UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

JOÃO DANIEL RUBIM BERTUCCI
NYAHN EKYÊ FERNANDES DUARTE

SISTEMA DE TRANSMISSÃO/RECEPÇÃO COM CÓDIGO CORRETOR DE ERRO (HAMMING) UTILIZANDO LINGUAGEM VHDL

Limeira
2024

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

JOÃO DANIEL RUBIM BERTUCCI
NYAHN EKYÊ FERNANDES DUARTE

SISTEMA DE TRANSMISSÃO/RECEPÇÃO COM CÓDIGO CORRETOR DE ERRO (HAMMING) UTILIZANDO LINGUAGEM VHDL

Monografia apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Graduado em Engenharia de Telecomunicações.

Orientador: Prof. Dr. Cristhof Johann Roosen Runge

Limeira
2024

Ficha catalográfica
Universidade Estadual de Campinas (UNICAMP)
Biblioteca da Faculdade de Tecnologia
Felipe de Souza Bueno - CRB 8/8577

Bertucci, João Daniel Rubim, 1995-
B462s Sistema de transmissão/recepção com código corretor de erro (hamming) utilizando linguagem VHDL / João Daniel Rubim Bertucci, Nyahn Ekyê Fernandes Duarte. – Limeira, SP : [s.n.], 2024.

Orientador(es): Cristhof Johann Roosen Runge.
Trabalho de Conclusão de Curso (Graduação) – Universidade Estadual de Campinas (UNICAMP), Faculdade de Tecnologia.

1. Sistemas de transmissão de dados. 2. VHDL (Linguagem descritiva de hardware). I. Duarte, Nyahn Ekyê Fernandes, 1998-. II. Roosen Runge, Cristhof Johann, 1971-. III. Universidade Estadual de Campinas (UNICAMP). Faculdade de Tecnologia. IV. Título.

Informações complementares

Título em outro idioma: Transmission/reception system with error correction code (hamming) using VHDL language
Palavras-chave em inglês:
Data transmissão systems
VHDL (Computer hardware description language)
Área de concentração: Engenharia de Telecomunicações
Titulação: Bacharel
Banca examinadora:
Cristhof Johann Roosen Runge [Orientador]
Talía Simões dos Santos Ximenes
Edson Luiz Ursini
Data de entrega do trabalho definitivo: 06-12-2024

Autores: João Daniel Rubim Bertucci e Nyahn Ekyê Fernandes Duarte

Título: Sistema de Transmissão/Recepção com código corretor de erro (Hamming) utilizando linguagem VHDL

Natureza: Monografia (Graduação em Engenharia de Telecomunicações)

Instituição: Faculdade de Tecnologia, Universidade Estadual de Campinas

Aprovado em: _____/_____/_____.

BANCA EXAMINADORA

Orientador

Prof. Dr. Cristhof Johann Roosen Runge

Universidade Estadual de Campinas

Membro da banca

Profª. Dra. Talia Simões dos Santos Ximenes

Universidade Estadual de Campinas

Este exemplar corresponde à versão final da monografia aprovada.

Membro da banca

Prof. Dr. Edson Luiz Ursini

Universidade Estadual de Campinas

AGRADECIMENTOS

Eu, João Daniel, primeiramente gostaria de dedicar este trabalho ao meu avô Walter Bertucci, que infelizmente não está mais entre nós. Seu conhecimento serviu como fonte de inspiração e apoio durante minha jornada acadêmica, e cada minuto de convívio foram fundamentais para que eu pudesse chegar até aqui. Agradeço meus pais, João Geraldo e Maria Eliza, que não mediram esforços para que eu pudesse concluir este curso. Agradeço também à minha irmã Marília, minhas avós Lélia e Maria Aparecida, e minha tia Liane pelo total apoio e carinho.

Eu, Nyahn Ekyê, primeiramente gostaria de dedicar este trabalho ao sobrinho Koladê Babatundê Porto Duarte, que veio a falecer precocemente este ano, também dedico aos meus falecidos pais Jorge e Cláudia e carrego suas essências comigo em vida. Agradeço minha tia Rosana, tio José Carlos, primas Jaqueline e Carolina por todo apoio antes e durante a graduação, que me deram forças para continuar. Agradeço meu irmão Kynni Kayodê por sempre se manter forte e me inspirar, e minha companheira Viviane que sempre me incentivou a conquistar meus sonhos. Agradeço também a todos outros familiares e amizades que construí durante minha vida acadêmica, e por todos os momentos que jamais esquecerei.

Por fim, e não menos importante, expressamos nossa gratidão ao professor Cristhof e a todo o corpo docente por acreditarem em nós e nos incentivarem a crescer tanto pessoal quanto profissionalmente. Sua dedicação nos inspirou a buscar melhorias e a gerar impactos positivos na sociedade, preparando-nos para sermos futuros engenheiros de telecomunicações.

BERTUCCI, João Daniel Rubim; DUARTE, Nyahn Ekyê Fernandes. Sistema de transmissão/recepção com código corretor de erro (Hamming) utilizando linguagem VHDL. 2024. 66 f. Monografia (Graduação em Engenharia de Telecomunicações) – Faculdade de Tecnologia, Universidade Estadual de Campinas, Limeira, 2024.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de transmissão e recepção de dados com correção de erros, utilizando o código de Hamming (7,4) implementado em VHDL para síntese em FPGA. O objetivo é criar um kit educacional didático para mostrar a eficácia da correção de erros e a importância da sincronização com bits de *overhead* em comunicações digitais. O sistema utiliza uma sequência pseudoaleatória (PN) de 2^9-1 bits gerada por um gerador PN, implementado por meio de um registrador de deslocamento com realimentação linear (LFSR), correspondendo aos bits de informação a serem transmitidos. O codificador Hamming transforma os bits na saída do gerador PN em palavras-código, adicionando redundância para permitir a correção de erros no receptor. Os dados são organizados em quadros de 40 bits para transmissão, com bits de sincronização inseridos em posições específicas para possibilitar o processo de sincronização no receptor. O receptor processa a entrada serial vinda do canal de transmissão, identificando os bits de sincronização, para extração das palavras-código recebidas, fornecendo-as ao decodificador de Hamming para correção de erros. A implementação destaca não apenas a relevância dos códigos de correção de erros e dos mecanismos de sincronização, mas também a viabilidade de sua aplicação em sistemas embarcados e didáticos.

Palavras-chave: Sequência PN9. Código de Hamming. Sincronismo. Codificação. Correção de Erros.

ABSTRACT

This project presents the development of a data transmission and reception system with error correction using the Hamming (7,4) code implemented in VHDL for FPGA synthesis. The objective is to create an educational kit to demonstrate the effectiveness of error correction and the importance of synchronization with overhead bits in digital communications. The system uses a $2^9 - 1$ bit pseudo-random (PN) sequence generated by a PN generator, implemented through a linear feedback shift register (LFSR), corresponding to the information bits to be transmitted. The Hamming encoder transforms the bits from the PN generator into codewords, adding redundancy to enable error correction at the receiver. The data is organized into 40-bit frames for transmission, with synchronization bits inserted at specific positions to facilitate the synchronization process at the receiver. The receiver processes the serial input from the transmission channel, identifying the synchronization bits for extracting the received codewords and providing them to the Hamming decoder for error correction. The implementation highlights not only the relevance of error correction codes and synchronization mechanisms but also the feasibility of their application in embedded and educational systems.

Keywords: PN9 Sequence. Hamming Code. Synchronization. Coding. Error Correction.

LISTA DE ILUSTRAÇÕES

Figura 1 - Arquitetura genérica de uma placa FPGA e estrutura de bloco lógico 14
Figura 2 - Diagrama de um microcontrolador e de um FPGA 15
Figura 3 - Aplicação do paradigma FTT de alto desempenho 16
Figura 4 - Geração de Sequência Pseudoaleatória (PN) 19
Figura 5 - Representação de códigos corretores de erro 22
Figura 6 - Codificador para Hamming (7,4) 23
Figura 7 - Diagrama de Venn de um Código de Hamming (7,4) 24
Figura 8 - Sistema de Codificação Hamming (7,4) 26
Figura 9 - Processo VHDL - PN9 31
Figura 10 - Processo VHDL - Codificação 33
Figura 11 - Processo VHDL - Sincronização do contador 33
Figura 12 - Processo VHDL - *Enable* dos bits codificados 34
Figura 13 - Processo VHDL - Transmissão de dados codificados e bits de sincronismo ……35
Figura 14 - Processo VHDL - Sincronização de quadro 36
Figura 15 - Processo VHDL - Verificação da palavra de sincronismo 36
Figura 16 - Processo VHDL - Mecanismos de pré-sinc e sincronização 37
Figura 17 - Processo VHDL - Processo de extração da palavra-código 38
Figura 18 - Processo VHDL - Processo de decodificação e correção de erros 39
Figura 19 - Processo VHDL - Processo de serialização da palavra decodificada 40
Figura 20 - Simulação - PN9 e Codificação 42
Figura 21 - Simulação - Quadros de bits e *Enable* 43
Figura 22 - Simulação - Transmissão dos dados e bits de sincronismo 43
Figura 23 - Simulação - Pré-Sincronização 44
Figura 24 - Simulação - Sincronização 44
Figura 25 - Simulação - Extração da palavra-código 45
Figura 26 - Simulação - Saída do receptor 45
Figura 27 - Simulação - Sistema de transmissão e recepção completo 46

LISTA DE ABREVIATURAS E SIGLAS

Bit	*Binary Digit*
VHDL	*VHSIC Hardware Description Language*
FPGA	*Field-Programmable Gate Array*
BER	*Bit Error Rate*
CDMA	*Code Division Multiple Access*
LFSR	*Linear Feedback Shift Register*
PN	*Pseudo-Random Number*
RTL	*Register Transfer Level*
FTT	*Flexible Time-Triggered*
RAM	*Random Access Memory*
IoT	*Internet of Things*
ALU	*Arithmetic Logic Unit*
LUT	*Look-Up Table*
SSL	*Secure Sockets Layer*
WDM	*Wavelength Division Multiplexing*
FEC	*Forward Error Correction*
BCH	*Bose–Chaudhuri–Hocquenghem*
LDPC	*Low-Density Parity-Check*
HDLC	*High-Level Data Link Control*
FIFO	*First In, First Out*

SUMÁRIO

1. **Introdução** 12
   1.1 Contexto 12
   1.2 Motivação 12
   1.3 Objetivo Geral 13
   1.4 Objetivos Específicos 13
2. **Fundamentação Teórica** 14
   2.1 FPGA 14
   2.2 VHDL 16
   2.3 Códigos Pseudoaleatórios (PN) 18
   2.3.1 Geração de PN 18
   2.3.2 Propriedades e Aplicações das PN 19
   2.3.3 Sincronização de PN no Receptor 20
   2.4 Taxa de Erro de Bit (BER) 20
   2.5 Códigos Corretores de Erro 21
   2.5.1 Classificação dos Códigos Corretores de Erros 22
   2.5.2 Código de Hamming 23
   2.5.3 Outros Códigos e Aplicações 27
   2.6 Sincronismo 28
3. **Desenvolvimento** 29
   3.1 Metodologia 29
   3.1.1 Planejamento e Definição do Sistema ….29
   3.1.2 Desenvolvimento do Sistema 29
   3.1.3 Simulação e Testes....30
   3.2 Desenvolvimento do Transmissor 31
   3.3 Desenvolvimento do Receptor 35
4. **Resultados** 41
5. **Conclusão** 47
6. **Referências Bibliográficas** 49

Apêndices 52
Apêndice A.1: Tabela de variáveis do Transmissor 52
Apêndice A.2: Tabela de variáveis do Receptor 54
Apêndice B.1: Código VHDL do Transmissor 56
Apêndice B.2: Código VHDL do Receptor 60
Apêndice B.3: Código VHDL do *Testbench* 64

1 INTRODUÇÃO

1.1 Contexto

Desde os primórdios, os seres humanos sempre buscaram maneiras de expressar suas emoções. No passado, essas formas eram mais simples, como arte rupestre e gestos corporais, e, conforme as sociedades evoluíram, surgiram novos recursos, como cartas e telegramas, que facilitaram o envio de mensagens entre pessoas. Hoje, vivenciamos uma era de grande transformação digital, onde a Internet possibilita uma comunicação instantânea e global, rompendo fronteiras físicas.

Os dispositivos que usamos atualmente, além de nos oferecerem conforto e segurança, trazem uma característica indispensável: a capacidade de se conectarem uns aos outros. No mundo moderno, a conexão está no centro de tudo, permitindo localizar pessoas e objetos, acompanhar transmissões de rádio e TV, consultar a previsão do tempo, enviar mensagens de emergência, operar satélites, além do uso de diversos dispositivos com diferentes tecnologia *wireless* (WiFi, LoRa, Bluetooth, ZigBee, etc.).

Entretanto, no campo das telecomunicações, os sinais transmitidos por antenas, cabos e satélites estão sujeitos a interferências, ruídos e distorções. Fatores internos e externos aos sistemas de transmissão podem introduzir ruídos que comprometem a clareza e a integridade dos dados recebidos, deixando as mensagens imprecisas ou até incompletas. Para garantir a precisão em sistemas de comunicação que exigem alta confiabilidade, são aplicados códigos de correção de erros. Esses códigos adicionam bits extras às mensagens, permitindo ao receptor identificar e corrigir possíveis erros, mesmo quando o sinal passa por interferências no caminho.

