

UNIVERSIDADE FEDERAL DE MINAS GERAIS

Escola de Engenharia

Curso de Graduação em Engenharia Elétrica

Gabriel Pimentel Gomes

**Projeto digital de um sistema de codificação *run length limited* para a
IEEE 802.15.7**

Belo Horizonte
2023

Gabriel Pimentel Gomes

**Projeto digital de um sistema de codificação *run length limited* para a
IEEE 802.15.7**

Trabalho de Conclusão de Curso apresentado ao
Curso de Engenharia Elétrica da Universidade Fe-
deral de Minas Gerais, como requisito parcial para
o grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Ricardo de Oliveira Duarte

Belo Horizonte
2023

Agradecimentos

Gostaria primeiramente de agradecer à minha família, que me apoiou não apenas durante a graduação mas durante toda a minha vida. Sem eles eu talvez nunca chegaria onde estou hoje. Gostaria de agradecer a todos os professores da UFMG que participaram da minha formação, especialmente ao professor Ricardo, meu orientador, que me apoiou nesses últimos anos tão críticos para a conclusão do curso. Por fim, agradeço também à toda comunidade da UFMG, que mantém funcionando uma universidade que é referência no país e forma diversos cidadãos como eu.

“Technically walking is just controlled falling.”
(Wilson Percival Higgsbury, *Don’t Starve*)

Resumo

As comunicações por Radiofrequência têm experimentado um crescimento vertiginoso no que diz respeito a número de dispositivos conectados, especialmente com o advento do Wi-Fi e dos *smartphones*. Essa tendência está levando a uma escassez do espectro disponível, fenômeno chamado de *Spectrum Crunch*. Tendo em vista esse problema, a comunicação por luz visível (Visible Light Communication, VLC) tem se tornado uma alternativa atraente para pesquisadores e empresas, que vêm trabalhado para desenvolver a tecnologia. Isso resultou em uma produção intensa de técnicas e dispositivos para VLC, juntamente com uma demanda por padronização. Uma norma importante criada para VLC foi a IEEE 802.15.7 que, entre outras coisas, apresenta especificações para a camada física (*Physical*, PHY).

Existem muitos trabalhos na literatura que propõem plataformas de desenvolvimento para VLC e seguem a norma IEEE 802.15.7. A camada física desses trabalhos, geralmente, implementa apenas uma parte dos modos de operação previstos na norma, oferecendo uma avaliação limitada das capacidades da VLC. Outro ponto é que boa parte desses trabalhos constrói a camada PHY sobre plataformas embarcadas, numa situação em que é preciso utilizar *hardware* dedicado para garantir velocidade de transmissão. Além disso, soluções de código aberto que envolvem circuitos dedicados não são muito comuns.

Dado o contexto, um codificador e decodificador *run length limited* como parte da camada PHY foi projetado neste trabalho. Ele fará parte de um sistema que implementa uma camada física com os modos de operação das PHY I, II e III, utilizando *hardware* dedicado e de código aberto. O texto apresenta o levantamento de requisitos feito para o projeto, detalha a arquitetura do bloco e faz uma análise de performance do mesmo.

Palavras-Chave: Digital Systems, Physical Layer, Run Length Limited Code, IEEE 802.15.7, Visible Light Communication.

Abstract

Radiofrequency communications have experienced a tremendous growth in terms of the number of connected devices, especially with the advent of Wi-Fi and smartphones. This trend is leading to a scarcity of available spectrum, a phenomenon known as Spectrum Crunch. In light of this problem, Visible Light Communication (VLC) has become an attractive alternative for researchers and companies, who have been working to develop the technology. This has resulted in an intense production of techniques and devices for VLC, along with a demand for standardization. An important standard created for VLC is IEEE 802.15.7, which, among other things, provides specifications for the Physical (PHY) layer.

There are many works in the literature proposing development platforms for VLC that follow the IEEE 802.15.7 standard. The physical layer of these works usually implements only a subset of the operating modes specified in the standard, offering a limited evaluation of VLC capabilities. Another point is that a significant portion of these works builds the PHY layer on embedded platforms instead of on dedicated hardware, which is necessary due to transmission speed issues. Furthermore, open-source solutions involving dedicated circuits are not very common.

Given this context, a run length limited encoder and decoder as part of the PHY layer was designed in this study. It will be part of a system implementing a physical layer with the PHY I, II, and III modes of operation, using dedicated and open-source hardware. The text presents the requirements survey conducted for the project, details the architecture of the block, and carries out a performance analysis of the same.

Keywords: Digital Systems, Physical Layer, Run Length Limited Code, IEEE 802.15.7, Visible Light Communication.

Lista de Figuras

1	Geração de um sinal <i>Manchester</i> utilizando uma porta lógica XOR. Não corresponde à versão proposta na norma IEEE 802.15.7. ([1])	13
2	Representação da codificação 8B10B dividida em 2 blocos. ([2])	15
3	Recorte de uma tabela de codificação 8B10B. ([2])	15
4	Lógica para tratamento de disparidade na codificação 8B10B. ([2])	16
5	Arquitetura do sistema VLC implementado em [3] (p. 38)	18
6	Arquitetura geral prevista na IEEE 802.15.7 (p. 8)	19
7	Arquitetura interna da camada PHY I mostrada na IEEE 802.15.7 (p. 243)	20
8	Interface do FEC ([4], p. 58)	20
9	Pacote de dados da camada física (PPDU) descrito no IEEE 802.15.7 (p. 224)	21
10	Exemplo de ligação utilizando a interface de dados genérica ([4], p. 57)	25
11	Exemplo de transferência de dados do protocolo AMBA AXI4- <i>Streaming</i> ([5] p. 19 ou “2-3”)	26
12	Interface do codificador e decodificador <i>Run Length Limited</i> desenvolvido.	27
13	Arquitetura simplificada do Cod/Decod RLL.	28
14	Arquitetura interna do decodificador RLL	30
15	Estrutura interna do decodificador específico <i>Manchester</i>	31
16	Arquitetura Simplificada utilizada no exemplo de funcionamento.	33
17	Codificador de exemplo recebendo a palavra de entrada juntamente com o modo de operação.	33
18	Codificador de exemplo preenchendo os registradores do cod. específico 4B6B.	33
19	Codificador de exemplo executando a codificação de 4 bits para 6 bits.	34
20	Codificador de exemplo transferindo os bits para a FIFO.	34
21	Codificador de exemplo conferindo o registrador de entrada ao término da primeira rodada de codificação.	34
22	Codificador de exemplo pronto para entregar a saída.	35
23	Exemplo de formas de onda obtidos durante os testes do Cod/Decod. Representa o codificador no modo 4B6B com 4 bits de entrada.	36
24	Diagrama mostrando o fluxo que transforma bits de dados em formas de onda na camada PHY.	38
25	Interface do Cod/Decod com indicação de conexão com blocos adjacentes	44
26	Arquitetura Interna do Cod/Decod	44
27	Arquitetura interna do codificador RLL	45
28	Diagrama de estados do codificador RLL	46
29	Diagrama de estados do decodificador RLL	47
30	Arquitetura interna dos codificadores específicos	48
31	Arquitetura interna dos decodificadores específicos	49

Lista de Tabelas

1	Codificação <i>Manchester</i> segundo a norma IEEE 802.15.7	13
2	Resumo das características da codificação <i>Manchester</i>	14
3	Codificação 4B6B segundo a norma IEEE 802.15.7.	14
4	Resumo das características da codificação 4B6B.	14
5	Resumo das características da codificação 8B10B.	16
6	Recorte dos sistemas VLC encontrados na literatura.	17
7	Recorte da tabela com as características arquiteturais do FEC ([4], p. 60)	21
8	Descrição da interface de dados genérica ([4], p. 56)	25
9	Comparação entre os sinais da interface AMBA AXI4-Streaming e da interface de dados genérica	26
10	Descrição das interfaces do Cod/Decod RLL desenvolvido.	28
11	Modos de operação do Cod/Decod RLL com as respectivas larguras de bits na porta de dados.	29
12	Tabela de decodificação <i>Manchester</i>	32
13	Entrada da tabela 19 usada no exemplo de vazão para modo de operação da PHY I	38
14	Latência e Vazão obtidas para o codificador RLL	38
15	Latência e Vazão obtidas para o decodificador RLL	39
16	Parâmetros de tamanho obtidos para Cod/Decod RLL	39
17	Performance do Codificador RLL	50
18	Performance do Decodificador RLL	51
19	Modos de operação da PHY I da IEEE 802.15.7 (p. 213)	53
20	Modos de operação da PHY II da IEEE 802.15.7 (p. 213)	53
21	Modos de operação da PHY III da IEEE 802.15.7 (p. 214)	53
22	Relação entre o MCS_ID e o modo de operação, na IEEE 802.15.7 (p. 227)	54

Siglas

AMBA Advanced Microcontroller Bus Architecture.

ASIC Application Specific Integrated Circuit.

CA Corrente Alternada.

CC Corrente Contínua.

Cod/Decod Codificador e Decodificador.

CSK Color Shift Keying.

FEC Foward Error Correction.

FIFO First In First Out.

FPGA Field Programmable Gate Array.

HDL Hardware Description Language.

IOT Internet of Things.

IP Intelectual Property.

Li-Fi Light Fidelity.

LoS Line of Sight.

LSB Least Significant Bit.

MAC Medium Access Control.

MCS_ID Modulation and Coding Scheme Identification.

PHY Physical.

PISO Parallel In Serial Out.

PPDU Physical Layer Data Unit.

PSDU Phy Service Data Unit.

RF Radiofrequênciा.

RLL Run Length Limited.

RTL Register Transfer Level.

SE Sistema Embarcado.

SIPO Serial In Parallel Out.

SoC System on Chip.

VLC Visible Light Communication.

Sumário

1	Introdução	11
1.1	Características da Comunicação por Luz Visível	11
1.2	Padrões Estabelecidos para VLC	11
1.2.1	Norma IEEE 802.15.7-2011	12
1.3	Códigos <i>Run Length Limited</i>	12
1.3.1	Codificação <i>Manchester</i>	13
1.3.2	Codificação 4B6B	14
1.3.3	Codificação 8B10B	15
1.4	Motivações e Objetivos do Trabalho	16
2	Revisão Bibliográfica	17
2.1	Visão Geral	17
2.2	Referências mais Relevantes	17
2.2.1	Silva (2020)	17
2.2.2	Hussain (2015)	18
2.2.3	Boyette (2006)	18
3	Análise de Requisitos	19
3.1	Interface	19
3.2	Modos de Operação	21
3.2.1	Estrutura do Pacote da Camada Física	21
3.2.2	Identificação dos Modos	22
3.2.3	Codificação e Decodificação	22
3.3	Atribuições no Sistema	22
3.4	Performance	23
4	Materiais e Métodos	24
4.1	Ferramenta de Desenhos DIA	24
4.2	Intel® Quartus Prime e IPs	24
4.3	Método RTL para Circuitos Digitais	24
4.3.1	Método de Projeto Baseado em Componentes	24
4.4	Interface de Dados Genérica	24
4.4.1	Protocolo AMBA® 4 AXI4- <i>Streaming</i>	25
5	Detalhamento do Codificador e Decodificador RLL	27
5.1	Interface e Comportamento	27
5.2	Arquitetura	28
5.2.1	Arquitetura do Decodificador	28
5.2.2	Codificadores/Decodificadores <i>Manchester</i> e 4B6B	31
5.2.3	Codificador/Decodificador <i>NOCODE</i>	32
5.2.4	Codificador/Decodificador 8B10B	32
5.3	Exemplo de funcionamento	32
6	Verificação e Performance	36
6.1	Verificação Funcional Por Meio de Simulação	36
6.2	Medidas de Performance	37
6.2.1	Latência	37
6.2.2	Vazão	37
6.2.3	Tamanho do circuito	38
7	Conclusão e Trabalhos Futuros	40
Apêndices		43
Apêndice A: Diagramas RTL com a arquitetura do Cod/Decod RLL	44	
Apêndice B: Resultados de performance do Cod/Decod RLL	50	

Anexos	52
Anexo A: Modos de Operação dos Diferentes Tipos de Camada PHY	53

1 Introdução

Comunicação por luz visível - *Visible Light Communication (VLC)* - ganhou muito espaço nos últimos anos devido à necessidade de um meio de comunicação capaz de complementar ou mesmo substituir aqueles por radiofrequência (**RF**). Isso ocorre num contexto em que estamos experimentando um aumento vertiginoso do número de dispositivos ligados à internet. *Smartphones, tablets* e todo tipo de aparelho que se enquadra na Internet das Coisas - *Internet of Things (IOT)* criaram uma demanda enorme por comunicação sem fio, geralmente Wi-Fi [6]. Esse fenômeno levou a uma sobrecarga do espectro de **RF**, chamada de *Spectrum Crunch*. O uso de dispositivos compatíveis com **VLC** pode ser uma alternativa para esse problema, além de oferecer outras vantagens.

1.1 Características da Comunicação por Luz Visível

A **VLC** apresenta várias características interessantes que chamaram a atenção de pesquisadores e engenheiros ao redor do mundo [4]:

- Não é necessário nenhum tipo de licença para utilizar as frequências da luz visível.
- Os LEDs utilizados para **VLC** já estão presentes em grande parte das residências e da infraestrutura das cidades, tornando a implementação mais rápida e mais barata [7].
- As maiores frequências do espectro visível oferece velocidade de comunicação até 3 ordens de magnitude maior, quando comparados a alguns dispositivos **RF**. [8].
- Enquanto **RF** atravessa paredes e outros obstáculos, a luz não. Além disso, é possível direcionar o feixe de luz com certa facilidade, fenômeno conhecido como Linha de Visão - *Line of Sight (LoS)*. Isso possibilita comunicações mais seguras.
- É possível utilizar **VLC** em lugares onde **RF** é banida, como em hospitais e plataformas de petróleo.
- Pode ser utilizada para complementar o Wi-Fi sem muitas dificuldades [9].

Apesar de possuir várias vantagens, o uso de **VLC** ainda apresenta algumas barreiras. Primeiramente, devido à maior frequência das ondas eletromagnéticas usadas, as perdas no meio são maiores e o alcance de comunicação fica bastante reduzido [10]. Depois, se por um lado o efeito de **LoS** torna a comunicação mais segura, ele dificulta comunicações em *broadcast*, pois a luz se espalha menos. Além disso, o meio de propagação está sujeito a muita interferência, seja de outras fontes luminosas ou do sol. Por isso, dispositivos que usam **VLC** precisam de técnicas de codificação e de correção de erros para mitigar esse problema.

1.2 Padrões Estabelecidos para VLC

Tendo em vista o crescente interesse em **VLC**, foram estabelecidos alguns padrões ao longo dos anos para uniformizar a abordagem da tecnologia. Além disso, novas normas estão sendo criadas para acompanhar as novas tendências. A primeira iniciativa de normatização foi em 2003 pelo Consórcio **VLC** no Japão, que resultou em dois padrões, o CP-1221 e o CP-1222 [11]. Em 2011 foi publicado o IEEE 802.15.7 [12]. Em 2014 foi lançado o IEEE 802.15.7m, expandindo o padrão antigo. Em 2018 foi publicada a versão mais recente do IEEE 802.15.7 [13], que acrescenta vários modos de operação mas mantém os já descritos na norma de 2011. Essa versão também expande o espectro eletromagnético disponível, prevendo o uso de frequências fora da faixa visível. Além disso, o padrão IEEE 802.11bb [14], que ainda não foi publicado em sua versão final, passou a prever uma cooperação entre o Wi-Fi e **Li-Fi** (*Light Fidelity*). **Li-Fi** nada mais é que o uso de comunicação óptica, não necessariamente visível, focado na conexão com a *Internet*.

Este trabalho tem como referência a norma IEEE 802.15.7 publicada em 2011.

1.2.1 Norma IEEE 802.15.7-2011

A IEEE 802.15.7 - *IEEE Standard for Local and metropolitan area networks— Part 15.7: Short-Range Wireless Optical Communication Using Visible Light* - se propõe a padronizar várias características da comunicação via luz visível (380 nm até 780 nm de comprimento de onda). Ela trata de algumas características básicas a nível de rede, padroniza diversas questões da camada de enlace/controle de acesso ao meio (**MAC**) e trata em detalhes a camada física (**PHY**).

A camada **MAC** (*Medium Access Control*) é responsável por arbitrar as transmissões: ela define qual aplicação terá seus dados enviados, quando enviá-los, endereçar os envios, entre outras coisas. Seguindo essas regras, ela monta um quadro e o entrega para a camada **PHY**. A camada **PHY** (*Physical*), por outro lado, trata do meio físico onde ocorre cada transmissão: decide quanto rápido os dados devem ser transmitidos selecionando a frequência de operação, decide qual modulação será aplicada, como fazer o tratamento dos dados para tornar a comunicação mais robusta, etc. É válido relembrar que, neste trabalho, foi feita uma parte da camada **PHY** e que não houve nada relativo à camada **MAC** implementado.

São propostos três tipos de camada física([12], p. 6, 9):

- **PHY I:** para uso em locais abertos, com taxas de transmissão baixas, para aplicações como iluminação pública e comunicação inter veicular.
- **PHY II:** para uso em locais fechados, com taxas de transmissão moderadas, para aplicações como comunicação com *smartphones* e com outros dispositivos móveis.
- **PHY III:** para uso em locais fechados, com taxas de transmissão elevadas, para aplicações como comunicação com *smartphones* e com outros dispositivos móveis.

Dispositivos podem implementar mais de um modo de operação e os três modos podem coexistir em um mesmo ambiente.

Além disso, ela também trata de proteção à saúde humana no quesito de luzes tremulantes (*flckering*) e de controle de brilho da fonte luminosa (*dimming*).

1.3 Códigos *Run Length Limited*

Um dos principais objetivos deste trabalho é implementar em *hardware* códigos *Run Length Limited* (**RLL**). Como o nome indica, esses códigos limitam a quantidade de 1's ou de 0's consecutivos que ocorrem durante a transmissão de dados numa linha serial. Apesar de esses códigos diminuírem a taxa efetiva de transmissão devido ao acréscimo de bits extras, eles tornam a comunicação mais robusta e também mais compatível com o hardware [15].

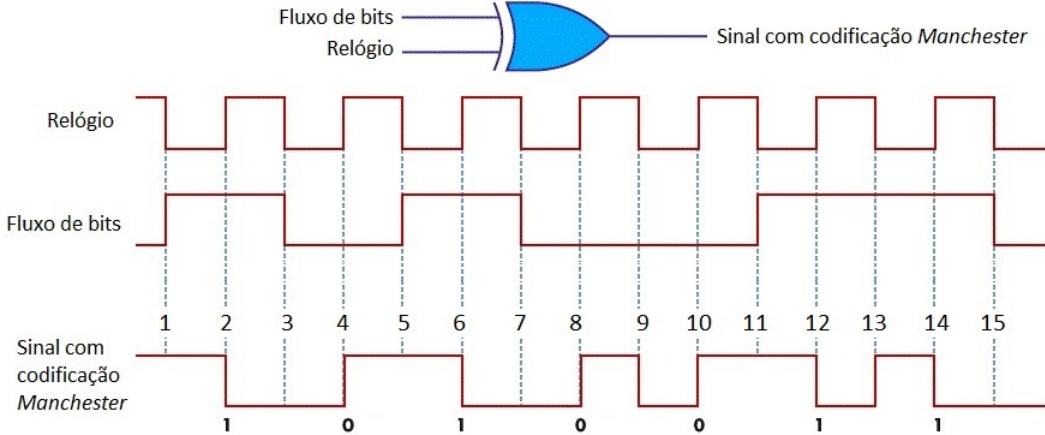
O primeiro ponto no qual essas codificações atuam é na recuperação do sinal relógio, uma vez que não há transmissão explícita desse sinal. O receptor e o transmissor possuem relógios separados e que devem estar sincronizados a todo momento. Quando há uma transição de nível lógico na transmissão, sabe-se que houve uma borda de relógio ali. Quando não há transição por um longo período, o receptor não consegue mais detectar quando ocorreu a borda de relógio e perde a sincronização. Por isso é importante manter um número mínimo de transições por unidade de tempo.

