MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE ELÉTRICA

1° Ten PAULO HENRIQUE SALGUEIRO COSTA
JEAN DA SILVA PAIXÃO

TRANSMISSÃO OFDM EM AMBIENTE GNU RADIO

Rio de Janeiro
2017

INSTITUTO MILITAR DE ENGENHARIA

1° Ten PAULO HENRIQUE SALGUEIRO COSTA
JEAN DA SILVA PAIXÃO

TRANSMISSÃO OFDM EM AMBIENTE GNU RADIO

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Comunicações do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Comunicações.

Orientador: Prof. Ernesto Leite Pinto - D.Sc.
Co-Orientador: Prof. Alexandre Amorim Pereira Júnior - D.Sc.

Rio de Janeiro
2017

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro – RJ CEP: 22290-270

621.384 Costa, Paulo Henrique Salgueiro

C837t Transmissão OFDM em ambiente GNU radio / Paulo Henrique Salgueiro Costa; Jean da Silva Paixão; orientados por Ernesto Leite Pinto; Alexandre Amorim Pereira Júnior – Rio de Janeiro: Instituto Militar de Engenharia, 2017.

43p. : il.

Projeto de Fim de Curso (PROFIC) – Instituto Militar de Engenharia, Rio de Janeiro, 2017.

1. Curso de Engenharia de Comunicações – Projeto de Fim de Curso. 2. Radio. I. Paixão, Jean da Silva. II. Pinto, Ernesto Leite. III. Pereira Junior, Alexandre Amorim. IV. Título. V. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

1° Ten PAULO HENRIQUE SALGUEIRO COSTA
JEAN DA SILVA PAIXÃO

TRANSMISSÃO OFDM EM AMBIENTE GNU RADIO

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Comunicações do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Comunicações.

Orientador: Prof. Ernesto Leite Pinto - D.Sc.

Co-Orientador: Prof. Alexandre Amorim Pereira Júnior - D.Sc.

Aprovado em 06 de Outubro de 2017 pela seguinte Banca Examinadora:

Prof. Ernesto Leite Pinto - D.Sc. do IME - Presidente

Prof. Alexandre Amorim Pereira Júnior - D.Sc. do IME

Prof. André Luís de Souza Araújo - M.Sc. do IME

Prof. Felipe Aurélio Caetano de Bastos - D.Sc. do IME

Rio de Janeiro
2017

A meu filho Gregório, minha esposa Jéssica, meu pai João, minha mãe Zita, minhas irmãs Emília e Rosilene, meu cunhado Felipe e meu sobrinho Bernardo - Jean

AGRADECIMENTOS

Agradeço a todas as pessoas que contribuíram de certa forma para que esse dia chegasse.

Inicialmente e de forma mais eminente, a Deus, Nosso Senhor, que me gerou, me recebeu, me nutre e perdoa os meus pecados.

Em seguida aos meus pais e às minhas irmãs, que me acolheram, me protegeram, me educaram, me sustentaram e me inspiram.

À minha esposa, que caminha ao meu lado e me ajuda a superar os desafios da vida.

Ao meu filho, por me fazer sorrir, mesmo quando tudo me traz tristeza.

Aos padres, por purificar a minha alma.

Aos meus demais familiares e amigos, que me motivaram e me aconselharam.

Aos professores, que me deram senso crítico.

De forma especial, aos professores Ernesto e Alexandre, que me orientaram de forma brilhante, com dedicação, apoio constante e foco em excelência neste trabalho.

Aos meus colegas de trabalho, que me fazem dar o máximo de mim.

Aos meus inimigos, que muitas vezes me fizeram ser uma pessoa melhor.

Jean da Silva Paixão

Agradeço especialmente a Deus por me sustentar com saúde e por permitir que eu iniciasse e concluísse esta grandiosa jornada que é se formar no Instituto Militar de Engenharia.

Agradeço aos meus pais, Ecidelmon e Glória Christina, e irmão, Vinícius, que sempre investiram e acreditaram nos meus sonhos sendo sempre minha fortaleza nas adversidades.

Agradeço a minha namorada Rosária por sempre me motivar e me acompanhar mesmo na dificuldade da distância.

Agradeço aos meus orientadores, Prof. Ernesto Leite e Prof. Alexandre Amorim, pelas suas dedicações e paciência no decorrer desse ano e pelos momentos de trabalho compartilhados.

Agradeço à meus camaradas do IME, em especial aos meus irmãos de caminhada Daniel, Luiz Felipe e Johnatan, aqueles que dividiram suas vidas comigo durante esses cinco anos.

Paulo Henrique Salgueiro Costa

“

Non nobis Domine, non nobis, sed nomini tuo da gloriam. Super misericordia tua et veritate tua nequando dicant gentes ubi est Deus eorum? Deus autem noster in caelo omnia quaecumque voluit fecit.

”

SALMO 113, 9-11

SUMÁRIO

LISTA DE ILUSTRAÇÕES 8

LISTA DE SIGLAS 10

1 INTRODUÇÃO 13

2 REVISÃO TEÓRICA 15
2.1 Orthogonal Frequency Division Multiplexing 15
2.2 Rádio definido por software 17
2.3 A plataforma GNU Radio 18
2.4 GNU Radio Companion 19

3 IMPLEMENTAÇÃO DE SISTEMAS OFDM NO GNU RADIO . . 21
3.1 Implementação OFDM no GNU Radio Companion 21
3.1.1 Fluxograma da Transmissão 21
3.1.2 Fluxograma da Recepção 24
3.2 Implementação OFDM diretamente por códigos python/c++ 26
3.2.1 O arquivo ofdm.py 26
3.3 Transmissor OFDM 26
3.4 Receptor OFDM 28
3.4.1 Os módulos *ofdm_receiver* e *ofdm_frame_sink* 29