1.2 Motivação

A motivação para o desenvolvimento deste projeto é tanto técnica quanto educacional, combinando a necessidade de confiabilidade nas comunicações digitais com a possibilidade de aprendizado prático que ela oferece. Em sistemas de comunicação, devido à presença de erros que podem comprometer a funcionalidade e a segurança, a adoção de estratégias que possam garantir a integridade dos dados é fundamental. Códigos corretores de erro, como os códigos de Hamming, são essenciais nesse contexto. Para estudantes e profissionais da

engenharia, trabalhar em um projeto envolvendo o uso de códigos corretores de erros é uma oportunidade para desenvolver habilidades técnicas, bem como o estudo da teoria de codificação, o uso da linguagem VHDL, além da simulação e síntese de circuitos digitais em FPGA, possibilitando o uso de ferramentas de uso industrial, como ModelSim e Quartus II, aproximando os alunos da prática profissional.

1.3 Objetivo Geral

Este projeto visa desenvolver um sistema de transmissão e recepção de dados funcionando como um kit didático para que alunos e pesquisadores possam explorar de maneira prática os conceitos vistos em disciplinas de engenharia de telecomunicações como Sistemas de Telecomunicações, Transmissão de Sinais e Circuitos Digitais.

1.4 Objetivos Específicos

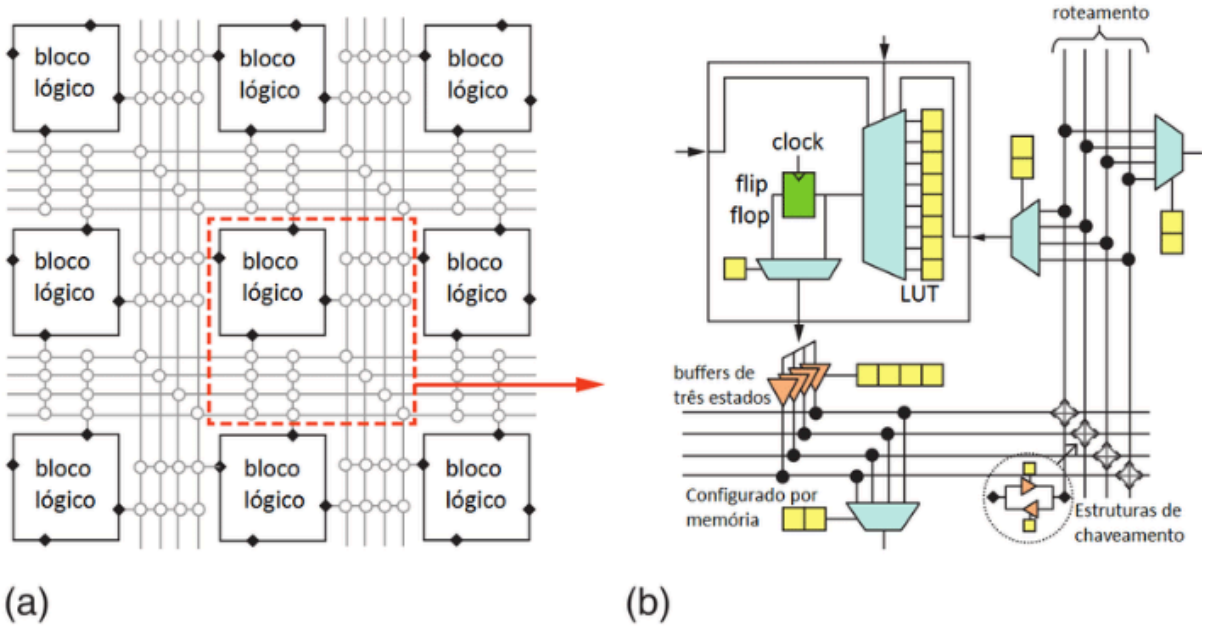
Para alcançar o objetivo do projeto, foram estabelecidas as seguintes metas específicas: desenvolver um transmissor que gera uma sequência pseudoaleatória PN9 e codifica os dados com o código de Hamming (7,4), transformando cada grupo de 4 bits em uma palavra-código de 7 bits para possibilitar a correção de erros; criar uma estrutura de saída serial que organiza as palavras-código em quadros de 40 bits e insere bits de sincronização em posições específicas (1, 9, 17, 25 e 33), possibilitando a sincronização e o processamento dos dados no receptor; desenvolver um receptor capaz de realizar o processo de sincronização dos dados transmitidos utilizando os bits de sincronização, delimitando os quadros e extraindo as palavras-código; e por fim, aplicar e validar o sistema no software Quartus II, sintetizando o código VHDL para garantir sua funcionalidade em hardware real.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 FPGA

Basicamente, os FPGAs são compostos por blocos lógicos configuráveis, chamados CLBs, conectados por uma malha programável que permite montar circuitos de acordo com a necessidade. Esses blocos podem executar desde operações simples, como somas e multiplicações, até cálculos mais complexos. Além disso, muitos FPGAs contam com blocos internos que podem ser utilizados como elementos de memória integrada, como *flip-flops* e RAM, que os tornam ainda mais poderosos [1]. Para programá-los, utiliza-se linguagens especializadas, como VHDL ou Verilog, que descrevem o comportamento desejado do circuito. A Figura 1 mostra, no item "a", uma arquitetura genérica de uma placa FPGA, e, no item b, a estrutura de um bloco lógico e a conexão das trilhas com o roteamento.

Figura 1 - (a) Arquitetura de uma placa FPGA (b) Estrutura de um bloco lógico e conexão das trilhas



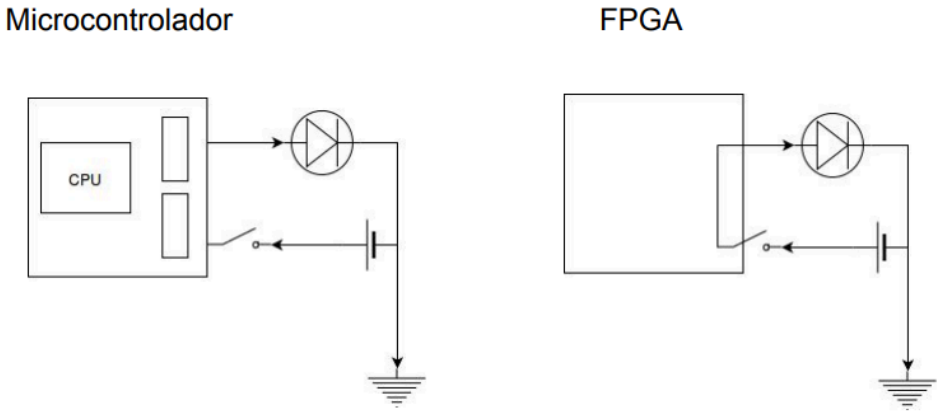
Fonte: [18]

Os microcontroladores e os FPGAs são dispositivos muito usados por engenheiros e entusiastas da eletrônica, cada um com suas características e vantagens. Os microcontroladores são mais simples de usar, pois já integram CPU, memória e periféricos.

São ideais para tarefas sequenciais, como automação e dispositivos IoT, além de normalmente consumirem pouca energia, o que é ótimo para aplicações em que precisam ser eficientes.

Já os FPGAs são mais flexíveis e podem ser configurados para fazer praticamente qualquer tarefa lógica, inclusive processar várias tarefas de forma paralela. Isso os torna perfeitos para trabalhos mais exigentes, como processamento de sinais e o uso em aplicações de inteligência artificial. No entanto, programá-los pode ser uma tarefa mais complexa, sendo que na implementação de circuitos mais complexos, poderá ser observado um aumento no consumo de energia, tornando-se um ponto negativo dependendo do projeto [2]. A Figura 2 mostra uma comparação entre esquemáticos simples de um microcontrolador e de uma placa FPGA.

Figura 2 - Diagrama de um microcontrolador e de um FPGA



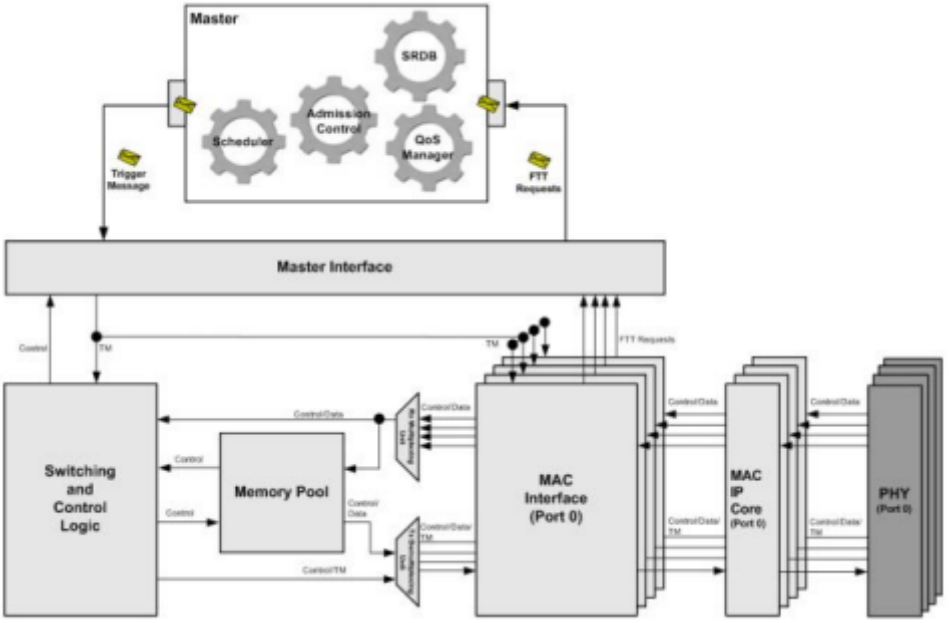
Fonte: [3]

Esses dispositivos, possuem uma arquitetura que divide o chip em três componentes principais: blocos lógicos configuráveis e de funções fixas, blocos de entrada/saída (E/S) e blocos de comutação. Os blocos lógicos são responsáveis pelo processamento das funções digitais e podem ser classificados em diferentes níveis de granularidade. Os blocos mais complexos podem incluir microprocessadores e unidades lógicas aritméticas (ALUs), enquanto blocos de média complexidade podem se constituir de combinações de duas ou mais tabelas de busca (LUTs) e *flip-flops*. Já os blocos mais simples, realizam funções lógicas básicas, como operações de duas entradas, multiplexadores e *flip-flops* individuais. Essa estrutura modular e escalável permite que os dispositivos sejam configurados para atender a uma ampla variedade de aplicações, desde cálculos simples até processamentos altamente complexos [3]. No ambiente da computação de alto desempenho, empresas como a Microsoft

utilizam FPGAs para otimizar algoritmos, como os empregados no Bing e na plataforma de nuvem Azure, demonstrando sua eficiência em aplicações de alta performance.

Os FPGAs são amplamente utilizados em telecomunicações por sua reconfiguração em tempo real. Sua alta performance permite a implementação em hardware de diferentes algoritmos de processamento digital de sinais, como moduladores, demoduladores, código corretores de erro e filtros. Esses blocos podem ser implementados compartilhando a mesma pastilha de silício e possibilitando reconfigurações dinâmicas como no caso da reprogramação "*on the fly*". Em *switches* de grande porte, os FPGAs aliviam a carga de servidores ao processar tarefas intensivas, como *Secure Sockets Layer (*SSL), decodificação de vídeo e aplicação de regras de *firewall* [3]. A Figura 3 apresenta uma implementação do paradigma *Flexible Time-Triggered (FTT)*, que ajusta o escalonamento de tráfego em tempo real, oferecendo uma solução eficiente e adaptável às demandas de alto desempenho em redes de comunicação.

Figura 3 - Aplicação do paradigma FTT de alto desempenho



Fonte: [3]

2.2 VHDL

O VHDL é uma linguagem essencial para quem trabalha com circuitos digitais, tanto na indústria quanto no meio acadêmico. Criada na década de 1980 pelo Departamento de

Defesa dos EUA, ela permite que engenheiros projetem circuitos complexos de forma eficiente em dispositivos programáveis como FPGAs. Diferentemente, linguagens de programação como C ou Assembly, onde as instruções são executadas uma após a outra, na sequência em que aparecem no código, o VHDL por ser uma linguagem de descrição de hardware, e capaz de produzir circuitos que são capazes de realizar atribuições de valores de forma simultânea e paralela. Desse modo, temos um estilo de programação declarativo, focado em descrever o que o circuito deve fazer, em vez de instruir passo a passo, como ocorre nas linguagens imperativas [4].

Um dos grandes pontos fortes da VHDL é sua flexibilidade, pois ela suporta diferentes níveis de abstração no projeto de hardware. Isso significa que os engenheiros podem descrever o circuito em um nível comportamental, no nível de transferência entre registradores (RTL) ou até no nível de portas lógicas. Essa abordagem é especialmente útil em projetos complexos, permitindo que o design comece de um nível mais amplo (*top-down*) e seja dividido em componentes menores e mais fáceis de gerenciar. Além disso, a reutilização de componentes é um benefício significativo: módulos descritos em VHDL podem ser reaproveitados em diferentes projetos, economizando tempo e garantindo consistência entre os *designs* [5].

Outro diferencial dessa linguagem é a possibilidade de simular e verificar o comportamento dos sistemas digitais antes mesmo da implementação física. Isso permite que os projetistas validem a lógica do sistema em um ambiente controlado, identificando e corrigindo erros desde as fases iniciais. Como resultado, minimizam-se retrabalhos e revisões mais custosas na fase final de implementação [6]. Também é possível especificar o momento exato em que uma atribuição deve ocorrer, oferecendo um controle preciso sobre o comportamento do sistema. Por exemplo:

```
A <= 40; -- atribuição válida no próximo intervalo de tempo discreto
X <= Y after 20 ns; -- atribuição válida após 20 nanosegundos.
```

Esse conceito é importante dentro do contexto do projeto que será descrito, para entendermos como pode ser feita uma inserção de erros, onde força-se uma subida para "1", de duração de um ciclo de relógio, no dado codificado, após um determinado tempo.

