UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Pedro Henrique Moura da Rosa

DESENVOLVIMENTO DE BLOCOS PARA PROCESSAMENTO DE DADOS UTILIZANDO PLATAFORMA OPEN-SOURCE DE SOFTWARE-DEFINED RADIO PARA CONTROLE DE SISTEMAS DE COMUNICAÇÃO POR LUZ VISIVEL

Santa Maria, RS
2023

Pedro Henrique Moura da Rosa

DESENVOLVIMENTO DE BLOCOS PARA PROCESSAMENTO DE DADOS UTILIZANDO PLATAFORMA OPEN-SOURCE DE SOFTWARE-DEFINED RADIO PARA CONTROLE DE SISTEMAS DE COMUNICAÇÃO POR LUZ VISIVEL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Engenharia Elétrica.** Defesa realizada por videoconferência.

Orientador: Carlos Henrique Barriquello, Dr. Eng.

Co-orientador: Vitalio Alfonso Reguera, Dr. Eng.

Santa Maria, RS

2023

Pedro Henrique Moura da Rosa

DESENVOLVIMENTO DE BLOCOS PARA PROCESSAMENTO DE DADOS UTILIZANDO PLATAFORMA OPEN-SOURCE DE SOFTWARE-DEFINED RADIO PARA CONTROLE DE SISTEMAS DE COMUNICAÇÃO POR LUZ VISIVEL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Engenharia Elétrica.**

Aprovado em 30 de março de 2023:

_____

Carlos Henrique Barriquello, Dr. Eng. (UFSM) - Videoconferência
(Presidente/Orientador)

_____

Felipe Loose, Dr. Eng. (UNIOVI) - Videoconferência

_____

Mateus Beck Rutzig, Dr. Eng. (UFSM) - Videoconferência

Santa Maria, RS
2023

DEDICATÓRIA

Á minha família.

AGRADECIMENTOS

Primeiramente agradeço a minha família por todo o amor e carinho durante toda minha jornada e que sempre me incentivaram a me dedicar nos estudos, e eu espero retribuir com resultados e alegria sobre os louros da vitória.

Agradeço ao grupo de pesquisa GEDRE, todos os professores e membros, que sempre me mostraram a mais genuína disposição a ensinar e aprender que eu já tive o prazer de presenciar em toda minha vida. Agradeço pelas amizades e ensinamentos. Apesar de um difícil período de pandemia em que era difícil se aproximar de novas pessoas, eu tenho que agradecer à algumas em especiais que marcaram sua passagem na minha vida:

- Rodrigo Fuchs Miranda, por todas as vezes que me ajudou e se mostrou presente e solícito nos períodos de dificuldades. O melhor veterano e amigo que eu poderia pedir;

- Igor Bertoncello Barboza, por cada convite descontraído para sair e interagir com o grupo de pesquisa que me fizeram sentir acolhido;

- Professor Lucas Teixeira, que desde os encontros virtuais do GEDRE durante a pandemia sempre me contagiou com seu entusiasmo e me ensinou muito sobre a tecnologia estudada neste trabalho;

- Professor Vitalio Alfonso Reguera, que desde o início da minha pesquisa me apoiou como um professor, mas também como um amigo. Que no momento em que me encontrava ao chão e sem expectativas, me estendeu a mão e me motivou novamente a dar tudo de mim. Por todo ensinamento, que hoje carrego comigo, meu mais sincero muito obrigado;

- Professor Carlos Henrique Barriquello, por toda a calma e paciência na reta final da orientação. Por dizer palavras de incentivo que me motivaram mais e mais a cumprir os objetivos. Sempre disposto a ajudar. Muito obrigado.

Á PPGEE e aos funcionários André e Luciana por sempre estarem presentes e darem suporte.

Aos membros da banca que aceitaram o convite e as contribuições ao meu trabalho.

Embora não tenha como citar todo mundo a quem sou grato, deixo meus agradecimentos a todos que de alguma forma me ajudaram no desenvolvimento deste trabalho ou que simplesmente desejaram o meu sucesso.

Este trabalho foi desenvolvido com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES/PROEX).

RESUMO

DESENVOLVIMENTO DE BLOCOS PARA PROCESSAMENTO DE DADOS UTILIZANDO PLATAFORMA OPEN-SOURCE DE SOFTWARE-DEFINED RADIO PARA CONTROLE DE SISTEMAS DE COMUNICAÇÃO POR LUZ VISIVEL

AUTOR: Pedro Henrique Moura da Rosa
ORIENTADOR: Carlos Henrique Barriquello
CO-ORIENTADOR: Vitalio Alfonso Reguera

Como um complemento aos meios de comunicação baseados em radiofrequência, a comunicação por luz visível ajuda a descongestionar o uso de dados no espectro eletromagnético. Este trabalho apresenta uma metodologia procedural para o desenvolvimento e implementação prática de um sistema que utiliza uma plataforma *Software-Defined Radio* para o controle digital das operações do sistema. A comprovação do funcionamento dos blocos para processamento de dados se deu através de testes experimentais realizados pela comunicação entre LED e fotodetector. Como base para a definição dos parâmetros da camada física do sistema foi utilizado o padrão IEEE 802.15.7-2018. O *software* utilizado para o processamento digital do sistema foi o GNU Radio que se caracteriza por trabalhar através da conexão entre blocos que desempenham rotinas únicas e individuais no sistema. Para o desenvolvimento experimental do sistema foi necessário complementar a biblioteca do GNU Radio com novos blocos que desempenhassem funções que não são nativas do programa. Por fim, a realização dos testes de comunicação foi feita com a frequência da portadora do sinal em 20MHz, Taxa de amostragem em 100kS/s e largura de banda em 300 kHz. A qualidade do sinal transmitido foi obtida ao comparar o número de amostras perdidas na recepção do sinal com as enviadas na proporção de 10^{-3}.

Palavras-chave: Comunicação por luz visível. Software-Defined Radio. Mensagem. Iluminação. LEDs. Metodologia. Programação. Transmissor e Receptor.

ABSTRACT

DEVELOPMENT OF DATA PROCESSING BLOCKS USING OPEN-SOURCE SOFTWARE-DEFINED RADIO PLATFORM FOR CONTROL OF VISIBLE LIGHT COMMUNICATION SYSTEMS

AUTHOR: Pedro Henrique Moura da Rosa
ADVISOR: Carlos Henrique Barriquello
CO-ADVISOR: Vitalio Alfonso Reguera

As a complement to radio frequency-based communication, the visible light communication helps to decongest the data usage in the electromagnetic spectrum. This work presents a procedural methodology for the development and practical implementation using a Software-Defined Radio platform for the digital control of the system's operations. The proof of the functioning of the blocks for data processing was done through experimental tests carried out by the communication between the LED and photodetector. As a basis for defining the parameters of the physical layer of the system, the IEEE 802.15.7-2018 standard was used. The software used for the digital processing of the system was the GNU Radio, which is characterized by working through the connection between blocks that perform unique and individual routines in the system. For the experimental development of the system, it was necessary to complement the GNU Radio library with new blocks that performed functions that are not native to the program. Therefore, the communication tests were made with the signal carrier frequency at 20MHz, sampling rate at 100kS/s and bandwidth at 300 kHz. The quality of the transmitted signal was obtained by comparing the number of samples lost in the reception of the signal with the ones being sent in a ratio of 10^{-3}.

Keywords: Visible light communication. Software-Defined Radio. Message. Lighting. LEDs. Methodology. Encoding. Transmitter and Receiver.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações gerais para esse trabalho, bem como a motivação para o seu desenvolvimento e os objetivos almejados. Por fim, a divisão do trabalho é apresentada.

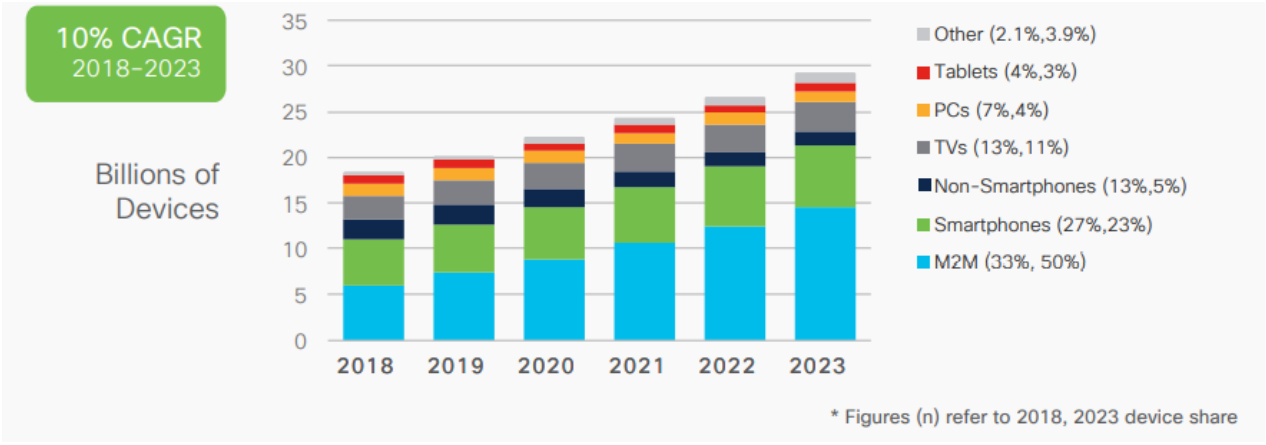
1.1 CONSIDERAÇÕES GERAIS

É possível dizer que o estudo da radiação eletromagnética, como se têm hoje, se deu pelas mãos do físico e matemático Clerk Maxwell ao unificar todos os fenômenos elétricos e magnéticos observados ao longo da história e propor as equações básicas que permitiam calcular o campo eletromagnético (VILLATE 2012). Mesmo que o estudo do magnetismo seja datado de muito antes, foram os resultados obtidos por Maxwell e posteriormente provados por Heinrich Rudolf Hertz, quando o mesmo demonstrou a existência da radiação eletromagnética ao elaborar aparelhos capazes de emitir e detectar ondas de rádio, que se criou a base fundamental dos estudos sobre o espectro eletromagnético que perduram até a atualidade.

A partir da descoberta de Hertz com a transmissão de informações por radiofrequência (RF), houve um grande interesse na pesquisa desta área, estabelecendo então padrões para a transmissão de sinais no espectro das ondas de rádio. Com uma grande amplitude de aplicações que o espectro eletromagnético é capaz de suprir, tem-se a possibilidade da expansão e difusão dos meios de comunicação, tornando os equipamentos responsáveis pela transmissão e recepção de sinais em objetos cotidianos.

O aumento exponencial de dispositivos de comunicação está causando consequentemente um aumento no tráfego de dados que ocupam a mesma zona do espectro eletromagnético. Naturalmente não haveriam problemas de congestionamento devido ao uso eficiente da frequência e o reuso espacial para este tráfego de informações, entretanto o atual espectro de ondas de RF está se mostrando escasso para o crescente aumento na demanda de dados (PATHAK, et al. 2015). Devido a isto, técnicas estão sendo discutidas para ajudarem a complementar a transmissão de dados via RF, onde conforme na Figura 1.1 fica evidenciado o aumento do uso de dispositivos eletrônicos que utilizam esta faixa do espectro para a transmissão de dados.

Figura 1.1 - Previsão de crescimento do uso de dispositivos eletrônicos no globo

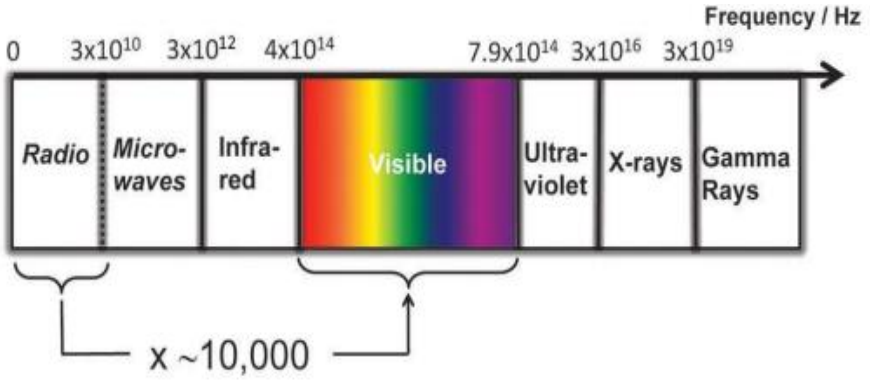


Fonte: adaptado de (CISCO SYSTEMS, INC. 2020).

Com a recente revolução no ramo da iluminação devido a técnica de iluminação de estado sólido (solid-state lighting) pelos Diodos Emissores de Luz (LEDs, do inglês - *Light-Emiting Diode*), novas tecnologias se tornaram possíveis. Com eficiência média que ultrapassam os 200 lumens/watt, e principalmente devido a capacidade de alterar entre diferentes níveis de intensidade em uma velocidade extremamente alta (não perceptíveis aos olhos humanos), os LEDs, diferentemente das tecnologias antigas, são capazes de serem usados para a comunicação aonde os dados são codificados e emitidos em diversas direções (PATHAK, et al. 2015).

Apresentando um espectro eletromagnético operando entre as frequências de 400 THz e 800 THz, os LEDs são dispositivos protagonistas da comunicação por luz visível (VLC – do inglês *Visible Light Communication*), ou seja, as ondas de radiação emitidas por este tipo sistema concentram-se nas faixas perceptíveis pelo olho humano, compreendidas entre 380 nm e 780 nm (CARVALHO, PEREIRA e CARVALHO 2015). Estes novos parâmetros ajudam a complementar a comunicação por RF, servindo como um escape para o congestionamento de dados que permeiam aquela zona do espectro eletromagnético. A Figura 1.2 mostra como o espectro eletromagnético é subdivido e acentua a largura de banda por luz visível a fim de comparação com a largura de RF.

Figura 1.2 - Largura de banda do espectro eletromagnético



Fonte: Adaptado de (HAAS 2013).

1.2 MOTIVAÇÃO

Os sistemas de VLC são referidos por muitos autores como a nova era da transmissão de dados, solucionando problemas criados pela utilização de RF bem como aumentando a taxa de transmissão a níveis superiores (KHAN 2017), possibilitando a comunicação em frequências muito superiores às utilizadas atualmente (ELGALA, MESLEH e HAAS 2009) (ELGALA, MESLEH e HAAS 2011). O primeiro a solucionar, é a capacidade. As ondas de RF estão atualmente trabalhando a toda carga no caso de sistemas de transmissão sem fio comerciais e não comerciais, tornando-os assim mais escassos e caros. Com uma previsão do avanço nos serviços multimídias e o aumento do consumo de dados surge a necessidade de um sistema complementar que possa suprir esta largura de banda adicional necessária. Neste âmbito o VLC é capaz de aumentar em até 10.000 vezes a largura do espectro utilizável sem interferência nenhuma com outros equipamentos (KHALIFEH, et al. 2018).

Em segundo, tem-se que outro problema da RF atual é a eficiência. As Estações Rádio Base (ERB) são conhecidas por serem uma das partes mais ineficientes na comunicação wireless pela grande quantidade de potência requerida para operarem (KHALIFEH, et al. 2018). Com a VLC atuando como complemento para os sistemas de RF, parte do problema de energia é minimizado pelo desafogamento do espectro eletromagnético, visto que a VLC conta com uma alta eficiência além do reaproveitamento da iluminação já existente como meio de comunicação. Nesta perspectiva o VLC serve como solução para aplicações tanto indoor quanto outdoor oferecendo a cobertura e a largura de banda necessária para funcionarem.

Nesse contexto, o desenvolvimento de novas formas, práticas, de utilizar o VLC como alternativa para a radiofrequência, se mostram necessárias. Como um componente fundamental no processamento de dados, os transceptores têm papel importante ao captar e analisar sinais. Ao

implementar o controle do VLC a partir do *Software-Defined Radio* (SDR), tem-se uma gama de equipamentos existentes capazes de prestarem os serviços requeridos para aplicações simples e também para situações que exijam um nível mais elaborado de processos. O valor do SDR se dá principalmente na agilidade fornecida por ele na prototipagem de novos experimentos permitindo a redução de trabalho durante a fase de testes (HUSSAIN, UGURDAG e UYSAL 2015). Além disto, o SDR é um tipo de transceptor muito flexível composto por uma camada *front-end* para envio e recepção de sinais de rádio e uma *back-end* que fornece suporte para o processamento digital de dados sendo assim ideal para sistemas experimentais.

1.3 OBJETIVOS

Este trabalho tem como objetivo principal desenvolver novos blocos para processamento de dados dentro da biblioteca do SDR para utilização em experimentos de VLC, seguindo o estabelecido no padrão IEEE 802.15.7-2018. Além disto, apresentar de maneira didática as funcionalidades do SDR para servir como apoio às novas pesquisas na área que utilizem a mesma ferramenta deste trabalho.

Como objetivo secundário, é desenvolvido um sistema físico para a transmissão e recepção de dados via luz visível a partir de LEDs e fotodiodos, com interesse na transmissão e recuperação dos dados enviados a fim de verificar a integridade da comunicação estabelecida entre transmissor (Tx) e receptor (Rx).

1.4 ESTRUTURA DO TRABALHO

O capítulo 2 tem como foco o apresentar o software utilizado para o desenvolvimento do trabalho. Esse capítulo introduz detalhes quando a arquitetura do programa que controla o SDR, apresentando suas ferramentas e recursos especiais, interações internas entre o próprio sistema como também externas com o usuário, e também modos de adicionar novos recursos à plataforma. Além disto, neste capítulo será mostrado como o SDR desempenha funções de interesse às aplicações VLC e como configura-los de acordo com a IEEE 802.15.7-2018.

O capítulo 3 apresenta o sistema VLC, trazendo informações sobre os LEDs e seus princípios de funcionamento bem como os conceitos necessários para a transmissão de dados via luz. Além disto, este capítulo contempla todo o trajeto da comunicação desde a criação da mensagem no TX até

a decodificação da mesma no RX, apontando os diferentes tipos de técnicas para o tratamento do sinal.

No capítulo 4 são apresentados os novos blocos desenvolvidos nesse trabalho para processamento de dados e suas funcionalidades no VLC. Após, na perspectiva de comunicação digital, os resultados simulados são apresentados a fim de validar o funcionamento destes novos blocos, bem como de alguns outros nativos do próprio *software* que foram utilizados para comporem a transmissão e recepção do sinal.

O capítulo 5 apresenta o sistema físico desenvolvido que é regido pelos blocos utilizados no capítulo 4 a partir do SDR. A mensagem enviada pelo TX pode ser configurada pelo software de um usuário e recebida pelo software de outro, variando de acordo com o número de pessoas conectadas ao SDR.

Por fim, o capítulo 6 apresenta as conclusões desse trabalho e sugestões de trabalhos futuros.

2 GNU RADIO & SOFTWARE DEFINED RADIO

Este capítulo tem como objetivo apresentar as ferramentas existentes no software que controla o SDR e é responsável por definir e manipular os dados utilizados para a realização da comunicação.

2.1 SOFTWARE DEFINED-RADIO

Um sistema SDR consiste num arquétipo de comunicação de rádio que usa software para modular e demodular sinais de rádio. O SDR é capaz de processar uma quantidade massiva de dados a partir de um computador de uso geral ou em micro-computadores reconfiguráveis geralmente compostos por uma placa única de processamento (RASPBERRY PI FOUNDATION 2022) (GARG 2007). Neste projeto, o SDR será utilizado de maneira não convencional, onde ao invés de processar informações para o controle de ondas de rádio, tem-se o intuito de controlar diretamente a luz emitida pelos LEDs.

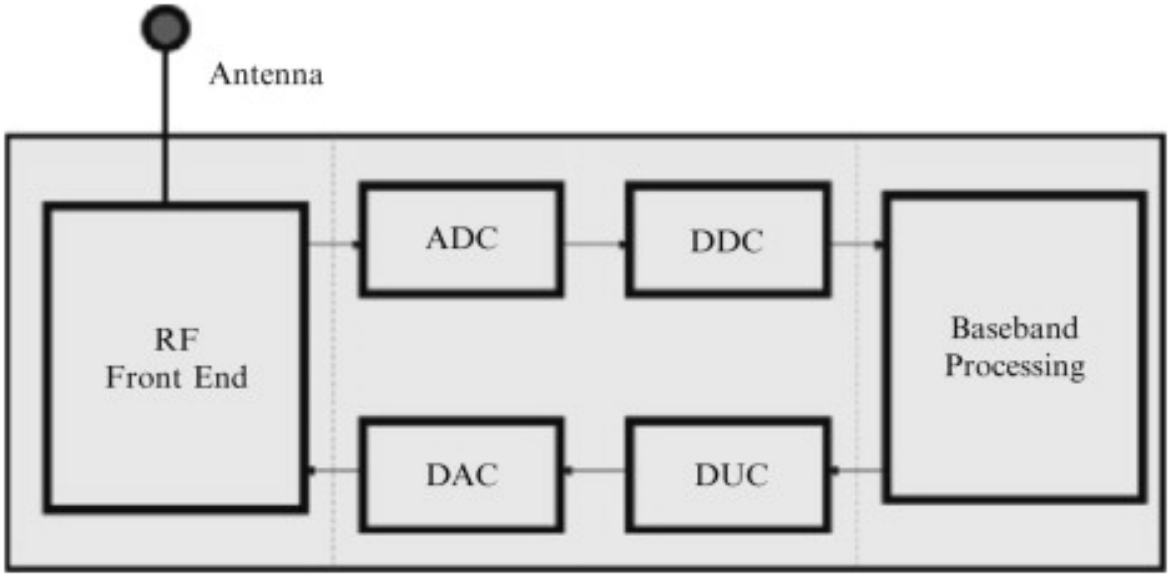
Também chamado de SD-VLC, a fim de facilitar as pesquisas e experimentações na área da comunicação por luz visível (CHE, KUBOKI e KATO 2017), e sendo comumente utilizado na indústria por fornecer agilidade na prototipagem (HUSSAIN, UGURDAG e UYSAL 2015) o SDR é um sistema para controle de sinal prático, que faz digitalmente o tratamento dos dados ao invés de utilizar circuitos eletrônicos. A partir de conversores analógicos digitais e um computador com processador e memória, o SDR recebe e envia informações, permitindo então analisar as amostras de dados através de programação lógica.

Consistindo de uma camada RF *front-end* super-heteródino, onde há a conversão dos sinais de RF para uma frequência intermediária fixa fazendo a conversão analógica/digital (A/D) e digital/analógica (D/A) (GARG 2007), Figura 2.1, o SDR através de drivers específicos responsáveis pela ligação do equipamento com a plataforma onde o usuário trabalha, é capaz de realizar a conexão com micro-computadores, sensores, equipamentos de áudio e vídeo, entre outras ferramentas que atribuem ao SDR toda sua versatilidade. A seguir estão algumas vantagens que o SDR pode apresentar (GARG 2007):

- O SDR pode ser reconfigurado quantas vezes for necessário, ou seja, é um dispositivo de comunicação universal capaz de ser adaptado de acordo com a necessidade do ambiente. Pode ser um telefone celular em um minuto, um *gadget* de Wi-Fi no próximo ou se necessário um receptor GPS.

- Os SDR podem transmitir e receber vários canais ao mesmo tempo.
- Devido a sua aplicabilidade digital, o SDR apresenta custos mais baratos e uma maior facilidade de implementação quando comparado com sua contraparte totalmente analógica (NASSAR 2002).
- Sistema com tamanho reduzido devido à substituição das funcionalidades analógicas pelas digitais.

Figura 2.1 - Princípio de operação do SDR



Fonte: adaptado de (WYGLINSKI, NEKOVEE e HOU 2010).

Existem diversos modelos de SDR presentes no mercado com os mais variados desempenhos dependendo da aplicação. Estes dispositivos funcionam através de seu respectivo programa ou de aplicativos que tenham suporte para o aparelho, devendo, portanto, ser realizada uma análise prévia sobre os requerimentos do projeto a ser implementado.

Neste trabalho, o SDR utilizado é o USRP (do inglês – *Universal Software Radio Peripheral*) que apresenta características como a conexão simultânea de mais de uma antena para transmitir e receber sinais usando MIMO (do inglês – *Multiple Input Multiple Output*) e a possibilidade de conexão de placas filhas adicionais para ampliação das funcionalidades. O software escolhido para controle do USRP foi o GNU Radio por se tratar de uma plataforma "*open-source"*, isto é, de código aberto ao público para que sejam realizados estudos e modificações de acordo com a vontade do projetista. O GNU Radio apresenta em sua documentação uma lista com todos os SDR comercialmente disponíveis que ele tem suporte e atribui a cada um deles suas especificações técnicas (GNU RADIO PROJECT 2022).

A tabela 1 abaixo exibe alguns dos SDR, apresentados na documentação do GNU Radio, que possuem drives para a portabilidade do equipamento com a plataforma (GNU RADIO PROJECT 2022).