Também existe a possibilidade de o receptor ser incapaz de identificar sua entrada quando o sinal recebido fica muito tempo num mesmo nível lógico. Isso ocorre quando o receptor possui um capacitor de acoplamento de Corrente Alternada (**CA**) e é capaz de identificar apenas a parte alternada do sinal. Sinais que ficam muito tempo sem transições acabam, portanto, sendo bloqueados por esse capacitor. Esse fenômeno ocorre, por exemplo, em comunicadores **RF** simples como aqueles de controles de portões de garagem [16].

Além disso, é importante ressaltar que esses códigos devem ter equilíbrio de Corrente Contínua (**CC**), ou seja, o nível médio do sinal deve ser nulo. Em termos de bits, significa que o código deve ter o número de bits '1' igual ao número de bits '0' para uma dada transmissão.

Em termos de **VLC**, a codificação **RLL** possui várias funções. Facilitar o sincronismo e tratar a questão de acoplamento **CA** são algumas delas. O equilíbrio **CC**, por sua vez, é útil para tratar a questão de controle de brilho (*Dimming*) em transmissores que, além de enviar dados, também possuem o papel de iluminação. Um exemplo desse tipo de transmissor seriam os postes em vias públicas. Caso os códigos **RLL** possuissem um nível **CC**, em certas situações, transmiti-los poderia significar alterar o brilho da fonte luminosa, algo indesejado.

Figura 1: Geração de um sinal *Manchester* utilizando uma porta lógica XOR. Não corresponde à versão proposta na norma IEEE 802.15.7. ([1])



Dois parâmetros relevantes para se avaliar um código *RLL* é a sua eficiência de transmissão, e o número máximo de bits iguais seguidos. A eficiência é simplesmente a razão entre o número de bits de informação enviados e o número total de bits transmitidos. O número máximo de bits iguais seguidos é o pior caso a se esperar numa transmissão. Tais conceitos serão exemplificados nas seções a seguir.

1.3.1 Codificação *Manchester*

A codificação *Manchester* foi inicialmente concebida em 1949 na Universidade de *Manchester*, por G. E. Thomas. A motivação para sua criação foi o armazenamento de dados em fitas magnéticas presentes no computador *Mark I* [17]. Apesar de ter sido feita inicialmente para guardar dados, a codificação *Manchester* foi muito bem recebida na área de telecomunicações e é usada nesse campo até hoje por se tratar de uma codificação simples e robusta.

A codificação *Manchester* consiste em mapear bits em transições de nível lógico na linha. No tempo reservado para a transmissão de cada bit, ocorre uma borda de subida ou uma borda de descida no nível lógico da linha.

Na figura 1 temos uma ilustração da codificação *Manchester* em que bits 1 são formados com bordas de descida e bits 0 com bordas de subida. É importante notar que apenas transições no meio do tempo do bit são consideradas como de informação e que transições junto à mudança de bit são desconsideradas. Na figura, apenas as transições com marcação de número par são consideradas como de informação. Esse tipo de sinal pode ser gerado, por exemplo, com uma operação lógica XOR entre os dados e o sinal de relógio que marca a cadência dos bits.

A codificação *Manchester* não é única, existindo diversas variações que invertem o significado das bordas ou que fazem um tratamento diferencial dos dados. Essas variações não são foco deste trabalho.

A versão utilizada na norma IEEE 802.15.7 (p. 249) aparece na tabela 1. Além de usar uma versão específica, a norma prevê que o sinal *Manchester* não é enviado diretamente no canal, mas sim transformado em símbolos que mais tarde serão enviados. Esses símbolos têm 2 bits e o bit menos significativo (*Least Significant bit - LSB*) é enviado primeiro. Dependendo de como os dados forem tratados, pode-se considerar que o bit 1 corresponde a uma borda de subida e o bit 0 a uma de descida.

Tabela 1: Codificação *Manchester* segundo a norma IEEE 802.15.7

bit	Símbolo Manchester
0	01
1	10

Na tabela 2 é apresentado um resumo das características da codificação *Manchester* utilizada neste trabalho.

Tabela 2: Resumo das características da codificação *Manchester*.

Codificação Manchester	
Equilíbrio CC	Sim
Número máximo de bits iguais seguidos	2
Eficiência	50%
Complexidade	Baixa

Na tabela 1 podemos ver que cada símbolo *Manchester* já possui equilíbrio CC, garantido esse equilíbrio na transmissão como um todo. Também é possível notar que cada bit de informação gera um símbolo de dois bits. Isso significa que a eficiência dessa codificação é de 1/2 ou 50%. Na tabela 2 temos um resumo dessas informações.

1.3.2 Codificação 4B6B

A codificação 4B6B simplesmente propõe um mapeamento de palavras de 4 bits para símbolos de 6 bits. Esses símbolos contém exatamente três 1's e três 0's, além de limitarem o tempo máximo num nível lógico. É um tipo de codificação pouco usual, mas foi encontrado um exemplo semelhante num sistema RF [16]. Sua definição aparece na tabela 3, presente na norma IEEE 802.15.7 (p. 248).

Tabela 3: Codificação 4B6B segundo a norma IEEE 802.15.7.

4B (Entrada)	6B (Saída)
0000	001110
0001	001101
0010	010011
0011	010110
0100	010101
0101	100011
0110	100110
0111	100101
1000	011001
1001	011010
1010	011100
1011	110001
1100	110010
1101	101001
1110	101010
1111	101100

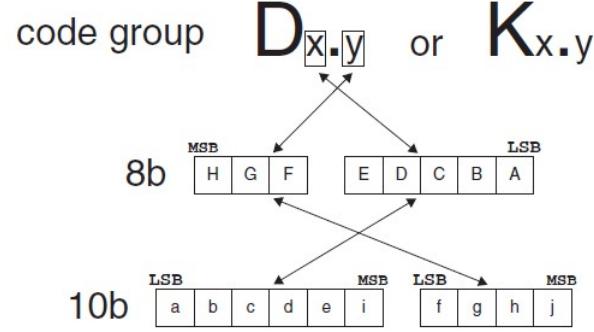
Na norma são listadas algumas características da codificação 4B6B, como o equilíbrio CC presente em cada símbolo codificado, capacidade de detectar erros, número máximo de bits iguais seguidos de 4 e resincronização de relógio razoável.

Sua eficiência é de 4/6 ou de aproximadamente 66% e um resumo das características da codificação 4B6B pode ser visto na tabela 4.

Tabela 4: Resumo das características da codificação 4B6B.

Codificação 4B6B	
Equilíbrio CC	Sim
Número máximo de bits iguais seguidos	4
Eficiência	66%
Complexidade	Média

Figura 2: Representação da codificação 8B10B dividida em 2 blocos. ([2])



1.3.3 Codificação 8B10B

A codificação 8B10B foi inicialmente proposta pelos engenheiros Widmer e Franaszek, da empresa IBM, em 1983 [18] e rendeu uma patente no ano seguinte [19]. Ela tem sido muito útil em sistemas de comunicações e foi utilizada, por exemplo, no padrão *Ethernet* IEEE 802.3 [20]. Depois que a patente expirou, na década de 2000, ela passou a ser usada diretamente por diversos fabricantes.

Segundo seus criadores, a codificação 8B10B seria implementada a partir de duas tabelas, uma que faz uma transformação de 3 bits para 4 bits e outra que faz uma transformação de 5 bits para 6 bits. Além disso, existia um sinal de controle K que alteraria a codificação normal para símbolos especiais, dependendo da entrada. Veja uma representação na figura 2.

Apesar de o código apresentar equilíbrio CC, nem todos os símbolos são equilibrados por si só. Existem símbolos com seis bits 1, com cinco bits 1 e com quatro bits 1. O primeiro desses três tipos é considerado com disparidade RD-, ou seja, tem bits 1 sobrando. O último, naturalmente é considerado com disparidade RD+, com bits 1 faltando. O do meio é considerado com disparidade neutra (RD+ ou RD-), pois o número de bits 1 e de bits 0 é igual a cinco. Veja alguns exemplos de codificações na figura 3.

Figura 3: Recorte de uma tabela de codificação 8B10B. ([2])

Code Group	kin/ kout	8-bit data		10-bit data (RD-) abcdei fghj		10-bit data (RD+) abcdei fghj	
		HGF	EDCBA	abcdei	fghj	abcdei	fghj
D0.0	0	000	00000	100111	0100	011000	1011
D1.0	0	000	00001	011101	0100	100010	1011
D2.0	0	000	00010	101101	0100	010010	1011
D3.0	0	000	00011	110001	1011	110001	0100
:							
D31.0	0	000	11111	101011	0100	010100	1011

Para manter o equilíbrio CC da linha, o codificador deve ser capaz de ficar alternando entre símbolos RD+ e RD-, de forma a manter o número total de bits 1 na saída igual ao número de bits 0. Uma possível lógica baseada em máquina de estados para tratar esse problema pode ser visto na figura 4.

Considerando todas as combinações possíveis de símbolos de 10 bits, o máximo de bits iguais seguidos é de 5.

Na norma IEEE 802.15.7 (p. 251), a codificação 8B10B utilizada é a definida em outra norma, a ANSI/INSCITS 373. Como essa última norma precisaria ser comprada e adquiri-la não era possível, optou-se por utilizar a definição original da codificação e seguir a patente de 1984. Tal escolha não implicou em nenhum impacto relevante no projeto. Algo a ressaltar é que todo o controle do meio é feito com os cabeçalhos dos quadros trocados entre dispositivos, como será mostrado em seções futuras. Isso significa que as palavras de controle e o bit K não serão usados.

Veja um resumo da codificação 8B10B na tabela 5.

Figura 4: Lógica para tratamento de disparidade na codificação 8B10B. ([2])

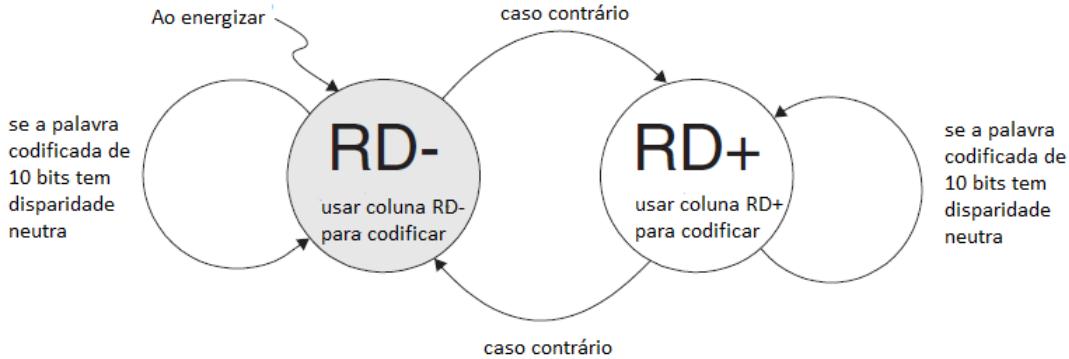


Tabela 5: Resumo das características da codificação 8B10B.

Codificação 8B10B	
Equilíbrio CC	Sim
Número máximo de bits iguais seguidos	5
Eficiência	80%
Complexidade	Alta

1.4 Motivações e Objetivos do Trabalho

O provável *Spectrum Crunch* da **VLC** colocou a **VLC** em pauta como alternativa. Tanto *startups* quanto empresas já consolidadas começaram a investir na tecnologia [21], tendo em vista uma expectativa de grande crescimento do mercado na área [7]. A maioria das plataformas de desenvolvimento encontradas na literatura são implementadas em *software* via sistemas embarcados, atuando mais como protótipos do que dispositivos capazes de atender aos requisitos da IEEE 802.15.7. Para cumprir as exigências da norma, principalmente as de performance, é imperativo um *hardware* dedicado, mas há poucos trabalhos nesse sentido. Também há uma escassez de literatura abordando sistemas **VLC** como um todo, levando em conta questões como as energéticas e de orçamento [22]. Além disso, é pouco comum uma plataforma de livre acesso (*Open Source*) para desenvolvimento de dispositivos com **VLC**, dificultando o avanço de pesquisas [23].

Este trabalho de conclusão de curso visa entregar uma propriedade intelectual (*Intellectual Property - IP*) de livre acesso capaz de implementar as codificações **RLL** especificados nas camadas **PHY I, II e III** da norma IEEE 802.15.7. O projeto foi feito utilizando uma linguagem de descrição de hardware e teve seus arquivos disponibilizados. Eles são acompanhados de uma documentação detalhada e de instruções de uso.

A norma IEEE 802.15.7 foi escolhida como ponto de partida para este trabalho pois ele se trata da continuação de um trabalho anterior [4], que adotou essa norma por diversos motivos. Tudo que foi desenvolvido aqui é compatível com os módulos criados no outro trabalho. Este esforço acrescentará mais uma parte à camada **PHY** em desenvolvimento e a deixará mais próxima do seu objetivo final de montar uma plataforma de testes **VLC** mais completa.

Espera-se que com este trabalho seja dado um passo à frente na pesquisa e desenvolvimento de dispositivos compatíveis com **VLC**. Um grande problema enfrentado nessa área é que existem poucas opções de hardware dedicado, poucos trabalhos que levam em conta o sistema como um todo e menos ainda plataformas de livre acesso. O dispositivo desenvolvido neste trabalho e no anterior já representam grande parte de uma camada **PHY** completa, utilizando *hardware* dedicado e de livre acesso, possivelmente capaz de mitigar tais problemas e ajudar a comunidade científica.

2 Revisão Bibliográfica

Uma vez estabelecido o contexto e os objetivos deste trabalho, devem-se apresentar os trabalhos relacionados ao assunto e analisar o conteúdo reunido como um todo. De maneira geral, foram encontradas implementações da camada **PHY** de forma parcial, utilizando diferentes tecnologias, de código aberto ou fechado. Nem todas seguiram a norma IEEE 802.15.7. Além disso, foram encontrados trabalhos que tratam especificamente dos codificadores implementados. Tudo isso será discutido a seguir.

2.1 Visão Geral

Existem muitos trabalhos implementando uma camada física para comunicação **VLC**. Alguns não seguem a norma IEEE 802.15.7, seja por praticidade ou pela necessidade de se usar um método diferente. Apesar disso, a camada física implementada por eles ainda tem uma arquitetura muito semelhante à proposta pela norma [24, 25].

Dentre os sistemas que seguem a norma, a maioria deles implementa somente a camada **PHY I**, e ainda com capacidade de operar em apenas um ou em poucos dos modos de operação [23, 26]. Existe um trabalho que implementou parcialmente as camadas **PHY I, II e III**. [4]. Não foi encontrado nenhum projeto em que os três tipos de camada física fossem implementados de maneira completa. Essa incompletude pode ter ocorrido pela limitação técnica de se trabalhar em tantas frentes ao mesmo tempo. Um caso a ser pontuado como exemplo é uma pesquisa em que a camada **PHY III** não foi implementada devido ao escopo limitado do projeto [27].

Com relação às tecnologias utilizadas nos trabalhos observados, houve implementações em sistemas embarcados (**SE**) [23, 27, 28], **FPGA**, **ASIC** e **SoC** [7, 3, 24]. A implementação predominante, principalmente nas plataformas *open source*, foi a de **SEs** por ser mais fácil de se utilizar em prototipação, apesar de limitar a performance do sistema.

A tabela 6 lista algumas das fontes encontradas em um quadro comparativo.

Tabela 6: Recorte dos sistemas **VLC** encontrados na literatura.

Autores e Data	Open Source	Tecnologia	Usa a IEEE 802.15.7	Implementou
Setiawan et al. 2019 [24]	Não	SoC	Não	-
Hussain 2015 [3]	Não	FPGA/ASIC	Sim	PHY I
Pradana et al. 2015 [22]	Não	SE	Não	-
Che et al. 2014 [26]	Não	FPGA/ASIC	Sim	PHY I
Gavrincea et al. 2014 [27]	Não	FPGA + PC	Sim	PHY I, II
Silva 2020 [4]	Sim	FPGA	Sim	PHY I, II, III (parcial)
Yin et al. 2018 [29]	Sim	SE	Sim	PHY I
Schmid et al. 2016 [30]	Sim	SE	Não	-
Hewage et al. 2016 [31]	Sim	SE ou FPGA	Não	-
Wang et al. 2015 [23]	Sim	SE	Sim	PHY I

2.2 Referências mais Relevantes

2.2.1 Silva (2020)

Essa dissertação de mestrado [4] é o texto mais importante dentre as referências aqui citadas, pelo fato de o trabalho aqui desenvolvido ser uma continuação desse. O bloco **RLL** se comunicará diretamente com o bloco de *Forward Error Correction* (**FEC**) dele. Isso significa que os protocolos propostos, as interfaces utilizadas e os índices de performance seguidos do **FEC** devem estar de acordo com as características do codificador e decodificador (**Cod/Decod**) **RLL**. A decisão de seguir a norma IEEE 802.15.7 de 2011, por exemplo, foi feita na dissertação e por isso adotada neste trabalho.

O **FEC** implementa boa parte da camada **PHY**, incluindo o codificador *Reed-Solomon*, o *interleaver*, o codificador convolucional e o bloco de *puncture*, que podem ser vistos na arquitetura em 7. A parte que faltou para uma camada **PHY** completa foram a codificação **RLL**. No entanto, ainda é preciso um modulador gerenciando os LEDs para montar um dispositivo funcional.

O IP desenvolvido foi cuidadosamente verificado utilizando diferentes *softwares*. Além disso, o FEC foi implementado tanto em **FPGA** quanto em **ASIC** e vários aspectos de performance foram analisados, como tamanho do circuito, frequência máxima de operação e potência dissipada.

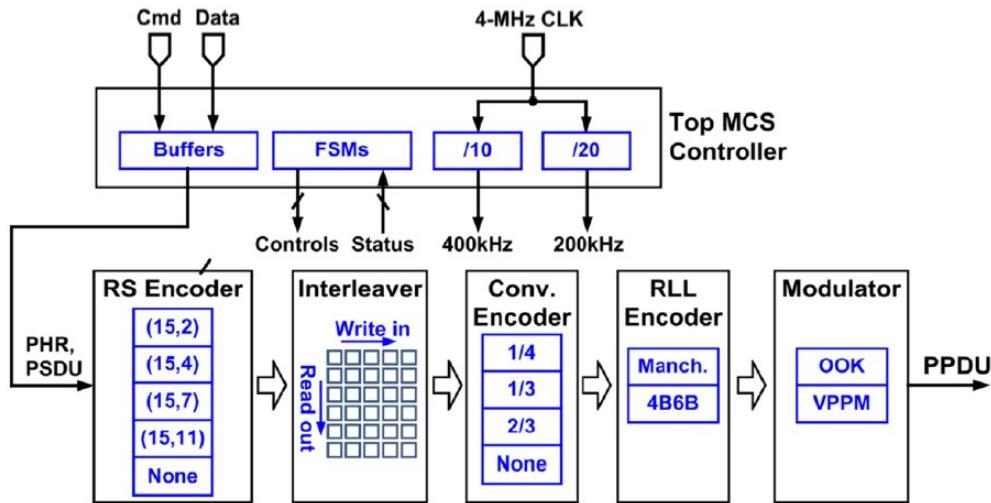
Os pontos relevantes de implementação do FEC não serão discutidos nesta seção, pois eles serão tratados em detalhes na seção 3.