2.3 Sequências Pseudoaleatórias (PN)

As *Pseudo-Random Binary Sequences* (PN), ou Sequências Binárias Pseudoaleatórias, são sinais amplamente utilizados em diversas aplicações, tais como geração de padrões, testes de sistemas de transmissão, codificação, criptografia, aproximação de ruído branco e para medidor de taxa de erro de bits (BER) [7]. Essas sequências possuem propriedades que as tornam fundamentais em sistemas digitais e ópticos, sendo aplicadas em instrumentação de diagnóstico, reconhecimento de padrões binários e técnicas de verificação de paridade, além de encriptação e decriptação de dados [9].

Essas sequências são usadas para simular canais ruidosos, analisar a capacidade de detecção e correção de erros e manter a sincronia entre transmissor e receptor. Sua ampla aplicação decorre das propriedades pseudoaleatórias, evitando assim, a ocorrência de padrões "viciados", que as tornam adequadas para ambientes que exigem sinais de alta qualidade para testes e medições [7].

2.3.1 Geração de PN

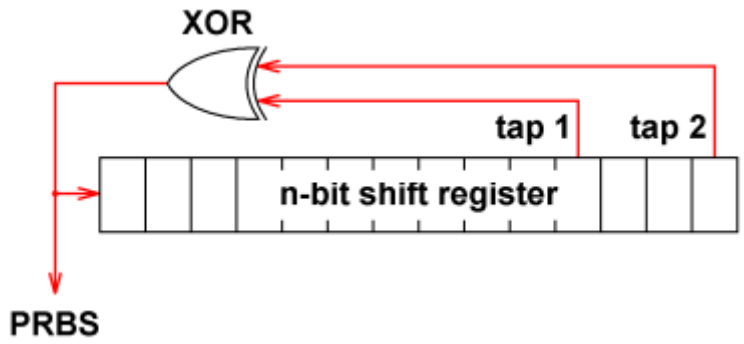
A geração de uma PN é realizada com base em um registrador de deslocamento com realimentação linear (LFSR, *Linear Feedback Shift Register*). Esse tipo de registrador combina *flip-flops* do tipo D em um loop fechado com realimentação baseada em portas lógicas, geralmente operações XOR ou XNOR. As saídas dos *flip-flops* são combinadas com uma função lógica para alimentar o primeiro *flip-flop* do circuito, produzindo uma sequência binária que aparenta ser aleatória. O comprimento máximo da sequência depende do número de estágios do LFSR (n) e do polinômio característico utilizado, conforme a equação:

$$Lpn = 2^n - 1$$

Aqui, Lpn é o comprimento máximo da sequência gerada, e "n*"* representa o número de bits no registrador de deslocamento. Quando o polinômio de realimentação é um polinômio primitivo, a PN atinge seu comprimento máximo e é denominada *M-sequence* (sequência de comprimento máximo) [8]. A Figura 4 mostra um exemplo desse registrador de deslocamento, configurado por uma combinação lógica implementada por meio de uma porta

XOR. O registrador ilustra como a sequência binária é gerada e realimentada, destacando o papel do polinômio de realimentação no comportamento do sistema.

Figura 4 - Geração PN



Fonte: [8]

2.3.2 Propriedades e Aplicações das PN

As PN têm propriedades matemáticas únicas, como o deslocamento e soma (*shift-and-add*), que permite que uma sequência seja deslocada ciclicamente e combinada com uma réplica de si mesma, gerando novas sequências com a mesma periodicidade. Essa técnica é amplamente utilizada em sistemas que requerem maior taxa de repetição sem aumentar a frequência de operação do gerador original. Por exemplo, ao multiplexar duas cópias de uma PN de $2^7 - 1 = 127$ bits, é possível dobrar a taxa de repetição da sequência [9]. Em sistemas de comunicação, sequências pseudoaleatórias de diferentes comprimentos, como PN9 $(2^9 - 1)$ e PN23 $(2^{23} - 1)$, são usadas para medições de BER, ajudando a simular canais ruidosos e avaliar a capacidade de correção de erros de um sistema. Essas sequências também desempenham um papel crucial na sincronização entre transmissor e receptor, garantindo que a sequência gerada no receptor corresponda à sequência transmitida [8].

Em aplicações práticas, o polinômio $x^9 + x^5 + 1$ (PN9) é comumente utilizado para geração de PN devido à sua simplicidade e capacidade de gerar sequências confiáveis para sistemas de comunicação de alta velocidade.

2.3.3 Sincronização de PN no Receptor

Para medir a taxa de erro de bits, é necessário que o receptor conte com uma fonte de dados de referência, geralmente um LFSR configurado com os mesmos parâmetros do transmissor. A sincronização é realizada preenchendo o registrador do receptor com os bits recebidos. No entanto, qualquer erro inicial no preenchimento pode causar um desalinhamento entre as sequências, resultando em uma alta taxa de erro percebida.

Para lidar com esse problema, são implementados métodos de ressincronização no receptor. Esses métodos funcionam em três estados principais:

- Ressincronização: O registrador do receptor é preenchido com os bits recebidos até que *n* bits sem erro sejam armazenados.

- Verificação: Durante este estado, um número de bits consecutivos é monitorado para verificar a ausência de erros. Qualquer erro detectado reinicia o processo de ressincronização.

- Sincronizado**:** Quando o receptor entra nesse estado, os erros são corretamente avaliados e não há necessidade de ajustes adicionais, salvo novas detecções de perda de sincronia.

Esse processo permite garantir a integridade dos dados transmitidos e minimizar discrepâncias, mesmo em canais ruidosos ou com falhas de sincronização temporárias [9].

2.4 Taxa De Erro De Bit (BER)

A Taxa de Erro de Bits (BER, do inglês *Bit Error Rate*) é uma métrica fundamental na avaliação da qualidade de sistemas de comunicação digital. Ela representa a proporção de bits transmitidos que são recebidos com erro, fornecendo uma medida direta da confiabilidade do sistema. Com o avanço das telecomunicações digitais e a crescente integração de voz, dados e vídeo em alta velocidade, a BER tornou-se uma medida essencial como forma de avaliar a integridade das transmissões, especialmente em enlaces onde erros podem comprometer significativamente o desempenho [10].

Diversos fatores influenciam a BER, incluindo potência de transmissão, ruído no canal, interferências externas e a técnica de modulação utilizada. O sincronismo entre transmissor e receptor é um dos aspectos mais críticos, já que desalinhamentos podem comprometer a delimitação correta de quadros, gerando erros adicionais. Esse impacto é particularmente notável em sistemas de transmissão serial, que dependem da detecção precisa de padrões de sincronização [10].

A BER é geralmente expressa em potências de 10, com valores que variam conforme a aplicação. Por exemplo, redes *Ethernet* locais de 10 Mbit/s requerem uma BER inferior a 10^{-9}, enquanto sistemas ópticos avançados baseados em multiplexação por divisão de comprimento de onda (WDM) demandam uma BER de até 10^{-15}. Esses requisitos refletem a crescente necessidade de precisão em sistemas modernos de comunicação, especialmente em redes de alta velocidade e baixa tolerância a erros [10].

A avaliação da BER está diretamente relacionada ao desempenho do sistema diante de ruídos e distorções. Ferramentas como o diagrama de olho são amplamente empregadas para diagnosticar problemas no sinal, permitindo identificar atenuações, *jitter* e falhas de sincronização. Para medições mais precisas de BER, sequências de teste como as PN são utilizadas. Essas sequências possibilitam uma comparação bit a bit entre transmissor e receptor, utilizando portas XOR para identificar discrepâncias. A sincronização adequada das PN é indispensável para evitar erros artificiais gerados por desalinhamentos entre as sequências transmitidas e recebidas [10].

Mais do que uma métrica de desempenho, a BER é uma ferramenta estratégica para otimizar sistemas de comunicação digital. Em aplicações como redes de fibra óptica e comunicações sem fio, onde a tolerância a erros é mínima, a análise contínua da BER assegura que os sistemas operem de acordo com os padrões de confiabilidade necessários para o ambiente tecnológico atual [10].

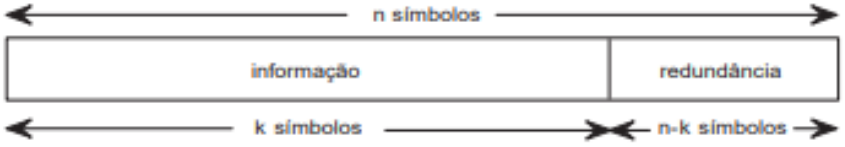
2.5 Códigos Corretores De Erro

Após discutir a importância da Taxa de Erro de Bits (BER) para avaliar a qualidade de sistemas de comunicação digital, é essencial abordar as técnicas empregadas para mitigar erros durante a transmissão de dados: os códigos corretores de erros (FEC, *Forward Error Correction*). Esses códigos são elementos cruciais em sistemas de comunicação e

armazenamento, garantindo a confiabilidade dos dados em canais sujeitos a ruídos e interferências inevitáveis.

Os códigos corretores de erros funcionam adicionando redundância à mensagem original, permitindo ao receptor identificar e corrigir erros sem a necessidade de retransmissão. Isso é particularmente importante em sistemas onde a latência ou o custo de retransmissões é elevado, como comunicações por satélite ou redes móveis. A Figura 5 mostra uma sequência codificada, com “n” símbolos, sendo eles, “k” de dados e ‘‘n-k’’ de redundância, que são utilizados pelo receptor para detectar e corrigir discrepâncias, assegurando a integridade dos dados mesmo em cenários com alta taxa de erros ([11]; [13]).

Figura 5 - Representação de códigos corretores de erro.



Fonte: [15]

2.5.1. Classificação dos Códigos Corretores de Erros

Os códigos corretores de erros podem ser classificados em duas categorias principais: códigos de bloco e códigos convolucionais, cada qual adaptado a diferentes cenários e requisitos de sistemas

- Códigos de Bloco: Esses códigos processam os dados em blocos independentes, associando a cada bloco um conjunto de bits redundantes que permitem a detecção e correção de erros. Exemplos notáveis incluem o código de Hamming e o código BCH. Pela simplicidade e eficiência, são amplamente empregados em sistemas de armazenamento e comunicações de curta distância, onde os dados são processados em segmentos autônomos [11].

- Códigos Convolucionais: Diferentemente dos códigos de bloco, os códigos convolucionais operam de forma contínua, gerando saídas que dependem dos dados de entrada atuais e dos estados anteriores. Essa característica aumenta sua robustez contra

erros acumulados ao longo da transmissão, tornando-os ideais para cenários de longa distância e alta taxa de erro, como em comunicações por satélite e redes móveis. Esses códigos também são frequentemente utilizados em sistemas que requerem correções em tempo real [12].

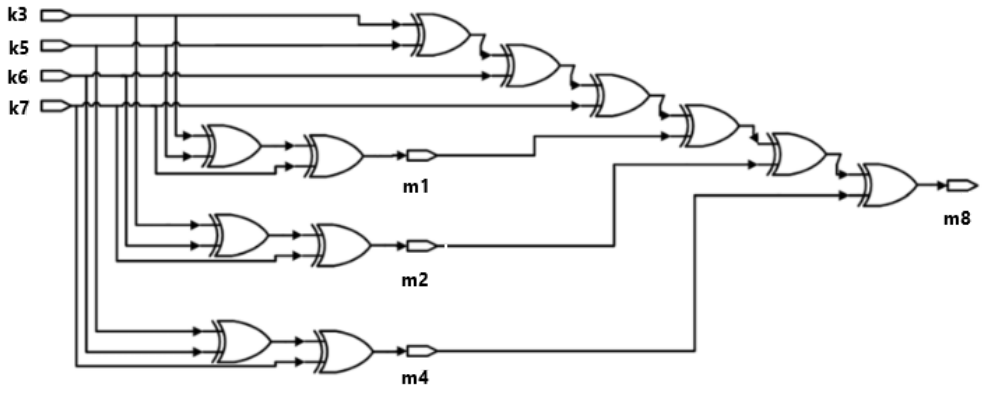
2.5.2 Código De Hamming

Os códigos de Hamming, desenvolvidos pelo matemático Richard Hamming, são uma técnica eficiente e simples para detectar e corrigir erros, com o objetivo de tornar as transmissões de dados mais confiáveis, especialmente em contextos de processamento de sinais e telecomunicações. Eles ajudam a minimizar os efeitos do ruído, adicionando bits extras, conhecidos como bits de redundância, em posições específicas dos blocos de dados. Esses bits de redundância permitem, por exemplo,que o sistema detecte até dois erros e corrija um erro por bloco transmitido, para o caso do código de Hamming (7,4) [20][21]. O número de bits de redundância necessários é determinado pela fórmula:

$$k \ + \ m \leq 2^{m} - 1$$

onde "k" é o número de bits de dados e " m " o número de bits de redundância. O código gerado é representado pela notação (n, k), em que "n" é o total de bits (dados e redundância), e "k" é o número de bits de dados [14]. A Figura 6 mostra um esquema de um codificador de Hamming implementado a partir do uso de portas XOR.

