. A representação polinomial pode ser apropriada quando se faz operações módulo um polinômio.

Na representação de potência, $\alpha^{2^m-1}=1$. Nota-se que $\alpha^{2m}=\alpha\alpha^{2^m-1}=\alpha$, $\alpha^{2^m+1}=\alpha^2\alpha^{2^m-1}=\alpha^2$, etc. Isto quer dizer que as potências de α são calculadas módulo 2^m-1. Aplicando o mesmo argumento, mostra-se que $\alpha^{-1}=\alpha^{-1+2^m-1}=\alpha^{2^m-2}$. Para o Exemplo 6 apresentado anteriormente, $\alpha^{-1}=\alpha^{2^3-2}=\alpha^6$. Em geral, a inversa $\beta^{-1}=\alpha^k$ de um elemento $\beta=\alpha^\ell$ é encontrada por $k, 0\leq k<2^m-1$ na qual $\alpha^{\ell+k}=1$ e pode ser expressada como $\ell+k=0\ mod\ (2^m-1)$. Entretanto, $\ell=2^m-1-k$. Da mesma forma, na representação polinomial, a equação $p(\alpha)=0$ é usada para reduzir as expressões. No Exemplo 6, $\alpha^3=\alpha^3+0=\alpha^3+(\alpha^3+\alpha+1)=\alpha+1$.

5.3 Tabelas de *log* e *anti-log*

Uma outra forma de se executar operações de soma e multiplicação sobre$GF(2^m)$ é usar duas tabelas *look-up*. Isso permite a troca entre representação polinomial (vetor) e representação de potência de um elemento de $GF(2^m)$.

A tabela de *anti-log* $A(i)$ é útil quando executadas operações de adição. A tabela dá o valor de um vetor binário, representado como um inteiro em representação natural. $A(i)$, que corresponde ao elemento α^i. A tabela de *log* $L(i)$ é usada quando executadas operações de multiplicação. Essa tabela dá o valor da potência de alfa, $\alpha^{L(i)}$ que corresponde ao vetor binário correspondente representado pelo inteiro i, dado pela seguinte equação:

$$\alpha^{L(i)}=A(i). \tag{5.14}$$

A melhor forma para entender o uso de tabelas de *log* e *anti-log* no cálculo de operações artiméticas em $GF(2^m)$ é através do exemplo.

Exemplo 7 Considere $GF(2^3)$ com $p(\alpha)=\alpha^3+\alpha+1$ e $\alpha^7=1$. As tabelas de *log* e *anti-log* respectivamente são apresentadas na Tabela 5.2.

Considere o cálculo de um elemento $\gamma=\alpha(\alpha^3+\alpha^5)^3$ na forma vetorial. Usando as propriedades de $GF(2^3)$, γ pode ser calculado como segue: $\alpha^3+\alpha^5=110\oplus111=001=\alpha^2$. Então, $\gamma=\alpha(\alpha^2)^3=\alpha^{1+6}=\alpha^7(=1)$.

Por outro lado, usando tabelas de *log* e *anti-log*, o cálculo de γ é realizado como segue: $\gamma=A(L(A(3)\oplus A(5))*3+1)=A(L(3\oplus7)*3+1)=A(L(4)*3+1)=A(2*3+1)=A(7)=(A(0)=1)$. Tabelas de *log* e *anti-log* são usadas para execução de adição e multiplicação sobre$GF(2^m)$. As regras para formação das tabelas de *log* e *anti-log* são apresentadas em [31].

Tabela 5.2: Representação das tabelas de *log* e *anti-log*

Índice	$GF(2^m)$ para vetor Tabela de Anti-log, $A(i)$	Vetor para $GF(2^m)$ Tabela de Log, $L(i)$
0	1	-1
1	2	0
2	4	1
3	3	3
4	6	2
5	7	6
6	5	4
7	0	5

5.4 Exemplo de Codificação e Decodificação BCH

Essa seção tem por objetivo apresentar um exemplo dos processos de codificação, decodificação e recuperação da mensagem corrompida. Para o exemplo apresentado aqui, foi utilizado o padrão BCH(7,4), onde a mensagem é composta de 7 bits, sendo 4 de dados e os outros 3 de paridade, com capacidade de correção de 1 bit da mensagem original.

5.4.1 Regras de Formação das Matrizes Geradora e de Paridade

Como apresentado anteriormente, o algoritmo BCH trabalha com uma matriz geradora chamada G e outra matriz de paridade H. A matriz G possui dimensão 4×7 (4 linhas e 7 colunas) e a regra de formação é apresentada na Seção 5.1.3. O polinômio utilizado para esse padrão é dado por

$$g(x) = x^3 + x + 1 \tag{5.15}$$

A matriz geradora obtida é

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{5.16}$$

A matriz G é composta por 2 sub-matrizes. A sub-matriz chamada deI_1 de dimensão (4×4) é representada por uma matriz identidade, na qual os valores de sua diagonal são '1', enquanto os demais valores são dados por '0'. A sub-matriz chamada deG_1 de dimensão (4×3) apresenta os valores de síndrome calculados através da divisão de polinômios pelo polinômio gerador como segue:

$$\begin{aligned} x^6 \ mod \ (x^3 + x + 1) &= x^2 + 1 \quad (101) \\ x^5 \ mod \ (x^3 + x + 1) &= x^2 + x + 1 \quad (111) \\ x^4 \ mod \ (x^3 + x + 1) &= x^2 + x \quad (110) \\ x^3 \ mod \ (x^3 + x + 1) &= x + 1 \quad (011) \end{aligned}$$

As sub-matrizes I_1 e G_1 são apresentadas como segue:

$$I_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.17}$$

$$G_1 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \tag{5.18}$$

A matriz H^T de dimensão (7×3) é composta da sub-matriz G_1 e da matriz identidade $I(3 \times 3)$. A regra de composição da mesma é dada por

$$H^T = \begin{pmatrix} G_1 \\ \hline I \end{pmatrix} \tag{5.19}$$

A matriz H^T resultante é apresentada como

$$H^T = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.20}$$

5.4.2 Codificação BCH

À partir das matrizes geradora e de paridade formadas, é possível codificar qualquer mensagem composta de 4 bits a ser transmitida. Por exemplo, para codificar a mensagem $u_1 = (0110)$, é necessário realizar a seguinte operação:

$$\overline{u}_1 = u_1 \times G \tag{5.21}$$

Ou seja, multiplica-se o vetor da mensagem a ser codificada (u_1) pela matriz geradora G, como resultado tem-se um vetor de 7 bits contendo nas 4 primeiras posições a mensagem original, seguida dos 3 bits de paridade anexados a mensagem, logo

$$\overline{u}_1 = (0110) \times \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = (0110001) \tag{5.22}$$

A mensagem a ser transmitida, dada por $\overline{u}_1$ é então calculada e o vetor resultante é dado por $\overline{u}_1 = (0110001)$.

5.4.3 Decodificação BCH

Ao receber a mensagem transmitida $(\overline{u}_1)$, multiplica-se pela matriz de paridade H^T, a fim de obter o vetor de síndromes (S'). Logo, tem-se:

$$S' = \overline{u}_1 \times H^T \tag{5.23}$$

onde

$$S' = 0110001 \times \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (000) \tag{5.24}$$

Nesse caso o valor de síndrome resultou em (0 0 0), representando que a mensagem recebida não foi corrompida durante o processo de transmissão e que não precisará ser corrigida. Agora, suponha-se que a mensagem tenha sido corrompida durante a transmissão e ela é recebida na seguinte forma $\overline{u}_1 = (1110001)$, alterando o bit mais significativo (bit mais à esquerda). O processo de cálculo da síndrome é executado, logo

$$S' = 1110001 \times \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (101) \tag{5.25}$$

O próximo passo é encontrar na matriz G, o valor correspondente a síndrome $S' = (101)$. Seguindo a matriz apresentada na Equação 5.16, o valor correspondente a síndrome(101) é dado por $e' = (1000)$. O valor da síndrome é localizado através de uma busca na matriz$G$ partindo na linha 1 até a linha 4, entre as colunas 5 e 7 de cada linha tem-se os valores de síndrome. Logo, o valor associado a síndrome (101) é encontrado na primeira linha, e o valor referente é dado por (1000), chamado de e' (representando um erro estimado).

5.4.4 Correção da Mensagem Corrompida

Para corrigir e restaurar a mensagem original transmitida, basta executar uma operação de OU EXCLUSIVO entre a mensagem recebida (4 bits da mensagem sem os bits de paridade) $\overline{r}_1 = (1110)$ e o vetor de erro estimado ($e' = (1000)$). Logo tem-se:

$$\begin{aligned} u_1 &= \overline{r}_1 \oplus e' \\ u_1 &= 1110 \oplus 1000 = 0110 \end{aligned} \tag{5.26}$$

Então, a mensagem corrompida é restaurada e a mensagem original (0110) está pronta para ser processada. Por fim, é apresentado o exemplo onde a mensagem é corrompida em 2 bits, e o processamento não é capaz de corrigir a mesma pois o padrão BCH(7,4) permite correção de no máximo 1 bit da mensagem transmitida.

Como exemplo, a mensagem transmitida originalmente é a mesma usada nos exemplos anteriores, $\overline{u}_1 = (0110001)$. Suponha-se então que a mensagem recebida seja corrompida nos dois bits mais significativos (bits mais à esquerda), resultando na mensagem recebida$\overline{r}_1 = (1010001)$. Inicia-se o processo de decodificação da mensagem, sendo o primeiro passo a ser executado dado pelo cálculo do valor de síndrome, sendo:

$$S' = 1010001 \times \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (010) \tag{5.27}$$

Ao tentar encontrar o valor de síndrome $S' = (010)$ na matriz G da Equação 5.16, verifica-se que não existe um vetor de erro estimado para esse padrão de síndrome, logo não é possível corrigir a mensagem recebida. Então o algoritmo sinaliza de alguma forma esse evento e suspende seu processamento. No projeto do circuito decodificador de telecomando BCH, quando não é possível corrigir um telecomando corrompido, o circuito sinaliza aos demais módulos que o telecomando não foi restaurado. Na seqüência, um pedido de re-envio do telecomando para a estação terrestre é realizado, garantindo assim, que só serão processados os telecomandos recebidos de forma correta, dando maior confiabilidade ao sistema. Esse exemplo apresentou a correção de erros com a capacidade de correção de apenas 1 bit por *codeblock*, seguindo as recomendações CCSDS. Para correção de mais de um bit por *codeblock* deve-se usar *look-up tables*.

Esse capítulo apresentou a classe de códigos corretores de erros BCH, utilizada para atividades de correção de telecomandos corrompidos durante o processo de transmissão de dados provenientes da estação terrestre para o veículo espacial. No Capítulo 6 é introduzida a classe de códigos Reed-Solomon, largamente empregada em projetos das maiores agências espaciais mundiais e utilizada para correção de erros ocorridos durante a transmissão de telemetria do veículo espacial para a estação terrestre.

Capítulo 6

Algoritmo de Reed-Solomon

Em 1960, Irving Reed e Gus Solomon publicaram um artigo no *Journal of the Society for Industrial and Applied Mathematics* [1], descrevendo uma nova classe de códigos corretores de erros chamados Códigos Reed-Solomon. Esses se mostraram bastante eficientes sendo atualmente utilizados em diversas aplicações, entre estas as da área espacial. Essa seção apresenta os algoritmos de codificação e decodificação dessa classe de códigos corretores de erros.

Códigos Reed-Solomon são códigos cíclicos não-binários constituídos de seqüências de m-bits, onde m é qualquer inteiro positivo maior que 2 [5]. Um código cíclico é aquele que, a partir de um vetor U pertencente ao subespaço vetorial do código cíclico, é possível gerar todos os demais códigos através do deslocamento sucessivo de U. O vetor U pode ser descrito como: $U = (\mu_0\ \mu_1\ \mu_2\ ...\ \mu_{n-1})$.

Seqüências RS são representadas na forma $RS(n, k)$, onde n representa o número total de símbolos, e k o número total de símbolos a serem codificados.

$$(n, k) = (2^m - 1, 2^m - 1 - 2t) \tag{6.1}$$

Na Equação 6.1, t representa a capacidade de correção de símbolos com erros do código, e $r = 2t$ o número de símbolos de paridade anexados a mensagem a ser transmitida.

Códigos RS são particularmente úteis para correção de erros em rajada[1]. Também podem ser usados eficientemente em canais onde o conjunto de símbolos de entrada é consideravelmente grande.

6.1 Polinômio Primitivo Usado para Definir o Corpo Finito

Uma classe de polinômios, chamada *polinômios primitivos*, se faz necessária para definir os códigos RS. A seguinte condição é necessária e suficiente para garantir que um polinômio seja primitivo. Um polinômio irredutível, $f(X)$ de grau m é dito como primitivo se o menor inteiro positivo n na qual $f(X)$ divide $X^n + 1$ é $n = 2^m - 1$. Um polinômio irredutível não pode ser fatorado em polinômios de ordem menor. Por exemplo, o polinômio $x^2 + x + 1$ é irredutível, mas $x^2 + 1$ não, porque $(x + 1)(x + 1) = x^2 + 2x + 1 = x^2 + 1$ (módulo 2). Entretanto, nem todos os polinômios irredutíveis são primitivos. A Tabela 6.1 apresenta o mapeamento dos8 elementos pertencentes ao Corpo de Galois representado por $GF(2^3)$ com $f(x) = x^3 + x + 1$. Há diferentes maneiras para representar um polinômio, entre elas notação binária e notação decimal. As notações na seqüência apresentada na Equação 6.2 representam o mesmo polinômio.

[1] seqüência de erros consecutivos, também conhecidos como *burst errors*.

$$x^5 + x^2 + 1 = 100101_2 = 37_{10} \tag{6.2}$$

Tabela 6.1: Mapeamento de elementos no corpo para $GF(2^3)$ com $f(x) = 1 + X + X^3$

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

A Tabela 6.2 apresenta uma lista de polinômios primitivos, com m variando de 3 a 24 [4]. No caso de aplicações espaciais no padrão CCSDS, cada pacote é dividido em blocos de 256 bytes, sendo necessários 256 elementos que compõe o Corpo de Galois para representar cada um dos bytes. O polinômio que representa esses elementos é definido como sendo de grau 8, ou seja, $m = 8$.

Tabela 6.2: Classe de polinômios primitivos

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

O seguinte exemplo ilustra o algoritmo RS com símbolos de 4 bits. O gerador polinomial define um corpo finito sobre todas as operações que são calculadas da seguinte maneira: O termo GF(2^4) significa que o corpo finito possui 16 elementos [16]. O cálculo dos corpos começa com um elemento primitivo α, que nesse caso é 2 ou 0010 (x). Um incremento na potência de α representa cada membro sucessivo pertencente ao corpo; então, α é chamada de raíz primitiva porque sua potência representa todos os membros pertencentes ao corpo diferentes de zero. A Tabela 6.3 mostra os cálculos dos elementos pertencentes à GF(2^4).

Após definidos todos os elementos pertencentes a GF(2^4), operações de soma, subtração, multiplicação e divisão são executadas. As operações de soma e subtração são as mesmas e trabalham com portas XOR de elementos representados por valores numéricos [32]. Por exemplo, $\alpha^5 + \alpha^6 = 0110$ XOR $1100 = 1010 = \alpha^9$. Operações de multiplicação e divisão são executadas através de soma e subtração de potência de elementos, lembrando que $\alpha^{15} = 1$. Por exemplo,

Tabela 6.3: Cálculos dos elementos de GF(2^4)

Potência	**Cálculo**	**Valor Numérico**
$\alpha = x$	x	$0010_2 = 2_{10}$
$\alpha^2 = x \times x$	x^2	$0100_2 = 4_{10}$
$\alpha^3 = x \times x \times x$	x^3	$1000_2 = 8_{10}$
$\alpha^4 = \alpha \times \alpha^3$	$x^4 = x + 1$	$0011_2 = 3_{10}$
$\alpha^5 = \alpha \times \alpha^4$	$x^5 = x^2 + x$	$0110_2 = 6_{10}$
$\alpha^6 = \alpha \times \alpha^5$	$x^6 = x^3 + x^2$	$1100_2 = 12_{10}$
$\alpha^7 = \alpha \times \alpha^6$	$x^7 = x^4 + x^3 = x^3 + x + 1$	$1011_2 = 11_{10}$
$\alpha^8 = \alpha \times \alpha^7$	$x^8 = x^4 + x^2 + x = x^2 + 1$	$0101_2 = 5_{10}$
$\alpha^9 = \alpha \times \alpha^8$	$x^9 = x^3 + x$	$1010_2 = 10_{10}$
$\alpha^{10} = \alpha \times \alpha^9$	$x^{10} = x^4 + x^2 = x^2 + x + 1$	$0111_2 = 7_{10}$
$\alpha^{11} = \alpha \times \alpha^{10}$	$x^{11} = x^3 + x^2 + x$	$1110_2 = 14_{10}$
$\alpha^{12} = \alpha \times \alpha^{11}$	$x^{12} = x^4 + x^3 + x^2 = x^3 + x^2 + x + 1$	$1111_2 = 15_{10}$
$\alpha^{13} = \alpha \times \alpha^{12}$	$x^{13} = x^4 + x^3 + x^2 + x = x^3 + x^2 + 1$	$1101_2 = 13_{10}$
$\alpha^{14} = \alpha \times \alpha^{13}$	$x^{14} = x^4 + x^3 + x = x^3 + 1$	$1001_2 = 9_{10}$
$\alpha^{15} = \alpha \times \alpha^{14}$	$x^{15} = x^4 + x = 1$	$0001_2 = 1_{10}$

$\alpha^2 \times \alpha^4 = \alpha^6$; $\alpha^{13} \times \alpha^9 = \alpha^{22} = \alpha^{15} \times \alpha^7 = \alpha^7$; $\alpha^4/\alpha^2 = \alpha^2$. O próximo passo é montar a relação de cada elemento do corpo do polinômio. Para simplificar a demonstração, o polinômio utilizado é de grau 3. A Equação 6.3 apresenta α^3 como sendo a soma de α-termos de menor ordem.

$$\alpha^3 = 1 + \alpha \tag{6.3}$$

De fato, todas as potências de α podem ser então representadas. Por exemplo, considere a Equação 6.4.

$$\alpha^4 = \alpha.\alpha^3 = \alpha.(1 + \alpha) = \alpha + \alpha^2 \tag{6.4}$$

Sabendo que $\alpha^4 = \alpha + \alpha^2$, considere a Equação 6.5.

$$\alpha^5 = \alpha.\alpha^4 = \alpha.(\alpha + \alpha^2) = \alpha^2 + \alpha^3 \tag{6.5}$$

Das Equações 6.3 e 6.5 obtém-se

$$\alpha^5 = 1 + \alpha + \alpha^2 \tag{6.6}$$

Agora, usando a Equação 6.6, tem-se

$$\alpha^6 = \alpha.\alpha^5 = \alpha.(1 + \alpha + \alpha^2) = \alpha + \alpha^2 + \alpha^3 = 1 + \alpha^2 \tag{6.7}$$

Usando a Equação 6.7, obtém-se

$$\alpha^7 = \alpha.\alpha^6 = \alpha.(1 + \alpha^2) = \alpha + \alpha^3 = 1 = \alpha^0 \tag{6.8}$$

Note que $\alpha^7 = \alpha^0$, e portanto, os oito elementos finitos do corpo de $GF(2^3)$ são

$$\{0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\} \tag{6.9}$$

Com o desenvolvimento dessas funções é possível obter a relação de todos os elementos pertencentes a $GF(2^3)$, pois cada elemento é representado pela soma de termos de menor ordem. A Tabela 6.4 representa as operações de adição e multiplicação de elementos em um corpo finito de grau 3 na representação de matrizes.