4 SIMULAÇÕES REALIZADAS E RESULTADOS OBTIDOS 31
4.1 Simulações na implementação GNU Radio 31
4.1.1 Executando os arquivos *benchmark_tx.py* e *benchmark_rx.py* 31
4.1.2 Resultados Obtidos 32
4.2 Simulações na implementação GNU Radio Companion 34
4.2.1 Transmissão OFDM BPSK e QPSK 35

5 CONCLUSÃO E TRABALHOS FUTUROS 37

6 REFERÊNCIAS BIBLIOGRÁFICAS 38

7 APÊNDICES 39
7.1 APÊNDICE 1: Tela de configuração dos arquivo *benchmark* 40

7.2 APÊNDICE 2: Código *benchmark_ tx.py utilizado* 41

LISTA DE ILUSTRAÇÕES

FIG.2.1 Técnica FDM. Fonte: Tutors (2017). 15
FIG.2.2 Comparação entre FDM e OFDM.Fonte: Instruments (2017). 16
FIG.2.3 Sistema OFDM. Fonte: Communication (2017). 16
FIG.2.4 Subportadoras-piloto. Fonte: Communication (2017). 17
FIG.2.5 Intervalo de guarda. Fonte: Instruments (2017). 17
FIG.2.6 *Frontend* de RF 18
FIG.2.7 Estrutra em Blocos do GNU Radio. Fonte: Project (2017). 19
FIG.2.8 Blocos Hierárquicos. Fonte: Project (2016). 20

FIG.3.1 Principais blocos para uma implementação de sistema OFDM no GRC, Transmissor. 21
FIG.3.2 Bloco OFDM Carrier Allocator. 22
FIG.3.3 Bloco da FFT. 23
FIG.3.4 Bloco do OFDM Cyclic Prefixe. 23
FIG.3.5 Principais blocos para uma implementação de sistema OFDM a nível GRC, Receptor. 24
FIG.3.6 Bloco do OFDM Channel Estimation. 25
FIG.3.7 Bloco do OFDM Frame Equalizer. 25
FIG.3.8 Bloco do OFDM Serializer. 26
FIG.3.9 Constituição do Payload. 27
FIG.3.10 Sequência de processamento da classe *ofdm_mod*. 27
FIG.3.11 Módulos utilizados em *receive_path.py*. Fonte: Nguyen (2013). 29
FIG.3.12 Módulos utilizados em *ofdm_receiver.py*. Fonte: Nguyen (2013). 29
FIG.3.13 Máquina de estados do módulo *ofdm_frame_sink*. Fonte: Youssef et al. (2012). 30

FIG.4.1 Sinal modulado BPSK 33
FIG.4.2 Sinal modulado QPSK 33
FIG.4.3 Símbolos BPSK na recepção. 33
FIG.4.4 Símbolos QPSK na recepção. 33
FIG.4.5 Símbolos BPSK na recepção. 34
FIG.4.6 Símbolos QPSK na recepção. 34
FIG.4.7 Comportamento da TEB medida. 36

FIG.7.1 Captura de tela do comando *python benchmark_ tx.py - - help.* 40

LISTA DE SIGLAS

RDS	Rádio Definido por Software
OFDM	Orthogonal Frequency Division Multiplexing
GNU	GNU não é Unix
GRC	GNU Radio Companion
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
DC	Direct Current

RESUMO

O presente trabalho é uma introdução ao desenvolvimento de um sistema de transmissão de Rádio Definido por Software. Foi escolhida a técnica de multiplexação OFDM para melhor aproveitamento do espectro. O GNU Radio foi escolhido como software para esse propósito. Considera-se que o leitor já possua conhecimento básico em sistemas digitais e técnicas de transmissão para completa compreensão do texto. Nesse trabalho será discutido o que é preciso para se desenvolver de forma simplificada um enlace OFDM no GNU Radio, como funcionam blocos importantes nos transmissores e receptores OFDM e os resultados de alguns dos experimentos realizados para auxiliar na compreensão dos processos.

ABSTRACT

The present work is an introduction to the development of a Software Defined Radio transmission system. The OFDM technique was chosen in order to improve the spectral usage. For this purpose, the GNU Radio was chosen as software. It is considered that the reader has basic knowledge in digital systems and transmission techniques in order to fully understand of the text. Here, we will discuss what is needed to develop in a simplified way, an OFDM link in GNU Radio, how important blocks work in OFDM transmitters and receivers and the results of some of the experiments made to comprehend the processes of OFDM transmission.

1 INTRODUÇÃO

A evolução no hardware dos computadores da segunda metade do século passado, e consequentemente, a evolução no processamento digital de sinais, alavancada com a implementação da FFT, possibilitou o ressurgimento de uma série de invenções que estavam engavetadas.

Os processadores digitais abrem um leque enorme de possibilidades de evolução dos sistemas de codificação, transmissão e recepção, mas inserem uma complexidade de implementação. Enquanto que numa implementação convencional é possível se analisar o comportamento de cada componente de um sistema de forma isolada por meio de um osciloscópio ou equivalente, em um Rádio Definido por Software, é preciso entender bem a função de cada etapa intermediária (bloco) para poder se verificar se o resultado obtido foi como esperado ou não.

Existem inúmeras possibilidades de se elaborar um Rádio Definido por Software mas o GNU Radio chama a atenção pelo fato de ser gratuito e colaborativo. Uma grande desvantagem reside no fato de não existir um canal de suporte, exceto os fóruns na internet. Isso dificulta muito o trabalho dos iniciantes, que ficam sem referência de início.

A crescente demanda por altas taxas de transmissão, a limitação da largura de banda, o ambiente de propagação sujeito a reflexões por multipercurso e a consequente atenuação do sinal por desvanecimento seletivo em frequência são exemplos de motivações para o aprofundamento no estudo de sistemas OFDM. Esta técnica está presente nas tecnologias mais atuais que vão desde a Televisão Digital de alta definição, as comunicações sem-fio e o desenvolvimento de rádios cognitivos. Demonstra-se, portanto, a importância deste presente estudo, uma vez que visa a implementação de um sistema OFDM em ambiente *open source* e coloborativo.