2.2.2 Hussain (2015)

Essa dissertação de mestrado faz uma análise completa de um sistema de comunicação VLC utilizando um **ASIC** como transmissor e um **FPGA** como receptor. Ele também faz uma análise de orçamento do enlace de dados (*link budget*). Mas o que torna esse trabalho importante é a explicação detalhada da arquitetura, que foi utilizada como referência para parte do codificador RLL.

O texto explica como os codificadores e decodificadores RLL foram implementados, propondo comportamentos para casos não explicitados na norma (e. g. tratamento de dados inválidos chegando ao receptor). Como apenas a **PHY I** foi implementada, as codificações Manchester e 4B6B foram detalhadas enquanto a 8B10B não. Veja um diagrama na figura 5, que deixa isso claro.

Figura 5: Arquitetura do sistema VLC implementado em [3] (p. 38)



Além disso, é feita uma explicação sobre o *Top MCS Controller* do diagrama, responsável por configurar o caminho de dados. Esse tipo componente será necessário em nossa arquitetura mas ainda não foi implementado e, portanto, o desse trabalho pode servir de exemplo.

2.2.3 Boyette (2006)

Dos três tipos de codificação, a 8B10B é a mais complexa de se implementar. Por isso, foi utilizado um bloco *open source* disponibilizado no site da *OpenCores* [32]. Publicado depois que a patente expirou, esse bloco é praticamente uma transcrição em VHDL do circuito original [19].

Além de fornecer o código fonte do codificador e do decodificador 8B10B, são disponibilizados *testbenches* para experimentar com os circuitos e também são citadas algumas referências para se estudar a codificação.

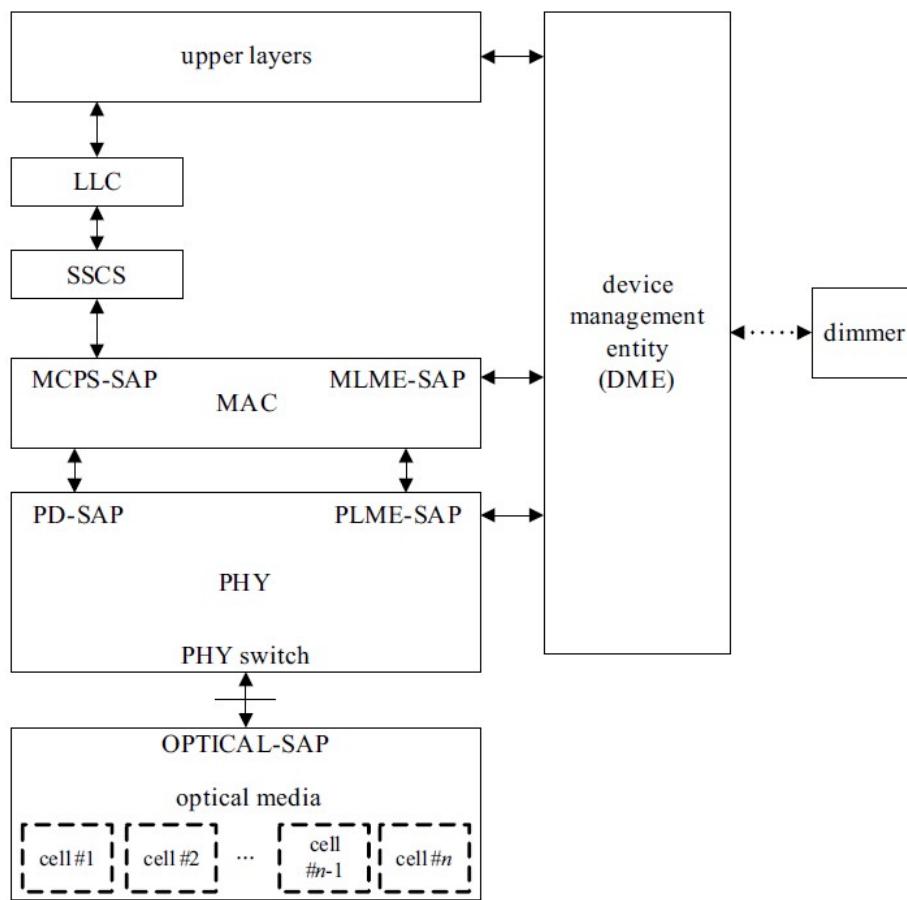
3 Análise de Requisitos

Esta seção tem como foco detalhar as decisões de projeto tomadas neste trabalho, explicitando as referências utilizadas como base. A maior parte das decisões foi tomada a partir da IEEE 802.15.7 e da dissertação detalhando o [FEC](#). Outras decisões, relacionadas à implementação de partes específicas da arquitetura, foram tomadas a partir de textos na literatura implementando circuitos semelhantes ou de blocos *open source* que foram incluídos.

3.1 Interface

A norma se propõe a especificar a camada física e a camada [MAC](#) do sistema de comunicação, presentes em dispositivos de rede tradicionais. Essas são as duas camadas explicadas em detalhes no documento. A descrição das demais camadas presentes na figura 6 está fora do escopo da norma.

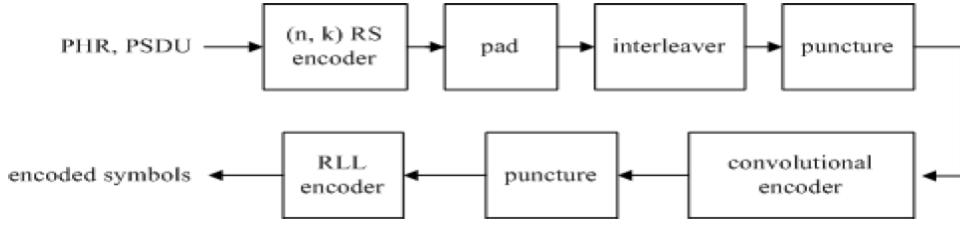
Figura 6: Arquitetura geral prevista na IEEE 802.15.7 (p. 8)



Como se pode ver na figura 6, a camada física faz interface com três outras partes da arquitetura. São elas: o dispositivo óptico (OPTICAL-SAP), a camada [MAC](#) e o DME. O bloco [RLL](#) deste trabalho participa somente da comunicação com a parte óptica. Essa observação, no entanto, não é suficiente para caracterizar o [Cod/Decod](#). É preciso ver partes mais internas da arquitetura para ter uma noção mais clara da localização do bloco.

Na figura 7 pode-se ver a arquitetura interna da camada [PHY I](#). Ela recebe os dados vindo da camada [MAC](#) (PSDU, explicado nas seções seguintes) e faz com que os dados passem por várias etapas antes de os *encoded symbols* serem entregues ao dispositivo óptico. O bloco [RLL](#) é o último antes de enviar os dados para o meio óptico e o bloco que vem logo antes dele é o de *puncture*. A arquitetura da camada [PHY II](#) é muito semelhante à da I, e não precisa ser explicada. A camada [PHY III](#) não possui [Cod/Decod RLL](#).

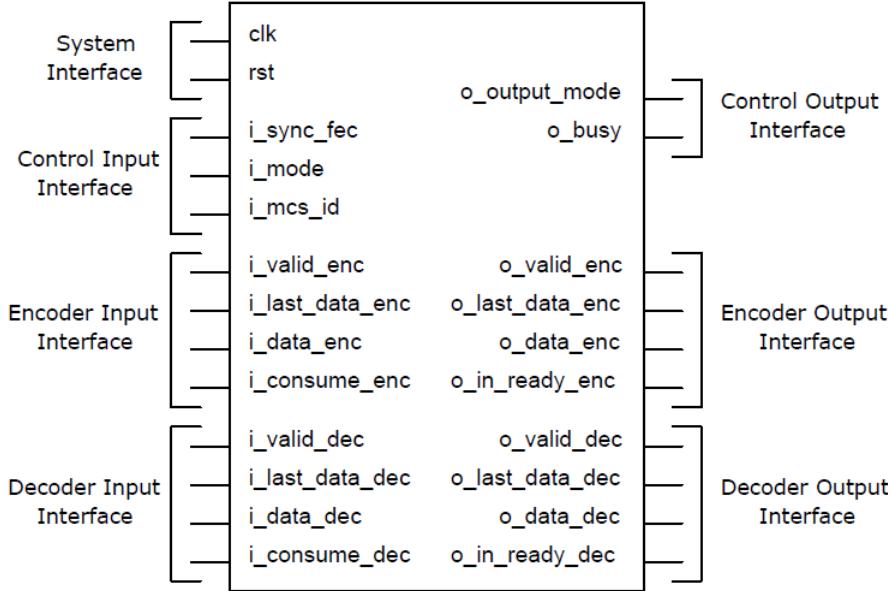
Figura 7: Arquitetura interna da camada PHY I mostrada na IEEE 802.15.7 (p. 243)



Levando em conta a localização do bloco, é possível ver que ele faz interface com o **FEC** [4] e com o dispositivo óptico. É importante notar que também deve haver alguma comunicação entre o bloco e a controladora do sistema. Com essas informações, já é possível definir a interface do codificador **RLL**.

A interface do **FEC** já está bem definida, e suas especificações são claras. Os sinais de interface aparecem na figura 8 e são descritos em detalhes na dissertação. Resumidamente, as portas agrupadas como *Encoder Input Interface*, *Encoder Output Interface*, *Decoder Input Interface* e *Decoder Output Interface* seguem a interface de dados genérica descrita na seção 4.4. As portas agrupadas como *Control Input Interface* e *Control Output Interface* comunicam-se com a controladora do sistema. O **Cod/Decod** irá se comunicar com o **FEC** por meio das portas presentes em *Encoder Output Interface* e *Decoder Input Interface*.

Figura 8: Interface do **FEC** ([4], p. 58)



Um ponto importante de se destacar é que as portas de dados *i_data_dec/o_data_enc* possuem largura de 8 bits, mas nem todos os eles são utilizados, dependendo da configuração. A tabela 7 foi retirada da especificação do **FEC** e mostra como o número de bits válidos nas portas varia.

A interface com o modulador e demodulador está em aberto, uma vez que ainda serão implementadas. No entanto, algumas características dessa interface podem ser assumidas com certa segurança. Primeiramente, ela seguirá a interface genérica utilizada na dissertação detalhando o **FEC**. Outro ponto importante é que a comunicação feita deve ser serial, ou seja, o sinal de dados deve ter 1 bit de largura. Por fim, de acordo com a IEEE 802.15.7 (p. 224), o **LSB** de cada byte, começando do byte menos significativo do quadro, deve ser enviado primeiro no meio óptico. Optou-se por enviar nesse mesmo formato para o modulador e assumiu-se, também, que o demodulador entregará os dados para o bloco **RLL** nesse formato.

Tabela 7: Recorte da tabela com as características arquiteturais do **FEC** ([4], p. 60)

MCS ID	Data rate	i.data_enc (width)	o.data_enc (width)	i.data_dec (width)	o.data_dec (width)
PHY I					
000000	11.67 kb/s	4	4	1	4
000001	24.22 kb/s	4	3	1	4
000010	48.89 kb/s	4	3	1	4
000011	73.3 kWb/s	4	4	4	4
000100	100 kb/s	8	8	8	8
000101	35.56 kb/s	4	4	4	4
000110	71.11 kb/s	4	4	4	4
000111	124.4 kb/s	4	4	4	4
001000	266.6 kb/s	8	8	8	8
PHY II					
010000	1.25 Mb/s	8	8	8	8
010001	2 Mb/s	8	8	8	8
010010	2.5 Mb/s	8	8	8	8
010011	4 Mb/s	8	8	8	8
010100	5 Mb/s	8	8	8	8
010101	6 Mb/s	8	8	8	8
010110	9.6 Mb/s	8	8	8	8
010111	12 Mb/s	8	8	8	8
011000	19.2 Mb/s	8	8	8	8
011001	24 Mb/s	8	8	8	8
011010	38.4 Mb/s	8	8	8	8
011011	48 Mb/s	8	8	8	8
011100	76.8 Mb/s	8	8	8	8
011101	96 Mb/s	8	8	8	8
PHY III					
100000	12 Mb/s	8	8	8	8
100001	18 Mb/s	8	8	8	8
100010	24 Mb/s	8	8	8	8
100011	36 Mb/s	8	8	8	8
100100	48 Mb/s	8	8	8	8
100101	72 Mb/s	8	8	8	8
100110	96 Mb/s	8	8	8	8

A comunicação com a controladora do sistema também está em aberto, mas optou-se por fazer uma interface semelhante à do **FEC**, presente nos sinais em *Control Input/Output Interface* da figura 8.

3.2 Modos de Operação

3.2.1 Estrutura do Pacote da Camada Física

Para se entender os modos de operação necessários no codificador **RLL**, deve-se observar a estrutura do pacote da camada física, ou *physical layer data unit* (**PPDU**) na figura 9.

Figura 9: Pacote de dados da camada física (PPDU) descrito no IEEE 802.15.7 (p. 224)

Preamble	PHY header	HCS	Optional fields	PSDU
SHR	PHR			PHY payload

No **PPDU**, a ordem que os campos são enviados é da esquerda para a direita. Os cinco campos são o preâmbulo, o *phy header*, o HCS, os *optional fields* e o **PSDU**. O primeiro é uma sequência predominantemente do tipo “01010101” utilizada para sincronização entre os comunicadores, e é enviada sem nenhum tipo de codificação. O segundo é um cabeçalho com diversas informações sobre o **PPDU**. Um detalhe importante sobre esse cabeçalho é que ele deve ser enviado na menor taxa de transmissão, dada uma frequência de relógio óptico, de acordo com os modos de operação no apêndice. O terceiro é um código usado para verificação de integridade do cabeçalho. O quarto são itens opcionais, como

número bits 0 adicionados ao fim do pacote que às vezes são necessários. O quinto e último são os dados propriamente ditos vindos da camada **MAC**.

Esse formato de pacote tem algumas implicações no **Cod/Decod** a ser desenvolvido. A primeira delas é que a camada física precisará ter uma configuração que não codifica os dados para que o preâmbulo seja enviado. Outro ponto é que, durante a transmissão do *phy header*, o modo de operação pode ser diferente daquele a ser usado no restante do quadro. A camada física deve estar preparada para essa mudança.

3.2.2 Identificação dos Modos

Uma vez entendido como os modos de operação devem aparecer numa transmissão, é possível detalhar cada um deles e ver os seus requisitos. Os modos de operação são descritos em diversas tabelas presentes na norma, que foram transcritas no **apêndice**. Unindo as informações de todas essas tabelas, é possível fazer um mapa de todos os modos de operação semelhante ao encontrado na tabela 7, relacionando cada **MCS.ID** a uma configuração do bloco **RLL**. Além disso, também é possível identificar algumas condições de velocidade de operação em cada configuração.

Na camada **PHY I**, devem ser implementadas as codificações *Manchester* e 4B6B. A taxa de transmissão de dados é da ordem de dezenas a poucas centenas de kbps, indicando que a velocidade de operação do não será um problema, pois as frequências de relógio em **FPGAs** são da ordem de MHz.

Na camada **PHY II**, devem ser implementadas as codificações 4B6B e a 8B10B. A taxa de transmissão é da ordem de unidades até várias dezenas de Mbps, indicando que será necessário tomar certo cuidado no projeto para que ele atenda às restrições de velocidade.

Na camada **PHY III** deve ser implementado um caminho sem codificação alguma, e as restrições de velocidade também se aplicam.

O bloco **RLL** será capaz de identificar qual codificação deve ser aplicada a partir de um **MCS.ID** fornecido pela controladora do sistema.

3.2.3 Codificação e Decodificação

Uma vez de posse das especificações de cada modo de operação, é necessário decidir como implementar cada um deles. Em resumo, o **Cod/Decod** deve implementar três tipos de codificação (*Manchester*, 4B6B e 8B10B), além de possuir um caminho sem codificação alguma.

As codificações *Manchester* e 4B6B são diretas e podem ser feitas simplesmente como uma transcrição das tabelas 1 e 3. As decodificações, por outro lado, não foram explicadas em detalhes pela IEEE 802.15.7, fazendo-se necessária uma pesquisa por soluções. O principal problema era que não foi deixado claro o que deveria ser feito caso palavras inválidas entrassem no decodificador. Felizmente, o sistema **VLC** detalhado em [3] apresentou uma proposta de decodificação bem clara, e foi adaptada para este trabalho.

A codificação 8B10B, como já explicado, foi feita a partir de um bloco de código aberto [32]. A codificação 8B10B, portanto, deveria respeitar a arquitetura desse bloco. A principal questão a ser solucionada era o monitoramento de disparidade (equilíbrio entre bits 1 e 0) do codificador 8B10B, que exigia um controle de fluxo no componente. A solução proposta aparece na seção 5.2.4.

3.3 Atribuições no Sistema

O **Cod/Decod RLL** fará parte de um sistema com a camada **PHY** completa e, por isso, precisará interagir com o **FEC**, com o modulador e com a controladora do sistema. Como foi apresentado anteriormente, a camada **PHY** deverá atender ao requisito de mudar o modo de operação dentro do mesmo pacote, ao fim do *phy header*.

Para se planejar a interface, foi preciso assumir um certo comportamento da controladora, pois o **Cod/Decod** irá se configurar de acordo com os sinais enviados por ela. Veja o comportamento esperado na lista a seguir.

- A primeira palavra de cada bloco de dados recebido deve vir acompanhada de seu **MCS.ID** para que a configuração correta seja feita.
- Quando for necessário mudar a configuração, a controladora envia um dado marcado como último (*i_last_data_** na interface genérica). E recomeça com um novo **MCS.ID**.

Essas definições são necessárias para que o comportamento do bloco **RLL** com relação ao sinal **MCS_ID** fique claro.

3.4 Performance

A fim de saber se o **Cod/Decod** será adequado para as aplicações, é necessário definir parâmetros de performance e atribuir valores mínimos para estes parâmetros. Os parâmetros entendidos como relevantes foram a latência, a vazão *throughput* e o tamanho do circuito.

A latência geralmente é calculada como o tempo discorrido entre o dispositivo receber um dado e enviar a resposta relativa a esse mesmo dado. Na norma IEEE 802.15.7 (p. 27) existe uma restrição de tempo entre quadros que se relaciona diretamente com a latência. Outro ponto importante a ser considerado é que, quando os componentes são ligados em série, um entregando dados para o outro, a latência desses componentes se soma. Como se vê na figura 7, a camada **PHY** é composta por vários blocos menores, ligados serialmente. Por isso, a latência do bloco **RLL** representa apenas uma parte da latência total. Estimou-se que a latência do **Cod/Decod** seria cerca de dez vezes menor que a do **FEC**.

A vazão de dados é outro parâmetro muito importante. Ela é definida como a quantidade de bits que são transmitidos dentro de um intervalo de tempo. Esse parâmetro define se a camada **PHY** implementada será ou não capaz de fornecer a taxa de dados necessária. A vazão máxima do sistema é limitada pela vazão do bloco mais lento. Para o desenvolvimento do **Cod/Decod**, esse quesito deve ser avaliado com atenção.

O tamanho do circuito é outro parâmetro muito relevante que se relaciona com a performance e com o dispositivo a ser escolhida. Caso o circuito seja grande demais para um determinado **FPGA**, por exemplo, será necessário trocar o dispositivo por outro maior. Caso o circuito ocupe quase todo o espaço disponível no chip, a performance é afetada. Comparativamente, espera-se que o **Cod/Decod RLL** seja pequeno dentro do sistema.

4 Materiais e Métodos

Nesta seção serão apresentadas as diferentes ferramentas utilizadas para a realização deste trabalho. O entendimento de cada uma foi fundamental para a conclusão do projeto, e ajuda a entender algumas decisões arquiteturais.

4.1 Ferramenta de Desenhos DIA

Para se fazer um projeto digital é muito importante visualizá-lo durante a sua concepção e mais ainda em sua construção e depuração. Desenhar o circuito em diagramas de blocos e com símbolos especiais é muito útil para se ter esse tipo de visualização.