Tabela 6.4: Mapeamento de soma e multiplicação de elementos para$GF(2^3)$

Tabela de Adição									Tabela de Multiplicação							
	α^0	α^1	α^2	α^3	α^4	α^5	α^6			α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	0	α^3	α^6	α^1	α^5	α^4	α^2		α^0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^1	α^3	0	α^4	α^0	α^2	α^6	α^5		α^1	α^1	α^2	α^3	α^4	α^5	α^6	α^0
α^2	α^6	α^4	0	α^5	α^1	α^3	α^0		α^2	α^2	α^3	α^4	α^5	α^6	α^0	α^1
α^3	α^1	α^0	α^5	0	α^6	α^2	α^4		α^3	α^3	α^4	α^5	α^6	α^0	α^1	α^2
α^4	α^5	α^2	α^1	α^6	0	α^0	α^3		α^4	α^4	α^5	α^6	α^0	α^1	α^2	α^3
α^5	α^4	α^6	α^3	α^2	α^0	0	α^1		α^5	α^5	α^6	α^0	α^1	α^2	α^3	α^4
α^6	α^2	α^5	α^0	α^4	α^3	α^1	0		α^6	α^6	α^0	α^1	α^2	α^3	α^4	α^5

As tabelas de adição e multiplicação representam uma matriz de dimensão$(2^m-1)\times(2^m-1)$ e contém todas as associações aditivas e multiplicativas dos elementos pertencentes a$GF(2^3)$.

6.2 Codificação Reed-Solomon

A Equação 6.1 apresenta a forma mais convencional de códigos RS em termos dos parâmetros n, k, t e qualquer inteiro positivo $m > 2$.

Um gerador polinomial, representado por $g(x)$, representa todos os polinômios válidos pertencentes ao Corpo de Galois. Sendo o gerador polinomial de grau$2t$, ele deve ser precisamente $2t$ potências sucessivas de α que são raízes do polinômio [1] [4] [33]. A Equação 6.10 descreve o gerador polinomial com $2t = n - k = 4$ raízes.

$$\begin{aligned} g(X) &= (X-\alpha)(X-\alpha^2)(X-\alpha^3)(X-\alpha^4) \\ &= (X^2-(\alpha+\alpha^2)X+\alpha^3)(X^2-(\alpha^3+\alpha^4)X+\alpha^7) \\ &= (X^2-\alpha^4X+\alpha^3)(X^2-\alpha^6X+\alpha^0) \\ &= X^4-(\alpha^4+\alpha^6)X^3+(\alpha^3+\alpha^{10}+\alpha^0)X^2-(\alpha^4+\alpha^9)X+\alpha^3 \\ &= X^4-\alpha^3X^3+\alpha^0X^2-\alpha^1X+\alpha^3 \end{aligned} \tag{6.10}$$

Seguindo o formato de baixa ordem para alta ordem, e trocando os sinais negativos para positivo na Equação 6.1, sendo que em corpos binários $+1 = -1$, o gerador $g(X)$ pode ser representado na forma

$$g(X) = \alpha^3 + \alpha^1X + \alpha^0X^2 + \alpha^3X^3 + X^4 \tag{6.11}$$

A seguir é demonstrado o passo de codificação para a mensagem apresentada na Equação 6.12.

$$\begin{matrix} \underbrace{010} & \underbrace{110} & \underbrace{111} \\ \alpha^1 & \alpha^3 & \alpha^5 \end{matrix} \tag{6.12}$$

Primeiramente multiplica-se o polinômio da mensagem $\alpha^1 + \alpha^3 X + \alpha^5 X^2$ por $X^{n-k} = X^4$, resultando em $\alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6$.

O próximo passo é dividir o polinômio da mensagem pelo gerador polinomial da Equação 6.11, $\alpha^3 + \alpha^1 X + \alpha^0 X^2 + \alpha^3 X^3 + X^4$. É possível verificar que a divisão de polinômios resulta no seguinte polinômio de paridade:

$$p(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 \tag{6.13}$$

Então, o polinômio do bloco de códigos a ser transmitido pode ser escrito como

$$U(X) = \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \tag{6.14}$$

6.2.1 Codificação Sistemática com um Registrador de Deslocamento de $(n-k)$ Estágios

Um LFSR (*Linear Freedback Shift Register*) é composto de um registrador de deslocamento e uma função de retorno. Um registrador de deslocamento é um dispositivo cuja função é deslocar seu conteúdo em posições adjacentes ao registrador ou deslocar o conteúdo para a saída. O conteúdo de um registrador de deslocamento é geralmente composto de '1's e '0's. Se um registrador de deslocamento contém o seguinte padrão de bit "1101", um deslocamento, para a direita nesse caso, resultaria em "0110", com mais um deslocamento para direita obteria-se "0011".

Dois usos para registradores de deslocamento são:

1. conversão entre dados paralelos e serial;
2. atraso em uma *stream* de bit serial;

A função de conversão pode ser feita dos seguintes modos: preencher todas as posições do registrador de deslocamento de uma única vez (paralelo) e então deslocar os bits para a saída (serial). A função de atraso simplesmente desloca os bits do final de um registrador de deslocamento para outro, fornecendo um atraso igual ao comprimento do registrador de deslocamento. A Figura 6.1 apresenta o fluxo no processo de conversão de dados paralelos em serial e vice-versa.

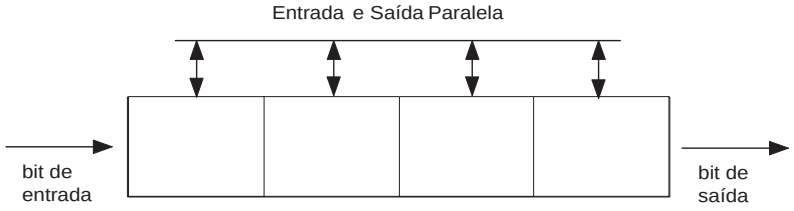

Figura 6.1: Registrador de deslocamento

Em um LFSR, os bits contidos nas posições selecionadas no registrador de deslocamento são combinados no mesmo tipo de função e o resultado é colocado de volta nos registradores de entrada. A função de retorno em um LFSR possui alguns nomes: XOR (ou exclusivo), paridade ímpar, soma módulo 2. Qualquer que seja o nome, a função é simples: 1) Somar os valores de bits selecionados, 2) se a soma for ímpar, a saída da função é 1; senão a saída é 0. A Tabela 6.5 mostra a saída para uma função XOR de 3 entradas:

O uso de um circuito para codificar uma seqüência de 3 símbolos na forma sistemática com o Código RS(7,3) requer a implementação de uma LFSR, como mostrado na Figura 6.2. Podemos

Tabela 6.5: Função XOR de 3 entradas

Entrada A	Entrada B	Entrada C	Saída
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

verificar que os fatores de multiplicação da esquerda para a direita correspondem aos coeficientes do polinômio na Equação 6.11 (baixa ordem para alta ordem). O código RS(7,3) é constituído de $2^m - 1 = 7$ símbolos, e cada símbolo possui $m = 3$ bits.

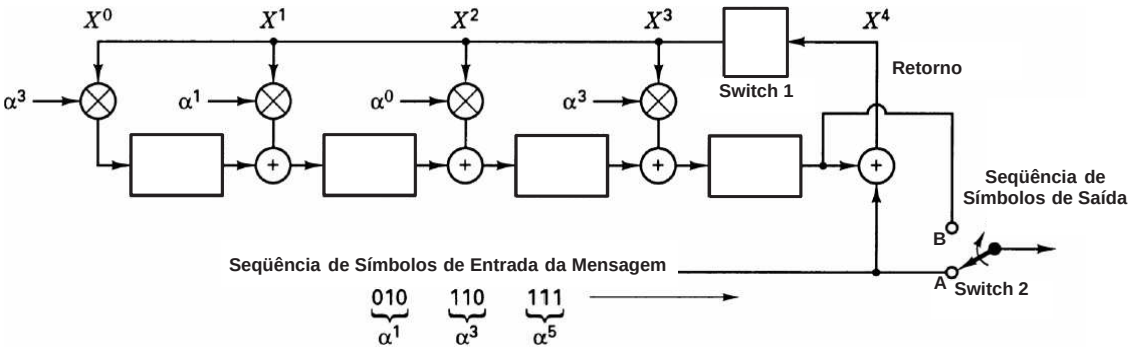

Figura 6.2: Codificador LFSR para RS(7,3)

A operação implementada pelo codificador da Figura 6.2 resulta em *codewords* na forma não sistemática, procedendo do mesmo modo como feito em códigos binários. Os passos podem ser descritos como segue:

1. Switch 1 é fechado durante os primeiros k ciclos de clock para permitir o deslocamento dos símbolos da mensagem em $(n - k)$ estágios no registrador de deslocamento.
2. Switch 2 está na posição A durante os primeiros k ciclos de clock para permitir transferência de símbolos da mensagem diretamente para um registrador de saída.
3. Depois de transferir os kth símbolos da mensagem para o registrador de saída, switch 1 é aberto e switch 2 é movido para a posição B.
4. Os $(n-k)$ ciclos de clock restantes apagam os símbolos de paridade contidos no registrador de saída.
5. O número total de ciclos de clock é igual a n, e o conteúdo do registrador de saída é o polinômio $p(X) + X^{n-k}m(X)$, onde $p(X)$ representa os símbolos de paridade e $m(X)$ os símbolos da mensagem em forma de polinômio.

Os passos operacionais durante os primeiros $k = 3$ deslocamentos do circuito codificador da Tabela 6.6 são os seguintes:

Tabela 6.6: Conteúdo dos registradores no processo de codificação LFSR

FILA DE ENTRADA			CICLO DE CLOCK	REGISTRADORES				RETORNO
α^1	α^3	α^5	0	0	0	0	0	α^5
	α^1	α^3	1	α^1	α^6	α^5	α^1	α^0
		α^1	2	α^3	0	α^2	α^2	α^4
		-	3	α^0	α^2	α^4	α^6	-

Depois do terceiro ciclo de clock, o registrador contém os 4 símbolos de paridade, representados por α^0, α^2, α^4 e α^6, como mostrado. Então, o switch 1 do circuito é aberto, switch 2 é levado para a posição B e os símbolos contidos no registrador são deslocados para a saída. Conseqüentemente o *codeword* de saída, escrito na forma polinomial, pode ser representado como apresentado na Equação 6.15.

$$\begin{aligned} U(X) &= \sum_{n=0}^{6} u_n X^n \\ U(X) &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \\ &= (100) + (001)X + (011)X^2 + (101)X^3 + (010)X^4 + (110)X^5 + (111)X^6 \end{aligned} \tag{6.15}$$

6.3 Decodificação Reed-Solomon

Neste capítulo, de forma a exemplificar o algoritmo de decodificação, uma mensagem de teste codificada na forma sistemática usando um código RS(7,3) resultou em uma*codeword* polinomial descrita pela Equação 6.15. Agora, assume-se que durante a transmissão essa *codeword* foi corrompida resultando no recebimento de 2 símbolos com erro. O número de erros corresponde a capacidade máxima de correção do código. Para a *codeword* de 7 símbolos desse exemplo, o erro padrão pode ser descrito na forma polinomial representada na Equação 6.16.

$$e(X) = \sum_{n=0}^{6} e_n X^n \tag{6.16}$$

Para esse exemplo, dois símbolos de erros são representados comoα^2 e α^5 conforme a Equação 6.17.

$$\begin{aligned} e(X) &= 0 + 0X + 0X^2 + \alpha^2 X^3 + \alpha^5 X^4 + 0X^5 + 0X^6 \\ &= (000) + (000)X + (000)X^2 + (001)X^3 + (111)X^4 + (000)X^5 + (000)X^6 \end{aligned} \tag{6.17}$$

Como visto anteriormente, os quatro primeiros elementos do polinômio representam a paridade e os outros três elementos representam a mensagem a ser transmitida. Um símbolo de paridade foi corrompido com 1-bit de erro (α^2), e um símbolo de dados corrompido com 3 bits de erro (α^5). O polinômio da *codeword* recebida com erro $r(X)$ é então representada pela soma da *codeword* transmitida com o polinômio de erro padrão como segue:

$$r(X) = U(X) + e(X) \tag{6.18}$$

Seguindo a Equação 6.18, soma-se $U(X)$ da Equação 6.15 com $e(X)$ da Equação 6.17 para produzir

$$\begin{aligned} r(X) &= (100) + (001)X + (011)X^2 + (100)X^3 + (101)X^4 + (110)X^5 + (111)X^6 \\ &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^0 X^3 + \alpha^6 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \end{aligned} \tag{6.19}$$

Nesse exemplo de correção de erros de 2 símbolos, existem quatro elementos desconhecidos - duas localizações de erros e dois valores de erros. É importante notar a diferença entre decodificação binária e decodificação não binária, na primeira o decodificador só precisa encontrar a localização do erro. Sabendo que existe um erro em uma localização particular, basta trocar o bit de 1 para 0 ou vice-versa. Símbolos não binários necessitam não somente saber a localização do erro, mas também os valores dos símbolos corretos nas suas localizações [34]. Uma vez que existem quatro elementos desconhecidos nesse exemplo, quatro equações são necessárias para sua solução.

6.3.1 Cálculo da Síndrome

A síndrome é o resultado da verificação de paridade executada em$r$ para determinar se r é um membro válido do conjunto de *codeword* [4]. Se de fato r é um membro, então a síndrome S tem o valor 0. Qualquer valor de S diferente de zero indica a presença de erros. Similarmente ao caso binário, a síndrome S é composta de $n-k$ símbolos, $\{S_i\}(i = 1, ..., n-k)$. Então, para o código RS(7,3) existem quatro símbolos que compõe o vetor de síndromes; esses valores podem ser calculados a partir do polinômio recebido $r(X)$. Para esse exemplo, os quatro símbolos de síndrome são encontrados como segue:

$$\begin{aligned} S_1 = r(\alpha) &= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^{10} + \alpha^8 + \alpha^{11} \\ &= \alpha^0 + \alpha^3 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha^1 + \alpha^4 \\ &= \alpha^3 \end{aligned} \tag{6.20}$$

$$\begin{aligned} S_2 = r(\alpha^2) &= \alpha^0 + \alpha^4 + \alpha^8 + \alpha^6 + \alpha^{14} + \alpha^{13} + \alpha^{17} \\ &= \alpha^0 + \alpha^4 + \alpha^1 + \alpha^6 + \alpha^0 + \alpha^6 + \alpha^3 \\ &= \alpha^5 \end{aligned} \tag{6.21}$$

$$\begin{aligned} S_3 = r(\alpha^3) &= \alpha^0 + \alpha^5 + \alpha^{10} + \alpha^9 + \alpha^{18} + \alpha^{18} + \alpha^{23} \\ &= \alpha^0 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha^4 + \alpha^4 + \alpha^2 \\ &= \alpha^6 \end{aligned} \tag{6.22}$$

$$\begin{aligned} S_4 = r(\alpha^4) &= \alpha^0 + \alpha^6 + \alpha^{12} + \alpha^{12} + \alpha^{22} + \alpha^{23} + \alpha^{29} \\ &= \alpha^0 + \alpha^6 + \alpha^5 + \alpha^5 + \alpha^1 + \alpha^2 + \alpha^1 \\ &= 0 \end{aligned} \tag{6.23}$$

O resultado confirma que a *codeword* recebida contém um erro (aqui inseridos) desde que $S \neq 0$.

6.3.2 Localização dos Erros

Supondo que existam v erros na *codeword* na localização $X^{j_1}, X^{j_2}, ...X^{j_v}$. Então o polinômio de erro mostrado nas Equações 6.16 e 6.17 pode ser escrito como

$$e(X) = e_{j_1}X^{j_1} + e_{j_2}X^{j_2} + ... + e_{j_v}X^{j_v} \tag{6.24}$$

Os índices 1,2,...,v indicam o $1^{st}, 2^{nd}, ..., vth$ erro, e o índice j indica a localização do erro. Para corrigir a *codeword* corrompida, cada valor de erro e_{j_l} e sua localização X^{j_l}, onde $l = 1, 2, ...v$ deve ser determinada. Foi definido um número localizador de erros como$\beta_l = \alpha^{j_l}$. No próximo passo, são obtidos os $n - k = 2t$ símbolos de síndrome substituindo α^i no polinômio recebido para $i = 1, 2, ...2t$:

$$\begin{aligned} S_1 &= r(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + ... + e_{j_v}\beta_v \\ S_2 &= r(\alpha^2) = e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + ... + e_{j_v}\beta_v^2 \\ &... \\ S_{2t} &= r(\alpha^{2t}) = e_{j_1}\beta_1^{2t} + e_{j_2}\beta_2^{2t} + ... + e_{j_v}\beta_v^{2t} \end{aligned} \tag{6.25}$$

Existem $2t$ elementos desconhecidos (t valores de erros e t localizações), e $2t$ equações simultâneas. No entanto, essas $2t$ equações não podem ser resolvidas de modo usual por serem não-lineares. Uma técnica que resolva esse sistema de equações é conhecida como algoritmo de decodificação Reed-Solomon.