Ressalta-se, portanto, que uma das principais contribuições do presente trabalho é justamente a consolidação da descrição dos componentes dos sistemas OFDM disponibilizados pelo GNU Radio mencionados. Para cumprir este objetivo, foram realizados estudos de dois tipos de implementação de um sistema OFDM em GNU Radio. Primeiramente, foram realizadas simulações diretamente com os códigos em Python, tais arquivos são descritos pelos trabalhos Nguyen (2013) e Youssef et al. (2012). A análise segundo a implementação GNU Radio Companion foi realizada com fluxogramas disponíveis em API (2016).

Na seção 2, os conceitos fundamentais para a compressão do trabalho realizado são apresentados.

Na seção 3, há uma descrição detalhada do funcionamento dos transmissores e receptores OFDM no GNU Radio. Somente a partir do conhecimento dos componentes desses transmissores e receptores, que são bem específicos e esparsamente encontrados nos fóruns, que o projeto pode evoluir.

Na seção 4, os principais resultados do presente trabalho são apresentados. Destaca-se que não são relatados todos os experimentos realizados, uma vez que muitos deles serviram apenas para propiciar uma visão geral e compreensão do que está explicado na seção 3.

Na seção 5, conclui-se com observações gerais a respeito do trabalho bem como sugestões para trabalhos futuros.

Por fim, apresenta-se uma seção de apêndice com informações úteis relacionadas ao trabalho.

2 REVISÃO TEÓRICA

2.1 ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING

A técnica de multiplexação por divisão em frequências ortogonais, OFDM é uma evolução da técnica, multiplexação por divisão em frequência, FDM. A multiplexação consiste em transmitir vários canais ao mesmo tempo e a divisão em frequência, nesse caso, significa que cada canal ocupará uma faixa de frequências exclusiva.

A Figura 2.1 abaixo ilustra como os canais são divididos em frequências diferentes.

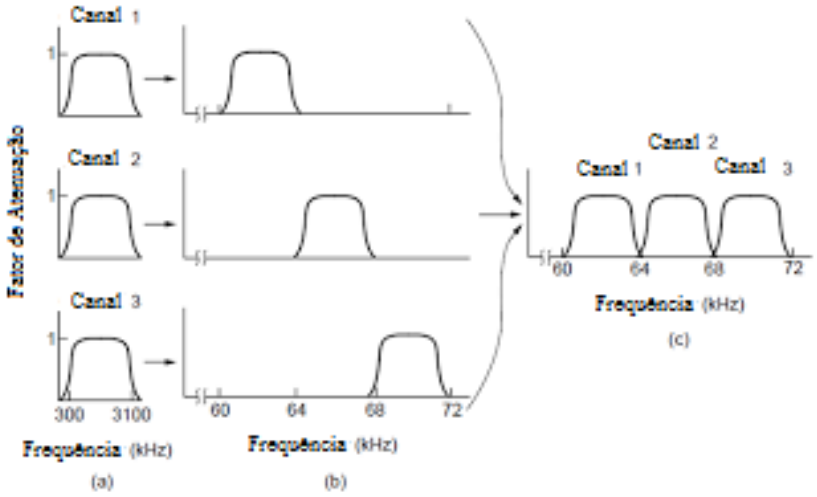

FIG. 2.1: Técnica FDM. Fonte: Tutors (2017).

A técnica FDM foi planejada de modo que cada canal ocupasse uma faixa exclusiva do espaço, ou seja, não houvesse interseção entre os canais. A separação do sinal de cada canal, referente a uma subportadora distinta, nesse caso, poderia ser realizada com a utilização de filtros passa-faixa, um filtro para cada subportadora.

A largura de banda para uma transmissão é um recurso limitado e portanto, surge a necessidade de se tentar transmitir o máximo de informação possível dentro da menor largura de banda possível, de modo a otimizar a utilização do espectro.

A técnica OFDM surge nesse contexto. Baseando-se no fato de que o produto escalar entre senoídes de duração limitada com frequências apropriadamente escolhidas em função desta duração é zero, foi possível empregar subportadoras ortogonais entre si, de modo que dois canais distintos sejam ortogonais entre si. A Figura 2.2 abaixo compara as

técnicas FDM e OFDM e mostra a economia de largura de banda proporcionada pela técnica OFDM.

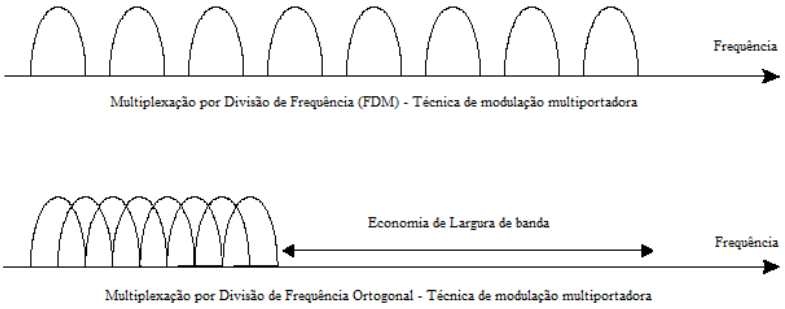

FIG. 2.2: Comparação entre FDM e OFDM.Fonte: Instruments (2017).

A desvantagem da OFDM original em relação a FDM é sua demodulação, que é bem mais complexa. Com a evolução das técnicas de processamento digital de sinais e surgimento da FFT, esse problema pôde ser resolvido e os transmissores e receptores OFDM puderam ser desenvolvidos.

A Figura 2.3 abaixo mostra um diagrama de blocos de um transmissor e um receptor OFDM.