Para realizar tal tarefa foi utilizada a ferramenta de desenhos DIA [33], que possui um pacote especializado para desenhar circuitos digitais. A partir dos desenhos feitos, a descrição em *hardware description language HDL* foi muito mais rápida, com menos erros e de mais fácil depuração.

4.2 Intel® Quartus Prime e IPs

O Intel® Quartus Prime é uma ferramenta completa de desenvolvimento de projetos em **FPGA**, com o *software* de simulação ModelSim incluído. O sintetizador do Quartus e o ModelSim foram as ferramentas utilizadas para verificar o funcionamento do **Cod/Decod**.

Algo importante de se destacar é que foram utilizados alguns **IPs** para o desenvolvimento do trabalho, que são blocos prontos descritos em **HDL** disponibilizados no *Quartus*. Foram utilizados o registrador de deslocamento *lpm_shifreg* [34] e a **FIFO scfifo** [35]. Esses blocos são de código fechado e desenvolvidos especificamente para **FPGAs** da Intel®, podendo representar um problema numa possível implementação em dispositivos de outros fabricantes. No entanto, como são blocos muito simples, eles podem ser facilmente substituídos por uma versão de outro fabricante ou mesmo por uma versão genérica.

4.3 Método RTL para Circuitos Digitais

Para se fazer este trabalho, foram seguidos cuidadosamente os princípios de projeto *Register Transfer Lever RTL* presentes em [36]. Além da proposta do livro, foi preciso criar níveis hierárquicos onde cada componente dentro do bloco **RLL** possuía interface e comportamento bem definidos. Essa definição era feita antes de esses componentes serem projetados de fato, numa abordagem *top - down*.

4.3.1 Método de Projeto Baseado em Componentes

Num projeto **RTL**, existe como decisão arquitetural escolher o tamanho dos componentes que farão parte do sistema. É possível utilizar vários componentes, cada um com pouca lógica, ou ir ao extremo oposto e construir todo o sistema do ponto de vista de um componente só. Componentes grandes demais geralmente são difíceis de se entender e de se depurar, enquanto num projeto de vários componentes menores o contrário ocorre. Apesar disso, grandes mudanças arquiteturais são mais difíceis em vários componentes conectados, uma vez que alterações em um deles podem levar a um efeito cascata e obrigar a alteração de vários outros. Como a facilidade de entendimento foi priorizada, foi escolhido o método de projeto que consiste da interligação de vários componentes pequenos e simples.

Além disso, um princípio que foi seguido no projeto foi o de uso de componentes e técnicas bem conhecidos (e. g. conversão serial-paralelo com registrador de deslocamento) em vez de soluções personalizadas. Isso torna também o projeto mais fácil de se entender e de se reutilizar, no caso de outra pessoa querer aproveitá-lo.

4.4 Interface de Dados Genérica

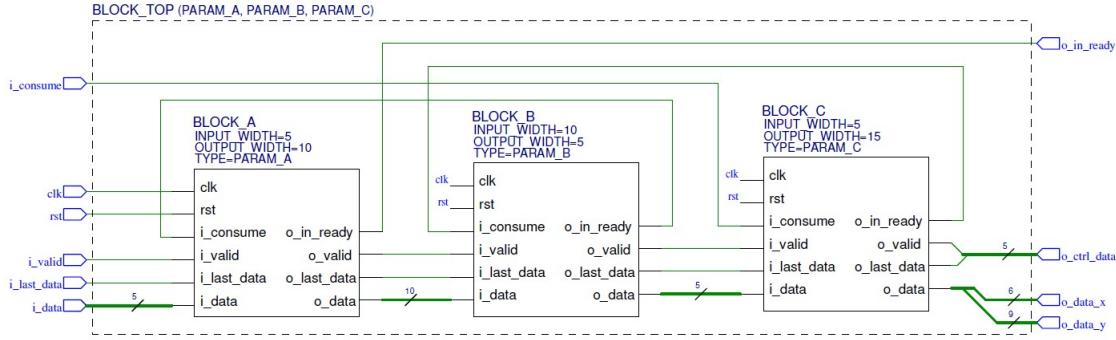
Um ponto fundamental deste trabalho é a interface de dados genérica adotada para transferir dados do **Cod/Decod RLL** para o exterior ou para receber dados de fora. Ela é simplesmente a interface adotada pelo **FEC** em [4], tratando os sinais segundo o protocolo **AMBA® AXI4-Streaming** [5].

Esse tipo protocolo é muito útil porque ele consegue absorver o comportamento de diversos tipos de componente, especialmente aqueles que tratam um fluxo de dados num *pipeline*, como é o caso deste trabalho. Veja a descrição da interface e um exemplo de ligação de blocos na tabela 8 e na figura 10,

Tabela 8: Descrição da interface de dados genérica ([4], p. 56)

Nome	Tipo	Direção	Descrição
i_valid_*	std_logic	entrada	Informa a validade de i_data_*
i_last_data_*	std_logic	entrada	Último pedaço do bloco de dados de entrada
i_data_*	std_logic_vetor(7 downto 0)	entrada	Entrada de dados
i_consume_*	std_logic	entrada	Componente externo pronto para consumir o_data_*
o_valid_*	std_logic	saída	Informa a validade o_data_*
o_last_data_*	std_logic	saída	Último pedaço do bloco de dados de saída
o_data_*	std_logic_vetor(7 downto 0)	saída	Saída de dados
o_in_ready_*	std_logic	saída	Pronto para consumir i_data_*

Figura 10: Exemplo de ligação utilizando a interface de dados genérica ([4], p. 57)



respectivamente. Os prefixos “i_” são relativos a sinais de entrada e os prefixos “o_” aos de saída. O asterisco significa que qualquer sufixo pode ser utilizado para completar o nome do sinal.

O funcionamento do protocolo para essa interface é explicado a seguir. Os sinais *i_valid* e *o_in_ready* controlam o fluxo de entrada da seguinte maneira: somente quando ambos os sinais estiverem em nível alto é considerado que houve uma transferência de dados. Assim, o bloco que recebe os dados, caso esteja sobrecarregado, pode interromper o fluxo colocando *o_in_ready* em nível baixo, mas também pode ser interrompido pelo bloco que fornece os dados por meio do sinal *i_valid*. O controle de fluxo de saída é simétrico. Os sinais *last_data_** são associados aos sinais de *valid_**, ou seja, só podem ser lidos quando houver um dado válido disponível. Eles delimitam blocos de dados.

Algumas questões, no entanto, não ficam claras na descrição do protocolo. Não existe nenhuma indicação de restrição sobre os sinais, ditando seu comportamento. Existe um protocolo, por exemplo, em que o sinal equivalente ao *o_valid_** só poderia ficar em nível alto caso o sinal *i_consume* já estivesse em nível alto [37]. Por isso, assumiu-se que o **FEC** é diretamente compatível com o protocolo **AMBA**, que trata com clareza essas restrições. O **Cod/Decod** foi feito seguindo esse protocolo também.

4.4.1 Protocolo AMBA® 4 AXI4-Streaming

O protocolo *Advanced Microcontroller Bus Architecture* (**AMBA**) AXI4-Streaming é propriedade da empresa ARM e é muito utilizado em diversas aplicações que envolvem **FPGAs** ou **ASICs**. Ele é apenas um dos vários protocolos **AMBA**, e tem como foco transferências de dados do tipo *Streaming*. Transferências *Streaming* tem como característica um fluxo unidirecional de dados, com transmissor e receptor bem definidos, em que a taxa de transmissão geralmente é alta. Além disso, para que a transferência de dados ocorra de maneira adequada, existem várias ferramentas para controle de fluxo.

Esse protocolo de *streaming* propõe uma interface com diversos sinais, sendo alguns deles obrigatórios e outros opcionais, por tratarem de funções específicas. Este trabalho utiliza apenas os sinais obrigatórios e eles podem ser traduzidos para a interface de dados genérica apresentada na seção anterior. Veja esses sinais na tabela 9. Além disso, existem diversas funcionalidades avançadas nesse protocolo que não serão tratadas neste texto.

A principal diferença entre os protocolos do **FEC** e o **AMBA** são as restrições sobre os sinais. O protocolo da ARM prevê algumas regras para os sinais *TVALID* e *TREADY* ([5] p. 19 ou “2-3”). Essas regras existem num contexto com duas entidades: uma delas é o produtor de dados, que gera o sinal de dados e o sinal *TVALID*, e a outra é o consumidor, que gera o sinal *TREADY*. Veja as regras

Tabela 9: Comparação entre os sinais da interface AMBA AXI4-Streaming e da interface de dados genérica

AMBA AXI4-Streaming	Interface de Dados Genérica
TVALID	i_valid_*
TREADY	o_in_ready_*
TLAST	i_last_data_*
TVALID (1)	o_valid_*
TREADY (1)	i_consume_*
TLAST (1)	o_last_data_*

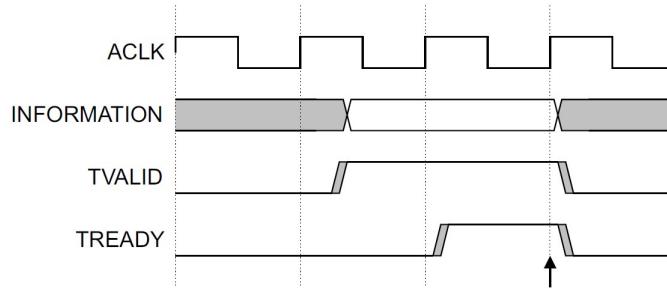
(1) mesmo sinal, mas do ponto de vista do produtor de dados

na lista a seguir.

- Tanto *TVALID* quanto *TREADY* são ativos em nível alto e cada um deles pode sinalizar a qualquer momento. O acionamento de um deles pode ocorrer antes, depois ou juntamente com o do outro.
- O produtor de dados não pode esperar o consumidor sinalizar com *TREADY* antes de colocar *TVALID* em nível alto.
- Uma vez que *TVALID* foi a nível alto pelo produtor, ele deve se manter assim até que o consumidor receba o dado utilizando *TREADY*.
- O consumidor de dados pode esperar pelo sinal *TVALID* antes de sinalizar com *TREADY*.
- Caso o consumidor sinalize com *TREADY*, ele pode desfazer a sinalização antes que *TVALID* vá para o nível alto.

O ponto mais importante a ser considerado para o [Cod/Decod RLL](#) é aquele relativo ao *TVALID* ter que ficar em nível alto até a transferência. Ele gera o efeito que, uma vez que o bloco estiver com um dado válido na saída, ele deve ficar esperando até que o dado seja lido. Em termos de máquina de estados isso significa um *loop* aguardando a leitura, sem poder realizar nenhuma outra operação. Esse tipo de transação com aguardo é inclusive ilustrado na especificação [AMBA](#).

Figura 11: Exemplo de transferência de dados do protocolo [AMBA](#) AXI4-Streaming ([5] p. 19 ou “2-3”)



A figura 11 mostra uma transferência de dados em que o produtor precisa aguardar com o sinal *TVALID* até que o consumidor sinalize com *TREADY*. O dado presente em *INFORMATION* é transferido na borda do relógio *ACLK* marcada com uma seta.

Outro ponto, mas que não é exclusivo do protocolo da ARM é que, como não há indicação explícita de início de pacote, é preciso inferi-lo com o sinal *TLAST* ou por outro meio. Isso significa que ocorrerá erro caso haja transferência de dados entre os pacotes.

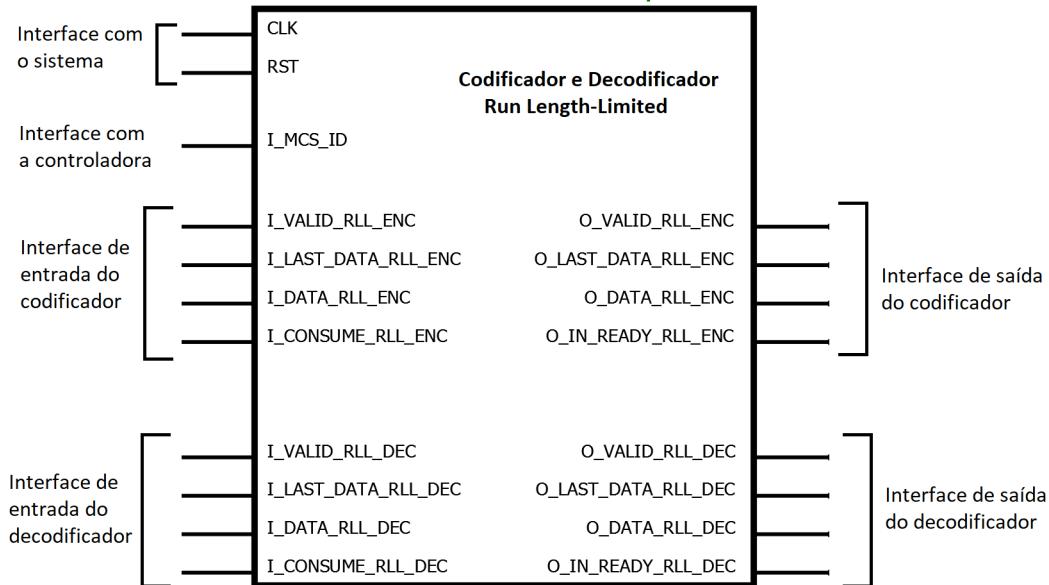
5 Detalhamento do Codificador e Decodificador RLL

Nesta seção será apresentado o codificador e decodificador *run length limited* propriamente dito. As seções anteriores serviram de base para que pudessem ser explicadas melhor as soluções utilizadas para atender aos requisitos. Primeiramente será feita uma análise num nível mais alto de abstração, com informação suficiente para leitores que pretendem apenas utilizar o bloco pronto. Em seguida, será feita uma descrição da arquitetura interna para que o restante do trabalho feito seja apresentado.

5.1 Interface e Comportamento

O primeiro ponto a ser apresentado é a interface do componente, ilustrada na figura 12 e descrita na tabela 10. Segundo o padrão do **FEC**, as entradas possuem o prefixo *i*_ e as saídas o prefixo *o*_. Os sufixos *_rll_enc* e *_rll_dec* utilizados servem para diferenciar os sinais do codificador e do decodificador. Foi utilizada a interface de dados genérica para a transferência de dados do bloco. A comunicação com a controladora do sistema é feita por meio do sinal **MCS_ID**. Deve-se também explicar que o bloco pode operar exclusivamente como codificador ou decodificador, nunca os dois ao mesmo tempo. Essa limitação foi feita devido à arquitetura do sistema com a camada **PHY** completa.

Figura 12: Interface do codificador e decodificador *Run Length Limited* desenvolvido.



Depois de se definir a interface, é necessário definir o comportamento do bloco. Como já explicado na seção 4.4, os sinais da interface de dados genérica seguem um protocolo específico. O ciclo de trabalho do bloco será explicado no passo a passo a seguir.

1. O primeiro dado lido após o *reset* é considerado o primeiro dado do quadro.
2. O **MCS_ID** é lido juntamente com o primeiro dado para definir a configuração do bloco. Caso esse sinal seja inválido, os dados são descartados até que apareça um válido.
3. Os dados são lidos e entregues no ritmo ditado pelo protocolo da interface de dados genérica.
4. Quando o último dado é recebido, o bloco aguarda o início do próximo quadro e recomeça no passo de número 2.

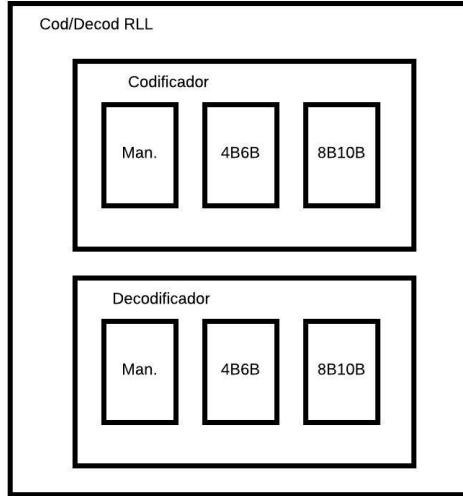
Por fim, faz-se necessário um mapeamento das configurações possíveis do **Cod/Decod**, presente na tabela 11. Nela, cada **MCS_ID** é apresentado com sua respectiva codificação, largura em bits das portas. É necessário indicar a largura pois nem sempre todos os 8 bits são utilizados. Deve-se também pontuar que esse bloco precisa se comunicar com o **FEC** e, por isso, as larguras nas tabelas 7 e 11 devem ser compatíveis.

Tabela 10: Descrição das interfaces do [Cod/Decod RLL](#) desenvolvido.

Nome	Tipo	Direção	Descrição
clk	std_logic	entrada	Relógio do Sistema
rst	std_logic	entrada	Reset do Sistema
i_mcs_id	std_logic_vector(5 downto 0)	entrada	Informa a configuração para o próximo quadro
i_valid_rll_enc	std_logic	entrada	Informa validade de i_data_rll_enc
i_last_data_rll_enc	std_logic	entrada	Último dado do quadro a ser codificado
i_data_rll_enc	std_logic_vector(7 downto 0)	entrada	Entrada de dados a serem codificados
i_consume_rll_enc	std_logic	entrada	Apto a consumir o dado em o_data_rll_enc
i_valid_rll_dec	std_logic	entrada	Informa validade de i_data_rll_dec
i_last_data_rll_dec	std_logic	entrada	Último dado do quadro a ser decodificado
i_data_rll_dec	std_logic	entrada	Entrada de dados a serem decodificados
i_consume_rll_dec	std_logic	entrada	Apto a consumir o dado em o_data_rll_dec
o_valid_rll_enc	std_logic	saída	Informa a validade de o_data_rll_enc
o_last_data_rll_enc	std_logic	saída	Último dado do quadro codificado
o_data_rll_enc	std_logic	saída	Saída de dados codificados
o_in_ready_rll_enc	std_logic	saída	Apto a receber o dado em i_data_rll_enc
o_valid_rll_dec	std_logic	saída	Informa a validade de o_data_rll_dec
o_last_data_rll_dec	std_logic	saída	Último dado do quadro decodificado
o_data_rll_dec	std_logic_vector(7 downto 0)	saída	Saída de dados decodificados
o_in_ready_rll_dec	std_logic	saída	Apto a receber o dado em i_data_rll_dec

5.2 Arquitetura

Esta seção tem como objetivo apresentar a arquitetura interna do bloco desenvolvido. Ela foi dividida em três camadas de hierarquia. Um diagrama simplificado pode ser visto na figura 13. A primeira camada mostra os dois componentes que fazem parte do [Cod/Decod RLL](#). Eles são simplesmente o codificador e o decodificador principais. A segunda camada apresenta a arquitetura interna desses dois componentes. A terceira camada apresenta a arquitetura dos codificadores e decodificadores específicos (Man., 4B6B, 8B10B).

Figura 13: Arquitetura simplificada do [Cod/Decod RLL](#).

O único ponto a se destacar na primeira camada é que o codificador é quase que completamente independente do decodificador. Essa proposta de componentes independentes foi feita para que qualquer um dos dois pudesse ser facilmente extraído do bloco principal e usado separadamente, caso necessário.

5.2.1 Arquitetura do Decodificador

Como indicado na figura 13, as arquiteturas do codificador e do decodificador são muito semelhantes. Por isso, para manter a progressão do texto, nesta seção será mostrada em detalhes a arquitetura do decodificador enquanto a do codificador será apenas comparada à do primeiro.

A discussão deve começar com o diagrama RTL do decodificador, que é apresentado na figura 14.

Tabela 11: Modos de operação do [Cod/Decod RLL](#) com as respectivas larguras de bits na porta de dados.

