UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RAFAEL HESS ALMALEH

Avaliação do Rádio Definido por Software LimeSDR utilizando a plataforma GNURadio

Porto Alegre

2019

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RAFAEL HESS ALMALEH

Avaliação do Rádio Definido por Software LimeSDR utilizando a plataforma GNURadio

Relatório Final do Projeto de Diplomação II apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para Graduação em Engenharia Elétrica

Orientador: Prof. Dr. Ivan Müller

Porto Alegre
2019

RAFAEL HESS ALMALEH

Avaliação do Rádio Definido por Software LimeSDR utilizando a plataforma GNURadio

Relatório Final do Projeto de Diplomação II apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para Graduação em Engenharia Elétrica

---

Prof. Dr. Ivan Müller
Orientador - UFRGS

Prof. Dr. Luiz Fernando Ferreira
Chefe do Departamento de Engenharia Elétrica (DELET) - UFRGS

Aprovado em 04 de julho de 2019.

BANCA EXAMINADORA

Prof. Dr. Ivan Müller
UFRGS

Prof. Dr. Hamilton Duarte Klimach
UFRGS

Prof. Dr. Tiago Roberto Balen
UFRGS

À minha mãe, Carla, a quem devo tudo que sou.

Agradecimentos

Primeiramente, gostaria de agradecer à Universidade Federal do Rio Grande do Sul por me propiciar uma formação de excelência. Estendo os agradecimentos a todos os servidores e profissionais desta instituição, bem como à sociedade brasileira por ser financiadora de meus estudos. Espero trazer um retorno à altura.

Agradeço, em especial, o Departamento de Engenharia Elétrica, onde passei a maior parte da minha vida nos últimos seis anos e cujos mestres me propiciaram um imenso aprendizado.

Agradeço, também, ao meu orientador Prof. Dr. Ivan Müller, com quem tive ótimo relacionamento não só durante este trabalho, mas também ao longo de diversas cadeiras da graduação. Com certeza as aulas do Prof. Ivan foram decisivas para meu interesse no assunto deste trabalho.

Agradeço aos colegas e amigos, sempre unidos e dispostos a ajudar e cooperar.

No âmbito pessoal, agradeço a Deus e a minha família, em especial a minha mãe, a pessoa que sempre lutou por mim e acreditou no meu potencial.

O aumento do conhecimento é como uma esfera dilatando-se no espaço: quanto maior a nossa compreensão, maior o nosso contato com o desconhecido

Blaise Pascal

Resumo

Este trabalho busca avaliar a utilização do rádio definido por software LimeSDR baseado no circuito integrado LMS7002M . O objetivo é verificar a eficácia da utilização do mesmo junto à plataforma de programação GNU Radio e também avaliar o seu uso em trabalhos futuros de pesquisa e ensino. Para isto, são elaborados estudos de caso práticos, primeiramente em ambiente simulado e posteriormente em ambiente real, comparando os resultados coletados pela plataforma GNU Radio com medições de um analisador de espectro. Posteriormente, são apresentadas as facilidades e as dificuldades encontradas ao longo da utilização das duas plataformas. Os resultados obtidos mostraram-se adequados, com as medições por meio do GNU Radio sendo coerentes com as realizadas pelo analisador de espectro. Ao final, são apresentadas as dificuldades encontradas e, a partir delas, são sugeridos trabalhos futuros.

Palavras-chave: Radio, Software, RDS, QPSK, GFSK, GNURadio.

Abstract

This work aims to evaluate the use of LimeSDR software defined radio based on the LMS7002M integrated circuit. The main goal is to verify the effectiveness of the use of the SDR alongside with the programming platform GNU Radio and also its use in future researches and teaching applications. For this, practical case studies are elaborated, first in simulated environment and later in real applications, comparing the results gathered by GNU Radio with measurements of a spectrum analyzer. Subsequently, the easiness and difficulties encountered during the use of the two platforms are presented. The obtained results were adequate, with the measurements made by GNU Radio being coherent with the ones from the spectrum analyzer. The difficulties encountered are identified and, from them, future work is suggested.

Keywords: Radio, Software, SDR, QPSK, GFSK, GNURadio.

Lista de Figuras

Figura 1 – Diagrama simplificado de um SDR ideal. 17
Figura 2 – Diagrama de um sistema para recepção por um SDR. 17
Figura 3 – Módulos do RDS brasileiro. 18
Figura 4 – Exemplo de diagrama de blocos para recebimento de sinal FM Banda Estreira (NBFM). 20
Figura 5 – Exemplo de modulação BFSK. 21
Figura 6 – Ocupação espectral da modulação BFSK. 22
Figura 7 – Curvas Gaussianas normalizadas. 22
Figura 8 – Representação de um pacote do protocolo 802.11b. Destaque para as partes que utilizam a modulação QPSK. 24
Figura 9 – Constelação de modulação QPSK. 25
Figura 10 – Representação dos 3 canais utilizados pelo 802.11b e do último canal da faixa de 2,4GHz. 27
Figura 11 – Visão geral da GUI do software GNU Radio Companion. 28
Figura 12 – LimeSDR Utilizado. 31
Figura 13 – Diagrama de blocos para GFSK. 33
Figura 14 – Arquivo de texto enviado para teste. 36
Figura 15 – Diagrama de blocos para implementação simulada de envio QPSK. . . 37
Figura 16 – Diagrama de blocos para implementação real de envio QPSK. 40
Figura 17 – Sinal enviado. 41
Figura 18 – Sinal recebido. 41
Figura 19 – Sinal enviado no domínio frequência. 42
Figura 20 – Sinal enviado no domínio frequência com visualização *Waterfall*. 42
Figura 21 – Visualização espectral do sinal GFSK medida no analisador de espectro. 43
Figura 22 – Arquivo de texto recebido. 43
Figura 23 – Sinal GFSK recebido com ruído. 44
Figura 24 – Espectro do sinal GFSK recebido com ruído. 44
Figura 25 – Sinal enviado. 45
Figura 26 – Sinal recebido. 45
Figura 27 – Bits recebidos. 45
Figura 28 – Espectro do sinal enviado. 46
Figura 29 – Espectro do sinal recebido antes e após filtro passa-baixas. 46
Figura 30 – Taxa de erro de *bits*. 46
Figura 31 – Pontos onde há recepção errada de *bits*. 46
Figura 32 – Diagrama IQ enviado. 47
Figura 33 – Diagrama IQ recebido. 47

Figura 34 – *Bytes* recebidos. 47
Figura 35 – *Bits* recebidos. 47
Figura 36 – *Bytes* recebidos para *Threshold* de 0,5. 49
Figura 37 – Taxa de erro de *bits* para *Threshold* de 0,5. 50
Figura 38 – *Bits* recebidos para *Threshold* de 0,5. 50
Figura 39 – Sinalização de troca de *bits* para *Threshold* de 0,5. 50
Figura 40 – Configuração do experimento na bancada do laboratório. 51
Figura 41 – Largura de banda medida pelo analisador espectral via antena. 52
Figura 42 – Largura de banda medida pelo analisador espectral via cabo. 53
Figura 43 – Largura de banda do sinal enviado medida pelo software GNURadio. . 53
Figura 44 – Diagrama de constelação do sinal recebido. 54
Figura 45 – Constelação com ruído. 54
Figura 46 – Representação em frequência com pico de ruído ao centro. 54

Lista de Tabelas

Tabela 1 – Características funcionais do LimeSDR 30
Tabela 2 – Parametrização do bloco *LimeSuite Sink (TX)* 35
Tabela 3 – *Threshold* x Taxa de Erro 49

Lista de Abreviaturas e Siglas

ASCII	*American Standard Code for Information Interchange*
BER	*Bit Error Rate*
BFSK	*Binary Frequency Shift Keying*
BPSK	*Binary Phase Shift Keying*
CMOS	*Complementary Metal Oxide Semiconductor*
CPU	*Central Processing Unit*
dBm	Decibel miliWatt
DECT	*Digital Enhanced Cordless Communication*
DQPSK	*Differential Quadrature Phase Shift Keying*
DSSS	*Direct-Sequence Spread Spectrum*
FIR	*Finite Impulse-Respone*
FLL	*Frequency-Locked Loop*
FM	Frequência Modulada
FPGA	*Field Programmable Gate Array*
GFSK	*Gaussian Frequency Shift Keying*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
LTE	*Long Term Evolution*
MIMO	*Multiple Inputs Multiple Outputs*
Mbps	Megabits por segundo
MSps	*Megasamples per second*
NRZ	*No Return to Zero*

PLL	*Phase-Locked Loop*
QPSK	*Quadrature Phase Shift Keying*
RF	Rádio Frequência
RX	Receptor
SDR	*Software Defined Radio*
SDRAM	*Synchronous Dynamic Random-Access Memory*
SISO	*Single Input Single Output*
TX	Transmissor
USAF	*United States Air Force*
USB	*Universal Serial Bus*
WCDMA	*Wide-Band Code-Division Multiple Acces*
WECA	*Wireless Ethernet Compatibility Alliance*
Wi-Fi	*Wireless Fidelity*
XML	*Extensible Markup Language*
XOR	*Exclusive Or*

Sumário

1 INTRODUÇÃO 14

2 REFERENCIAL TEÓRICO 16
2.1 Rádio Definido por Software 16
2.2 GNU Radio 18
2.3 GFSK 20
2.4 QPSK - 802.11b 22

3 MATERIAIS E MÉTODOS 28
3.1 Materiais 28
3.1.1 Software GNU Radio 28
3.1.2 Computador 30
3.1.3 Rádio LimeSDR 30
3.1.4 Analisador de Espectro 31
3.2 Métodos 31

4 ESTUDOS DE CASO 33
4.1 Estudo de Caso 1: Modulação GFSK 33
4.2 Estudo de Caso 2: Modulação QPSK 36
4.2.1 Modo Simulado 36
4.2.2 Implementação Física 39

5 RESULTADOS E DISCUSSÃO 41
5.1 GFSK 41
5.2 QPSK - Simulado 44
5.3 QPSK - Implementação Física 51

6 CONCLUSÃO 55

7 TRABALHOS FUTUROS 57
7.1 DSSS 57
7.2 Blocos GNU Radio 58
7.3 LimeSDR 58

REFERÊNCIAS BIBLIOGRÁFICAS 60

1 Introdução

A partir da invenção do rádio, o modo como conhecemos a comunicação mudou radicalmente. As distâncias entre as pessoas foram reduzidas drasticamente. O que antes levava dias ou meses para se propagar, passou a chegar a quase todo o globo quase que instantaneamente. O interesse da sociedade pela velocidade de propagação da informação aumentou, porém, com a informação sendo transmitida livremente pelo ar, começaram a surgir preocupações com a confidencialidade das mesmas, principalmente no âmbito militar.

Dessa preocupação, surgiram diversas tecnologias na área das telecomunicações, dentre elas o Rádio Definido por Software (SDR, na sigla em inglês). Esta é uma ferramenta versátil que vem saindo da esfera militar e sendo empregada na vida civil, assim como ocorreu com o GPS, por exemplo.

Graças a sua versatilidade, o SDR foi adotado na academia e na indústria como ferramenta de desenvolvimento e prototipagem. O fato de ser reprogramável permite que os custos com equipamento sejam mínimos e as possibilidades de exploração sejam aumentadas. Estas características fazem dos Rádios Definidos por software uma excelente plataforma para uso em simulações e prototipagens, etapas essenciais na área de pesquisa e desenvolvimento. Além disso, apresentam a base para implementação de rádios cognitivos, dispositivos que podem se reconfigurar automaticamente conforme as condições do entorno, evitando interferências e garantindo máxima eficiência na transmissão de dados. Para o consumidor final, existem as vantagens de um equipamento mais barato, pois com o SDR os custos de produção e desenvolvimento podem ser reduzidos, com o compartilhamento de hardware para diferentes aplicações. Pode-se citar, ainda, o uso acadêmico, com o SDR permitindo ao aluno explorar diferentes protocolos de comunicação, gerar sinais de rádio para testes em laboratório, receber sinais de procolos usuais como GPS e até experimentar com novas tecnologias, como a Internet das coisas. Pelo lado da Instituição de Ensino, temos a vantagem de que não será mais necessário gastar com hardware diversos para aplicações específicas, gerando uma economia importante sem perda de capacidade de desenvolvimento.

Recentemente, foram adquiridos dois Rádios Definidos por Software LimeSDR da LimeMicrosystems pela UFRGS. Estudos de caso foram implementados junto com o software GNU Radio, uma plataforma aberta que permite uma programação simples e intuitiva do SDR por meio de diagramas de blocos, características que aceleram mais ainda o desenvolvimento. Como pano de fundo, o principal estudo de caso para esta monografia foi a proposta de implementação de parte da camada física do protocolo 802.11b.

O objetivo deste trabalho é analisar este novo equipamento e verificar sua utilidade como ferramenta de pesquisa e ensino junto a Universidade e também como plataforma de desenvolvimento, apresentando o seu potencial e as limitações encontradas, servindo de base para trabalhos futuros no assunto.

A escolha deste tema e estudo de caso deu-se pelas razões expostas a seguir. Cada vez mais, a sociedade demanda dispositivos portáteis, potentes e versáteis, exigindo funcionalidades que consumam o mínimo de energia possível, para o desenvolvimento de dispositivos alimentados à bateria, cada vez menores e mais leves. Estima-se que o consumo de energia por dispositivos ligados a redes sem fio chegue a 10% da geração global de energia em 2020 (SMIL, 2016).

Em termos de comunicação, existem pelo menos quatro diferentes rádios em um celular, como o rádio para telefonia, rádio FM, Bluetooth, Wi-Fi, GPS entre outros. Todas essas funcionalidades acabam por aumentar o consumo e consequentemente diminuem a autonomia dos dispositivos.

A nova geração das telecomunicações, o 5G, vai permitir a conexão em rede de diversos dispositivos que vão tornando-se *smarts*. Assim como ocorreu com os telefones celulares e com a televisão, a tendência é que outros aparelhos tornem-se inteligentes, dos eletrodomésticos aos automóveis. Essa conexão em massa exige que as redes 5G sejam rápidas, seguras, escaláveis, ágeis e sustentáveis.

Nesse sentido, os SDRs apresentam uma boa solução: o fato de serem reprogramáveis permite que se tenha diversas soluções em um hardware único, de modo que haja menor consumo e melhor aproveitamento de espaço. Além disto, eles podem estar sempre otimizados com os algoritmos mais avançados, bastando um *download* para configurar o dispositivo. Não há dúvida de que os rádios definidos por software serão protagonistas dessa revolução nas telecomunicações.

Soma-se a esses fatores o fato de que os rádios definidos por software são uma boa alternativa para as Universidades públicas, que muitas vezes, graças a orçamentos insuficientes, têm dificuldades em fornecer uma estrutura ideal de laboratório e equipamentos para os pesquisadores.

A escolha do estudo de caso do protocolo 802.11b deu-se pelo fato da popularidade do mesmo no mundo das telecomunicações. Além disto, o desenvolvimento deste trabalho apresenta a possibilidade de ser aproveitado para uso no laboratório como plataforma de teste e simulação de tráfego nos diferentes canais que o pesquisador desejar, garantindo uma contribuição ao trabalho de outros pesquisadores que venham a trabalhar com esta tecnologia.

2 Referencial Teórico

2.1 Rádio Definido por Software

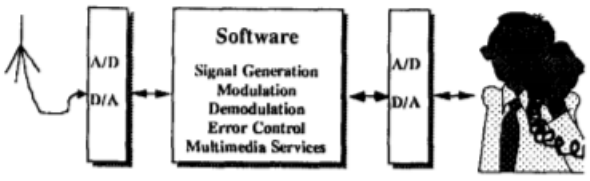
Dada a importância da comunicação nos conflitos mundiais, as forças armadas de diversos países serviram como força motriz importante para o desenvolvimento das telecomunicações. Podemos citar como exemplo desde a invenção do receptor superheterodino por Edwin Armstrong na Primeira Grande Guerra (LATHI; DING, 2010), até a ARPANET, embrião da Internet, financiada pelo Departamento de Defesa Americano.

Os Rádios Definidos por software seguiram um caminho semelhante. Eles tiveram seus primeiros passos dados nos Estados Unidos da América na década de 1970, culminando no SPEAKeasy para uso pela USAF (Força Aérea dos Estados Unidos, na sigla em inglês), em 1991. A ideia da USAF ao adotar o SPEAKeasy era garantir um hardware robusto para fins militares e ser compatível com as tecnologias de modulação do futuro (COOK; BONSER, 1999). O equipamento deveria ser compatível com dez protocolos militares de comunicação e operar em uma faixa de frequência indo dos 2MHz aos 2GHz. O dilema do Departamento de Defesa dos Estados Unidos da América, quando decidiu iniciar o programa SPEAKeasy, era como manter uma comunicação com os seus aliados garantindo suporte global, ser robusto às interceptações de nações inimigas e ainda dar conta das rápidas mudanças de tecnologia, tudo isso sem estourar o orçamento militar (LACKEY; UPMAL, 1995).

Os SDRs consistem em um sistema de comunicação em que os hardware tradicionais - como misturadores, filtros, amplificadores e outros - passam a ser implementados em software ao ter-se um hardware único programável, como um FPGA (*Field Programmable Gate Array*), que pode ser configurado para desempenhar diversas funções. A definição oficial do SDR é: “Radio em que algumas ou todas as funções da camada física são definidas por software” (SDRFORUM, 2007). Desta forma, consegue-se atualizar facilmente os dispositivos e garantir interoperabilidade, sem grandes custos.