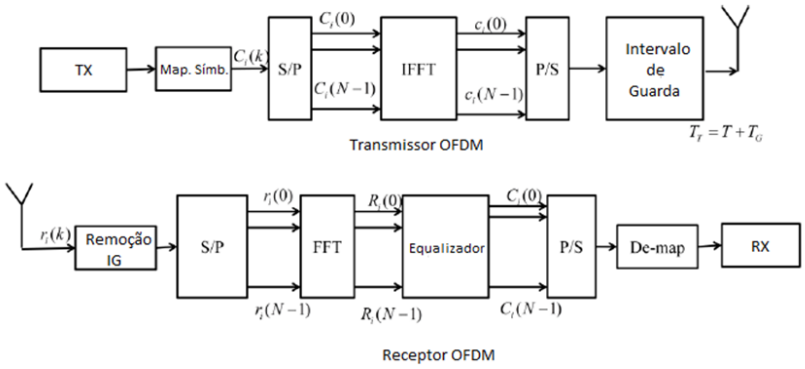

FIG. 2.3: Sistema OFDM. Fonte: Communication (2017).

Os símbolos complexos oriundos da modulação empregada (BPSK, QPSK, etc) são aplicados à IFFT, a fim de gerar o símbolo OFDM. No entanto, nem todas as subportadoras são utilizadas com canais. Algumas são reservadas para portadoras-piloto, que servirão para realizar o sincronismo na demodulação e para uma estimativa do canal para uma modulação adaptativa.

A Figura 2.4 abaixo mostra um exemplo de como as subportadoras são alocadas em um sinal OFDM.

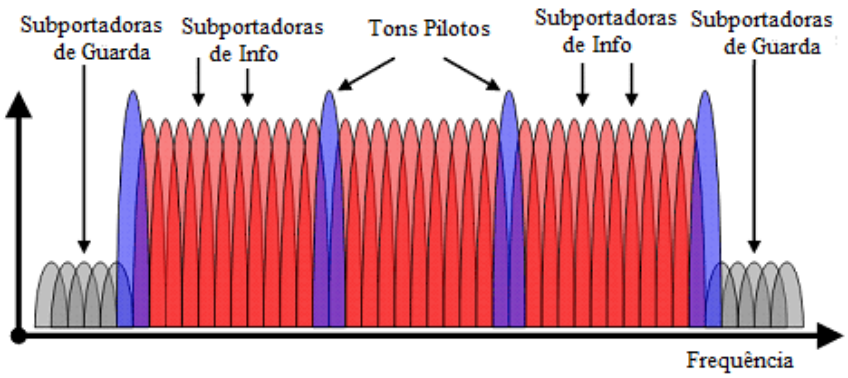

FIG. 2.4: Subportadoras-piloto. Fonte: Communication (2017).

Adiciona-se então um prefixo cíclico para se criar um intervalo de guarda. Isso é necessário porque um símbolo limitado em frequência é ilimitado no tempo. O intervalo de guarda é uma proteção contra a interferência entre símbolos adjacentes. Como ilustrado na Figura 2.5

O sinal é então transmitido. Para a demodulação realiza-se o processo inverso, retira-se o prefixo cíclico; aplica-se a FFT; realiza-se o mapeamento inverso; e serializa-se a informação.

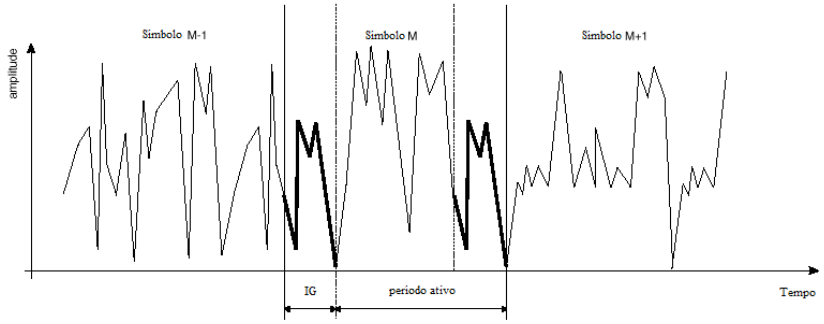

FIG. 2.5: Intervalo de guarda. Fonte: Instruments (2017).

2.2 RÁDIO DEFINIDO POR SOFTWARE

A evolução do processamento digital de sinais possibilitou o desenvolvimento de sistemas digitais de comunicação. Os componentes eletrônicos misturadores, filtros, amplificadores, moduladores, demoduladores e detectores puderam ser substituídos por equivalentes lógicos, com a mesma funcionalidade.

Um rádio definido por software possui todas e ou grande parte de seus componentes implementados em software. Para sua operação apenas é necessário um computador conectado a um *front-end* de RF e uma antena, como indicado na Figura 2.6 abaixo.

As principais vantagens de se utilizar um rádio definido por software residem no fato

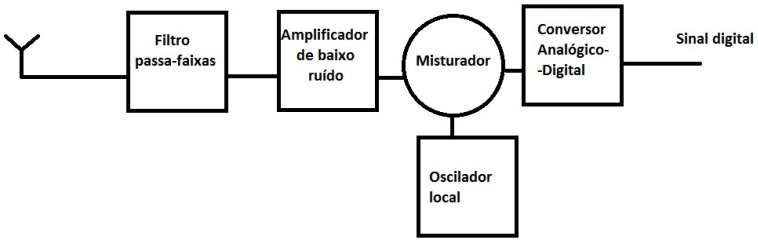

FIG. 2.6: *Frontend* de RF

de sua enorme adaptabilidade. Enquanto em um rádio convencional é preciso projetar minuciosamente os componentes para se atingirem as especificações de transmissão e recepção, para um rádio definido por software, é muito improvável que seja necessário substituir um componente do *front-end* de RF ou a antena, exceto para se trabalhar em uma faixa diferente de frequências. Portanto, os projetos com rádio definido por software são mais fáceis de se alterar e possuem custo de desenvolvimento menor.

Além disso, o processamento digital de sinais ainda proporciona a utilização de técnicas de codificação de canal, para diminuir a probabilidade de se detectar um bit errado e técnicas de criptografia, para impedir que um agente estranho tenha acesso aos dados.