	MCS_ID	Taxa de Dados	i_data_rll_enc (largura)	o_data_rll_dec (largura)	Codificação
PHY I					
0	000000	11.67 kb/s	4	1	<i>Manchester</i>
1	000001	24.44 kb/s	3	1	
2	000010	48.89 kb/s	3	1	
3	000011	73.3 kb/s	4	4	
4	000100	100 kb/s	8	8	
5	000101	35.56 kb/s	4	4	
6	000110	71.11 kb/s	4	4	
7	000111	124.4 kb/s	4	4	<i>4B6B</i>
8	001000	266.6 kb/s	8	8	
PHY II					
16	010000	1.25 Mb/s	8	8	<i>4B6B</i>
17	010001	2 Mb/s	8	8	
18	010010	2.5 Mb/s	8	8	
19	010011	4 Mb/s	8	8	
20	010100	5 Mb/s	8	8	
21	010101	6 Mb/s	8	8	
22	010110	9.6 Mb/s	8	8	
23	010111	12 Mb/s	8	8	<i>8B10B</i>
24	011000	19.2 Mb/s	8	8	
25	011001	24 Mb/s	8	8	
26	011010	38.4 Mb/s	8	8	
27	011011	48 Mb/s	8	8	
28	011100	76.8 Mb/s	8	8	
29	011101	96 Mb/s	8	8	
PHY III					
32	100000	12 Mb/s	8	8	<i>Nenhuma</i>
33	100001	18 Mb/s	8	8	
34	100010	24 Mb/s	8	8	
35	100011	36 Mb/s	8	8	
36	100100	48 Mb/s	8	8	
37	100101	72 Mb/s	8	8	
38	100110	96 Mb/s	8	8	

Ele mostra todos os componentes presentes na arquitetura com suas respectivas ligações, sendo que algumas ligações são feitas por meio do uso de bandeiras. É importante notar 4 componentes no centro: o bloco *Manchester*, o *4B6B*, o *8B10B* e o *NOCODE*. Eles são decodificadores específicos que serão explicados nas seções [5.2.2](#), [5.2.3](#) e [5.2.4](#), por se tratarem de componentes mais complexos.

Acompanhando o diagrama da figura [14](#), vemos que há um fluxo de dados da entrada para a saída, indo da esquerda para a direita. Além disso, há uma controladora que se liga a praticamente todo o caminho de dados. A proposta do decodificador é que, dado o modo de operação, os bits recebidos na entrada passem por um dos caminhos possíveis e sejam entregues na saída.

A seleção do percurso que os dados irão percorrer começa com o sinal de entrada [MCS_ID](#). Uma vez recebido o sinal, um componente específico o traduz e aciona todos os multiplexadores e demultiplexadores presentes no caminho de dados. A controladora trabalha apenas com os sinais já multiplexados de maneira que o modo de operação é transparente para ela.

Para que os bits sejam apresentados no formato correto para os decodificadores específicos, é preciso realizar uma conversão serial-paralelo por meio dos componentes *Serial In Parallel Out SIPO*. Quando o conversor correspondente é preenchido, os dados são decodificados.

Depois de decodificados, os dados devem ser convertidos novamente de paralelo para serial por meio de componentes *Parallel In Serial Out PISO* e entregues para um registrador de deslocamento ligado à saída de dados. Existe essa necessidade porque o número de bits na saída depende do modo de operação.

O controle da largura da saída é feito com um contador progressivo/regressivo que acompanha o número de bits no decodificador. A partir do modo de operação, ele define quantos bits devem entrar (e. g. [MCS_ID](#) 4: código *Manchester* com 8 bits na saída requer 16 bits na entrada). O contador acompanha os bits que entram até alcançar o valor definido e bloqueia a entrada. Depois, ele vai decrementando até que todos os bits sejam decodificados.

Por fim, é importante destacar os sinais de erro que aparecem no caminho de dados. Antes de mais nada, deve-se dizer que eles são apenas para depuração e não são repassados para a camada superior. Além disso, alguns deles vêm da controladora enquanto outros vêm diretamente do caminho de dados. Eles servem para indicar três situações possíveis de erro. A primeira delas é quando uma palavra inválida chega no decodificador. A segunda é quando um [MCS_ID](#) inválido entra como configuração. A terceira é quando o quadro termina antes de o decodificador receber o número adequado de bits (e. g. no [MCS_ID](#) 4 devem entrar 16 bits para uma decodificação correta).

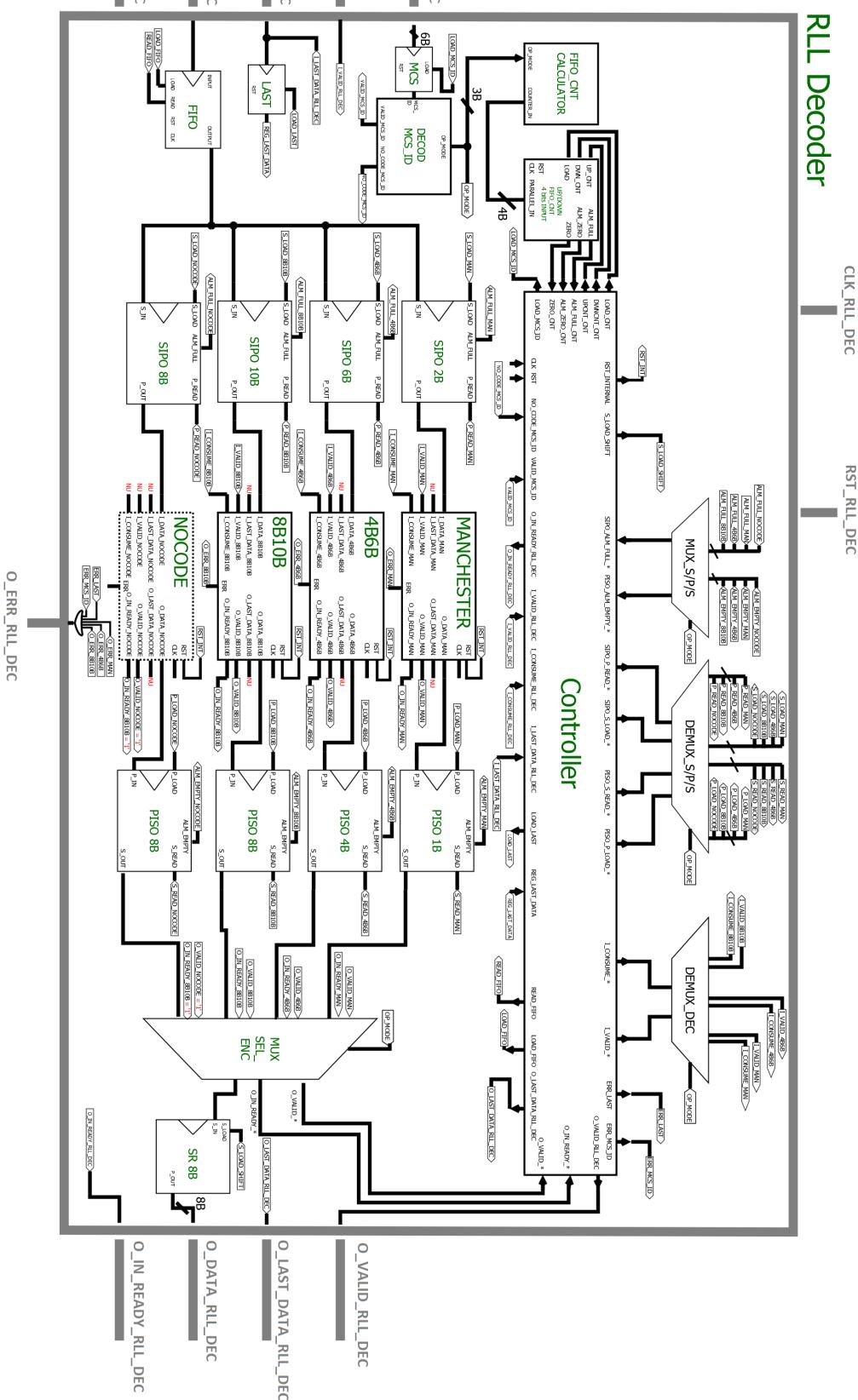


Figura 14: Arquitetura interna do decodificador RLL

Nenhum dos erros interrompe o fluxo de dados e a sinalização serve apenas como indicador de algum problema. O tratamento do primeiro tipo de erro é simplesmente usar a palavra inválida para gerar uma saída válida de maneira arbitrária. O do segundo tipo é ler e descartar os dados até que um **MCS_ID** válido apareça. O do terceiro é colocar bits de lixo no espaço que falta.

O codificador trabalha de maneira muito semelhante ao decodificador, uma vez que os blocos são simétricos. As principais diferenças entre os dois são que a **FIFO** e o registrador de deslocamento ficam em posições trocadas e que não existe erro devido a palavras inválidas pois todas as entradas possíveis são mapeadas. A arquitetura interna do codificador aparece no [apêndice](#).

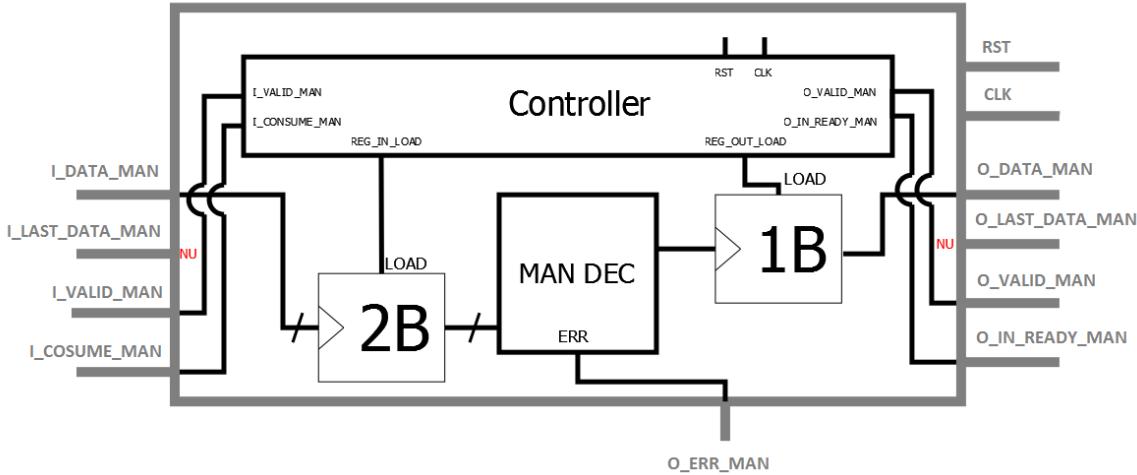
Uma vez que os principais blocos construtivos já foram descritos, é interessante tomar uma nota sobre a arquitetura. Como o modo de operação é transparente para a controladora, é muito simples acrescentar ou retirar um modo do caminho de dados. Para isso, basta seguir a interface utilizada (**SIPO** + interface de dados genérica + **PISO**) e configurar o seletor que controla os multiplexadores. Como a interface de dados genérica consegue se adaptar a diversos casos de uso, vários blocos diferentes podem ser criados e encapsulados com ela.

5.2.2 Codificadores/Decodificadores *Manchester* e *4B6B*

Uma vez descrito o comportamento geral do codificador e do decodificador, é necessário descrever os blocos específicos de cada tipo de codificação. Eles são o núcleo do circuito e, por isso, muito da arquitetura foi definida a partir deles. Deve-se dizer que esses blocos seguem a interface de dados genérica e que a largura de dados em suas entradas e saídas é fixa. Nesta seção serão explicados os codificadores e decodificadores específicos *Manchester* e *4B6B* enquanto na seção seguinte serão explicados os restantes.

Os blocos *Manchester* e *4B6B* foram implementados do zero, sem o uso de nenhum código de terceiros, por se tratarem de blocos bastante simples e parecidos. Tanto a codificação quanto a decodificação foram implementadas na forma de tabelas usando circuitos combinacionais. A estrutura deles é quase igual e a do decodificador *Manchester* pode ser vista na figura 15.

Figura 15: Estrutura interna do decodificador específico *Manchester*



A codificação foi feita simplesmente como uma transcrição das tabelas 1 e 3 com suas definições. As decodificações, no entanto, não possuíam essa tradução direta. Foi preciso encontrar em [3] um tratamento razoável para as palavras inválidas que possivelmente chegariam. A decodificação *Manchester* foi feita seguindo a tabela 12, que escolhia um valor arbitrário para a saída e indicava um estado de erro para entradas inválidas. A decodificação *4B6B* também foi feita com uma tabela, de maneira semelhante. Foi utilizada a saída da palavra válida com a menor distância de *Hamming* da palavra que entrasse no decodificador. Caso a palavra fosse inválida, o sinal de erro ficaria em nível alto.

Tabela 12: Tabela de decodificação *Manchester*

Entrada	Saída	Erro
00	0	1
01	0	0
10	1	0
11	1	1

5.2.3 Codificador/Decodificador *NOCODE*

O codificador e o decodificador *NOCODE* possuem comportamento bastante diferente dos *Manchester* e 4B6B, pois ele é feito por meio de um tratamento especial. Ele serve apenas para passar os dados dos registradores de entrada para os registradores de saída sem alterá-los. O caminho dos dados implementado foi bastante diferente quando se compara o codificador e decodificador específicos.

Quando o codificador entra no modo *NOCODE*, os dados do registrador de deslocamento na entrada vão direto para a **FIFO** de saída, num caso excepcional previsto pela controladora.

Por outro lado, quando o decodificador entra no modo *NOCODE*, os dados precisam passar por um tratamento diferenciado, que aparece na figura 14. Eles passam pelo **SIFO** e pelo **PISO** para que os dados fiquem no formato correto. Quanto ao bloco *NOCODE*, ele simplesmente mimetiza a interface de dados genérica e repassa os dados sem modificá-los.

5.2.4 Codificador/Decodificador 8B10B

O codificador e o decodificador específicos 8B10B, como já explicado, encapsulam os blocos fornecidos por [32] na interface de dados genérica. Para fazer o encapsulamento correto, foi preciso estudar os blocos prontos e controlá-los de maneira adequada.

O decodificador da *OpenCores* simplesmente apresentava uma latência de dois ciclos de relógio e bastava esperá-lo para se obter a decodificação correta. Um ponto importante a se destacar é que o bloco pronto decodificador não faz nenhum tratamento de palavras inválidas, apresentando sempre uma saída para o que quer que apareça em sua entrada. Não foi feito nenhum tratamento de palavras inválidas nesse caso.

O codificador da *OpenCores* teve um tratamento um pouco mais complexo. Como descrito na seção 1.3.3, as palavras do código 8B10B apresentam uma disparidade, o que significa que o próximo dado de 10 bits a sair depende do que havia sido codificado anteriormente. O bloco da *OpenCores* consumia os dados um seguido do outro, modificando sua disparidade sem parar. Como a interface de dados genérica prevê interrupções no fluxo de dados, o bloco continuaria operando nesses momentos de espera e a disparidade se perderia. A solução para isso foi utilizar a técnica de *clock gating* sobre o codificador 8B10B para que ele atuasse somente sobre dados válidos e ficasse parado quando não houvesse demanda.

Tanto para o codificador quanto para o decodificador específicos houve um tratamento para o caractere especial de controle K. Como o controle da transmissão é feito em outras camadas do sistema, esse caractere não teria uso. No codificador esse sinal foi mantido como desativado, e no decodificador ele foi usado como indicador de erro pois não é prevista nenhuma palavra de controle durante a transmissão.

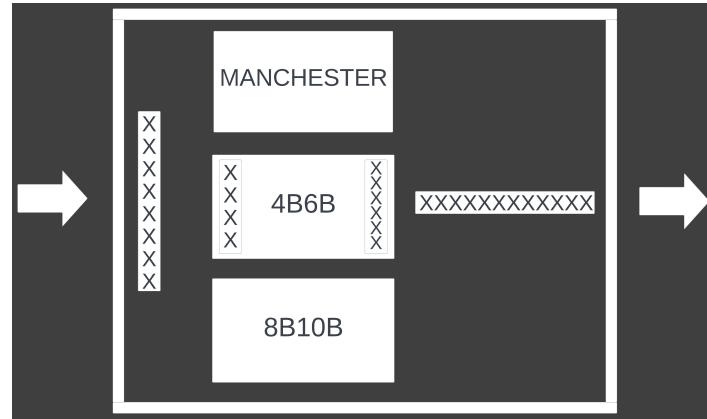
A estrutura tanto do codificador específico 8B10B quanto do decodificador é semelhante à do *Manchester* da figura 15 e pode ser visto no [apêndice](#).

5.3 Exemplo de funcionamento

Esta seção consiste em um conjunto de imagens mostrando o funcionamento do [Cod/Decod RLL](#) em um caso específico de operação, a partir de uma arquitetura simplificada. O objetivo dessa demonstração é fornecer uma visão geral da arquitetura.

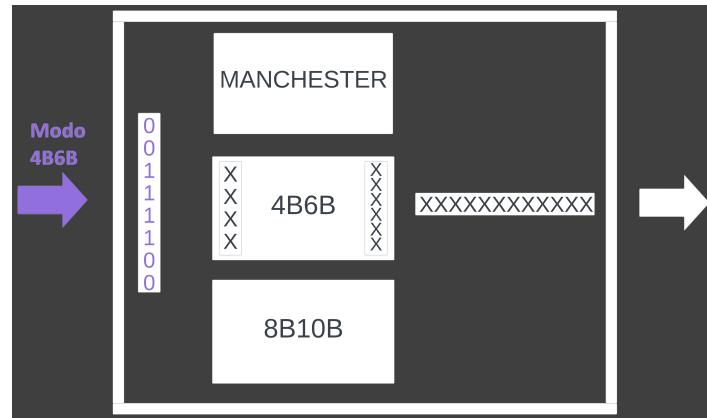
Primeiramente deve ser feita uma apresentação da arquitetura simplificada. Na figura 16, temos o codificador **RLL** e seus três tipos de codificação dentro (*Manchester*, 4B6B e 8B10B). Os retângulos com "X" são registradores com lixo armazenado. Como nesta demonstração a codificação utilizada foi a 4B6B, apenas esse bloco teve seus registradores destacados. Por fim, as setas ao redor do codificador **RLL** representam a direção do fluxo de bits.

Figura 16: Arquitetura Simplificada utilizada no exemplo de funcionamento.



O exemplo começa quando o codificador recebe a primeira palavra do quadro, juntamente com o modo de operação “4B6B”. Os oito bits que entram no primeiro registrador são válidos e coloridos de roxo, vistos na figura 17.

Figura 17: Codificador de exemplo recebendo a palavra de entrada juntamente com o modo de operação.



Uma vez que o codificador **RLL** receber a sua configuração, ele prepara o caminho de dados selecionando o bloco “4B6B”. Quando o caminho de dados estiver pronto, os bits da entrada são transferidos de maneira serial até o registrador do “4B6B”. Veja na figura 18.

Figura 18: Codificador de exemplo preenchendo os registradores do cod. específico 4B6B.

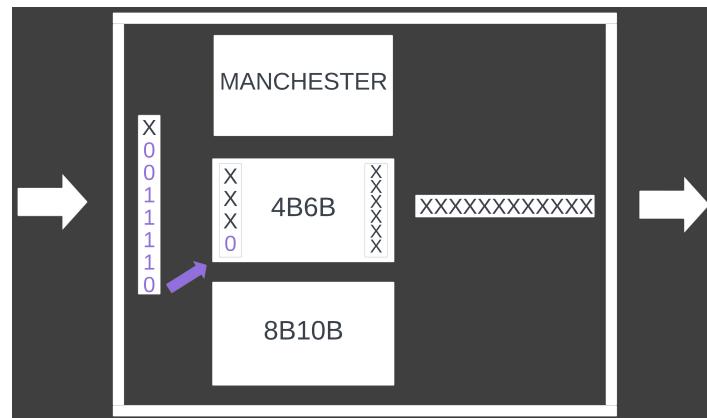
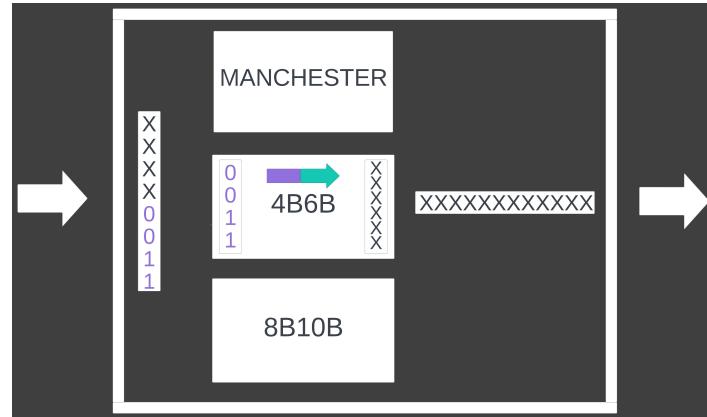


Figura 19: Codificador de exemplo executando a codificação de 4 bits para 6 bits.