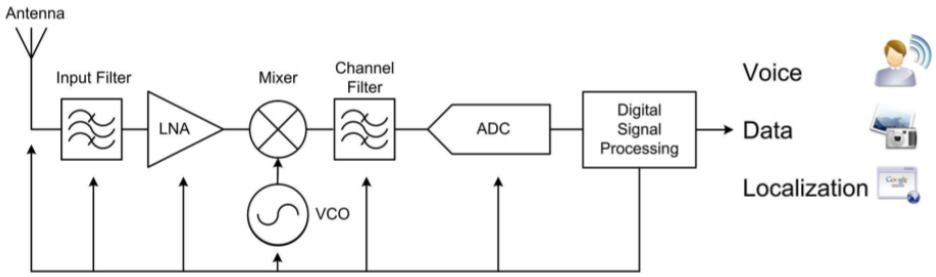
O termo *Software Radio* tem sua alcunha atribuida a Joseph Mitola III, no seu artigo *Software Radio: Survey, Critical Analysis and Future Directions* (MITOLA, 1992), apesar da empresa E-Systems (atual Raytheon), na qual Mitola trabalhava, ter usado o termo em um boletim informativo já em 1984. Na figura 1, pode-se ver um modelo ideal de SDR proposto por Mitola e na figura 2, têm-se um diagrama mais realista do para a recepção por um SDR.

Figura 1 – Diagrama simplificado de um SDR ideal.



Fonte: retirado de Mitola (1992).

Figura 2 – Diagrama de um sistema para recepção por um SDR.



Fonte: retirado de Rivet et al. (2009).

A programação de um Rádio Definido por Software pode ser realizada por diversas ferramentas dedicadas, algumas de uso livre, como o GNURadio, e outras de uso proprietário, como o LabVIEW, da National Instruments, e o Simulink, da MathWorks.

Para se ter uma ideia do potencial de utilização dos SDRs, Staple e Werbach (2004) citam duas hipóteses que poderiam revolucionar o uso dos dispositivos móveis: como um telefone celular que possa baixar um novo software da internet e tornar-se capaz de receber sinais digitais de televisão, sem nenhuma mudança de hardware; e também a ideia de cooperação entre rádios cognitivos, em que o aparelho celular analisaria as transmissões ao redor e adaptaria a sua transmissão para minimizar interferências.

Os SDRs também apresentam grandes vantagens à indústria desenvolvedora de hardware, como o fato de que uma solução de um único *chip* consegue atingir três aspectos econômicos importantes, como elencam Rivet et al. (2009):

1. Custo de desenvolvimento reduzido (tecnologia barata, como CMOS, é utilizada);

2. Custo energético reduzido (um único *chip* implica uma redução de consumo energético);

3. Custo de pessoal reduzido (todos os engenheiros trabalham em um único projeto).

As forças armadas brasileiras trabalham também na concepção de um SDR. Denominado de RDS-Defesa, o programa é capitaneado pelo Centro de Tecnologia do Exército (CTEx) e faz parte do projeto estratégico de defesa cibernética, dois módulos de RF podem ser vistos na figura 3. O RDS-Defesa tem como objetivo entregar às forças armadas um rádio para uso militar capaz de unificar a comunicação das três forças, sendo passível de atualizações sempre que necessário (CTEX, 2019).

Figura 3 – Módulos do RDS brasileiro.

Fonte: retirado de DefesaTV (2019).

Dentre as novas áreas em que os rádios definidos por software vêm sendo aplicados, pode-se citar o trabalho de Machado (2018), em que um dispositivo SDR foi utilizado para o envio e captação de pacotes ADS-B, um protocolo de comunicação utilizado como sistema de vigilância para aeronaves. No trabalho citado, foram utilizados dois SDRs USRP da National Instruments.

2.2 GNU Radio

O GNU Radio é um conjunto de ferramentas de software que fornece blocos processadores e geradores de sinais para a programação de rádios definidos por software (GNURADIO, 2018b). Pode ser usado tanto com um hardware de RF externo ou apenas como plataforma de simulação.

O desenvolvimento inicial da plataforma deu-se em 2001 por Eric Blossom, responsável pela criação do código e gestão do projeto, graças a um aporte financeiro do entusiasta do projeto GNU, John Gilmore (NUTAQ, 2019). Atualmente, ele é mantido pela *GNU Radio Foundation* e por milhares de colaboradores independentes que contribuem com o desenvolvimento desta plataforma aberta (GNURADIO, 2018b).

GNU Radio Companion é o nome dado à interface gráfica que permite a construção de sistemas por meio de diagramas de blocos, o que deixa o trabalho mais rápido e intuitivo, com uma estrutura similar aos blocos da linguagem XML. Um exemplo do software e de diagrama de blocos do GNU Radio pode ser visto na figura 4.

Pode-se compor um diagrama de blocos utilizando os componentes pré-existentes na ferramenta, mas também pode-se programar os próprios blocos para que desempenhem uma função específica, garantindo um universo de possibilidade de desenvolvimento.

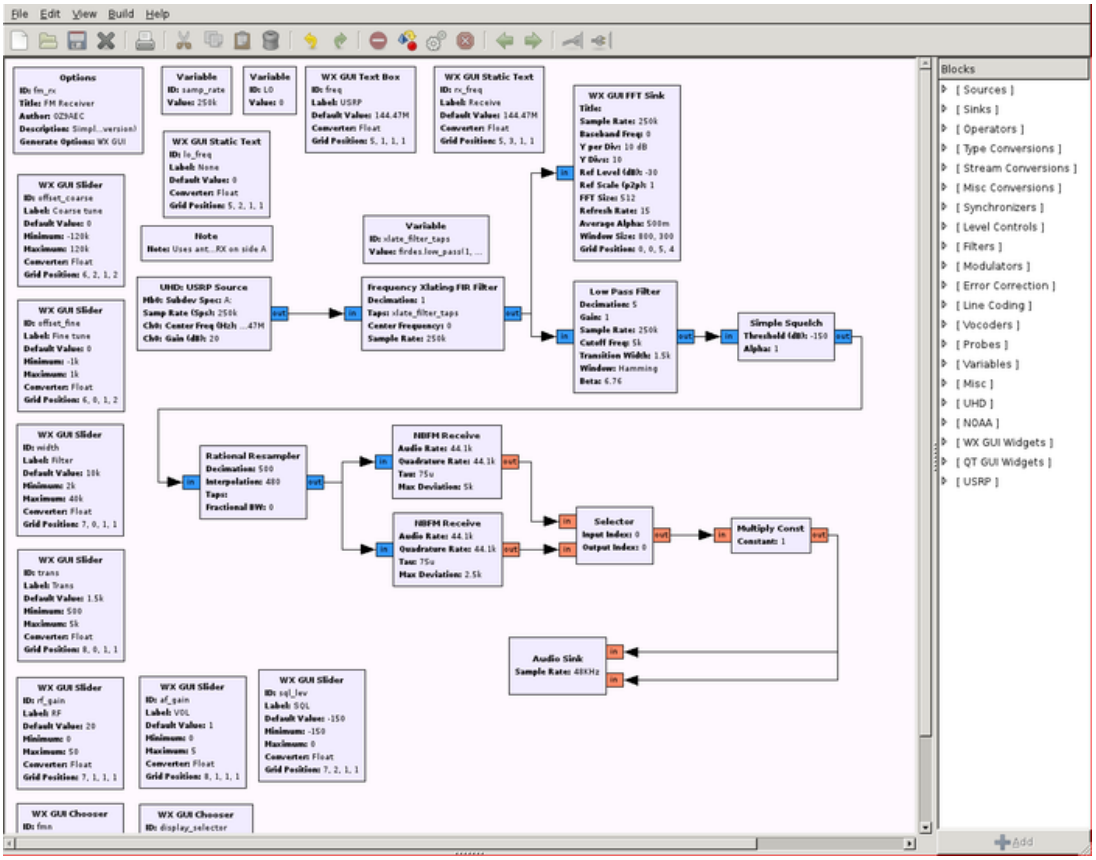
A interface com hardware é realizada através de blocos que funcionam como parametrizadores dos equipamentos. Dentre estes, pode-se citar os USRP, LimeSDR e RTL-SDR. O primeiro é utilizado para a linha de SDRs USRP (*Universal Software Radio Peripheral*) da Ettus Research, uma divisão da National Instruments dedicada aos rádios definidos por software. O segundo é dedicado ao SDR da LimeMicrosystem, que foi o utilizado neste projeto. Finalmente, o terceiro é um bloco do projeto Osmocom (*Open source mobile communications*), que promove o desenvolvimento de ferramentas de uso livre para entusiastas e profissionais que queiram trabalhar com a área de telecomunicações. A sua biblioteca para rádios definidos por software, RTL-SDR, oferece suporte a uma grande variedade de rádios, o que o faz ser popular.

Cada uma destas famílias de SDRs citadas acima deve possuir, no mínimo, dois tipos de blocos para parametrizar um rádio, os *Sources* e os *Sinks*. O primeiro, do inglês, significa a fonte, ou seja, é o bloco responsável pela transmissão do sinal, pelo envio de dados (TX). O segundo apresenta a função oposta, sendo o sumidouro, em tradução livre, ou seja, é responsável pela captação do sinal (RX).

Pelo fato de ser um software livre, existem diversos exemplos e bibliotecas de funções para o GNU Radio que são compartilhadas em repositórios da ferramenta GitHub. Esta é uma ferramente de hospedagem de códigos com controle de versão que permite que usuários cadastrados possam contribuir com o desenvolvimento de um projeto, sendo ele privado ou de uso livre (GITHUB, 2019). Foi adquirido pela Microsoft em junho de 2018 pela quantia de US$ 7,5 bilhões (CENTER, 2019).

Como o GNU Radio é uma ferramente dedicada a desenvolvedores, é essencial que cada bloco existente, ou a ser criado, contenha uma documentação que permita o acesso e compreensão por futuros utilizadores. O local onde pode-se ter acesso a essas informações é o *GNU Radio Manual and C++ API Reference* (GNURADIO, 2019a). Esta plataforma é gerada por meio da ferramenta Doxygen, uma aplicação de uso livre que permite criar documentação a partir de comentários dentro do código-fonte (DOXYGEN, 2019). No manual do GNU Radio encontram-se todas as classes que fazem parte dos blocos nativos, no entanto, muitas vezes a descrição fornecida não é suficiente para a total compreensão da função do bloco. Nestes casos, dado a popularidade do GNU Radio, uma pesquisa em

Figura 4 – Exemplo de diagrama de blocos para recebimento de sinal FM Banda Estreira (NBFM).



Fonte: retirado de GNURadio (2019b).

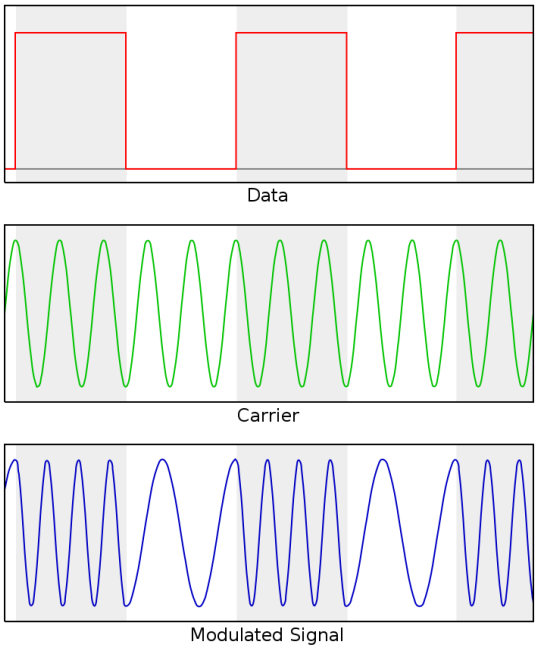
fóruns de desenvolvedores, como o Stack Overflow (STACKOVERFLOW, 2019), permite achar exemplos concretos de utilização, o que ajuda no entendimento do bloco.

2.3 GFSK

Uma das técnicas de modulação de portadora de RF empregadas neste trabalho é o GFSK (*Gaussian Frequency-shift Keying*), ou chaveamento em frequência usando filtro gaussiano, na tradução em português. A modulação por chaveamento em frequência é uma forma de enviar dados digitais através de mudanças discretas de frequência de uma onda portadora (KENNEDY; DAVIS, 1992) em que cada símbolo corresponde a um valor de frequência. No caso mais simples, o BFSK, tem-se dois valores diferentes, um para o símbolo 1 e outro para o 0, conforme pode-se ver na figura 5.

O termo gaussiano refere-se à adoção de um filtro deste tipo para que as transições de frequência a cada mudança de símbolo não sejam tão abruptas. Uma mudança abrupta acarreta numa maior ocupação do espectro de RF, o que pode causar interferências, dado

Figura 5 – Exemplo de modulação BFSK.



Fonte: retirado de Wikipedia (2019a).

as componentes de alta frequência contidas no chaveamento. Um filtro gaussiano apresenta como resposta ao impulso uma função gaussiana. Eles são muito usados para modelagem de pulsos justamente por sua característica de apresentarem uma transição mais suave.

Para um BFSK, temos os seguintes sinais no domínio tempo para o símbolo 1 e para o símbolo 0, respectivamente:

$$S_H(t) = A\cos(2\pi f_H t) \tag{2.1}$$

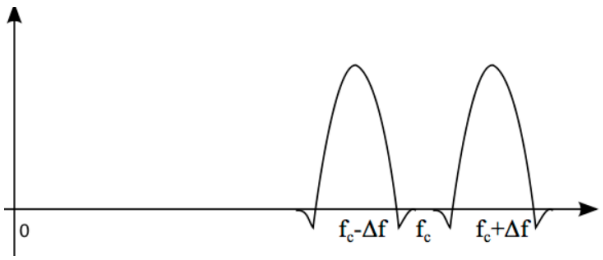
$$S_L(t) = A\cos(2\pi f_L t) \tag{2.2}$$

Onde $f_H = f_C + \Delta f$, $f_L = f_C - \Delta f$ e T_b é o tempo de duração de cada bit.

O espectro ocupado por um sinal FSK, para frequências positivas e com $2\Delta f T_b \gg 1$ pode ser visto na figura 6.

Com a largura de banda sendo, então $2\Delta f + 2B$, onde B é a largura do sinal de banda base. Ao passarmos os bits do sinal de banda base por um filtro gaussiano, conseguimos eliminar as componentes de alta frequência, diminuindo o valor de B. Para uma onda quadrada ideal, a largura de banda seria composta de infinitas harmônicas da frequência fundamental, o que é impossível. Com um filtro gaussiano atuando como

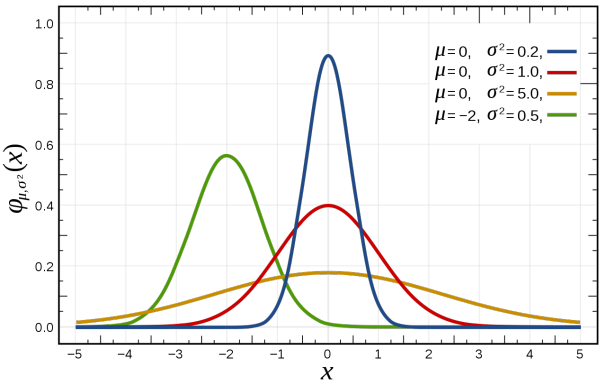
Figura 6 – Ocupação espectral da modulação BFSK.



Fonte: retirado de Berkeley (2019).

passa-baixas, podemos limitar essa largura de banda. A frequência de corte do filtro gaussiano é definida como $f = \frac{1}{2\pi\sigma}$, onde σ é o desvio padrão da distribuição gaussiana.

Figura 7 – Curvas Gaussianas normalizadas.



Fonte: retirado de Wikipedia (2019b).

Na figura 7, podemos observar que quanto menor o valor de σ, mais abrupta a subida, aproximando-se do comportamento de uma onda quadrada. Esta análise corresponde à definição da frequência de corte do filtro gaussiano, que torna-se mais elevada para um pequeno valor de σ.

A modulação GFSK é utilizada em diversas aplicações, como o protocolo Bluetooth e DECT (*Digital Enhanced Cordless Communications*, utilizado em telefones sem-fio) (ETSI, 2019).

2.4 QPSK - 802.11b

O protocolo de comunicação sem fio 802.11b, empregado pela organização Wi-Fi, está plenamente difundido nos dispositivos móveis como celulares e tablets. Apesar de

existirem versões mais avançadas do protocolo 802.11, a versão denominada "b" é ainda muito importante, tendo sido amplamente adotada graças a sua relação custo benefício elevada, garantindo redução de custo da tecnologia e alta taxa de transmissão de dados (TEKTRONIX, 2013).

O 802.11 é um protocolo de comunicação sem fio estabelecido pelo IEEE em 1997. A versão "b" foi estabelecida em 1999, utilizando a frequência de transmissão de 2,4GHz e capaz de atingir uma taxa de transmissão de 11Mbps (IEEE, 1997). Concomitantemente à sua criação, foi estabelecida a versão "a", que opera em uma frequência maior (5GHz) e também garante uma taxa de transferência de dados mais elevada (54Mbps). Uma das desvantagens da versão "b" é o fato de a mesma operar na frequência de diversos aparelhos domésticos, como o forno de micro-ondas e telefones sem-fio, aumentando a chance de ruídos no sinal. Apesar de apresentar um desempenho inferior, a versão "b" alcançou maior difusão entre os utilizadores pelo fato de ser mais barata e por ter sido lançada antes no mercado que a versão "a".