Figura 6 - Codificador para Hamming (7,4)



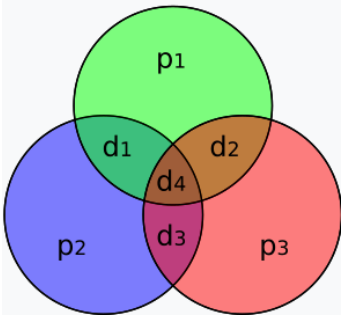
Fonte: [19]

Os códigos de Hamming podem ser representados e trabalhados de diferentes maneiras, cada uma com características específicas e aplicações práticas distintas. As abordagens mais comuns incluem a representação algorítmica, os diagramas de conjuntos (Venn) e a abordagem matricial, que é fundamental para a análise e implementação em sistemas modernos e será aplicada neste projeto.

- Abordagem por Diagramas

Os códigos de Hamming podem ser representados por diagramas de Venn, onde os bits de redundância são associados a conjuntos e os dados são representados pelas interseções entre esses conjuntos. Embora essa visualização seja útil para mensagens menores, torna-se complexa para códigos maiores devido à dificuldade de mapear muitas interseções [20]. A Figura 7 mostra um exemplo desse diagrama, sendo d = bits de informação, e p = bits de redundância.

Figura 7 - Diagrama de Venn de um Código de Hamming (7,4)



Fonte: [20]

- Abordagem Algorítmica

Na abordagem algorítmica, os bits de redundância são calculados diretamente com base em subconjuntos dos bits de dados. Para o código de Hamming (7,4), os bits de redundância $p1, p2, p3$ são inseridos de forma que cada um verifique a paridade de subconjuntos sobrepostos dos bits de dados $d1, d2, d3, d4$. A Tabela 1 a seguir resume os subconjuntos avaliados:

Tabela 1: Relação entre bits de redundância e subconjuntos avaliados.

Bit de redundância	Subconjunto avaliado
$P1$	$d1, d2, d4$
$P2$	$d1, d3, d4$
$P3$	$d2, d3, d4$

Fonte: Autoral

Cada bit de redundância é responsável por garantir que o número de bits "1" no subconjunto avaliado seja par. Na decodificação, aplica-se a mesma lógica ao vetor recebido para identificar discrepâncias, corrigir erros em até um bit e recuperar os dados originais [21].

- Abordagem Matricial

A abordagem matricial é uma ferramenta poderosa para a codificação e decodificação de palavras-código. Ela oferece uma representação sistemática e escalável, adequada para implementações computacionais e análises teóricas. Na codificação matricial, uma palavra-código é obtida pela multiplicação de um vetor de dados u pela matriz geradora G. Como o código é sistemático (a parte correspondente aos bits de informação na palavra-código não é alterada), a matriz G pode ser dividida em duas partes: à esquerda, encontramos uma matriz identidade de tamanho 4, chamada I_k, que representa os bits originais da mensagem. À direita, está a parte Q^T, que contém os bits de redundância [15]. Dessa forma, para o código de Hamming (7,4), a matriz G pode ser representada como:

$$G = [I_k | Q^T]$$

A palavra-código v é então calculada como:

$$v = uG$$

Para verificar a integridade de uma palavra recebida, utiliza-se a matriz de verificação de paridade H, que é definida como:

$$H = [Q \ | I_m]$$

A verificação é feita multiplicando a palavra recebida r pela transposta de H:

$$S = rH^T$$

O vetor S, conhecido como síndrome, indica se houve erros na transmissão. Um valor $S \neq 0$ identifica a posição do bit com erro, permitindo sua correção. Esse método é extremamente eficiente, pois as matrizes G e H podem ser pré-calculadas e armazenadas, permitindo a execução rápida de operações durante a codificação e decodificação [22]. A Figura 8 apresenta o funcionamento do sistema de codificação Hamming (7,4), sendo o item (a) é a relação dos valores dos dados da entrada u e a saída codificada c. No item (c), é apresentada a matriz geradora do código Hamming (7,4) e o item (b) apresenta a matriz de verificação de paridade H.

Figura 8 - Sistema de Codificação Hamming (7,4)

Entrada	Saída	
$u_1u_2u_3u_4$	$c_1c_2c_3c_4$	$c_5c_6c_7$
0000	0000	000
0001	0001	011
0010	0010	110
0011	0011	101
0100	0100	111
0101	0101	100
0110	0110	001
0111	0111	010
1000	1000	101
1001	1001	110
1010	1010	011
1011	1011	000
1100	1100	010
1101	1101	001
1110	1110	100
1111	1111	111

(a)

$$H = \overbrace{\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}}^{Q} \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{I_m} \ m \times n$$

(b)

$$G = \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{I_k} \overbrace{\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}}^{Q^T} \ k \times n$$

(c)

Fonte: [15]

A distância de Hamming entre duas palavras-código é definida como o número de posições mínimas essas palavras diferem. Por exemplo, para as palavras $v1 = [0110011]$ e $v2 = [1110110]$ a distância de Hamming é 3 [22]. Essa propriedade está diretamente relacionada à capacidade de detecção e correção de erros do código, sendo que:

- A capacidade de detecção: é $dmin - 1$, onde $dmin$ é a distância mínima entre duas palavras-código qualquer.

- A capacidade de correção: é dada por $t = [(dmin - 1)/2]$. Para o código de Hamming (7,4), $dmin$ = 3, o que possibilita corrigir 1 erro e detectar até 2 erros [22].

2.5.3. Outros Códigos e Aplicações

Além dos códigos de bloco e convolucionais, outros códigos corretores de erros desempenham papéis fundamentais em sistemas mais avançados, como:

- Código Reed-Solomon: Reconhecido por sua capacidade de corrigir longas sequências de erros, é comumente utilizado em CDs, DVDs e sistemas ópticos. Sua robustez o torna ideal para cenários onde erros ocorrem em rajadas [16].

- Códigos LDPC (*Low-Density Parity-Check*): Amplamente empregados em padrões modernos, como redes Wi-Fi e 5G, devido à sua eficiência em canais de alta capacidade e suporte a altas taxas de transmissão [17].

Esses avanços são fundamentados na teoria desenvolvida por Claude Shannon, que estabeleceu os limites fundamentais da transmissão de informações em canais ruidosos, e Richard Hamming, que aplicou esses conceitos para criar sistemas práticos de correção de erros. Essas contribuições moldaram a base dos códigos modernos de correção e desempenham um papel essencial na evolução das tecnologias de comunicação [16].

2.6 Sincronismo

Em sistemas de transmissão de dados, a sincronização é essencial para garantir comunicação eficiente e sem erros. Ela é bastante usada em transmissões seriais, onde os dados são enviados um bit por vez. A sincronização assegura que o receptor saiba exatamente quando e onde fazer a leitura de cada bit, garantindo a recepção correta da informação. Antes da transmissão, os dados passam por um processo de codificação, que adiciona bits extras para a detecção e correção de erros. Esses bits, conhecidos como bits de redundância, permitem que o receptor verifique a integridade dos dados e corrija erros eventualmente inseridos pelo canal durante a transmissão [23].

A sincronização é essencial para que o receptor decodifique esses dados corretamente. O receptor precisa saber onde os bits começam e terminam, o que pode ser complicado em canais afetados por ruídos ou variações temporais, que causam erros na leitura. Para resolver isso, adiciona-se um conjunto de bits de sincronização ao quadro de dados. Esses bits, posicionados em locais fixos, são usados pelo receptor no processo de sincronização de quadro para identificar os momentos exatos de leitura. Por exemplo, podem ser colocados nos primeiros bits do quadro ou em intervalos regulares, indicando o início de cada bloco de dados. Ao receber os dados, o receptor usa esses pontos de sincronização para alinhar sua leitura. Após detectar os bits de sincronização, o receptor pode extrair os dados nas posições corretas [23].

A sincronização de bits oferece várias vantagens, especialmente quando combinada com mecanismos de correção de erros, como neste projeto. Ela assegura a recepção correta dos dados, mesmo em condições de transmissão adversas, como canais ruidosos. Porém, a introdução de bits extras para sincronização tem um custo em eficiência. Cada quadro transmitido inclui dados que não carregam informações úteis, mas são necessários para a sincronização [24].

3. DESENVOLVIMENTO

3.1 Metodologia

O desenvolvimento deste trabalho seguiu uma abordagem estruturada, combinando modelagem teórica e simulação computacional, visando uma validação prática, com foco na implementação de um sistema de transmissão e recepção de dados com correção de erros baseada no código de Hamming (7,4) para ser sintetizado em uma FPGA. A seguir, descrevem-se as etapas principais.

3.1.1 Planejamento e Definição do Sistema

O sistema foi projetado como um kit didático, com duas funções principais: transmissão e recepção de dados. Para garantir a confiabilidade da transmissão, foi utilizado o código de Hamming (7,4), que permite a correção de um erro de bit em cada palavra transmitida. A funcionalidade do sistema foi planejada para destacar conceitos como geração de sequências pseudoaleatórias, codificação e decodificação de dados, e sincronização de quadros. As ferramentas utilizadas foram:

- Quartus II: Utilizado para a modelagem e síntese do código VHDL, permitindo a geração do arquivo de *bitstream* para FPGA.

- FPGA: Utilizado como dispositivo alvo para validar o funcionamento do sistema em *hardware*.

- ModelSim: Base do projeto e onde foram simulados os módulos para verificação do comportamento lógico e funcional do sistema.

3.1.2 Desenvolvimento do Sistema

Foi implementado um registrador de deslocamento com realimentação linear (LFSR) para gerar uma sequência pseudoaleatória (PN9). O polinômio de realimentação foi configurado para produzir sequências confiáveis, atendendo às necessidades de testes e

simulação. A base lógica do PN9 foi desenvolvida na disciplina de Introdução ao Trabalho de Conclusão de Curso, e adaptada para o projeto.

A codificação dos dados foi realizada utilizando uma matriz geradora G, que transformou blocos de 4 bits em palavras de 7 bits, com adição de bits de redundância para detecção e correção de erros. A lógica de codificação e decodificação de Hamming foi baseada em [25].

Cada quadro foi estruturado com 40 bits, sendo 35 de dados e 5 de sincronização. Os bits de sincronização foram posicionados em posições fixas (1, 9, 17, 25 e 33) para facilitar a delimitação no receptor.

O receptor utilizou um circuito de sincronização de quadro para identificar os bits de sincronismo e alinhar os quadros. Após a delimitação, as palavras-código foram extraídas, decodificadas e convertidas para o formato paralelo (tanto o codificador como o decodificador de Hamming utilizaram entradas paralelas para o processamento). Em seguida, os dados foram reconvertidos para o formato serial para compatibilidade com outros dispositivos.

3.1.3 Simulação e Testes

- Simulação no ModelSim**:** Foram realizados testes unitários nos módulos do transmissor e receptor, para validar a correção de erros e a sincronização dos quadros.

- *Testbench* Integrado**:** Foi desenvolvido um módulo de teste para avaliar o sistema completo, integrando transmissor e receptor.

- Validação no Quartus II**:** O código VHDL foi sintetizado e testado em FPGA, verificando-se a integridade e desempenho do sistema.

Os testes analisaram aspectos como a correção de erros, a taxa de erro de bits (BER) e a sincronização dos quadros. Essa abordagem modular e iterativa permitiu atingir os objetivos propostos, validando a aplicabilidade do sistema no contexto acadêmico e industrial.

3.2 Desenvolvimento do Transmissor

O projeto iniciou-se com a criação de um código na linguagem VHDL, feito no ambiente do Quartus II, de uma sequência pseudoaleatória de PN9. O polinômio usado foi $x^9 + x^5 + 1$, gerado por meio de um registrador de deslocamento com realimentação linear (LFSR).

No trecho de código apresentado na Figura 9, o registrador "reg_tx" é inicializado com uma semente, de valor 100000000. A cada borda de subida do *clock*, o bit mais significativo é atualizado com a operação XOR dos bits nas posições 8 e 4. Após isso, desloca-se o registrador para a direita, inserindo novo bit. Os 4 bits menos significativos são armazenados em "out_PN_fifo" para serem usados no próximo processo de codificação.

Figura 9 - Processo VHDL - PN9

```
PROCESS(clk)
BEGIN
IF(rising_edge(clk)) THEN
IF(enable_inf = '1') THEN

reg_tx(0) <= (reg_tx(8) xor reg_tx(4));
reg_tx(8 downto 1) <= reg_tx(7 downto 0);

out_PN_fifo(0) <= reg_tx(8);
out_PN_fifo(3 downto 1) <= out_PN_fifo(2 downto 0);

END IF;
END IF;
END PROCESS;
```

Fonte: Autoral

O processo de codificação percorre cada linha da matriz G, executando operações XOR para obter os bits de redundância. Esses bits permitem a correção de erros de um único bit por palavra codificada no receptor. A codificação transforma cada bloco de 4 bits de dados em uma palavra-código de 7 bits, adicionando 3 bits de redundância. A matriz geradora G do código de Hamming (7,4) é definida como:

$$\begin{matrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Essa matriz foi implementada com uma abordagem adaptada, utilizando um referencial invertido em relação à forma clássica. Normalmente, as linhas da matriz G representam os bits de saída da palavra-código, enquanto as colunas correspondem aos bits da mensagem de entrada. No entanto, para otimizar o desenvolvimento no ambiente de hardware descritivo, optou-se por organizar G como sua transposta (G^T), onde as colunas representam os bits de saída e as linhas os bits de entrada. Essa adaptação foi realizada para simplificar os cálculos no código VHDL, sem prejuízo às propriedades matemáticas do código Hamming. Assim, a matriz preserva sua funcionalidade, garantindo que as palavras geradas pertençam ao espaço vetorial definido pelo código, mantendo sua capacidade de correção de erros intacta.

No trecho de código na Figura 10, é apresentada a parte do código responsável pela multiplicação bit a bit entre a matriz G e “out_PN_fifo”, onde estão armazenados os dados da sequência PN9, conforme descrito anteriormente. O XOR dos resultados de cada linha gera os bits codificados.