Tabela 2.1 - Características de SDRs compatíveis com o GNU Radio

X	XTRX CS	XTRX Pro	USRP B2x0	bladeRF	bladeRF Micro 2.0	LimeSDR	LimeSDR Mini	Red Pitaya SDRlab122-16	RTL-SDR R820T2	RTL-SDR E4000	ADALM-Pluto	New Horizons	Hack RF One
Alcance de Frequência	30 MHz - 3.7 GHz	30 MHz - 3.7 GHz	70 MHz - 6 GHz	300 MHz - 3.8 GHz	47 Mhz - 6 Ghz	100 kHz - 3.8 GHz	10 MHz - 3.5 GHz	300 kHz - 60 MHz (550 MHz)	22 MHz - 2.2 GHz	65 MHz - 2300 MHz, gap@1100 MHz	325 MHz - 3800 MHz	70 MHz - 6 GHz	1 MHz - 6000 MHz
Duplex	Full MIMO	Full MIMO	Full MIMO	Full SISO	Full MIMO	Full MIMO	Full SISO	Full 2x2 MIMO	RX only	Rx only	Full SISO	Full MIMO	SISO Half Duplex
Resolução AD/DA	12-bit	12-bit	12-bit	12-bit	12-bit	12-bit	12-bit	16-bit	8-bit	8-bit	12-bit	12-bit	8-bit
Largura de Banda MAX	120 MHz	120 MSPS SISO / 90 MSPS MIMO	56 MHz	28 MHz	56 Mhz	61.44 MHz	30.72 MHz	550 MHz	3.2 Mhz		20 MHz *Limitado pelo USB 2.0 e software até ~4Mhz	56 MHz / CH	
Canais	2	2	1 (2 para B210)	1	2	2	1	2 RX + 2 Tx	1 Rx	1 Rx	1	2	1 Half Duplex
Potência Transmissível	0 a 10dBm	0 a 10dBm	10dBm+	6dBm	8dBm	0 a 10dBm	0 a 10dBm	±0.5V / +4 dBm			7 dBm	9.7	0 - 15 dBm
Transceptor	LMS7002M	LMS7002M	AD9364 or AD9361	LMS6002M	AD9361	LMS7002M	LMS7002M	LTC2185 + AD9767	R820T2	E4000	AD9363	AD9361	MAX 2837/Max5864
Matriz de Portas Programáveis	Xilinx Artix7 35T	Xilinx Artix7 50T	Xilinx XC6SLX75	Altera 40KLE/115KLE	Intel Cyclone V	Altera 40KLE	Altera MAX 10	Xilinx Zynq 7020	Sem	Sem	Xilinx Zynq 7000	Zync-7020	XC2C64A-7VQ100C CPLD
Sensor de Temperatura	Sim	Sim	Não	Não	Sim	Sim	Não	Sim	Não	Não			
Frequência de Estabilidade	±0.5	±0.1	±2 ppm	±1 ppm	±2.5 ppm	±2.5 ppm	±2.5 ppm	até ±12.00	±0.5-25 ppm	±0.5-25 ppm	± 25 ppm	± 15 ppm	
Sincronização GPS	Integrada	Integrada	Avulsa (+$636)	Sem	Sem	Sem	Sem	Sem	Sem	Sem	Sem	Opcional	Sem
Portas de Entrada	PCIe	PCIe	USB 3	USB 3	USB 3	USB 3	USB 3	Gbit Ethernet + opcional WiFi	USB 2	USB 2	USB 2.0 OTG	USB 2.0+ETH	USB 2.0 HS
CPU					Cypress FX3	CY3014	-	RTL2832U	RTL2832U	Dual A9,667MHz,		LPC4320	
Dimensões	30 × 51 mm	30 × 51 mm	97 x 155 mm	87 x 131 mm	63 x 102 mm	100 x 60 mm	69 x 31.4 mm	110 x 67 mm	40 x 60 mm typical	40 x 60 mm typical	117 x 79 mm	75mm*102mm	
Recursos Extras	GPIO, GPS, Cartão SIM	GPIO, GPS, Cartão SIM	GPIO	GPIO		GPIO	GPIO	16GPIO, Cartão SD, LinuxOS, USB	SMA Opcional	SMA Opcional		LCD Opcional, GPS	GPIO, RTC
Sincronização Clock	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Opcinal	Não	Não		Com conector	Sim
Bias T	Não	Não	Não	Sim	Opcional	Não	Não		Opcional	Opcional			Não
Preço - Típico US$	$260	$490	$686 - $1,119	$415	$480-$720	$299	$139	700$	$10 - $40	$10 - $40	$249	$642	$300

Fonte: adaptado de (GNU RADIO PROJECT 2022).

2.2 GNU RADIO

Para este trabalho, o software adotado para trabalhar foi o GNU Radio (GNU RADIO 2022). Por contar com uma arquitetura baseada em blocos de controle codificáveis, o GNU Radio é um software que permite ao usuário interagir com uma gama de equipamentos analógicos e passar

comandos para os mesmos que são definidos através de linhas de códigos carregadas em cada bloco do sistema. Sendo uma ferramenta para desenvolvimento de software gratuita e de código aberto, o GNU Radio provém uma vasta gama de blocos de processamento de dados para a implementação de experimentos de RF.

Uma das vertentes de utilização do software mais comum consiste em se aproveitar do ambiente virtual para reproduzir situações diversas para a obtenção de resultados simulados, partindo de custos reduzidos ou até nulos. Além disso o GNU Radio apresenta uma documentação muito completa quanto a sua instalação, manuseio e obtenção de resultados, o que torna ainda mais atrativo.

A documentação fornece um passo a passo para a instalação do GNU Radio dependo do sistema operacional (SO) do computador utilizado. Para este trabalho foi utilizado o SO Linux Ubuntu 22.10, o qual conta com o aplicativo da GNU Radio v3.10.1.1 diretamente disponível para download na sua bolsa de aplicativos, sendo muito mais conveniente a instalação quando comparada ao concorrente Windows e macOS.

2.2.1 Método de instalação GNU Radio

A seguir seguem os métodos de instalação do GNU Radio para os três tipos de sistemas operacionais mais comuns no mercado:

2.2.1.1 Instalação Windows

Sendo o SO com maior complexidade para instalação, isto porque os processos são todos baseados em ferramentas e rotinas do baseadas em sistemas Linux. Além disto pelo GNU Radio ser considerado como uma extensão do Python, há problemas potenciais no Windows se este estiver rodando simultaneamente bibliotecas diferentes usadas para o GNU Radio e Python (GNU RADIO 2022).

Na documentação são sugeridos dois tipos de processos. O primeiro consiste na construção manual de todos os arquivos necessários que o GNU Radio utiliza para fazer seu trabalho como CMake, Doxygen, Perl, entre outros. O segundo método e recomendado é utilizar um tipo de software pré-configurado que faça o download e instalação de todos estes arquivos automaticamente. Curiosamente o segundo método consiste majoritariamente de aplicativos que simulem o ambiente do sistema operacional Linux.

Realizar a instalação do GNU Radio utilizando o Subsistema do Windows para Linux (WSL – do inglês Windows Subsystem for Linux) é uma das opções mais recomendadas visto que é possível

instalar um SO Ubuntu 20.04 (ou mais recente) diretamente da Loja da Microsoft. Assim, a partir do terminal do Ubuntu é só instalar como no Linux descrito abaixo.

2.2.1.2 Instalação Linux Ubuntu

Devido ao GNU Radio ser uma aplicação desenvolvida em Linux a instalação neste SO se revela muito simples podendo até mesmo ser encontrada uma versão do GNU Radio diretamente na loja de aplicativos do sistema.

Uma versão alternativa mais robusta e personalizável é feita ao abrir o Terminal (ferramenta pré-instalada no SO) deve-se digitar os seguintes comandos:

```
sudo apt-get install gnuradio (1)
```

Assim a versão do GNU Radio 3.10.1.1 será automaticamente instalada. Para atualizar o software para versões mais recentes, digitam-se os comandos:

```
sudo add-apt-repository ppa:gnuradio/gnuradio-releases (2)
```

```
sudo apt-get update (3)
```

```
sudo apt-get install gnuradio python3-packaging (4)
```

2.2.1.3 Instalação macOS

A principal recomendação de instalação para macOS é através do software MacPorts que tem que ser instalado independentemente. Para isto, o usuário deve instalar o *Xcode* e *Xcode Command Line Tools* e então instalar o MacPorts para o modelo do seu macOS.

Com MacPorts devidamente instalado, basta seguir procedimentos similares ao do item 2.2.1.2 digitando as seguintes linhas de comando:

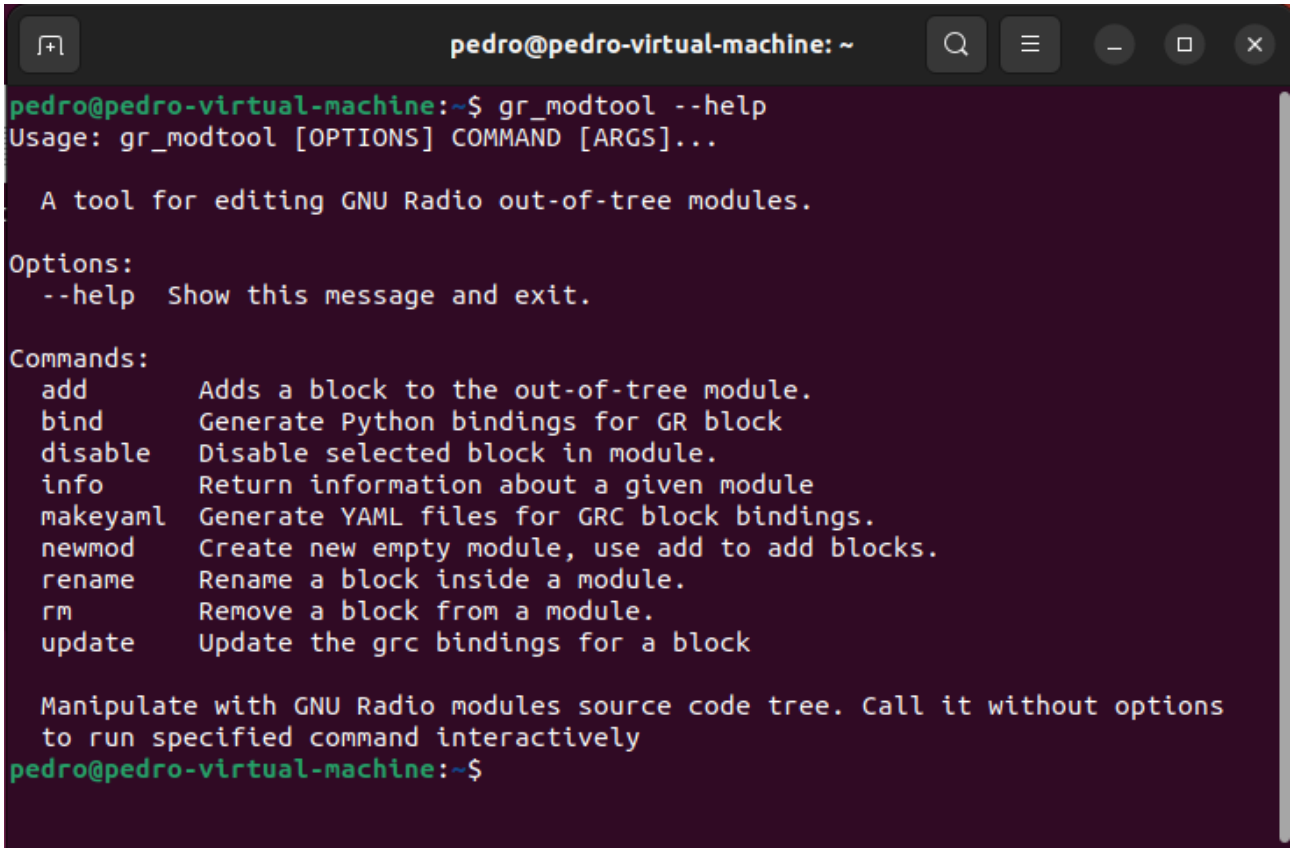
```
sudo port install gnuradio (5)
```

```
sudo port install gnuradio-devel (6)
```

2.2.2 Desenvolvimento de Novos Blocos para Processamento de Dados

Além de contar com uma biblioteca nativa contendo blocos interessantes para a utilização no VLC e que têm suas características definidas na Standard IEEE 802.15.7-2018 (IEEE STANDARDS ASSOCIATION 2018), o GNU Radio permite a criação de novos blocos através de programação em Python ou C++ com a ferramenta ‘gr_modtool’, Figura 2.2. Deste modo é possível ampliar ainda mais a biblioteca do software permitindo que mais pessoas tenham acesso a rotinas de códigos que antes não estavam programadas.

Figura 2.2 - Recurso para a adição de novos blocos de codificação



Fonte: adaptado de (GNU RADIO 2022).

A ferramenta ‘gr_modtool’ é desenvolvida para criar os principais arquivos que o GNU Radio usa para interpretar os códigos e repassar os comandos para o SDR. Primeiramente deve-se criar um novo modulo, que será usado para adicionar uma nova categoria, com novos itens, à lista de componentes existentes.

```
sudo gr_modtool newmod ‘nome da pasta’ (7)
```

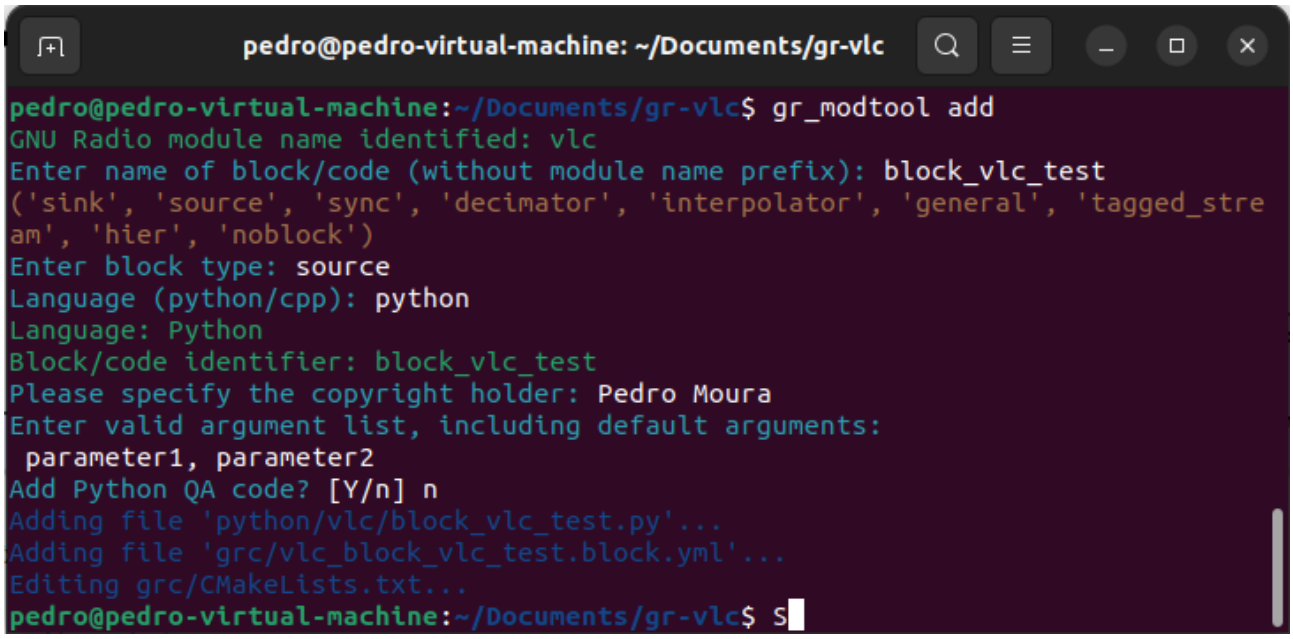
Criada a nova categoria, basta procurar no terminal a pasta criada antes de dar prosseguimento a adição de blocos. Acessando este novo endereço é necessário utilizar o comando ‘add’ para que sejam gerados os dois arquivos editáveis do sistema onde vão as linhas de código para a realização das tarefas que o usuário deseja.

```
sudo gr_modtool add ‘nome do bloco’ (8)
```

Este será o comando principal na hora de estabelecer os parâmetros iniciais do código elaborado. Ao pressionar Enter, será exibido no Terminal que tipo de bloco do GNU Radio o novo se identifica a fim de fornecer um cabeçalho correspondente com a função do bloco. Mesmo que seja definido um tipo de bloco que não corresponda de fato com a função em que está sendo criado é possível altera-la mais tarde, visto que esta funcionalidade apenas altera entre os *templates* existentes

que o GNU Radio usa como referência para a interpretação dos códigos, mas que podem ser digitados manualmente também na hora da codificação e parcialmente modificados. Posteriormente serão definidos outros parâmetros, como qual linguagem de programação será utilizada para a codificação dos blocos, a adição prévia de algumas variáveis que serão utilizadas no código e montagem de um cabeçalho para *copyright*, como pode ser visto na Figura 2.3.

Figura 2.3 - Parâmetros iniciais para criação de novos blocos



Fonte: adaptado de (GNU RADIO 2022).

A escolha da linguagem de programação que vai ser usada para a codificação dos blocos é essencial para a elaboração de atividades específicas que requerem algum tipo de lógica programável, isto é, ao mudar a linguagem de programação torna-se possível acessar algumas ferramentas e atalhos lógicos que só são acessáveis através desta linguagem. Os experimentos realizados no GNU Radio são escritos principalmente usando a linguagem de programação Python, enquanto o caminho de processamento de sinal fornecido é implementado em C++ usando extensões de ponto flutuante do processador, quando disponíveis. Assim, o desenvolvedor é capaz de implementar sistemas de rádio de alto rendimento em tempo real em um ambiente de desenvolvimento de aplicativos rápido e simples de usar (GNU RADIO PROJECT 2022).

Concluído estes passos, o módulo de referência e o bloco de codificação que o usuário tem acesso para modificar já estão criados no caminho selecionado restando agora vinculá-los à base de dados do *software* GNU Radio. Através dos comandos mostrados abaixo no Terminal, será criado o

diretório (arquivo com formato Pasta) ao qual o GNU Radio irá acessar para poder instalar e '*linkar*' todos os arquivos gerados no processo de desenvolvimento de novos blocos.

```
mkdir build (9)
```

```
cd build (10)
```

```
cmake ../ (11)
```

```
sudo make (12)
```

```
sudo make install (13)
```

```
sudo ldconfig (14)
```

Antes de realizar o comando (9) é necessário acessar no Terminal o caminho do novo bloco criado em (8), isto é, acessar a pasta em que se deseja criar o diretório. Feito a criação do diretório '*build*' deve-se acessá-lo por (10) para que os comandos (11), (12) e (13) referentes ao *software* CMake funcionem da maneira correta. CMake é um sistema multi-plataforma usado para gerenciar e automatizar os processos de *'build'*, ou seja, controlar o processo de conversão de arquivos código-fonte para um formato padrão aceito pelo software que está sendo usado para o trabalho. Por fim em (14) são armazenadas em *cache* (lugar para acessar os arquivos temporários de modo mais rápido, quando necessário) as mais recentes bibliotecas compartilhadas encontradas nos diretórios especificados na linha de comando (KERRISK 2021).

Como pode ser observado na Figura 2.3, mostrada anteriormente, na criação dos novos blocos de codificação são definidos alguns *templates* dependendo da função que se espera que ele desempenhe, no caso da Figura 2.3 foi definido como *source*. Cada um desses *templates* são codificados de uma maneira específica para que desempenhem a função desejada, mas antes de compreender quais as regras para o desenvolvimento destes algoritmos, é necessário entender como o GNU Radio interpreta e processa as informações que são usadas para a construção deles.

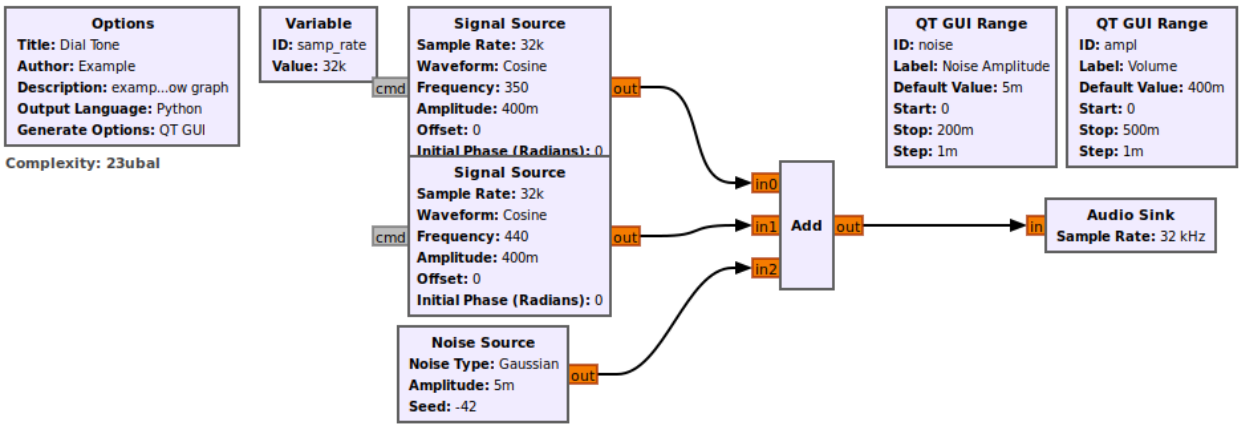
Através do uso de blocos nativos do GNU Radio Companion (GRC), que é a interface gráfica do GNU Radio, tem-se na Figura 2.4 um exemplo simples para ilustrar o funcionamento do sistema de '*dial_tone*'. Este exemplo é encontrado no arquivo *dial_tone*.py e constrói um aplicativo GNU Radio muito simples que combina duas ondas senoidais para criar um tom de discagem. Ele chama uma função para construir o fluxograma que lida com a geração das fontes de sinal. A função *build_graph* que existe internamente no bloco QT GUI Range, configura a taxa de amostragem, que usaremos para definir a taxa de dissipação de áudio, bem como a amplitude, que usamos para dimensionar o sinal para controlar o volume da saída (GNU RADIO PROJECT 2022).

Este exemplo usa três blocos de processamento de sinal. Os dois primeiros são blocos que geram ondas senoidais nas frequências de 350 e 440 Hz. Em seguida, criamos a conexão com o

sistema de alto-falantes usando *gr::audio::sink*, que recebe a taxa de amostragem que usará para produzir o sinal de saída. O bloco de dissipação de áudio também pode receber um segundo parâmetro para definir o nome do dispositivo de saída, que usamos se houver um conflito de recursos ou se estiver usando uma taxa de amostragem que o hardware não suporta naturalmente.

Em seguida, pegamos os três blocos que construímos e conectamos o fluxograma. O fluxograma conecta as fontes aos coletores por meio de outros blocos de processamento de sinal. Aqui, estamos conectando diretamente duas fontes a um único coletor.

Figura 2.4 - Exemplo de Dial-Tone usando o GRC



Fonte: adaptado de (GNU RADIO 2022).

Ao executar este exemplo, é produzido um sinal que simula o tom usado para chamadas telefônicas diretamente na placa de som do dispositivo que está sendo utilizado no momento.

Como pode ser observado na Figura 2.4, o GRC funciona através do fluxo de dados entre os blocos conectados no sistema. A maioria dos blocos consistem de entradas e saídas que são responsáveis por receberem/enviarem dados aos demais blocos depois de realizarem um processo interno definido na codificação usada na sua construção.

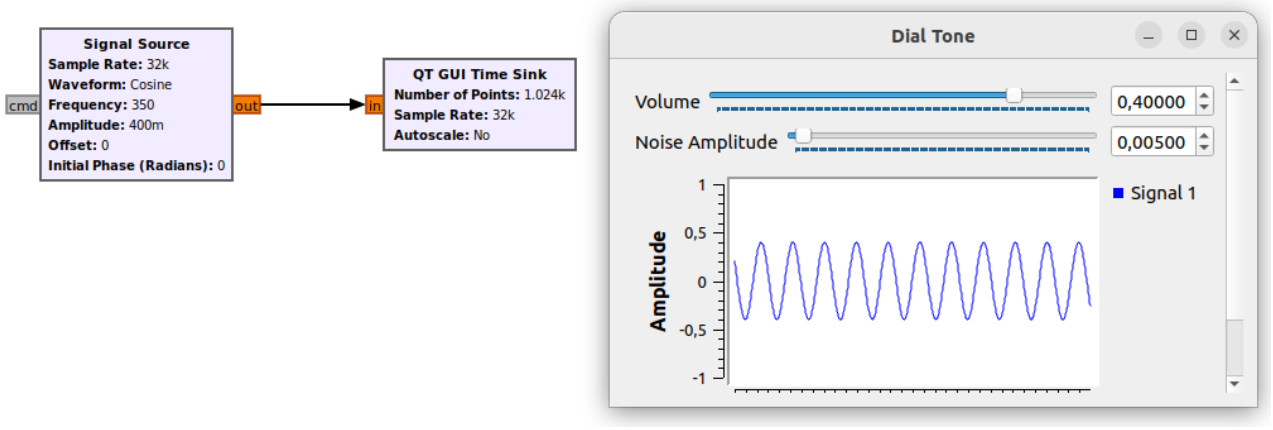
Entre os principais tipos de blocos tem-se os *sync*, *decimator*, *interpolator* e *basic*.