Em 2000, a *Wireless Ethernet Compatibility Alliance* (WECA), criada um ano antes, lançou o selo *Wireless-Fidelity* (Wi-Fi) como uma certificação junto aos fabricantes (WECA, 2019). Com o passar do tempo, Wi-Fi tornou-se o termo utilizado como sinônimo para as tecnologias 802.11.

A família 802.11 teve seu primeiro sucesso comercial, em 1999, quando a Apple Inc. resolveu incluir conectividade sem-fio à internet em seus computadores portáteis iBooks, com o nome de AirPort (HETTING, 2019).

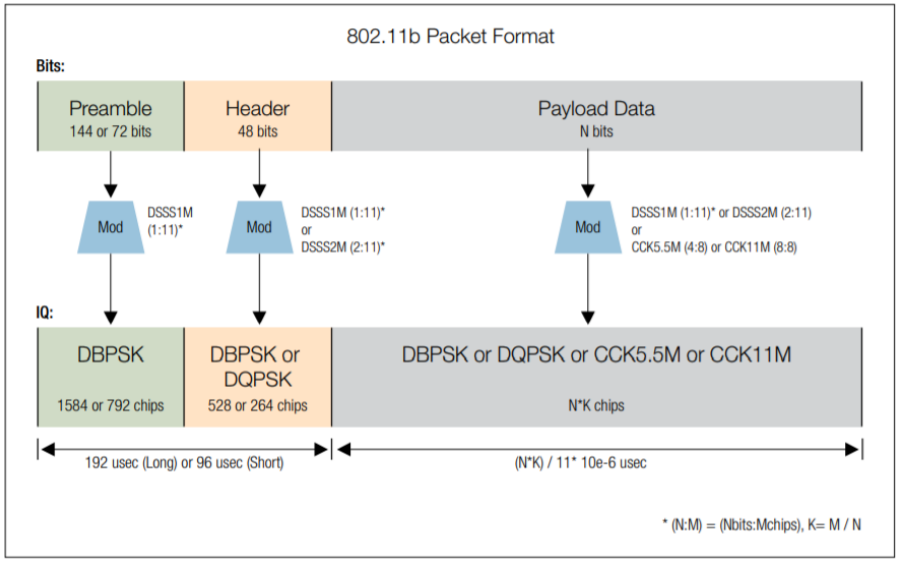
A camada física do protocolo 802.11b utiliza modulação DQPSK (*Differential Quadrature Phase Shift Keying*) para envio dos dados, bem como a técnica DSSS (*Direct-Sequence Spread Spectrum*) para garantir maior robustez do protocolo em face aos diversos ruídos existentes na faixa dos 2,4GHz. Modulação QPSK pode ser utilizada na transmissão do *Header* e dos bits de dados, com o preâmbulo sendo enviado utilizando a técnica BPSK, conforme pode-se ver na estrutura do pacote na figura 8.

A técnica de modulação em fase QPSK consiste na representação em constelação por quatro símbolos diferentes deslocados em fase, com cada símbolo contendo 2 *bits* de informação. PSK significa chaveamento por mudança de fase, indicando que a portadora utilizada tem a sua fase variada pelo sinal enviado. A expressão *Quadrature* é utilizada, pois os seus símbolos estão deslocados de 90° um do outro, ou seja, estão representados em quadratura. Finalmente, o termo *Differential* é empregado, pois o que é medido são as diferenças de fase entre um símbolo e outro, não havendo um referencial absoluto.

Pode-se representar matematicamente a modulação QPSK por meio da seguinte equação:

$$s(t) = A\cos(2\pi f_c t + \theta_n), 0 \leq t \leq T_{sym}, n = 1, 2, 3, 4 \tag{2.3}$$

Figura 8 – Representação de um pacote do protocolo 802.11b. Destaque para as partes que utilizam a modulação QPSK.



Fonte: retirado de Tektronix (2013).

onde a fase do sinal é dada por $\theta_n = (2n-1)\frac{\pi}{4}$.

A equação acima pode ser re-escrita na forma: $s(t) = A\cos\theta_n \cos(2\pi f_c) - A\sin\theta_n \sin(2\pi f_c)$, onde verificamos a ocorrência de duas funções ortogonais, seno e cosseno. Deste modo, em um sistema bidimensional possuindo a função cosseno como abscissa e a função seno como ordenada, os quatro símbolos são representados de acordo com os valores de $A\cos\theta_n$ e $A\sin\theta_n$.

O mapeamento dos quatro símbolos utiliza codificação Gray, em que elementos adjacentes diferem apenas de um *bit*. Deste modo, o sistema apresenta uma maior imunidade a ruído, pois uma interpretação errada de um símbolo acarreta apenas o erro de recepção um único *bit*, conforme pode-se ver na constelação da figura 9.

Uma modulação BPSK é representada por:

$$s(t) = A\cos(2\pi f_c t + \theta_n), n = 0, 1 \tag{2.4}$$

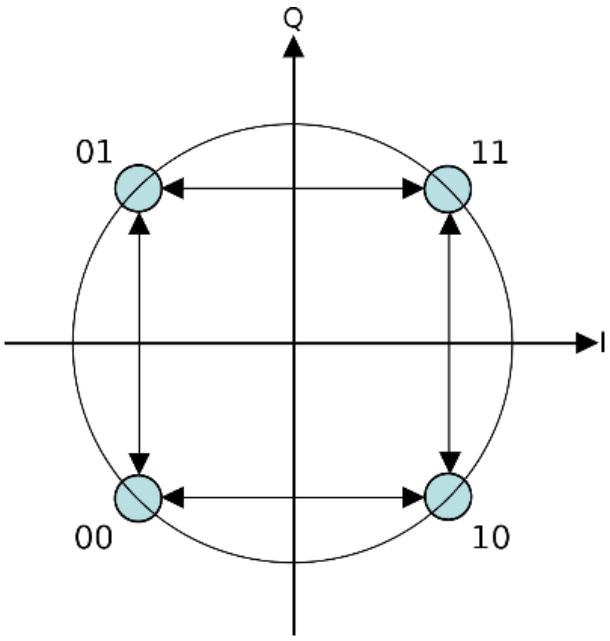
onde $\theta_n = n\pi$. Ou seja, há uma diferença de π radianos entre os símbolos. Considerando os eixos ortogonais $\cos(2\pi f_c t)$, $\sin(2\pi f_c t)$ e que:

$$A\cos(2\pi f_c t + \pi) = A\cos\pi\cos(2\pi f_c t) - A\sin\pi\sin(2\pi f_c t) = -A\cos(2\pi f_c t)$$

e $A\cos(2\pi f_c t)$, temos os pontos mapeados em +A e -A do eixo $\cos(2\pi f_c t)$.

Se considerarmos ainda outro BPSK descrito pela mesma equação, porem com uma defasagem de $\pi/2$, teríamos os pontos mapeados em:

Figura 9 – Constelação de modulação QPSK.



Fonte: retirado de QPSK... (2018).

$A\cos(\pi + \pi/2)\cos(2\pi f_c t) - A\sin(\pi + \pi/2)\sin(2\pi f_c t) = A\sin(2\pi f_c t)$ e

$A\cos(2\pi f_c t + \pi/2) = A\cos(\pi/2)\cos(2\pi f_c t) - A\sin(\pi/2)\sin(2\pi f_c t) = -A\sin(2\pi f_c t)$,

com os pontos mapeados em +A e -A do eixo $\sin(2\pi f_c t)$. Pode-se, então, interpretar o QPSK como um duplo BPSK, onde um *bit* corresponde a um símbolo. Enquanto o tempo de envio de um símbolo BPSK corresponde ao tempo de um *bit*, o tempo de envio de um símbolo QPSK corresponde a dois *bits*. Assim, o QPSK apresenta um desempenho superior, pois consegue transmitir o dobro de *bits* para uma mesma largura de banda em relação ao BPSK, ou, para uma mesma taxa de dados, consegue utilizar metade do espectro necessitado pelo BPSK.

Uma modulação BPSK pode ser atingida utilizando a seguinte sequência: os dados digitais são codificados na forma NRZ, onde os *bits* 1 são representados por uma tensão positiva e os *bits* 0 por uma tensão negativa. Em seguida há a modelagem dos pulsos e então mistura-se o sinal com a portadora $\cos(2\pi f_c t)$.

Para o QPSK, o caminho é similar. O sinal digital de entrada codificado em NRZ é dividido em dois: *bits* pares e ímpares. O primeiro grupo é misturado com a portadora em fase (I) $\cos(2\pi f_c t)$ e o segundo grupo é mixado com a portadora em quadratura (Q) $\sin(2\pi f_c t)$. Finalmente, o sinal final é composto por $I - Q$.

No lado do receptor, o sinal BPSK é misturado novamente com a portadora para recepção síncrona, levando o sinal para a banda-base. Após isto, um comparador determina

se o *bit* recebido corresponde a um 0 ou um 1. Analogamente à transmissão, a recepção QPSK funciona como dois BPSK em paralelo. O sinal recebido é misturado tanto com a portadora em fase quanto com a em quadratura. Como as componentes I e Q são ortogonais, ao misturarmos o sinal com a portadora em fase, recuperamos os *bits* pares que haviam sido modulados no transmissor; e ao misturarmos o sinal com a portadora em quadratura, recuperamos os *bits* ímpares. Ao final, os *bits* são combinados recuperando-se o sinal enviado.

Em relação à taxa de erro de *bits* (BER, na sigla em inglês), temos que, para um ruído branco (ou gaussiano), a BER de uma modulação BPSK apresenta a seguinte probabilidade(STERN; MAHMOUD, 2004):

$$P_e = \frac{1}{2} erfc(\sqrt{\frac{E_b}{N_0}}) \tag{2.5}$$

Onde *erfc* é a função erro complementar definida como: $erfc(x) = 1 - \frac{1}{\sqrt{\pi}} \int_0^x e^{-t^2}\ \mathrm{d}t$;

E_b é a energia por *bit*;

N_0 é o dobro da densidade espectral de potência do ruído.

Como um símbolo BPSK possui apenas um *bit*, essa também é a probabilidade de erro de símbolo.

Como a modulação QPSK pode ser vista como dois envios BPSK independentes, a probabilidade de erro de *bits* é a mesma. No entanto, para atingir essa mesma probabilidade, a QPSK utiliza o dobro da potência utilizada pela BPSK. Considerando uma razão sinal-ruído alta, a taxa de erro de símbolo de uma modulação QPSK pode ser aproximada pela expressão (MEGHDADI, 2008):

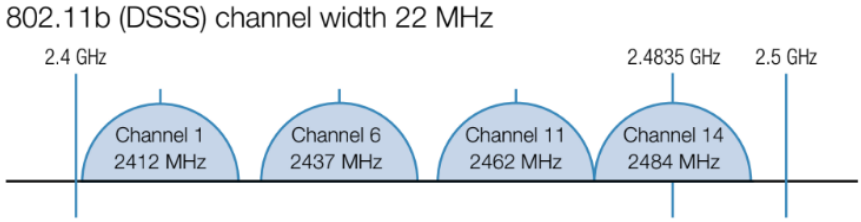
$$P_s \approx 2(\frac{1}{2} erfc(\sqrt{\frac{E_b}{N_0}})). \tag{2.6}$$

Ou seja, apesar de se ganhar em maior velocidade de transmissão ou uso de espectro, perde-se do lado da probabilidade de erro de símbolos. Fisicamente isso pode ser interpretado como consequência da separação entre os símbolos. Na modulação BPSK, os símbolos estão o mais separado possível. No entanto, a medida em que aumentamos a quantidade de símbolos, a distância entre eles vai diminuindo, isto é, a fronteira entre eles fica cada vez mais estreita.

A técnica DSSS por sua vez, é implementada multiplicando-se o sinal a ser enviado por uma sequência pseudoaleatória composta de 1 e -1, aumentando consideravelmente a largura de banda do sinal, de forma proposital, para adicionar robustez à comunicação. A forma de onda resultante tem a aparência de um ruído branco, porém o sinal original

pode ser recuperado no destino ao multiplicar-se o sinal recebido pela mesma sequência pseudoaleatória, uma vez que 1*1=1 e (-1)*(-1)=1.

Figura 10 – Representação dos 3 canais utilizados pelo 802.11b e do último canal da faixa de 2,4GHz.



Fonte: retirado de Tektronix (2013).

Ao multiplicarmos o sinal a ser enviado, promove-se o espalhamento do espectro, de onde vem a expressão *Spread-Spectrum* (espectro espalhado). Se uma outra transmissão é feita no mesmo canal que o sinal enviado por DSSS, porém com uma sequência pseudo-aleatória diferente, o sinal recebido, ao ser multiplicado pela sequência original, não irá apresentar efeitos desses outros sinais, apresentando grande taxa de rejeição a ruído.

O protocolo 802.11b utiliza 3 dos 14 canais disponíveis na faixa dos 2,4GHz. Os canais utilizados não se sobrepõem (canais 1, 6 e 11 da figura 10) e possuem largura de banda de 22MHz, com espaçamento de 5MHz entre canais adjacentes (TEKTRONIX, 2013).

3 Materiais e Métodos

Neste capítulo serão apresentados os materiais e os métodos utilizados no desenvolvimento deste trabalho. O desenvolvimento deu-se ao longo de dois semestres letivos, indo de agosto de 2018 até julho de 2019.

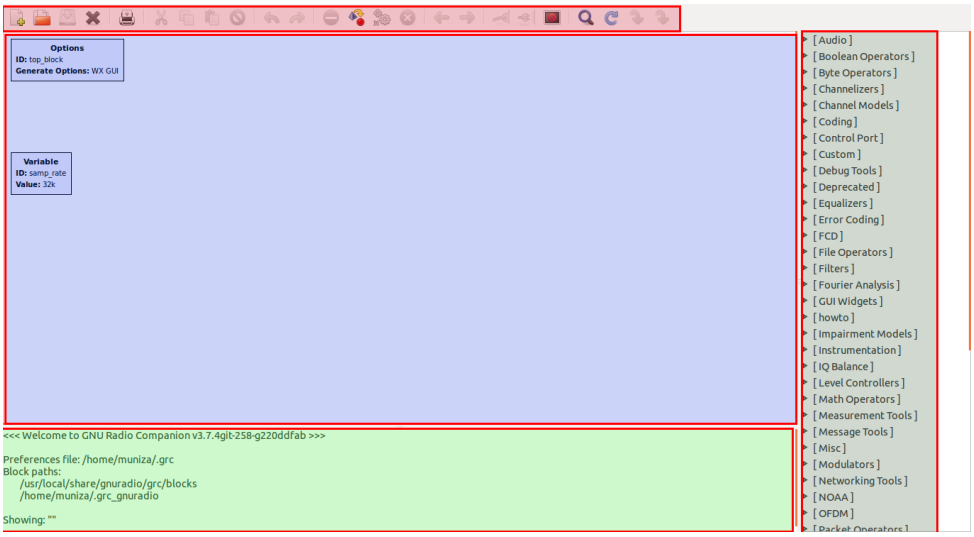
3.1 Materiais

Os materiais utilizados no desenvolvimento deste trabalho foram o rádio definido por software LimeSDR, o software GNU Radio, um computador pessoal do tipo *notebook* e um analisador de espectro. A seguir, cada um deles será detalhado.

3.1.1 Software GNU Radio

O software utilizado nesta análise foi o GNU Radio Companion.

Figura 11 – Visão geral da GUI do software GNU Radio Companion.



Existem quatro regiões básicas no software, conforme figura 11. Ao centro, vemos o ambiente de desenvolvimento, onde realizamos as parametrizações dos blocos e as conexões entre eles. Todo o diagrama de blocos do GNU Radio necessita de um bloco obrigatório: o bloco *top_block*. Nesse bloco é onde são escolhidas algumas opções do diagrama, como seu nome e tipo de interface gráfica a ser utilizada (WX ou QT GUI). A interface gráfica do tipo QT é a mais recomendada, com a WX permanecendo para fins de compatibilidade com diagramas anteriores. Além disso, o bloco *samp_rate* é praticamente uma presença

constante nos diagramas. Ele é responsável por determinar a frequência de amostragem dos sinais.

Os blocos possuem a seguinte estrutura: apresentam entrada e saída, que podem ser únicas ou múltiplas; e parâmetros, que variam para cada bloco. As entradas e saídas obedecem um código de cor que facilita a identificação do tipo de variável que deve ser injetada ou coletada. As cores a serem detalhadas serão baseadas nas variáveis mais utilizadas. A cor roxa indica que a variável é do tipo *char*, composta de 8 *bits*, podendo ir de -127 a 128, em base decimal. Caso queira-se utilizar apenas números positivos, pode-se colocar o bloco *SignedToUnsigned*, transformando a informação em 0 a 255, em base decimal. A cor amarela corresponde a variável do tipo *short*, composta de 16 *bits*. A cor verde representa variáveis do tipo *int*, que contém 32 *bits*. As variáveis *float* são representadas por 32 *bits* com ponto flutuante. Finalmente, o tipo *complex* é adequado para representação de sinais complexos, deste modo ele é constituído de uma composição de duas variáveis *float* de 32 *bits*.

O bloco *samp_rate* não tem o mesmo sentido físico da taxa de amostragem, ele serve apenas como parâmetro para o GNU Radio saber quantas amostras o período de uma onda sinusoidal possui para desenhá-la adequadamente, por exemplo. Sem a presença de um hardware, o GNU Radio sempre enviará seus dados na maior velocidade que ele conseguir, o que depende do poder computacional da máquina a ser utilizada. Deste modo, para evitar uma sobrecarga na CPU durante aplicações em que não há hardware acoplado, recomenda-se a utilização do bloco *Throttle* de modo que a execução do diagrama não trave, ficando limitada ao parâmetro *samp_rate*.