Quando um vetor de síndromes diferente de zero (um ou mais dos seus símbolos são diferentes de zero) é corrompido [4], significa que um erro foi recebido. Para isso, é necessário descobrir a localização do erro ou erros. Um polinômio localizador de erro pode ser definido como

$$\begin{aligned} \sigma(X) &= (1 + \beta_1 X)(1 + \beta_2 X)...(1 + \beta_v X) \\ &= 1 + \sigma_1 X + \sigma_2 X^2 + ... + \sigma_v X^v \end{aligned} \tag{6.26}$$

As raízes de $\sigma(X)$ são $1/\beta_1$, $1/\beta_2$,...,$1/\beta_v$. A recíproca das raízes de $\sigma(X)$ são os números de localização de erros padrão $e(X)$. Usando técnica de modelagem auto-regressiva é formada uma matriz de síndromes, onde as primeiras t síndromes são usadas para prever a próxima síndrome. Que é

$$\begin{bmatrix} S_1 & S_2 & S_3 & ... & S_{t-1} & S_t \\ S_2 & S_3 & S_4 & ... & S_t & S_{t+1} \\ & & & ... & & \\ S_{t-1} & S_t & S_{t+1} & ... & S_{2t-3} & S_{2t-2} \\ S_t & S_{t+1} & S_{t+2} & ... & S_{2t-2} & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_t \\ \sigma_{t-1} \\ ... \\ \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{t+1} \\ -S_{t+2} \\ ... \\ -S_{2t-1} \\ -S_{2t} \end{bmatrix} \tag{6.27}$$

Para o código RS(7,3) corretor de erros de duplos símbolos, o tamanho da matriz é2×2 e o modelo é escrito como

$$\begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} S_3 \\ S_4 \end{bmatrix} \tag{6.28}$$

$$\begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} \alpha^6 \\ \alpha^0 \end{bmatrix} \tag{6.29}$$

Para resolver os coeficientes σ_1 e σ_2 do polinômio localizador de erros $\sigma(X)$, primeiramente calcula-se a inversa da matriz da Equação 6.29. A inversa de uma matriz [A] é encontrada como segue:

$$Inv[A] = \frac{cofator[A]}{det[A]}$$

Portanto,

$$\begin{aligned} det \begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \alpha^3\alpha^6 - \alpha^5\alpha^5 = \alpha^9 + \alpha^{10} \\ = \alpha^2 + \alpha^3 = \alpha^5 \end{aligned} \tag{6.30}$$

$$cofator \begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix} \tag{6.31}$$

e

$$Inv \begin{bmatrix} \alpha^3 & \alpha^5 \\ \alpha^5 & \alpha^6 \end{bmatrix} = \frac{\begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix}}{\alpha^5} = a^{-5} \begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix} \tag{6.32}$$

$$= \alpha^2 \begin{bmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha^3 \end{bmatrix} = \begin{bmatrix} \alpha^8 & \alpha^7 \\ \alpha^7 & \alpha^5 \end{bmatrix} = \begin{bmatrix} \alpha^1 & \alpha^0 \\ \alpha^0 & \alpha^5 \end{bmatrix} \tag{6.33}$$

Para verificar se a inversão foi feita corretamente, multiplica-se a matriz original pela matriz invertida, o resultado deve ser uma matriz identidade.

Continuando a Equação 6.29, inicia-se a pesquisa para localizar os erros pela resolução dos coeficientes do polinômio localizador de erro $\sigma(X)$ como segue:

$$\begin{bmatrix} \sigma^2 \\ \sigma^1 \end{bmatrix} = \begin{bmatrix} \alpha^1 & \alpha^0 \\ \alpha^0 & \alpha^5 \end{bmatrix} \begin{bmatrix} \alpha^6 \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha^7 \\ \alpha^6 \end{bmatrix} = \begin{bmatrix} \alpha^0 \\ \alpha^6 \end{bmatrix} \tag{6.34}$$

Das Equações 6.26 e 6.34 tem-se:

$$\begin{aligned} \sigma(X) &= \alpha^0 + \sigma_1 X + \sigma_2 X^2 \\ &= \alpha^0 + \alpha^6 X + \alpha^0 X^2 \end{aligned} \tag{6.35}$$

As raízes de $\sigma(X)$ são recíprocas as localizações dos erros. Uma vez encontradas, a localização do erro passa a ser conhecida. Em geral, as raízes de $\sigma(X)$ podem ser um ou mais elementos do corpo. Nós determinamos essas raízes por teste exaustivo do polinômio $\sigma(X)$ com cada um de seus elementos, como mostrado na seqüência. Qualquer elemento X que resultar em $\sigma(X) = 0$ é uma raíz no qual permite a localização de um erro:

$$\begin{aligned}
\sigma(\alpha^0) &= \alpha^0 + \alpha^6 + \alpha^0 = \alpha^6 \neq \mathbf{0} \\
\sigma(\alpha^1) &= \alpha^2 + \alpha^7 + \alpha^0 = \alpha^2 \neq \mathbf{0} \\
\sigma(\alpha^2) &= \alpha^4 + \alpha^8 + \alpha^0 = \alpha^6 \neq \mathbf{0} \\
\sigma(\alpha^3) &= \alpha^6 + \alpha^9 + \alpha^0 = \mathbf{0} \Rightarrow \mathbf{ERRO} \\
\sigma(\alpha^4) &= \alpha^8 + \alpha^{10} + \alpha^0 = \mathbf{0} \Rightarrow \mathbf{ERRO} \\
\sigma(\alpha^5) &= \alpha^{10} + \alpha^{11} + \alpha^0 = \alpha^2 \neq \mathbf{0} \\
\sigma(\alpha^6) &= \alpha^{12} + \alpha^{12} + \alpha^0 = \alpha^0 \neq \mathbf{0}
\end{aligned}$$

Como visto na Equação 6.26, as localizações dos erros são a inversa das raízes do polinômio. Então, $\sigma(\alpha^3) = 0$ indica que uma raíz existe em $1/\beta_l = \alpha^3$. Logo, $\beta_l = 1/\alpha^3 = \alpha^4$. Similarmente, $\sigma(\alpha^4) = 0$ indica que outra raíz existe em $1/\beta_{l'} = 1/\alpha^4 = \alpha^3$, onde (para esse exemplo) l e l' referem-se ao 1^{st} e 2^{nd} erro. Como existem 2 símbolos de erros aqui, o polinômio de erro é configurado da seguinte forma:

$$e(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} \tag{6.36}$$

Os dois erros foram encontrados nas localizações α^3 e α^4. Note que o índice do número de localização do erro é completamente arbitrário. Então, para esse exemplo, podem ser escolhidos os valores $\beta_l = \alpha^{j_l}$ como $\beta_1 = \alpha^{j_1} = \alpha^3$ e $\beta_2 = \alpha^{j_2} = \alpha^4$.

6.3.3 Valores dos Erros

Um erro é denotado por e_{j_l}, onde o índice j refere-se a localização do erro e o índice l identifica o lth erro. Uma vez que cada valor de erro está associado a uma localização particular, a notação e_{j_l} pode ser simplificada por e_l. Agora, preparando para determinar os valores de erro e_1 e e_2, associado com as localizações $\beta_1 = \alpha^3$ e $\beta_2 = \alpha^4$, qualquer das quatro equações de síndrome pode ser usadas. Da Equação 6.25, são usados S_1 e S_2:

$$\begin{aligned}
S_1 = r(\alpha) &= e_1\beta_1 + e_2\beta_2 \\
S_2 = r(\alpha^2) &= e_1\beta_1^2 + e_2\beta_2^2
\end{aligned} \tag{6.37}$$

Nós podemos escrever essas equações em forma de matriz como segue:

$$\begin{bmatrix} \beta_1 & \beta_2 \\ \beta_1^2 & \beta_2^2 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} \tag{6.38}$$

$$\begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^8 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} \alpha^3 \\ \alpha^5 \end{bmatrix} \tag{6.39}$$

Para encontrar os valores de erro e_1 e e_2, a matriz na Equação 6.39 é invertida da mesma forma, resultando em:

$$Inv \begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & \alpha^1 \end{bmatrix} = \frac{\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix}}{\alpha^3\alpha^1 - \alpha^6\alpha^4}$$

$$= \frac{\begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix}}{\alpha^4 + \alpha^3} = \alpha^{-6} \begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix} = \alpha^1 \begin{bmatrix} \alpha^1 & \alpha^4 \\ \alpha^6 & \alpha^3 \end{bmatrix} \tag{6.40}$$

$$= \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^7 & \alpha^4 \end{bmatrix} = \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^0 & \alpha^4 \end{bmatrix}$$

Agora, resolvendo a Equação 6.39 para os valores de erros tem-se:

$$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} \alpha^2 & \alpha^5 \\ \alpha^0 & \alpha^4 \end{bmatrix} \begin{bmatrix} \alpha^3 \\ \alpha^5 \end{bmatrix} = \begin{bmatrix} \alpha^5 + \alpha^{10} \\ \alpha^3 + \alpha^9 \end{bmatrix} = \begin{bmatrix} \alpha^5 + \alpha^3 \\ \alpha^3 + \alpha^2 \end{bmatrix} = \begin{bmatrix} \alpha^2 \\ \alpha^5 \end{bmatrix} \tag{6.41}$$

6.3.4 Correção do Polinômio Recebido

O polinômio de erros é formado a partir das Equações 6.36 e 6.41, resultando no polinômio apresentado na Equação 6.42.

$$\begin{aligned} \bar{e}(X) &= e_1 X^{j_1} + e_2 X^{j_2} \\ &= \alpha^2 X^3 + \alpha^5 X^4 \end{aligned} \tag{6.42}$$

O algoritmo demonstrado repara o polinômio recebido, resultando em uma estimativa da *codeword* transmitida, recuperando a mensagem original transmita. A Equação 6.43 apresenta o processo de restauração da *codeword* transmitida.

$$\begin{aligned} \bar{U}(X) &= r(X) + \bar{e}(X) = U(X) + e(X) + \bar{e}(X) \\ r(X) &= (100) + (001)X + (011)X^2 + (100)X^3 + (101)X^4 + (110)X^5 + (111)X^6 \\ \bar{e}(X) &= (000) + (000)X + (000)X^2 + (001)X^3 + (111)X^4 + (000)X^5 + (000)X^6 \\ \bar{U}(X) &= (100) + (001)X + (011)X^2 + (101)X^3 + (010)X^4 + (110)X^5 + (111)X^6 \\ &= \alpha^0 + \alpha^2 X + \alpha^4 X^2 + \alpha^6 X^3 + \alpha^1 X^4 + \alpha^3 X^5 + \alpha^5 X^6 \end{aligned} \tag{6.43}$$

Nesse capítulo foram apresentados os cálculos envolvidos nos processos de codificação e decodificação de telemetria utilizando a classe de códigos corretores de erros Reed-Solomon. No decorrer do capítulo é apresentado um exemplo de codificação/decodificação e correção da telemetria corrompida durante o processo de transmissão da mesma. O Capítulo 7 apresenta os pacotes de telemetria e telecomando segundo o padrão CCSDS. Nele são apresentados todos os campos de informações presentes em cada pacote juntamente com a aplicação dos algoritmos BCH e RS em cada um deles.

Capítulo 7

Telemetria e Telecomando no Contexto Espacial

O *Consultative Committee for Space Data Systems* (CCSDS) é uma organização oficialmente estabelecida pelas principais agências espaciais do mundo. Esse comitê se reúne periodicamente para discutir problemas comuns, ocorridos em sistemas de comunicação, a todos os participantes do comitê, e formular soluções técnicas seguras para esses problemas. Uma vez que a participação no CCSDS é completamente voluntária, os resultados das ações tomadas pelo comitê são designadas recomendações e não são consideradas obrigatórias em qualquer agência espacial.

O CCSDS publicou algumas recomendações de formato de dados e métodos para transmissão de telemetria e telecomando. A recomendação do pacote de telemetria define o padrão de unidades de dados para transferência de telemetria de um veículo espacial para a estação terrestre. A recomendação de telecomando define o padrão de unidade de dados para enviar comandos da estação terrestre para o veículo espacial. Esse Capítulo tem por objetivo apresentar os quadros de telemetria e telecomando estabelecidos por esse padrão e utilizados para o desenvolvimento do presente trabalho. Na Seção 7.1 são apresentados dados referentes ao*frame* de telemetria CCSDS enquanto a Seção 7.2 apresenta o conjunto de informações inseridas no*frame* de telecomando.

Basicamente, o fluxo de envio de telecomando e recebimento de telemetria na base terrestre é ilustrado na Figura 7.1. Inicialmente o*frame* a ser transmitido é colocado dentro do pacote no padrão CCSDS. Após essa etapa o telecomando é codificado utilizando o algoritmo de correção de erros BCH através de um módulo codificador presente na estação terrestre. A partir de então o telecomando é transmitido para o segmento espacial, o qual contém um módulo responsável pela decodificação do telecomando recebido. Esse módulo é responsável por corrigir os possíveis erros ocorridos durante a transmissão e está implementado em hardware em um dispositivo reconfigurável do tipo FPGA.

Após a decodificação, o*frame* é processado e as ações definidas pelo telecomando são disponibilizadas para execução nos diferentes módulos do segmento espacial. No caso da telemetria, antes do envio de um pacote, o*frame* é codificado utilizando o algoritmo RS. Esse módulo está implementado também na forma de um SoC, presente em um dispositivo FPGA. O*frame* de telemetria é então enviado para a estação terrestre que possui um módulo RS responsável pela decodificação do *frame* de telemetria recebido. Esse trabalho apresenta uma proposta de implementação do codificador RS em hardware reconfigurável para codificação de telemetria e do algoritmo BCH para decodificação do telecomando, ambos de acordo com o padrão CCSDS.

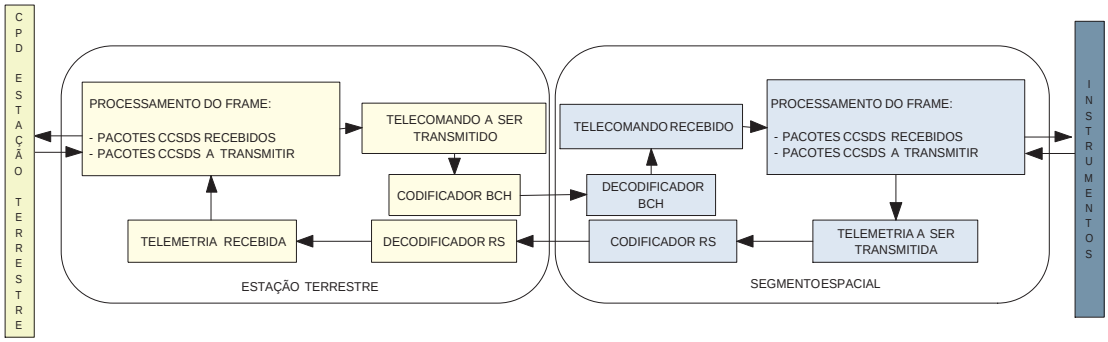

Figura 7.1: Comunicação entre a estação terrestre e o segmento espacial

7.1 Modelo de Serviço de Telemetria

O propósito da telemetria é prover o estado atual do veículo espacial bem como os dados de plataforma do mesmo. Nesta seção, algumas convenções e definições são usadas para representar os dados no padrão CCSDS.

a. O primeiro bit no campo a ser transmitido (bit mais a esquerda) representado na Figura 7.2 é definido como bit '0'; o próximo bit é definido como bit '1' e o último como sendo bit 'N-1'. Quando o campo é usado para expressar valores binários (como contadores), o Bit Mais Significativo (MSB[1]) deve ser o primeiro bit transmitido do campo, por exemplo bit '0'.

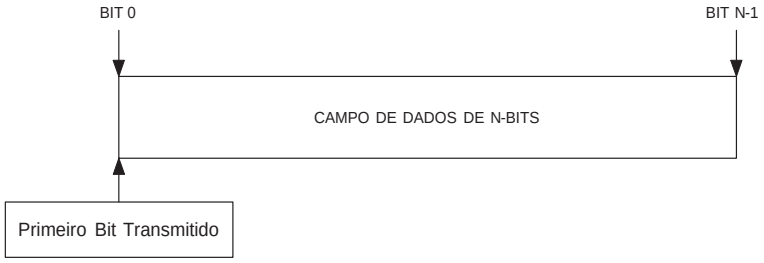

Figura 7.2: Convenção utilizada para numeração dos bits

b. De acordo com a prática da comunicação moderna, campos de dados de veículos espaciais são freqüentemente agrupados em palavras de 8 bits, as quais são chamadas de *bytes*.

c. A numeração dos bytes dentro da estrutura começa com o valor 0.

7.1.1 Formato do *Frame* de Telemetria

A essência do pacote de telemetria é permitir múltiplos processos de aplicação rodar em fontes *on-board* diferentes, a fim de criar unidades de dados com melhor ajuste para cada fonte. Uma fonte poder ser apresentada como sendo um instrumento ou sistema presente no veículo espacial.

[1] do inglês Most Significant Bit

Assim sendo, é possível fazer com que o sistema de bordo transmita as unidades de dados sobre um canal de comunicação espaço-terra, de modo que o sistema terrestre receba as unidades de forma individual com alta confiabilidade, repassando-as para os processos destino em terra em seqüência. Essas fontes *on-board* podem ser instrumentos, como por exemplo, sensores paca captura de dados científicos, estado atual do veículo espacial ou demais sub-sistemas.

O modelo de serviço de telemetria é dado por um protocolo composto de7 camadas: *aplicação, gerência do sistema, empacotamento, segmentação, transferência, codificação e física* A Figura 7.3 ilustra as camadas e os serviços prestados por cada uma.

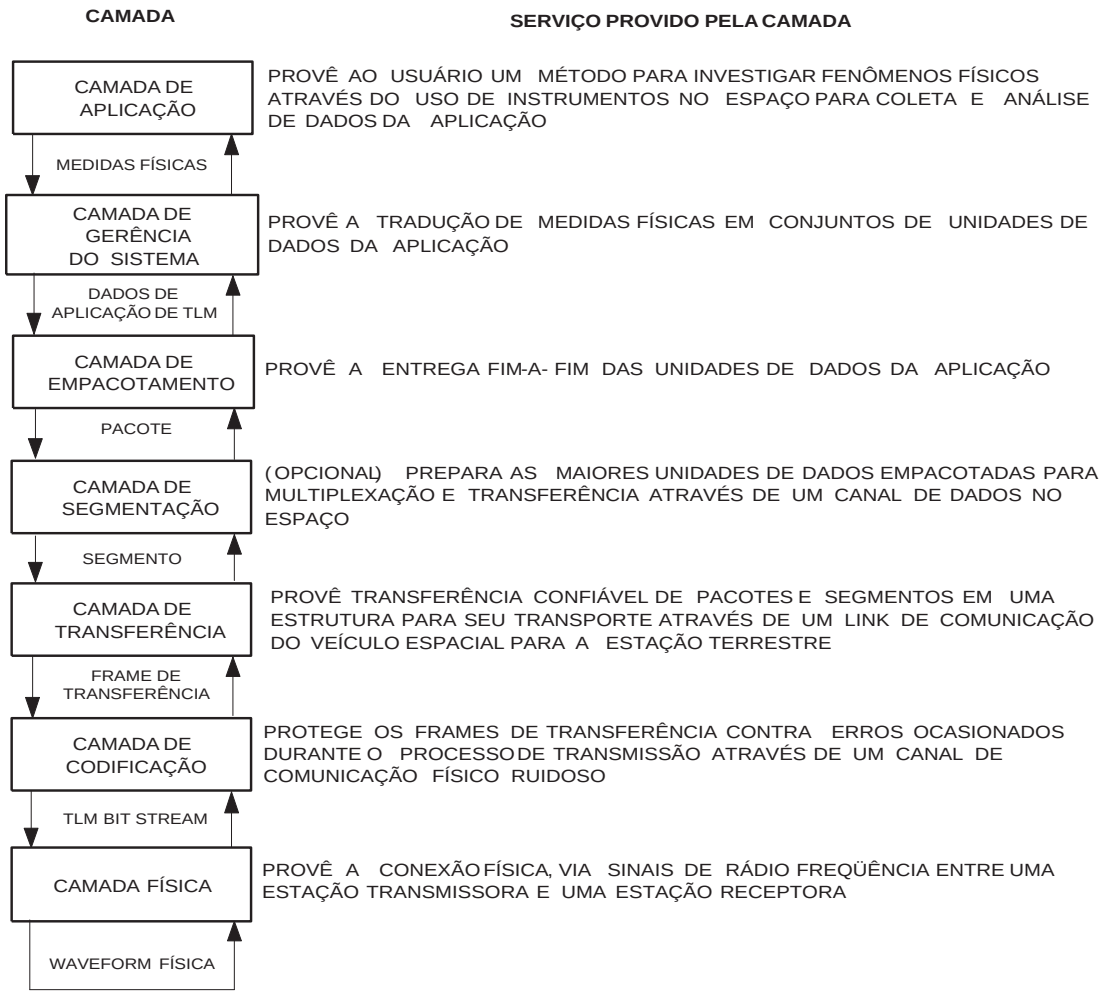

Figura 7.3: Modelo de serviço da telemetria em camadas

A camada de aplicação provê ao usuário um método para investigar fenômenos físicos através do uso de instrumentos no espaço para coleta e análise de dados da aplicação. Logo abaixo encontra-se a camada de gerência do sistema, a qual provê a tradução de medidas físicas em conjuntos de unidades de dado de aplicação. A camada de empacotamento fornece a entrega fim-a-fim das unidades de dados de aplicação. Como serviço opcional, a camada de segmentação prepara as maiores unidades de dados empacotadas para multiplexação e transferência através

de um canal de dados no espaço. A camada de transferência provê transferência confiável de pacotes e segmentos em uma estrutura para seu transporte através de um link de comunicação do veículo espacial para a estação terrestre. O *frame* de transferência é então repassado para a camada de codificação, responsável por proteger os *frames* contra erros ocasionados durante o processo de transmissão através de um canal de comunicação físico ruidoso. Por fim, e no nível mais baixo da pilha de camadas de telemetria, encontra-se a camada física, a qual provê a conexão física via sinais de radio freqüência entre a estação transmissora e a estação receptora.