2.2.2.1 Sync

Os blocos desta categoria se caracterizam por apresentarem um fluxo constante de dados entre a entrada e saída do bloco. Podendo conter 0 ou diversas entradas e 0 ou diversas saídas, os blocos do tipo *sync* são os mais comuns de se encontrarem na biblioteca do GNU Radio, e isto se deve ao fato de que a maioria das aplicações consistem em realizar uma operação de 1:1 entre entrada e saída, onde para cada dado de entrada será realizada uma operação e resultará em um valor de saída.

É nesta categoria de blocos que se encontram as *sources* e *sinks* responsáveis por gerarem sinais e por lerem os mesmos e definirem como apresentar estes valores ao usuário. Como exemplo deste tipo de blocos, tem-se a maioria das fontes de onda usadas para a simulação de sinais elétricos bem como os blocos utilizados como osciloscópio para as experimentações virtuais do sistema. Na Figura 2.5 pode ser visto o formato de onda resultante de uma das fontes existentes no exemplo da Figura 2.4, observado pelo bloco *sink*.

Figura 2.5 – Utilização de blocos *Sync*



Fonte: adaptado de (GNU RADIO 2022).

Para a criação de novos blocos em python, pode-se utilizar o seguinte cabeçalho (GNU RADIO PROJECT 2021):

```
class my_sync_block(gr.sync_block):
    def __init__(self):
        gr.sync_block.__init__(self,
            name = "my sync block",
            in_sig = [numpy.float32, numpy.float32],
            out_sig = [numpy.float32],
        )
    def work(self, input_items, output_items):
        output_items[0][:] = input_items[0] + input_items[1]
        return len(output_items[0])
```

Os *input_items* e *output_items* são listas de listas, ou seja, um tipo de variável em que dentro de cada índice da lista principal está contida uma outra lista com valores independentes. O *input_items* contém um vetor de amostras de entrada, e o *output_items* é um vetor para cada fluxo de saída.

Algumas observações:

• O comprimento de todos os vetores de entrada e todos os vetores de saída é idêntico

• *in_sig* = *None* transformaria isso em um bloco de origem

• *out_sig = None* iria transformar isso em um bloco de leitura, *sink*.

• Ao contrário do C++, onde usamos a classe gr::io_signature, aqui podemos apenas criar uma lista Python dos tamanhos de dados de E/S usando tipos de dados numpy, por exemplo: numpy.int8, numpy.int16, numpy.float32

2.2.2.2 Decimator e Interpolator

Os blocos de codificação do tipo *Decimator* e *Interpolator* são semelhantes aos *Sync* por apresentarem uma taxa de dados fixa, isto é, para x valores de entrada, tem-se y valores na saída independentemente das funções especificadas em código.

Decimação é o processo de redução da taxa de amostragem. Isso geralmente implica na filtragem passa-baixa de um sinal e, em seguida, no descarte de algumas de suas amostras. Nas comunicações e em aplicativos típicos do GNU Radio, a decimação é comumente usada para reduzir a taxa de amostragem de um sinal super-amostrado, a fim de reduzir a complexidade computacional do sistema.

A razão para usar o *Decimator* é reduzir a taxa de amostragem na saída de um sistema para facilitar o recebimento do sinal em outros sistemas. Além disso, este procedimento ajuda a reduzir o custo de processamento: o cálculo e/ou memória necessária para implementar um sistema digital geralmente é proporcional à taxa de amostragem, portanto, o uso de uma taxa de amostragem mais baixa geralmente resulta em uma implementação mais barata.

O cabeçalho utilizado para elaborar este tipo de bloco é o seguinte (GNU RADIO PROJECT 2021):

```
class my_decim_block(gr.decim_block):
    def __init__(self, decim_rate):
        gr.decim_block.__init__(self,
            name="my block",
            in_sig=[numpy.float32],
            out_sig=[numpy.float32],
            decim = decim_rate)
        self.set_relative_rate(1.0/decim_rate)
        self.decimation = decim_rate
```

Observa-se, em comparação com o item 2.2.2.1, que existem duas novos parâmetros usados nesta codificação.

• *set_relative_rate* é responsável por configurar a relação entrada/saída do bloco do GNU Radio.

• *decimation* é a variável que terá seu conteúdo atribuído pelo usuário para definir a taxa de decimação desejada para o sistema.

Em contrapartida os blocos do tipo *Interpolation* funcionam de modo inversamente proporcional aos de decimação, isto é, eles têm por objetivo aumentar a taxa de amostragem na saída do bloco para que outro sistema operando em uma taxa de amostragem mais alta possa receber o sinal.

O *layout* da codificação é muito semelhante ao de decimação (GNU RADIO PROJECT 2021):

An example interpolation block in Python:

```
class my_interp_block(gr.interp_block):
    def __init__(self, interpolation):
        gr.interp_block.__init__(self,
            name="my block",
            in_sig=[numpy.float32],
            out_sig=[numpy.float32])
        self.set_relative_rate(interpolation)
```

Quando o usuário define o parâmetro de interpolação, automaticamente a função *set_relative_rate* ajusta os vetores de entrada e saída do bloco para desempenhar a função desejada.

2.2.2.3 Basic

Os blocos do tipo *basic* se caracterizam por terem uma relação de entrada e saída totalmente manipulável, não fornecendo nenhuma relação entre o número de itens de entrada e o número de itens de saída. Todos os outros blocos são apenas simplificações do bloco *basic* (GNU RADIO PROJECT 2021).

Com uma codificação um pouco mais complexa que os anteriores, o bloco *basic* tem o seguinte *template* para codificação em Python:

```
import numpy as np
from gnuradio import gr

class my_basic_adder_block(gr.basic_block):
    def __init__(self):
        gr.basic_block.__init__(self,
            name="another_adder_block",
            in_sig=[np.float32, np.float32],
            out_sig=[np.float32])

    def general_work(self, input_items, output_items):
        #buffer references
        in0 = input_items[0][:len(output_items[0])]
        in1 = input_items[1][:len(output_items[0])]
        out = output_items[0]

        #process data
        out[:] = in0 + in1
```

```
        #consume the inputs
        self.consume(0, len(in0)) #consume port 0 input
        self.consume(1, len(in1)) #consume port 1 input


        #return produced
        return len(out)
```

Como pode ser observado, este tipo de bloco utiliza a função *general_work* e não a *work*, como é o caso dos itens anteriores. Isto ocorre porque este tipo de bloco necessita controlar os *buffers* de entrada e saída manualmente, enquanto a função *work* faz isto de modo automático segundo a taxa de amostras esperada. Segue alguns pontos acerca do funcionamento deste tipo de bloco:

• *ninput_items* é o vetor que descreve o tamanho dos *buffers* de entrada.

• O número de itens nos *buffers* é estabelecido a partir do número de itens na saída através de *noutput_items*.

• A função *general_work* necessita consumir manualmente os dados guardados nos *buffers* de entrada para que possa se manter em operação contínua.

2.3 IEEE 802.15.7-2018

A IEEE 802.15.7 foi elaborada como um padrão para definir algumas técnicas base para a construção do VLC a curtas distancias. O padrão subdivide a camada PHY em seis tipos, sendo estas a PHY-I, PHY-II, PHY-III, PHY-IV, PHY-V e PHY-VI. A camada física é responsável pelas seguintes tarefas (IEEE STANDARDS ASSOCIATION 2018):

• Ativação/Desativação do transceptor OWC (do inglês – *Optical Wireless Communication*).

• Seleção do canal utilizado.

• Transmissão de dados e recepção.

• Correção de erros.

• Sincronização.

A seguir, tem-se a caracterização de cada tipo de camada PHY (IEEE STANDARDS ASSOCIATION 2018):

a) PHY-I: Este tipo de camada se destina a aplicações externas que necessitem de baixa taxa de dados. Utilizando o tipo de modulação OOK (do inglês – *On-Off Keying*) e modulação VPPM (do inglês – *Variable Pulse Position Modulation*) esta camada PHY atinge taxas de dados entre dezenas a centenas de kbps.

b) PHY-II: Este tipo PHY é usado em aplicações necessárias dentro das edificações (*indoor*), e se caracteriza por ter uma taxa de dados moderada. Este modo também utiliza as

modulações OOK e VPPM para a transmissão dos dados e é capaz de atingir algumas dezenas de Mbps.

c) PHY-III: Esta camada funciona através da modulação CSK (do inglês – *Color-Shift Keying*) a partir de múltiplos emissores e receptores, sendo capaz de atingir taxas de dados similares às da camada PHY-II na casa das dezenas de Mbps.
d) PHY-IV: Nesta camada PHY, a modulação utilizada é a UFSOOK (do inglês – *Undersampled Frequency Shift ON-OFF*) que se assemelha ao deslocamento da frequência por chaveamento (ROBERTS 2013) e atinge uma transmissão de até 22 kbps. PHY destina-se ao uso com fontes de luz discretas com taxas de dados de até 22 kbps usando
e) PHY-V: Através de fontes de luz de superfície difusa, este tipo de camada PHY transmite dados até 5,71 kbps, funcionando com até quatro tipos de modulações diferentes.
f) PHY-VI: Destinada para o uso com *displays* de vídeo, esta camada pode ser utilizada com até seis tipos de modulações diferentes e atinge uma taxa de dados na faixa dos kbps.

Para este trabalho o enfoque será majoritariamente na camada PHY-I, com alguns testes na camada PHY-II.

A camada PHY-I é constituída por dois tipos de modulações sendo estas a OOK e a VPPM (do inglês *Variable Pulse Position Modulation*) subdividindo-se assim em duas categorias. Ao utilizar códigos corretores de erro em série, como o *Reed-Solomon* (RS) e o *Convolutional Coding* (CC) para cada uma das modulações, tem-se a separação da camada PHY-I em 8 modos de operação que variam de acordo com as configurações dos corretores de erro e da modulação selecionada.

Além dos fatores citados, a camada PHY-I suporta também dois tipos de códigos RLL sendo estes o Manchester e o 4B6B. Estes códigos são responsáveis por manter o controle DC, recuperar o clock e mitigar os efeitos do *flicker*.

A Figura 2.6 exibida a seguir revela as configurações estabelecidas para os ensaios com a camada PHY-I:

Figura 2.6 - Modos de Operação da camada PHY-I

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

Fonte: adaptado de (IEEE STANDARDS ASSOCIATION 2018).

Nesta topologia, a codificação é usada com uma combinação do CC externo e um RS interno para funcionarem a partir do sistema de correção de erro por codificação concatenada. Deste modo a saída do RS é preenchida com zeros para a criação de um cabeçalho de entrelaçamento, onde estes são descartados logo após e enviados ao CC.

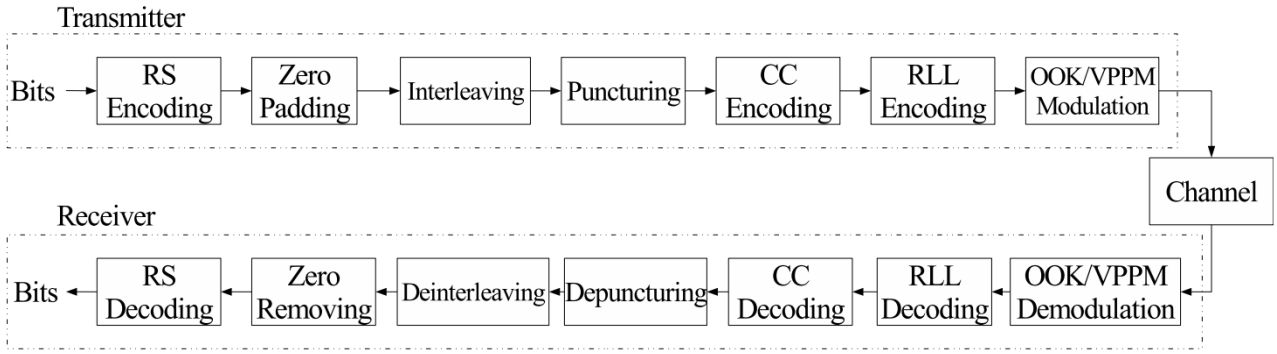
Na Figura 2.7 é exibido o fluxograma para o processamento do sinal e o envio dos dados ao transmissor. Os bits entram primeiramente no codificador RS aonde são rearranjados em uma nova estrutura. Na camada PHY-I o código RS apresenta quatro configurações diferentes, sendo estas RS (15,11), RS (15,7), RS (15,4) e RS (15,2) com o *Galois Field* de tamanho 2^4, isto é, com 16 elementos (GEISEL 1990). O bloco de intercalação é colocado entre os blocos corretores de erro a fim de aumentar a performance do sistema. Com um tamanho “n” fixo, mas profundidade “D” variável em função do tamanho do frame enviado, o intercalador faz um cruzamento entre o fluxo de bits aliviando os *burst errors*, isto é, diminuindo a ocorrência de erros entre o primeiro e último bit de um sinal. Junto dele é implementado um bloco de *puncturing* para minimizar os erros durante o preenchimento do vetor de bits (IEEE STANDARDS ASSOCIATION 2018). Após, o CC gera os símbolos de paridade com a taxa de 1/3 e tamanho máximo K = 7. As taxas de 1/4 e 2/3 são obtidas pela combinação do CC com o bloco de *puncturing*. Por fim, tem-se os códigos RLL, sendo estes o Manchester ou o 4B6B, onde para a modulação OOK se usa somente o Manchester e para a VPPM somente o 4B6B. Enquanto o código Manchester duplica o fluxo de bits ao expandir cada bit para um

símbolo 2-bit, o código 4B6B aumenta o tamanho do vetor em 1,5 vezes, ao expandir o conjunto de 4-bits para 6-bits.

Assim, o fluxo de bits finalmente chega no bloco modulador, seja este OOK ou VPPM. Para a modulação OOK os bits são representados através do nível ‘alto’ e ‘baixo’, ‘ligado’ e ‘desligado’, ‘1’ e ‘0’. Em contrapartida a modulação VPPM define os dados através da borda de ‘subida’ ou de ‘descida’ do seu sinal, isto é, na transição entre um nível e outro (HUSSAIN, UGURDAG e UYSAL 2015). Após isto, o sinal vai para o canal onde sofre alterações devido a ruídos do meio e é captado pelo receptor, responsável de enviar os dados que chegam ao SDR para que seja feita a demodulação do sinal e o tratamento inverso para cada um dos blocos utilizados na modulação.

A partir do esquemático apresentado é possível processar o sinal VLC via SDR.

Figura 2.7 - Fluxograma do processamento de sinal da camada PHY-I



Fonte: Adaptado de (TURAN, et al. 2016)

2.4 CONCLUSÕES PARCIAIS

Esse capítulo abordou os principais aspectos que compõe os sistemas SDR bem como os modelos existentes no mercado. A partir disto, foi descrito o modelo de SDR aplicado aos experimentos deste trabalho e o aplicativo utilizado para realizar o controle geral das atividades do sistema desde sua instalação até a criação dos novos blocos de codificação para a operação de atividades não nativas do sistema. Após, foram descritas as topologias de camada PHY definidas no *standard* utilizado como referência neste trabalho e então a pesquisa foi mais aprofundada na camada PHY utilizada para os ensaios.

3 VISIBLE LIGHT COMMUNICATION

Este capítulo apresentará uma breve revisão sobre aspectos relacionados ao VLC. O sistema VLC consiste em um aglomerado de dispositivos eletrônicos que quando atuam em conjuntos são capazes de modular a frequência do espectro eletromagnético emitido pelas lâmpadas LEDs. Neste tópico serão tratados sobre estes equipamentos, dando ênfase na função especifica de cada um e métodos de implementação.

3.1 LIGHT-EMITING DIODE

Se caracterizando por ser um aparelho semicondutor com uma junção PN dividindo as cargas positivas (sem elétrons) das cargas negativas (com excesso de elétrons) a criação desta tecnologia revolucionou os sistemas de iluminação (SCHUBERT 2003), atuando como um diodo, como o próprio nome diz, que permite a passagem de corrente elétrica em um determinado sentido e a bloqueia em outro conforme for polarizado, os LEDs podem ser utilizados como transmissores em sistemas de comunicação onde é possível realizar a modulação da luz gerada devido ao fenômeno da eletroluminescência criado a partir de estímulos elétricos nos LEDs, alternando entre níveis altos e baixos, em uma velocidade alta de modo que não seja perceptível aos olhos humanos. Além disso, o aumento significativo na vida útil da iluminação LED, variando de 25.000 a 50.000 horas, comparada com a fluorescente compacta (10.000 horas) tornou muito mais atrativo a sua utilização. Devido a estes benefícios e a adoção constante pela tecnologia, espera-se que 75% de toda iluminação vai ser baseada em LEDs até o ano de 2030 (U. S. DEPARTMENT OF ENERGY 2019)

Para a constituição de um sistema de comunicação por luz visível, pode-se considerar um ou mais conjuntos de LEDs sendo definidos de acordo com sua aplicação. Com o desenvolvimento dos μLEDs – compostos por Nitreto de Gálio (GaN – do inglês *Gallium Nitride*), as pesquisas em VLC os têm como componentes fundamentais para a transferência de sinais com alta velocidade (LIU, LIN, et al. 2018). Tais características despertam o interesse par a comunicação de dados a altas frequências, devido a largura de banda elevada fornecida por estes dispositivos quando comparadas com os LEDs comuns utilizados unicamente para a iluminação.

No geral, a alta velocidade de transmissão permitida pelos μLEDs se deve às suas propriedades como tamanho pequeno, permitindo o somatório de mais fontes de iluminação no mesmo espaço físico, alta injeção de corrente elétrica, largura de banda com modulação elevada, alta sensibilidade do receptor em relação a eles e otimização das antenas óticas (LIU, TIAN, et al. 2017).

Segundo (MCKENDRY, et al. 2010) ao utilizar os micro LEDs é possível atingir uma taxa de transmissão dedos de 1 Gbps ao custo de uma baixa eficiência luminosa. Estes equipamentos mesmo sendo superiores aos LEDs convencionais para transmissão não são necessariamente obrigatórios para a comunicação VLC, visto que a mesma necessita unicamente da modulação da intensidade luminosa obtida pelas fontes de luz de estado sólido podendo assim serem feitas a partir das iluminações com lâmpadas LED convencionais.

Os elementos geradores de luz podem ser categorizados pelo tipo de fonte geradora de cor, sendo estas:

- LEDs azuis com uma camada de fósforo;
- Ou o conjunto RGB.

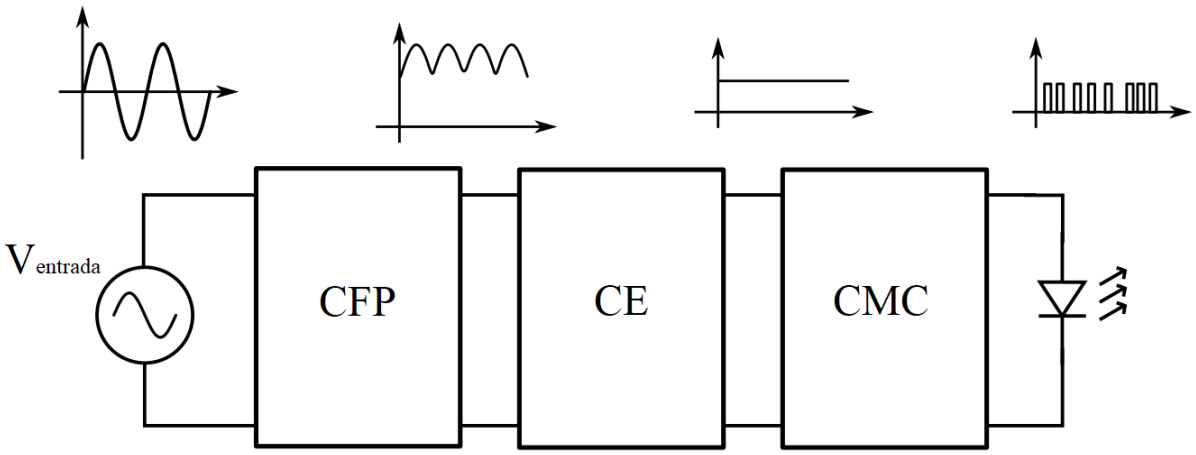
Para o primeiro tipo, as luzes emitidas pelos LEDs azuis passam por uma camada de fósforo. Quando a luz azul atravessa essa camada, a combinação das cores produz a luz branca que podem variar de acordo com a quantidade de fósforo depositado sobre o invólucro (PATHAK, et al. 2015). O segundo conjunto, RGB, tem a luz branca emitida pela combinação das cores gerados por LEDs vermelhos, verdes e azuis, onde são usados no mínimo três LEDs simultaneamente. Devido ao aumento na quantidade de LEDs este segundo tipo de geração de luz branca é mais caro que o conjunto com a camada de fósforo, entretanto em termos de qualidade na comunicação de dados a camada de fósforo limita a velocidade com que o LED pode alterar de intensidade para poucos MHz, tornando-se assim menos interessante para essas aplicações (PATHAK, et al. 2015).

3.2 CIRCUITO DE ACIONAMENTO

O circuito responsável pelo controle da corrente elétrica incidente no LED, alterando a intensidade da iluminação, é chamado de *driver*. Sempre que a luminária é utilizada para comunicação o circuito *driver* atua modulando a corrente elétrica de acordo com os dados que espera-se transmitir (PATHAK, et al. 2015).

O *driver* usado em VLC, Figura 3.1, pode ser dividido em três estágios de operação desde que começa a atuar até enviar o sinal modulado: Correção de Fator de Potência (CFP), Controle de Energia (CE), e Controle de Modulação para Comunicação (CMC) (TEIXEIRA, et al. 2019). No primeiro estágio, o *driver* retifica a entrada CA enquanto mantém os padrões de qualidade de energia ao evitar altos índices de distorção da corrente drenada da rede. No segundo são geradas grandezas CC, estáveis, para a alimentação da carga (LED). E por fim, no terceiro estágio, os valores de corrente e tensão são modulados de uma maneira pré-estabelecida a fim de transmitir dados.

Figura 3.1 – Estrutura convencional do driver para comunicações VLC



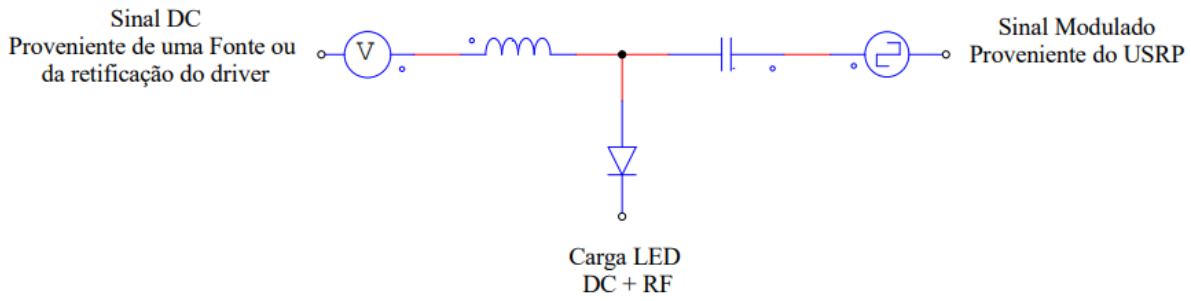
Fonte: Adaptado de (TEIXEIRA, et al. 2019)

Para unir as duas funcionalidades distintas do VLC, iluminação e comunicação, sem comprometer a qualidade do sinal enviado e tão pouco os índices de iluminação incidente no ambiente da utilização, busca-se obter um ponto ótimo para os mais variados casos a partir da análise sobre conversores lineares, chaveados e híbridos (LOOSE, et al. 2022), (Teixeira 2020), (CHEN e LOH 2020).

Neste trabalho, para os experimentos, utilizou-se uma fonte de tensão CC variável dispensando assim grande parte das funcionalidades do driver, ficando apenas com o terceiro estágio de operação. A escolha de utilizar uma fonte de tensão CC se deu devido ao propósito deste trabalho ser a criação de novos blocos de codificação capazes de enviar ordens específicas na saída do SDR para que seja realizada a modulação da Luz.

Assim sendo, para realizar o casamento entre o circuito de comunicação e o de potência, definiu-se o uso de um circuito *bias-t* (HUANG, et al. 2018) (CHI, et al. 2018). A *bias-t* consiste de um circuito com dois seguimentos que se encontram em um nó e derivam para o cátodo do LED. A Figura 3.2 apresenta o circuito base *bias-t*.

Figura 3.2 - Circuito *Bias-T*



Fonte: do Autor.