O bloco *top_block* é importante, pois cada vez que um diagrama do GNU Radio é executado, um arquivo python é gerado com o nome contido na *id* do bloco. Deste modo, recomenda-se a escolha de um nome único para o diagrama a fim de evitar confusões.

No topo da GUI, tem-se a barra de tarefas, onde encontram-se comandos como "Novo Arquivo", "Abrir", "Run" e "Stop". Embaixo está o *console*, onde pode-se ver informações sobre a execução do código, como avisos e erros. Ainda, pode-se adicionar um quadro onde é possível a visualização das variáveis presentes no diagrama.

Finalmente, à direita, está a lista dos módulos presentes no GNU Radio. Cada módulo contém uma série de blocos, que são as unidades básicas a serem utilizadas. No módulo *Filters*, por exemplo, existem diversos blocos que implementam variados tipos de filtro, como *Band Pass Filter* e *Root Raised Cosine Filter*, para citar alguns. Um diagrama pode ser salvo como um programa, ou pode ser salvo como se fosse um bloco único, com entradas e saídas, podendo ser usado em outros diagramas sem que a estrutura fique confusa. É possível, ainda, instalar blocos criados por terceiros, que podem desempenhar uma função específica.

3.1.2 Computador

O computador utilizado foi o *notebook* modelo Ideapad320 (80YH0000BR) da marca Lenovo, com as seguintes especificações:

1. Processador: Intel Core i7 - 7500U;
2. Memória RAM: 16GB;
3. GPU: Nvidia GeForce 940MX 4GB;
4. Sistema Operacional: Windows 10 Home Edition.

Para o funcionamento adequado do rádio é necessário haver porta USB de terceira geração (USB 3.0).

3.1.3 Rádio LimeSDR

O Rádio Definido por Software utilizado foi o LimeSDR da Lime Microsystems apresentado na figura 12 capaz de suportar protocolos atuais de comunicação sem-fio, como LTE, GSM, LoRa, Bluetooth, Wi-Fi e outros.

O LimeSDR é composto de um *transceiver* de Rádio Frequência (RF) (LMS7002M FPRF), um FPGA Altera Cyclone IV EP4CE40F23, possui 256MB de memória SDRAM, um controlador USB 3.0 para interface com o computador, um oscilador na frequência de 30,72MHz e 12 conectores de RF (6RX, 4TX e 2 I/O para *clock*) culminando nas seguintes características de operação:

Tabela 1 – Características funcionais do LimeSDR

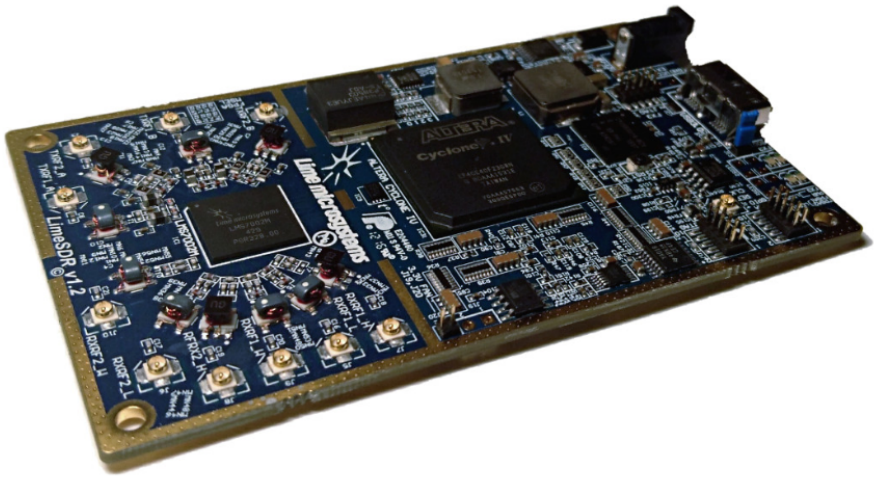
Faixa de Frequência	100KHz - 3,8GHz
Largura de Banda	61,44MHz
Multiplexação	2x2 MIMO
Potência de Saída	até 10dBm

Fonte: retirado de LimeSDR (2018)

A vantagem do LimeSDR é a sua fácil manipulação e versatilidade. A sua característica de hardware e ferramentas de desenvolvimento abertas, combinada com a integração junto ao distribuidor de aplicativos *Snappy* do Ubuntu, permite um grande número de aplicações para a plataforma (LIMEMICROSYSTEMS, 2018).

Os projetos de código aberto são divulgados na comunidade Myriad-RF, indo desde informações sobre plataformas de desenvolvimento até dados de concepção da própria placa.

Figura 12 – LimeSDR Utilizado.



Fonte: retirado de LimeSDR (2018).

3.1.4 Analisador de Espectro

Para visualização dos dados transmitidos, utilizou-se um analisador de espectro. O equipamento é produzido pela Agilent Technologies, modelo FieldFox RF Analyzer N9912A com frequência máxima de 6GHz.

3.2 Métodos

Este projeto apresenta um aspecto eminentemente prático, deste modo, o método utilizado consistiu na exploração prática da ferramenta.

Inicialmente, concebeu-se uma proteção em acrílico para o LimeSDR com um suporte para colocação de quatro antenas para operar na faixa de RF de 2,4GHz. As mesmas foram conectadas nos pinos RX1_H, RX2_H, TX1_1 e TX2_1, que são apropriados para a faixa de frequência maiores que 1,5GHz (LIMESDR. . . , 2018).

O próximo passo consistiu no teste do equipamento. Conforme recomendado, executou-se o *self-test* na plataforma LimeSuite do fabricante. Este software permite visualizar informações de baixo nível do *transceiver* LMS7002M, como o estado dos seus registradores. No entanto, como não era este o enfoque deste trabalho, ela foi utilizada apenas para este teste inicial. O procedimento consistiu no envio e recebimento em *loopback* de um sinal WCDMA (modulação por código presente na tecnologia 3G) e sua visualização (LIMESDR-USB. . . , 2018).

Uma vez atestado o correto funcionamento da placa, a metodologia seguida foi a implementação de estudos de caso utilizando a plataforma GNU Radio, primeiro em modo

simulado, depois em modo real e finalmente comparando os resultados medidos com o analisador de espectro com os mostrados pelo GNU Radio.

Foram implementados dois estudos de caso: Modulação GFSK e Modulação QPSK, este último tendo sido o enfoque principal deste trabalho. Os estudos de caso serão detalhados a seguir.

O objetivo destes estudos de caso foi verificar o comportamento da plataforma LimeSDR com o GNURadio, levantando pontos importantes observados ao longo do desenvolvimento, principalmente as dificuldades encontradas para que este trabalho possa servir como ponto de partida para os futuros alunos que desejarem trabalhar com as duas plataformas.

4 Estudos de Caso

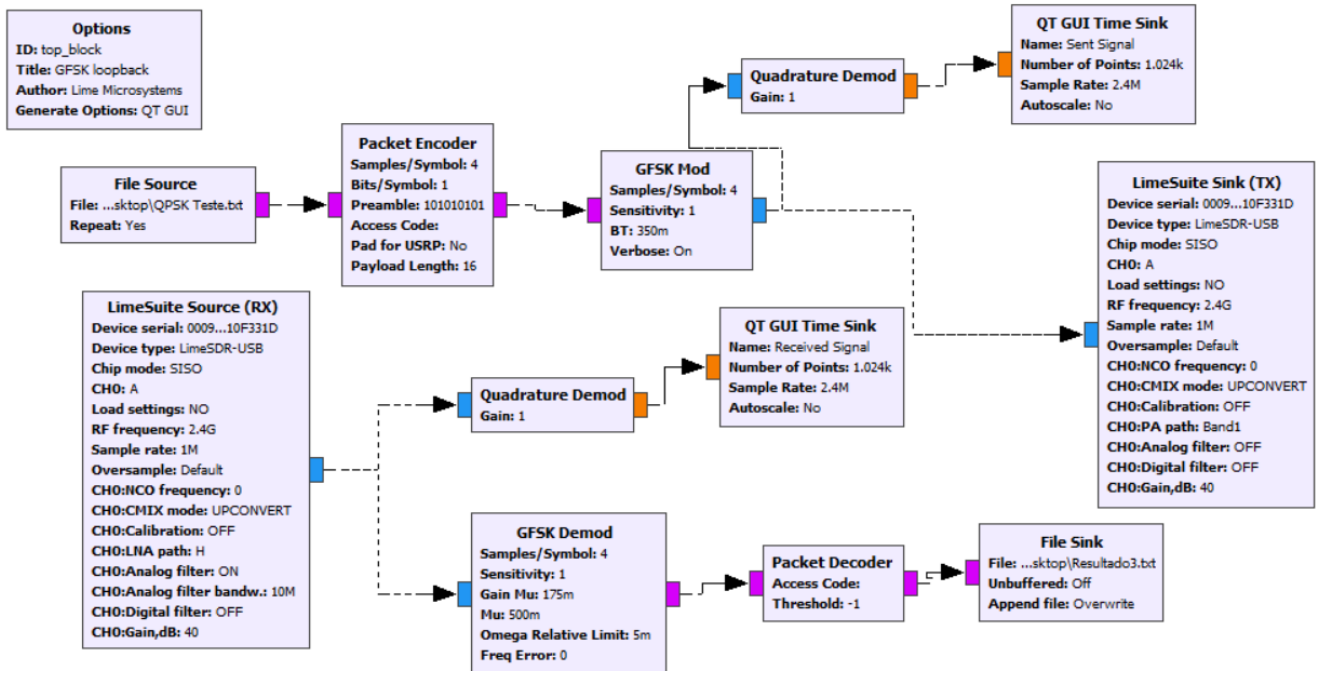
A seguir são apresentados os dois estudos de caso para verificação das capacidades e limitações das plataformas GNURadio e LimeSDR.

4.1 Estudo de Caso 1: Modulação GFSK

Este foi o primeiro estudo de caso realizado, em simulações e experimentos em laboratório. O principal intuito deste primeiro desenvolvimento era compreender melhor a plataforma GNU Radio e sua integração com o LimeSDR. Como o mesmo utiliza um diagrama de blocos relativamente simples, ele foi essencial para dar os primeiros passos na parametrização dos blocos e configuração do SDR. Este estudo de caso foi sugerido pela comunidade Myriad-RF para ser usado como exemplo de uso da plataforma (GFSK. . . , 2018).

O seu diagrama de blocos é detalhado na figura 13.

Figura 13 – Diagrama de blocos para GFSK.



Fonte: O Autor.

No canto superior esquerdo, pode-se ver o bloco *Options* com as informações de identificação do diagrama, bem como o tipo de interface gráfica que será utilizada (QT GUI).

O diagrama começa com o bloco *File Source*. Este bloco é o responsável pela fonte de informação, que no caso é um arquivo de texto. Com a opção *Repeat* ativada, ele é enviado continuamente.

Em seguida passa-se ao bloco *Packet Encoder*, o codificador. Neste bloco, os *bits* do arquivo fonte são codificados em pacotes com um certo número de *bytes* significativos (parâmetro *Payload*), um *Header* (*Access Code*) e um preâmbulo sincronizador (*Preamble*). Além disto, deve-se definir um valor de número de amostras por símbolo (*Samples per Symbol*), que deve ser o menor possível ou adequado para conseguir uma taxa de envio de *bits* desejada ao utilizar-se um determinado hardware (GNURADIO, 2018a). Neste caso, foi escolhido o valor de quatro amostras por símbolo. O parâmetro *Bits per Symbol* é o que define como será feita a montagem do pacote. Para uma modulação GFSK, temos que um *bit* em "0" corresponde a uma certa frequência e um *bit* em "1" corresponde a outra, causando uma variação de frequência na portadora. Assim, podemos representar estes dois estados com apenas um *bit*.

Em seguida, o pacote é passado pelo bloco *GFSK Mod*, que é o responsável pela modulação GFSK em banda base. Neste bloco, deve-se colocar o mesmo valor de *Samples per Symbol* utilizado previamente para que não haja problemas de amostragem. O parâmetro *BT* corresponde à largura de banda excessiva do filtro gaussiano para a representação dos *bits*, conhecido como fator de decaimento (ou *Roll-off*). Utilizou-se o valor de 0,35, garantindo um compromisso de duração no tempo do pulso e utilização de espectro.

Ao sair deste bloco, o sinal é demodulado para visualização do mesmo no tempo e também é injetado no bloco *LimeSuite Sink (TX)*. Este bloco foi criado pela LimeMicrosystems para ser usado para configuração do LimeSDR. A sua parametrização é descrita a seguir:

Tabela 2 – Parametrização do bloco *LimeSuite Sink (TX)*

Device Serial	Selecionar qual dispositivo utilizar, caso haja mais de um.
Device Type	Permite escolher um dispositivo de uma lista pré-definida e carregar configurações apropriadas para cada dispositivo. Os dispositivos suportados são: LimeSDR-USB, LimeSDR-Mini.
Chip Mode	Permite escolher entre os modos SISO (Single Input and Single Output) e MIMO (Multiple-Input and Multiple-Output).
CH0	Quando em modo SISO, selecionar o canal ativo como canal zero.
Load Settings	Permite o carregamento de configurações de um arquivo.
File	Gerar arquivo .ini com o LimeSuite.
Center Frequency	Selecionar a frequência central de RF para TX (ambos os canais).
Sample Rate	Selecionar a frequência de amostragem para TX.
Oversample	Permite selecionar sobreamostragem para TX. Valor *Default* é zero.
NCO Frequency	Ajuste do oscilador controlado numericamente para cada canal. 0 significa que o NCO está desligado.
CMIX Mode	Controla a direção do NCO para cada canal.
Calibration	Ativa nível CC e liga/desliga calibração desbalanceada de IQ para cada canal.
Calibration Bandw.	Quando a calibração estiver ligada, este parâmetro é utilizado para configurar a calibração para cada canal. Este valor deve ser igual à largura de banda do sinal.
PA Path	Seleciona o caminho do amplificador de potência para cada canal.
Analog Filter	Liga/desliga o filtro passa baixa para cada canal.
Analog Filter Bandw.	Insira a largura de banda para cada canal.
Digital Filter	Liga/desliga filtro digital (GFIR) para cada canal.
Digital Filter Bandw.	Insira a largura de banda para cada canal. A largura de banda não deve ser maior que a frequência de amostragem.
Gain	Controla ganho na TX. Ganho deve estar entre 0 e 60 dB.

Fonte: retirado de (MICROSYSTEMS, 2018).

Foram escolhidos os valores de 2,4 GHz para frequência central, taxa de amostragem de 1MSps e ganho de 40dB. Os demais parâmetros foram mantidos nos seus valores padrões para uma transmissão SISO com o LimeSDR-USB.

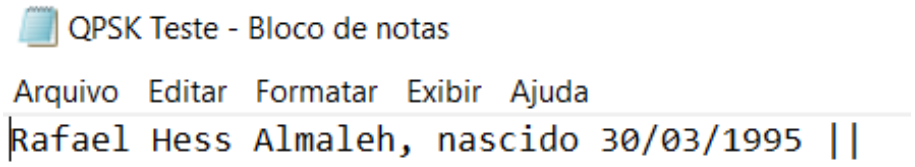
Analogamente, temos um bloco para recepção do sinal transmitido, o bloco *LimeSuite Source (RX)*. A sua parametrização difere do bloco TX pela presença do *LNA Path* ao invés do *PA Path*. Este parâmetro serve para a escolha do caminho do amplificador de baixo ruído. Como o sistema opera em 2,4GHz, o valor deve ser colocado em H (de *high*, alto, em inglês). O sinal passa então por um demodulador GFSK.

Neste bloco encontram-se alguns parâmetros novos: *Gain Mu*, *Mu*, *Omega Relative Limit* e *Freq. Error*. O primeiro parâmetro (*Gain Mu*) é o fator de correção baseado na diferença de tempo entre os símbolos, o valor de 0,175 é o recomendado para o demodulador GFSK. O segundo parâmetro (*Mu*) está relacionado ao ajuste da amostragem devido ao

sinal de erro e seu valor recomendado é de 0,5. Finalmente, o último parâmetro a ser configurado (*Omega Relative Limit*) diz respeito aos valores máximo e mínimo em relação a ômega (que é o *samples per symbols*), seu valor recomendado é de 0,005. O erro em frequência foi deixado em zero. Estes parâmetros foram extraídos de Spench (2019) para a configuração do bloco *M&M Clock Recovery*, que implementa o algoritmo de sincronização Mueller & Müller. Este é o método utilizado internamente pelo bloco demodulador para conseguir demodulação síncrona. Em seguida, tem-se um decodificador do pacote e o bloco onde os dados são salvos em outro arquivo de texto.

O arquivo enviado para teste foi um documento de texto com uma frase, conforme figura 14 apresenta.

Figura 14 – Arquivo de texto enviado para teste.



Fonte: O Autor.

4.2 Estudo de Caso 2: Modulação QPSK

Após a implementação do primeiro estudo de caso sugerido para fins de teste, passou-se à implementação de uma estrutura mais complexa. As configurações para o modo simulado e para o modo real tiveram as suas diferenças, sendo explanadas à seguir.