Para o trabalho, foram implementadas as 3 camadas consideradas interessantes para uma funcionalidade básica de um sistema de telemetria no padrão CCSDS: camada de empacotamento, transferência e codificação. A Figura 7.4 apresenta o formato do pacote na camada de empacotamento. A seguir são apresentados os campos que compõe o pacote da fonte.

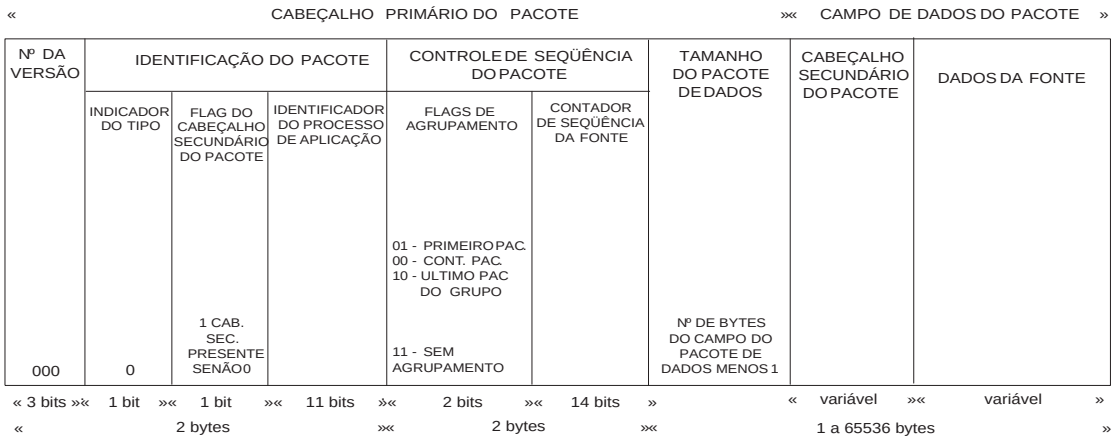

Figura 7.4: Formato do pacote da fonte

1. *Cabeçalho Primário do Pacote*

   a. O *cabeçalho primário do pacote* é obrigatório e deve consistir de quatro campos, posicionados na seguinte seqüência:

	Tamanho em bits
Número de Versão	3
Identificação do Pacote	13
Controle de Seqüência do Pacote	16
Tamanho do Pacote de Dados	16

Número de Versão

   a. O *número de versão* deve estar contido dentro do cabeçalho primário do pacote, o campo possui tamanho de 3 bits.

   b. O campo de 3 bits identifica a unidade de dados do pacote da fonte e deve ser fixado em "000".

      O número de versão é usado para reservar a possibilidade de introdução de outras estruturas de dados.

Identificação do Pacote

a. O campo de identificação do pacote é representado pelos bits de3 a 15 do cabeçalho primário do pacote.

b. Os 13 bits do campo devem ser divididos em 3 sub-campos:

	Tamanho em bits
Indicador do Tipo	1
Flag do Cabeçalho Secundário do Pacote	1
Identificador do Processo de Aplicação	11

A identificação do pacote verifica o tipo de pacote (pacote da fonte de telemetria), indica se o pacote carrega um cabeçalho secundário ou não e fornece informação na fonte de dados, por ex. o processo de aplicação.

Indicador do Tipo

a. O bit 3 do cabeçalho primário do pacote contém o indicador do tipo que fornece o tipo de unidade de dados.

b. O indicador de tipo deve ser fixado em '0'.

Devido o fato do telecomando CCSDS usar uma estrutura de dados similar, o indicador de tipo diferencia unidade de dados de telecomando e telemetria (para pacotes de telecomando o indicador de tipo deve ser fixado em '1'.

Flag do Cabeçalho Secundário do Pacote

a. O bit 4 do cabeçalho primário do pacote contém o flag do cabeçalho secundário do pacote.

b. O flag indica a presença ou ausência de cabeçalho secundário do pacote dentro de seu pacote da fonte. Ele deve conter '1' se o cabeçalho secundário do pacote se faz presente e '0' caso contrário.

c. O flag é estático com relação ao identificador do processo de aplicação durante toda a missão.

d. O flag é fixado em '0' para pacotes ociosos.

Identificador do Processo de Aplicação

a. Os bits de 5 a 15 do cabeçalho primário do pacote contém o identificador do processo de aplicação.

b. O identificador deve ser diferente para cada processo presente no mesmo canal mestre.

c. Para pacotes ociosos o identificador do processo de aplicação deve conter todos os bits em '1', por exemplo, "11111111111111".

Controle de Seqüência do Pacote

a. O campo de controle de seqüência do pacote é representado pelos bits 16 a 31 no cabeçalho primário do pacote.

b. Os 16 bits do campo são subdivididos em 2 sub-campos como segue:

	Tamanho em bits
Flags de Agrupamento	2
Contador de Seqüência da Fonte	14

O campo de controle fornece um contador seqüencial de pacotes gerados com o mesmo identificador do processo de aplicação, se o recurso de agrupamento for aplicado, fornece informações da posição de um pacote da fonte em um grupo.

Flags de Agrupamento

a. Os bits 16 e 17 do cabeçalho primário do pacote contém os flags de agrupamento.
b. Os flags de agrupamento são compostos de:
   - "01" para o primeiro pacote da fonte de um grupo;
   - "00" para um pacote da fonte contínuo de um grupo;
   - "10" para o último pacote da fonte de um grupo;
c. Para o pacote da fonte não presente em qualquer grupo de pacotes, o flag de agrupamento deve consistir de "11".
d. Todos os pacotes pertencentes a um grupo de pacotes específico devem ser originados de um mesmo processo de aplicação identificado por um único identificador de processo.

Contador de Seqüência da Fonte

a. Os bits 18 a 31 do cabeçalho primário do pacote contém o contador de de seqüência da fonte.
b. O contador de seqüência fornece um contador binário seqüencial de cada pacote gerado por um processo de aplicação identificado por um identificador de processo exclusivo.
c. O contador de seqüência deve ser contínuo, módulo 16384 (2^{14}).
d. Para pacotes ociosos não é necessário incrementar o contador de seqüência.

O objetivo desse campo é ordenar os pacotes gerados pelo mesmo processo de aplicação, embora sua ordem possa ser modificada durante a transmissão para os processos na estação terrestre.

Tamanho do Pacote de Dados

a. O campo de tamanho do pacote é representado pelos bits de 32 a 47 do cabeçalho primário do pacote.
b. O campo de 16 bits contém um número binário igual ao número de octetos no campo de dados do pacote menos 1.
c. O valor contido no campo de tamanho do pacote pode ser variável e estar numa faixa de valores compreendidos entre 0 e 65535, correspondendo 1 a 65536 octetos.

2. Campo de Dados do Pacote

a. O campo de dados do pacote segue, sem intervalo, o cabeçalho primário do pacote.
b. O campo de dados é obrigatório e consiste de no mínimo um dos dois campos, posicionados em seqüência como segue:

	Tamanho em bits
Cabeçalho Secundário do Pacote	variável
Campo de Dados da Fonte	variável

c. O campo de dados do pacote deve conter no mínimo um octeto.

Cabeçalho Secundário do Pacote

a. Quando presente, o cabeçalho secundário do pacote segue, sem intervalo, o campo de tamanho do pacote de dados.

b. O cabeçalho secundário é obrigatório se o campo de dados da fonte está presente, caso contrário ele é opcional. A presença ou ausência do cabeçalho secundário é assinalada por um flag no campo de identificação do pacote, conforme apresentado anteriormente.

c. Quando presente, o cabeçalho secundário deve consistir de:

   - um campo de dados do cabeçalho secundário;
   - ou um campo de código de tempo;
   - ou um campo de código de tempo seguido de um campo de dados do cabeçalho secundário;

   A escolha da opção permanece estática para um identificador de processo de aplicação específico durante toda a missão.

   O propósito de um cabeçalho secundário é permitir (mas não exigir) um meio definido pelo CCSDS para alocar dados como: tempo, dados internos, posição do veículo espacial, altitude, etc. com um pacote da fonte.

Campo de Dados da Fonte

a. Quando presente, o campo de dados da fonte segue, sem intervalo, o cabeçalho secundário do pacote (se o cabeçalho estiver presente) ou o campo de tamanho dos dados do pacote (se o cabeçalho secundário não estiver presente).

b. O campo de dados da fonte é obrigatório se o cabeçalho secundário do pacote não se fizer presente, caso contrário é opcional.

c. O campo de dados da fonte contem os dados da fonte do processo de aplicação ou dado ocioso.

d. O tamanho do campo de dados da fonte pode ser variável. Ele contém um número inteiro de octetos.

A camada de transferência de telemetria é definida em [35]. Nesta seção o*frame* de telemetria na camada de empacotamento foi descrito e os dados que compõe o mesmo detalhados separadamente por campo. O padrão CCSDS propõe esse formato de pacote e por isso as implementações apresentadas nesse trabalho utilizam esse padrão de pacote para dados de telemetria.

7.2 Modelo de Serviço de Telecomando

Da mesma forma como o modelo de serviço de telemetria, o modelo de serviço de telecomando é dado por um protocolo composto de7 camadas: *aplicação, gerência do sistema, empacotamento, segmentação, transferência, codificação e física* A Figura 7.5 ilustra as camadas e os serviços prestados por cada uma.

A camada de aplicação permite o usuário do sistema supervisionar processos remotos através da interface com os sistemas de telecomando espaciais. A camada de gerência do sistema converte as diretivas de comando do usuário em unidades de dados transportáveis e gerencia sua

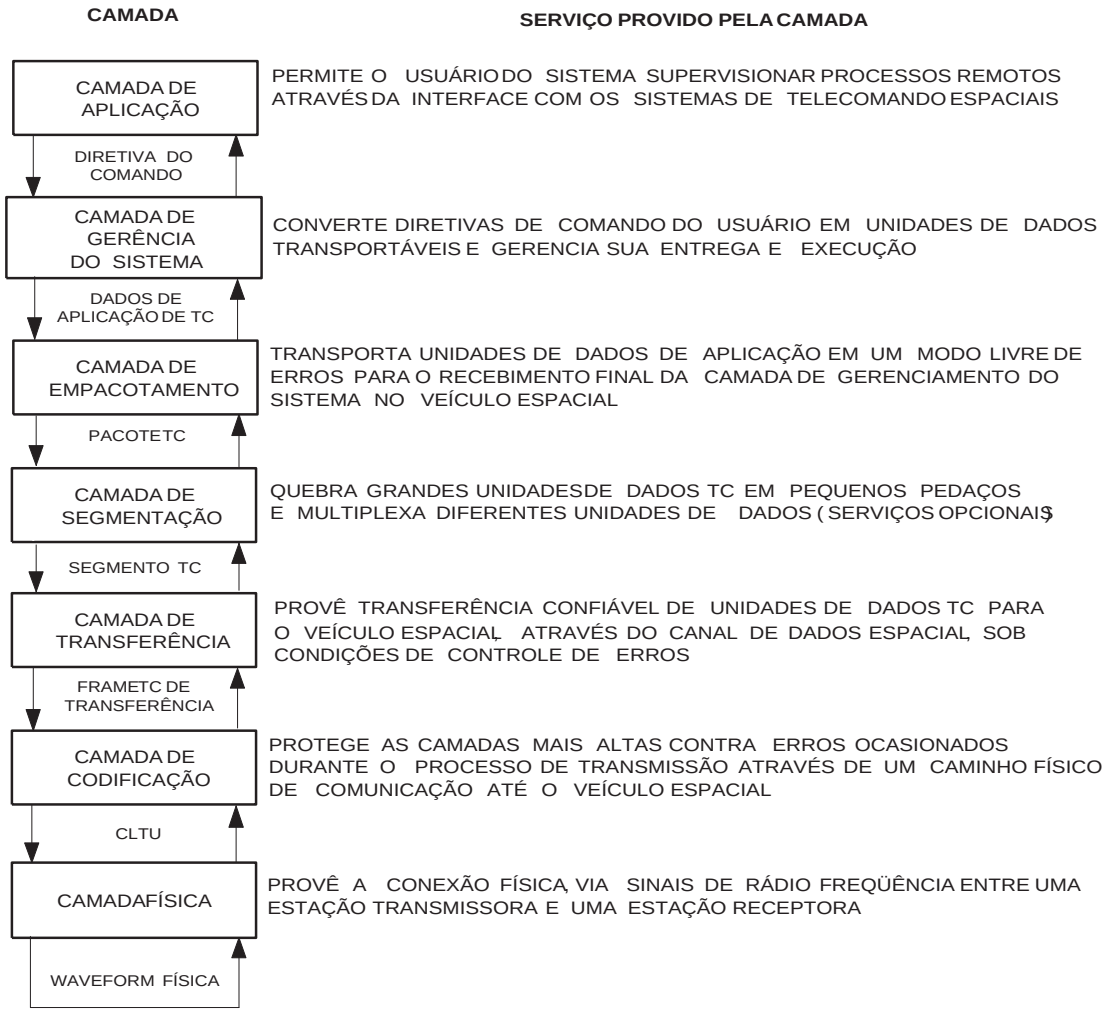

Figura 7.5: Modelo de serviço da telecomando em camadas

entrega e execução. A camada mais abaixo, chamada de camada de empacotamento, transporta unidades de dados de aplicação em um modo livre de erros para o recebimento final da camada de gerenciamento do sistema no veículo espacial. A camada responsável pela quebra de grandes unidades de dados em pequenos pedaços é chamada de camada de segmentação. A camada de transferência provê transferência confiável de unidades de dados TC para o veículo espacial, através do canal de dados espacial, sob condições de controle de erros. A camada de codificação, principal foco do presente trabalho, protege as camadas acima contra erros ocasionados durante o processo de transmissão através de um caminho físico de comunicação até o veículo espacial. Por fim, a camada física provê a conexão, via rádio freqüência entre a estação transmissora e a estação receptora de dados.

O serviço prestado pelo canal de telecomando permite um caminho de dados para tratamento de erros a ser estabelecido para transferência de telecomandos para o veículo espacial. O serviço contém duas camadas distintas de operações de transferência de dados:

1. CAMADA DE CODIFICAÇÃO, que permite o conjunto de bits de informação do telecomando a ser transmitido, ser mais confiável através de um canal de dados físico ruidoso, através do uso de técnicas de codificação de canal. A camada de codificação também fornece informação sobre o começo do conteúdo de *codeblocks* válidos e a continuidade da *stream* de dados, e repassa o conteúdo dos *codeblocks* para a camada acima.

2. CAMADA FÍSICA, que contém a freqüência de radio e as capacidades de modulação que podem ser utilizadas para estabelecer o canal físico de dados. A camada física também contém os procedimentos de operações da camada física (PLOPs[2] que provê os métodos de ativação e desativação do canal físico.

7.2.1 Camada de Codificação: Padrão de Estrutura de Dados e Procedimentos

A camada de codificação estabelece a confiabilidade do canal de dados através do uso de bits de dados de telecomando a serem transferidos. Os dados são codificados a fim de reduzir os efeitos de ruído na camada física do canal. Um bloco de código foi escolhido para fornecer essa proteção. A sincronização para o *codeblock* e a delimitação do começo de dados do usuário são fornecidos pela estrutura de dados CLTU (*Command Link Transmission Unit*).

7.3 Formato do *Codeblock* de TC

O formato do *codeblock* de TC é a estrutura de dados de comprimento fixo ilustrado na Figura 7.6. O *codeblock* é formado usando técnicas sistemáticas de codificação o qual contém56 bits de informação nos primeiros bytes (bytes mais a esquerda) e o controle de erros nos últimos bytes. O *codeblock* de TC contém 8 bytes, representando os 64 bits.

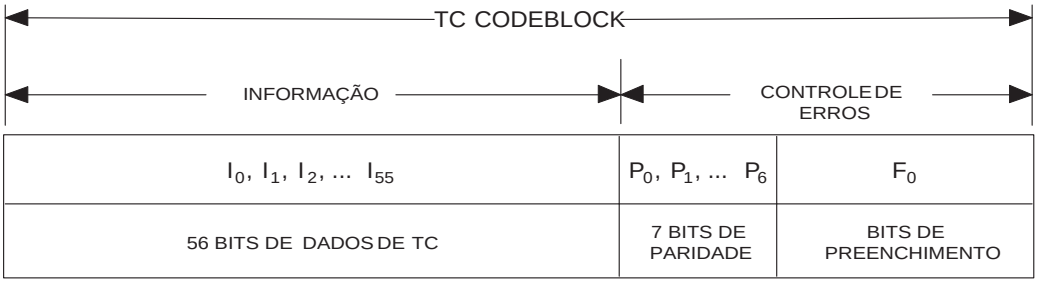

Figura 7.6: Formato do *codeblock* de TC

Na Figura 7.6 os símbolos $I_0, I_1, I_2, ...I_{55}$ representam os 56 bits de dados de telecomando a serem transmitidos, enquanto os símbolos $P_0, P_1, ...P_6$ representam os 7 bits de paridade anexados a mensagem de telecomando a ser enviada. O símbolo F_0 representa o bit de preenchimento, fazendo com que o *frame* de TC a ser transmitido possua comprimento fixo de 64 bits. Esse preenchimento é feito com a colocação do bit 0 para que o comprimento do *frame* de TC seja igual a 64.