Existem diversos tipos de modulações possíveis para aplicações VLC, porém como definido no item 2.3, neste trabalho serão utilizadas apenas a modulação OOK e a VPPM que compõe o escopo da camada PHY-I (IEEE STANDARDS ASSOCIATION 2018). Com a alteração da forma de onda da corrente de entrada dos LEDs é possível fazer pequenas variações na intensidade luminosa emitida, de modo que ao seguirem determinadas regras, podem ser interpretadas. Tais variações ocorrem de acordo com o desejado pelo sistema, de modo que para o estado alto se tenha um valor alto, 1, de corrente e para o estado baixo se tenha um valor baixo, 0, de corrente.

Não necessariamente o nível baixo de luz emitida pela comunicação VLC tem que ser em um estado totalmente desligado, visto que dependendo das técnicas utilizadas na modulação é possível interpretar os sinais de nível baixo mesmo apresentando luminosidade. Segundo (DIMITROV e HAAS 2015) (HAAS e CHEN 2015) (ISLIM e HAAS 2016) três tipos de modulações digitais são citadas para VLC, sendo estas:

- *Single Carrier Modulation* (SCM);
- *Multi-Carrier Modulation* (MCM);
- *Color Domain Based Modulation* (CDBM);

A SCM é um interessante esquema de modulação para VLC devido a sua simplicidade e moderada velocidade de transmissão (RAMADHANI e MAHARDIKA 2018). Nesta categoria, incluem-se as modulações OOK, *Pulse Amplitude Modulation* (PAM) e *Pulse Position Modulation* (PPM). Conforme a taxa de dados aumenta, a SCM começa a sofrer distorção de sinal não-linear e interferências entre símbolos (ISI – do inglês *Inter-Symbol Interference*), tornando-se assim menos atrativa quando comparada com a MCM que é desenvolvida, justamente, para corrigir algumas das desvantagens existentes na SCM (IDRIS, et al. 2019). Nesta categoria, tem-se destaque para a técnica de multiplexação OFDM (do inglês – *Orthogonal Frequency Division Multiplexing*). Por fim, tem-se que para a CDBM um método utilizado e proposto pela IEEE 802.15.7 para dimerização e mitigação de *flicker* é o *Color Shift Keying* (CSK).

3.2.1 Single Carrier Modulation (SCM)

A utilização de SCM é conhecida por modular exclusivamente a corrente de entrada dos LEDs para a variação desejada do fluxo luminoso. A forma mais comum deste tipo de modulação é a OOK, também chamada de Variação de Amplitude por Chaveamento (ASK – do inglês *Amplitude Shift Keying*). A modulação OOK consiste no controle da amplitude da corrente I_{in} dos LEDs a partir dos níveis altos e baixos, 1 e 0, ligado e desligado. O formato dos pulsos para esta modulação consiste em códigos do tipo Return-to-Zero (RZ), e Non-Return-to-Zero (NRZ) (ELGANIMI 2013).

A modulação OOK é a mais utilizada dentro os sistemas existentes, principalmente pela sua resiliência à ruídos e interferências. Além disso apresenta um design simples de receptor e custos reduzidos quando comparada com modulações mais complexas (IJAZ, et al. 2010). De modo análogo à OOK, a PAM é um tipo de modulação que varia a amplitude em valores onde $0 < Ps < 1$, onde Ps é a intensidade de potência da amostra analisada.

Segundo (SADIQ IDRIS 2019) a PPM, que consiste em uma modulação realizada pela diferença entre o intervalo dos pulsos, se mostra mais eficiente que a OOK no quesito de potência requerida e além disso tem menos interferência a ruídos quando comparada com a PAM, tornando-se assim uma técnica mais atrativa para situações que envolvam custo benefício direto. Do mesmo modo, o sistema PWM (do inglês – *Pulse Width Modulation*) ao variar a largura dos pulsos emitidos ajuda na dimerização dos LEDs devido a seu controle de baixa intensidade.

Para contornar os pontos fracos de cada um desses tipos de modulação, é proposto em (SADIQ IDRIS 2019) (LEE e PARK 2011) a combinação das técnicas para compensarem umas às outras dando origem a diversas outras modulações.

3.2.2 Multi-Carrier Modulation (MCM)

Devido a necessidade constante de altas taxas de dados em redes de comunicação ocorre a necessidade de um sistema que consiga suprir esta alta demanda, principalmente em sistemas VLC que são conhecidos por terem uma ampla largura de banda e espectro eletromagnético. Deste modo, a utilização de SCM apresenta limites no quesito de usufruir o máximo da tecnologia VLC e para tal foram desenvolvidas técnicas que pudessem explorar esta área com tantas possibilidades.

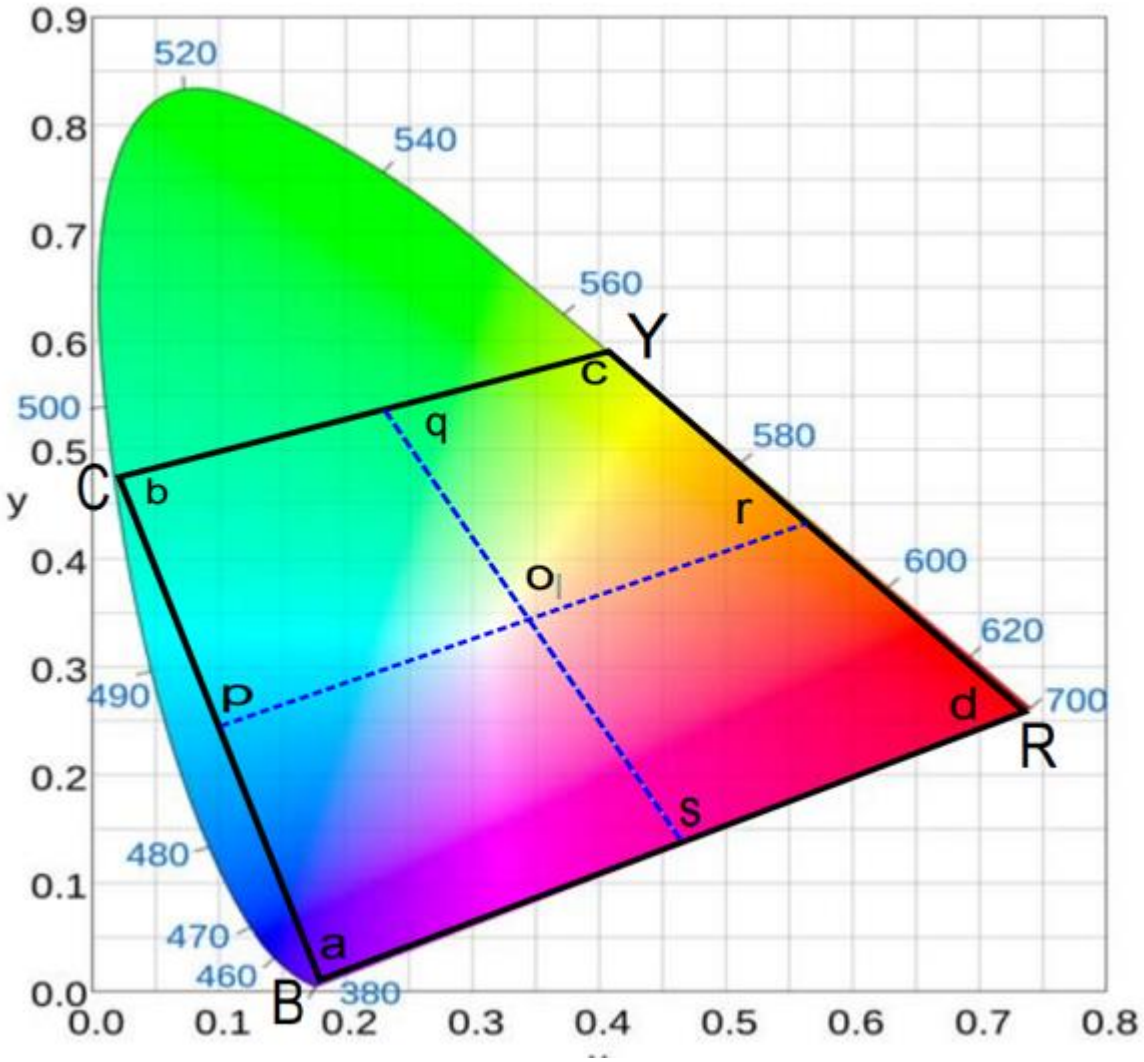
Desenvolvido para transmitir dados em subcanais, tipicamente ortogonais de propagação, o MCM torna-se relativamente imune a atenuação causada pelo meio, também chamado de "*fading*" (GOLDSMITH 2005). A ideia básica da MCM é dividir o fluxo de bits (*bitstream*) em vários outros sub-fluxos para então envia-los a diferentes sub-canais. A taxa de dados de cada um destes sub-canais é muito inferior a total, e do mesmo modo a largura de banda também. Essa modulação é usada em diversos sistemas sem fio, no entanto, não é uma técnica recente sendo que foi usada pela primeira vez em rádios militares no final dos anos 1950 e início de 1960 (GOLDSMITH 2005).

É interessante ressaltar que os sub-canais da MCM não necessitam serem vizinhos, ou estarem próximos para funcionarem a uma alta taxa de transmissão. Deste modo, não é necessário um grande bloco de espectro próximo. Além disso, a MCM é implementada digitalmente de forma discreta, também conhecida como OFDM que reduz a interferência dividindo o fluxo de dados em diferentes canais de banda estreita em diferentes frequências (ROHLING e GALDA 2004)

3.2.3 Color Shift Keying (CSK)

A *Color Shift Keying* é considerada como uma modulação que transmite um sinal através das intensidades da cor de luz emitida pelos LEDs vermelhos, verdes e azuis (RGB). Os sinais emitidos pela topologia CSK são mapeados através do triangulo proposto pela CIE 1931, no diagrama de cromaticidade, exibido na Figura 3.3. De modo geral, a intensidade de saída da modulação CSK é constante, mas a intensidade relativa entre as múltiplas cores utilizadas varia (SADIQ IDRIS 2019).

Figura 3.3 – Espaço operacional do CSK baseado no Diagrama de Cromaticidade (CIE 1931)



Fonte: adaptado de (SINGH, O'FARRELL e DAVID 2015)

Observa-se que em CSK, a intensidade da luz é modulada para diferentes cores, aonde as coordenadas serão compreendidas entre os pontos de Azul, Ciano, Amarelo e Vermelho (BRYC).

Apesar de apresentar um design mais complexo que os demais tipos de modulação, por necessitar exclusivamente de um transmissor e receptor que sejam compatíveis com RGB, a modulação CSK apresenta algumas vantagens principalmente quanto a evitar as frequências

causadoras do efeito *Flicker*, no qual a variação de intensidade luminosa pode acabar causando problemas de mal-estar e comprometer a saúde das pessoas (PATHAK, et al. 2015).

3.3 RECEPTOR VLC

Quando se trata de comunicação, tão importante quanto o transmissor é o equipamento responsável por captar a mensagem transmitida e decodifica-la de forma correta. O receptor no âmbito do VLC pode ser por diversos componentes, mas majoritariamente são fotodiodos do tipo PIN e fotodiodos do tipo Avalanche (APD – do inglês Avalanche PhotoDiode).

Os principais indicadores de qualidade do fotodiodo são: sensibilidade (S), responsividade (ρ), a eficiência quântica (η), corrente escura, capacitância, frequência de corte (fc), área do detector (Af) e custo. Na tabela 3.1, tem-se os indicadores citados para os fotodetectores PIN e ADP em três materiais diferentes sendo: Silício, Germânio, e Arseneto de Índio e Gálio.

Tabela 3.1 - Indicadores para escolha de Fotodiodos

Parâmetro	**Silício**		**Germânio**		**InGaAs**	
	PIN	**APD**	**PIN**	**APD**	**PIN**	**APD**
Comprimento de Onda [nm]	400-1000		800-1800		900-1700	
Pico [nm]	900	830	1550	1300	1550	1550
Responsividade [A/W]	0.6	77-130	0.65-0.7	3-28	0.75-0.97	
Eficiência Quântica [%]	65-90	77	50-55	55-75	60-70	60-70
Ganho	1	150-250	1	5-40	1	10-3
Tensão Bias [V]	45-100	220	6-10	20-35	5	30
Corrente Escura [nA]	1-10	0.1-1.0	50-500	10-500	1-20	1-5
Capacitância [pF]	1.2-3	1.3-2	2-5	2-5	0.5-2	0.5
Tempo de Subida [ns]	0.5-1	0.1-2	0.1-0.5	0.5-0.8	0.06-0.5	0.1-0.5

Fonte: adaptado de (MAPUNDA, et al. 2020)

Para este trabalho, foi selecionado o dispositivo PDA10A2 da ThorLabs® que além de apresentar o fotodiodo para recepção do sinal, também tem um amplificador embutido em seu circuito.

3.4 CONCLUSÕES PARCIAIS

Nesse capítulo foram abordados os principais aspectos construtivos que compõem um sistema VLC. A partir do sistema de transmissão onde os dados são mesclados com as grandezas de corrente e tensão para manterem o propósito de iluminação a partir do controle que o *driver* impõe sobre o sistema, este sinal é captado pelo sistema receptor onde são processados e encerram o trabalho de comunicação da mensagem enviada/recebida.

4 DESENVOLVIMENTO DE BLOCOS PROGRAMÁVEIS

Este capítulo apresenta as codificações elaboradas para o desenvolvimento de novos blocos de funções compatíveis com o estabelecido na IEEE 802.15.7-2018.

4.1 METODOLOGIA

Conforme definido no item 2.2.2 deste trabalho, a criação de novas funções para o SDR consiste exclusivamente por linhas de programação tanto em *Python* quanto em C++. De acordo com os *templates* apresentados anteriormente, decidiu-se criar blocos úteis ao VLC dentro do *software* do GNU Radio a fim de permitir a transmissão de sinais via luz visível seguindo os padrões estabelecidos na IEEE 802.15.7-2018 para a camada PHY-I.

Para ser estabelecido um sistema de comunicação por luz visível segundo os padrões da IEEE, tem-se a necessidade de que determinados blocos para a modulação do sinal sejam utilizados:

- Mensagem a ser enviada (gerador de bits aleatório ou coordenado);
- Codificador Reed-Solomon;
- Preenchimento de Zeros (*zero padding*);
- *Puncturing*;
- *Run-Length Limited* (RLL) Manchester ou 4B6B;
- Modulação OOK ou VPPM.

Para a recepção do sinal, deve-se realizar o caminho reverso a fim de tratar o sinal recebido para que se tenha como resultado a mensagem inicial enviada.

4.1.1 Geração da Mensagem

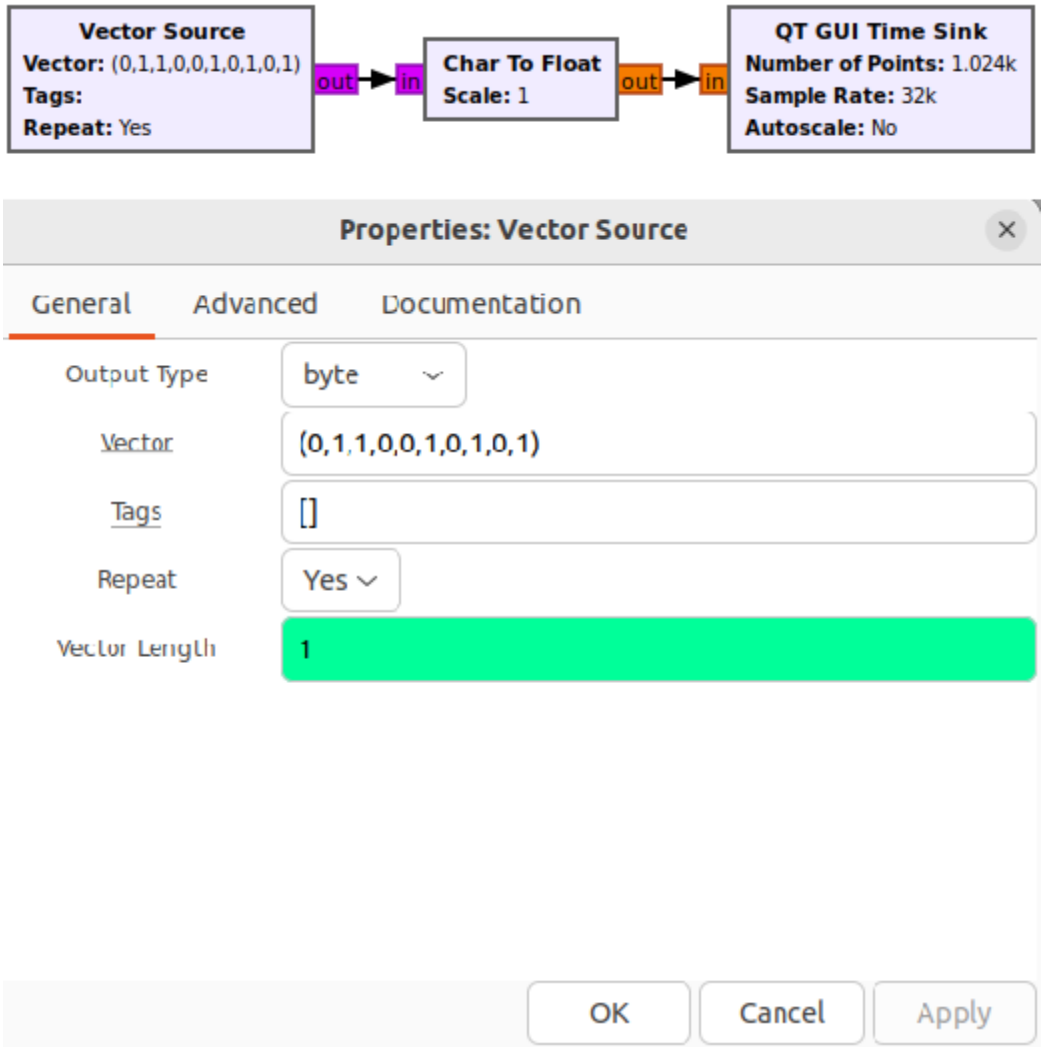
Para todos as simulações descritas a seguir, utilizou-se um bloco nativo do GNU Radio capaz de enviar dados no formato de bytes para os demais. O bloco *Vector Source* é um componente capaz de gerar amostras (*samples*) de acordo com o definido pelo usuário onde o fluxo de saída do bloco é uma sequência de bytes.

O tamanho, *Length*, da sequência de bytes que se deseja transmitir pode ser definido através dos chamados *objects_blocks* que são componentes que funcionam como variáveis que guardam dados. Além disto, para mensagens mais curtas enviadas repetidamente, como no caso deste trabalho, pode-se definir diretamente no próprio componente.

A Figura 4.1 abaixo exibe a sintaxe deste tipo de bloco para determinar os principais parâmetros como *vector* que é a mensagem enviada propriamente dita e *length* que definirá o tamanho

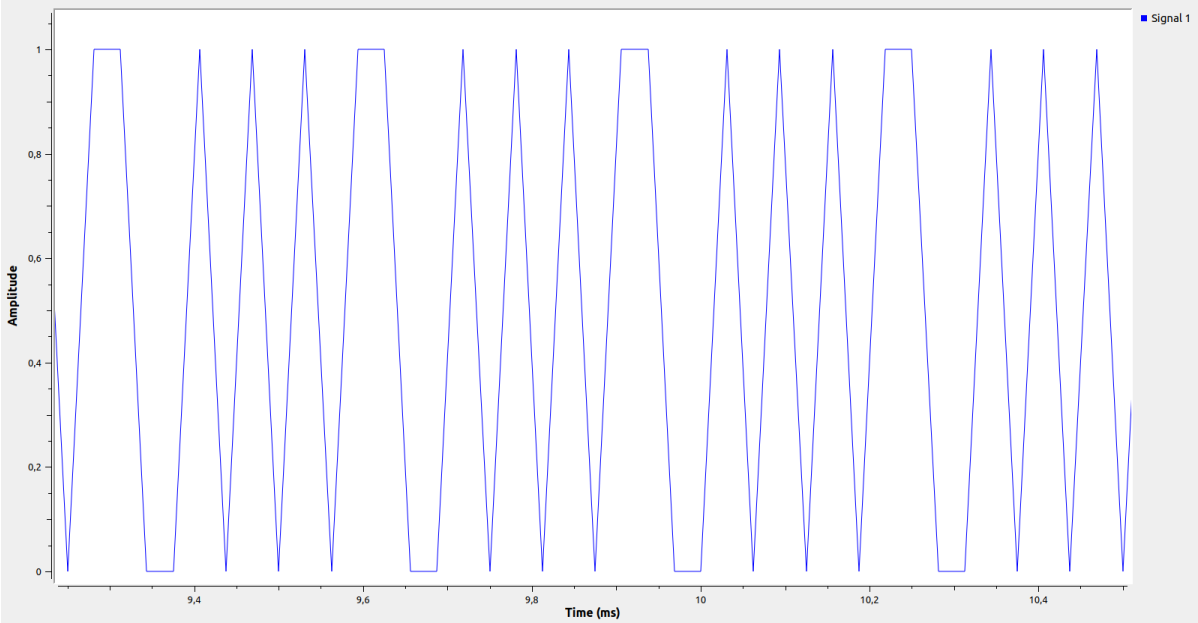
da mensagem na saída. Para valores de *length* diferentes de 1, tem-se a equação (15) que define o número de itens na saída do bloco. A Figura 4.2 evidencia o funcionamento deste bloco em função da mensagem definida.

Figura 4.1 - Conexão e Configuração Gerador de Bits



Fonte: do Autor.

Figura 4.2 - Forma de Onda da Vector Source



Fonte: do Autor.

Observa-se que para a análise dos dados simulados, utilizou-se o componente QT GUI Time Sink, que funciona como um osciloscópio digital para a análise das amostras. Como a geração de sinal está gerando bits entre 0 e 1, é indiferente o uso do bloco conversor para dados do tipo Char (-127 a +127) ou unsigned Char (0 a +255), visto que ambos compreendem este intervalo.

4.1.2 Bloco para Correção de Erros Reed-Solomon com *Zero Padding*

Um codificador Reed-Solomon realiza a leitura dos dados digitais dos blocos e adiciona bits redundantes extras. O decodificador Reed-Solomon processa todos os blocos e tenta corrigir erros e recuperar os dados originais em que o número e o tipo de erros a serem corrigidos se baseiam na característica do código Reed-Solomon (RHEE, et al. 2010).

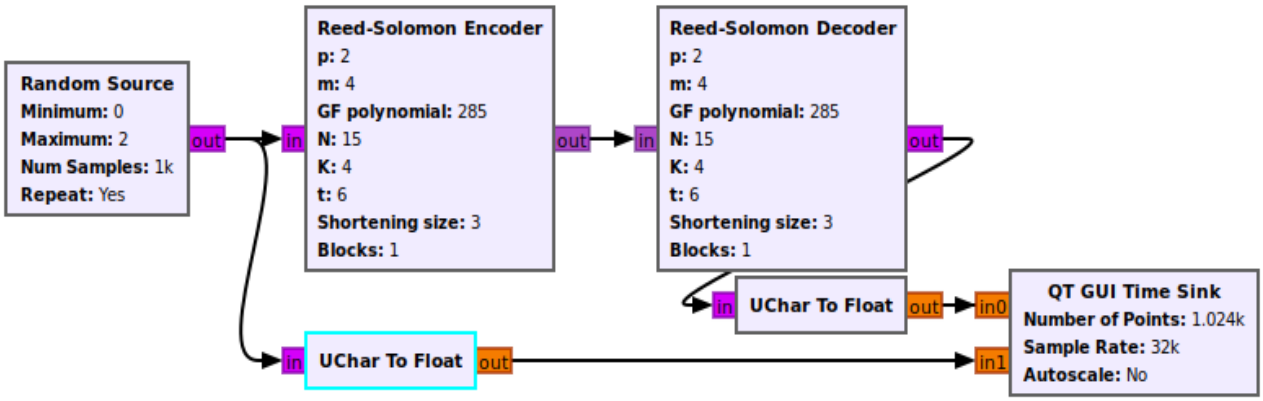
Os códigos Reed-Solomon foram inventados por Irving S. Reed e Gustave Solomonin 1960, e continuam a ter uma ampla gama de aplicações em comunicação e armazenamento digital. Com o intuito de reduzir a perda de dados na saída à probabilidade de um erro no restante dos dados decodificados é sempre menor do que quando o Reed-Solomon não é usado.

Baseados no chamado *Galois Field*, onde as operações aritméticas nos elementos do campo têm um resultado no campo em si é possível descobrir quando um bit esperado para tal posição não se encontra e então preenche-lo para a correção.