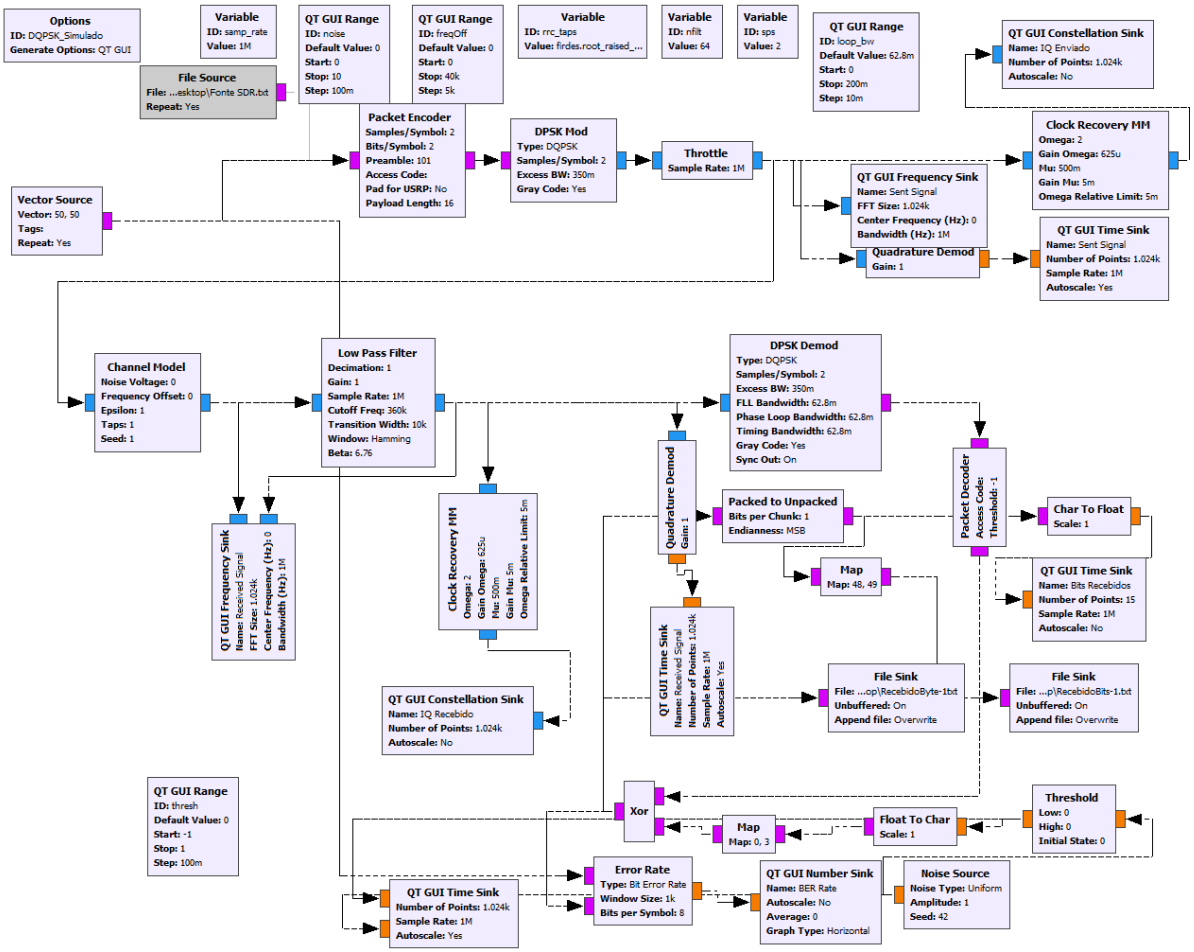
4.2.1 Modo Simulado

Similarmente ao estudo de caso anterior, a ideia central é ter uma informação a ser enviada, que será codificada em um pacote, sendo este então modulado com um esquema desejado, neste caso o QPSK. A informação modulada é então enviada e recuperada no receptor seguindo o caminho oposto: demodulação e decodificação do pacote. O diagrama de blocos é apresentado na figura 15.

Inicialmente, o arquivo a ser enviado era o mesmo da figura 14. Contudo, para se ter maior controle sobre os *bits* recebidos com erro, optou-se por usar também o bloco *Vector Source* como fonte de informação. O bloco foi configurado para enviar continuamente

o valor 50, que corresponde ao caractere "2" na tabela ASCII. O bloco seguinte é o já detalhado *Packet Encoder*, desta vez parametrizado com *Samples per Symbol* igual a dois, podendo ser um valor baixo já que é um ambiente simulado. O sinal passa, então, pelo bloco *Throttle* para limitar a carga na CPU, já que não há um hardware acoplado. Visualiza-se, então, o diagrama de constelação do sinal enviado bem como a sua representação no domínio tempo e em frequência. A correta sincronização da constelação com os símbolos enviados é garantida pelo bloco *M&M Clock Recovery*, com *Ômega* recebendo o valor de *Samples per Symbols*; *Gain Omega* corresponde à expressão $0,25 * GainMu^2$, conforme Spench (2019), com *Gain Mu* tendo o valor recomendado de 0,05. Para *Mu* foi escolhido o valor de 0,5, que é, também, recomendado por Spench (2019) e encontra-se na metade da faixa dos valores recomendados pela documentação oficial do GNU Radio. Para *Omega Relative Limit* seguiu-se o mesmo raciocínio do estudo de caso anterior.

Figura 15 – Diagrama de blocos para implementação simulada de envio QPSK.



Fonte: O Autor.

O sinal modulado em quadratura passa por um bloco cuja finalidade é simular um modelo de canal, o *Channel Model*. Os parâmetros deste bloco são os seguintes: *Noise Voltage*, que representa o nível médio de um ruído branco aditivo gaussiano (AWGN, da sigla em inglês) ; *Frequency Offset*, correspondendo ao desvio em frequência normalizado, onde zero significa nenhum desvio e 0,25 significa um quarto da taxa de envio de símbolos; *Epsilon*, que é o desvio da taxa de amostragem, servindo para emulação das diferentes taxas entre as amostras de *clock* do transmissor e do receptor; *Taps*, correspondendo ao número de *taps* (comprimento) de um filtro FIR para emular um perfil de distorção por multicaminhos; e *Seed*, correspondendo à semente para a geração dos ruídos. Estes parâmetros foram associados à variáveis a fim de serem controlados. Na saída do canal, passa-se o sinal por um filtro passa-baixa a fim de remover as componentes de frequência indesejadas. Neste filtro, devemos configurar os parâmetros *Gain* e *Decimation* para indicar se o filtro insere algum ganho ou se diminui o número de amostras, neste caso, deixa-se o valor em "1". Devemos indicar a taxa de amostragem e também a frequência de corte do mesmo, configurada em 360kHz, que é um pouco maior que a mais elevada componente do sinal enviado. Foi escolhido uma transição de 10kHz de largura e uma janela do tipo *Hamming*. O parâmetro *Beta* foi deixado no seu valor padrão, pois ele é utilizado apenas para uma janela do tipo *Kaiser*. Após a passagem pelo filtro, visualiza-se a constelação recebida, sempre com o auxílio do bloco *M&M Clock Recovery*, o sinal no tempo e seu espectro em frequência.

O bloco seguinte é o responsável pela demodulação do sinal, o *DPSK Demod*. Para conseguir demodular corretamente, este bloco já tem embutido um laço de travamento em fase (PLL) e um laço de travamento em frequência (FLL) para sincronismo. Os parâmetros a serem configurados são o tipo de modulação, no caso a DQPSK; O parâmetro *Samples per Symbols*, já detalhado; O *Excess BW*, que corresponde ao fator de decaimento do filtro de cosseno elevado, explicado anteriormente; Em seguida tem-se três parâmetros que correspondem à largura de banda dos filtros nos laços de travamento de fase, frequência e de sincronização. Foi deixado o valor padrão de 62,8m ($2 * pi/100$) radianos por amostra, que é o recomendado pela documentação oficial do GNU Radio para laços de sincronização. Estes parâmetros estão relacionados com a velocidade de travamento dos laços, quanto maior a largura de banda dos filtros, mais rápida é a sincronização, porém torna-se mais difícil de se controlar a estrutura. É selecionada codificação do tipo Gray e a função de *Sync Out* não foi determinada, sendo deixada no seu valor padrão. O sinal é então decodificado com o *Packet Decoder*.

Com a finalidade de simular uma recepção errônea de *bits*, podendo-se variar a probabilidade de erro (BER, do inglês *Bit Error Rate*). Para conseguir esta simulação, colocou-se uma fonte de ruído uniforme com amplitude unitária. A saída desta fonte de ruído é então passada por um bloco *Threshold*, que estabelece um limiar variável para comparação. Caso o valor do ruído seja maior que esse limiar, o bloco retorna "1", caso

contrário, “0”. Esses dois valores do comparador são então mapeados como “0” e “3”, pois a ideia é que, em um número de 8 *bits*, tenha-se o valor 0000 0000 e 0000 0011. Esses valores binários e o sinal recebido são operados com um a operação lógica XOR, ou seja: se a saída do comparador for “0”, nada ocorre, se for “1”, os dois *bits* menos significativos são trocados. O sinal retornado pelo “XOR” é armazenado em dois arquivos texto, um com os valores correspondentes na tabela ASCII, e outro com a sua representação binária. Foi incluso, também, um visualizador do sinal binário no domínio tempo para melhor compreensão. Ademais, a saída do “XOR” é também comparada com o sinal de entrada por meio do bloco *Error Rate*, que mede a taxa de erro de *bits* em relação ao sinal original. Ao variar-se o limiar da fonte de ruído durante o envio contínuo do número “2” (0011 0010, na notação binária para a tabela ASCII), os *bits* acabam por serem trocado mais frequentemente. Pode-se escolher quantos e quais *bits* serão trocados mudando-se o mapeamento. Se trocar-se o “3” por um “2” no mapeamento, o bit a ser mudado seria o segundo bit menos significativo (uma vez que “2” corresponde a 0000 0010). Seguindo o mesmo raciocínio, se a escolha fosse por um “255” (1111 1111, em binário) no lugar do “3”, todos os *bits* recebidos seriam trocados. Por meio do parâmetro *Bits per Symbol*, podemos medir a taxa de erro de *bytes* indiretamente, ao escolhermos o valor de “2” ao invés de “8”. A transmissão com ruído pode também ser simulada variando-se o nível de ruído branco do bloco *Channel Model*.

Analogamente ao estudo de caso GFSK, poderíamos ter utilizado o arquivo de teste mostrado na figura 14 ou qualquer outro de nosso interesse. Contudo, prezou-se pelo envio de apenas um caractere para explorar a simulação de erro na recepção dos *bits*. Ao adotar o envio de apenas um caractere continuamente, fica mais fácil a identificação e quantificação de recepções erradas ao comparar o conteúdo recebido com o conteúdo enviado.

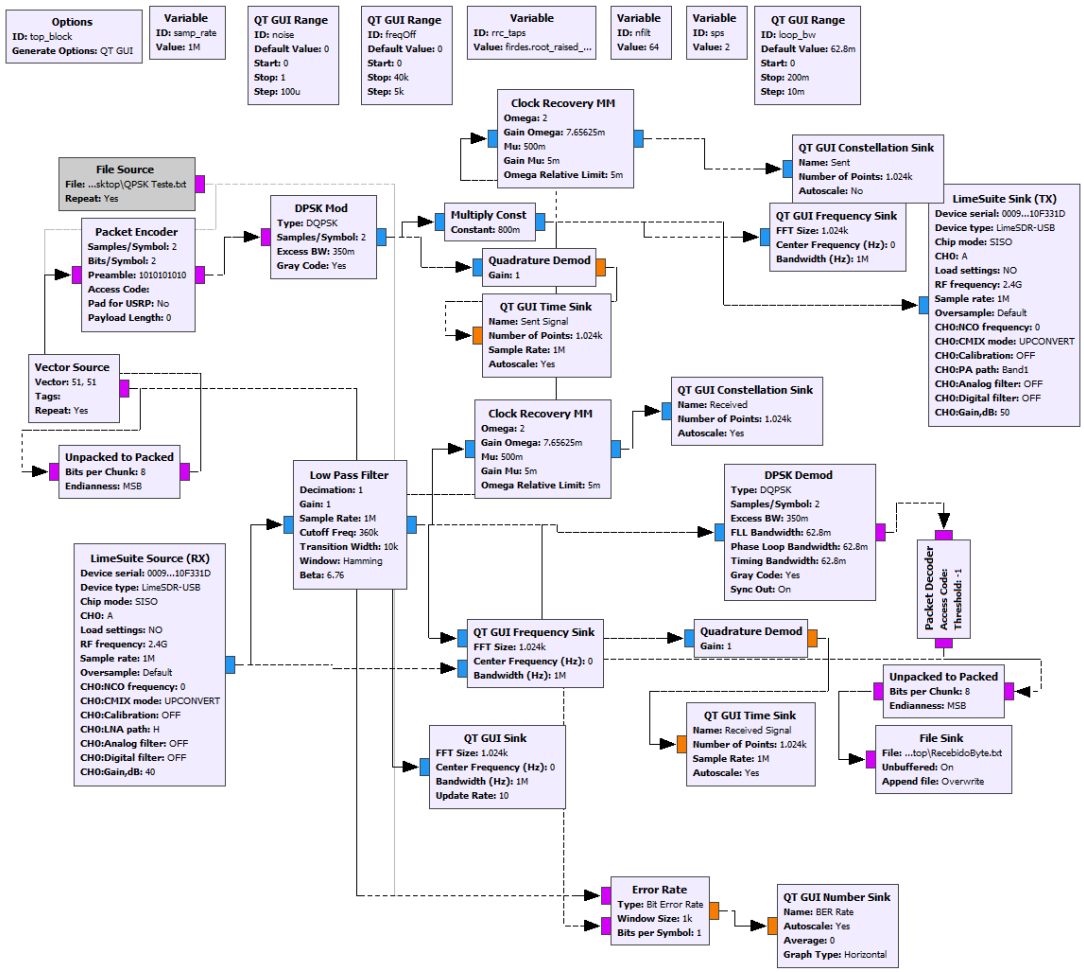
As medições a serem realizadas no caso simulado são a constelação do sinal (enviado e recebido),resposta em frequência (do sinal enviado e recebido), visualização no domínio tempo (do sinal enviado e recebido), visualização dos *bits* recebidos, visualização da quantidade de *bits* invertidos e taxa de erro de *bits*.

4.2.2 Implementação Física

Para a operação com o LimeSDR alguns blocos foram retirados e outros adicionados, a estrutura, no entanto, continuou sendo a mesma, conforme pode-se ver na figura 16. Os blocos adicionados foram os relacionados à configuração do rádio, tanto para recepção quanto para transmissão. Necessitou-se, também, adicionar um bloco *Multiply Const* para uma atenuação do sinal antes da saída. Foi identificado experimentalmente que com um valor menor que 0,8 o rádio não conseguia transmitir, retornando um erro. Como esta limitação não era desejável, escolheu-se o maior valor possível para este parâmetro.

Dentre os blocos removidos, tem-se o bloco modelador de canal (*Channel Model*), que se tornou desnecessário uma vez que agora a comunicação ocorre fisicamente pelo ar. Outros blocos removidos foram os relacionados a parte da simulação de recepção errada de *bits*, que se torna desnecessária agora que os *bits* podem ser realmente recebidos de maneira correta ou errada baseado em interferências reais.

Figura 16 – Diagrama de blocos para implementação real de envio QPSK.



Fonte: O Autor.

O estudo de caso no modo real consistiu no envio constante do caractere “3” da tabela ASCII, a verificação se o sinal foi recebido corretamente, e a comparação das medições feitas pelo GNU Radio com as realizadas pelo analisador de espectro.

As medições a serem realizadas pelo GNU Radio são a constelação do sinal (enviado e recebido), resposta em frequência (do sinal enviado e recebido), visualização no domínio tempo (do sinal enviado e recebido) e taxa de erro de *bits*. Com o analisador de espectro, pode-se medir a resposta em frequência do sinal e sua intensidade.

5 Resultados e Discussão

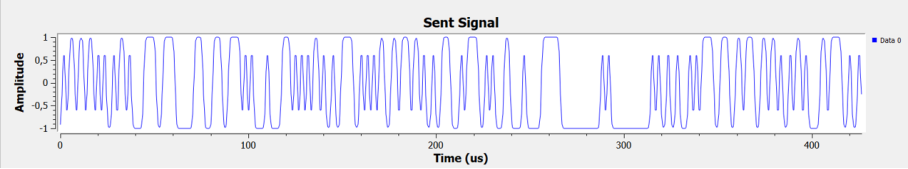
A seguir serão expostos e analisados os resultados dos experimentos realizados.

5.1 GFSK

A modulação GFSK foi a primeira a ser testada, por ser de fácil implementação e por estar disponível pela LimeMicrosystems, fabricante do SDR.

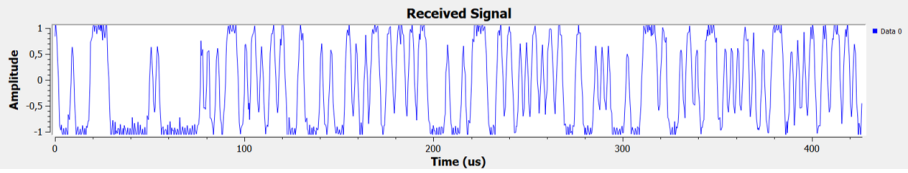
Nas figuras 17 e 18 pode-se visualizar os sinais GFSK no domínio tempo. Percebe-se a semelhança entre os sinais, com o recebido apresentando leves deformações. Apesar de não ser o domínio ideal para esta análise, consegue-se visualizar diferentes frequências nos sinais, indicando que há uma variação em frequência, característica da modulação GFSK.

Figura 17 – Sinal enviado.



Fonte: O Autor.

Figura 18 – Sinal recebido.