Figura 10 - Processo VHDL - Codificação

```
PROCESS(clk)
BEGIN
IF(rising_edge(clk)) THEN
IF(enable_inf = '0') THEN

for i in 0 to 6 loop
for j in 0 to 3 loop
temp_G(i,j) <= G(i,j) and out_PN_fifo(j);
end loop;

yo(i) <= temp_G(i,0) xor temp_G(i,1) xor temp_G(i,2) xor temp_G(i,3);
end loop;

dados_coded <= yo;
END IF;
END IF;
END PROCESS;
```

Fonte: Autoral

Após a codificação, foi desenvolvido um mecanismo que permite a sincronização adequada no receptor. Para isso, as sequências foram organizadas em quadros de 40 bits, sendo 35 bits destinados às palavras codificadas e 5 bits reservados para sincronização.

Os bits de sincronização são adicionados nas posições 1, 9, 17, 25 e 33, conforme no trecho de código apresentado na Figura 11. Para verificação dessa sincronia, foi definida uma máscara de sincronização igual a "10011".

Figura 11 - Processo VHDL - Sincronização do contador

```
PROCESS(clk)
BEGIN
IF(rising_edge(clk)) THEN
IF(reset = '1') THEN
IF count = 40 THEN

count <= 1;

ELSE
count <= count + 1;

END IF;
IF(count = 1 or count = 9 or count = 17 or count = 25 or count = 33) THEN

enable_dados <= '0';

ELSE
enable_dados <= '1';

END IF;
END IF;
END IF;
END PROCESS;
```

Fonte: Autoral

Na geração dos *enables* dos bits codificados, foi desenvolvido um processo em que, a cada borda de subida do *clock*, caso o sinal de *reset* esteja ativado, o contador é incrementado de 1 até 7, reiniciando para 1 ao atingir o valor máximo. Esse contador assegura que 7 bits sejam transmitidos antes de reiniciar o ciclo.

No trecho de código apresentado na Figura 12, observa-se que, quando o sinal "conta_bits" atinge o valor 5 ou superior, o sinal "enable_inf" é desativado, o que indica a necessidade de inserção de bits de redundância. Esse mecanismo de controle dinâmico garante que apenas os bits de dados úteis sejam transmitidos nos primeiros ciclos, enquanto os ciclos subsequentes são destinados aos bits de redundância, assegurando a integridade da comunicação.

Figura 12 - Processo VHDL - *Enable* dos bits codificados

```
PROCESS(clk)
BEGIN
IF(rising_edge(clk)) THEN
IF(reset = '1') THEN
IF enable_dados = '1' THEN
IF conta_bits = 7 THEN

conta_bits <= 1;

ELSE
conta_bits <= conta_bits + 1;
END IF;

IF conta_bits < 5 THEN

enable_inf <= '1';
ELSE
enable_inf <= '0';

END IF;
END IF;
END IF;
END IF;
END PROCESS;
```

Fonte: Autoral

Durante a transmissão, os dados codificados são armazenados no registrador, enquanto os bits de sincronização são transmitidos com base na leitura constante da máscara. Esse procedimento permite ao receptor identificar corretamente o início de novos quadros de dados.

No trecho de código apresentado na Figura 13, o envio é controlado pelo contador "conta_sinc", responsável por gerenciar a sequência e a temporização dos bits de sincronização, garantindo uma transmissão ordenada e sincronizada. Quando o sinal "enable_dados" está ativo ('1'), o sistema realiza o deslocamento dos bits armazenados para a

direita, preparando-os para transmissão sequencial. Nesse processo, o bit mais significativo do registrador é enviado para a saída “d_out”, possibilitando a transmissão serializada dos dados codificados.

Figura 13 - Processo VHDL - Transmissão de dados codificados e bits de sincronismo

```
PROCESS(clk)
VARIABLE conta_sinc: integer RANGE 0 to 5 := 0;
BEGIN
IF(rising_edge(clk)) THEN
IF enable_dados = '0' THEN

dados_coded_in <= dados_coded;

IF conta_sinc = 5 THEN
conta_sinc := 0;
END IF;
d_out <= mask(conta_sinc);
conta_sinc := conta_sinc + 1;
END IF;
IF (enable_dados = '1') THEN

dados_coded_in(6 downto 1) <= dados_coded_in(5 downto 0);
d_out <= dados_coded_in(6);
END IF;
END IF;
END PROCESS;
```

Fonte: Autoral

3.3 Desenvolvimento do receptor

No receptor, a entrada serial é processada por um sistema que identifica os bits de sincronização em suas respectivas posições, permitindo a delimitação precisa dos quadros. Para isso, é utilizado um circuito sincronizador que realiza a busca pelas posições dos bits de sincronização, variando de 1 a 40. Quando o contador alcança o limite de 40, ele é reiniciado, e o contador de sincronismo é zerado e deslocado em um período de relógio para reiniciar a busca pela palavra de sincronismo.

No trecho de código apresentado na Figura 14, o sinal “conta_sinc” é incrementado a cada detecção correta de um bit da máscara nas posições específicas, permitindo ao receptor validar a sincronização com base no número de correspondências encontradas dentro do quadro.

Figura 14 - Processo VHDL - Sincronização de quadro

```
PROCESS(clk)
BEGIN
IF (rising_edge(clk)) THEN
IF (reset = '0') THEN

conta_sinc <= 0;
count <= 0;

ELSIF (reset = '1') THEN

IF count = 40 THEN

count <= 1;
conta_sinc <= 0;

ELSE
count <= count + 1;
END IF;
```

Fonte: Autoral

A verificação da palavra de sincronismo é efetuada por meio da comparação entre os bits recebidos e os valores da máscara nas posições específicas do quadro onde os bits de sincronismo foram transmitidos.

No trecho de código apresentado na Figura 15, a cada correspondência encontrada, o contador de sincronismo é incrementado. Por exemplo, se o bit recebido na posição 1 coincide com o primeiro bit da “mask(0)”, o sinal “conta_sinc” é ajustado para 1. Nas posições subsequentes, cada correspondência adicional com os bits da máscara resulta em um incremento do valor de “conta_sinc”. Esse mecanismo permite ao receptor validar se a palavra de sincronismo foi recebida corretamente e determinar o alinhamento necessário para a sincronização dos quadros.

Figura 15 - Processo VHDL - Verificação da palavra de sincronismo

```
IF (count = 1 AND d_in = mask(0)) THEN
conta_sinc <= 1;
END IF;
IF (count = 9 AND d_in = mask(1)) THEN
conta_sinc <= conta_sinc + 1;
END IF;
IF (count = 17 AND d_in = mask(2)) THEN
conta_sinc <= conta_sinc + 1;
END IF;
IF (count = 25 AND d_in = mask(3)) THEN
conta_sinc <= conta_sinc + 1;
END IF;
IF (count = 33 AND d_in = mask(4)) THEN
conta_sinc <= conta_sinc + 1;
END IF;
```

Fonte: Autoral

Quando mais de 3 dos 5 bits da máscara são corretamente identificados no quadro atual, o circuito de pré-sincronismo é incrementado, indicando um aumento na probabilidade de sincronização. Se menos de 3 bits forem detectados, o contador é decrementado, indicando uma possível perda de alinhamento.

No trecho de código apresentado na Figura 16, quando o valor do contador atinge 5, o receptor interpreta que está sincronizado, ativando o sinal “enable_quadro” para processar os dados do quadro. Em contrapartida, caso o contador "sinc" retorne a 0, o receptor identifica a perda de sincronização, desativando o sinal “enable_quadro” e reiniciando a busca pela palavra de sincronismo. Adicionalmente, o sinal “enable_dados” é ativado em todas as posições do quadro que não contém bits de sincronismo, permitindo a extração dos dados úteis transmitidos.

Figura 16 - Processo VHDL - Mecanismos de pré-sinc e sincronização

```
IF (count = 34 AND conta_sinc > 3 AND sinc < 5) THEN
sinc <= sinc + 1;
ELSIF (count = 34 AND conta_sinc < 3 AND sinc > 0) THEN
sinc <= sinc - 1;
END IF;


IF (sinc = 5) THEN
enable_quadro <= '1';
ELSIF (sinc = 0) THEN
enable_quadro <= '0';
IF count < 40 THEN
count <= count + 1;
ELSE
count <= 2;
END IF;
END IF;

IF (count = 1 OR count = 9 OR count = 17 OR count = 25 OR count = 33) THEN
enable_dados <= '0';

ELSE
enable_dados <= '1';

END IF;
END IF;
END IF;
END PROCESS;
```

Fonte: Autoral

Após a sincronização ser estabelecida, o receptor inicia a extração dos dados úteis do fluxo recebido. Esse processo ocorre somente quando o quadro está sincronizado, isto é, quando o sinal “enable_quadro” está ativo (“enable_quadro = 1”) e a posição atual não corresponde a um bit de sincronismo (“enable_dados = 1”).

No trecho de código apresentado na Figura 17, os bits recebidos são armazenados a partir da posição menos significativa do vetor “data_word”, enquanto os bits previamente armazenados são deslocados para a direita, preservando o conteúdo existente. Esse deslocamento possibilita a construção sequencial da palavra de dados recebida em formato paralelo, acumulando os bits úteis até a formação completa da palavra.

Figura 17 - Processo VHDL - Processo de extração da palavra-código

```
PROCESS(clk)
BEGIN
IF (falling_edge(clk)) THEN
IF enable_quadro = '1' THEN
IF enable_dados = '1' THEN


data_word(0)<= d_in;
data_word(6 downto 1)<=data_word(5 downto 0);



END IF;
END IF;
END IF;
END PROCESS;
```

Fonte: Autoral

No processo de decodificação e correção de erros em um sistema de transmissão que utiliza o código de Hamming (7,4), são estabelecidas etapas específicas para identificar e corrigir eventuais erros em uma palavra recebida. Esse funcionamento baseia-se no cálculo dos bits de redundância, que são comparados aos valores esperados para determinar a ocorrência de erros.

No trecho de código apresentado na Figura 18, observa-se a presença de dois laços aninhados (“for i” e “for j”) que percorrem a matriz de paridade H e realizam operações lógicas AND entre os elementos dessa matriz e os bits do código recebido, armazenando os resultados intermediários em “temp_H”. Posteriormente, calcula-se F, uma sequência de bits que indica a posição do erro, caso exista, utilizando operações XOR sobre os valores de “temp_H”. Se F for igual a "000", conclui-se que não há erros, e a saída corrigida, “corrected_code”, é idêntica ao código recebido. Caso contrário, o índice do erro é identificado, e o bit correspondente é invertido para corrigir o erro, indicando que um erro foi detectado (“error_detected <= 1”). Após a correção, o código ajustado é processado para extrair os 4 bits de dados originais, disponibilizados no sinal “data_out”.

Figura 18 - Processo VHDL - Processo de decodificação e correção de erros

```
process(clk)
begin
if rising_edge(clk) then

for i in 0 to 2 loop
for j in 0 to 6 loop
temp_H(i, j) <= H(i, j) and code_in(j);
end loop;

F(i) <= temp_H(i, 0) xor temp_H(i, 1) xor temp_H(i, 2) xor temp_H(i, 3) xor
temp_H(i, 4) xor temp_H(i, 5) xor temp_H(i, 6);
end loop;

if F = "000" then
index_error <= -1;
else
index_error <= to_integer(unsigned(F)) - 1;
end if;

if index_error /= -1 then
temp <= code_in;
temp(index_error) <= not code_in(index_error);
error_detected <= '1';
else
temp <= code_in;
error_detected <= '0';
end if;

corrected_code <= temp;
data_out <= temp(6) & temp(5) & temp(4) & temp(2);

end if;
end process;
```

Fonte: Autoral

Por fim, conforme apresentado no trecho de código na Figura 19, a palavra decodificada é transferida para o registro “shift_reg”, que é utilizado para gerar a saída serial. A cada borda de subida do *clock*, os bits armazenados no registro são deslocados para a direita, enquanto o bit mais significativo é enviado para a saída “serial_out”. Esse mecanismo assegura que os dados sejam disponibilizados na forma serial, possibilitando sua conexão com diversos dispositivos.

Figura 19 - Processo VHDL - Processo de serialização da palavra decodificada

```
process(clk)
begin
if rising_edge(clk) then

if enable_dados = '0' then
shift_reg <= corrected_code;

elsif enable_dados = '1' then
serial_out <= shift_reg(6);
shift_reg <= shift_reg(5 downto 0) & '0';

end if;
end if;
end process;
```

Fonte: Autoral

4 RESULTADOS

Os resultados apresentados neste capítulo exploram de forma prática o funcionamento do sistema de transmissão e recepção de dados, conectando teoria e aplicação no contexto da engenharia de telecomunicações. Durante o desenvolvimento do projeto, foram enfrentados desafios significativos que demandaram algumas adaptações. Inicialmente, foi utilizado o software Quartus II para validar cada bloco do sistema separadamente. No entanto, a integração desses blocos em dois módulos principais, transmissão (Tx) e recepção (Rx), trouxe o primeiro impasse, pois era necessário garantir coerência entre os módulos, principalmente em sua sincronização. Com o suporte do orientador, foi possível estruturar a integração, organizando cada etapa do sistema e implementando um *testbench* para validar a funcionalidade do projeto.

Porém, o Quartus II apresentou limitações para a simulação de *testbenchs*, sendo necessário migrar para o ModelSim, uma ferramenta mais adequada para realizar simulações completas. Essa mudança foi fundamental, permitindo a validação de cada etapa do processo, facilitando a análise e possibilitando uma abordagem mais didática.