Quando o primeiro registrador do codificador “4B6B” estiver preenchido, os 4 bits serão codificados em 6 bits. Esses novos bits são armazenados em um outro registrador, coloridos de verde, como se pode ver na figura 19.

Os 6 bits codificados são, então, transferidos para o registrador horizontal, que no caso é uma **FIFO**. A transferência também é feita de forma serial. Isso é ilustrado na figura 20.

Figura 20: Codificador de exemplo transferindo os bits para a FIFO.

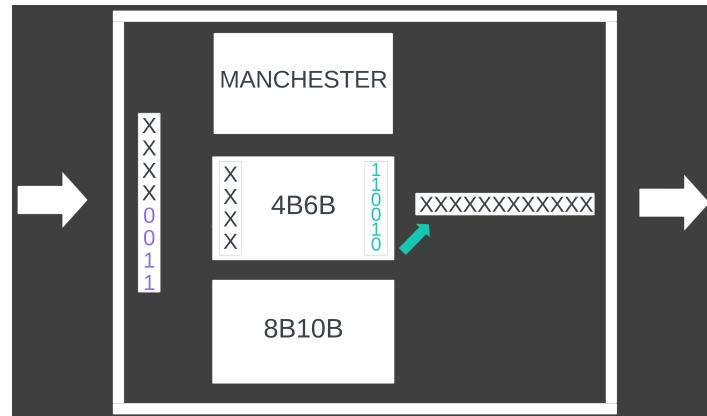
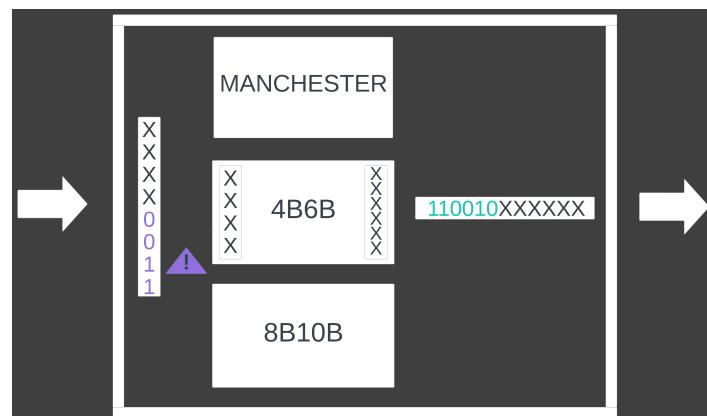


Figura 21: Codificador de exemplo conferindo o registrador de entrada ao término da primeira rodada de codificação.

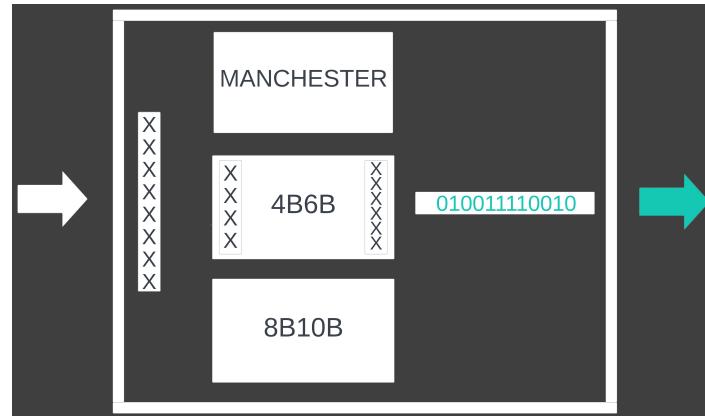


Quando o registrador do “4B6B” tiver esvaziado, o codificador **RLL** confere se o registrador de

entrada, com bits roxos, já foi esvaziado. Como se pode ver na figura 21, o registrador ainda possui outros 4 bits a serem codificados. Nesse caso, os passos das figuras anteriores são executados novamente até que todos os bits estejam na FIFO.

Uma vez que todos os bits forem codificados, a saída se torna válida e o codificador começa a entregar os bits. Veja na figura 22. Note que a métrica para decidir que a saída está pronta é o registrador de entrada esvaziar e não a FIFO estar preenchida. É completamente viável, por exemplo, a FIFO estar com apenas dois bits e a saída estar pronta.

Figura 22: Codificador de exemplo pronto para entregar a saída.



6 Verificação e Performance

Nesta seção será descrito o método utilizado para verificação dos componentes do [Cod/Decod RLL](#) e também do bloco como um todo. Além disso, serão apresentados os ensaios feitos para se extrair os parâmetros de performance. Todos os testes foram feitos utilizando o simulador [RTL](#) do *ModelSim* presente no *Quartus Prime*.

Simulações desse tipo tem como princípio aplicar diferentes sinais de entrada no componente a ser testado para verificar seu comportamento. Existem algumas opções para se gerar esses sinais. Eles podem ser gerados diretamente pelo simulador, por comandos em [HDL](#), ou ainda por meio de um componente personalizado conectado ao componente a ser testado. Além disso, o resultado da simulação pode ser analisado manualmente ou com alguma forma de automação.

Como, por princípio de projeto, os componentes criados são relativamente simples, foi possível utilizar o método que gera os sinais com comandos em [HDL](#). Também optou-se por fazer uma análise manual dos gráficos gerados nas simulações. Para as partes pequenas do [Cod/Decod](#) esse método foi bastante razoável, mas para o codificador ou decodificador gerais ele se mostrou menos efetivo.

6.1 Verificação Funcional Por Meio de Simulação

A proposta da verificação funcional é de se exercitar cada componente com o máximo de entradas distintas possível. Claramente, num sistema muito grande, isso não é viável devido ao número elevado de possibilidades. Entretanto, nos componentes simples que compõem o [Cod/Decod](#), o método é bastante eficaz.

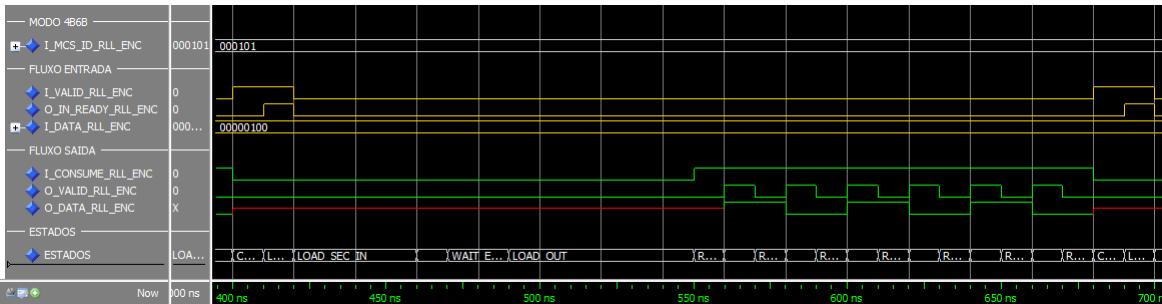
Vários componentes como registradores de deslocamento e [FIFOs](#) puderam ser testados de quase todas as formas possíveis, injetando os sinais de entrada de várias formas arbitrárias. Blocos combinacionais como o que controla os multiplexadores puderam ser testados à exaustão, com todas as combinações de entrada possíveis.

As controladoras foram verificadas criando rotas que juntas exercitavam todas as transições presentes no seu diagrama de estados. As controladoras dos codificadores e decodificadores específicos são consideravelmente simples, enquanto aquelas do codificador e do decodificador principais são mais complexas. Os diagramas de estados destas últimas aparecem no [apêndice](#)).

O codificador e o decodificador principais não puderam ser testados de maneira completa, devido à grande variedade de possibilidades de uso. Nesse caso, foram propostos casos que exercitavam todos os modos de operação presentes na tabela 11. Neles, o acionamento dos sinais seguia o comportamento padrão esperado (e. g. o protocolo [AMBA](#) é sempre respeitado).

Esses últimos testes foram aproveitados para se extraírem os parâmetros de performance do bloco construído. É possível ver um exemplo de simulação feita do *ModelSim* na figura 23. Nela, é visto um teste do codificador operando no modo 4B6B em que a entrada recebe 4 bits ([MCS.ID](#) 5). Os sinais relativos à entrada de dados foram marcados em dourado e os relativos à saída em verde. Os sinais de [MCS.ID](#) e de estados foram marcados em cinza.

Figura 23: Exemplo de formas de onda obtidos durante os testes do [Cod/Decod](#). Representa o codificador no modo 4B6B com 4 bits de entrada.



O trecho exibido na janela ocorre no caso em que o codificador está no meio de um quadro, lendo e entregando dados seguindo um *loop* na máquina de estados. O sinal *O_IN_READY* em nível alto no início indica que o bloco acabou de receber mais uma palavra e a atividade no sinal *O_VALID* significa que os dados estão sendo entregues, encerrando o ciclo. Pode-se ver que o resultado da codificação

está correto pois, para um sinal de entrada “0100”, a saída correspondente deve ser “010101”. Note que o **LSB** é enviado primeiro. Além disso, é possível perceber que as situações que consomem mais tempo do codificador são a entrega de dados e as conversões serial-paralelo que são feitas nos estados *LOAD_SEC_IN* e *LOAD_OUT*. Essa distribuição de tempo se repete para os demais modos de operação.

6.2 Medidas de Performance

Como descrito na seção 3.4, foram considerados três parâmetros de performance para se caracterizar o codificador **RLL**, sendo eles a latência, a vazão de dados (*throughput*) e o tamanho do circuito. A vazão e a latência são parâmetros que dependem do modo de operação, pois o caminho de dados é utilizado de maneira diferente. O tamanho do circuito é fixo e depende apenas do tipo de dispositivo no qual o bloco é sintetizado. Para fins de comparação, foram seguidas as mesmas premissas de [4], em que o **FPGA** selecionado foi um *Cyclone V* operando a uma frequência de relógio de 80 MHz. Essa unificação de condições de operação tem como objetivo facilitar uma visão de performance da camada **PHY** completa a ser construída. Essa decisão possibilita, por exemplo, fazer uma comparação mais direta do tamanho dos circuitos.

Antes de detalhar os testes realizados, é importante fazer algumas definições. No contexto destes testes, uma palavra é um conjunto de bits enviados através de um barramento. No caso do **Cod/Decod RLL**, dependendo do modo de operação, uma palavra pode ter 1, 3, 4 ou 8 bits e é transferida entre blocos pelos sinas *i_data_** e *o_data_**. Quadro é um conjunto muito maior de bits, em que cada parte dele possui um significado - cabeçalho, dados da camada **MAC**, verificação de erro, etc. As fronteiras entre quadros são demarcadas com o sinal *i_last_data_** na Interface de Dados Genérica.

6.2.1 Latência

A latência foi medida como o tempo desde o recebimento da primeira palavra de um quadro até o envio da primeira palavra de saída relativa a esse quadro. Como essa latência depende do tempo que os blocos adjacentes levam para fornecer os dados ao **Cod/Decod**, foi considerado o caso ideal em que esses blocos estão sempre aptos a fornecê-los.

Quando comparado com o tempo entre quadros definido pelo IEEE 802.15.7, essa latência se encaixa quase exatamente com a definição. Isso porque, assim que o bloco **RLL** entrega a última palavra do quadro atual, no ciclo de relógio seguinte ele já pode receber a primeira palavra do próximo quadro.

De acordo com os resultados nas tabelas 14 e 15, pode-se notar que a latência do circuito é realmente da ordem de dez vezes menor que a do **FEC**, como discutido na seção 3.4. Por isso, ela terá pouca influência na latência total do sistema, com algumas exceções.

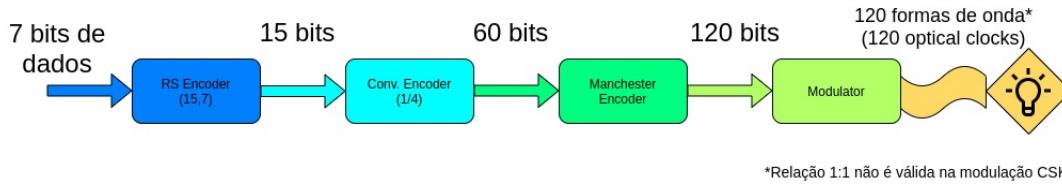
6.2.2 Vazão

A medida da vazão não é tão direta quanto a da latência ou a do tamanho e depende de alguns cálculos e considerações para ser estimada. Primeiramente, no cálculo é considerado um ciclo completo do processamento de uma palavra, ou seja, o **Cod/Decod** começa vazio e termina vazio. Em segundo lugar, a vazão foi considerada no caso mais comum, no meio de um quadro, onde o **Cod/Decod** não precisa se preocupar em ler o **MCS_ID** para escolher a configuração nem em tratar o fim do quadro. Além disso, foi considerado que tanto as entradas quanto as saídas do **Cod/Decod** estavam sempre disponíveis para receber dados, sem nenhuma espera. Essas duas últimas considerações significam que a vazão estimada é a máxima possível.

Por fim, é preciso explicar a opção por se utilizar como unidade de vazão o relógio óptico suportado e não a vazão de fato em Mbps. Como o **Cod/Decod RLL** está ligado ao dispositivo óptico, “a taxa de bits” nessa interface corresponde à frequência possível do relógio óptico, e não à taxa de bits de informação (*Data Rate*). Na figura 24 pode-se ver como os bits são transformados antes de serem enviados no meio. Os 7 bits de dados se transformam em 120 bits no meio óptico. O *RS Encoder*, quando no modo (15,7), entrega 15 bits na saída a cada 7 bits que entram. Em seguida o *Convolutional Encoder*, no modo (1/4) aumenta o número de bits numa razão de 1 para 4. Por fim, a codificação *Manchester* dobra o número de bits na linha de acordo com a sua definição. A decodificação, por sua vez, é apenas o caminho inverso dos bits, seguindo as mesmas proporções.

As duas unidades, *Data Rate* e *Optical Clock Rate* são intercambiáveis pois uma pode ser calculada a partir da outra observando modo de operação. Veja um exemplo de conversão a partir de uma parte

Figura 24: Diagrama mostrando o fluxo que transforma bits de dados em formas de onda na camada **PHY**.



da tabela 19, destacada na tabela 13.

Tabela 13: Entrada da tabela 19 usada no exemplo de vazão para modo de operação da **PHY I**

Modulation	RLL code	Optical clock rate	FEC		Data rate
			Outer code (RS)	Inner code (CC)	
OOK	Manchester	200 kHz	(15,7)	1/4	11.67 kb/s

A partir da eficiência da codificação *Manchester*, do código externo (15,7) e do código interno 1/4 presentes na tabela 13, é possível calcular a taxa de dados utilizando o relógio óptico.

$$11.67kb/s \times \frac{15}{7} \times \frac{4}{1} \times \frac{2}{1} = 200k \text{ optical clock periods per second}$$

Fazer essa comparação entre o relógio óptico (*optical clock rate*) e a velocidade de transmissão (*Data Rate*), infelizmente, não pôde ser feita em modos com a modulação **CSK**, que utiliza três LEDs (RGB) para enviar os dados. A relação entre bits do bloco **RLL** e períodos de relógio óptico não é de 1:1 como nos demais modos e requer um cálculo mais complexo.

Os resultados de vazão nas tabelas 14 e 15 apontam para alguns gargalos de performance no sistema. Quando comparados com a norma IEEE 802.15.7, os requisitos são alcançados em boa parte da camada **PHY**. Duas tabelas completas com todos os modos de operação e com os requisitos de performance aparecem no [apêndice](#).

6.2.3 Tamanho do circuito

O tamanho do circuito descrito em **HDL** foi medido a partir do relatório de compilação do *Quartus Prime*, que descreve o quanto de lógica e de memória do chip *Cyclone V* foi utilizado no projeto. Esses valores são comparados com o total de recursos do **FPGA**.

Como se pode ver a partir dos resultados na tabela 16, o tamanho do **Cod/Decod RLL** é insignificante e não representará nenhum gargalo de performance.

Tabela 14: Latência e Vazão obtidas para o codificador **RLL**

Tipo de Codificação	Largura da Entrada	Latência*	Relógio óptico máximo suportado*
Manchester	3	0,288 us	14,5 MHz
	4	0,363 us	14,9 MHz
	8	0,663 us	15,4 MHz
4B6B	4	0,225 us	17,1 MHz
	8	0,388 us	18,1 MHz
8B10B	8	0,363 us	17,0 MHz
Nenhuma	8	0,163 us	23,7 MHz

* Estimativas feitas considerando um relógio de sistema de 80 MHz.

Tabela 15: Latência e Vazão obtidas para o decodificador [RLL](#)

Tipo de Codificação	Nº de bits na entrada	Latência*	Relógio óptico máximo suportado*
Manchester	2	0,175 us	10,7 MHz
	8	0,550 us	14,2 MHz
	16	1,050 us	15,1 MHz
4B6B	6	0,363 us	16 MHz
	12	0,675 us	17,5 MHz
8B10B	10	0,575 us	17,0 MHz
Nenhuma	8	0,475 us	17,0 MHz

* Estimativas feitas considerando um relógio de sistema de 80 MHz

Tabela 16: Parâmetros de tamanho obtidos para [Cod/Decod RLL](#)

Parâmetro	Porcentagem
Lógica Utilizada	1%
Bits de memória	<1%

7 Conclusão e Trabalhos Futuros

Uma vez apresentado todo o desenvolvimento do projeto do [Cod/Decod RLL](#), desde o estudo das técnicas envolvidas até a verificação do resultado final, deve-se fazer uma retrospectiva de tudo que foi feito e avaliar o que poderia ser considerado para o futuro.

Quanto à arquitetura proposta, foi possível ver que um sistema compostos por vários blocos pequenos realmente tornou o trabalho de depuração mais simples e rápido, além de ter sido fundamental para proporcionar a flexibilidade que o circuito possui para o acréscimo/retirada de novos modos de operação. Utilizar a interface de dados genérica também trouxe muita regularidade ao projeto. Essa flexibilidade, no entanto, veio com o custo de várias conversões serial-paralelo, que representam parcela significativa do tempo de operação.

Quanto à performance, é possível dizer que o tamanho do circuito é tão pequeno que não seria relevante num sistema com blocos grandes como o [FEC](#) em [4]. Sobre latência, viu-se que o [Cod/Decod](#) apresenta valores inferiores aos do [FEC](#), mas que devem ser levados em conta. A vazão mostrou-se o pior gargalo de performance, pois apenas cerca três quartos dos modos de operação das [PHY I, II e III](#) são atendidos.

Após a análise feita nos parágrafos anteriores, percebe-se que o [Cod/Decod RLL](#) ainda precisa de alguns ajustes antes de poder ser usado como proposto e atender aos requisitos das camadas [PHY I, II, e III](#). Talvez seja necessário até mudar a proposta da arquitetura do sistema e implementar o modulador de uma maneira diferente da imaginada de início. A lista a seguir pontua algumas coisas que podem ser feitas para se melhorar o projeto do bloco [RLL](#).