[2] do inglês *Physical Layer Operations Procedures*

7.3.1 Formato do CLTU

A CLTU é uma estrutura de dados que carrega os dados de TC como uma série contínua de *codeblocks* de TC através do canal de serviço. Os dados de TC codificados dentro da CLTU consistem dos dados de entrada da camada acima. Os componentes que formam a CLTU são apresentados na Figura 7.7.

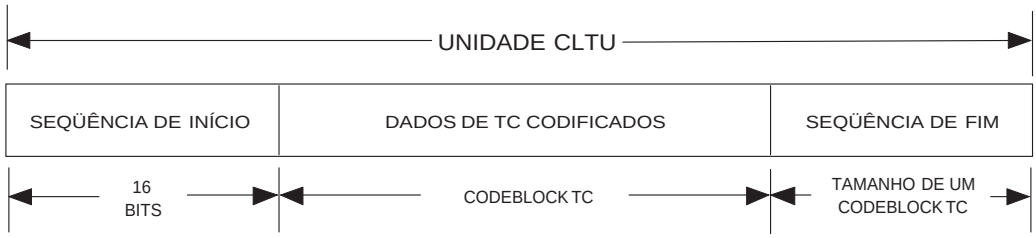

Figura 7.7: Componentes da CLTU

O campo de seqüência de início da CLTU delimita o começo dos dados de TC codificados dentro da CLTU. Consiste de um padrão de sincronização de 16 bits e deve ter o formato apresentado na Figura 7.8.

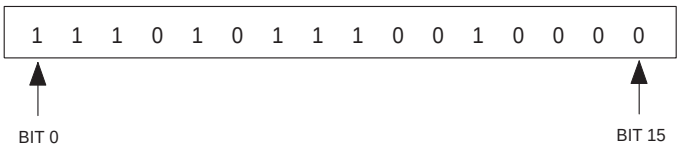

Figura 7.8: Seqüência de início da CLTU

O campo de dados de TC codificados consiste de um conjunto de *codeblock* de TC os quais foram codificados de acordo com o procedimento de codificação de TC apresentado anteriormente. Além dos bits de controle de erros, os *codeblocks* contém os dados de entrada para essa camada, mais o bit de preenchimento anexado ao *codeblock* a fim de preencher o *codeblock* de tal forma a ficar com 64 bits de comprimento. O campo de dados de TC codificados pode ter sido preenchido aleatoriamente antes da codificação ou não, como escolhido para a missão.

O campo de seqüência de fim é uma estrutura de dados que é construída especificamente para ser uma seqüência não corrigível que delimita o fim da CLTU fazendo com que o processo de decodificação seja parado. A seqüência de fim deve conter o mesmo tamanho do *codeblock* utilizado e deve conter o seguinte padrão 11000101, repetido sucessivamente até o último byte da seqüência de fim ser encontrado. O último byte completa o campo de seqüência de fim e sempre tem o padrão 01111001. Então, o padrão de bits que compõe a seqüência de fim deve ser descrito como:

11000101 11000101 11000101 11000101 11000101 11000101 11000101 011111001

Maiores detalhes sobre os pacotes de telecomando seguindo o padrão CCSDS encontram-se no documento [35]. Nesse capítulo foram apresentados os pacotes de telecomando e telemetria seguindo o padrão CCSDS. Para cada pacote são apresentados os campos que compõe o mesmo bem como as informações necessárias para a formação dos mesmos. O Capítulo 8 apresenta os

passos executados para a implementação e validação dos procedimentos de codificação e decodificação de telemetria e telecomando. Nele é apresentada a estratégia utilizada para levantamento de dados de polinômios bem como procedimentos e ferramentas usadas para processamento de informações necessárias para a implementação dos módulos em hardware.

Capítulo 8

Protótipo em Hardware para TM/TC

Esse Capítulo tem por objetivo apresentar os passos seguidos para implementação e validação dos módulos de telemetria e telecomando respectivamente. As funções da estação terrestre são implementadas em um computador pessoal e consistem, basicamente em:

- formatação de pacotes e *frames* TC;
- inserção de bits de paridade para correção de erros utilizando o código BCH;
- inserção de pacotes TC no campo de dados de *frames* TC;
- decodificação RS;
- recepção de *frames* TM;
- extração de pacotes TM do campo de dados de *frames* TM;
- decodificação da TM e, caso não existam erros, os dados de TM são consumidos;
- caso existam erros, o *frame* de TM é corrigido utilizando o algoritmo de Reed-Solomon.

As funções do seguimento espacial (módulo de bordo) foram implementadas totalmente em hardware na forma de núcleos IPs prototipados em um dispositivo FPGA. Algumas delas foram implementadas também implementadas em software para fins de validação do sistema. Essas funções consitem, basicamente, em:

- formatação de *frames* TM;
- cálculo dos símbolos de paridade utilizando o código Reed-Solomon e inserção dos mesmos no final dos *frame* de TM;
- inserção de pacotes TM no campo de dados de *frames* TM;
- recepção de *frames* TC;
- extração de pacotes TC do campo de dados de *frames* TC.
- decodificação do TC e, caso não existam erros, os dados de TC são consumidos;

- caso existam erros, o *frame* de TC é corrigido utilizando o algoritmo BCH.

A primeira implementação em software do algoritmo RS foi desenvolvida concorrentemente com atividades da segunda implementação, onde o codificador RS foi implementado em hardware na forma de um núcleo IP (*Intellectual Property*). Após a implementação e validação em software e em hardware do algoritmo RS, partiu-se para a implementação do algoritmo BCH, sendo o codificador em software e o decodificador em hardware, na forma de um núcleo IP. A implementação em software é uma boa alternativa para fins de teste e validação, devido a maior facilidade e agilidade na depuração e execução. O trabalho visa o projeto e implementação de módulos em hardware para codificação de telemetria e decodificação de telecomando seguindo o padrão CCSDS. Futuramente será considerada também uma implementação para ASIC (*Application Specific Integrated Circuit*), que poderá vir a ser utilizada em missões da AEB (Agência Espacial Brasileira). Essa implementação segue os padrões CCSDS para transmissão de pacotes de telemetria. A Seção 8.1 apresenta a estratégia utilizada para implementação e validação do codificador de telemetria em hardware, enquanto a Seção 8.2 apresenta a estratégia utilizada para codificação e decodificação de telecomando.

8.1 Implementação e Validação dos Módulos IPs para Telemetria CCSDS

Esta seção apresenta a metodologia de projeto utilizada no desenvolvimento e validação das camadas que compõem o protocolo para transmissão de telemetria seguindo o padrão CCSDS. O modelo de serviço para telemetria é composto de 7 camadas: *aplicação, gerência do sistema, empacotamento, segmentação, transferência, codificação e física* Entre elas, foram escolhidas as 3 camadas consideradas interessantes para uma funcionalidade básica de um sistema de telemetria no padrão CCSDS: *camadas de empacotamento, transferência e codificação* Para essas três camadas, foi gerado um circuito escrito em linguagem VHDL no qual é responsável por formatar os dados em estrutura de dados para as camadas de empacotamento e transferência, e codificá-los na camada de codificação, antes de serem enviados para a camada física.

8.1.1 Camada de Empacotamento

A camada de empacotamento é responsável por fornecer a entrega fim-a-fim das unidades de dados de aplicação. A Figura 8.1 apresenta a entidade da camada de empacotamento CCSDS implementada em linguagem VHDL. O circuito descrito é responsável pela formatação dos dados a serem enviados e é composto de 6 sinais de entrada e 2 de saída.

O sinal de *clock* é responsável pela sincronização do circuito implementado enquanto o sinal de *reset* serve para indicar a inicialização dos registradores e variáveis do módulo responsável pelo empacotamento. O sinal *data_in* representa os 64 bits de dados, sendo o sinal de *proc_id* composto dos 11 bits de identificação do processo de aplicação. O sinal *grouping_flag*, conforme o padrão CCSDS, indica se os dados a serem enviados representam o primeiro pacote ("01"), pacote de continuação ("00"), último pacote ("10") ou sem agrupamento ("11"), ele é usado para reagrupar os pacotes de telemetria enviados quando chegam ao seu destino. O sinal de *enable* indica que os dados estão disponíveis na entrada do circuito e pronto para serem formatados. Os outros dois sinais de saída representam quando o circuito acabou seu processamento, indicado pelo sinal de *ready* e os dados do pacote, representados pelo sinal *data_out*.

O circuito da camada de empacotamento CCSDS foi simulado através do software ModelSim e os dados de simulação são representados na Figura 8.2.

Figura 8.1: Entidade da camada de empacotamento de telemetria CCSDS

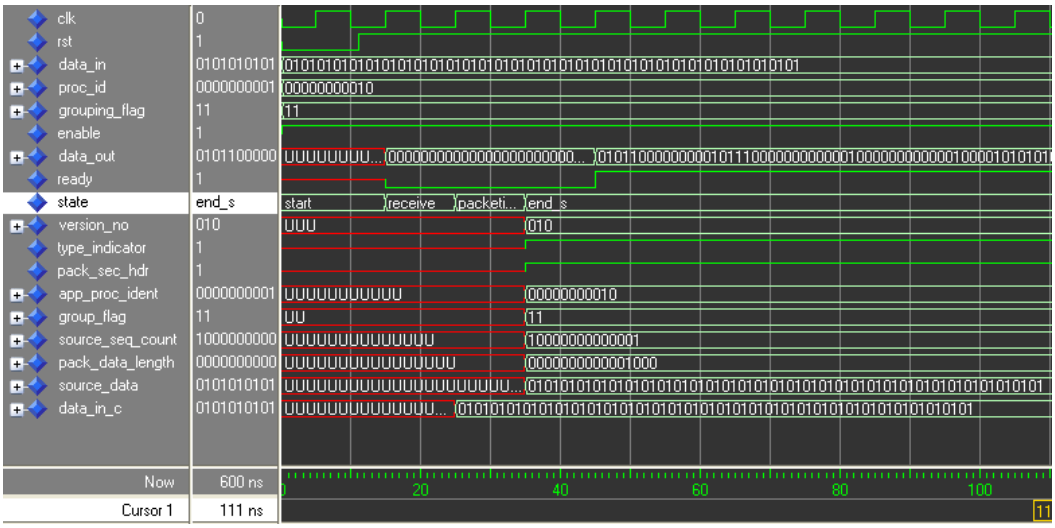

Figura 8.2: Representação dos dados de simulação da camada de empacotamento CCSDS

O circuito é capaz de formatar os dados de entrada e colocá-los em uma estrutura de dados o qual é apresentada pelo padrão CCSDS. A partir da validação da camada de empacotamento de dados partiu-se para a implementação e validação da camada de transferência de telemetria CCSDS.

8.1.2 Camada de Transferência

A camada de transferência é responsável por prover transferência confiável de pacotes e segmentos em uma estrutura para seu transporte através de um link de comunicação do veículo espacial para a estação terrestre. A Figura 8.3 apresenta a entidade da camada de transferência descrita em linguagem VHDL.

O circuito é composto de 7 sinais de entrada e 2 sinais de saída. O sinal de *clock* é responsável pela sincronização do circuito implementado enquanto o sinal de*reset* serve para indicar a inicial-

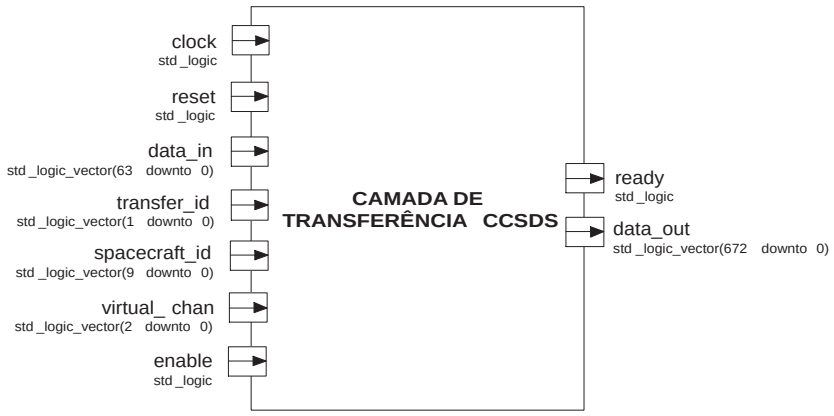

Figura 8.3: Entidade da camada de transferência de telemetria CCSDS

ização dos registradores e variáveis do módulo responsável pela camada de transferência. O sinal *data_ in* representa os 64 bits de dados, sendo o sinal *transfer_ id* dado pelo número de versão do *frame* de transferência. Segundo a recomendação CCSDS, o sinal deve ser setado para "00". O sinal *spacecraft_ id* é definido pelo padrão CCSDS e fornece a identificação do veículo espacial que criou o *frame* de dados. O identificador do veículo espacial deve ser estático durante todas as fases da missão. O sinal *virtual_ chan* fornece a identificação do canal virtual de dados enquanto o sinal *enable* sinaliza que os dados de entrada estão prontos para serem processados pelo circuito. Os dois sinais de saída representam quando o circuito acabou seu processamento, indicado pelo sinal de *ready* e os dados do *frame* de transferência, representados pelo sinal *data_ out*. O circuito de camada de transferência CCSDS foi simulado através do software ModelSim e os dados de simulação são representados através da Figura 8.4.

8.1.3 Camada de Codificação

A principal contribuição do trabalho se dá na investigação e implementação de códigos corretores de erros para telecomando e telemetria seguindo o padrão sugerido pelo CCSDS. A próxima etapa do trabalho consistiu na investigação, implementação e validação de módulos de correção de telemetria utilizando o algoritmo de Reed-Solomon e correção de telecomando utilizando o algoritmo BCH. Esta etapa iniciou com a validação do algoritmo RS, escrito em C, utilizando dois computadores pessoais conectados por intermédio de uma interface de comunicação RS-232C e sistema operacional Linux. A Figura 8.8 apresenta o modelo de validação para envio e recebimento de pacotes TC/TM respectivamente. O computador à direita representa o veículo espacial, o qual codifica o pacote de telemetria a ser enviado para a estação terrestre. O computador à esquerda tem por objetivo simular o funcionamento da estação terrestre, a qual decodifica o pacote de telemetria recebido.

A partir da validação em uma linguagem de alto nível do codificador/decodificador RS CCSDS, partiu-se para a validação do sistema em software para codificação/decodificação de telemetria executando em um processador embarcado Leon. A Figura 8.6 apresenta o segundo passo da validação do sistema, agora presente parte em software e parte em hardware. A plataforma de prototipação, com o processador Leon, simula o funcionamento do computador de bordo do veículo espacial, o qual codifica e decodifica o pacote de telemetria a ser enviado para a estação terrestre, enquanto que o PC rodando Linux, executa o papel de estação terrestre.

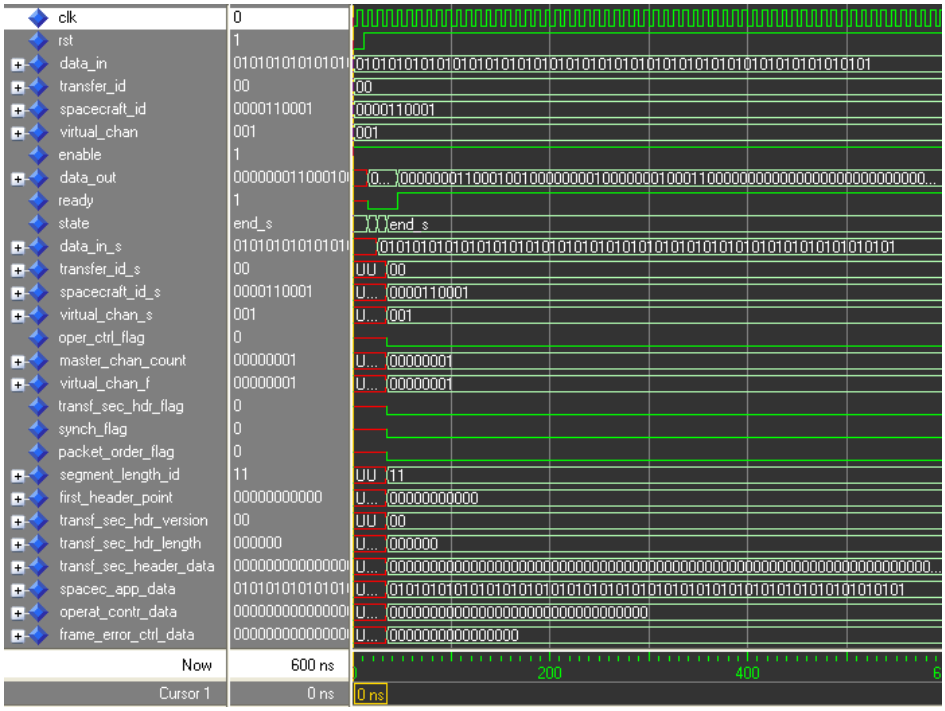

Figura 8.4: Representação dos dados de simulação da camada de transferência CCSDS

Figura 8.5: Codificador/Decodificador RS CCSDS escrito em C, transferindo dados entre dois PCs Linux

O terceiro passo da validação, conforme mostrado na Figura 8.7, consistiu na implementação do codificador RS CCSDS em VHDL na forma de núcleo IP, fazendo com que o mesmo seja executado em conjunto com um IP responsável pela comunicação serial entre o PC e o FPGA. O dado codificado é transferido entre um PC Linux e um FPGA configurado com o núcleo IP codificador Reed-Solomon e IP serial RS232-C. Para o módulo IP serial RS232-C foi reutilizado um IP responsável pela comunicação serial entre PC e FPGA desenvolvido no grupo de pesquisa GAPH [36] da PUCRS.