2.3 A PLATAFORMA GNU RADIO

A plataforma GNU Radio é uma ferramenta colaborativa para o desenvolvimento de Rádios Definidos por Software.

O GNU Radio foi desenvolvido para Linux, mas também há possibilidade de ser utilizado em Windows ou Mac.

No GNU Radio são usadas duas linguagens de programação para melhor desempenho.Devido a seu maior poder computacional, a linguagem C++, é utilizada para implementar operações de processamento digital de sinais como por exemplo filtragem, operações de I/O, FFT/IFFT, operações de codificação/decodificação e modulação/demodulação. De um modo geral, a linguagem Python, possui códigos mais enxutos e fáceis de serem compreendidos sendo utilizada principalmente para conexão dos blocos funcionais e o controle do fluxo de dados entre os blocos . Além disso, por ser orientada a objetos, possibilita a utilização de blocos hierárquicos, como será comentado na próxima seção. A interface de interoperação entre Python e C++ é denominada Gerador de Interfaces e empacotador simples (SWIG).

Os blocos do GNU Radio diferentes tipos de dados, dentre eles: complexo (8 Bytes),

float (4 Bytes), inteiro (4 Bytes), short (2 Bytes) e char. Muitos blocos do GNU Radio foram nomeados de modo que seja possível identificar qual é o tipo de entrada e qual é o tipo de saída. Por exemplo, em "grs_rms_cf", o sufixo "cf" significa que a entrada é complexa e a saída é um float.

Há duas funções principais no núcleo de processamento do GNU Radio, forecast() e work(). A função forecast() estima quantas unidades de entrada são necessárias para o módulo produzir um dado número de saídas e work() é a função que produz a saída a partir da entrada.

A taxa de informação de duas entradas distintas em um mesmo bloco não precisa ser a mesma, mas todas as saídas devem possuir a mesma taxa. Há um buffer tanto para os fluxos de entrada, quanto de saída. Os blocos se comunicam por meio desses buffers, que são do tipo FIFO, isto é, cada entrada é única, mas uma saída pode ir para mais de um lugar.

Em Python, a conexão dos blocos ocorre por meio da função connect, que indica onde as entradas e saídas serão relacionadas.

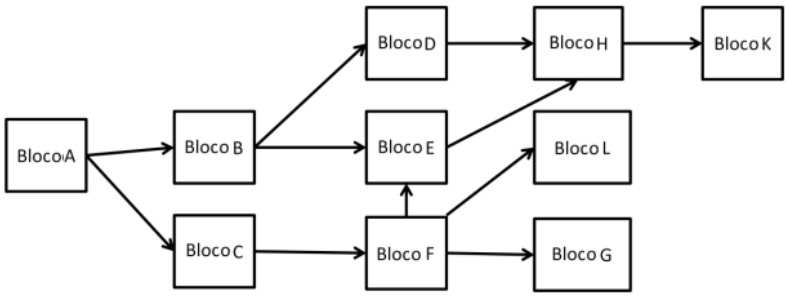

FIG. 2.7: Estrutra em Blocos do GNU Radio. Fonte: Project (2017).

2.4 GNU RADIO COMPANION

O GNU Radio Companion (GRC) é uma interface gráfica para auxiliar no projeto de sistemas de rádio pelo GNU Radio. O sistema implementado no GRC é representado por um fluxograma que é organizado em blocos e setas, de modo que cada bloco faz um processamento de dados e transmite ao próximo bloco para que esse processo continue. A figura 2.7 acima ilustra essa situação.

Os blocos hierárquicos são estruturas formadas pela composição de dois ou mais blocos e ligações, para formar blocos maiores. Dessa forma é possível um grande aproveitamento dos blocos existentes e uma integração mais fácil, proporcionando um ganho de escala para o desenvolvimento de sistemas de RDS.

A Figura 2.8 mostra como é a estrutura de um bloco hierárquico.

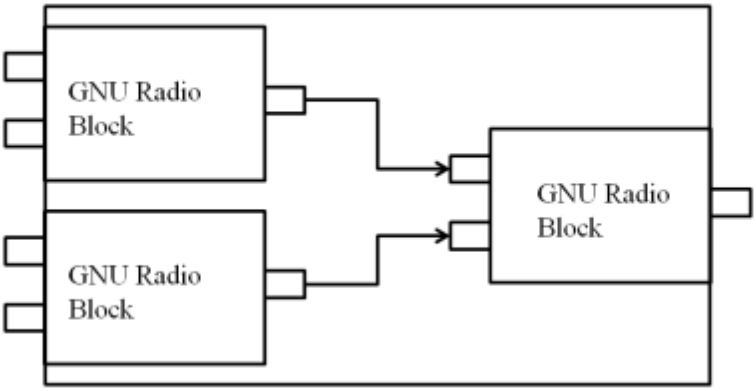

FIG. 2.8: Blocos Hierárquicos. Fonte: Project (2016).

3 IMPLEMENTAÇÃO DE SISTEMAS OFDM NO GNU RADIO

A distribuição do GNU Radio contém duas implementações de sistema OFDM, uma que emprega fluxogramas e blocos do GRC e outra que é realizada diretamente em códigos Pyhton que utilizam as classes disponibilizadas pelo GNU Radio. O objetivo dessa seção é mostrar como os sistemas são implementados nos dois níveis: no GNU Radio Companion e nos códigos em Python. No primeiro tópico a explicação será baseada nos blocos *default* já existentes no GRC e, no segundo, os códigos *python* do exemplo de OFDM, presentes no diretório do programa, elucidarão o entendimento.