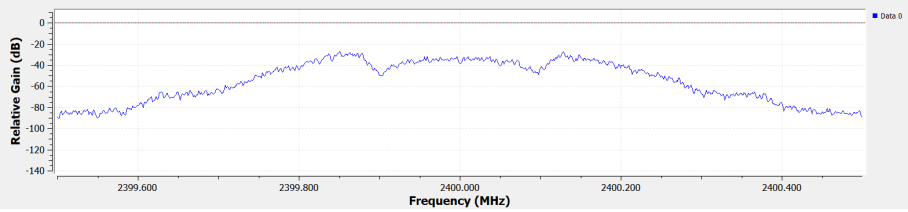
Fonte: O Autor.

Sabe-se que este tipo de modulação ocorre variando-se a frequência em dois níveis, assim, espera-se que ocorram dois picos no domínio frequência. Como este chaveamento ocorre de maneira mais suave, estes picos serão mais largos, ocupando uma largura de banda maior.

Modificou-se, então, o diagrama original de modo a adicionar um visualizador no domínio frequência e também um visualizador do tipo *Waterfall* (espectro em frequência

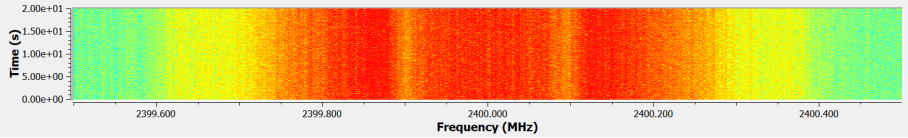
em função do tempo). O resultado encontra-se nas figuras 19 e 20. Nestas duas imagens consegue-se verificar que há um pulso em $f_c + 130kHz$, e em $f_c - 130kHz$. Estes dois pulsos são os representantes dos *bits* um e zero.

Figura 19 – Sinal enviado no domínio frequência.



Fonte: O Autor.

Figura 20 – Sinal enviado no domínio frequência com visualização *Waterfall*.



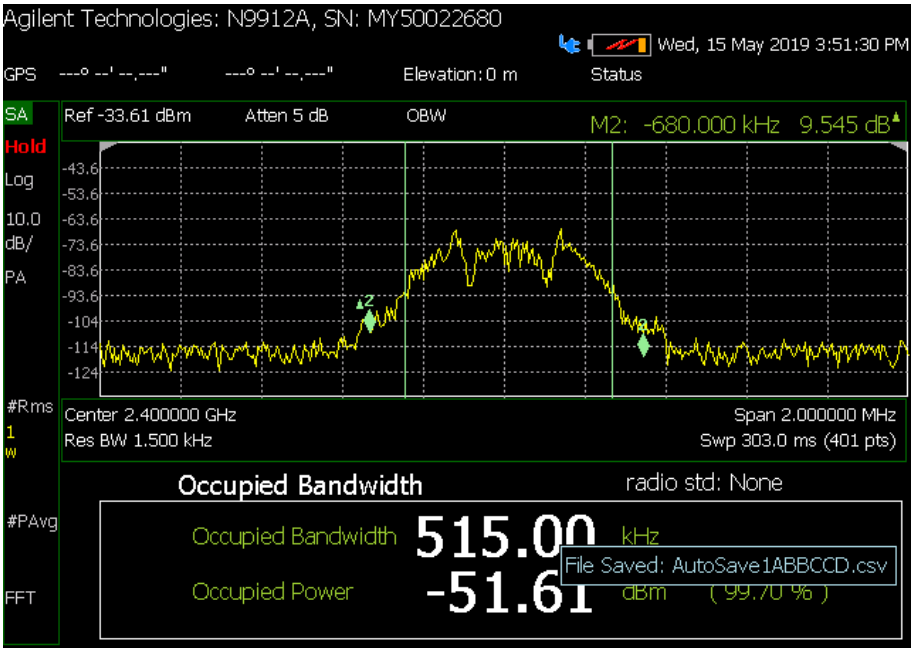
Fonte: O Autor.

Com o analisador de espectro, observa-se uma representação em frequência fiel a mostrada pelo GNU Radio. Enquanto que no software visualiza-se uma largura de banda de aproximadamente 770kHz, no analisador de espectro encontra-se um valor próximo a 756kHz (conforme figura 21). Ressalta-se, aqui, que a medição automática de largura de banda com o analisador de espectro não é muito precisa, tendo-se que aferir manualmente com o uso de marcadores ou olhando a escala dos gráficos.

Teoricamente, a largura de banda de um sinal GFSK pode ser estimada pela expressão $BW = taxadedados + 2 * (desvioentresímbolos)$. Os dados estão sendo transmitidos a uma taxa de $1\frac{Mamostras}{segundo}$, têm-se, também, quatro amostras por símbolo e um *bit* por símbolo, resultando em uma taxa de dados de 250 kbps. A diferença entre os símbolos é da ordem de $130kHz - (-130kHz) = 260kHz$. A combinação desses valores resulta em uma largura de banda da ordem de 770kHz, que é coerente com os resultados obtidos.

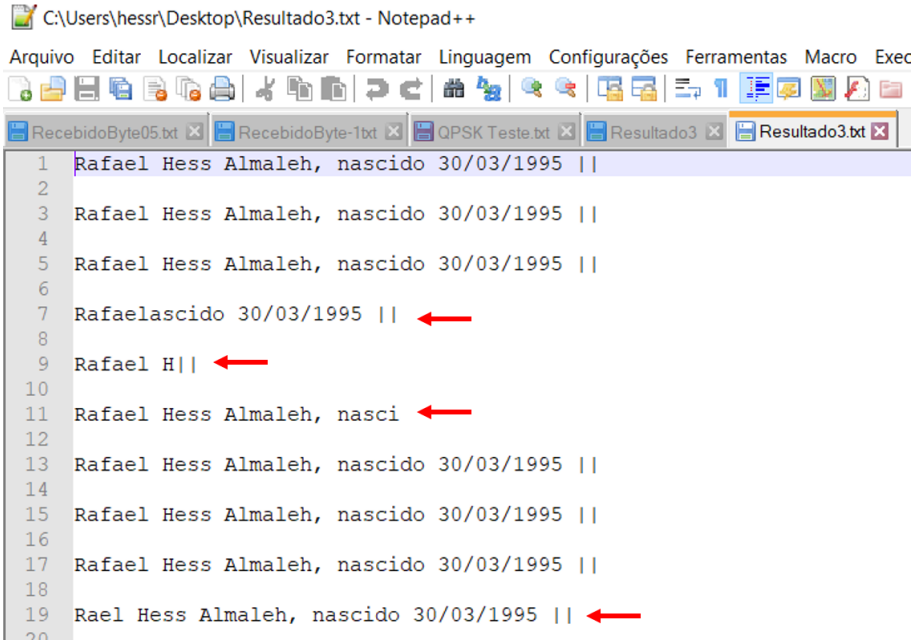
No que diz respeito ao arquivo recebido, pode-se verificar que certos erros de recepção aparecem, com informações que são perdidas e não captadas na parte receptora. Na figura 22, pode-se ver que em quatro linhas há erros de recepção da informação.

Figura 21 – Visualização espectral do sinal GFSK medida no analisador de espectro.



Fonte: O Autor.

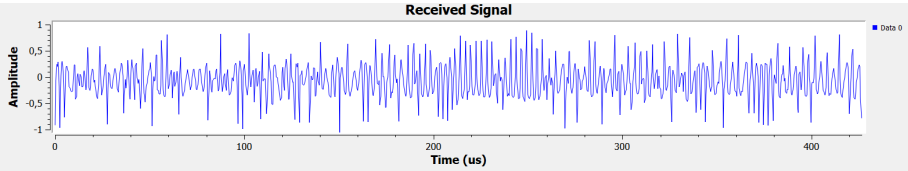
Figura 22 – Arquivo de texto recebido.



Fonte: O Autor.

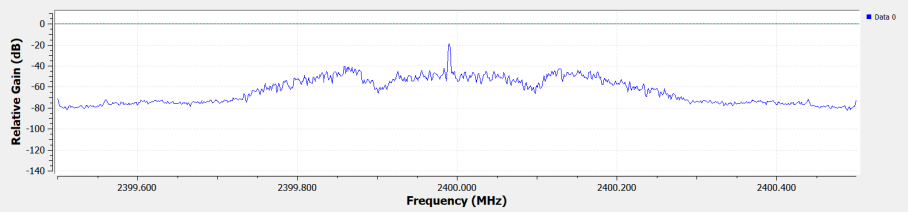
Procurou-se medir a taxa de erro de *bits* com a geração de ruído de onda contínua por uma das saídas do analisador espectral e envio constante do caractere “2”. Não foi utilizada uma metodologia específica para um certo tipo de protocolo de comunicação, pois nenhum foi implementado em sua totalidade. Começou-se o procedimento com atenuação máxima na saída do ruído e foi-se aumentando o valor até perceber-se alguma mudança nos visualizadores do GNU Radio. Com uma atenuação de 24dB, percebe-se um pico perto dos 2,4GHz. Com atenuação de 16dB este pico fica mais pronunciado em relação ao sinal, conforme pode-se ver na figura 24. A perturbação fica evidente, também, no domínio tempo. Na figura 23, vê-se que o sinal está irreconhecível em relação ao sinal “limpo” da figura 18.

Figura 23 – Sinal GFSK recebido com ruído.



Fonte: O Autor.

Figura 24 – Espectro do sinal GFSK recebido com ruído.



Fonte: O Autor.

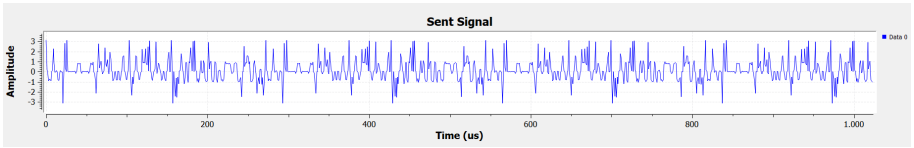
Apesar disto, não ocorreram erros na recepção do arquivo de texto. A explicação para o ocorrido pode ser o fato de que o ruído, por ser muito estreito em frequência, não interfere nos símbolos recebidos.

5.2 QPSK - Simulado

Os resultados encontrados para o envio simulado do caractere “2” com modulação QPSK com os parâmetros descritos anteriormente e com um *Threshold* máximo para inversão de bits são exposto a seguir.

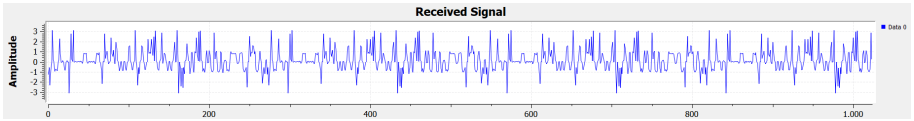
(i) Visualização no domínio tempo do sinal enviado e recebido (figuras 25 e 26):

Figura 25 – Sinal enviado.



Fonte: O Autor.

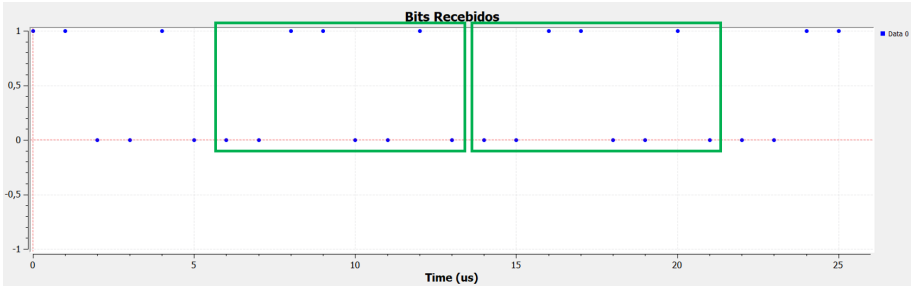
Figura 26 – Sinal recebido.



Fonte: O Autor.

(ii) Bits Recebidos (figura 27):

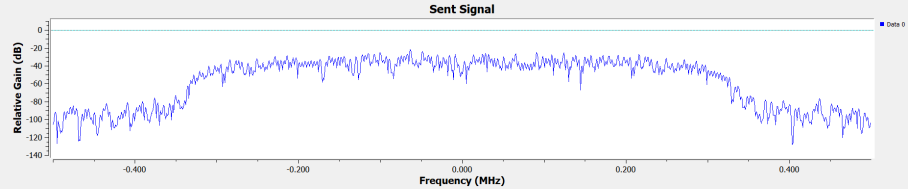
Figura 27 – Bits recebidos.



Fonte: O Autor.

(iii) Representação em frequência do sinal enviado (figura 28):

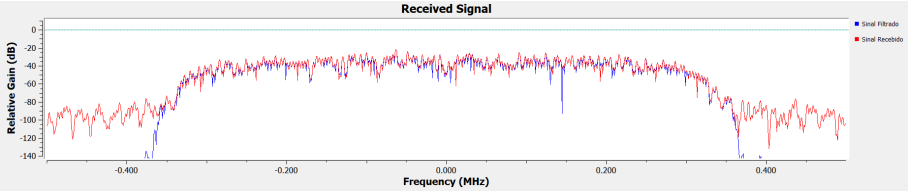
Figura 28 – Espectro do sinal enviado.



Fonte: O Autor.

(iv) Representação em frequência do sinal recebido (figura 29):

Figura 29 – Espectro do sinal recebido antes e após filtro passa-baixas.



Fonte: O Autor.

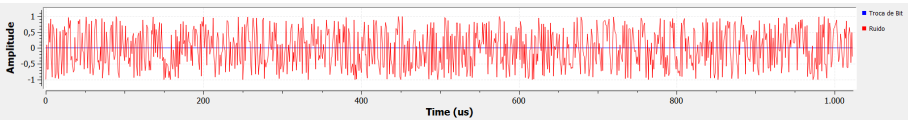
(v) Medidor da taxa de erro de *bits* (BER) na recepção (figura 30):

Figura 30 – Taxa de erro de *bits*.

Fonte: O Autor.

(vi) Ruído e sinalização dos pontos de troca de *bits* (figura 31):

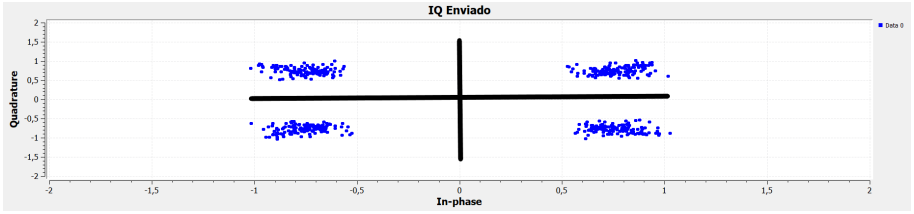
Figura 31 – Pontos onde há recepção errada de *bits*.



Fonte: O Autor.

(vii) Diagrama IQ enviado (figura 32):

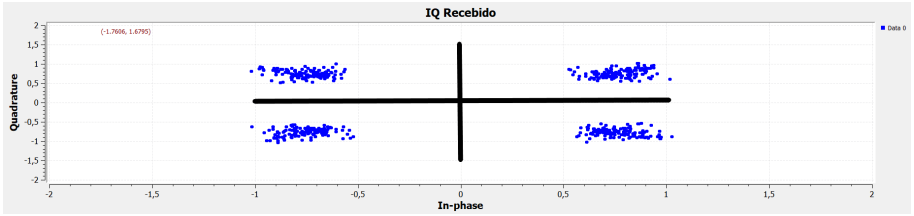
Figura 32 – Diagrama IQ enviado.



Fonte: O Autor.

(viii) Diagrama IQ recebido (figura 33):

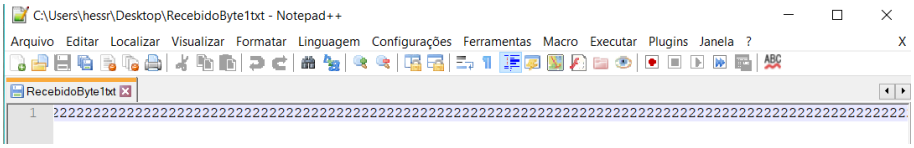
Figura 33 – Diagrama IQ recebido.



Fonte: O Autor.

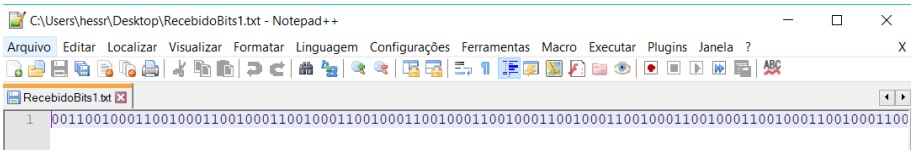
(ix) Arquivos com os resultados recebidos (figuras 34 e 35):

Figura 34 – *Bytes* recebidos.



Fonte: O Autor.

Figura 35 – *Bits* recebidos.



Fonte: O Autor.

Começamos a análise pelo sinal visualizado no domínio frequência (representado nas figuras 28 e 29). Considerando que o parâmetro *samp_rate* foi configurado para o valor de 1MHz e o bloco *Throttle* nos garante que a frequência de envio do sinal ficará limitada a este valor, deveríamos ter a seguinte largura de banda: $\frac{2\frac{amostras}{símbolo}}{2\frac{bits}{símbolo}} = 1\frac{amostra}{bit}$. Considerando a taxa de amostragem de $1\frac{Mamostras}{segundo}$, temos a seguinte taxa de envio de dados: $1\frac{amostra}{bit} * \frac{1}{1\frac{Mamostras}{segundo}} = 1Mbps$.