- Fazer uma verificação mais completa do circuito, incluindo uma verificação funcional mais cuidadosa, verificação formal, análise de *timing*, etc. A verificação funcional feita, apesar de garantir as funcionalidades básicas, é muito limitada. Sem uma análise de *timing*, por exemplo, não é possível estimar a frequência máxima de operação do circuito com segurança.
- Ampliar o tratamento de erros para que possa ser feita uma depuração mais adequada do sistema.
- Fazer teste do circuito em [FPGA](#), pois vários erros presentes no projeto se manifestam mais claramente após uma implementação física. Uma vez que o compilador aceitou o projeto, a princípio, o *bitstream* que configura o chip já pode ser gerado.
- Avaliar se seria útil mudar a implementação flexível por uma mais especializada para reduzir os tempos de serialização e desserialização.
- Avaliar uma mudança na largura da porta ligada ao modulador, devido ao grande tempo gasto para se entregar os bits. Além disso, o modulador, às vezes, possui uma maior frequência de relógio devido à frente óptica. Isso significa que delegar tarefas simples a esse circuito, como serialização, podem representar uma melhoria de performance.

Além de melhorar o que já foi feito, é possível também fazer modificações no bloco para expandir suas capacidades de operação. Seria possível, por exemplo, adicionar um tipo de codificação completamente novo para se avaliar sua performance num sistema [VLC](#). Foi cogitado, durante o projeto, implementar um codificador 5B10B, visto em [38].

Por fim, espera-se que este trabalho possa ser útil para a comunidade científica e represente um avanço no desenvolvimento de dispositivos [VLC](#). O objetivo final do projeto maior, que é criar uma camada [PHY](#) completa e *open source*, ainda depende de mais blocos funcionais e de mais ajustes de performance nos blocos já feitos, mas está ficando cada vez mais próximo.

Referências

- [1] “Back to the future: Manchester encoding – part 1,” Embedded: News and Resources for the Electronics Community, 2008, date accessed: May 31th 2023. [Online]. Available: <https://www.embedded.com/back-to-the-future-manchester-encoding-part-1/>
- [2] Lattice Semiconductors, “8b/10b encoder/decoder,” 2015, reference design RD1012.
- [3] B. Hussain, “Design and implementation of visible light communication systems,” Master’s thesis, Hong Kong University, 2015.
- [4] M. G. Silva, “Digital design of a forward error correction system for ieee 802.15.7,” Master’s thesis, Universidade Federal de Minas Gerais, 2020.
- [5] “AMBA interfaces,” ARM, 2022, date accessed: Nov 20th 2022. [Online]. Available: <https://developer.arm.com/Architectures/AMBA>
- [6] H. Haas, “Lifi is a paradigm-shifting 5g technology,” *Reviews in Physics*, vol. 3, pp. 26–31, 2018.
- [7] F. Che, “Vlc transceiver system design using led for ieee 802.15. 7 standard,” Master’s thesis, Hong Kong University, 2015.
- [8] I. Stefan, H. Burchardt, and H. Haas, “Area spectral efficiency performance comparison between vlc and rf femtocell networks,” in *2013 IEEE international conference on communications (ICC)*. IEEE, 2013, pp. 3825–3829.
- [9] M. Ayyash, H. Elgala, A. Khreishah, V. Jungnickel, T. Little, S. Shao, M. Rahaim, D. Schulz, J. Hilt, and R. Freund, “Coexistence of wifi and lifi toward 5g: concepts, opportunities, and challenges,” *IEEE Communications Magazine*, vol. 54, no. 2, pp. 64–71, 2016.
- [10] H. T. Friis, “A note on a simple transmission formula,” *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, 1946.
- [11] S. Haruyama, V. Chairman, and J. Yokohama, “Japan’s visible light communications consortium and its standardization activities,” *Presentation at IEEE*, vol. 802, 2008.
- [12] IEEE, “Standard for local and metropolitan area networks - part 15.7: Short range wireless optical communication using visible light,” 2011.
- [13] ———, “Standard for local and metropolitan area networks - part 15.7: Short range wireless optical communication using visible light (revision of ieee std 802.15.7-2011),” 2018.
- [14] IEEE, “IEEE draft standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 7: Light communications,” Feb 2022. [Online]. Available: <https://standards.ieee.org/ieee/802.11bb/10823/>
- [15] J. Luo, “M-ary run length limited coding,” Master’s thesis, Rochester Institute of Technology, 1996.
- [16] “[discontinued] ash transceiver software designer’s guide,” RFMonolithics, 2002, date accessed: Dec 5th 2022. [Online]. Available: https://wireless.murata.com/media/products/apnotes/tr_swg05.pdf?ref=rpm.com
- [17] “Computer 50 the university of manchester celebrates the birth of the modern computer,” Manchester University, 1998, date accessed: May 31th 2023. [Online]. Available: <http://curation.cs.manchester.ac.uk/computer50/www.computer50.org/index.html>
- [18] A. X. Widmer and P. A. Franaszek, “A dc-balanced, partitioned-block, 8b/10b transmission code,” *IBM Journal of research and development*, vol. 27, no. 5, pp. 440–451, 1983.
- [19] P. A. Franaszek and A. X. Widmer, “Byte oriented dc balanced (0, 4) 8b/10b partitioned block transmission code,” Dec. 4 1984, uS Patent 4,486,739.

- [20] IEEE, “Ieee standard for ethernet - ieee std 802.3-2022 (revision of ieee std 802.3-2018),” 2022.
- [21] V. Jungnickel, M. Hinrichs, K. Bober, C. Kottke, A. Corici, M. Emmelmann, J. Rufo, P.-B. Bök, D. Behnke, M. Riege *et al.*, “Enhance lighting for the internet of things,” in *2019 Global LIFI Congress (GLC)*. IEEE, 2019, pp. 1–6.
- [22] A. Pradana, N. Ahmadi, and T. Adionos, “Design and implementation of visible light communication system using pulse width modulation,” *2015 International Conference on Electrical Engineering and Informatics*, 2015.
- [23] Q. Wang, D. Giustiniano, and D. Puccinelli, “An open source research platform for embedded visible light networking,” *IEEE Wireless Communications*, vol. 22, no. 2, pp. 94–100, 2015.
- [24] E. Setiawan, T. Adiono, and S. Fuada, “Modelling the ofdm-based phy layer in soc for visible light communication.” *Int. J. Recent Contributions Eng. Sci. IT*, vol. 7, no. 3, pp. 79–89, 2019.
- [25] ———, “Phy layer design of ofdm-vlc system based on soc using reuse methodology,” in *2018 International SoC Design Conference (ISOCC)*. IEEE, 2018, pp. 115–116.
- [26] F. Che, B. Hussain, L. Wu, and C. P. Yue, “Design and implementation of ieee 802.15. 7 vlc phy-i transceiver,” in *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 2014, pp. 1–4.
- [27] C. G. Gavrincea, J. Baranda, and P. Henarejos, “Rapid prototyping of standard-compliant visible light communications system,” *IEEE Communications Magazine*, vol. 52, no. 7, pp. 80–87, 2014.
- [28] L. M. Matheus, A. B. Vieira, M. A. Vieira, and L. F. Vieira, “Dyrp-vlc: A dynamic routing protocol for wireless ad-hoc visible light communication networks,” Master’s thesis, Universidade Federal de Juiz de Fora, 2019.
- [29] Yin, Smaoui, Heydariaan, and Gnawali, “Purplevlc: Accelerating visible light communication in room-area through pru offloading,” *EWSN ’18: Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, pp. 67–78, 2018.
- [30] Schmid, Richner, Mangold, and Gross, “Enlighting: An indoor visible light communication system based on newtworked light bulbs,” *2016 13th Annual IEEE International Conference on Sensing, Communication and Networking (SECON)*, pp. 1–9, 2016.
- [31] Hewage, Varshney, Hilmia, and Voigt, “modbulb: a modular light bulb for visible light communication,” *Proceedings of the 3rd Workshop on Visible Light Communication Systems*, pp. 13–18, 2016.
- [32] K. Boyette, “8b10b encoder/decoder,” Open Cores, 2006, date accessed: Oct 26th 2022. [Online]. Available: https://opencores.org/projects/8b10b_encdec
- [33] “Dia diagram editor,” The DIA Developers, 2014, date accessed: Nov 20th 2022. [Online]. Available: <http://dia-installer.de/>
- [34] “lpm_shitreg megafunction user guide,” Altera/Intel Corporation, 2006, date accessed: June 4th 2023. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/hdl/mega/mega_file_lpm_shiftreg.htm
- [35] “Fifo intel® fpga ip user guide,” Intel Corporation, 2023, date accessed: June 4th 2023. [Online]. Available: <https://www.intel.com/content/www/us/en/docs/programmable/683522/18-0/fifo-user-guide.html>
- [36] F. Vahid, *Digital Design*. Wiley Press, 2006.
- [37] “Avalon interfaces specifications,” Intel, 2022, date accessed: Nov 20th 2022. [Online]. Available: <https://www.intel.com/content/www/us/en/docs/programmable/683091/20-1/introduction-to-the-interface-specifications.html>
- [38] V. A. Reguera, “New rll code with improved error performance for visible light communication,” *arXiv preprint arXiv:1910.10079*, 2019.

Apêndice

Apêndice A: Diagramas RTL com a arquitetura do Cod/Decod RLL

Figura 25: Interface do Cod/Decod com indicação de conexão com blocos adjacentes

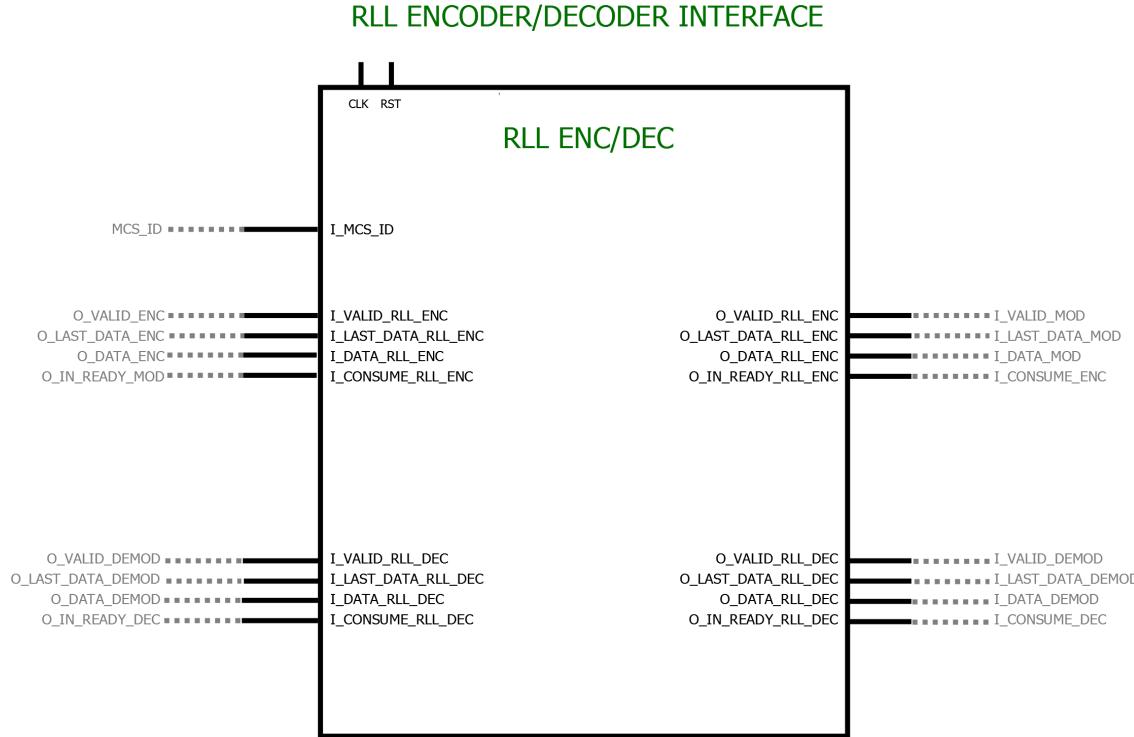


Figura 26: Arquitetura Interna do Cod/Decod

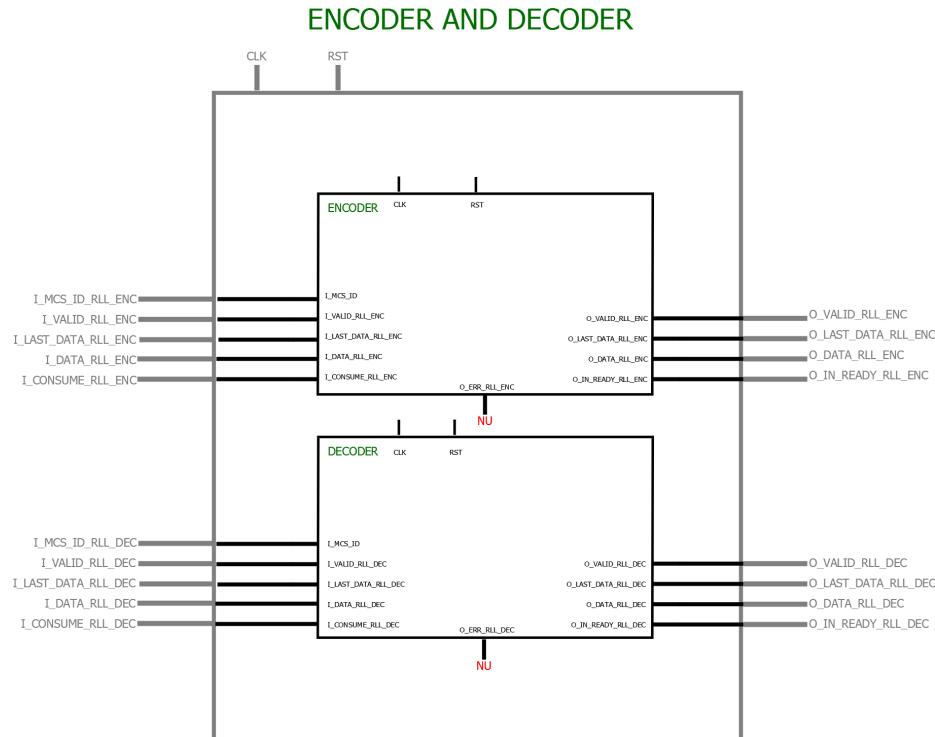


Figura 27: Arquitetura interna do codificador RLL.