3.1 IMPLEMENTAÇÃO OFDM NO GNU RADIO COMPANION

A implementação de um sistema de transmissão OFDM baseada no GRC é constituída de blocos interconectados que formam um fluxograma de processamento digital dos sinais envolvidos. Por exemplo, na transmissão, basicamente é necessária a conexão em cascatas de três blocos: *OFDM Carrier Allocator*, *FFT* e *OFDM Cyclic Prefixer*. A seguir serão analisados dois exemplos de fluxogramas para implementação de um sistema OFDM na transmissão e na recepção.

3.1.1 FLUXOGRAMA DA TRANSMISSÃO

A Figura 3.1 a seguir representa o fluxograma de um transmissor OFDM, implementado no GNU Radio Companion.

O primeiro bloco da transmissão, chamado *OFDM Carrier Allocator*, tem como entrada símbolos pré OFDM, ou seja, números complexos resultantes do mapeamento da modulação de bits de cabeçalho, inseridos com o objetivo de controlar a transmissão, ou de bits de informação propriamente dita. Este bloco realiza a atribuição destes símbolos complexos às portadoras OFDM, ou seja, cria um vetor de símbolos do tamanho da IFTT no qual alguns elementos são símbolos de cabeçalho/informação e outros são tons pilotos.

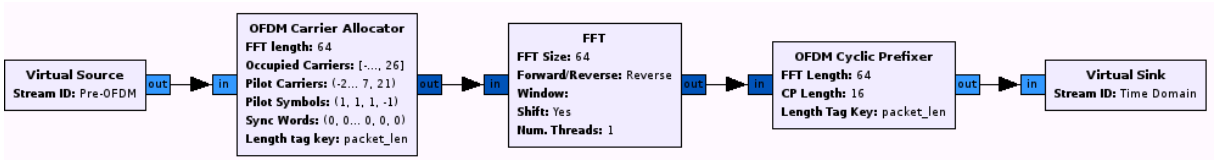

FIG. 3.1: Principais blocos para uma implementação de sistema OFDM no GRC, Transmissor.

Após a aplicação da IFFT (conforme exposto na seção 2.1) é necessária a introdução do intervalo de guarda, que neste exemplo é realizada pelo bloco *OFDM Cyclic Prefixer* por meio da adição de um prefixo cíclico.

A seguir, apresenta-se uma lista [1]dos principais parâmetros dos blocos utilizados neste exemplo de transmissor OFDM.

⋆ **OFDM Carrier Allocator**

Este bloco instancia um objeto da classe *DIGITAL_API ofdm_carrier_allocator_cvc*, implementada no arquivo *ofdm_carrier_allocator_cvc*. Possui como entrada uma *tagged stream* de símbolos (números complexos) e sua *tag* contém o número de símbolos em um quadro. Esta *stream* de entrada é transformada em uma *stream* de vetores de símbolos de tamanho igual ao número de pontos da IFFT (número de portadoras) que será executada logo adiante. Este bloco permite a inserção de portadoras pilotos.

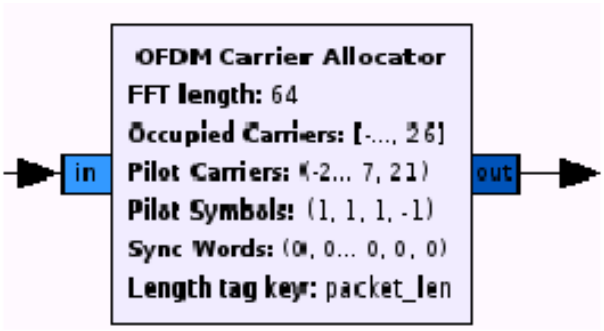

FIG. 3.2: Bloco OFDM Carrier Allocator.

Seus parâmetros principais são:

a) **FFT_length**: Tamanho da FFT (inteiro positivo) que será utilizada, ou seja, o número de portadoras empregadas no sistema OFDM. Naturalmente, deve ser idêntico ao parâmetro do bloco da FFT.

b) **Occupied_Carriers**: Vetor de inteiro que indexa as portadoras ocupadas. Reserva-se o índice 0 para a portadora DC, sendo os índices negativos paras as portadoras abaixo da DC e os positivos para as acima.

c) **Pilot_Carriers**: Vetor de inteiros que indexa as portadoras pilotos.

d) **Pilot_Symbols**: Vetor de complexos que representam os respectivos símbolos dos tons pilotos.

[1]Retirada do site API (2016).

⋆ **FFT**

Este bloco instancia um objeto da classe *FFT_API fft_vcc*, implementada no arquivo *fft_vcc*. Realiza a transformada discreta de Fourier de um vetor de números complexos, símbolos, sendo sua saída também um vetor de complexos. Neste mesmo bloco pode-se realizar a transformada direta ou inversa de Fourier.

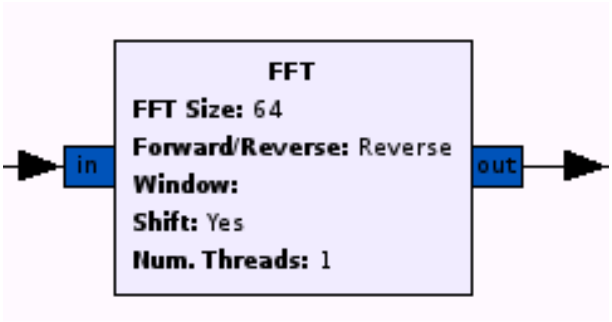

FIG. 3.3: Bloco da FFT.

Seus parâmetros principais são:

a) **FFT_Size**: Tamanho da FFT (inteiro positivo).

b) **Forward\Reverse**: Parâmetro para indicar se a FFT utilizada será a direta ou inversa.

c) **Window**: Tipo de janelamento que será utilizado na implementação da FFT.

⋆ **OFDM Cyclic Prefixer**

Este bloco instancia um objeto da classe *DIGITAL_API ofdm_cyclic_prefixer*, implementada no arquivo *ofdm_cyclic_prefixer*. Recebe os símbolos OFDM no domínio do tempo e adiciona o prefixo cíclico ao quadro, realizando também a formatação de pulso sendo este do tipo cosseno levantado.