A Figura. 4.3 e 4.4 apresentam o funcionamento do bloco nativo para Reed-Solomon. Como pode ser observado em na Figura 4.3 uma fonte randômica está gerando 1000 amostras de ‘0s’ e ‘1s’

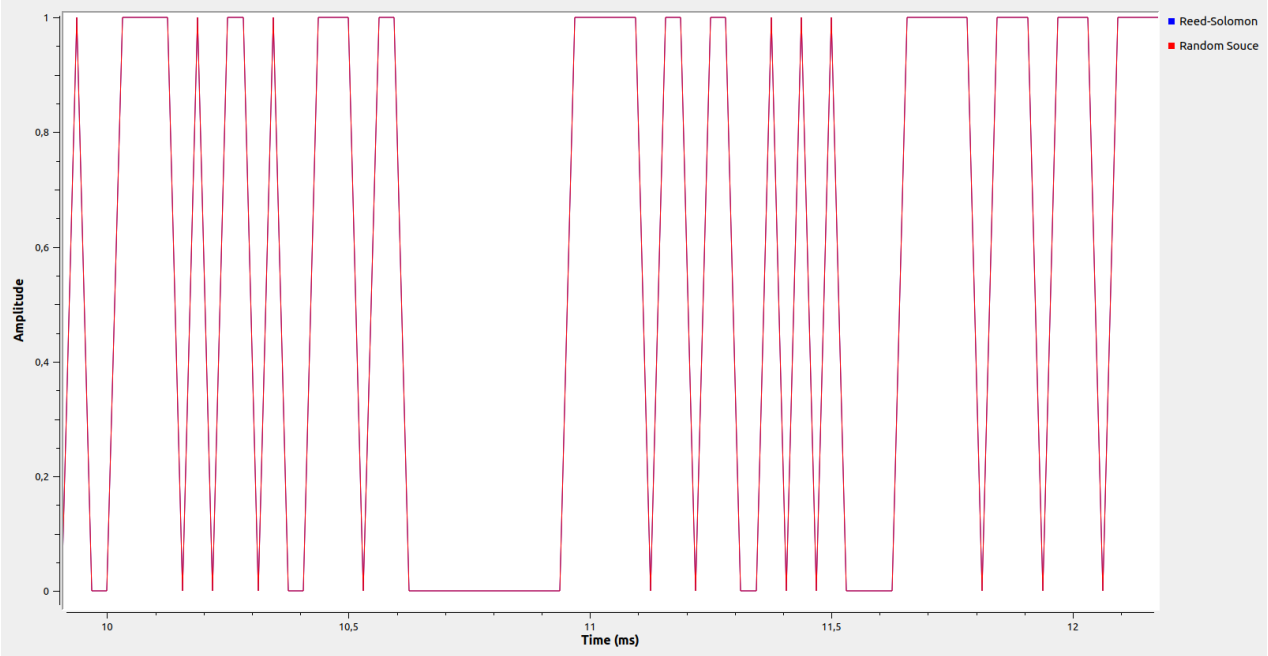
que são codificadas e decodificadas pelos blocos RS antes de serem medidas e comparadas com o *bitstream* original. Como visto na Figura4.4 as formas de onda se sobrepõem após a decodificação.

Figura 4.3 - Fluxograma blocos *Reed-Solomon*



Fonte: do Autor.

Figura 4.4 - Comparação entre *bitstream* original e *bitstream* pós processamento RS



Fonte: do Autor.

Os blocos de RS foram estruturados seguindo o padrão (IEEE STANDARDS ASSOCIATION 2018) com o Galois Field [GF(16)] (definido por 'p^m'), com campo composto pelo polinômio primo 0x11d (285), e na configuração RS(15,4). Além disto, preenchimento de zeros (*padding*) é definido por 's' e já está compreendido neste bloco.

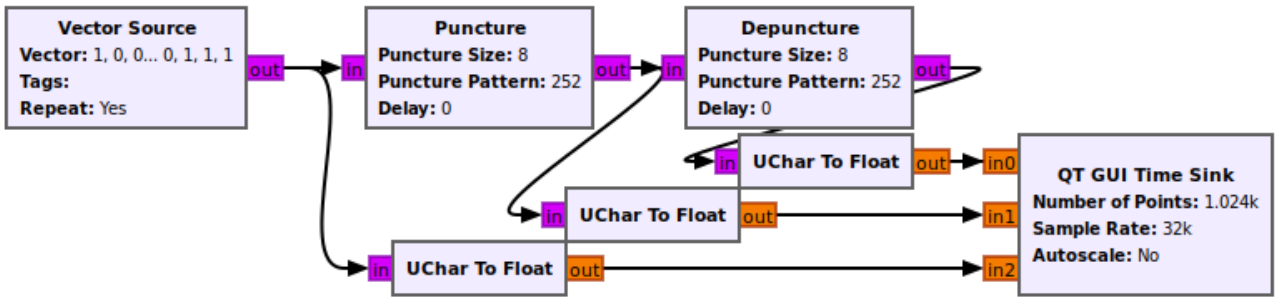
4.1.3 Puncturing

O uso da técnica do *puncturing* consiste na perfuração de bits da mensagem original através de um padrão binário estabelecido pelo projetista, isto é, através de uma sequência de bits pré-

estabelecida é definido um padrão para que determinados bits de uma sequência sejam apagados (perfurados) de modo que na recepção pelo processo de *depuncturing* possam ser adicionados novamente. Este processo permite o aumento na taxa de dados transmitidos pela redução do tamanho da mensagem enviada.

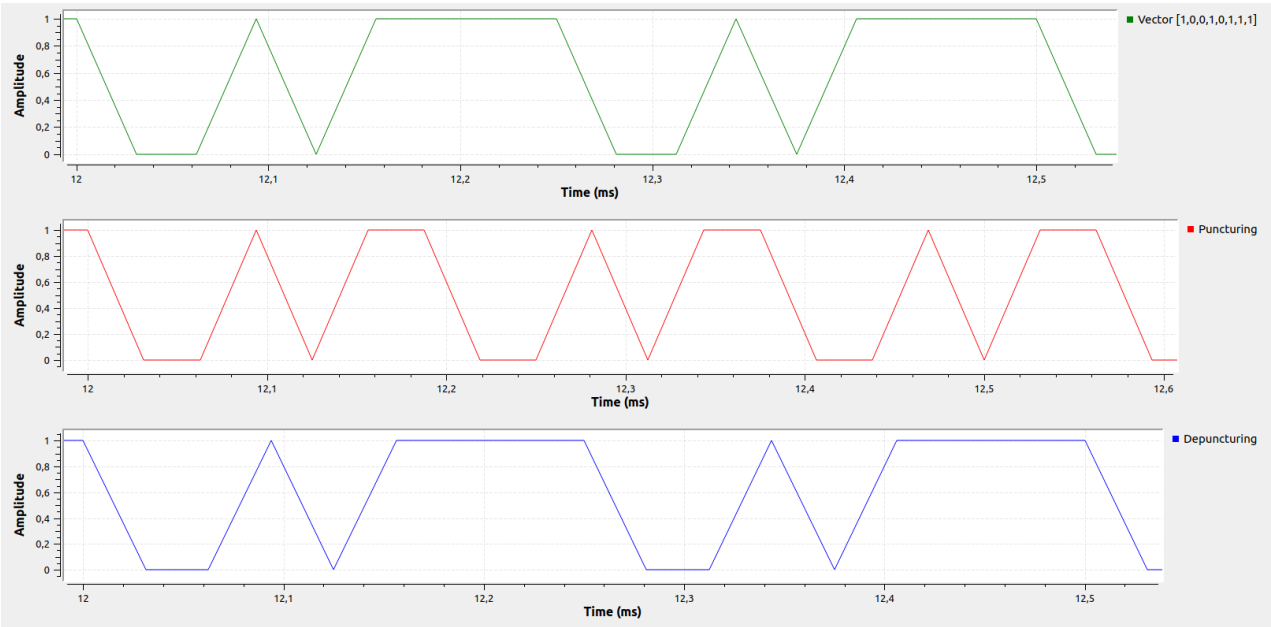
Na Figura 4.5 e 4.6 é evidenciado o funcionamento do *puncturing*. Utilizando-se do vetor [1,0,0,1,0,1,1,1] como fonte de sinal, o fluxo de blocos mostrado na Figura 4.5 foi capaz de remover e preencher de maneira correta os bits referentes ao decimal 252, que em binário apresenta os dois últimos dígitos menos significativos sendo '0'.

Figura 4.5 - Fluxograma *Puncturing* e *Depuncturing*



Fonte: do Autor.

Figura 4.6 - Comparação entre *bitstream* original, *bitstream* pós-*Puncturing* e *bitstream* pós-*Depuncturing*



Fonte: do Autor.

Observa-se que na forma de onda bem de cima (em verde) o *bitstream* original apresenta dois ciclos enquanto que na forma de onda do meio (em vermelho) o *bitstream* completa aproximadamente três ciclos no mesmo intervalo de tempo. Isto se dá porque após o *puncturing* de 252, os dois últimos

bits de cada byte foram cortados, transformando-se assim em símbolos com 6 bits. De modo similar, na forma de onda bem de baixo (em azul) o processo de *depuncturing* faz o preenchimento correto dos bytes com os bits faltantes.

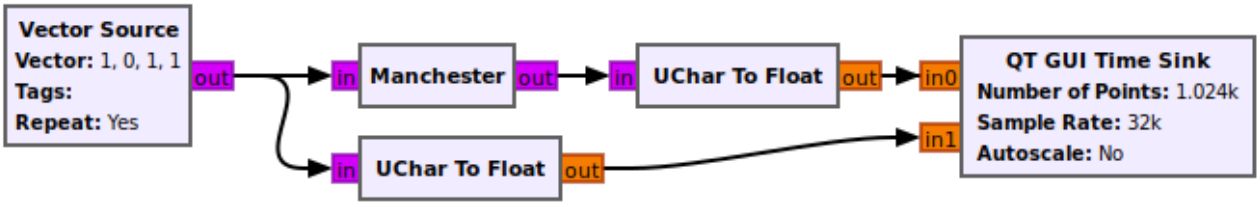
4.1.4 Run-Length Limited

O uso de códigos RLL é essencial para sistemas VLC que são muito sensíveis a variação da luz. Em sistemas nos quais a iluminação é o objetivo primário e a comunicação é complementar se torna ainda mais imprescindível o uso dos códigos RLL em função de garantirem uma corrente DC balanceada no LED bem como remover totalmente o efeito do *flicker* ao evitar longas sequencias de ‘0’na transmissão do sinal (TURAN, et al. 2016).

O GNU Radio não conta com nenhum tipo de codificação RLL em sua biblioteca, e neste caso, adicionaram-se dois tipos manualmente através do *gr_modtool* e suas funções, apresentadas no item 2.2.2 deste trabalho.

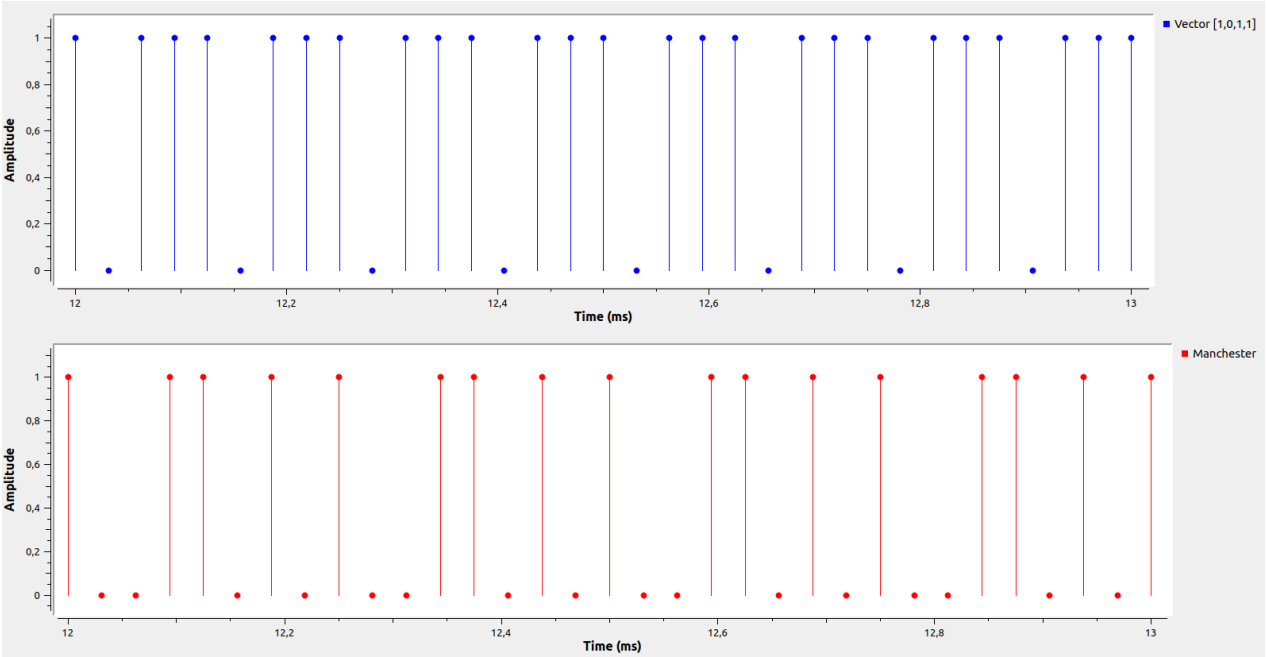
As codificações envolvendo os codificadores e decodificadores Manchester e 4B6B se encontram nos Apêndices A, B, C e D. Nas Figuras 4.7 e 4.8 são mostrados, respectivamente, o fluxograma do GNU Radio para o uso do bloco Manchester criado e como o sinal fica após a passagem do mesmo.

Figura 4.7 - Fluxograma Manchester – Encode



Fonte: do Autor.

Figura 4.8 - Comparação entre *bitstream* original e *bitstream* pós processamento bloco RLL Manchester



Fonte: do Autor.

Como pode-se observar no uso do bloco Manchester o bit em nível alto ‘1’ se torna o conjunto ‘1’ e ‘0’, nesta ordem, enquanto que o bit ‘0’ se transforma em ‘0’ e ‘1’, conforme definido na tabela 4.1. Assim, o *bitstream* de [1,0,1,1] se transforma em [1,0,0,1,1,0,1,0] apresentando, portanto, duas vezes mais amostras na saída do bloco, se caracterizando desta vez como um bloco interpolador, dentro da engine do GNU Radio.

Tabela 4.1 - Padrão para codificação Manchester

Bit	Símbolo Manchester
0	01
1	10

Fonte: adaptado de (IEEE STANDARDS ASSOCIATION 2018).

O uso da Manchester se caracteriza por aumentar o número de bits necessários para enviar uma mensagem em 2x reduzindo assim a taxa de transmissão de dados como é comprovado na Figura 2.6. O uso da codificação 4B6B busca reduzir o impacto do aumento de símbolos na mensagem a ser transmitida, mas mantendo ainda o controle da incidência de bits ‘0’ na saída do transmissor para evitar períodos extensos com sinal baixo.

A codificação 4B6B faz o aumento de símbolos na saída em 1,5x e é padronizado segundo a tabela 4.2 exibida abaixo.

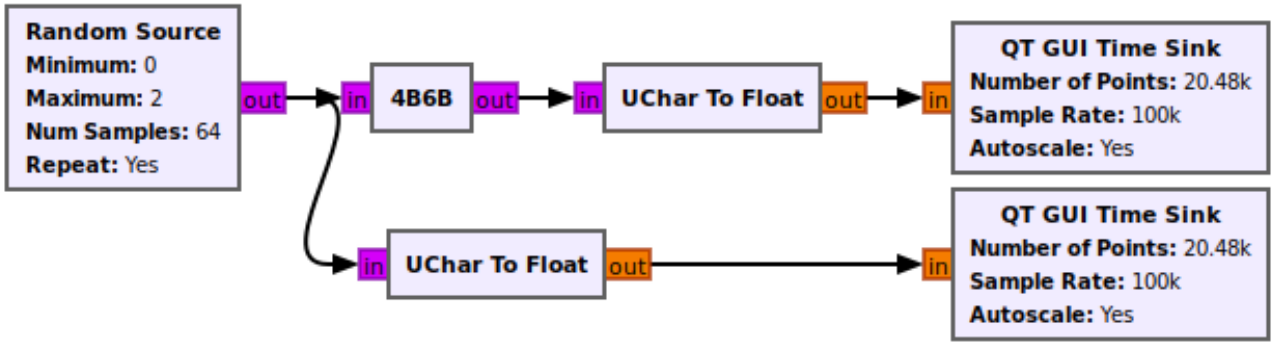
Tabela 4.2 - Mapeamento dos simbolos 4B6B

4B (input)	6B (output)	Hex
0000	001110	0
0001	001101	1
0010	010011	2
0011	010110	3
0100	010101	4
0101	100011	5
0110	100110	6
0111	100101	7
1000	011001	8
1001	011010	9
1010	011100	A
1011	110001	B
1100	110010	C
1101	101001	D
1110	101010	E
1111	101100	F

Fonte: adaptado de (IEEE STANDARDS ASSOCIATION 2018).

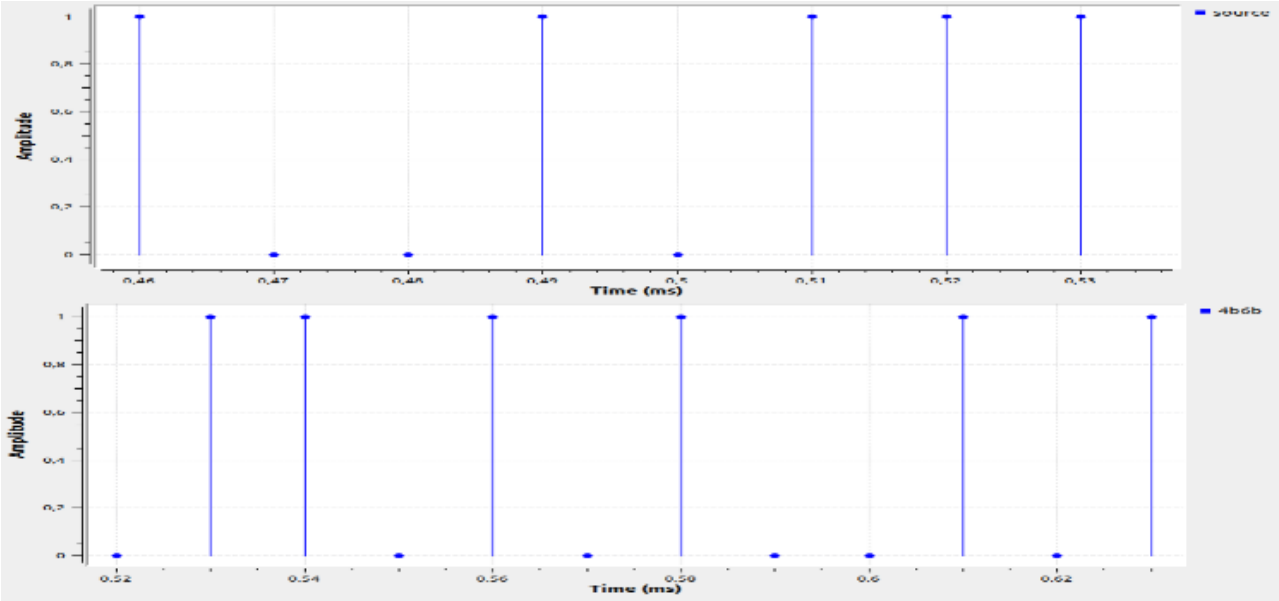
Como pode ser observado, na saída (output) sempre há a presença de pelo menos três bits em nível alto de modo que assim sempre haja uma taxa de 50%. Nas Figuras 4.9 e 4.10 são exibidos, respectivamente, o fluxograma construtivo do codificador 4B6B e o formato do sinal após a sua operação, comparando com o sinal de entrada.

Figura 4.9 - Fluxograma 4B6B – Encode



Fonte: do Autor.

Figura 4.10 - Comparação entre *bitstream* original (em cima) e *bitstream* pós processamento bloco RLL 4B6B (embaixo)



Fonte: do Autor.

Através das figuras do codificador 4B6B, verifica-se que o *bitstream* [1,0,0,1,0,1,1,1] se transforma em [0,1,1,0,1,0,1,0,0,1,0,1] aumentando o número de amostras na saída em 1,5 vezes sendo também considerado como um bloco do tipo interpolador definido no item *2.2.2.2* deste trabalho.

4.1.5 Modulação OOK e VPPM

As modulações OOK e VPPM são ambas modulações de amplitude e não são nativas do GNU Radio e, portanto, devem ser programadas externamente e adicionadas via *gr_modtool* dentro da interface do software.

A modulação OOK apresenta um nível alto para representar o bit ‘1’ e um nível baixo para representar o bit ‘0’. Para os sistemas VLC, o funcionamento do LED é definido pelas variações das grandezas de corrente e tensão causadas pela modulação, de forma que permite que outras modulações de amplitude como a BPSK ou QPSK sejam interpretadas como um tipo de modulação OOK em que a única diferença entre elas é que para o nível baixo o bit transmitido é de ‘-1’.

Assim sendo, para os testes em que o padrão IEEE 802.15.7-2018 define a modulação OOK para o sistema, optou-se por utilizar o bloco modulador do próprio GNU Radio denominado *Constellation Modulator* o qual é capaz de reproduzir a modulação BPSK que aplicada ao VLC apresenta o mesmo comportamento da OOK.

Ao contrário da modulação OOK, a modulação VPPM se caracteriza por apresentar diferentes níveis altos e baixos em função do *duty cycle* definido. A tabela 4.3 abaixo apresenta o padrão definido para a programação dos blocos de modulação VPPM.

Tabela 4.3 - Definição do mapeamento de dados para modulação VPPM

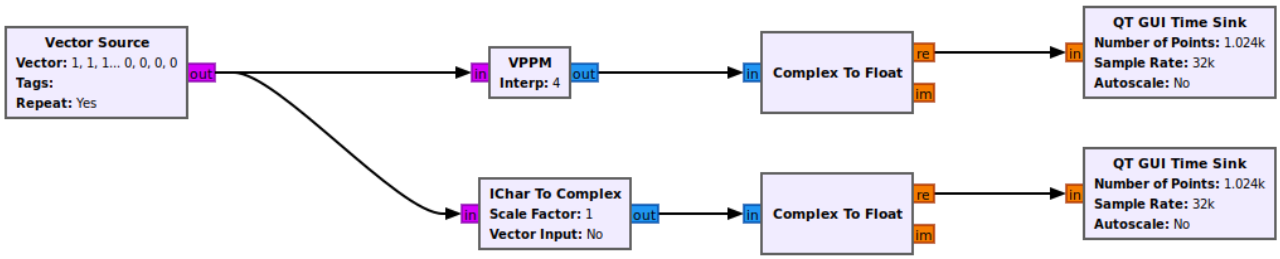
Bit	Símbolo	
	d é o *duty cycle* VPPM ($0.1 \leq d \leq 0.9$)	
0	Alto	$0 \leq t < dT$
	Baixo	$dT \leq t < T$
1	Baixo	$0 \leq t < (1-d)T$
	Alto	$(1-d)T \leq t < T$

Fonte: adaptado de (IEEE STANDARDS ASSOCIATION 2018)

No apêndice E e F está descrito a codificação usada para desenvolver os blocos de modulação VPPM. Como pode ser observado na tabela 4.3, deve-se definir um valor de interpolação responsável por definir o período em nível alto e baixo do sistema.