Outro desafio enfrentado foi a indisponibilidade de acesso prático à placa FPGA durante a disciplina TT411 - Circuitos Digitais II, ministrada remotamente durante a pandemia. Essa limitação impediu a realização de testes no hardware por falta de familiaridade do dispositivo, exigindo uma abordagem alternativa. Foi desenvolvido, então, um sistema completo que pudesse ser simulado virtualmente, mantendo os conceitos teóricos e garantindo que o projeto atendesse aos objetivos propostos. Esse enfoque não apenas validou o sistema no ambiente virtual como também preparou o projeto para futuras implementações práticas.

Essas simulações têm um papel crucial no desenvolvimento de sistemas digitais. Elas não apenas ajudam a identificar possíveis ajustes e melhorias, mas também preparam o projeto para uma futura implementação prática em dispositivos reais, como placas FPGA ou ASICs. O estudo traz à tona questões importantes, como a organização dos quadros de dados, o alinhamento entre transmissor e receptor, e a recuperação precisa das informações transmitidas.

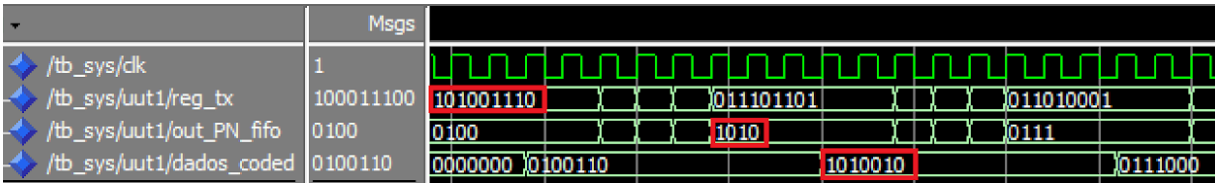
Esse processo de validação do sistema em um ambiente virtual reforça a confiança na aplicação prática, aproximando o projeto de possíveis usos reais em redes e dispositivos embarcados. Com isso, este trabalho mostra a importância de simulações no desenvolvimento

de soluções em telecomunicações, ligando o aprendizado teórico às demandas práticas da faculdade, com a aplicação de disciplinas ministradas no curso de Engenharia de Telecomunicações como circuitos digitais, sistema de telecomunicações e transmissão de sinais, e do mercado de trabalho (também no cotidiano) como o uso em dispositivos *wireless.*

A seguir, serão apresentados os resultados obtidos no projeto desenvolvido, destacando cada etapa do sistema. Por meio das simulações realizadas no ModelSim, será possível visualizar o comportamento dos sinais no *waveform* e analisar o funcionamento completo do sistema, desde a criação, codificação e transmissão dos dados, passando pela recepção, correção de erros e decodificação síncrona, devidamente alinhada com o transmissor. Essas simulações não apenas validam o projeto, mas também oferecem uma visão clara de como cada componente contribui para o desempenho geral do sistema, permitindo compreender a integração entre os módulos e o fluxo completo da comunicação.

A Figura 20 apresenta a codificação da sequência PN9 no início do processo de transmissão. Na simulação, o sinal de relógio (“clk”) é exibido na primeira linha, com um período de 20ns. Na segunda linha, observa-se o registrador “reg_tx” responsável pela geração da sequência pseudoaleatória, enquanto os 4 primeiros dados dessa sequência são apresentados na terceira linha (“out_PN_fifo”). Por fim, a palavra codificada é representada na última linha (“dados_coded”).

Figura 20 - Simulação - PN9 e Codificação

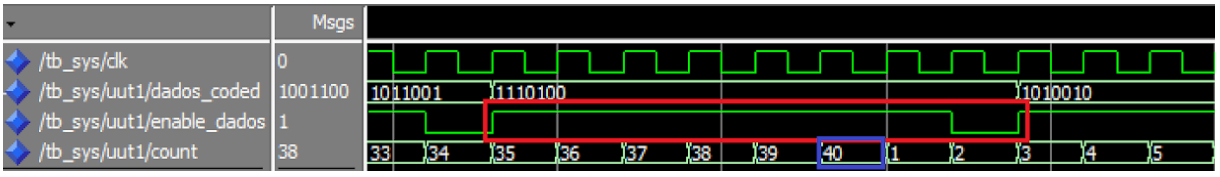


Fonte: Autoral

Analisando o comportamento dessa parte do código, identifica-se um registrador de deslocamento que implementa o polinômio $x^9 + x^5 + 1$, uma FIFO que armazena os 4 primeiros bits do registrador e uma codificação que, por meio do Código de Hamming (7,4), transforma os dados armazenados na FIFO em uma palavra-código.

Na Figura 21, é apresentada a delimitação dos quadros e o funcionamento do sinal *enable* no transmissor. Cada quadro possui 40 bits, conforme destacado em azul na figura. Dentre esses 40 bits, é possível observar a presença de 1 bit de sincronismo e 7 bits de dados, intercalados 5 vezes até completar o quadro.

Figura 21 - Simulação - Quadros de bits e *Enable*

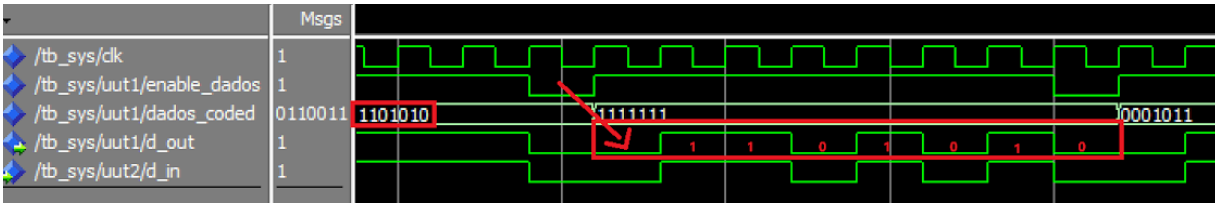


Fonte: Autoral

Quando o sinal “enable_dados” está ativo (nível alto), ocorre a transmissão da palavra-código. Por outro lado, quando o sinal está inativo (nível baixo), a posição correspondente contém um bit de sincronismo, interrompendo momentaneamente a transmissão da palavra-código.

No processo de transmissão, apresentado na Figura 22, a palavra-código é serializada, de modo que, a cada ciclo de *clock*, um dos 8 bits da sequência (sincronismo + dados) é transmitido. A palavra-código, inicialmente em paralelo no sinal “dados_coded”, é convertida para o formato serial na saída “d_out”.

Figura 22 - Simulação - Transmissão dos dados e bits de sincronismo

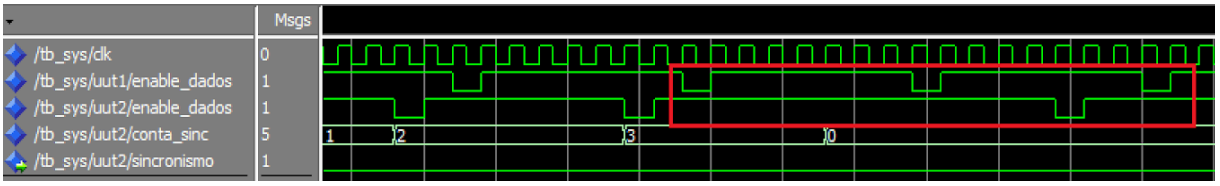


Fonte: Autoral

A simulação também apresenta a entrada “d_in”, conectada à saída do transmissor por meio de uma ligação estabelecida no módulo de *testbench*, permitindo a transmissão dos dados para o receptor. Além disso, destaca-se o papel do sinal “enable_dados” na transmissão: antes de cada sequência transmitida, é inserido um bit de sincronismo, conforme indicado pela seta em vermelho na figura.

Na Figura 23, é apresentado o processo de aquisição de sincronismo no receptor. Os sinais “enable_dados” do transmissor (“uut1”) e do receptor (“uut2”), representados na segunda e terceira linha, respectivamente, não estão inicialmente sincronizados no espaço temporal. O sincronismo é verificado pelo sinal “conta_sinc”, comparando a máscara “10011” com os bits recebidos. Quando ocorre uma correspondência, o contador é incrementado em 1. Caso contrário, o valor é reiniciado para 0, como apresentado na simulação.

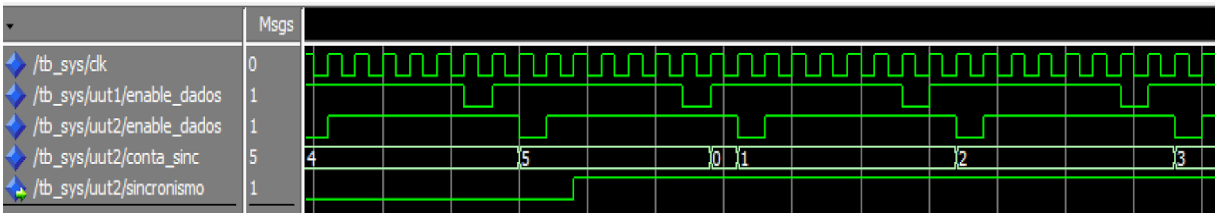
Figura 23 - Simulação - Pré Sincronização



Fonte: Autoral

Na Figura 24, observa-se que, quando o valor da máscara é detectado 5 vezes consecutivas, o sistema é considerado sincronizado, resultando na ativação do sinal de sincronismo, conforme descrito anteriormente. Nesse momento, os sinais “enable_dados” do transmissor e do receptor passam a operar em perfeita sincronia.

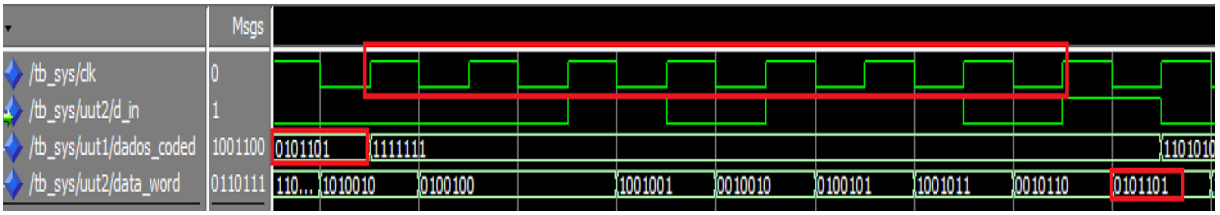
Figura 24 - Simulação - Sincronização



Fonte: Autoral

No trecho de simulação apresentado na Figura 25, observa-se a extração da palavra-código “0101101” a partir da entrada “d_in”, com a remoção dos bits de sincronização. Esse procedimento é viabilizado pelos processos anteriores, que permitiram ao decodificador localizar o início das palavras-código e extrair exclusivamente os dados relevantes.

Figura 25 - Simulação - Extração da palavra-código

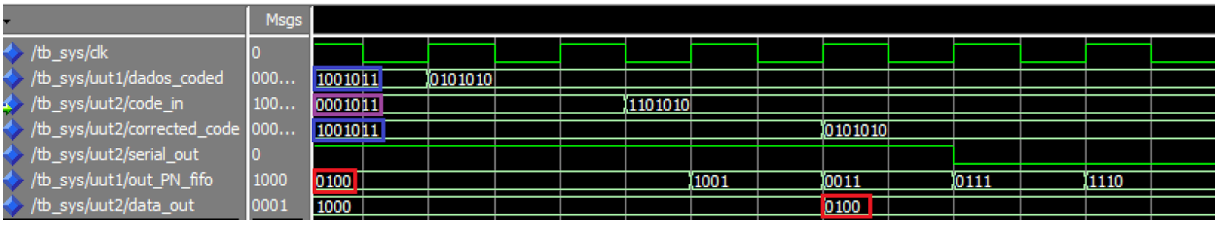


Fonte: Autoral

Na simulação, é possível identificar a palavra codificada gerada pelo transmissor, destacada na figura, e, logo abaixo, a mesma palavra já decodificada pelo receptor. Cabe ressaltar que a palavra decodificada no receptor surge com um atraso de 7 ciclos de *clock* em relação à palavra transmitida, devido ao processo de conversão serial-paralelo necessário para reconstruir a palavra a partir da sequência serial.

A Figura 26 apresenta o funcionamento prático do decodificador Hamming (7,4). As palavras-código geradas pelo transmissor são apresentadas em azul e, em seguida, submetidas à inserção de erros. Na sequência, o sinal “code_in”, destacado em roxo, indica a inserção de um erro no bit mais significativo da palavra-código.

Figura 26 - Simulação - Saída do receptor

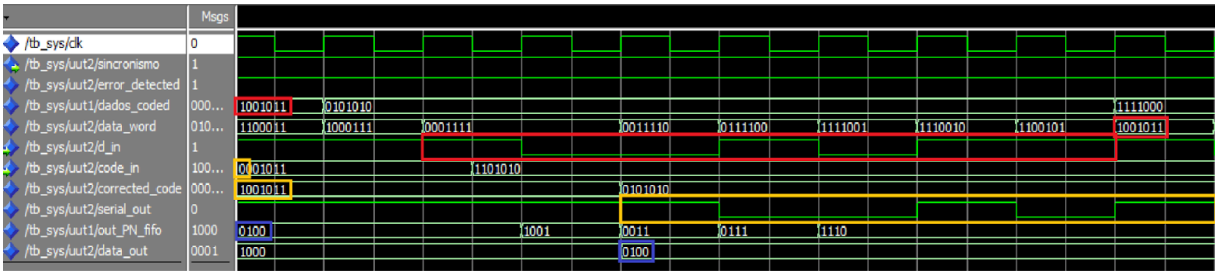


Fonte: Autoral

Na quarta linha, observa-se a mesma sequência após a aplicação do processo de correção de erros, comprovando a eficácia do decodificador. As marcações em vermelho validam o funcionamento do sistema de codificação e decodificação Hamming, mostrando que os 4 bits originais da sequência PN9, gerada no início do projeto, são corretamente decodificados ao final do processo.