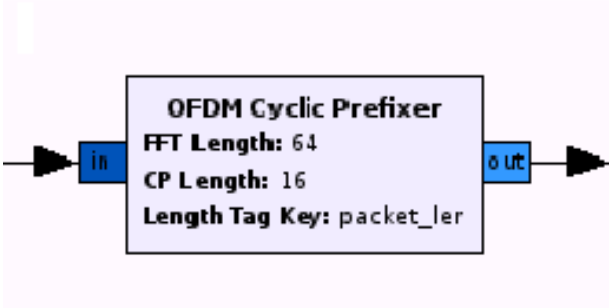

FIG. 3.4: Bloco do OFDM Cyclic Prefixe.

Seu principal parâmetro é:

a) **CP_Length**: Número de amostras do prefixo cíclico, normalmente utiliza-se um quarto do tamanho da FFT.

3.1.2 FLUXOGRAMA DA RECEPÇÃO

Na representação da Figura 3.5 abaixo, o símbolo OFDM tomado como entrada do bloco FFT é uma *stream* de vetores de símbolos referentes ãs informações de cabeçalhos ou aos dados propriamente ditos. O bloco *OFDM Channel Estimation* é responsável por obter estimativas da resposta do canal, que serão empregadas pelo bloco seguinte, *OFDM Frame Equalizer*, responsável pela equalização de cada portadoras OFDM. Após esse bloco, as portadoras que possuem símbolos de informação são separadas dos tons pilotos no bloco *OFDM Serializer*, podendo ser então demodulados e o sinal de informação recuperado.

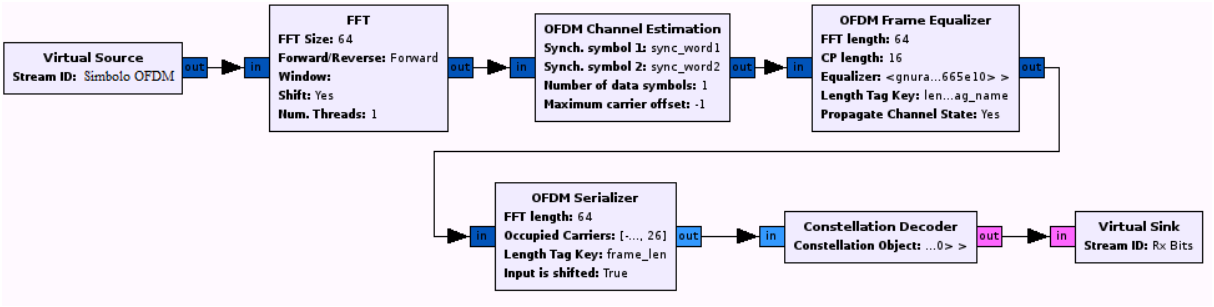

FIG. 3.5: Principais blocos para uma implementação de sistema OFDM a nível GRC, Receptor.

A descrição dos blocos específicos da recepção se encontra abaixo.

⋆ **OFDM Channel Estimation**

Este bloco realiza a estimativa do canal e assim efetuar a correção dos deslocamentos de frequência de *offset* a partir de símbolos conhecidos. Sem essa sincronização a transmissão OFDM fica comprometida, pois o demodulador OFDM não conseguirá identificar o início dos símbolos. Os símbolos conhecidos utilizados para sincronização são denomidados *Synchronisation symbols* sendo retirados do quadro após esse bloco. A saída deste bloco constitui-se portanto apenas dos símbolos OFDM.

Seus parâmetros principais são:

a) **Synch. symbol 1 e 2**: Símbolos utilizados para estimativa do canal, ou seja, símbolos OFDM esperados na recepção por isso são pré-definidos.

b) **Number of data symbols**: Número de símbolos OFDM que são precedidos por um símbolo OFDM de sincronização.

c) **Maximum carrier offset**: Valor máximo do desvio em frequência dado em número de subportadoras.

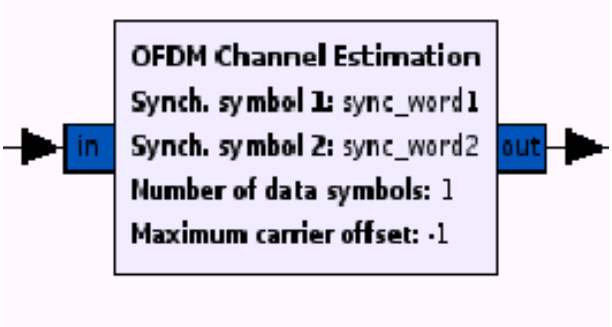

FIG. 3.6: Bloco do OFDM Channel Estimation.

⋆ **OFDM Frame Equalizer**

Este bloco realiza a equalização do sinal OFDM propagado com base na estimativa do canal realizada anteriormente. A saída deste bloco é constituída pelo sinal de entrada equalizado e com os possíveis desvios de frequência corrigidos.

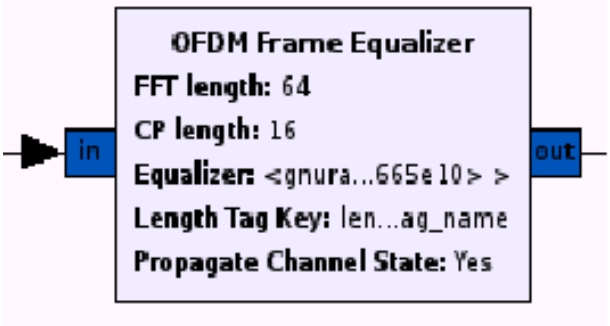

FIG. 3.7: Bloco do OFDM Frame Equalizer.

Seus parâmetros principais são:

a) **Equalizer**: Objeto utilizado para a equalização.

b) **Propagate Channel State**: Parâmetro para definir se as informações acerca do estado do canal será propagada, possibilitando assim que outros blocos também possam efetuar correções adicionais ao sinal recebido.