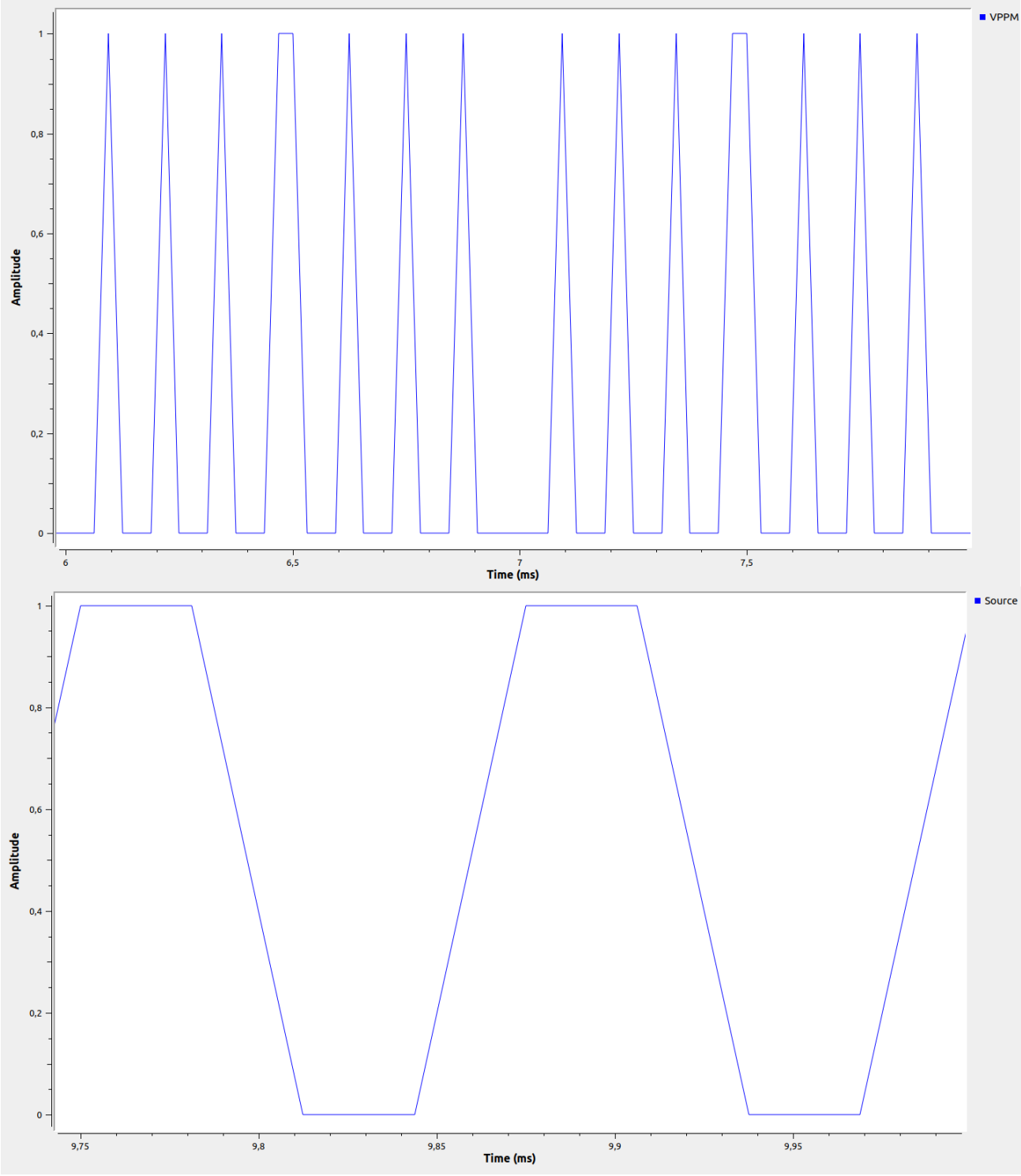
As Figuras 4.11 e 4.12 demonstram, respectivamente, o fluxograma da modulação VPPM usado para a comparação do sinal e também a mensagem enviada.

Figura 4.11 - Fluxograma Modulação VPPM



Fonte: do Autor.

Figura 4.12 - Comparação entre *bitstream* modulado pelo VPPM e o original



Fonte: do Autor.

Observa-se que para a mensagem enviada contendo 8 bits [1,1,1,1,0,0,0,0] a partir da modulação VPPM com o ajuste de *duty cycle* definido pelo coeficiente de interpolação diretamente proporcional ao tempo, tem-se a saída composta por uma sequência de binária de [0,0,0,1,0,0,0,1, 0,0,0,1,0,0,0,1,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0].

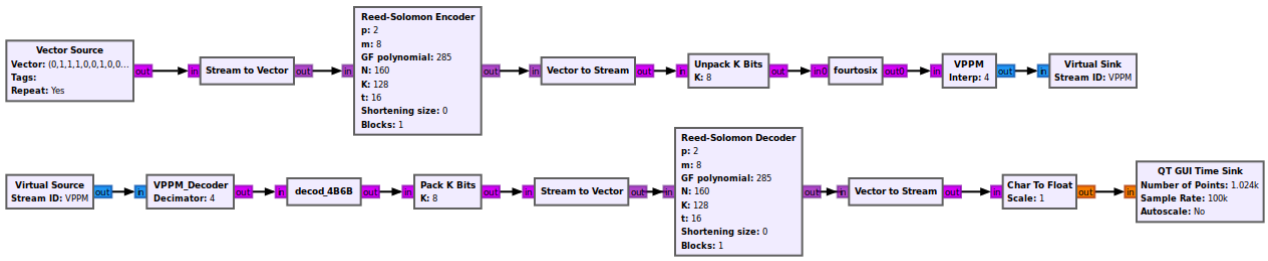
Apesar do número de amostras na saída ser superior em quatro (4) vezes, elas são compensadas pela quantidade que equipamento de medição requer.

Todos os ensaios deste capítulo foram realizados com uma taxa de amostragem de 32.000 amostras, ou bits/s. Isto significa que cada bit enviado pela fonte mantém seu nível durante um período de 0,03125 ms. Para o caso da modulação VPPM, a duração de cada bit é reduzida em '*n*' vezes, sendo n igual ao coeficiente de interpolação para o *duty cycle*, mantendo assim a taxa de dados do sistema.

4.2 ENSAIOS SIMULADOS

Para verificar o comportamento da mensagem através do *flowgraph* do GNU Radio, simulou-se no GNU Radio um sistema equivalente à camada PHY-II da IEEE 802.15.7-2018, conforme mostrado na Figura 4.13.

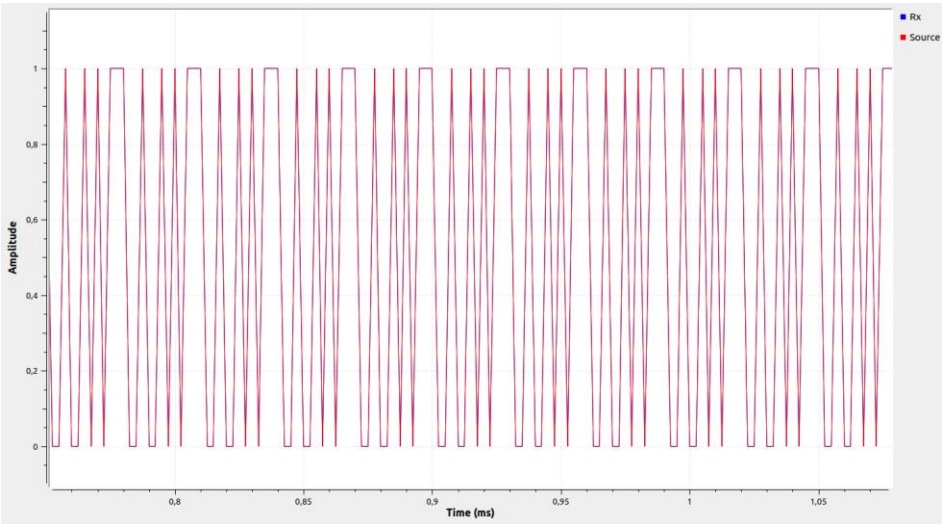
Figura 4.13 - Fluxograma Camada PHY-II



Fonte: do Autor.

A taxa de amostragem convencionada foi de 100.000 amostras por segundo. Como é observado na Figura 4.14 a forma de onda da mensagem enviada pela fonte e a mensagem decodificada pelo receptor (Rx) se sobrepõe.

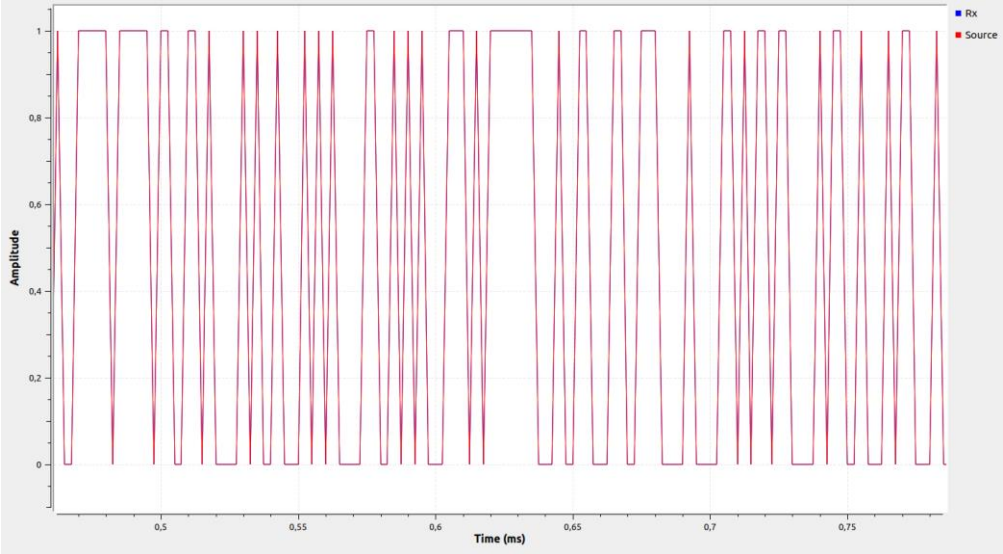
Figura 4.14 - Comparação 1- Bits enviados e recebidos



Fonte: do Autor.

Usando um gerador de sinal randômico, a Figura 4.15 evidencia o mesmo comportamento, sem a incidência de erros pela utilização dos novos blocos de modulação e RLL.

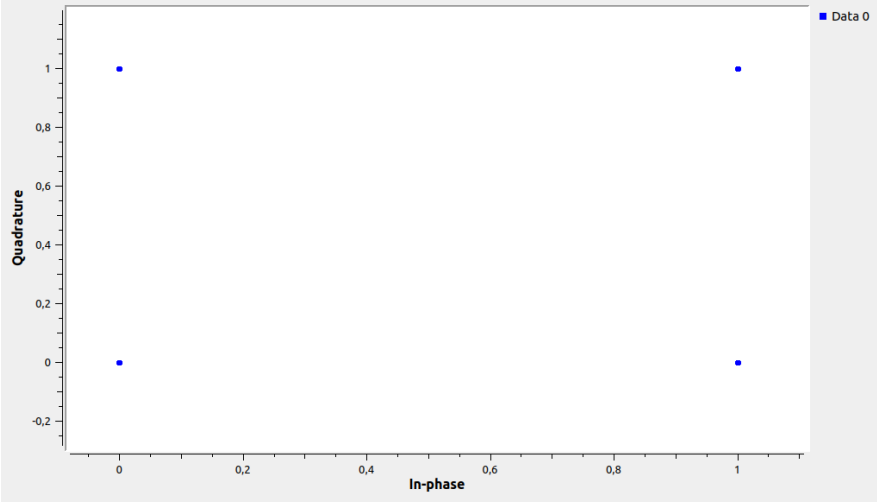
Figura 4.15 - Comparação 2 - Bits enviados e recebidos



Fonte: do Autor.

Por se tratar de um ambiente simulado sem a incidência de fatores externos para interceptar o sinal e gerar erros o gráfico de constelação se deu como na Figura 4.16.

Figura 4.16 - Gráfico de constelação do sinal transmitido sem interferência



Fonte: do Autor.

4.3 CONCLUSÕES PARCIAIS

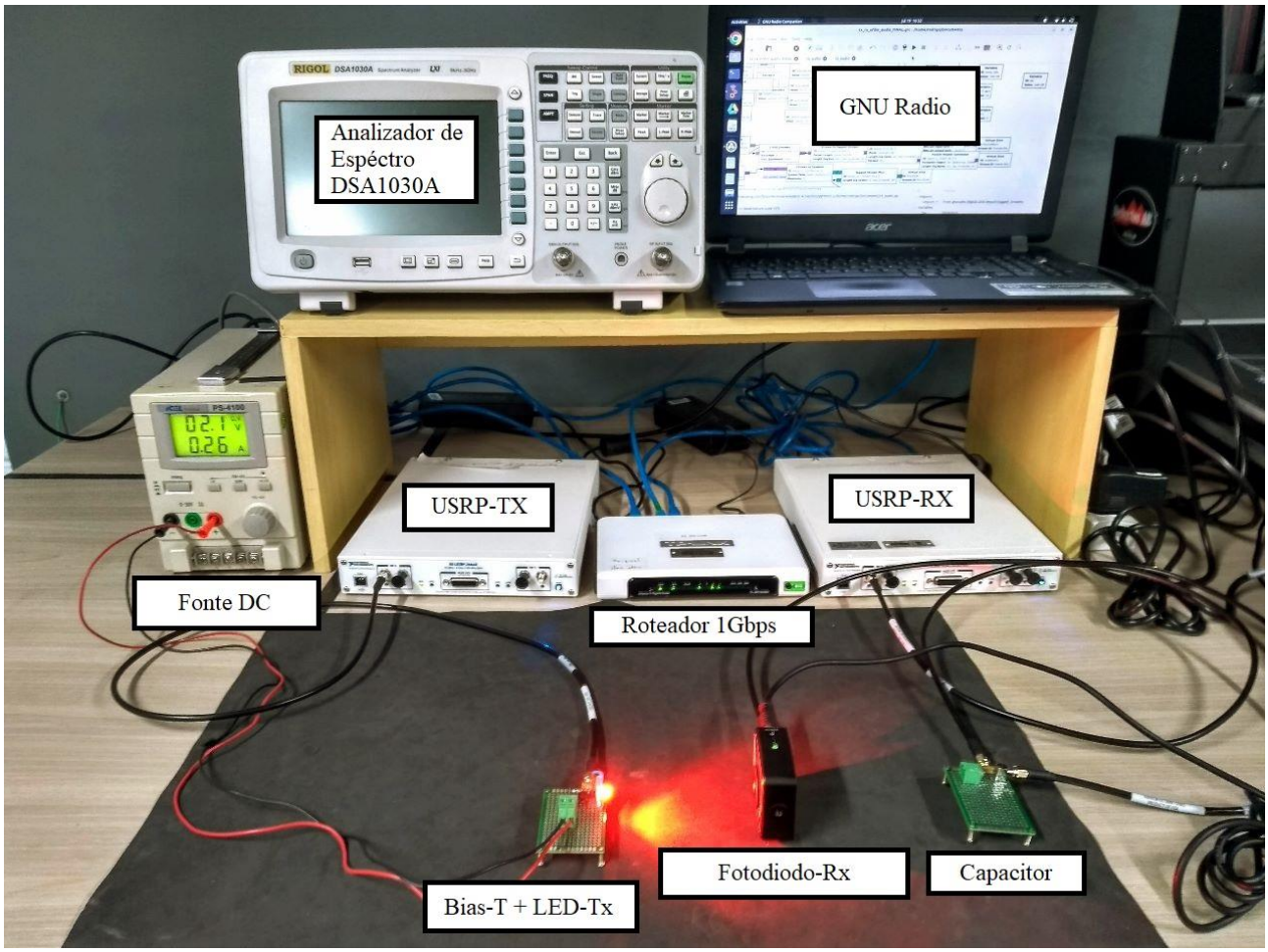
Nesse capítulo foram apresentados os blocos de comunicação referentes ao VLC que são presentes no GNU Radio, bem como definido novos blocos para a modulação VPPM e os códigos RLL Manchester e 4B6B. Por fim foi testado o circuito simulado para a camada PHY-II a fim de verificar o comportamento do sistema.

5 DESENVOLVIMENTO EXPERIMENTAL

Este capítulo apresenta os resultados experimentais do sistema prático criado para testes de comunicação a partir do padrão estabelecido na IEEE 802.15.7-2018 para camada a PHY-I e PHY-II.

Os experimentos realizados foram feitos utilizando um USRP 2944 da National Instruments (NATIONAL INSTRUMENTS 2022) com a transmissão a partir de um LED CREE XLamp XR-C (CREE LED 2021) e recepção do sinal através de um fotodetector amplificado PDA10A2 fabricado pela Thorlabs (THORLABS 2019). A Figura 5.1 exibe todos os equipamentos utilizados para a realização dos ensaios sobrepostos em bancada.

Figura 5.1 - Configuração dos componentes para ensaios dispostos em bancada



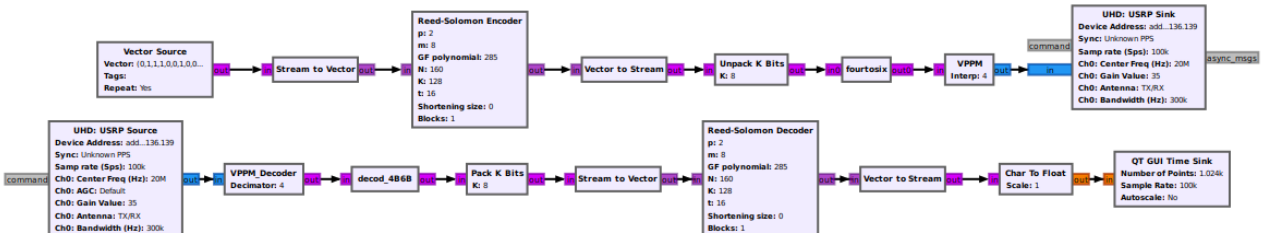
Fonte: adaptado de (MIRANDA 2022).

Para a realização do experimento, o transmissor e o receptor foram posicionados na bancada de testes e afastados apenas 10 centímetros um do outro.

5.1 TRANSMISSÃO DE SINAL – CAMADA PHY-II

Para a realização dos primeiros testes, utilizou-se um circuito similar ao da Figura 4.13 para a checagem da transmissão de dados segundo a camada PHY-II com o uso dos blocos 4B6B e VPPM adicionados. Os dados foram extraídos através do analisador de espectro na saída do capacitor do *Bias-T* a fim de verificar se o sinal que estava sendo transmitido pelo LED estava de acordo com o definido no SDR. A realização do teste de comunicação foi feita com a frequência da portadora do sinal em 20MHz, Taxa de amostragem em 100kS/s e largura de banda em 300 kHz.

Figura 5.2 - Fluxograma Camada PHY-II ensaio prático



Fonte: do Autor.

Diferentemente dos ensaios simulados, em que não há a necessidade de conectar o GNU Radio com o USRP, para a realização dos ensaios práticos tem-se a necessidade de utilizar os blocos *UHD:USRP Sink* e *UHD:USRP Source*. Nestes blocos são definidos parâmetros usados na comunicação como a frequência da portadora e largura de banda utilizada no sistema.

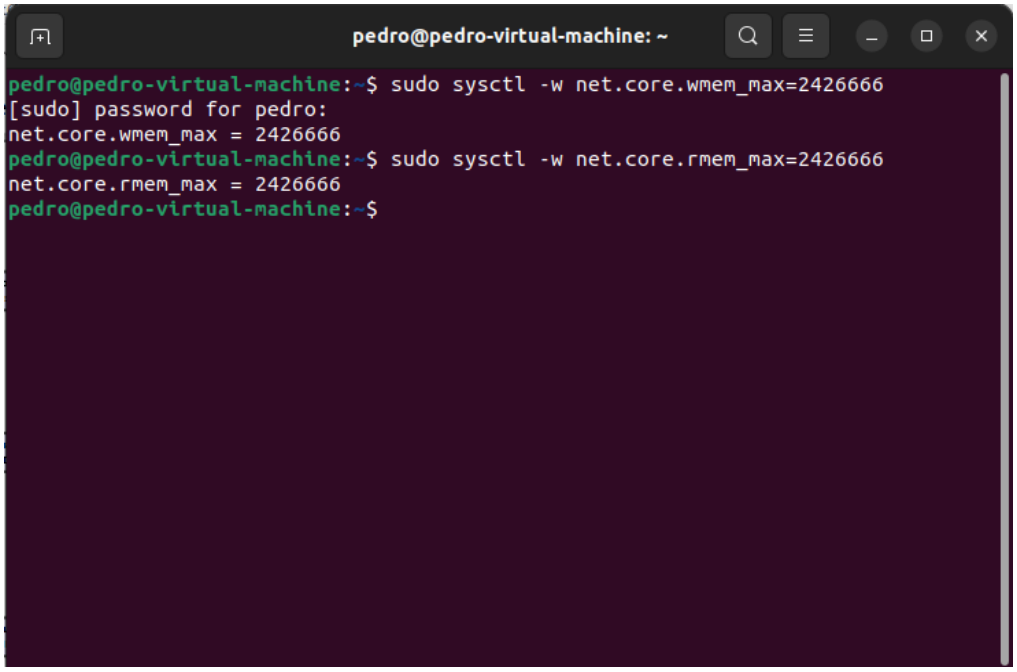
Inicialmente o sistema solicitou um ajuste no tamanho dos *buffers* em decorrência do sistema linux agora usar o *User Datagram Protocol* (UDP) como o protocolo padrão no Linux para comunicação entre processos, como transferências de buffer de cache entre as instâncias. Os buffers de recebimento são usados pelo TCP e UDP para manter os dados recebidos até que sejam lidos pelo GNU Radio. O buffer de recebimento não pode estourar porque o par não tem permissão para enviar dados além da janela de tamanho do buffer. Isso significa que os datagramas serão descartados se não couberem no buffer de recepção do soquete. Isso pode fazer com que o remetente sobrecarregue o destinatário. Deste modo, através dos comandos abaixo, foi necessário ajustar o tamanho dos buffers de envio e recebimento de dados para o valor necessário no uso do USRP 2944.

```
sudo sysctl -w net.core.rmem_max=2426666 (15)
```

```
sudo sysctl -w net.core.wmem_max=2426666 (16)
```

A Figura 5.3 apresenta as linhas de comando no terminal do Linux.

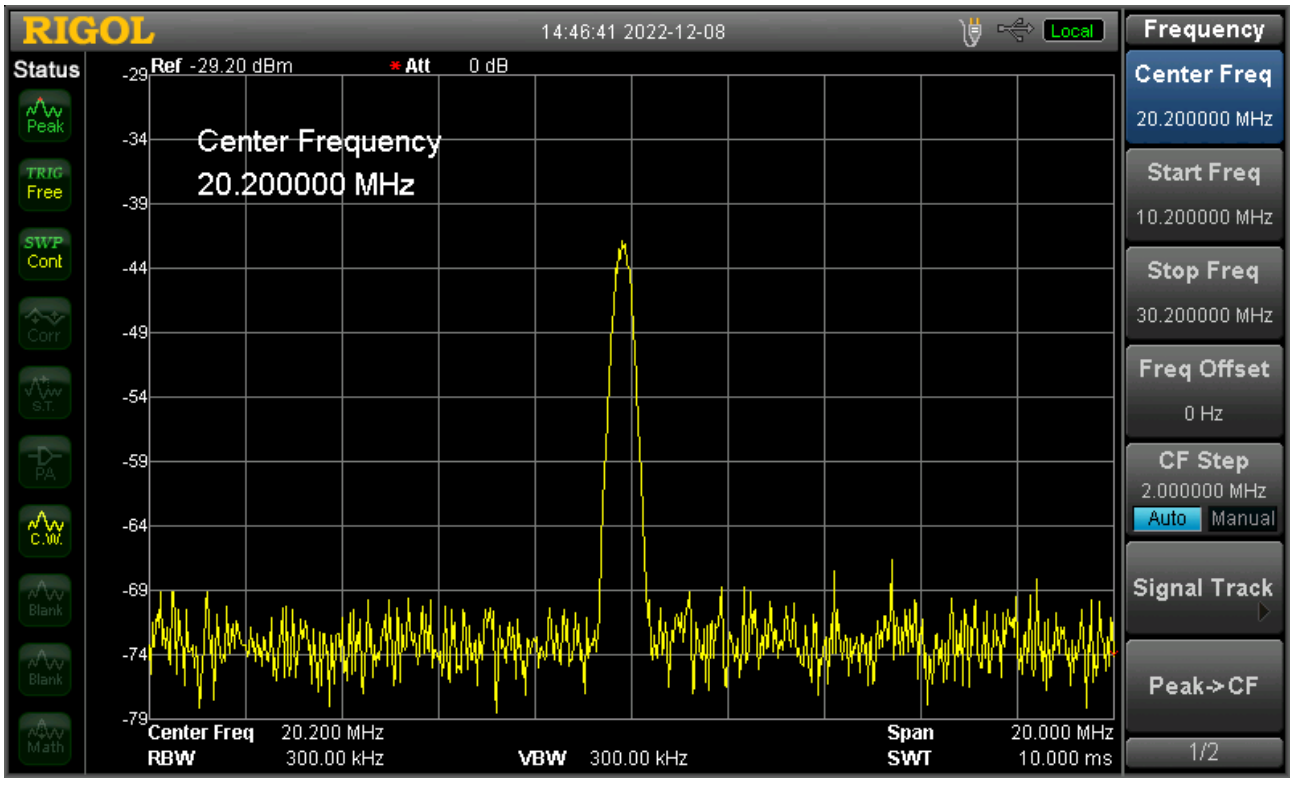
Figura 5.3 - Alteração do tamanho dos buffer no Linux



Fonte: do Autor.