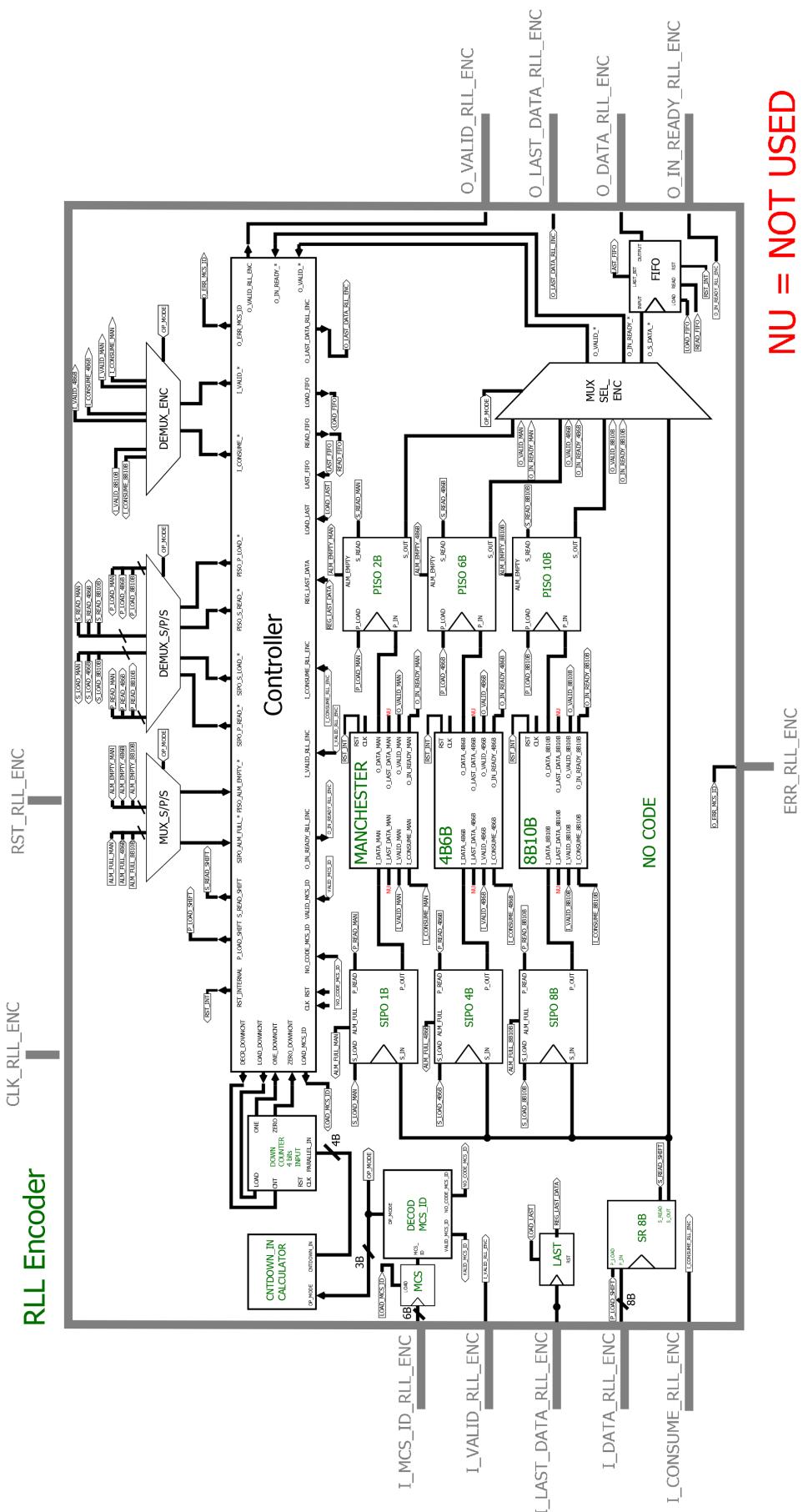


Figura 28: Diagrama de estados do codificador RLL

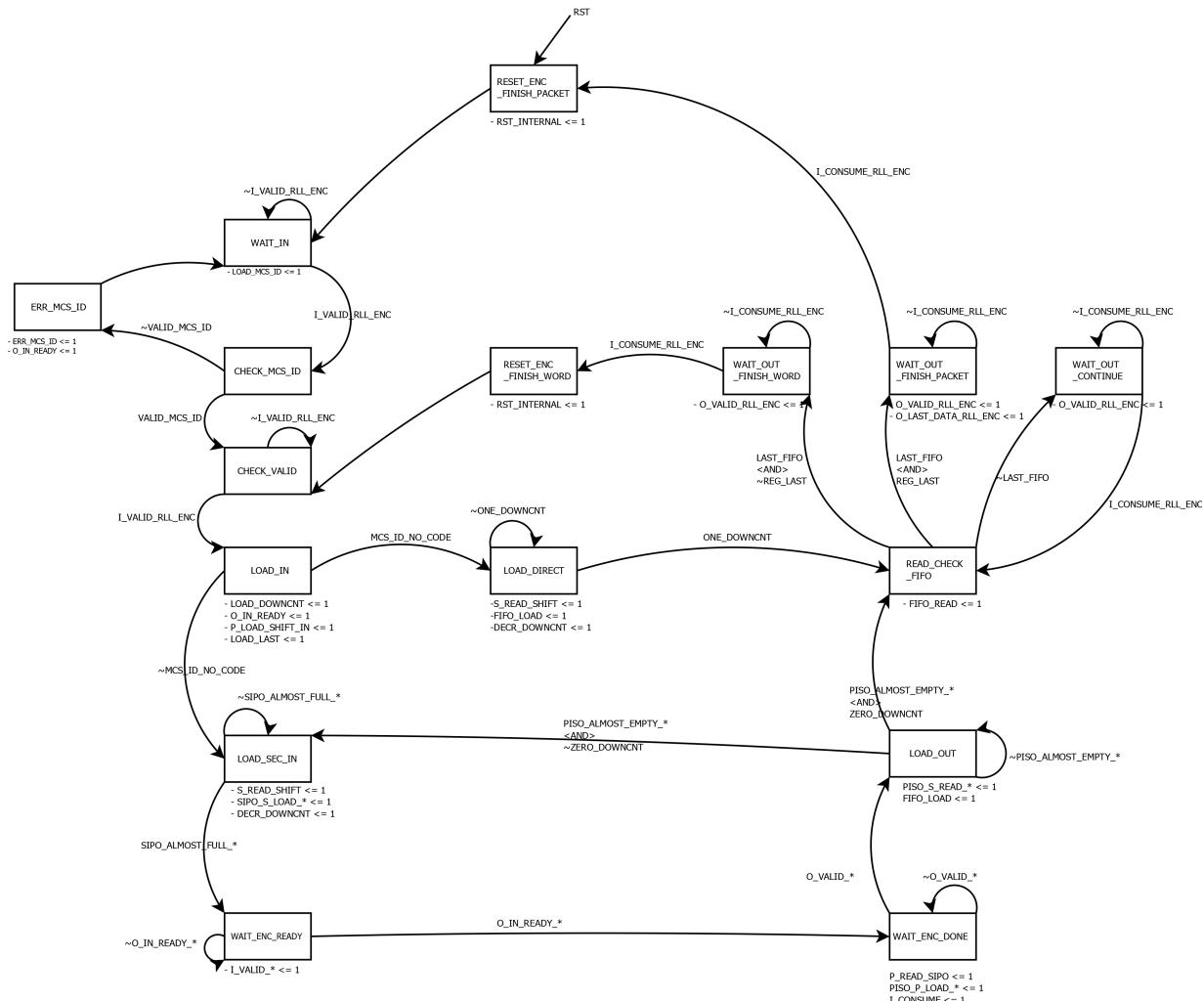


Figura 29: Diagrama de estados do decodificador RLL

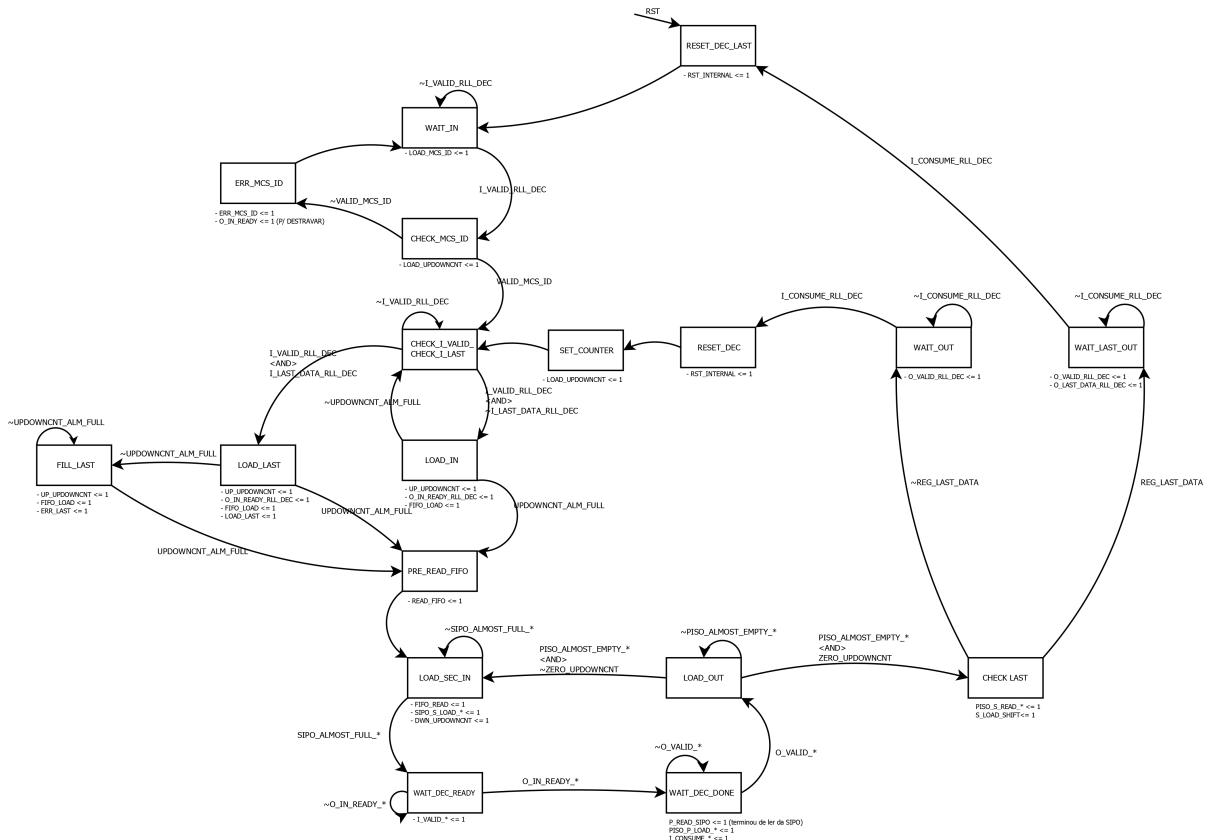
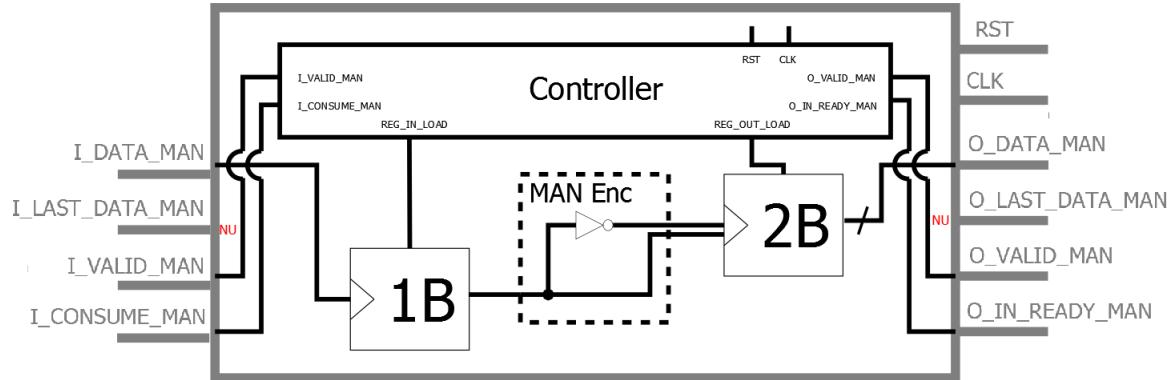
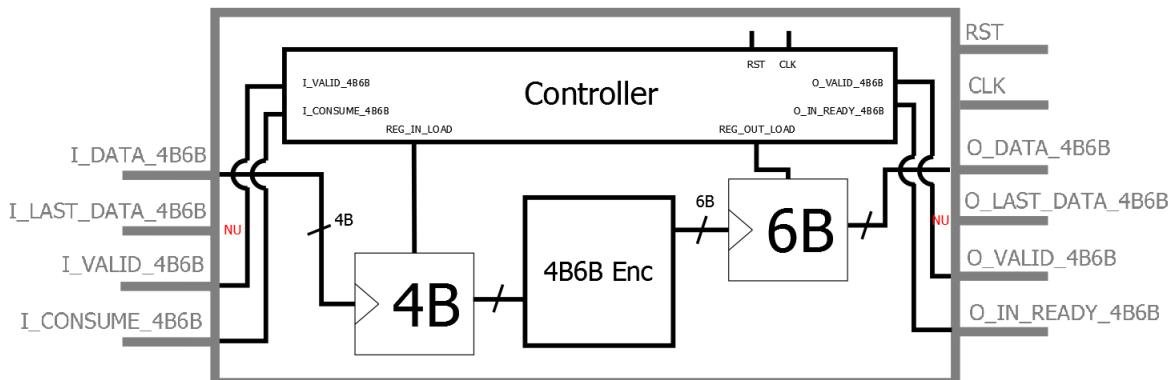


Figura 30: Arquitetura interna dos codificadores específicos

MANCHESTER ENCODER



4B6B ENCODER



8B10B ENCODER

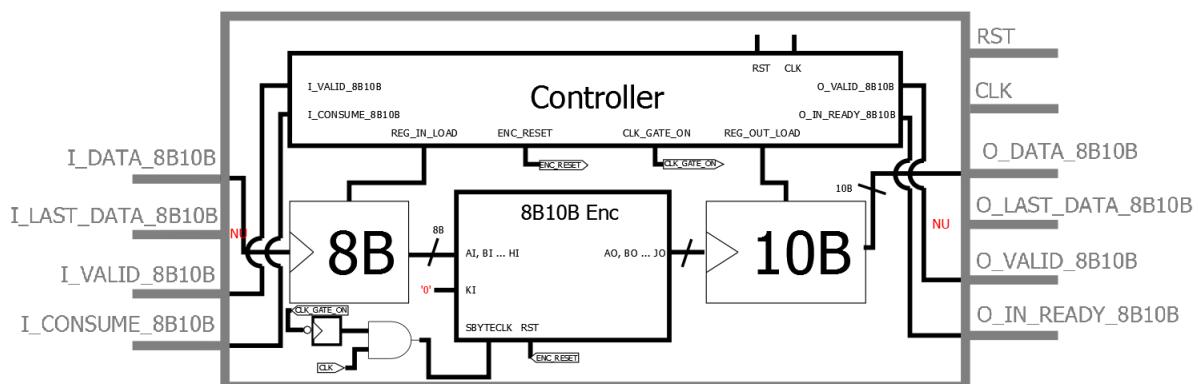
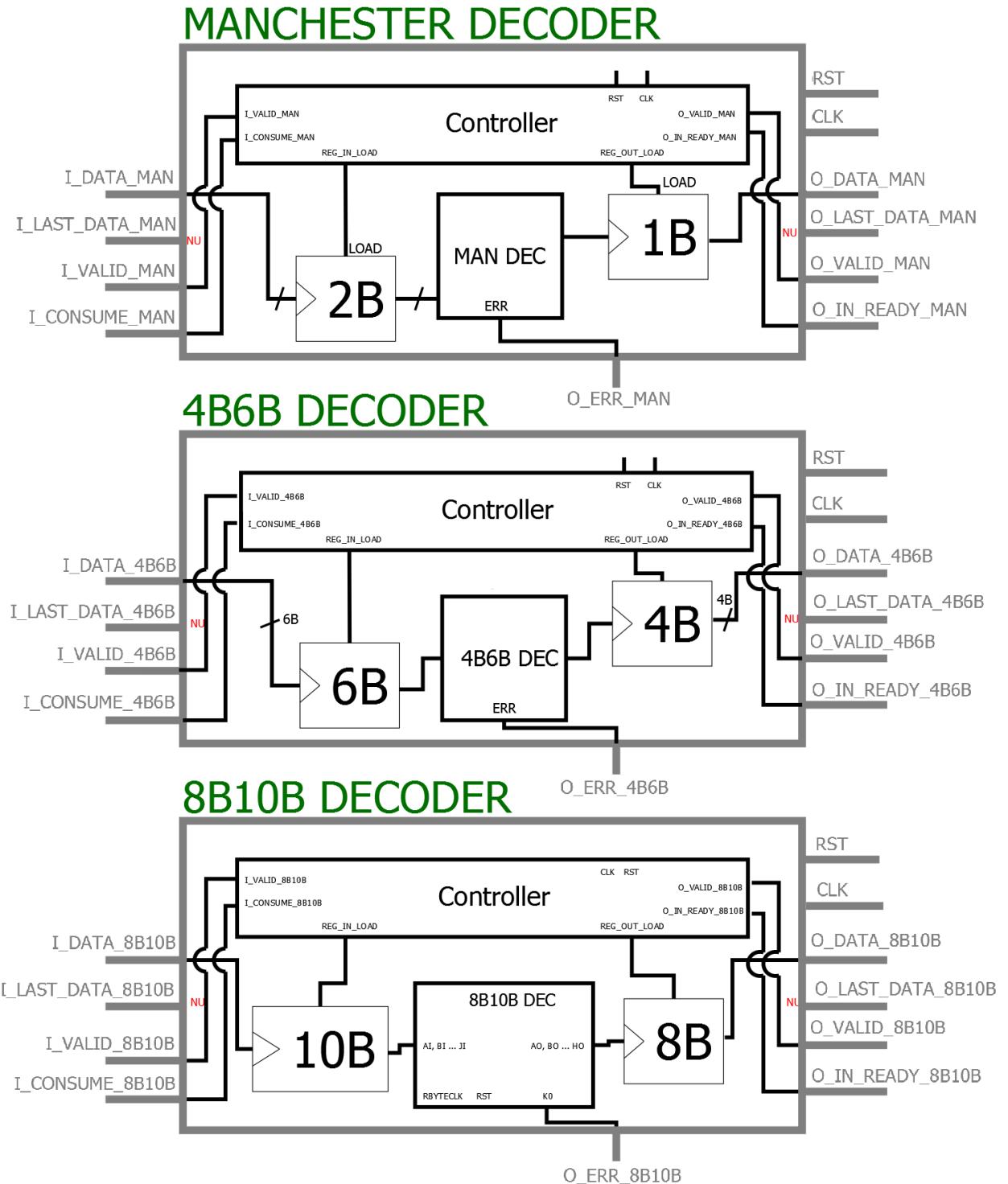


Figura 31: Arquitetura interna dos decodificadores específicos



Apêndice B: Resultados de performance do Cod/Decod RLL

Tabela 17: Performance do Codificador RLL

MCS_ID	Codificação	bits na entrada	Latência*	Relógio óptico suportado*	latência requerida (RIFS)	Relógio óptico requerido			
PHY I									
0	Manchester	4	0,363 us	14,9 MHz	200 us	200 kHz			
1		3	0,288 us	14,5 MHz					
2		4	0,363 us	14,9 MHz					
3		8	0,663 us	15,4 MHz					
4		4	0,225 us	17,1 MHz					
5	4B6B	8	0,388 us	18,1 MHz	100 us	400 kHz			
6		4	0,225 us	17,1 MHz					
7		8	0,388 us	18,1 MHz					
8		4	0,225 us	17,1 MHz					
PHY II									
16	4B6B	8	0,388 us	18,1 MHz	10,6 us	3,75 MHz			
17					5,3 us	7,5 MHz			
18		8B10B	0,363 us	17,0 MHz	2,667 us	15 MHz			
19					1,333 us	30 MHz			
20					0,667 us	60 MHz			
21					0,333 us	120 MHz			
22									
23	8B10B	8	0,363 us	17,0 MHz					
24									
25									
26									
27									
28	PHY III								
29	PHY III								
32	Nenhuma	8	0,163 us	-	3,333 us	-			
33					1,667 us	-			
34		-	-	-					
35									
36									
37	Vermelho: não atingiu o requisito								
38	Amarelo: parcela considerável da latência requerida (>25%)								

* Estimativas feitas considerando um relógio de sistema de 80 MHz

- Cálculo de vazão não se aplica

Vermelho: não atingiu o requisito

Amarelo: parcela considerável da latência requerida (>25%)

Tabela 18: Performance do Decodificador RLL

MCS.ID	Codificação	bits na entrada	Latência*	Relógio óptico suportado*	latência requerida (RIFS)	Relógio óptico requerido		
PHY I								
0	Manchester	2	0,175 us	10,7 MHz	200 us	200 kHz		
1								
2								
3		8	0,550 us	14,2 MHz				
4								
5	4B6B	6	0,363 us	16 MHz	100 us	400 kHz		
6								
7		12	0,675 us	17,5 MHz				
8								
PHY II								
16	4B6B	12	0,675 us	17,5 MHz	10,6 us	3,75 MHz		
17					5,3 us	7,5 MHz		
18					2,667 us	15 MHz		
19		10	0,575 us	17,0 MHz	1,333 us	30 MHz		
20					0,667 us	60 MHz		
21					0,333 us	120 MHz		
22								
23	8B10B	10	0,575 us	17,0 MHz				
24								
25								
26								
27								
28								
29								
PHY III								
32	Nenhuma	8	0,475 us	-	3,333 us	-		
33					1,667 us	-		
34								
35		8	0,475 us	-				
36								
37								
38								

* Estimativas feitas considerando um relógio de sistema de 80 MHz

- Cálculo de vazão não se aplica

Vermelho: não atingiu o requisito

Amarelo: parcela considerável da latência requerida (>25%)

Anexo

Anexo A: Modos de Operação dos Diferentes Tipos de Camada PHY

Tabela 19: Modos de operação da **PHY I** da IEEE 802.15.7 (p. 213)

Modulation	RLL code	Optical clock rate	FEC		Data rate
			Outer code (RS)	Inner code (CC)	
OOK	Manchester	200 kHz	(15,7)	1/4	11.67 kb/s
			(15,11)	1/3	24.44 kb/s
			(15,11)	2/3	48.89 kb/s
			(15,11)	none	73.3 kb/s
			none	none	100 kb/s
			(15,2)	none	35.56 kb/s
VPPM	4B6B	400 kHz	(15,4)	none	71.11 kb/s
			(15,7)	none	124.4 kb/s
			none	none	266.6 kb/s

Tabela 20: Modos de operação da **PHY II** da IEEE 802.15.7 (p. 213)

Modulation	RLL code	Optical clock rate	FEC	Data rate
VPPM	4B6B	3.75 MHz	RS(64,32)	1.25 Mb/s
			RS(160,128)	2 Mb/s
		7.5 MHz	RS(64,32)	2.5 Mb/s
			RS(160,128)	4 Mb/s
			none	5 Mb/s
OOK	8B10B	15 MHz	RS(64,32)	6 Mb/s
			RS(160,128)	9.6 Mb/s
		30 MHz	RS(64,32)	12 Mb/s
			RS(160,128)	19.2 Mb/s
		60 MHz	RS(64,32)	24 Mb/s
			RS(160,128)	38.4 Mb/s
		120 MHz	RS(64,32)	48 Mb/s
			RS(160,128)	76.8 Mb/s
			none	96 Mb/s

Tabela 21: Modos de operação da **PHY III** da IEEE 802.15.7 (p. 214)

Modulation	Optical clock rate	FEC	Data rate
4-CSK	12 MHz	RS(64,32)	12 Mb/s
8-CSK		RS(64,32)	18 Mb/s
4-CSK	24 MHz	RS(64,32)	24 Mb/s
8-CSK		RS(64,32)	36 Mb/s
16-CSK		RS(64,32)	48 Mb/s
8-CSK		none	72 Mb/s
16-CSK		none	96 Mb/s

Tabela 22: Relação entre o MCS_ID e o modo de operação, na IEEE 802.15.7 (p. 227)

	MCS indication	PHY	Data rate	Unit
0	000000	I	11.67	kb/s
1	000001		24.44	
2	000010		48.89	
3	000011		73.3	
4	000100		100	
5	000101		35.56	
6	000110		71.11	
7	000111		124.4	
8	001000		266.6	
16	010000	II	1.25	Mb/s
17	010001		2	
18	010010		2.5	
19	010011		4	
20	010100		5	
21	010101		6	
22	010110		9.6	
23	010111		12	
24	011000		19.2	
25	011001		24	
26	011010		38.4	
27	011011		48	
28	011100		76.8	
29	011101		96	
32	100000	III	12	Mb/s
33	100001		18	
34	100010		24	
35	100011		36	
36	100100		48	
37	100101		72	
38	100110		96	
	others		reserved	