Para a descrição em hardware do circuito codificador de telemetria foram executados passos intermediários objetivando a obtenção de dados de polinômios para o circuito. Inicialmente,

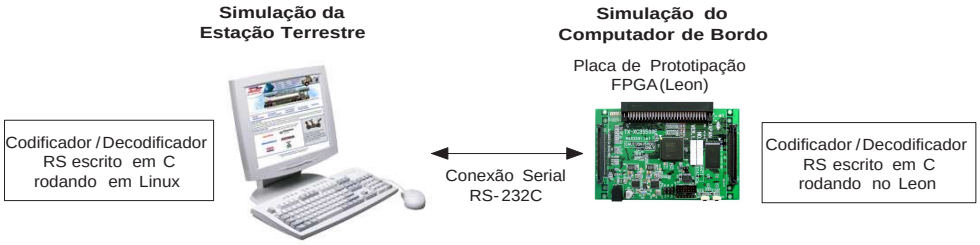

Figura 8.6: Codificador/Decodificador RS CCSDS escrito em C, rodando no processador Leon

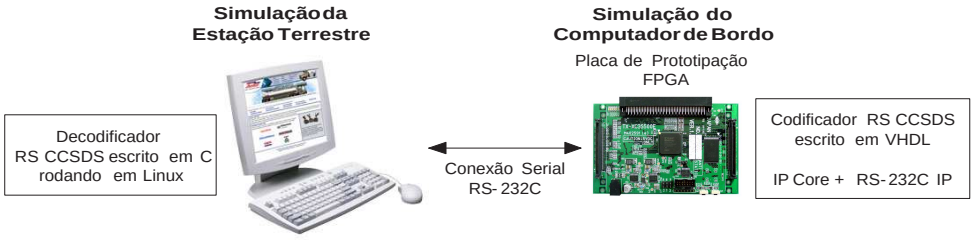

Figura 8.7: Codificador RS CCSDS escrito em VHDL juntamente com um núcleo IP serial

como apresentado no primeiro método de validação, foi feito o uso de um programa escrito em linguagem C, adaptado para suportar o padrão CCSDS, utilizado nesse trabalho. A partir do uso do codificador Reed-Solomon foi possível obter, em software, as tabelas de *log* e *anti-log* utilizadas para o processo de codificação e inseridas estaticamente na descrição em hardware do circuito codificador. A justificativa para tal se dá no padrão CCSDS utilizado, tendo em vista que o padrão proposto é RS(255,223), então é possível descrever a relação de tabelas de *log* e *anti-log* de forma a fazer com que o circuito não precise calculá-las em tempo real, economizando tempo de processamento na codificação, tornando o circuito mais rápido. O segundo ponto diz respeito à ocupação de área do circuito. Para o cálculo dessas tabelas seria necessário, no mínimo, duas vezes a área ocupada na abordagem de forma a colocá-las estaticamente na descrição do circuito. Com isso, além do circuito ser mais rápido para a execução do processo de codificação de telemetria, o mesmo ocupa menos espaço em hardware, sendo favorável em dois aspectos no projeto de circuitos para satélites.

Esta seção apresentou a metodologia utilizada para projeto, implementação e validação do modelo de telemetria CCSDS utilizando diferentes mecanismos. Inicialmente foram implementadas as camadas de empacotamento e transferência em linguagem de descrição de hardware VHDL. Os circuitos foram validados através de simulação utilizando o software ModelSim. Após essa etapa, partiu-se para a implementação e validação do algoritmo Reed-Solomon, representado pela camada de codificação, principal contribuição nesse trabalho. Inicialmente o algoritmo foi implementado em software utilizando dois PCs interconectados via porta serial, padrão RS-232C, rodando sistema operacional Linux. Após essa etapa, partiu-se para a implementação do circuito codificador de telemetria em hardware e validação através de simulação e comunicação física entre um PC e o dispositivo FPGA.

8.2 Implementação e Validação dos Módulos IP para Telecomando

Figura 8.8: Codificador/Decodificador BCH CCSDS escrito em C, transferindo dados entre dois PCs Linux

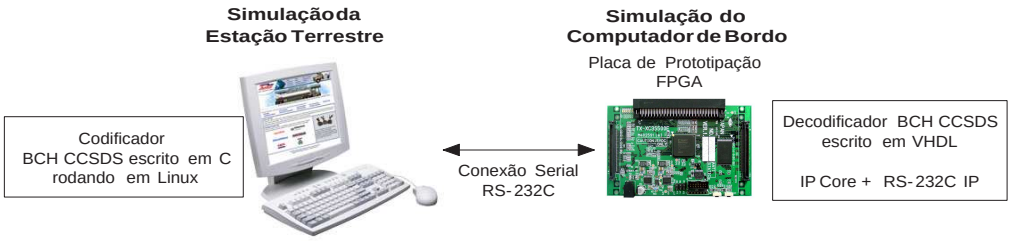

Figura 8.9: Decodificador BCH CCSDS escrito em VHDL juntamente com um núcleo IP serial

Os módulos codificador e decodificador BCH utilizado para processamento de dados de telecomando foram descritos em linguagem VHDL e validados através de simulação na ferramenta ModelSim [37]. Os dados de simulação são apresentados no Capitulo 9, essa Seção tem por objetivo apresentar o processo de captura dos dados de polinômios a serem colocados estaticamente no código VHDL a fim de deixar o circuito mais rápido, ocupando menor área.

Inicialmente o polinômio utilizado foi o proposto pelo padrão CCSDS, e é dado por $g(x) = x^7 + x^6 + x^2 + 1$. Os próximos passos consistem na formação das matrizes de geração de paridade (G) e matriz de verificação de paridade (H).

A matriz geradora G possui dimensão (56×63), sendo 56 linhas e 63 colunas. Esta é composta por duas sub-matrizes, a matriz G_1 de dimensão (56×56) é composta por uma matriz identidade, e a matriz G_2 de dimensão (56×7) é composta pelos 56 padrões de síndrome (S'). Esses padrões de síndrome são calculados efetuando a divisão modular de cada polinômio pelo gerador polinomial $g(x)$, sendo $S'(x)_i = x^i \ mod \ g(x)$, onde $n - k \leq i < n$, o processo de construção da matriz de paridade G é mostrado na Seção 5.4. Para a captura dos valores de divisão modular entre os polinômios foi escrito um programa utilizando o software Maple. O programa é apresentado como segue:

```
1.  g := x^7+x^6+x^2+1;
2.  n := 63;
3.  k := 56;
```

```
4.  i := n-1;
5.  z := n-k;
6.  j := 1;
7.  r := array(1..56);
8.  while i >= z do
9.    a := x^i;
10.   r[j] := rem(a,g,x) mod 2;
11.   sort(r[j]);
12.   j := j+1;
13.   i := i-1;
14. od;
15. r;
```

A linha 1 do código descrito apresenta a atribuição do polinômio gerador $g(x)$ proposto pelo padrão CCSDS. O padrão BCH utilizado nesse trabalho é o BCH$(63, 56)$, logo as variáveis n e k recebem respectivamente esses valores, n representa o número de bits da mensagem a ser enviada e k representa o número de bits de dados da mensagem. A diferença entre eles, apresentada pela variável z, representa o número de bits de paridades anexados aos dados codificados. As variáveis i e j representam apenas variáveis para controle de índices, sendo $z \leq i < n$. A variável a recebe o polinômio que realizará a divisão pelo polinômio gerador $g(x)$. A operação é realizada módulo 2 pois os dados são apresentados em notação binária, conforme trabalha o algoritmo de BCH, diferentemente do algoritmo Reed-Solomon que trabalha com símbolos de m bits. Ao final do processamento, tem-se os 56 padrões de síndrome possíveis para o polinômio gerador $g(x) = x^7 + x^6 + x^2 + 1$. Assim, a matriz G é gerada e os polinômios inseridos nela são calculados no Maple. A Figura 8.10 apresenta a execução do programa escrito em Maple tendo como resultados os 56 polinômios representando os padrões de síndrome gerados.

```
g := x^7+x^6+x^2+1;
n := 63;
i := n-1;
k := 56;
z := n-k;
r := array(1..56);
j := 1;
while i >= z do
   a := x^i;
   i := i-1;
   r[j] := rem(a,g,x) mod 2;
   sort(r[j]);
   j := j+1;
od;
r;
```

$$\left[\left[x^6+x^5+x, x^5+x^4+1, x^6+x^5+x^4+x^3+x, x^5+x^4+x^3+x^2+1, x^6+x^5+x^4+x^3+x^2, x^5+x^4+x^3+x^2+x,\right.\right.$$
$$x^4+x^3+x^2+x+1, x^6+x^5+x^3+x^2+1, x^6+x^4+x^2, x^5+x^3+x, x^4+x^2+1, x^6+x^5+x^3, x^5+x^4+x^2, x^4+x^3+x,$$
$$x^3+x^2+1, x^6+x^5+x^2, x^5+x^4+x, x^4+x^3+1, x^6+x^5+x^3+x^2+x, x^5+x^4+x^2+x+1, x^6+x^5+x^4+x^3+1,$$
$$x^6+x^4+x^3+x^2+x, x^5+x^3+x^2+x+1, x^6+x^5+x^4+x^2+1, x^6+x^4+x^3, x^5+x^3+x^2, x^4+x^2+x, x^3+x+1,$$
$$x^6+x^5+x^2+x+1, x^6+x^4+1, x^6+x^3+x, x^5+x^2+1, x^6+x^5+x^4, x^5+x^4+x^3, x^4+x^3+x^2, x^3+x^2+x, x^2+x+1,$$
$$x^6+x^5+1, x^6+x^4+x, x^5+x^3+1, x^6+x^5+x^4+x^2+x, x^5+x^4+x^3+x+1, x^6+x^5+x^4+x^3+x^2+x+1,$$
$$x^6+x^4+x^3+x^2+1, x^6+x^3+x^2, x^5+x^2+x, x^4+x+1, x^6+x^5+x^3+x+1, x^6+x^4+x^2+x+1, x^6+x^3+1, x^6+x^2+x,$$
$$\left.\left.x^5+x+1, x^6+x^5+x^4+x+1, x^6+x^4+x^3+x+1, x^6+x^3+x^2+x+1, x^6+x^2+1\right]\right]$$

Figura 8.10: Execução do programa para geração de padrões de síndrome utilizando o software Maple

O primeiro polinômio resultante é dado por $x^6 + x^5 + x$, em representação binária 1100010, logo, esse padrão de síndrome compõe a primeira linha da matriz G a partir da coluna 57, onde se encontram os padrões de síndrome. Dessa forma, toda a matriz G é gerada e os padrões de síndrome são colocados na mesma seqüência com que foram gerados.

Como apresentado na Seção 5.4, a matriz de verificação de paridade H^T é dada pela junção das sub-matrizes G_2 e da matriz identidade I. Logo, tem-se as duas matrizes necessárias para a execução dos processos de codificação e decodificação BCH. Para verificar se as duas matrizes foram geradas de forma correta e garantir que os polinômios e síndromes foram calculados corretamente, multiplica-se a matriz G pela matriz H^T, o resultado de $G \times H^T$ deverá ser igual a uma matriz contendo em todos os seus elementos o valor 0. Dessa forma, verifica-se que as matrizes estão corretas e podem ser inseridas estaticamente no código VHDL dos circuitos descritos.

A partir desse passo, as matrizes G e H^T foram colocadas estaticamente no código VHDL a fim de economizar área e tempo de processamento. Caso fosse descrito um circuito para gerar essas matrizes, a ocupação do FPGA seria maior e o tempo de processamento para geração das mesmas seria acrescido ao tempo de execução dos algoritmos de codificação e decodificação. Logo, optou-se pela otimização em termos de área e tempo de processamento nos passos envolvidos para a execução do algoritmo BCH em hardware.

A validação dos circuitos codificador/decodificador BCH foi feita utilizando a ferramenta ModelSim e os dados de simulação são apresentados no Capítulo 9. Esse capítulo apresentou os passos executados na estratégia utilizada para o projeto, implementação e validação dos circuitos Reed-Solomon e BCH em hardware além das camadas de empacotamento e transferência de telemetria seguindo o padrão CCSDS. O Capítulo 9 apresenta os dados de área, desempenho e simulação dos circuitos propostos nesse trabalho.

Capítulo 9

Resultados

Este Capítulo tem por objetivo apresentar os resultados obtidos através de simulação e sínteses lógica e física dos circuitos codificador Reed-Solomon, codificador/decodificador BCH, camada de empacotamento e transferência para TM e TC. A Seção 9.1 apresenta o fluxo de validação do algoritmo Reed-Solomon enquanto na Seção 9.2 dados levantados utilizando o algoritmo BCH são apresentados. Por fim, a Seção 9.3 apresenta os dados de área dos circuitos implementados para as camadas de empacotamento e transferência TM e TC.

9.1 Codificador RS CCSDS em Hardware

O módulo codificador RS foi validado através de dois métodos. O primeiro consistiu na simulação do circuito através do simulador ModelSim, ferramenta da empresa Mentor Graphics. O circuito codificador RS é composto de três sinais de entrada (*clock*, *reset* e *data*) e dois sinais de saída (*ready*, *encoded_ data*), apresentados na Figura 9.1.

Figura 9.1: Interfaces de entrada e saída do circuito codificador RS CCSDS

O sinal de *clock* é responsável por gerar o pulso de sincronismo para o circuito, enquanto o sinal de *reset* serve para sinalizar o circuito que o mesmo deve ser inicializado, fazendo com que a máquina de estados direcione o processamento para seu estado inicial e carregue o conteúdo em seus registradores. O circuito inicia seu processamento quando o sinal de *reset* passa do nível lógico '0' para o nível lógico '1'. A Figura 9.1 apresenta as interfaces de entrada e saída do circuito. O terceiro sinal de entrada, chamado *data*, contém os 223 bytes da mensagem de telemetria a ser codificada. A saída é composta de dois sinais: o sinal de *ready* indica o momento em que o circuito acabou seu processamento, enquanto o sinal *encoded_ data* contém os 255 bytes de telemetria incluindo os 32 bytes de paridade anexados aos bytes de dados.

Para a simulação do funcionamento do circuito foi criado um arquivo de *testbench* que gera sinais de estímulo para o circuito codificador. Além dos sinais de *clock* e *reset*, é atribuído ao sinal *data* os 223 bytes de telemetria a serem codificados. Para a simulação apresentada nessa seção, foram utilizados bytes variando de 1 a 223, representando os 223 bytes de entrada. A Figura 9.2 ilustra o momento em que o circuito começa a codificar.

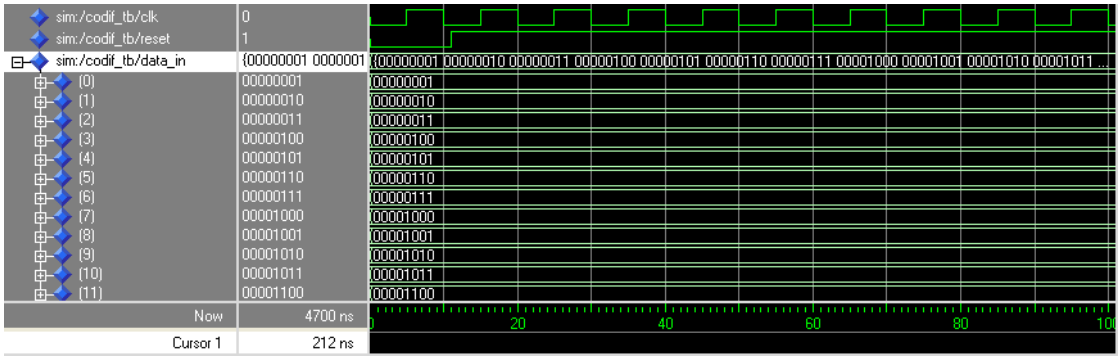

Figura 9.2: Início do processamento do codificador RS CCSDS

Ao término da codificação, o circuito coloca em sua saída, os 32 bytes de paridade a serem anexados à mensagem de telemetria. A Figura 9.3 apresenta o momento em que os bytes de paridade são calculados pelo circuito codificador. Os bytes dos índices 224 a 255 representam os 32 bytes de paridade gerados pelo circuito. O sinal de *ready* passa para o nível lógico '1' após o processamento do algoritmo, e pode ser observado aproximadamente aos $4485ns$ de simulação.

Através desse método, foi possível validar o funcionamento lógico do circuito codificador de telemetria proposto. O segundo método de validação se deu através da síntese física do circuito na plataforma Virtex-II Pro da Xilinx. O circuito foi prototipado em hardware e validado através do uso de um IP para comunicação serial [36], padrão RS-232C desenvolvido pelo Grupo de Apoio ao Projeto de Hardware (GAPH), da PUCRS. O módulo serial é responsável pela comunicação entre o aplicativo em linguagem Java, rodando sobre um PC, e o IP serial prototipado em hardware. Com o uso do módulo IP serial foi possível adequá-lo para comunicar com o módulo IP do codificador RS, também prototipado em hardware.

A Figura 9.4 apresenta o fluxo de validação do módulo IP através do uso de um aplicativo em Java para envio de dados de telemetria. Para a sincronização do aplicativo em Java com o módulo serial IP presente em hardware, é necessário enviar o byte 55, representado na Figura 9.4 por *BS* (Byte de Sincronização), no início da *stream* de bytes de telemetria a ser codificada em hardware.

Logo, a *stream* de telemetria a ser enviada para o módulo IP serial e, posteriormente ao módulo IP codificador, é composta de 224 bytes, o primeiro representado por 55 e os demais bytes compostos de valores de 1 a 223, representados em notação hexadecimal. Os dados de telemetria são codificados no IP codificador presente em hardware e os dados de paridade são retornados via porta serial para o aplicativo em java. Esse processo é representado pela Figura 9.5.

A partir da simulação e prototipação do codificador RS em hardware foi possível validar o circuito codificador de telemetria. A Tabela 9.1 apresenta dados de área referente a síntese física do circuito codificador RS. O mesmo foi prototipado sobre a plataforma Virtex-II Pro da Xilinx. A área ocupada no FPGA é medida em número de *Look-up Tables* (LUTs) de quatro entradas

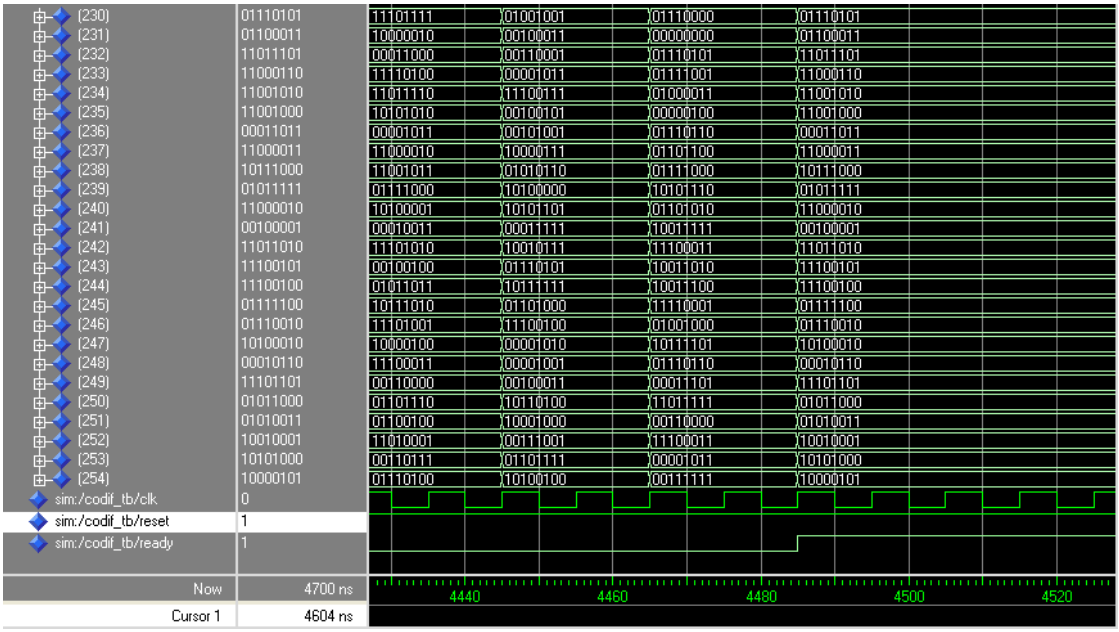

Figura 9.3: Fim do processamento do codificador RS CCSDS

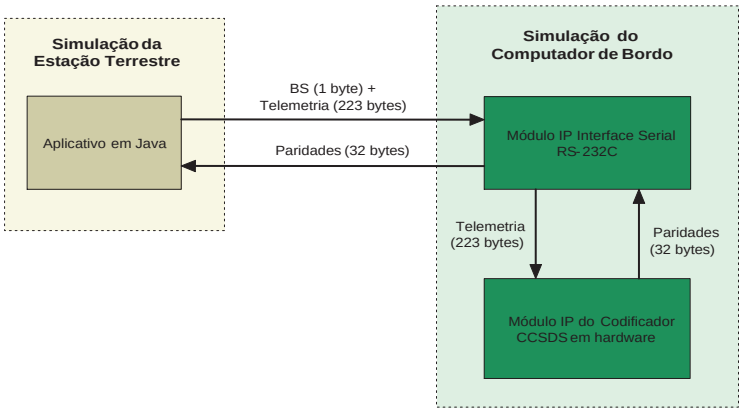

Figura 9.4: Fluxo de validação do módulo IP codificador CCSDS utilizando aplicativo em Java para envio de dados de telemetria

usadas para implementar a lógica do circuito.