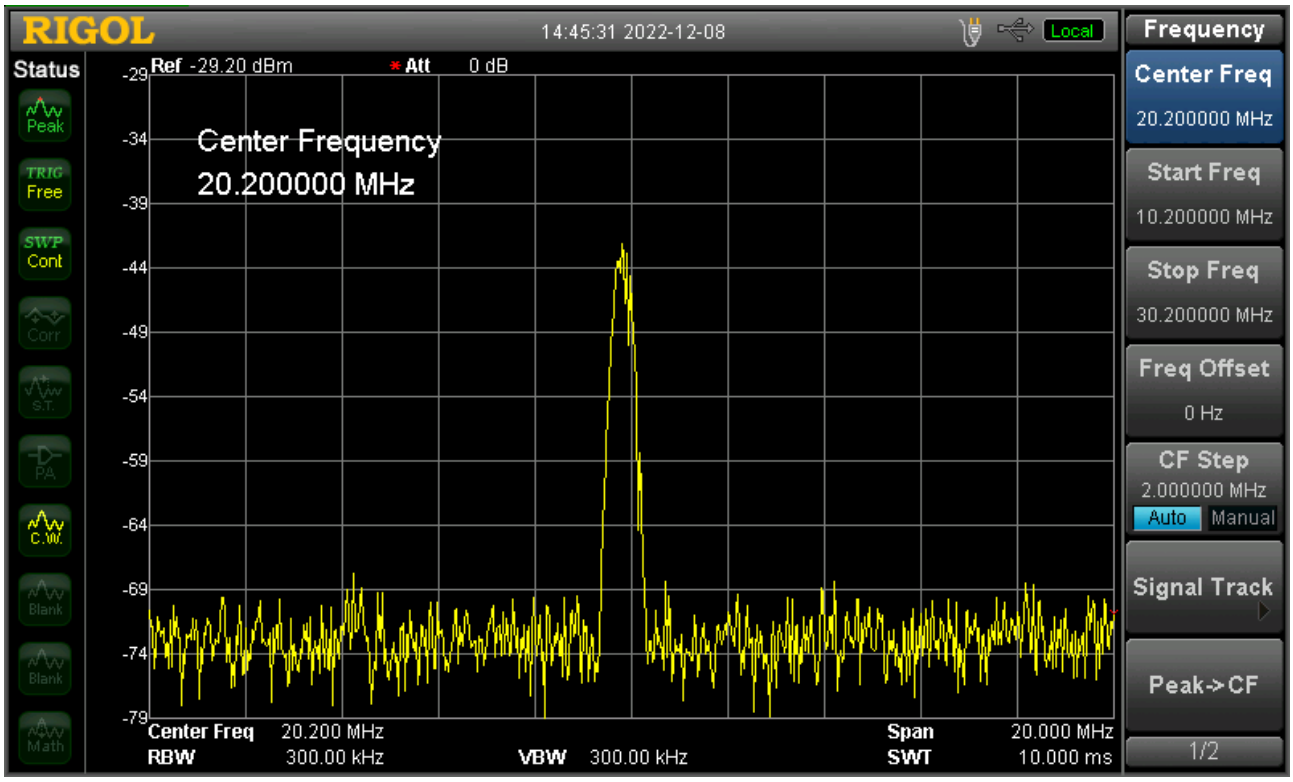
Para a realização dos testes, foi considerado o índice de interpolação do modulador VPPM de: Figura 5.4 – dois (2), Figura 5.5 – três (3) e Figura 5.6 – quatro (4). Deste modo, quanto maior o coeficiente mais menor o *duty cycle* e, consequentemente, menor o tempo de sinal em nível alto para a transmissão dos bits.

Figura 5.4 - Comunicação VPPM, coeficiente 2



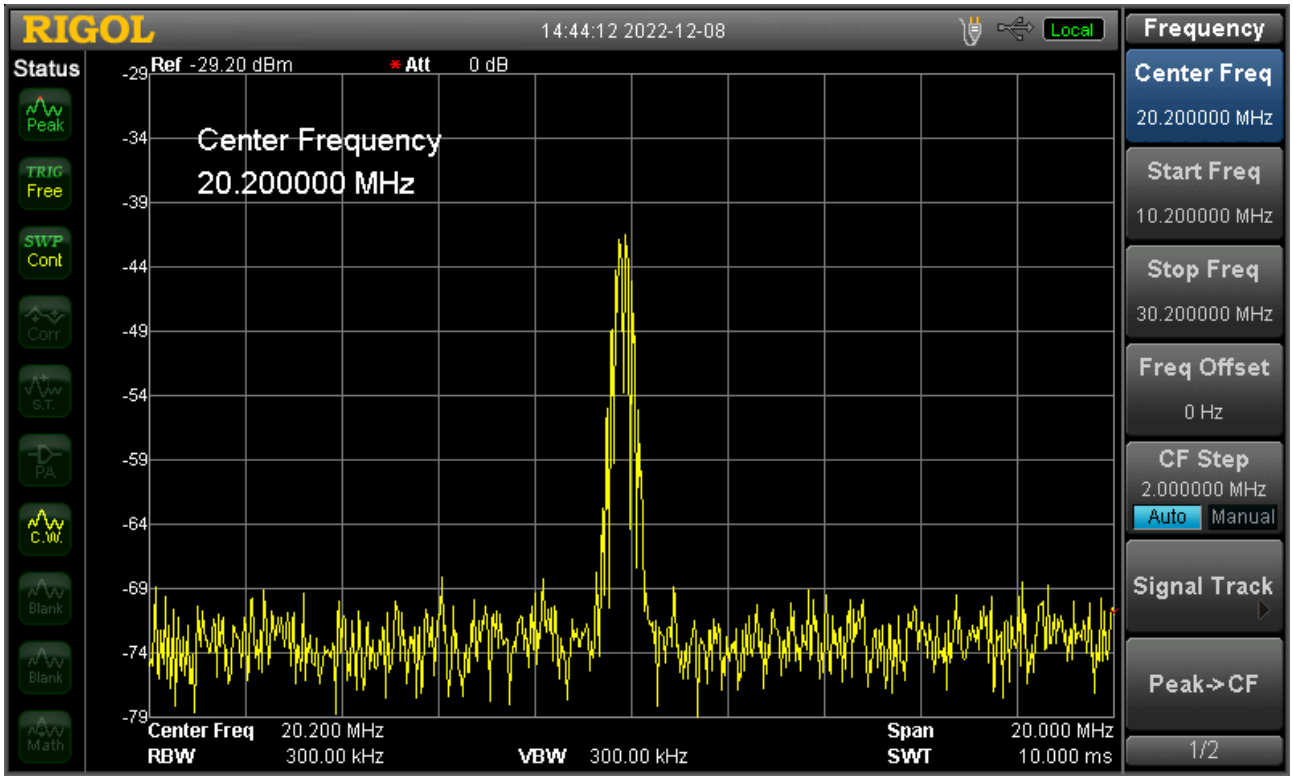
Fonte: do Autor.

Figura 5.5 - Comunicação VPPM, coeficiente 3



Fonte: do Autor.

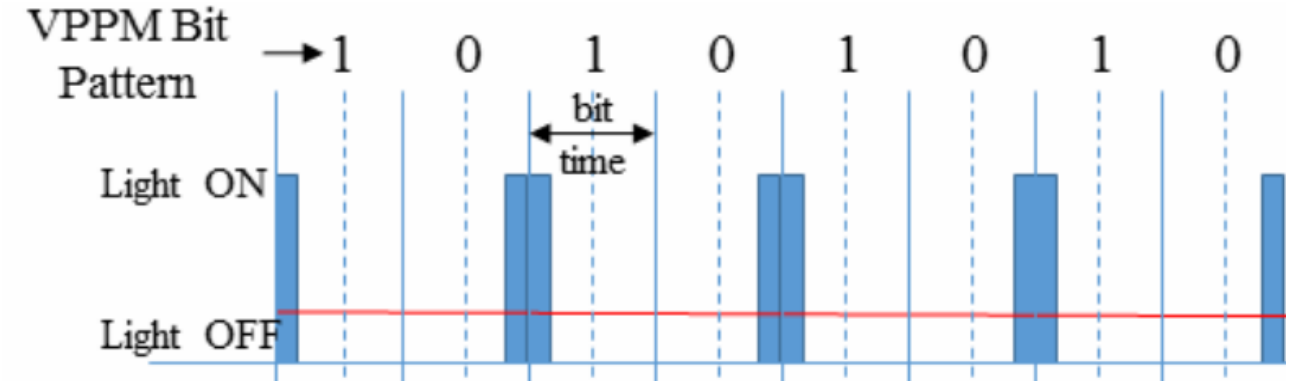
Figura 5.6 - Comunicação VPPM, coeficiente 4



Fonte: do Autor.

Os resultados apresentaram uma degradação no sinal enviado devido ao aumento do período em nível baixo característico da modulação VPPM que mantem o mesmo tempo de sinal em nível alto enquanto desloca a posição do pulso no eixo de modo diferente para bits ‘1’s e ‘0’s, como pode ser observado na Figura 5.7.

Figura 5.7 - Formato dos pulsos Modulação VPPM

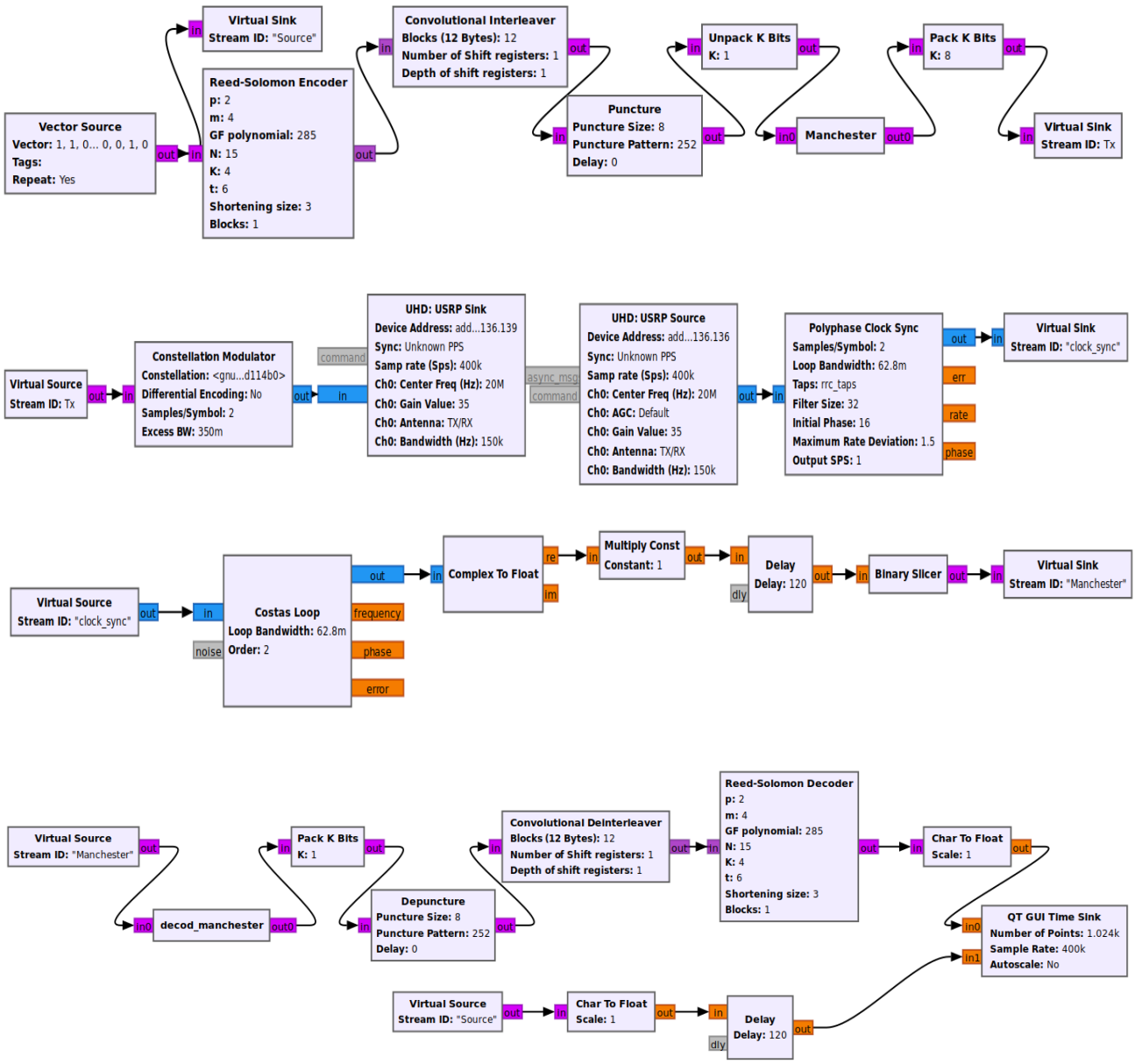


Fonte: adaptado de (IEEE STANDARDS ASSOCIATION 2018).

5.2 TRANSMISSÃO DE SINAL – CAMADA PHY-I

Conforme dito no item 4.1.5, a modulação utilizada para a realização dos testes práticos na comunicação, foi a BPSK que modula o sinal entre o nível ‘+1’ e ‘-1’. Devido a modulação BPSK ser nativa do GNU Radio ao contrário da OOK que não consta na biblioteca de blocos, sua aplicação se justificou devido a ela ser uma modulação de dois níveis, de modo similar a OOK, diferindo unicamente que para o sinal baixo a OOK envia ‘0’ e a BPSK ‘-1’, deste modo, quando aplicada ao VLC, o sinal transmitido apresenta a mesma mensagem para os níveis altos e baixos. Os blocos para processamento do sinal no GNU Radio podem ser observados na Figura 5.8.

Figura 5.8 - Fluxograma camada PHY-I



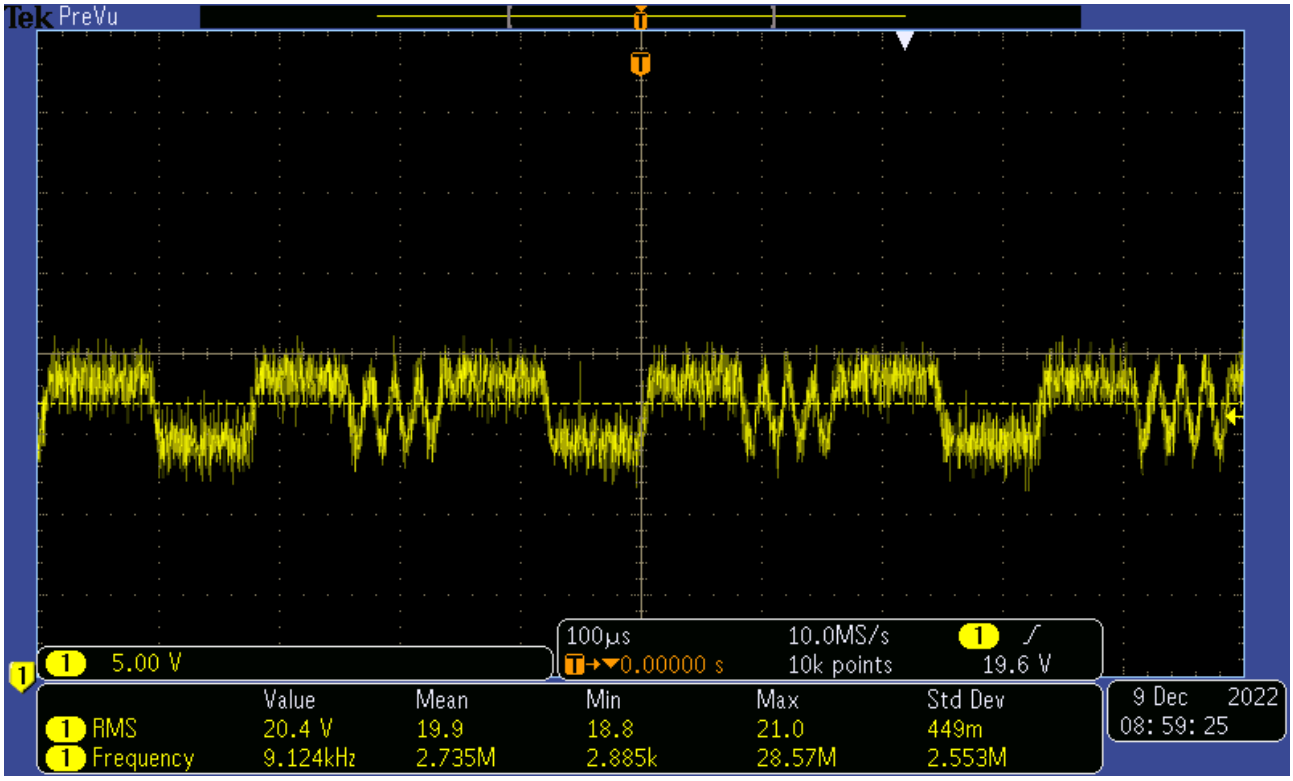
Fonte: do Autor.

Começando pelos blocos da parte superior, tem-se a parte de processamento digital do transmissor, contemplando o RS, *Interleaver*, *Puncture* e Manchester. Logo abaixo estão os blocos do USRP, onde ligado ao transmissor está o bloco referente a modulação BPSK e ligado ao receptor está o *Polyphase Clock Sync* responsável por iniciar a sincronização nos sistemas PAM a fim de buscar máximo de SNR para cada amostra, bem como reduzir o *Inter Symbol Interference* (ISI). Logo em seguida estão os blocos referentes à sincronização do sinal obtido com o sinal enviado em relação ao tempo, neste estágio o uso do *Costas Loop* e o *Delay* fornecem o suporte necessário. Por fim, tem-se os blocos para processamento digital do sinal recebido pelo Rx do USRP, contemplando todos os blocos de decodificação existentes no Tx. Nota-se que o uso do código RLL Manchester invés do

4B6B se deu devido à Standard IEEE 802.15.7 defini-lo como bloco padrão para comunicações por luz visível que utilizem a modulação OOK. O bloco RLL 4B6B apesar de consumir menos dados para a codificação da mensagem, segundo o padrão, deve ser utilizado para a modulação VPPM (experimento no item 5.2).

Inicialmente para visualizar a mensagem enviada, utilizou-se uma sequência pré-estabelecida de 32 bits [70,0,255,0]. Através do analisador de espectro, teve-se o seguinte resultado na Figura 5.9.

Figura 5.9 - Mensagem transmitida



Fonte: do Autor.

Como pode ser observado, a forma de onda é compatível com a mensagem enviada (em binário).

Como teste definitivo para o experimento, foi realizada a transmissão de um sinal aleatório de 64 bits, repetidamente, a fim de verificar a integridade do sinal recebido através da BER (do inglês - *Bit Error Rate*) do sistema. Ao codificar um bloco no GNU Radio para armazenar os dados transmitidos e recebidos em um arquivo e compará-los, torna-se possível verificar a incidência de erros ao longo do tempo.

Deste modo foram realizados 4 (quatro) ensaios nos quais a corrente elétrica no Tx (I_{led}) foi diminuída em 0,05A sucessivamente, como pode ser observado na tabela 5.1.

Tabela 5.1 - Corrente do Tx para cada Experimento

Nº do Experimento	Corrente no LED (A)
1	0.35
2	0.3
3	0.25
4	0.2

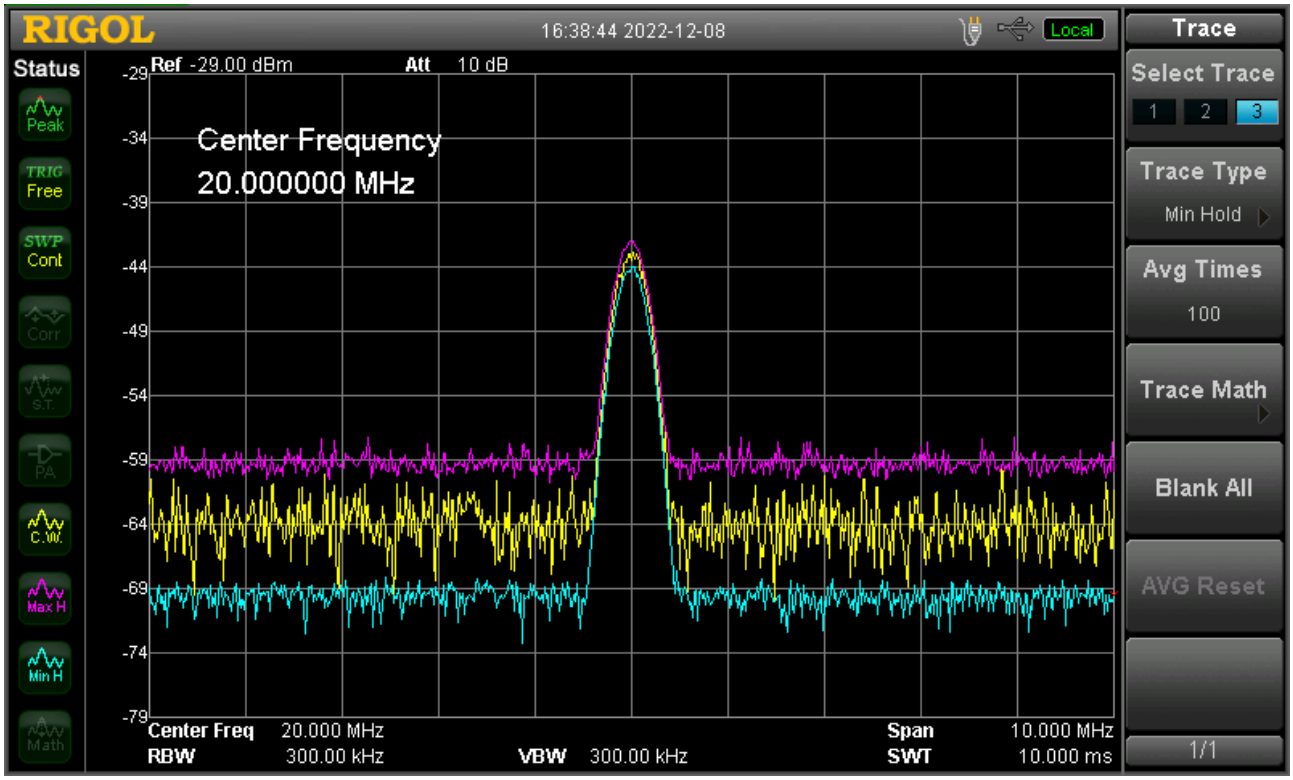
Fonte: do Autor.

Para o cálculo da BER (do inglês – *bit error ratio*) tem-se o seguinte:

$$BER = \frac{bits\ recebidos\ com\ erro}{bits\ totais\ transmitidos} \quad (1)$$

Primeiramente, foi averiguado se o sinal estava de fato sendo transmitido. A Figura 5.10 mostra o sinal do primeiro ensaio, observado por um analisador de espectro.

Figura 5.10 - Sinal observado pelo analisador de espectro



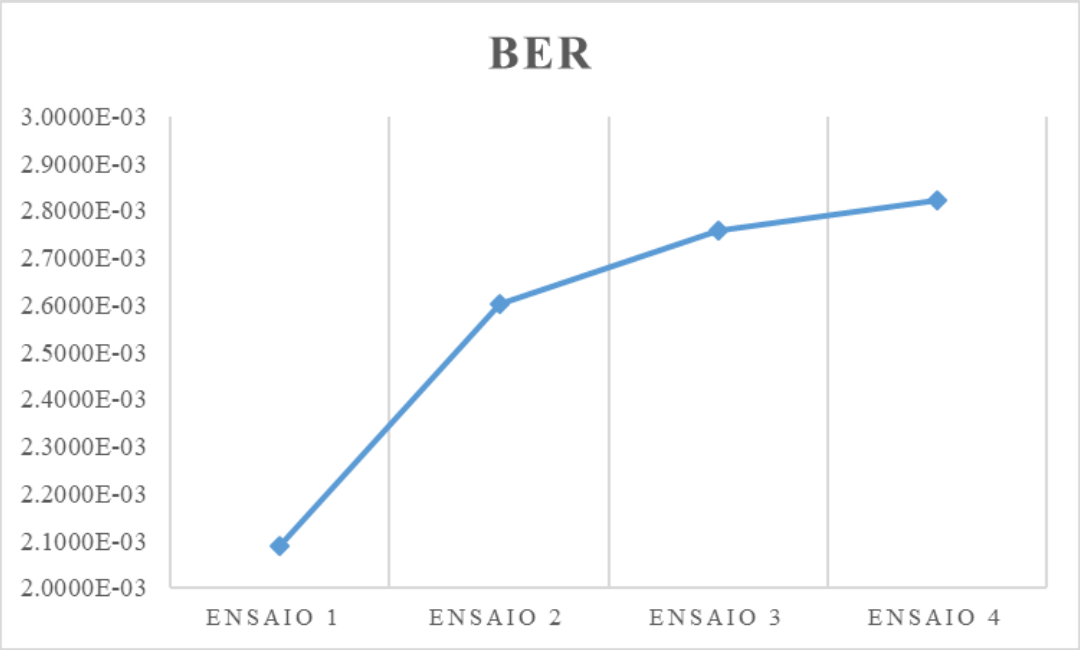
Fonte: do Autor.

Como mostrado, o sinal recebido pelo fotodiodo, representado pela amplitude próxima a -44dbm, tem sua frequência central em 20MHz e uma largura de banda de 300kHz, valores estes que coincidem com os que foram configurados no próprio GNU Radio para que o USRP operasse.