Por fim, a Figura 27 apresenta a simulação do projeto em sua totalidade, abrangendo desde os sinais de sincronismo e de erro detectado, representados na segunda e terceira linhas, respectivamente, até a validação do sistema, indicada pelas marcações em azul.

Figura 27 - Simulação - Sistema de transmissão e recepção completo



Fonte: Autoral

O sinal de sincronismo permanece constantemente em nível alto, pois a análise foi realizada em um intervalo temporal no qual a sincronização já estava consolidada. Na terceira linha, o sinal de erro detectado também se mantém em nível alto, uma vez que erros foram inseridos deliberadamente em todas as palavras-código para facilitar a análise das simulações.

Na quarta e quinta linha, observa-se a palavra-código que será analisada e, posteriormente, sua extração no receptor. Esse processo inclui a inserção do bit de sincronismo e a serialização da saída do transmissor, simulando uma possível integração com outros dispositivos. Nas linhas 7 e 8, destacadas em amarelo, verifica-se a introdução de um erro de bit no bit mais significativo da palavra-código, que é corrigido em seguida por meio dos cálculos realizados com a matriz H no decodificador. A linha 9, também em amarelo, representa a serialização da palavra decodificada e corrigida, visando futura implementação em um sistema real.

Nas duas últimas linhas da simulação, ocorre a validação do sistema de transmissão e recepção. Observa-se que os 4 bits menos significativos da sequência PN9, gerados no início do módulo de transmissão, são corretamente recuperados ao final do receptor, comprovando o funcionamento eficaz do sistema.

5 CONCLUSÃO

O sistema foi projetado para funcionar como um kit acadêmico, oferecendo uma plataforma prática para estudar técnicas como geração de sequências pseudoaleatórias, codificação e decodificação de dados, além de sincronização de quadros. A aplicação do código de Hamming tornou possível a implementação de mecanismos de detecção e correção de erros, demonstrando de forma clara a relevância da confiabilidade nas comunicações digitais.

A abordagem adotada combinou modelagem teórica e simulação computacional, utilizando ferramentas como Quartus II e ModelSim, essenciais para a síntese e simulação dos módulos em VHDL. Inicialmente, foi necessário readaptar aos ambientes desses softwares, utilizados pela última vez no segundo ano do curso, na disciplina de “Circuitos Digitais II”. Após esse período de aprendizado e readaptação, junto a estudos teóricos e a um processo intensivo de tentativa e erro, foi possível familiarizar-se com o software Quartus II, com a própria linguagem VHDL e FPGA. Um adendo, nesse processo, foi a mudança do ambiente e simulação do “*Waveform*” do Quartus II para o ModelSim, onde foi possível criar um módulo de *testbench*, facilitando a inserção de parâmetros na simulação. Além disso, os sinais da arquitetura poderiam ser vistos na simulação, sem a necessidade da criação de um pino de saída na entidade.

O sistema desenvolvido é uma solução completa de transmissão e recepção utilizando o código de Hamming, que se destaca por sua robustez, proporcionada pela geração pseudoaleatória, e confiabilidade, garantida pelo sincronismo eficiente. Além disso, o projeto possibilita validar, de forma prática, a eficácia do código corretor de erros em um ambiente controlado. Embora seja uma aplicação relativamente simples, este trabalho serve como base para expandir a abordagem para sistemas mais complexos, como a implementação de um gerador PN23 (sequência pseudoaleatória de maior ordem) em conjunto a um código BCH (15,11), que possui maior capacidade de correção (2 erros corrigidos), permitindo também a análise de desempenho sob diferentes taxas de erro.

Outro ponto importante é a aplicabilidade prática deste projeto e suas variações em dispositivos FPGA. Apesar das dificuldades enfrentadas pela dupla no mapeamento do sistema para a placa, isso reforça a necessidade de aprofundar as atividades didáticas no curso de telecomunicações voltadas ao uso dessa tecnologia, dada sua versatilidade para aplicações acadêmicas e profissionais. Porém, isso não foi um impedimento para a conclusão deste

projeto, pois buscamos uma simulação e análise detalhada do sistema, o que possibilita consolidar conceitos teóricos e práticos e abrir caminhos para novas pesquisas e desenvolvimentos em sistemas digitais e de telecomunicações.

Como ferramenta educacional, o projeto oferece inúmeros benefícios, como a modernização no ensino no campo das telecomunicações. Ele possibilita aos alunos vivenciar, de maneira interativa, conceitos teóricos abordados em sala de aula, facilitando o entendimento de temas que fazem (mas não se limitam) parte das áreas de circuitos digitais, sistemas de telecomunicações e transmissão de sinais. Além disso, promove o uso dos softwares de engenharia Quartus II e Modelsim, da linguagem VHDL e das placas FPGAs.

O projeto também incentiva a resolução de problemas reais na comunicação digital e na própria criação de um projeto embarcado. Espera-se que este trabalho inspire novas iniciativas que promovam a aproximação entre o meio acadêmico e as demandas do mercado de trabalho, contribuindo para o avanço de soluções tecnológicas, como aplicações em IoT, LoRa, Zigbee, sistemas ópticos e comunicações via satélite. Além disso, o projeto visa à formação de profissionais qualificados e inovadores, alinhados às necessidades atuais do setor.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[1] EASTLAND, Nate. *FPGA Configurable Logic Block*. Disponível em: https://digilent.com/blog/fpga-configurable-logic-block/. Acesso em: 06 dez. 2024.

[2] PCBPORTUGAL. *FPGA vs Microcontrolador*. Disponível em: https://pcbportugal.com/fpga-vs-Microcontrolador.html. Acesso em: 18 nov. 2024.

[3] BRANDÃO, Bruno. *Introdução ao FPGA*. Instituto de Matemática e Estatística – USP. Disponível em: https://linux.ime.usp.br/~brunobra/pdf/FPGA.pdf. Acesso em: 18 nov. 2024.

[4] HEXSEL, Roberto André**.** *VHDL: uma introdução ao VHDL*. Versão de 1 de março de 2021. Disponível em: https://www.inf.ufpr.br/roberto/ci210/vhdl.pdf. Acesso em: 21 nov. 2024.

[5] UNIVERSIDADE DE SÃO PAULO. *Apostila de Introdução ao VHDL*. 2014. Disponível em: https://edisciplinas.usp.br/pluginfile.php/530833/mod_resource/content/1/Apostila%20de%20Introdu%C3%A7%C3%A3o%20a%20VHDL_2014.pdf. Acesso em: 18 nov. 2024.

[6] MOREIRA, L. D. *Introdução ao VHDL*. Universidade do Estado do Rio de Janeiro. Disponível em: http://www.eng.uerj.br/~ldmm/arquitetura/Introducao_a_VHDL.pdf. Acesso em: 18 nov. 2024.

[7] ADVANTEST. *A pseudo-random binary sequence (PRBS)*. Disponível em: https://www3.advantest.com/documents/11348/3e95df23-22f5-441e-8598-f1d99c2382cb#:~:text=A%20pseudo%20random%20binary%20sequence. Acesso em: 18 nov. 2024.

[8] KUBÍČEK, X.; KOVÁČ, M. PRBS test sequence synchronization in bit error measurement of FSO links. In: *Proceedings of the 14th Conference on Student EEICT 2008*. Brno University of Technology, 2008. Disponível em: https://www.eeict.cz/eeict_download/archiv/sborniky/EEICT_2008_sbornik/03-Doktorske%20projekty/01-Elektronika%20a%20komunikace/08-xkubic18.pdf. Acesso em: 26 jun. 2024.

[9] SPRINGER. *Advanced techniques in PRBS synchronization and error detection*. Disponível em: https://link.springer.com/article/10.1007/s11082-021-03329-5. Acesso em: 20 nov. 2024.

[10] MAXWELL. *Pseudo-Random Binary Sequence*. Pontifícia Universidade Católica do Rio de Janeiro. Disponível em: https://www.maxwell.vrac.puc-rio.br/11319/11319_3.PDF. Acesso em: 18 nov. 2024.

[11] ALMEIDA, Gabriel Marchesan. *Códigos Corretores de Erros em Hardware para Sistemas de Telecomando e Telemetria em Aplicações Espaciais.* 2007. Disponível em: https://tede2.pucrs.br/tede2/bitstream/tede/5273/1/389779.pdf. . Acesso em: 18 nov. 2024.

[12] KYOTO UNIVERSITY. *Introduction to Error-Correcting Codes*. *Colloquium on Error Correction*. Disponível em: https://www.kurims.kyoto-u.ac.jp/EMIS/journals/em/docs/coloquios/NE-1.04.pdf. Acesso em: 18 nov. 2024.

[13] SPRINGER. *Forward Error Correction Techniques*. Disponível em: https://link.springer.com/chapter/10.1007/978-981-33-4687-1_24. Acesso em: 18 nov. 2024.

[14] VOB, João. *Correção de Erros em Transmissões*. Universidade Federal de Pernambuco. Disponível em: https://www.cin.ufpe.br/~jvob/correcao.html. Acesso em: 18 nov. 2024.

[15] PEDRONI, V. A. *Circuit Design with VHDL*. MIT Press, 2010.

[16] SOCIEDADE BRASILEIRA DE MATEMÁTICA APLICADA E COMPUTACIONAL. *Análise de Algoritmos de Correção de Erros*. Disponível em: https://proceedings.sbmac.org.br/sbmac/article/view/4276/4336. Acesso em: 18 nov. 2024.

[17] FIRER, Marcelo. *Códigos Corretores de Erros* – Notas de Aula. Universidade Estadual de Campinas – UNICAMP. Disponível em: https://www.ime.unicamp.br/~mfirer/3NotasFoz2006.pdf. Acesso em: 18 nov. 2024.

[18] SILVA, J.; PEREIRA, A.; SOUSA, M. *Implementação de um controlador digital em FPGA por síntese em alto nível: estudo de caso*. Disponível em: https://www.researchgate.net/figure/Figura-21-a-Arquitetura-generica-de-um-FPGA-b-Estrutura-de-um-bloco-logico-e-sua_fig1_329950763. Acesso em: 21 nov. 2024.

[19] SANTOS, L.; COSTA, M.; ALMEIDA, R. *Formal Verification of ECCs for Memories Using ACL2*. Disponível em: https://www.researchgate.net/figure/Encoder-for-Hamming-7-4-Code_fig3_346315336. Acesso em: 21 nov. 2024.

[20] WIKIPEDIA. *Código de Hamming*. Disponível em: https://pt.wikipedia.org/wiki/C%C3%B3digo_de_Hamming. Acesso em: 21 nov. 2024.

[21] EDSON, L. *Código de Hamming*. Disponível em: https://www.ic.unicamp.br/~edson/disciplinas/mc404/2012-1s/labs/codigo_hamming.html. Acesso em: 23 nov. 2024.

[22] RUNGE, Cristhof Johann Roosen. *Aula sobre códigos corretores de erro*. Disciplina: Transmissão de Sinais – Engenharia de Telecomunicações. Universidade Estadual de Campinas, 2024.

[23] POLO ELETRÔNICA. *O que é bit de sincronização*. Polo Eletrônica. Disponível em: https://poloeletronica.com.br/glossario/o-que-e-bit-de-sincronizacao/. Acesso em: 21 nov. 2024.

[24] ASC. *O objetivo da sincronização de bits*. Instituto Superior de Engenharia do Porto (ISEP). Disponível em: https://www.dei.isep.ipp.pt/~asc/doc/sincronismo.html#:~:text=O%20objectivo%20da%20sincroniza%C3%A7%C3%A3o%20de,bits%20entre%20emissor%20e%20receptor. Acesso em: 21 nov. 2024.

[25] ABDELKAWY, Mohamed Khaled Mohamed; et al. *Extended Hamming Encoder Decoder*. GitHub, 2024. Disponível em: https://github.com/mkmabdelkawy/Extended-Hamming-Encoder-Decoder/blob/master/EHED.vhdl. Acesso em: 23 nov. 2024.

APÊNDICE

Apêndice A.1.

Transmissor:

Nome	Descrição	Tipo	Dimensão	Valor inicial
mask	Máscara de sincronização, usada para identificar quadros no receptor.	std_logic_vector	4 DOWNTO 0	"10011"
dados_coded_in	Armazena temporariamente os dados codificados para a saída serial.	std_logic_vector	6 DOWNTO 0	Não aplicável
conta_bits	Contador para rastrear os 7 bits das palavras-código geradas pelo Hamming.	integer	1 TO 7	1
count	Contador para gerenciar a transmissão de quadros (40 bits).	integer	1 TO 40	1
enable_inf	Sinal que controla a inserção dos bits no registrador de deslocamento.	std_logic	-	Não aplicável
reg_tx	Registrador de deslocamento para gerar a sequência PRBS.	std_logic_vector	8 DOWNTO 0	"10000000"
out_PN_fifo	Sinal que armazena os 4 bits menos significativos da	std_logic_vector	3 DOWNTO	Não aplicável

	sequência PN		0	
dados_coded	Armazena as palavras codificadas (7 bits) geradas pelo código Hamming.	std_logic_vector	6 DOWNTO 0	Não aplicável
G	Matriz temporária usada no cálculo dos bits de redundância.	array	0 TO 6, 0 TO 3	Não aplicável
temp_G	Vetor auxiliar para armazenar os resultados do cálculo XOR (bits codificados).	array	0 TO 6, 0 TO 3	Não aplicável
yo	Vetor auxiliar para armazenar os resultados do cálculo XOR (bits codificados).	std_logic_vector	6 DOWNTO 0	Não aplicável
conta_sinc	Contador para rastrear a sequência de bits de sincronismo enviados.	integer	0 TO 5	0
d_out	Saída serial dos dados (dados ou bits de sincronismo) do transmissor.	std_logic	-	Não aplicável

Apêndice A.2.