O codificador RS apresentado em [17] é composto por um módulo gerador polinomial, cuja funcionalidade consiste na geração do polinômio que irá ser usado para compor todos os elementos pertencentes ao Corpo de Galois para os processos de codificação e decodificação. No trabalho proposto em [17], o circuito foi sintetizado utilizando a plataforma Xilinx VirtexE 600, dispositivo XCV600E, com capacidade para 600k portas lógicas. O dispositivo possui área de13.824 LUTs e o circuito utiliza 215 LUTs, representando $1,56\%$ de ocupação em área. É importante observar em [17] que o circuito gerador polinomial foi implementado e por esse motivo o mesmo apresenta

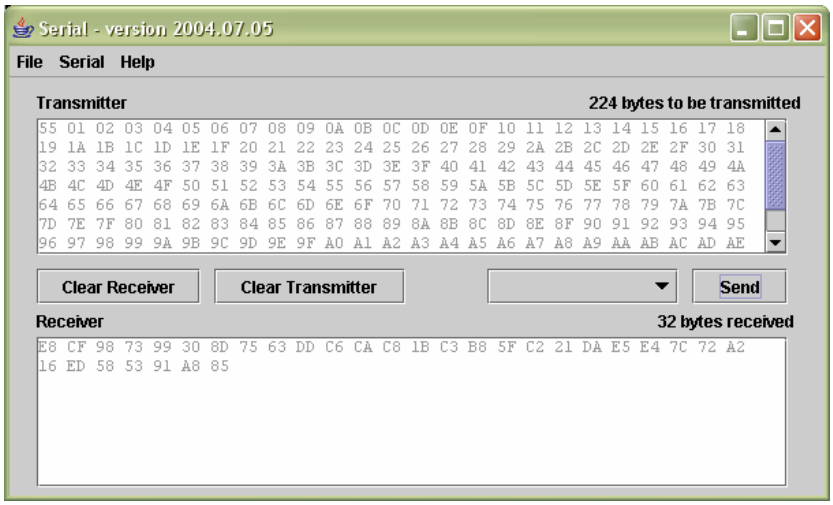

Figura 9.5: Aplicativo em java responsável pelo envio de dados de telemetria a serem codificados no módulo IP em hardware

Tabela 9.1: Dados de síntese do circuito codificador RS

Plataforma	Dispositivo FPGA	Área Disponível (#4-LUTs)	Área Ocupada (#4-LUTs)	Flip-flops
Xilinx Virtex-II Pro*	XC2VP20	18.560 (100%)	48 (0,26%)	35
Xilinx VirtexE 600**	XCV600E	13.824 (100%)	215 (1,56%)	-
Xilinx VirtexE 600***	XCV600E	13.824 (100%)	134 (0,97%)	-
* Dados levantados do codificador RS apresentado nesse trabalho				
** Dados do codificador RS proposto no artigo [17]				
*** Dados do codificador RS referentes ao artigo [16]				

área maior do que o proposto nesse documento. Para nossa proposta, o número de bytes de paridade está definido para o padrão RS CCSDS, representado por $RS(255, 223)$. Sendo assim, o gerador polinomial foi obtido em software e colocado estaticamente no código VHDL do circuito codificador RS. Com isso, a ocupação de área do circuito no FPGA reduziu significativamente e o desempenho do mesmo aumentou devido ao tempo economizado no processo de cálculo do polinômio gerador. O circuito RS, apresentado nesse trabalho, teve uma ocupação de área de48 LUTs, representando $0,26\%$ da área disponível no dispositivo FPGA utilizado.

A partir do levantamento de dados de ocupação de área no dispositivo por parte do circuito codificador RS, foi também medido o tempo de execução do processo de codificação de telemetria em hardware. A Figura 9.6 apresenta o tempo de execução do circuito codificador RS em hardware obtido através do uso de um analisador lógico.

O circuito opera com um sinal, chamado R, que permanece em nível lógico '1' enquanto o circuito codificador está em processamento. Ao final da execução, o nível lógico do sinal retorna para '0'. Assim, para se obter o tempo de processamento do codificador, bastou medir o período em que o sinal R permaneceu em nível lógico '1'. Com isso foi possível obter o tempo real que o circuito levou para codificar uma telemetria no padrão RS(255, 223) em hardware.

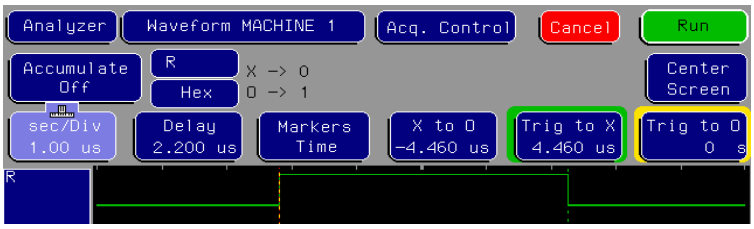

Figura 9.6: Análise do tempo de execução do circuito codificador RS em hardware

Essa seção apresentou os resultados de validação para os dois métodos propostos. O primeiro método consistiu na validação através de simulação utilizando o software ModelSim. O segundo método consistiu na prototipação do circuito RS em um dispositivo FPGA comunicando com um aplicativo escrito em linguagem Java. Para ambos os métodos, foram levantados dados referente a forma de ondas geradas pelo circuito codificador bem como dados de síntese, como ocupação de área (baseado em número de LUTs) e tempo de processamento do circuito codificador de telemetria. Por fim, é feita uma comparação de ocupação de área com os artigos propostos em [17, 16], justificando assim a contribuição científica do trabalho desenvolvido.

9.2 Codificador e Decodificador BCH CCSDS em Hardware

O módulo codificador BCH, da mesma forma como o módulo codificador RS, foi validado através de simulação utilizando ModelSim. O circuito codificador BCH é composto de quatro sinais de entrada (*clock*, *reset*, *enable* e *data*) e dois sinais de saída (*ready*, *encoded_ data*), apresentados na Figura 9.7.

Figura 9.7: Interfaces de entrada e saída do circuito codificador BCH CCSDS

O sinal de *clock* é responsável por gerar o pulso de sincronismo para o circuito, enquanto o sinal de *reset* serve para sinalizar o circuito que o mesmo deve ser inicializado, fazendo com que a máquina de estados direcione o processamento para seu estado inicial e carregue o conteúdo em seus registradores. A Figura 9.7 apresenta as interfaces de entrada e saída do circuito. O terceiro sinal de entrada, chamado *enable* serve para sinalizar o início do processamento de codificação. Quando seu nível lógico passa de '0' para '1', o circuito é sinalizado de que pode começar seu processamento. O quarto sinal de entrada, chamado de *data*, contém os 56 bits da mensagem de telecomando a ser codificada. A saída é composta de dois sinais: o sinal de *ready* indica o momento em que o circuito acabou seu processamento, enquanto o sinal *encoded_ data* contém

os 63 bits de telecomando, incluindo os 7 bits de paridade anexados aos bits de dados.

Para a simulação do funcionamento do circuito foi criado um arquivo de *testbench* que gera sinais de estímulo para o circuito codificador BCH. Além dos sinais de *clock*, *reset* e *enable*, é atribuído ao sinal *data* os 56 bits de telecomando a serem codificados. Para a simulação apresentada nessa seção, foi utilizado um telecomando no formato "00101010111111111111111111111111 11111111111111111111111111111", representando os 56 bits de entrada. A Figura 9.8 ilustra o momento em que o circuito começa a codificar.

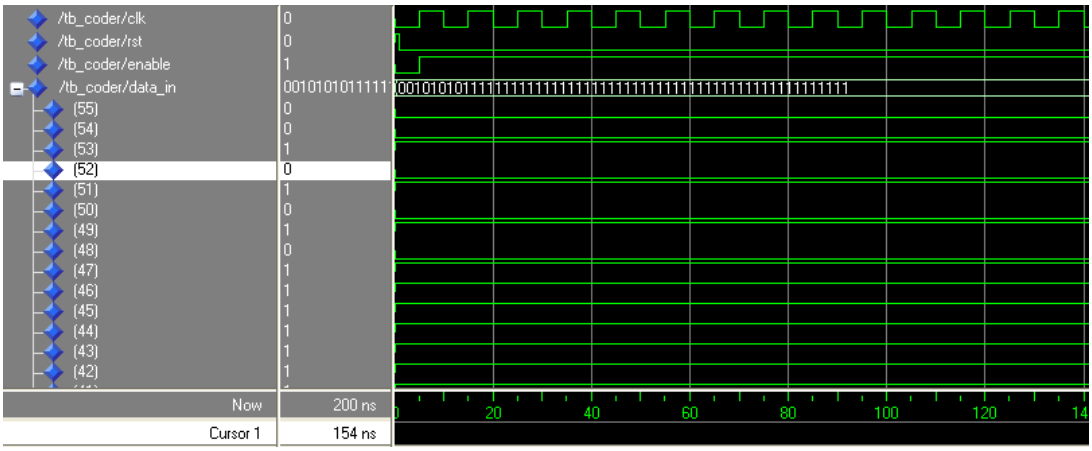

Figura 9.8: Início do processamento do codificador BCH CCSDS

Ao final do processamento de codificação, o circuito coloca em sua saída os dados de entrada anexados os 7 bits de paridade calculados pelo módulo codificador BCH. A Figura 9.9 ilustra o momento em que o sinal de *ready* passa do nível lógico '0' para nível lógico '1' indicando o fim do processamento de codificação.

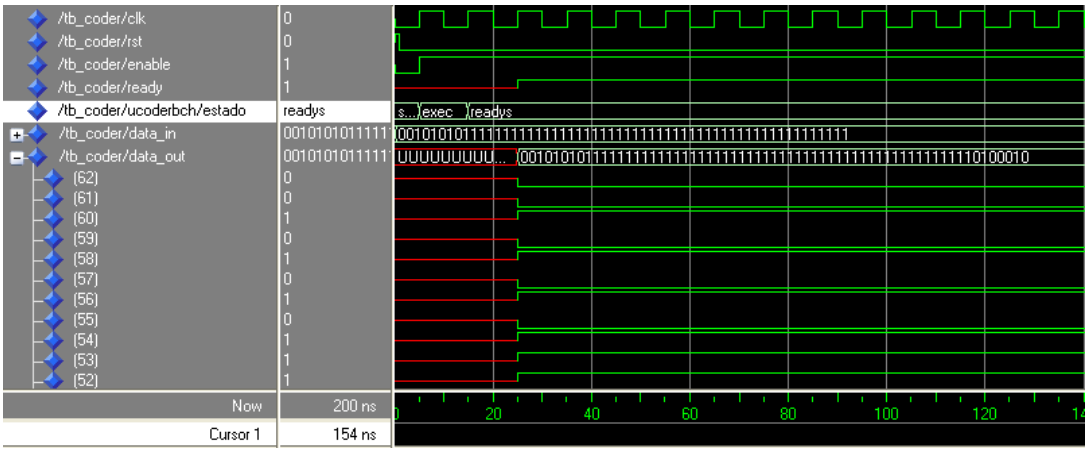

Figura 9.9: Final do processamento do codificador BCH CCSDS

É possível observar através dos sinais que o dado de saída é o mesmo dado de entrada somado os 7 bits de paridades gerados pelo módulo em hardware. Além de dados de simulação do circuito

Tabela 9.2: Dados de síntese do circuito codificador BCH

Plataforma	Dispositivo FPGA	Área Disponível (#4-LUTs)	Área Ocupada (#4-LUTs)	Flip-flops
Xilinx Virtex-IV*	XV4VSX35	30.720 (100%)	209 (0,68%)	128
* Dados levantados do codificador BCH apresentado nesse trabalho				

Tabela 9.3: Dados de síntese do circuito codificador BCH levantados através da ferramenta ISE

Selected Device:	4vsx35ff668-12	
Number of Slices:	118 out of 15360	0,77%
Number of Slice Flip Flops:	128 out of 30720	0,42%
Number of 4 input LUTs:	209 out of 30720	0,68%
Number of IOs:	123	
Number of bonded IOBs:	123 out of 448	27%
Number of GCLKs:	1 out of 32	3%

codificador BCH, foram levantados dados referente à ocupação de área do circuito no dispositivo Virtex-IV XV4VSX35 da Xilinx. A área ocupada no FPGA é medida em número de *Look-up Tables* (LUTs) de quatro entradas usadas para implementar a lógica do circuito e é apresentada na Tabela 9.2.

Outros dados de síntese levantados através da Ferramenta ISE da empresa Xilinx, são apresentados na Tabela 9.3.

A partir da validação do módulo codificador BCH em hardware, partiu-se para a implementação e validação do módulo decodificador BCH. O módulo decodificador BCH, da mesma forma como os módulos codificador BCH e codificador RS, foi validado através de simulação utilizando ModelSim. O circuito codificador BCH é composto de quatro sinais de entrada (*clock*, *reset*, *enable* e *encoded_ data*) e quatro sinais de saída (*ready*, *data*, *status* e *aff*), apresentados na Figura 9.10.

Figura 9.10: Interfaces de entrada e saída do circuito decodificador BCH CCSDS

O sinal de *clock* é responsável por gerar o pulso de sincronismo para o circuito, enquanto o sinal de *reset* serve para sinalizar o circuito que o mesmo deve ser inicializado, fazendo com que a máquina de estados direcione o processamento para seu estado inicial e carregue o conteúdo em seus registradores. A Figura 9.10 apresenta as interfaces de entrada e saída do circuito. O terceiro sinal de entrada, chamado *enable* serve para sinalizar o início do processamento de codificação. Quando seu nível lógico passa de '0' para '1', o circuito é sinalizado de que pode começar seu processamento. O quarto sinal de entrada é chamado de*encoded_ data*, e contém os

62 bits da mensagem de telecomando a ser decodificada. A saída é composta de quatro sinais: o sinal de *ready* indica o momento em que o circuito acabou seu processamento, enquanto o sinal *data* contém os 56 bits do telecomando corrigido. O sinal de *status* serve para sinalizar se o circuito conseguiu corrigir (nível lógico '1') ou não (nível lógico '0') o telecomando recebido. O sinal de *aff* (*affected*) indica se o telecomando recebido foi corrompido ou não, quando em nível lógico '1' indica que o telecomando foi corrompido e que passará pelo processo de correção, quando em nível lógico '0' indica que o telecomando não foi corrompido e que não precisa ser corrigido.

Para validar o decodificador BCH foram criados três arquivos de *testbench* que geram sinais de estímulo para o circuito decodificador. Além dos sinais de *clock*, *reset* e *enable*, é atribuído ao sinal *encoded_ data* os 63 bits de telecomando a serem decodificados. Para a validação do mesmo, foram criadas três situações de teste, onde a primeiro situação representa um telecomando enviado contendo os 56 bits de dados (00101010111111111111111111111111111111111111111111111111111111) somado com os 7 bits de paridade (0100010). Na primeira situação, o telecomando não sofre qualquer alteração, ou seja, não é corrompido, e o telecomando recebido é então enviado para o decodificador BCH, apresentado na Figura 9.11.

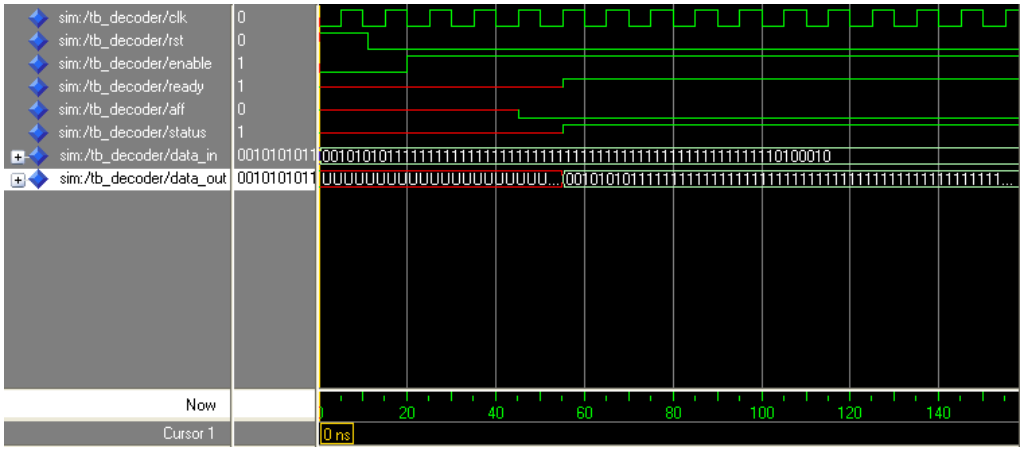

Figura 9.11: Processamento do decodificador BCH com telecomando não corrompido

Como o telecomando recebido não foi corrompido, o sinal de saída *aff* permanece em nível lógico '0' indicando o não corrompimento do telecomando, enquanto o sinal de *status* altera para nível lógico '1', sinalizando que o telecomando está correto, ou seja, como não foi preciso passar pelo processo de correção do telecomando, o sinal indica que ele está pronto para ser passado para os demais instrumentos presentes no veículo espacial.

A segunda situação representa o envio do mesmo telecomando porém o bit mais significativo, representado pelo bit mais à esquerda do telecomando, foi alterado durante a transmissão, sendo o telecomando recebido corrompido, com o bit mais significativo alterado. O telecomando recebido é representado por "10101010111111111111111111111111111111111111111111111111111110100010", diferentemente do telecomando enviado (00101010111111111111111111111111111111111111111111111111111110100010). O telecomando recebido é processado no módulo decodificador BCH o qual gera as saídas apresentadas na Figura 9.12.