Para sabermos a largura de banda em hertz corresponde a esta taxa, devemos considerar o fator de decaimento (*Roll-off*) para a representação dos bits. A equação correspondente para relacionar largura de banda e taxa de envio de bits é a seguinte:

$$BW = \frac{(1+\alpha)}{2} * f_s. \tag{5.1}$$

onde α é o fator de decaimento e f_s é a taxa de envio de dados em Mbps, resultando em uma largura de banda (BW) em MHz.

De acordo com a equação 5.1, para um $\alpha = 0,35$ e $f_s = 1Mbps$ devemos ter uma largura de banda mínima de 675kHz. Ao medir-se a largura de banda do sinal enviado, vê-se que a mesma varia aproximadamente entre -350kHz e 350kHz, resultando em uma largura de banda ocupada de aproximadamente 700kHz, que corrobora o cálculo teórico anterior. A diferença de valor é resultado de dados enviados que não foram considerados no cálculo, como os *bits* com a informação dos pacotes, assim como o *Header* e o preâmbulo sincronizador. Pode-se citar, também, os dados para a criação dos arquivos de texto. Por meio deste valor que se dimensionou o filtro passa-baixas na recepção.

No visualizador de *bits* recebidos (figura 27), consegue-se identificar a sequência enviada 00110010. Ademais, verifica-se que o tempo entre envio de *bits* é de 1μ s, conforme a nossa taxa de amostragem. No visualizador do ruído e troca de *bits* (figura 31), vemos o ruído em vermelho e o indicador de troca de bits em azul sempre em zero, pois todos os bits estão sendo recebidos corretamente. Este fato é refletido no indicador de BER na figura 30 com o valor nulo e também no visualizador da figura 27. Os arquivos recebidos, nas figuras 34 e 35, não possuem nenhum valor trocado ou sequência errada. As figuras 32 e 33 mostram os diagramas de constelação com as quatro regiões bem definidas, representando os símbolos 00, 01, 11 e 10.

A simulação torna-se mais importante ao selecionar um nível de *threshold* menor que 1,0. O ruído uniforme apresenta uma densidade de potência distribuída uniformemente pelo espectro do sistema. A amplitude do ruído utilizado é unitária, deste modo, ao selecionar-se um *Threshold* com um valor menor que a unidade, *bits* serão trocados.

Ao variar-se o parâmetro *Threshold* na simulação de erro de bits, obtêm-se os valores mostrados no quadro 3.

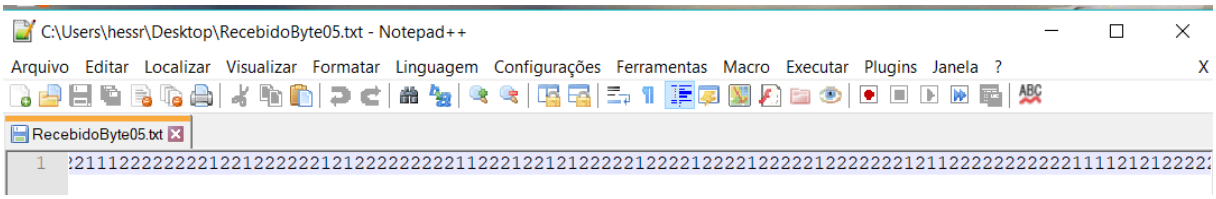
Tabela 3 – *Threshold* x Taxa de Erro

Threshold	*Bit Error Rate*	*Byte Error Rate*
1,0	0,000%	0,000%
0,9	1,244%	4,974%
0,8	2,492%	9,969%
0,7	3,761%	15,044%
0,6	5,008%	20,033%
0,5	6,253%	25,011%
0,4	7,515%	30,060%
0,3	8,765%	35,062%
0,2	10,022%	40,089%
0,1	11,271%	45,084%
0,0	12,514%	50,056%
-1.0	25,000%	100,000%

Fonte: o Autor.

Os valores deste quadro foram preenchidos analisando-se os arquivos de texto recebidos.

Figura 36 – *Bytes* recebidos para *Threshold* de 0,5.



Fonte: O Autor.

Ao analisar-se o quadro 3 observa-se e conclui-se: Para um *Threshold* de -1,0, têm-se que o ruído sempre será maior que o limiar, ou seja, os *bits* serão sempre trocados. De acordo com o diagrama, os dois *bits* menos significativos são os modificados (transformando 50 em 49), ou seja, com os *bits* sendo trocados 100% do tempo, teremos 6 *bits* corretos e 2 errados, resultando em uma taxa de erro de bits de 25%. Para o valor de *Threshold* nulo, teremos que na metade do tempo o ruído estará acima do limiar, ou seja, metade dos valores "50" serão mudados para "49", resultando em uma taxa de erro de *bytes* de 50% e, como a mudança de um *byte* está atrelada, neste caso, a uma mudança de dois dos oito *bits*, a taxa de erro de *bits* será um quarto da taxa de erro de *bytes*. Estendendo esta ideia para os outros valores, vemos que esta hipótese se concretiza.

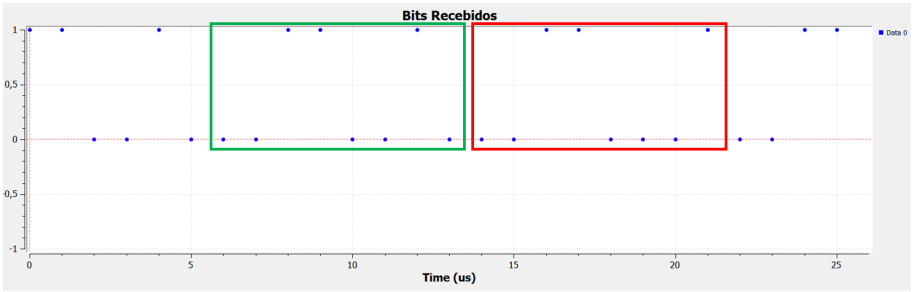
Comparando as medições realizadas por meio dos arquivos com as do GNU Radio, vê-se que há coerência, conforme figura 37 e o respectivo valor no quadro 3. Na figura 36, pode-se ver alguns *bytes* trocados para o caractere "1" (valor 49 na tabela ASCII).

Figura 37 – Taxa de erro de *bits* para *Threshold* de 0,5.

Fonte: O Autor.

No visualizador de *bits*, pode-se também visualizar essas trocas. Na figura 38 vê-se a sequência correta (00110010) na área verde e a sequência errada na área vermelha (00110001).

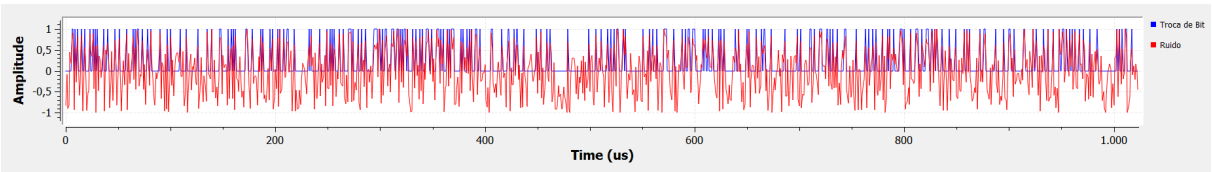
Figura 38 – *Bits* recebidos para *Threshold* de 0,5.



Fonte: O Autor.

No visualizador de troca de bits da figura 39, pode-se verificar os momentos em que as trocas ocorrem, sinalizado por pulsos em azul.

Figura 39 – Sinalização de troca de *bits* para *Threshold* de 0,5.



Fonte: O Autor.

A utilização do bloco *Channel Model* não foi eficaz pois o mesmo acabava travando o diagrama ao variar-se o nível de ruído, com o valor máximo chegando a 0,3V. Mesmo inicializando-se o bloco com um valor fixo, se o mesmo fosse maior que 0,3V, o diagrama não conseguia ser executado. Utilizando-se o valor de ruído entre 0V e 0,3V não foi verificada nenhuma interferência na recepção dos dados. Deste modo, a simulação de ruído foi executada com a troca de *bits* manual, para fins de demonstração de funcionamento do diagrama.

5.3 QPSK - Implementação Física

Após a implementação simulada do QPSK, passou-se à fase do uso do LimeSDR para envio e recebimento dos dados, conforme descrito no capítulo anterior. As medições foram realizadas em laboratório para comparação dos dados recebidos pelo analisador de espectro com os dados medidos pelos blocos de visualização do GNU Radio.

Uma fotografia da configuração dos experimentos realizados em laboratório pode ser vista na figura 40. Inicialmente, conectou-se o analisador de espectro diretamente ao conector do SDR. Posteriormente, realizou-se a medição com a utilização de antena.

Figura 40 – Configuração do experimento na bancada do laboratório.

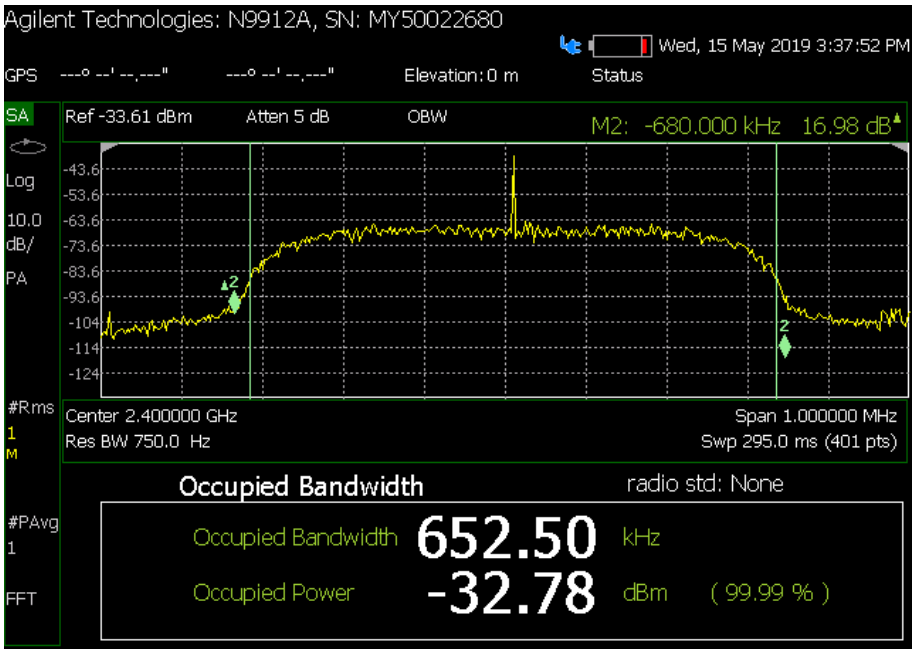
Fonte: O Autor.

Enviou-se o caractere “3” continuamente a uma taxa de $1\frac{Mamostra}{segundo}$. Seguindo o mesmo raciocínio da seção anterior, utilizando a equação 5.1, chegou-se à mesma largura de banda mínima teórica de 675MHz. No analisador de espectro, com conexão via cabo, observou-se que a largura de banda é 762,5kHz conforme figura 42. Este valor é da mesma ordem de grandeza porém em torno de 20% maior da menor largura de banda teórica, o que é razoável para a largura de banda ocupada (considerando 99% da potência do sinal contida nesse intervalo).

Na medição via antena, observou-se o valor de 680kHz para a largura de banda do sinal. Apesar do analisador de espectro calcular automaticamente uma largura de banda, muitas vezes o equipamento não consegue precisar o valor corretamente. Deste

modo, utilizou-se os marcadores para uma medição mais precisa da largura de banda do sinal. Conforme pode-se ver no canto superior direito da figura 41, a diferença entre os marcadores é de 680kHz. Este valor está bem próximo do valor teórico calculado de 675kHz, com um erro de 0,74%.

Figura 41 – Largura de banda medida pelo analisador espectral via antena.



Fonte: O Autor.

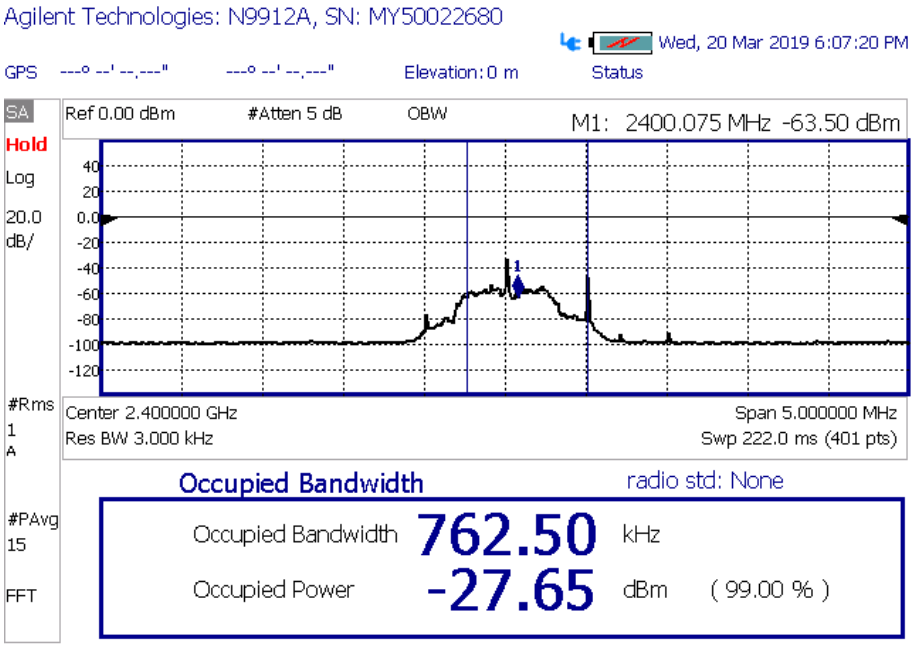
O valor medido no visualizador do GNU Radio também é coerente com o valor medido no analisador de espectro e com o valor teórico. No software foi medida uma ocupação espectral de aproximadamente 688kHz para o lobo principal do sinal (figura 43), que é próximo do valor teórico calculado e do aferido pelo analisador de espectro na medição via cabo (erro de 9,7%) e muito próximo do valor medido via antena (erro de 1,17%). O valor teórico corresponde à largura de banda correspondente ao ponto de 3dB, onde a potência do sinal é atenuada pela metade, e serve como referência para uma ordem de grandeza do valor real, podendo ser considerado um valor mínimo ideal do sistema.

Analisando o arquivo .CSV do analisador de espectro com conexão via cabo, filtrando os dados, foi verificada uma largura de banda de 750kHz para 99,69% da energia, o que é coerente com o valor apresentado pelo equipamento.

O diagrama de constelação do sinal recebido apresentou uma distorção em fase, causada pelo canal em que o sinal foi transmitido (ar), conforme pode-se ver na figura 44.

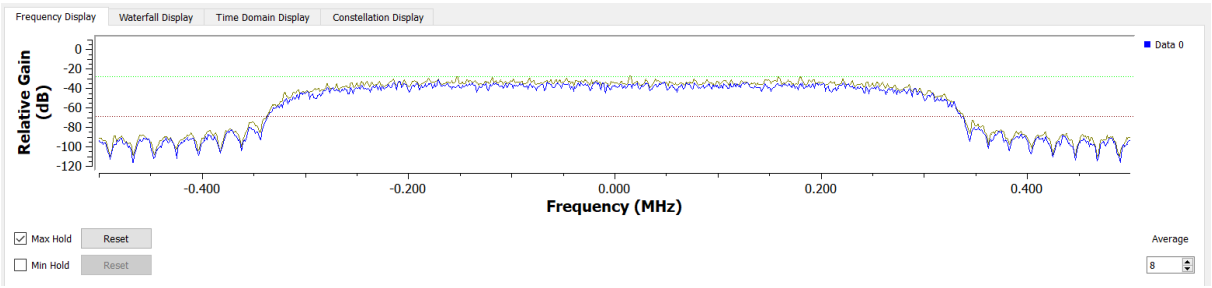
O próximo passo foi realizar a medição do sistema em presença de ruído. Utilizou-se um gerador de RF banda estreita, produzida pelo mesmo equipamento analisador de

Figura 42 – Largura de banda medida pelo analisador espectral via cabo.



Fonte: O Autor.

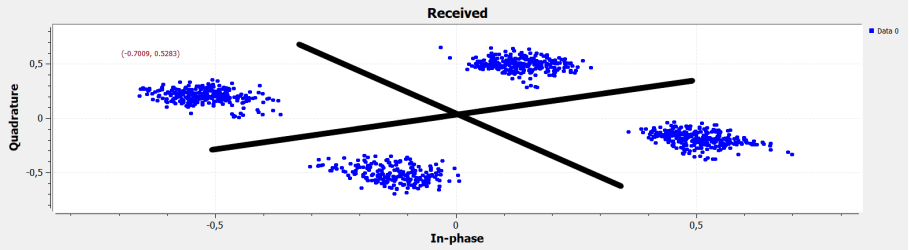
Figura 43 – Largura de banda do sinal enviado medida pelo software GNURadio.



Fonte: O Autor.

espectro. Não foi utilizado nenhum pacote específico para testes de ruído visto que não houve a implementação de um protocolo de comunicação completo. Partiu-se do menor valor de atenuação do equipamento e foi aumentando-se o nível até detectar alguma perturbação nos visualizadores do GNU Radio. De fato, conforme as figuras 45 e 46, percebe-se que houve uma perturbação na constelação e um pico na representação em frequência do sinal. Contudo, com atenuações menores que 24dB, o GNURadio acabou por travar a execução do diagrama. Não foi verificado nenhum caractere errado na recepção do sinal para este nível de ruído. Esta falta de perturbação nos dados recebidos pode ser

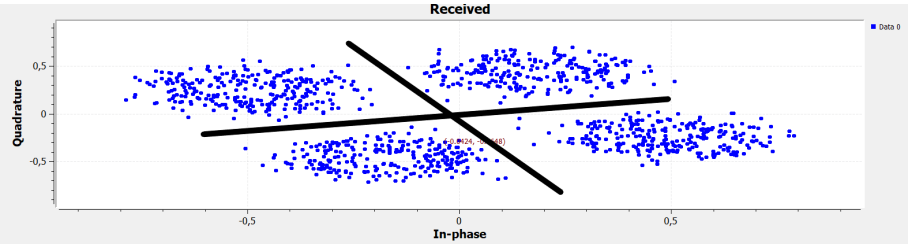
Figura 44 – Diagrama de constelação do sinal recebido.