Utilizando-se dos dados armazenados em bloco de notas com as informações referentes aos bits enviados e recebidos, e ao compará-los, verificou-se que para o total de 900.000 bits enviados para

cada ensaio, em todos os casos o número de bits que foram recebidos com erro excedeu o milhar. Deste modo, aplicando (1) para cada ensaio realizado, verificou-se que a BER para todos eles foram de 10^{-3}, com um pequeno aumento no número de bits recebidos com erro cada vez que se diminuía a corrente no LED, como pode ser observado na Figura 5.11.

Figura 5.11 - BER do sistema para os Experimentos 1,2,3 e 4



. Fonte: do Autor.

5.3 CONCLUSÕES PARCIAIS

Nesse capítulo foi apresentado o sistema desenvolvido utilizando os blocos para VLC descritos anteriormente. Nesse sistema foi possível verificar o funcionamento tanto da camada PHY-I quando da camada PHY-II definidos pelo uso dos blocos criados com uma portadora de 20 MHz. Para o sistema experimental da camada PHY-I foram implementados dois blocos necessários para o controle RLL, sendo eles:

- Codificador Manchester;
- Decodificador Manchester.

Para o sistema experimental da camada PHY-II, os blocos adicionados para o sistema estar de acordo com o padrão IEEE foram:

- Modulador VPPM;
- Demodulador VPPM;
- Codificador 4B6B;
- Decodificador 4B6B.

6 CONCLUSÕES

Este capítulo apresenta as considerações finais sobre o trabalho e sugestões para a continuidade do trabalho.

6.1 CONSIDERAÇÕES FINAIS

A motivação do trabalho desenvolvido foi a falta de recursos disponíveis para a realização de ensaios do VLC que é uma tecnologia emergente. A partir da criação de novos blocos codificáveis, que atuem segundo os padrões estabelecidos pelas normas regentes, é possível ampliar o horizonte de pesquisa para outras atividades. A criação da biblioteca com blocos pertinentes ao VLC, dentro do software GNU Radio, teve seu funcionamento comprovado através das simulações digitais, em que a passagem dos bits foi acompanhada bloco a bloco, e nos ensaios experimentais nos quais foi possível constatar os bits definidos no *software* GNU Radio diretamente sendo recebidos pelo receptor físico.

Com os novos blocos adicionados para o uso do SDR, é possível realizar a transmissão de dados dispensando grande parte dos componentes analógicos que compõe o driver do LED. Deste modo, a prototipagem de novos sistemas é acelerada, permitindo assim definir o comportamento do sistema de maneira direta através das linhas de codificação que antes, através de métodos analógicos seriam mais morosos.

Conforme foi verificado na transmissão de dados da camada PHY-I, tem-se que para o sistema proposto, ao variar a corrente no LED transmissor, verificou-se o aumento no número de bits errados recebidos na mensagem como esperado de qualquer sistema que apresente diminuição de potência. Observa-se que neste experimento, a não utilização do CC para a correção de erros de maneira concatenada com o RS, como é requerido no padrão IEEE 802.15.7-2018, certamente foi um dos fatores determinantes para a BER obtida nos ensaios.

6.2 SUGESTÃO DE TRABALHOS FUTUROS

- Adicionar novos tipos de blocos programáveis pertinentes à tecnologia VLC, principalmente aqueles estabelecidos na normativa IEEE 802.15.7-2018;

- Adicionar novas ferramentas no GNU Radio para análise de dados como BER e tempo de execução dos *loopings*.

- Otimizar, de maneira especializada, os blocos criados neste trabalho para obtenção de melhores resultados;

- Programar novos blocos em linguagem C++, que também é compatível com o GNU Radio, e então realizar ensaios para comparações com blocos de função idêntica a fim de verificar o impacto da linguagem de programação utilizada na criação dos blocos;

- Realizar experimentos em outros sistemas SDR para comparação de desempenho a partir da mesma topologia de diagrama de blocos elaborado.

- Utilização de computadores com maior capacidade de processamento de dados a fim de aproveitar o máximo da largura de banda possível;

REFERÊNCIAS

CARVALHO, Pablo Amaral De; PEREIRA, Almir Gonçalves; CARVALHO, Jaqueline Lindolores De Resende. **"LI-FI (LIGHT-FIDELITY): UMA LUZ NA EVOLUÇÃO DAS COMUNICAÇÕES."** *CES Revista* 29, nº 2 (2015).

CHE, Ming; KUBOKI, Takeshi; KATO, Kazutoshi. **"Proposal of Cost-efficient and Low-complexity Platform for Software Defined Visible Light Communication."** *22nd Microoptics Conference (MOC2017).* Tóquio, 2017. 330-331.

CHEN, Manxin; LOH, Poh Chiang. **"Advanced LED Driving Approach for Visible Light Communication Systems."** *8th International Conference on Power Electronics Systems and Applications (PESA).* Hong Kong: IEEE, 2020.

CHI, Nan; et al. **"LED-based high-speed visible light communications."** *Proceedings of the SPIE*. 2018.

CISCO SYSTEMS, INC. **"Cisco Annual Internet Report (2018 - 2023)."** 2020.

CREE LED. ***XLamp XR-C - Red***. 2021.

DIMITROV, Svilen; HAAS, Harald. ***Principles of LED Light Communications Towards Networked Li-Fi***. Cambridge University, 2015.

ELGALA, Hany; MESLEH, Raed; HAAS, Harald. **"Indoor Broadcasting via White LEDs and OFDM."** *IEEE Transactions on Consumer Electronics*, Agosto de 2009: 1127-1134.

ELGALA, Hany; MESLEH, Raed; HAAS, Harald. **"Indoor Optical Wireless Communication: Potential and State-of-the-Art."** *IEEE Communications Magazine*, Setembro 2011: 56-62.

ELGANIMI, Taissir Y. **"Performance Comparison between OOK, PPM and PAM Modulation Schemes for Free Space Optical (FSO) Communication Systems: Analytical Study."** *International Journal of Computer Applications* 79, nº 11 (2013): 22-27.

GARG, Vijay K. ***Wireless Communications & Networking.*** Elsevier Inc., 2007.

GEISEL, W.A. ***Tutorial on Reed-Solomon Error Correction Coding***. National Aeronautics and Space Administration (NASA) Tech. Briefs. Houston, Texas, 1990.

GNU RADIO . ***GNU Radio the Free & Open Software Radio Ecosystem***. 2022.

GNU RADIO PROJECT. **"GNU Radio - Hardware."** *GNU Radio.* 13 de Dezembro de 2022. https://wiki.gnuradio.org/index.php/Hardware.

GNU RADIO PROJECT. "**Types of Blocks."** *GNU Radio.* 17 de 11 de 2021. https://wiki.gnuradio.org/index.php/Types_of_Blocks.

GOLDSMITH, Andrea. **"Multicarrier Modulation."** Em *Wireless Communications*, 374-402. Cambridge University Press, 2005.

HAAS, Harald. **"High-speed wireless networking using visible light."** *SPIE,* 19 de Abril de 2013.

HAAS, Harald; CHEN, Cheng. **"What is LiFi?"** *European Conference on Optical Communication (ECOC).* Valencia, 2015.

HUANG, Yong; GUO, Zhiyou; HUANG, Hongyong; SUN, Huiqing. **"Influence of Current Density and Capacitance on the Bandwidth of VLC."** *IEEE Photonics Technology Letters* (IEEE) 30 (2018): 773 - 776.

HUSSAIN Waqas; UGURDAG, H. Fatih; UYSAL, Murat. **"Software Defined VLC System: Implementation and Performance Evaluation."** *2015 4th International Workshop on Optical Wireless Communications (IWOW).* Istanbul, 2015. 117-121.

IDRIS, Sadiq; AIBINU, Abiodun Musa; KOYUNLU, Gokhan; SANUSI, Jaafaru. **"A Survey of Modulation Schemes in Visible Light Communications."** *3rd International Conference on Trends in Electronics and Informatics (ICOEI).* Tirunelveli, 2019.

IEEE STANDARDS ASSOCIATION. ***IEEE Std 802.15.7 - 2018***. New York: IEEE Computer Society, 2018, 406.

IJAZ, M.; et al. **"Experimental Investigation of the Performance of Different Modulation Techniques under Controlled FSO Turbulence Channel."** *5th International Symposium on Telecommunications.* Tehran, Iran, 2010.

ISLIM, Mohamed Sufyan; HAAS, Harald. **"Modulation techniques for LiFi."** *ZTE Commun* 14, nº 2 (2016): 29-40.

MCKENDRY, Jonathan J. D.; et al.. **“High-Speed Visible Light Communications Using Individual Pixels in a Micro Light-Emitting Diode Array.”** *IEEE PHOTONICS TECHNOLOGY LETTERS*, 15 de Setembro de 2010: 1346-1348.

KERRISK, Michael. **“ldconfig(8) — Linux manual page.”** *The Linux Programming Interface.* 22 de 03 de 2021. https://man7.org/linux/man-pages/man8/ldconfig.8.html.

KHALIFEH, Ala’ F.; ALFASFOUS, Nael; THEODORY, Ramzi; GIHA, Serina; DARABKH, Khalid A.. **“An experimental evaluation and prototyping for visible light communication.”** *Computers and Electrical Engineering*, Setembro de 2018: 248-265.

KHAN, Latif Ullah. **“Visible light communication: Applications, architecture, standardization and research challenges.”** (Elsevier) 3, nº 2 (Maio 2017): 78-88.

LEE, Kwonhyung; PARK, Hyuncheol. **“Modulations for Visible Light Communications With Dimming Control.”** *IEEE PHOTONICS TECHNOLOGY LETTERS* 23, nº 6 (2011): 1136-1138.

LI, Xianbo; et al. **“Smart μLED Display-VLC System With a PD-Based/Camera-Based Receiver for NFC Applications.**” *IEEE Photonics Journal* 11, nº 1 (February 2019).

LIU, Xiaoyan; et al. **“An InGaN micro-LED based photodetector array for high-speed parallel visible light communication.”** *Asia Communications and Photonics Conference (ACP).* Shanghai, 2018.

LIU, Xiaoyan; et al. **“Gbps Long-Distance Real-Time Visible Light Communications Using a High-Bandwidth Communications Using a High-Bandwidth.”** *IEEE Photonics Journal*, Dezembro de 2017.

LOOSE, Felipe; et al. “**Efficient Hybrid Buck Converter for Visible Light Communication in LED.”** *IEEE Transactions on Industrial Electronics* 69 (2022): 1877 - 1887.

MAPUNDA, Galefang Allycan; et al. **“Indoor Visible Light Communication: A Tutorial and Survey.”** *Wireless Communications and Mobile Computing* 2020 (2020).

MIRANDA, Rodrigo Fuchs. ***Desenvolvimento de um Sistema de Comunicação por Luz Utilizando Dispositivo de Rádio Definido por Software***. Santa Maria, Rio Grande do Sul, Agosto de 2022.

NASSAR, Hassan Zamat; CARL R. **“Introducing Software Defined Radio to 4G Wireless: Necessity, Advantage, and Impediment.”** *JOURNAL OF COMMUNICATIONS AND NETWORKS*, Dezembro de 2002.

NATIONAL INSTRUMENTS. ***USRP 2944***. 2022.

PATHAK, Parth H.; FENG, Xiaotao; HU, Pengfei; MOHAPATRA, Prasant. **“Visible Light Communication, Networking, and Sensing: A Survey, Potential and Challenges.”** *IEEE COMMUNICATIONS SURVEYS & TUTORIALS* 17, nº 4 (2015).

RAMADHANI, E; MAHARDIKA, G. P. **“The Technology of LiFi: A Brief Introduction.”** *IOP Conf. Series: Materials Science and Engineering.* 2018.

RASPBERRY PI FOUNDATION. ***Raspberry Pi hardware***. Junho de 2022.

RHEE, Sunwook; KIM, Changgeun; KIM, Juhee; JEE, Yong. **“Concatenated Reed-Solomon Code with Hamming Code for DRAM Controller.”** *Second International Conference on Computer Engineering and Applications.* 2010. 291-295.

ROBERTS, Richard D. **“Undersampled Frequency Shift ON-OFF Keying (UFSOOK) for Camera Communications (CamCom).”** *22nd Wireless and Optical Communication Conference.* Chongqing, 2013. 645-648.

ROHLING, H.; D. GALDA. **“OFDM transmission technique: A strong candidate for next-generation mobile communications.”** *The Radio Science Bulletin* 310 (2004): 47-58.

SCHUBERT, E. Fred. ***Light-emitting diodes.*** **2a.** Cambridge: Cambridge University Press, 2003.

SINGH, R.; T. O’farrell; J. P. R. David. **“Higher Order Colour Shift Keying Modulation Formats for Visible Light Communications.”** *2015 IEEE 81st Vehicular Technology Conference (VTC Spring).* Glasgow, 2015.

TEIXEIRA, Lucas.; et al. **“A Review of Visible Light Communication LED Drivers.”** *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society.* IEEE, 2019. 4274-4279.

TEIXEIRA, Lucas. **“Análise da eficácia luminosa de sistemas de iluminação com comunicação agregada.”** *Tese (Doutorado).* Santa Maria, RS: UFSM, 2020.

THORLABS. ***PDA10A2 Si Amplified Fixed Gain Detector***. 2019.

TURAN, Bugra; NARMANLIOGLU, Omer; ERGEN, Sinem Coleri; UYSAL, Murat. **“Physical Layer Implementation of Standard Compliant Vehicular VLC.”** *IEEE 84th Vehicular Technology Conference (VTC-Fall).* Montreal, 2016.

U. S. DEPARTMENT OF ENERGY. ***Energy Savings Forecast of Solid-State Lighting in General Illumination Applications.*** 13 de Dezembro de 2019.

VILLATE, Jaime E. ***Fisica 2. Eletricidade e Magnetismo.*** Porto, 2012.

WYGLINSKI, Alexander M.; NEKOVEE, Maziar; HOU, Y. Thomas. ***Cognitive Radio Communications and Networks***. 2010.

APÊNDICE A – CODIFICADOR MANCHESTER

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Copyright 2022 Pedro Moura.
#
# SPDX-License-Identifier: GPL-3.0-or-later
#

import numpy as np
from gnuradio import gr
import time

class Manchester(gr.interp_block):
    def __init__(self,):
        gr.interp_block.__init__(self,
            name="yo",
            in_sig=[np.byte],
            out_sig=[np.byte],
            interp = 2)
    def work(self, input_items, output_items):
        one = np.array([1,0])
        zero = np.array([0,1])
        check = np.empty(0, int)
        in0 = input_items[0]
        out0 = np.empty(0, int)
        input_range = int(len(in0)/4)
        range_complem = int((len(output_items[0]) - len(input_items[0]))/2)
        error = np.full(5, 5, int)
        complementation = np.full(range_complem, 10, int)
        for i in range(len(input_items[0])):
            if (in0[i] == 1):
```

```
        out0 = np.append(out0,one)
    else:
        out0 = np.append(out0,zero)
output_items[0][:] = out0
return len(output_items[0])
```

APÊNDICE B – DECODIFICADOR MANCHESTER

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Copyright 2022 Pedro Moura.
#
# SPDX-License-Identifier: GPL-3.0-or-later
#



import numpy as np
from gnuradio import gr
import time

class decod_manchester(gr.decim_block):
    """
    docstring for block decod_manchester
    """
    def __init__(self):
        gr.decim_block.__init__(self,
            name="decod_manchester",
            in_sig=[np.byte],
            out_sig=[np.byte],
            decim = 2)



    def work(self, input_items, output_items):
        one = np.array([1])
        zero = np.array([0])
        check = np.empty(0, int)
        in0 = input_items[0]
        out0 = np.empty(0, int)
        for i in range(len(output_items[0])):
```

```
    if (in0[2*i] == 1 and in0[2*i+1] == 0):
        out0 = np.append(out0,one)
    else:
        out0 = np.append(out0,zero)
output_items[0][:] = out0
return len(output_items[0])
```

APÊNDICE C – CODIFICADOR 4B6B

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Copyright 2022 Pedro Moura.
#
# SPDX-License-Identifier: GPL-3.0-or-later
#



import numpy as np
from gnuradio import gr
import time

class fourtosix(gr.basic_block):

    def __init__(self):  # only default arguments here
        gr.basic_block.__init__(self,
            name='4b6b',   # will show up in GRC
            in_sig=[np.byte],
            out_sig=[np.byte]
        )

    def compare(self,vetor):
        a4b0 = np.array([0,0,0,0])
        a4b1 = np.array([0,0,0,1])
        a4b2 = np.array([0,0,1,0])
        a4b3 = np.array([0,0,1,1])
        a4b4 = np.array([0,1,0,0])
        a4b5 = np.array([0,1,0,1])
        a4b6 = np.array([0,1,1,0])
        a4b7 = np.array([0,1,1,1])
        a4b8 = np.array([1,0,0,0])
```

```
a4b9 = np.array([1,0,0,1])
a4bA = np.array([1,0,1,0])
a4bB = np.array([1,0,1,1])
a4bC = np.array([1,1,0,0])
a4bD = np.array([1,1,0,1])
a4bE = np.array([1,1,1,0])
a4bF = np.array([1,1,1,1])
a6b0 = np.array([0,0,1,1,1,0])
a6b1 = np.array([0,0,1,1,0,1])
a6b2 = np.array([0,1,0,0,1,1])
a6b3 = np.array([0,1,0,1,1,0])
a6b4 = np.array([0,1,0,1,0,1])
a6b5 = np.array([1,0,0,0,1,1])
a6b6 = np.array([1,0,0,1,1,0])
a6b7 = np.array([1,0,0,1,0,1])
a6b8 = np.array([0,1,1,0,0,1])
a6b9 = np.array([0,1,1,0,1,0])
a6bA = np.array([0,1,1,1,0,0])
a6bB = np.array([1,1,0,0,0,1])
a6bC = np.array([1,1,0,0,1,0])
a6bD = np.array([1,0,1,0,0,1])
a6bE = np.array([1,0,1,0,1,0])
a6bF = np.array([1,0,1,1,0,0])
if (vetor == a4b0).all():
  return a6b0
if (vetor == a4b1).all():
  return a6b1
if (vetor == a4b2).all():
  return a6b2
if (vetor == a4b3).all():
  return a6b3
if (vetor == a4b4).all():
  return a6b4
if (vetor == a4b5).all():
```

```
        return a6b5
    if (vetor == a4b6).all():
        return a6b6
    if (vetor == a4b7).all():
        return a6b7
    if (vetor == a4b8).all():
        return a6b8
    if (vetor == a4b9).all():
        return a6b9
    if (vetor == a4bA).all():
        return a6bA
    if (vetor == a4bB).all():
        return a6bB
    if (vetor == a4bC).all():
        return a6bC
    if (vetor == a4bD).all():
        return a6bD
    if (vetor == a4bE).all():
        return a6bE
    if (vetor == a4bF).all():
        return a6bF
    return


def general_work(self, input_items, output_items):
    ninput_items = 21844
    noutput_items = 32766
    check = np.empty(0, int)
    in0 = input_items[0][:ninput_items]
    out0 = np.empty(0, int)
    input_range = int(len(in0)/4)
    for i in range(input_range):
        check = self.compare(in0[:4])
        in0 = in0[4:]
```

```
    out0 = np.append(out0,check)
output_items[0][:noutput_items] = out0[:len(output_items[0])]
return len(output_items[0])
```

APÊNDICE D – DECODIFICADOR 4B6B

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Copyright 2022 Pedro Moura.
#
# SPDX-License-Identifier: GPL-3.0-or-later
#

import numpy as np
from gnuradio import gr
import time

class decod_fourtosix(gr.basic_block):
    """
    docstring for block decod_fourtosix
    """
    def __init__(self):
        gr.basic_block.__init__(self,
            name="decod_4B6B",
            in_sig=[np.byte],
            out_sig=[np.byte])

    def compare(self,vetor):
        a4b0 = np.array([0,0,0,0])
        a4b1 = np.array([0,0,0,1])
        a4b2 = np.array([0,0,1,0])
        a4b3 = np.array([0,0,1,1])
        a4b4 = np.array([0,1,0,0])
        a4b5 = np.array([0,1,0,1])
        a4b6 = np.array([0,1,1,0])
        a4b7 = np.array([0,1,1,1])
        a4b8 = np.array([1,0,0,0])
```

```
a4b9 = np.array([1,0,0,1])
a4bA = np.array([1,0,1,0])
a4bB = np.array([1,0,1,1])
a4bC = np.array([1,1,0,0])
a4bD = np.array([1,1,0,1])
a4bE = np.array([1,1,1,0])
a4bF = np.array([1,1,1,1])
a6b0 = np.array([0,0,1,1,1,0])
a6b1 = np.array([0,0,1,1,0,1])
a6b2 = np.array([0,1,0,0,1,1])
a6b3 = np.array([0,1,0,1,1,0])
a6b4 = np.array([0,1,0,1,0,1])
a6b5 = np.array([1,0,0,0,1,1])
a6b6 = np.array([1,0,0,1,1,0])
a6b7 = np.array([1,0,0,1,0,1])
a6b8 = np.array([0,1,1,0,0,1])
a6b9 = np.array([0,1,1,0,1,0])
a6bA = np.array([0,1,1,1,0,0])
a6bB = np.array([1,1,0,0,0,1])
a6bC = np.array([1,1,0,0,1,0])
a6bD = np.array([1,0,1,0,0,1])
a6bE = np.array([1,0,1,0,1,0])
a6bF = np.array([1,0,1,1,0,0])
if (vetor == a6b0).all():
  return a4b0
if (vetor == a6b1).all():
  return a4b1
if (vetor == a6b2).all():
  return a4b2
if (vetor == a6b3).all():
  return a4b3
if (vetor == a6b4).all():
  return a4b4
if (vetor == a6b5).all():
```

```
        return a4b5
    if (vetor == a6b6).all():
        return a4b6
    if (vetor == a6b7).all():
        return a4b7
    if (vetor == a6b8).all():
        return a4b8
    if (vetor == a6b9).all():
        return a4b9
    if (vetor == a6bA).all():
        return a4bA
    if (vetor == a6bB).all():
        return a4bB
    if (vetor == a6bC).all():
        return a4bC
    if (vetor == a6bD).all():
        return a4bD
    if (vetor == a6bE).all():
        return a4bE
    if (vetor == a6bF).all():
        return a4bF
    return


def general_work(self, input_items, output_items):
    ninput_items = 49152
    noutput_items = 21844
    check = np.empty(0, int)
    in0 = input_items[0]
    out0 = np.empty(0, int)
    completer = [1,1,1]
    input_range = int(len(in0)/6)
    for i in range(input_range):
        check = self.compare(in0[:6])
```

```
   in0 = in0[6:]
   out0 = np.append(out0,check)
if (len(in0) != 0):
    in0 = np.append(in0, np.zeros(6-len(in0)))
output_items[0][:] = out0[:len(output_items[0])]
self.consume(0, len(in0))
return len(out0)
```

APÊNDICE E – MODULADOR VPPM

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Copyright 2022 Pedro Moura.
#
# SPDX-License-Identifier: GPL-3.0-or-later
#



import numpy as np
from gnuradio import gr
import time

class VPPM(gr.interp_block):
    def __init__(self, interp=1):
        gr.interp_block.__init__(self,
            name="VPPM",
            in_sig=[np.byte],
            out_sig=[np.complex64],
            interp = interp)
        self.interp = interp

    def work(self, input_items, output_items):
        ppm = np.zeros(self.interp, int)
        #ppm = np.arange(self.interp)*0
        in0 = input_items[0]
        out0 = np.empty(0, int)
        for i in range(len(input_items[0])):
            if (in0[i] == 1):
                ppm = ppm*0
                ppm[self.interp-1] = 1
                out0 = np.append(out0,ppm)
```

```
    else:
        ppm = ppm*0
        ppm[0] = 1
        out0 = np.append(out0,ppm)
output_items[0][:] = out0
return len(output_items[0])
```

APÊNDICE F – DEMODULADOR VPPM

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Copyright 2022 Pedro Moura.
#
# SPDX-License-Identifier: GPL-3.0-or-later
#



import numpy as np
from gnuradio import gr
import time

class VPPM_Decoder(gr.decim_block):
    """
    docstring for block VPPM_Decoder
    """
    def __init__(self, decim_rate=1):
        gr.decim_block.__init__(self,
            name="VPPM_Decoder",
            in_sig=[np.complex64],
            out_sig=[np.byte],
            decim = decim_rate)
        #self.set_relative_rate(1.0/decim_rate)
        self.decimation = decim_rate



    def work(self, input_items, output_items):
        ppm = np.zeros(self.decimation, int)
        in0 = input_items[0]
        bytes = in0.real
        lista = bytes.astype(int)
```

```
out0 = np.empty(0, int)
input_range = int(len(lista)/self.decimation)
for i in range(input_range):
    check = lista[:self.decimation]
    lista = lista[self.decimation:]
    if (check[0] == 1):
        out0 = np.append(out0,0)
    else:
        out0 = np.append(out0,1)
output_items[0][:] = out0
return len(output_items[0])
```