Receptor:

Nome	Descrição	Tipo	Dimensão	Valor inicial
mask	Máscara de sincronização usada para comparar os bits recebidos.	std_logic_vector	4 DOWNTO 0	"10011"
conta_sinc	Contador que rastreia a detecção de bits da máscara de sincronização.	integer	0 TO 5	0
count	Contador para identificar a posição atual no quadro (40 bits).	integer	0 TO 40	0
sinc	Contador que rastreia o estado de sincronização (pré ou sincronizado).	integer	0 TO 5	0
enable_dados	Sinal que permite a extração de bits úteis do fluxo recebido.	std_logic	-	Não aplicável
enable_quadro	Sinal para indicar se o quadro atual está sincronizado.	std_logic	-	Não aplicável
data_word	Armazena temporariamente a palavra de dados extraída do fluxo recebido.	std_logic_vector	6 DOWNTO 0	Não aplicável
serial_out	Saída serial dos dados decodificados no receptor.	std_logic	-	Não aplicável

shift_reg	Vetor que armazena a palavra de dados decodificada antes da saída serial.	std_logic_vector	6 DOWNTO 0	Não aplicável
sincronismo	Sinal que indica se o receptor está corretamente sincronizado.	std_logic	-	Não aplicável
d_in	Entrada serial recebida pelo receptor.	std_logic	-	Não aplicável
corrected_code	Sinal de palavra-código corrigida	std_logic_vector	6 DOWNTO 0	Não aplicável
error_detected	Sinal de erro encontrado	std_logic	-	Não aplicável
data_out	Sinal de Mensagem de 4 bits	std_logic_vector	3 DOWNTO 0	Não aplicável
index_error	Variável que indica o índice do bit de erro detectado	integer	-1 TO 6	Não aplicável
F	Sinal da síndrome	std_logic_vector	2 DOWNTO 0	Não aplicável
temp_H	Vetor auxiliar para armazenar os resultados do cálculo XOR	array	0 to 2, 0 to 6	Não aplicável

Apêndice B.1.

```
----------------------------------------------------------------------------
-- Company: FT-UNICAMP
-- Engineers: João Daniel R. Bertucci/Nyahn Ekyê Fernandes Duarte
--
-- Create Date: 07:12:45 11/13/2024
-- Module Name: Tx_coded - Behavioral
-- Project Name: TCC
-- Description: Transmissor
----------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_signed.all;

ENTITY TX_coded IS
      PORT(
         clk : IN std_logic;
            reset : IN std_logic;
            code_out : OUT std_logic_vector(6 downto 0);
            d_out: OUT std_logic
                  );
END TX_coded;

ARCHITECTURE rtl OF TX_coded IS
   CONSTANT mask: std_logic_vector (4 DOWNTO 0):="10011";
   SIGNAL dados_coded_in: std_logic_vector(6 downto 0);
   SIGNAL conta_bits : integer RANGE 1 to 7:=1;
      SIGNAL count : integer RANGE 1 to 40:=1;
   SIGNAL enable_dados: std_logic;
      SIGNAL enable_inf: std_logic;
      SIGNAL reg_tx : std_logic_vector(8 downto 0) := "100000000";
   SIGNAL out_PN_fifo : std_logic_vector(3 downto 0);
      SIGNAL dados_coded : std_logic_vector(6 downto 0);
      TYPE G_MATRIX is array (0 to 6, 0 to 3) of std_logic;
   SIGNAL G : G_MATRIX := (
        "1101",
        "1011",
```

```
        "1000",
        "0111",
        "0100",
        "0010",
        "0001"
          );


            signal temp_G : G_MATRIX;
    signal yo : std_logic_vector(6 downto 0);

   BEGIN

PROCESS(clk)

      BEGIN
            IF(rising_edge(clk)) THEN
                  IF(reset = '1') THEN

                    IF count = 40 THEN
                            count <=1;
                        ELSE
                          count <= count + 1 ;
                          END IF;

                    IF(count = 1 or count =9 or count =17 or count =25 or
count =33  ) THEN


                      enable_dados <= '0';

                    ELSE


                             enable_dados <='1';
                    END IF;
--
                  END IF;
            END IF;
      END PROCESS;
```

```
PROCESS(clk)

      BEGIN
            IF(rising_edge(clk)) THEN
                  IF(reset = '1') THEN
                    IF enable_dados = '1'THEN
              IF conta_bits = 7 THEN
                         conta_bits <= 1;
              ELSE
                         conta_bits <= conta_bits+1;
                        END IF;
                    IF conta_bits < 5 THEN
                        enable_inf <='1';
                        ELSE
                        enable_inf <='0';
                        END IF;
                  END IF;
                  END IF;
            END IF;
      END PROCESS;


PROCESS (clk)

BEGIN

            IF(rising_edge(clk)) THEN
              IF(enable_inf = '1') THEN
                    reg_tx(0) <= (reg_tx(8) xor reg_tx(4));
                reg_tx (8 downto 1) <=reg_tx(7 downto 0);
                  out_PN_fifo(0) <= reg_tx(8);
                  out_PN_fifo(3 downto 1) <= out_PN_fifo(2 downto 0);
              END IF;
    END IF;
END PROCESS;

PROCESS(clk)
    BEGIN
        IF(rising_edge(clk)) THEN
            IF(enable_inf = '0') THEN
```

```
                for i in 0 to 6 loop
                    for j in 0 to 3 loop
                        temp_G(i,j) <= G(i,j) and out_PN_fifo(j);
                    end loop;

                    yo(i) <= temp_G(i,0) xor temp_G(i,1) xor temp_G(i,2)
xor temp_G(i,3);
                end loop;


dados_coded <= yo;
code_out <= dados_coded;

END IF;
END IF;
END PROCESS;

PROCESS (clk)
VARIABLE conta_sinc: integer RANGE 0 to 5:=0;
BEGIN

      IF(rising_edge(clk)) THEN
              IF enable_dados = '0' THEN
              dados_coded_in <= dados_coded;
              IF conta_sinc = 5 THEN
                conta_sinc :=0;
                END IF;
                d_out <= mask (conta_sinc);
                conta_sinc := conta_sinc+1;
                END IF;
                IF (enable_dados = '1')THEN
                 dados_coded_in(6 downto 1) <= dados_coded_in(5 downto 0);
                    d_out <= dados_coded_in(6);
              END IF;
      END IF;
END PROCESS;

END rtl;
```

Apêndice B.2.

```
----------------------------------------------------------------------------
-- Company: FT-UNICAMP
-- Engineers: João Daniel R. Bertucci/Nyahn Ekyê Fernandes Duarte
--
-- Create Date: 16:23:26 11/18/2024
-- Module Name: Rx_coded - Behavioral
-- Project Name: TCC
-- Description: Receptor
----------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

ENTITY RX_coded IS
    PORT(
       clk : IN std_logic;
       reset : IN std_logic;
       d_in: IN std_logic;
       code_in : IN std_logic_vector (6 DOWNTO 0);
       sincronismo: OUT std_logic
    );
END RX_coded;

ARCHITECTURE rtl OF RX_coded IS

type H_MATRIX is array (0 to 2, 0 to 6) of std_logic;
signal H : H_MATRIX := (
"1010101",
"0110011",
"0001111"
   );

signal temp_H : H_MATRIX;
signal F : std_logic_vector(2 downto 0);
signal index_error : integer range -1 to 6;
signal temp : std_logic_vector(6 downto 0);
SIGNAL mask: std_logic_vector (4 DOWNTO 0) := "10011";
```

```
SIGNAL conta_sinc: integer RANGE 0 TO 5 := 0;
SIGNAL count: integer RANGE 0 TO 40 := 0;
SIGNAL sinc: integer RANGE 0 TO 5 := 0;
SIGNAL enable_dados: std_logic;
SIGNAL enable_quadro: std_logic;
SIGNAL data_word: std_logic_vector (6 DOWNTO 0);
SIGNAL serial_out : std_logic;
signal data_out : std_logic_vector (3 DOWNTO 0);
signal corrected_code : std_logic_vector(6 downto 0);
signal error_detected : std_logic;
signal shift_reg : std_logic_vector(6 downto 0);

BEGIN
sincronismo <= enable_quadro;

PROCESS(clk)
BEGIN
    IF (rising_edge(clk)) THEN
        IF (reset = '0') THEN
            conta_sinc <= 0;
            count <= 0;
        ELSIF (reset = '1') THEN
            IF count = 40 THEN
                count <= 1;
                conta_sinc <= 0;
            ELSE
                count <= count + 1;
            END IF;

            IF (count = 1 AND d_in = mask(0)) THEN
                conta_sinc <= 1;
            END IF;
            IF (count = 9 AND d_in = mask(1)) THEN
                conta_sinc <= conta_sinc + 1;
            END IF;
            IF (count = 17 AND d_in = mask(2)) THEN
                conta_sinc <= conta_sinc + 1;
            END IF;
            IF (count = 25 AND d_in = mask(3)) THEN
                conta_sinc <= conta_sinc + 1;
```

```
            END IF;
            IF (count = 33 AND d_in = mask(4)) THEN
                conta_sinc <= conta_sinc + 1;
            END IF;

            IF (count = 34 AND conta_sinc > 3 AND sinc < 5) THEN
                sinc <= sinc + 1;
            ELSIF (count = 34 AND conta_sinc < 3 AND sinc > 0) THEN
                sinc <= sinc - 1;
            END IF;

            IF (sinc = 5) THEN
                enable_quadro <= '1';
            ELSIF (sinc = 0) THEN
                enable_quadro <= '0';
                IF count < 40 THEN
                    count <= count + 1;
                ELSE
                    count <= 2;
                END IF;
            END IF;

            IF (count = 1 OR count = 9 OR count = 17 OR count = 25 OR count
= 33) THEN
                enable_dados <= '0';
            ELSE
                enable_dados <= '1';
            END IF;
        END IF;
    END IF;
END PROCESS;

PROCESS(clk)
BEGIN
    IF (rising_edge(clk)) THEN
        IF enable_quadro = '1' THEN
            IF enable_dados = '1' THEN
                data_word(0) <= d_in;
                data_word(6 downto 1) <= data_word(5 downto 0);
            END IF;
```

```
        END IF;
    END IF;
END PROCESS;


process(clk)
begin
if rising_edge(clk) then

for i in 0 to 2 loop
for j in 0 to 6 loop
temp_H(i, j) <= H(i, j) and code_in(j);
end loop;

F(i) <= temp_H(i, 0) xor temp_H(i, 1) xor temp_H(i, 2) xor temp_H(i, 3) xor
temp_H(i, 4) xor temp_H(i, 5) xor temp_H(i, 6);
end loop;

if F = "000" then
index_error <= -1;
else
index_error <= to_integer(unsigned(F)) - 1;
end if;

if index_error /= -1 then
temp <= code_in;
temp(index_error) <= not code_in(index_error);
error_detected <= '1';
else
temp <= code_in;
error_detected <= '0';
end if;

corrected_code <= temp;
data_out <= temp(6) & temp(5) & temp(4) & temp(2);

end if;
end process;
```

```
process(clk)
begin
if rising_edge(clk) then

if enable_dados = '0' then
shift_reg <= corrected_code;

elsif enable_dados = '1' then
serial_out <= shift_reg(6);
shift_reg <= shift_reg(5 downto 0) & '0';

end if;
end if;
end process;


END rtl;
```

Apêndice B.3.

```
--------------------------------------------------------------------------------
-- Company: FT-UNICAMP
-- Engineers: João Daniel R. Bertucci/Nyahn Ekyê Fernandes Duarte
--
-- Create Date: 21:01:08 11/21/2024
-- Module Name: TB_sys - Behavioral
-- Project Name: TCC
-- Description: Testbench
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_signed.all;
USE std.textio.all;
USE ieee.std_logic_textio.all;

ENTITY TB_sys IS
```

```
END TB_sys;

ARCHITECTURE behavior OF TB_sys IS


    COMPONENT TX_coded
        PORT(
            clk : IN std_logic;
            reset : IN std_logic;
            d_out: OUT std_logic;
            code_out: OUT std_logic_vector(6 downto 0)
        );
    END COMPONENT;

    COMPONENT RX_coded
        PORT(
            clk : IN std_logic;
            reset : IN std_logic;
            d_in: IN std_logic;
            code_in : IN std_logic_vector(6 downto 0);
            sincronismo: OUT std_logic
        );
    END COMPONENT;


    signal clk : std_logic := '0';
    signal data : std_logic;
    signal dados: std_logic_vector (6 downto 0);
    signal sinc: std_logic;
    signal reset: std_logic := '0';
    signal dados_channel : std_logic_vector(6 downto 0);
    CONSTANT clock_period : time := 20 ns;

BEGIN


    uut1: TX_coded PORT MAP (
        clk => clk,
        reset => reset,
        d_out => data,
```

```
        code_out => dados
    );


    uut2: RX_coded PORT MAP (
        clk => clk,
        reset => reset,
        d_in => data,
        code_in => dados_channel,
        sincronismo => sinc
    );


    clk <= NOT clk AFTER clock_period / 2;
    reset <= '1' AFTER 60 ns;


    PROCESS(clk)
    BEGIN
                dados_channel <= dados;
                dados_channel(6) <= NOT dados(6);

    END PROCESS;

END behavior;
```