Fonte: O Autor.

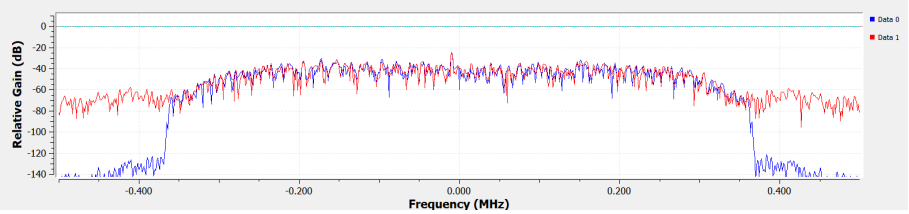
resultante de uma baixa amplitude do ruído ou do fato de o mesmo ser muito estreito em frequência, conforme pode-se ver na representação espectral na figura 46.

Figura 45 – Constelação com ruído.



Fonte: O Autor.

Figura 46 – Representação em frequência com pico de ruído ao centro.



Fonte: O Autor.

6 Conclusão

Este trabalho propiciou um grande aprendizado tanto na área das telecomunicações quanto no uso apropriado das plataformas utilizadas. O processo de desenvolvimento dos estudos de caso necessitou intensa pesquisa sobre o tipo de modulação a ser implementada, bem como o caminho a ser seguido no GNU Radio para conseguir transmitir os dados com sucesso.

Verificou-se que o LimeSDR é uma ferramenta relativamente fácil de trabalhar e apresenta bons resultados. Com o ambiente em diagrama de blocos proporcionado pelo GNU Radio, o desenvolvimento é intuitivo, eliminando as barreiras que a escrita de código pode impor a utilizadores leigos no assunto. No entanto, caso desejado, a programação de blocos específicos pode ser realizada, contemplando um amplo espectro de perfis de desenvolvedores. Ainda assim, apesar de ser uma ferramente poderosa, a programação do LimeSDR via GNU Radio peca na falta de documentação adequada para compreensão dos blocos.

Mesmo contando com uma seção "Documentation" em cada bloco, muitas vezes elas encontram-se sem nenhuma informação. Na plataforma Doxygen encontra-se informações mais voltadas a configurações de software, como o código de certas funções, que informações voltadas ao significado físico do bloco. Em grande parte das vezes, a explicação de um determinado bloco foi encontrada em fóruns, vídeos do YouTube e em outros meios de divulgação de projetos da comunidade desenvolvedora. Estas são as desvantagens de se utilizar uma plataforma de uso livre, com as informações mais dispersas e sem um suporte adequado.

O caráter modular do GNU Radio foi muito explorado, com reaproveitamento de configurações de blocos e conexões entre os diagramas dos estudos de caso, o que poupou grande tempo de desenvolvimento, uma vez que o funcionamento do bloco era desvendado. Os visualizadores do GNU Radio mostraram-se ferramentas importantes para aferimento se a solução estava conforme o esperado. O seu desempenho foi bem satisfatório, como pode-se constatar ao comparar os resultados do analisador de espectro com os visualizadores do GNU Radio.

O LimeSDR mostrou-se extremamente prático em sua utilização. À exceção do problema da atenuação do sinal antes da injeção no rádio, a configuração foi extremamente simples, não sendo necessário recorrer às instruções do fabricante, verdadeiramente uma plataforma *plug-and-play*. A existência de testes e exemplos fornecidos pela LimeMicrosystems foi também essencial para a compreensão dos passos para a configuração do LimeSDR via GNU Radio, servindo como um intermediário essencial entre os dois sistemas.

Mesmo com os problemas encontrados e não solucionados, numa autocrítica, considera-se que o resultado tenha sido positivo. A plataforma é simples, intuitiva e rápida de ser configurada. Espera-se que este trabalho, apresentando algumas configurações e conceitos básicos, assim como com as referências providas, sirva como ponto de partida para futuros usuários que queiram fazer uso do LimeSDR. Como trabalhos futuros, sugere-se a investigação das situações relatadas no capítulo de Trabalhos Futuros, como a implementação da técnica DSSS para complementar o protocolo 802.11b e o estudo da imunidade a ruído do LimeSDR.

7 Trabalhos Futuros

Neste capítulo serão apresentados potenciais desmembramentos e trabalhos futuros com base no que foi estudado e desenvolvido ao longo desta monografia.

7.1 DSSS

Como parte do protocolo 802.11b, era de grande interesse a implementação de espalhamento espectral por meio da técnica DSSS. Inicialmente, tentou-se a implementação por meio de associação de blocos existentes, contudo, para espalhamento em frequência deve-se alterar o tempo entre *bits*, o que não é possível com blocos comuns. Ainda tentou-se usar o bloco *Map* a fim de realizar o mapeamento dos *bits* zero e um para uma sequência de 11 *bits* para cada. Contudo, o bloco *Map*, para o tipo *char*, suportaria no máximo oito *bits*, não sendo possível escolher uma sequência de comprimento maior.

Após pesquisas extensivas, nos fóruns do GNU Radio e no GitHub, encontrou-se uma biblioteca feita por um terceiro para a implementação da técnica DSSS. Para fazer uso da mesma, instalou-se o sistema operacional Ubuntu em uma partição do disco rígido do computador utilizado, assim como o software GNU Radio.

Os blocos desenvolvidos para DSSS são de autoria do usuário GitHub *ericdegroot*. Apesar de seu pacote não conter instrução de instalação, seguiu-se o procedimento padrão com base em informações extraídas de outros desenvolvedores. A princípio, a instalação ocorreu normalmente, com os blocos aparecendo na lista e podendo ser colocados no diagrama. Durante a execução, contudo, foram encontrados problemas de dependências com a ferramenta *swig*, que foi utilizada na criação dos blocos, travando a execução dos mesmos. Procurou-se erros similares a este e as respectivas sugestões de solução, mas não obteve-se sucesso. Como esse pacote não apresentava nenhuma instrução de instalação ou uso, nem retorno de algum outro utilizador, desistiu-se da utilização do mesmo.

Finalmente, veio a ideia de tentar desenvolver por conta blocos para implementação do DSSS. Contudo, ao tentar testar o LimeSDR no sistema operacional Ubuntu, viu-se que o mesmo não funcionou adequadamente, travando a execução do diagrama. Dado estes problemas, decidiu-se tomar a decisão de abandonar a plataforma Ubuntu e retornar ao desenvolvimento no Windows, por questões de cronograma, deixando essa investigação para trabalhos futuros.

7.2 Blocos GNU Radio

Alguns blocos do GNU Radio não apresentaram um comportamento satisfatório durante a sua utilização. O bloco *Vector Source*, por exemplo, não permite o envio único de apenas um caractere, com o diagrama não sendo executado se a opção *Repeat* não estivesse habilitada. Com uma sequência fixa não podendo ser enviada, não foi possível controlar o fluxo dos caracteres enviados.

Outro bloco que apresentou um comportamento aquém do esperado foi o bloco *Channel Model*. Conforme explicado na seção que tratou sobre o experimento com o QPSK simulado, o bloco *Channel Model* acabava por travar a execução do diagrama se um valor maior que 0,3 é colocado no ruído. Isso tornou-se um impedimento importante na simulação, pois não possibilita a simulação de um canal não ideal. Foi por este motivo que optou-se pelo outro método de simulação de ruído.

Fica, então, a sugestão para uma investigação mais aprofundada do comportamento destes dois blocos, de modo que uma simulação mais assertiva seja possível de ser realizada.

7.3 LimeSDR

O próprio LimeSDR apresenta algumas incógnitas que não foram resolvidas neste trabalho. Primeiramente cita-se o fato da necessidade de um bloco atenuador para no máximo uma amplitude de 80% do sinal original antes de injetar o sinal no bloco TX do LimeSDR. Este foi um impeditivo apenas solucionado por experiência prévia de um colega alertando que atenuar o sinal poderia ser uma solução.

Outra limitação da plataforma foi o fato de que, na modulação QPSK, o diagrama parava de funcionar parcialmente, não enviando mais nenhuma amostra. Esta interrupção não obedecia um intervalo de tempo constante, podendo acontecer logo no começo da execução, após um longo tempo, ou nem chegar a acontecer. Ao travar o envio de dados, a mensagem de "popping from TX, x/y samples popped", onde "x" e "y" eram números variáveis a cada caso, aparecia no console do GNU Radio. Procurando-se na internet, viu-se que uma possível causa deste problema é que o bloco transmissor não conseguia receber todas as amostras necessária na hora de enviar, ou seja, o diagrama não estava rápido o suficiente. Como este problema era imprevisível e aleatório, não se conseguiu identificar as condições que o levavam a ocorrer, embora podemos levantar a hipótese de estar relacionado com a intensidade do uso da CPU no instante em que as amostras são perdidas. Esta situação não aconteceu na modulação GFSK.

Finalmente, sugere-se observar a magnitude do sinal de saída no rádio. Apesar de no bloco TX ter sido incluído um ganho de 50dB, o sinal chegava bem fraco nas medições do analisador de espectro. No bloco RX incluiu-se um ganho de 40dB para elevar o nível do

sinal na visualização. Fica a sugestão, então, de investigar melhor a questão da magnitude do sinal de saída do rádio.

Referências Bibliográficas

BERKELEY, U. of C. *Lab 6: Digital Communication with Audio Frequency Shift Keying (AFSK)*. 2019. Disponível em: <https://inst.eecs.berkeley.edu/~ee123/sp15/lab/lab6/Lab6-Part-A-Audio-Frequency-Shift-Keying.html>. Citado na página 22.

CENTER, M. N. *Microsoft to acquire GitHub for 7.5 billion*. 2019. Disponível em: <https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion>. Citado na página 19.

COOK, P.; BONSER, W. Architectural overview of the speakeasy system. *IEEE journal on selected areas in communications*, v. 1, 1999. Citado na página 16.

CTEX. *RDS-Defesa*. 2019. Disponível em: <http://www.eb.mil.br/web/noticias/noticiario-do-exercito/-/asset_publisher/MjaG93KcunQI/content/id/8146777>. Citado na página 18.

DEFESATV. *Centro Tecnológico do Exército recebe os Módulos de Rádio Frequência e Alimentação do Projeto RDS-Defesa*. 2019. Disponível em: <https://bit.ly/2Xw1ftu>. Citado na página 18.

DOXYGEN. *Generate documentation from source code*. 2019. Disponível em: <http://www.doxygen.nl/index.html>. Citado na página 19.

ETSI. *Digital Enhanced Cordless Telecommunications (DECT)*. 2019. Disponível em: <https://www.etsi.org/technologies/dect>. Citado na página 22.

GFSK Example. 2018. Disponível em: <https://wiki.myriadrf.org/Gr-limesdr_Plugin_for_GNURadio>. Citado na página 33.

GITHUB. *GitHub is how people build software*. 2019. Disponível em: <https://github.com/about>. Citado na página 19.

GNURADIO. *Guided Tutorial PSK Demodulation*. 2018. Disponível em: <12.https://wiki.gnuradio.org/index.php/Guided_Tutorial_PSK_Demodulation>. Citado na página 34.

GNURADIO. *What is GNURadio?* 2018. Disponível em: <https://www.gnuradio.org/about/>. Citado na página 18.

GNURADIO. *GNU Radio Manual and C++ API Reference 3.7.13.4*. 2019. Disponível em: <https://www.gnuradio.org/doc/doxygen/index.html>. Citado na página 19.

GNURADIO. *HowToUse*. 2019. Disponível em: <https://wiki.gnuradio.org/index.php/HowToUse>. Citado na página 20.

HETTING, C. *How a 1998 meeting with Steve Jobs gave birth to Wi-Fi*. 2019. Disponível em: <https://wifinowevents.com/news-and-blog/how-a-meeting-with-steve-jobs-in-1998-gave-birth-to-wi-fi/>. Citado na página 23.

IEEE. *IEEE Std. 802.11*. [S.l.], 1997. Citado na página 23.

JORNET, J. M. *EE450/550: Principles of Networking, Chapter 1: Introduction to Telecommunication Networks*. [S.l.]: University at Buffalo, 2015. Nenhuma citação no texto.

JORNET, J. M. *EE450/550: Principles of Networking, Chapter 3: Physical Layer*. [S.l.]: University at Buffalo, 2015. Nenhuma citação no texto.

KENNEDY, G.; DAVIS, B. *Electronic Communication Systems*. [S.l.]: McGraw-Hill, 1992. Citado na página 20.

LACKEY, R.; UPMAL, D. W. Speakeasy: The military software radio. *Communications Magazine, IEEE*, v. 33, p. 56 – 61, 06 1995. Citado na página 16.

LATHI, B.; DING, Z. *Modern Digital and Analog Communication Systems*. 4. ed. [S.l.]: Oxford University Press, 2010. Citado na página 16.

LIMEMICROSYSTEMS. 2018. Disponível em: <https://limemicro.com/products/boards/limesdr/>. Citado na página 30.

LIMESDR. 2018. Disponível em: <https://myriadrf.org/projects/component/limesdr/>. Citado 2 vezes nas páginas 30 e 31.

LIMESDR Hardware Installation. 2018. Disponível em: <https://wiki.myriadrf.org/LimeSDR_Hardware_Installation>. Citado na página 31.

LIMESDR-USB Quick Test. 2018. Disponível em: <https://wiki.myriadrf.org/LimeSDR-USB_Quick_Test>. Citado na página 31.

MACHADO, E. R. *Implementação de Sistema de Comunicação ADS-B por meio de dispositivos SDR*. 2018. Trabalho de Diplomação, Curso de Engenharia Elétrica, UFRGS. Citado na página 18.

MEGHDADI, V. Ber calculation. 2008. Citado na página 26.

MICROSYSTEMS, L. *Lime SDR Datasheet*. 2018. Disponível em: <https://wiki.myriadrf.org/Gr-limesdr_Plugin_for_GNURadio#Windows_installation>. Citado na página 35.

MITOLA, J. Software radios-survey, critical evaluation and future directions. In: *[Proceedings] NTC-92: National Telesystems Conference*. [S.l.: s.n.], 1992. p. 13/15–13/23. Citado 2 vezes nas páginas 16 e 17.

MYRIAD. *Tutorial instalação de blocos LimeSDR para GNURadio*. 2018. Disponível em: <http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm>. Nenhuma citação no texto.

NUTAQ. *A Short Story of Software Defined Radios*. 2019. Disponível em: <https://www.nutaq.com/blog/short-history-software-defined-radio-sdr-technology>. Citado na página 18.

QPSK Wikipédia. 2018. Disponível em: <https://en.wikipedia.org/wiki/Phase-shift_keying#Quadrature_phase-shift_keying_.28QPSK.29>. Citado na página 25.

Rivet, F. et al. From software-defined to software radio: Analog signal processor features. In: *2009 IEEE Radio and Wireless Symposium*. [S.l.: s.n.], 2009. p. 348–351. ISSN 2164-2958. Citado na página 17.

SDRFORUM. *SDRF Cognitive Radios Definition working document SDRF-06-R-0011-V1.0.0*. [S.l.], 2007. Citado na página 16.

SMIL, V. Embodied energy: Mobile devices and cars. *IEEE Spectrum*, 2016. Citado na página 15.

SPENCH. *GNU Radio Notes*. 2019. Disponível em: <https://wiki.spench.net/wiki/GNU_Radio_Notes>. Citado 2 vezes nas páginas 36 e 37.

STACKOVERFLOW. *Explore nossas perguntas*. 2019. Disponível em: <https://pt.stackoverflow.com/>. Citado na página 20.

Staple, G.; Werbach, K. The end of spectrum scarcity [spectrum allocation and utilization]. *IEEE Spectrum*, v. 41, n. 3, p. 48–52, March 2004. ISSN 0018-9235. Citado na página 17.

STERN, H.; MAHMOUD, S. *Communications Systems*. [S.l.]: Pearson Prentice Hall, 2004. Citado na página 26.

TEKTRONIX. *Wi-Fi: Overview of the 802.11 Physical Layer and Transmitter Measurements*. [S.l.], 2013. Citado 3 vezes nas páginas 23, 24 e 27.

WECA. *Wireless Ethernet Compatibility Alliance (WECA) Announces Independent Test Lab and Wi-Fi Technology Brand*. 2019. Disponível em: <https://www.wi-fi.org/news-events/newsroom/wireless-ethernet-compatibility-alliance-weca-announces-independent-test-lab>. Citado na página 23.

WIKIPEDIA. *An example of binary FSK*. 2019. Disponível em: <https://en.wikipedia.org/wiki/Frequency-shift_keying#/media/File:Fsk.svg>. Citado na página 21.

WIKIPEDIA. *Normalized Gaussian curves with expected value mi and variance sigma2. The corresponding parameters are , b = mi and c = sigma*. 2019. Disponível em: <https://en.wikipedia.org/wiki/Gaussian_function#/media/File:Normal_Distribution_PDF.svg>. Citado na página 22.