Como o telecomando foi corrompido, o sinal *aff* coloca em sua saída o sinal em nível lógico '1' indicando que o telecomando recebido foi corrompido durante a transmissão. Sendo assim o telecomando é repassado ao módulo responsável pela decodificação e recuperação do telecomando

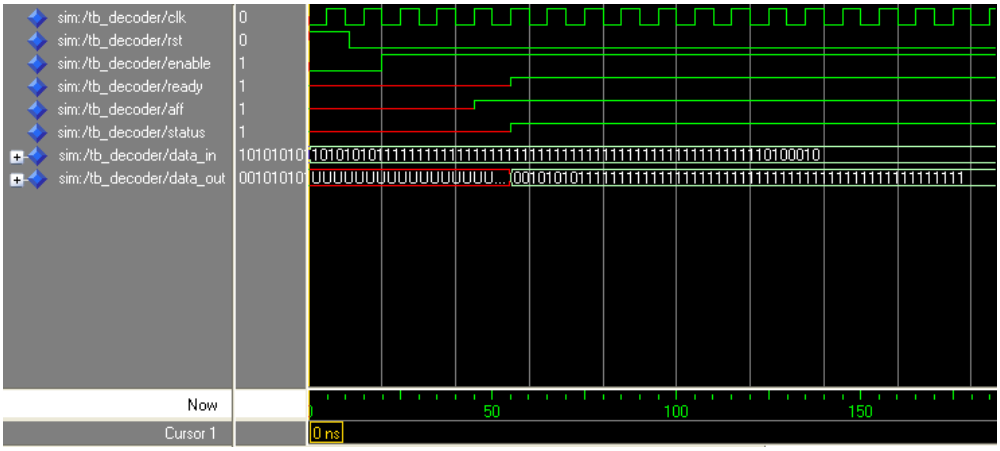

Figura 9.12: Processamento do decodificador BCH com telecomando corrompido e recuperado

originalmente enviado. O sinal *status* é então levado para nível lógico '1' indicando que o processo de recuperação do telecomando original foi concluído com êxito, e que o mesmo pode ser repassado para os demais instrumentos presentes no satélite. Em sua saída, representada por*data*, os dados do telecomando recuperado são apresentados.

A terceira e última situação representa o envio do mesmo telecomando utilizado nas situações, porém com os dois bits mais significativos alterados, simulando o corrompimento do telecomando com alteração de dois bits. O telecomando recebido então é dado da forma "111010101111111 1111111111111111111111111111111111111111110100010", sendo o original (00101010111111111111 1111111111111111111111111111111110100010). O telecomando recebido é então processado pelo módulo decodificador BCH o qual gera as saídas apresentadas na Figura 9.13.

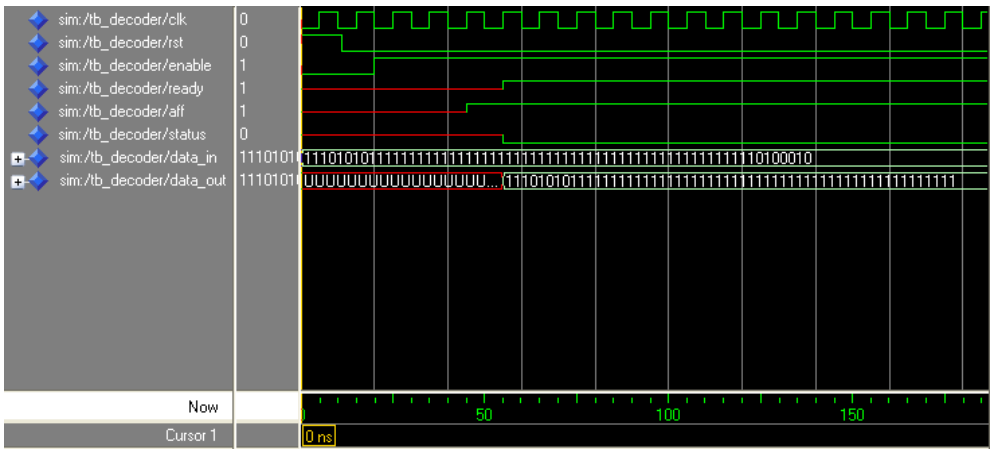

Figura 9.13: Processamento do decodificador BCH com telecomando corrompido e não recuperado

Como a capacidade de correção do polinômio proposto pelo padrão CCSDS é de apenas1 bit entre os 56 que compõe o *frame* de dados do telecomando, o decodificador não é capaz de

Tabela 9.4: Dados de síntese do circuito decodificador BCH

Plataforma	Dispositivo FPGA	Área Disponível (#4-LUTs)	Área Ocupada (#4-LUTs)	Flip-flops
Xilinx Virtex-IV*	XV4VSX35	30.720 (100%)	251 (0,82%)	86
* Dados levantados do decodificador BCH apresentado nesse trabalho				

Tabela 9.5: Dados de síntese do circuito decodificador BCH levantados através da ferramenta ISE

Selected Device:	4vsx35ff668-12	
Number of Slices:	133 out of 15360	0%
Number of Slice Flip Flops:	146 out of 30720	0%
Number of 4 input LUTs:	252 out of 30720	0%
Number of IOs:	125	
Number of bonded IOBs:	125 out of 448	27%
Number of GCLKs:	1 out of 32	3%

corrigir o telecomando corrompido em 2 bits. No entanto ele sinaliza através dos sinais *aff* e *status*, o primeiro em nível lógico '1' indicando que o telecomando foi corrompido, o segundo em nível lógico '0' indicando que não foi possível corrigir o telecomando. Com essas informações, é possível enviar uma solicitação para a base terrestre reenviar o telecomando para que seja possível recebê-lo de forma correta. Essa situação é ilustrada através da Figura 9.13.

Da mesma forma como feito com os demais módulos em hardware, foram levantados dados referente a ocupação de área do circuito no dispositivo Virtex-IV XV4VSX35 da Xilinx. A área ocupada no FPGA é medida em número de *Look-up Tables* (LUTs) de quatro entradas usadas para implementar a lógica do circuito e é apresentada na Tabela 9.4. Outros dados de síntese levantados através da Ferramenta ISE da empresa Xilinx, são apresentados na Tabela 9.5.

9.3 Dados de Área das Camadas de Empacotamento e Transferência TM e TC em Hw

A Tabela 9.6 apresenta os dados referente à área ocupada dos circuitos de empacotamento e transferência de TM e TC. Eles foram sintetizados em uma plataforma FPGA Virtex-II Pro da família Xilinx, dispositivo XC2VP20. Os dados de área são medidos pelo número de LUTs de quatro entradas ocupadas e número de flip-flops.

É possível observar que a ocupação do circuito é pequena, ou seja, qualquer dos circuitos não utiliza mais do que 100 LUTS de 4 entrada, capacidade encontrada em qualquer dispositivo

Tabela 9.6: Dados de síntese dos circuitos responsáveis pelas camadas de empacotamento e transferência de TM e TC

Circuito	Área Disponível (#4-LUTs)	Área Ocupada (#4-LUTs)	Flip-flops
Camada de Empacotamento de TM	18.560 (100%)	85 (0,46%)	223 (1,20%)
Camada de Transferência de TM	18.560 (100%)	87 (0,47%)	229 (1,23%)
Camada de Empacotamento de TC	18.560 (100%)	65 (0,35%)	184 (0,99%)
Camada de Transferência de TC	18.560 (100%)	69 (0,37%)	192 (1,04%)

FPGA, o que facilita e incentiva o projeto de hardware para as camadas de empacotamento e transferência.

Este capítulo apresentou os resultados obtidos com o desenvolvimento do presente trabalho. Nele são apresentados os resultados de ocupação de área e tempo de processamento do circuito codificador Reed-Solomon. Para o algoritmo de BCH, é apresentada a arquitetura proposta do circuito bem como as formas de onda para fins de validação do circuito capturadas através do software ModelSim. Por fim, é feito um levantamento de ocupação de área dos circuitos que representam as camadas de empacotamento e transferência para TM e TC.

Capítulo 10

Conclusões e Trabalhos Futuros

Esse trabalho apresentou o projeto de um SoC para codificação de telemetria e codificação/decodificação de telecomando CCSDS utilizando os algoritmos Reed-Solomon e BCH respectivamente. Inicialmente conceitos referentes aos sistemas de telecomando e telemetria seguindo o padrão CCSDS foram introduzidos a fim de facilitar o entendimento da proposta apresentada nesse trabalho. Em seguida, o algoritmo Reed-Solomon foi descrito, apresentando o padrão RS utilizado para codificação de telemetria. Alguns conceitos da álgebra abstrata empregada sobre Corpos de Galois foram descritos objetivando facilitar a compreensão do texto nos capítulos referentes aos algoritmos implementados. Logo após, foi apresentado o processamento do algoritmo de codificação RS, onde foi introduzido o circuito LFSR, utilizado para codificação de telemetria. Um exemplo do processo de codificação de mensagem no padrão RS(7,3) foi apresentado, descrevendo passo a passo os cálculos envolvidos na codificação de códigos Reed-Solomon. Na seqüência, foi apresentada a metodologia utilizada para validação em software e hardware dos módulos de codificação e decodificação do algoritmos RS e BCH . Inicialmente os algoritmos foram validados exclusivamente em software, utilizando dois PCs conectados via porta serial. Após essa etapa, foram implementadas as camadas de empacotamento e transferência de TC/TM CCSDS. Os módulos foram descritos na linguagem VHDL e validados por meio de simulação utilizando o software ModelSim.

O trabalho apresentou algumas pesquisas científicas que utilizam o algoritmo de Reed-Solomon em diferentes tipos de aplicações. Na seção de resultados foram apresentados dados referente às implementações propostas nesse trabalho em comparação à implementações propostas por outros autores. O circuito codificador RS proposto nesse trabalho apresenta ocupação de área de 48 LUTs de 4 entradas, comparadas à 134 e 215 LUTs de ocupação levantadas pelos artigos propostos em [17, 16]. No entanto, a proposta aqui apresentada, define o polinômio gerador e todos os elementos pertencentes ao Corpo de Galois de forma estática no circuito, devido ao fato de tê-los pré-definidos pelo padrão de codificação de telemetria CCSDS. Assim, não se faz necessária a implementação de um circuito para geração do polinômio gerador que melhor se adapta à aplicação, economizando área de ocupação do circuito e aumentando a velocidade no processamento de codificação de telemetria e decodificação de telecomando. O circuito codificador de telemetria proposto nesse trabalho executa o processo de codificação em um tempo de $4,46\mu s$. Ele foi mensurado através de um sinal que permanece em nível lógico '1' enquanto o circuito está em execução. No momento da finalização do procedimento de codificação, o sinal retorna ao nível lógico '0'. O tempo em que o sinal permaneceu em nível lógico alto é medido através do uso de um analisador lógico, permitindo assim, o levantamento do tempo em que o circuito gasta para executar o processo de codificação de telemetria CCSDS utilizando o algoritmo

de Reed-Solomon padrão RS(255,223).

Os circuitos implementados foram validados através de simulação e também por meio de um aplicativo em Java que envia dados de telecomando/telemetria a serem codificados/decodificados nos circuitos prototipados em um dispositivo FPGA, recebendo dados resultantes provenientes do mesmo. Todo o processo de projeto, implementação e validação foi feito visando otimização de área e diminuição no tempo de processamento nos processos de codificação e decodificação.

Esse trabalho possibilitou um maior estreitamento entre as universidades PUCRS e a*University of Sussex* durante o estágio realizado na Inglaterra pelo autor durante as fases de projeto e implementação dos circuitos de codificação e decodificação de telemetria e telecomando respectivamente. O trabalho fez parte do programa espacial brasileiro UNIESPAÇO, o qual agregou cerca dos quinze projetos mais relevantes na área de pesquisas espaciais. Como parte do programa, estavam participando universidades no âmbito nacional, sendo a PUCRS a única universidade privada a participar desse programa, representada pelo trabalho aqui proposto. Ainda no escopo do programa UNIESPAÇO, o Instituto Nacional de Pesquisas Espaciais (INPE) se mostrou bastante interessado nos resultados levantados por esse trabalho e está adquirindo uma plataforma da empresa ELTA, no valor de€ 22.775,00 para a execução de testes no sistema de telecomando e telemetria CCSDS agregando as 3 camadas implementadas nesse trabalho. Assim sendo, os circuitos serão melhorados a medida do possível e utilizados inicialmente em missões espaciais acadêmicas, passando gradativamente a fazer parte do sistema de bordo das próximas gerações de satélites brasileiros.

Referências Bibliográficas

[1] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM Journal of Applied Mathematics. JSTOR. Society for Industrial and Applied Mathematics. Philadelphia, PA*, vol. 8, no. 2, pp. 300–304, June 1960.

[2] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres (Paris)*, September 1959.

[3] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control. MIT Computer Science & AI Laboratory. Elsevier. Atlanta, GA*, vol. 3, no. 1, pp. 68–79, March 1960.

[4] B. Sklar, *Digital Communications: Fundamentals and Applications*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. 1070p, January 2001.

[5] S. B. Wicker and V. K. Bhargava, Eds., *Reed-Solomon Codes and Their Applications*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc. 336p., September 1999.

[6] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice Hall. 624p., October 1983.

[7] CCSDS, "Recommendation for space data system standards: Telemetry channel coding: Issue-1." Consultative Committee for Space Data Systems. Washington, DC, USA, May 1984.

[8] NASA, "Mariner-mars 1969 a preliminary report." NASA, Washington, DC., Tech. Rep., 1969. Acessado em 9 de Outubro de 2006. [Online]. Available: http://nssdc.gsfc.nasa.gov/nmc/tmp/1969-014A.html

[9] NASA., "Mariner 6 and 7 pictures of mars," NASA, Washington, DC., Tech. Rep., 1971. Acessado em 6 de Julho de 2006. [Online]. Available: http://nssdc.gsfc.nasa.gov/nmc/tmp/1971-051A.html

[10] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* New York, NY, USA: Dover Publications. 1046p, 1964.

[11] M. J. E. Golay, "Notes on digital coding," in *Proc. IRE (Institute of Radio Engineers) (IEEE). 657p*, vol. 37, New York, NY, USA, June 1949.

[12] E. R. Berlekamp, "Key papers in the development of coding theory." New York, NY, USA: IEEE Press. 300p, 1974.

[13] CCSDS, "The consultative committee for space data systems," Reston, VA, 1982. Acessado em 13 de Maio de 2006. [Online]. Available: www.ccsds.org/

[14] ESA, *Packet Telecommand Standard. PSS-04-107: Issue* Noordwijk, The Netherlands: European Space Agency - ESA. n/a, April 1992.

[15] T. H.Tiggeler and D.Zheng, "A system-on-a-chip for small satellite data processing and control," *Proceedings of Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD'2000), Laurel, Maryland US, NASA*, September 2000.

[16] G. Neuberger, F. Kastensmidt, and R. Reis, "An automatic technique for optimizing reed-solomon codes to improve fault tolerance in memories," *IEEE Design and Test of Computers*, vol. 22, no. 1, pp. 50–58. Santa Barbara, CA, USA, January/February 2005.

[17] G. Neuberger, "Multiple bit upset tolerant sram memory," *ACM Trans. Design Automation Electronic Systems*, vol. 8, no. 4, pp. 577–590. New York, NY, USA, October 2003.

[18] S. Mamidi and M. e. a. Schulte, "Instruction set extensions for reed-solomon encoding and decoding," in *ASAP'05: Proceedings of the 2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)* Washington, DC, USA: IEEE Computer Society, July 2005, pp. 364–369.

[19] ETSI, "Digital video broadcasting (dvb); framing structure, channel coding and modulation for digital terrestrial television," *European Telecommunications Standards Institute (ETSI)*, vol. EN 300 744 V1.4.1. France, January 2001.

[20] L. Atieno, J. Allen, D. Goeckel, and R. Tessier, "An adaptive reed-solomon errors-and-erasures decoder," in *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays* New York, NY, USA: ACM Press, 2006, pp. 150–158.

[21] S. Li, K. Pan, J. Yuan, A. Vigil, and A. Berg, "Adaptive reed-solomon coding for wireless atm communications," *IEEE Southeastcon*, pp. 27–30. Nashville, Tennessee, USA, April 2000.

[22] M. K. Song, E. B. Kim, H. S. Won, and M. H. Kong., "Architecture for decoding adaptive reed-solomon codes with variable block length," *CES IEEE Transactions on Consumer Electronics*, vol. 48(3), pp. 631–637. Glenview, USA, August 2002.

[23] T. D. Wolf, "Programmable, reconfigurable dsp implementation of a reed-solomon encoder/decoder," . USA Patent Issued on May 7, 2002. Patent Number: 6385751.

[24] Y. Katayama and S. Morioka, "One-shot reed-solomon decoder," *33rd Annual Conference on Information Science and Systems*, vol. 33, pp. 700–705. Baltimore, MD, 1999.

[25] V. K. Bhargava and T. L.-N. et al., "Software and hardware r-s codec for wireless communications," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, vol. 2, pp. 739–740. Victoria, BC, Canada, August 1997.

[26] L. Song, K. K. Parhi, I. Kuroda, and T. Nishitani, "Hardware/software codesign of finite field datapath for low-energy reed-solomon codecs," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 2, pp. 160–172. Piscataway, NJ, USA, 2000.

[27] S. Burris and H. Sankappanavar, "A course in universal algebra," Dept. of Pure Mathematics, University of Waterloo. Waterloo, Ontario, Canada, 1981. Acessado em 18 de Agosto de 2005. [Online]. Available: citeseer.ist.psu.edu/sankappanavar81course.html

[28] K. Hoffman and R. Kunze, *Linear algebra*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall. 407p, 1971.

[29] E. H. Connell, *Elements of Abstract and Linear Algebra* Florida, USA: Department of Mathematics University of Miami, March 2004. Acessado em 12 de Novembro de 2005. [Online]. Available: http://www.math.miami.edu/~ec/book/

[30] N. Jacobson, *Basic Algebra I*, 2nd ed. New York, NY, USA: W. H. Freeman & Co (Sd). 499p, Frebruary 1985.

[31] R. H. Morelos-Zaragoza, *The Art of Error Correcting Coding*, 1st ed. New York, NY, USA: John Wiley & Sons. 238p, April 2002.

[32] A. D. Houghton, *The Engineer's Error Coding Handbook* London, UK: Chapman & Hall. 253p, January 1997.

[33] J. Garcia and M. J. Schulte, "A combined 16-bit binary and dual galois field multiplier," in *Proceedings of the IEEE International Symposium on Circuits and Systems* San Diego, CA, USA: IEEE, October 2002, pp. 63–68.

[34] C. Chiou, L. Lin, F. Chou, and S. Shu, "Low-complexity finite field multiplier using irreducible trinomials," vol. 39, no. 24. Stevenage, United Kingdom: Electronics Letters, November 2003, pp. 1709–1711.

[35] CCSDS, "Telecommand part 1 – channel service. blue book. issue 3, ccsds 201.0-b-3." Consultative Committee for Space Data Systems. Washington, DC, USA, January 1987.

[36] F. G. Moraes, "Rs-232 compatible serial interface with autobaud." Porto Alegre, RS, Brasil, September 2003. Acessado em 10 de Outubro de 2005. [Online]. Available: http://toledo.inf.pucrs.br/~gaph/

[37] M. Graphics, "Modelsim - a compreensive simulation and debug environment for complex asic and fpga designs." Mentor Graphics, October 2006. Acessado em 4 de Outubro de 2006. [Online]. Available: http://www.model.com/