⋆ **OFDM Serializer**

Este bloco funciona de maneira contrária ao *OFDM Carrier Allocator*, ou seja, retira as portadoras pilotos e propaga apenas os símbolos de cabeçalho/informação. Ao realizar tal função, transforma a stream de vetores de símbolos em uma stream de símbolos de cabeçalho/informação a serem demodulados pelo bloco *Constellation Decoder*.

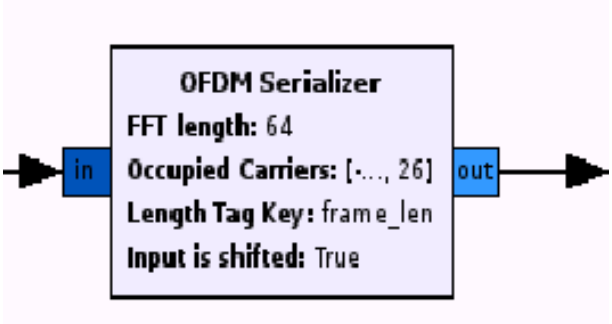

FIG. 3.8: Bloco do OFDM Serializer.

3.2 IMPLEMENTAÇÃO OFDM DIRETAMENTE POR CÓDIGOS PYTHON/C++

Na seção anterior foi explanada a implementação de um sistema OFDM utilizando o GNU Radio Companion, ou seja, a implementação de um fluxograma composto de blocos já presentes no GRC. O objetivo desta seção é detalhar uma implementação alternativa deste mesmo sistema OFDM utilizando diretamente códigos Python e C++ que, por sua vez, empregam as classes disponibilizadas pelo GNU Radio.

3.2.1 O ARQUIVO OFDM.PY

A sequência de ações tomadas em nível de códigos a fim de efetuar-se a multiplexação OFDM é implementada no arquivo python *ofdm.py*. A descrição de uma classe é feita por meio da descrição dos seus atributos e suas funções. Neste arquivo são definidas as duas principais classes:

a) ofdm_mod: Multiplexa símbolos, segundo uma modulação digital escolhida, de informação. Sua saída é uma *stream* de símbolos OFDM.

b) ofdm_demod: Realiza a sincronização, FFT e demultiplexação de símbolos OFDM. Sua é *stream* e sua saída são símbolos da modulação pré-definida.

Ambas as classes utilizam os seguintes parâmetros: *fft_length*, *occupied_tones* e *cp_length*. Tais parâmetros são passados para estas classes por níveis mais altos de programação (blocos hierárquicos).

3.3 TRANSMISSOR OFDM

O papel de realizar a multiplexação OFDM é da classe *ofdm_mod*, como foi citado acima, que é implementada no arquivo *ofdm.py*. Tal classe carrega os dados de informação que são passados pela função *send_pkt* e os transforma em pacotes utilizando a função

make_packet presente no módulo *python ofdm_packet_utils*. Esta função adiciona a estes dados um campo relativo ao código corretor de erro, CRC, e um cabeçalho. Desta maneira o pacote é formado por dados de informação, um campo relativo ao CRC e um campo relativo ao cabeçalho, cuja função no receptor é validar o pacote recebido. Tal pacote constitui o que chamaremos a partir de agora de *payload*, conforme ilustra a figura abaixo:

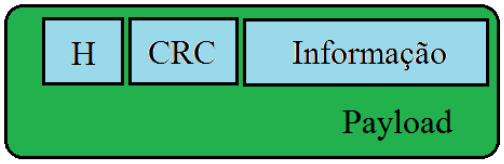

FIG. 3.9: Constituição do Payload.

A função *send_pkt* cria uma fila para estes pacotes de maneira a multiplexá-los, segundo a técnica OFDM, pela sequência de ações ilustradas na figura seguinte.

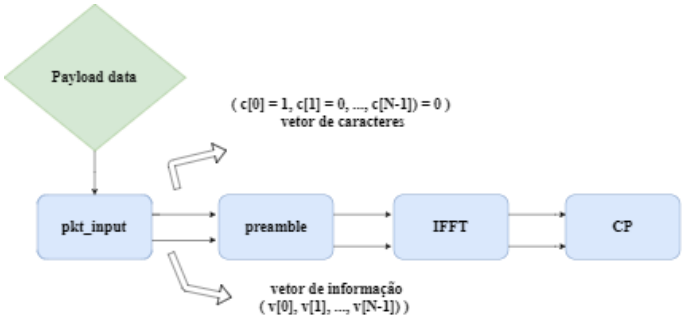

FIG. 3.10: Sequência de processamento da classe *ofdm_mod*.

O primeiro módulo dessa cadeia de processamento é o *pkt_input* que está definido no arquivo C++ ,*gr_ofdm_mapper_bcv.cc*. Sua função é transformar a *stream* de *payload* (bytes) em vetor de símbolos, segundo uma modulação pré-definida. A figura 3.10 indica as duas saídas desse módulo: um vetor de símbolos de informação e um vetor de caracteres. Estes caracteres demarcam o primeiro símbolo do quadro, quando c[i] = 1, para os casos contrários c[i]=0.

O vetor de símbolos de informação então segue para o segundo módulo, bem como o vetor de caracteres. O módulo *preamble*, escrito no arquivo C++ *gr_ofdm_insert_preambles.cc*, adiciona a cada quadro um preâmbulo. Este preâmbulo é uma sequência de símbolos conhecidos, representados pelos valores de +1 e -1, e definidos no arquivo *ofdm.py*. Da

mesma forma que no módulo anterior, as saídas do módulo *preamble* são vetores de símbolos de informação, porém agora adicionados do preâmbulo, e de um vetor de caracteres que demarcam o início de cada quadro, seguindo a mesma lógica anterior.

O módulo *gr_fft_vcc.c* então é responsável por receber a saída de símbolos da cadeia anterior e computar a IFFT. O mesmo módulo também é usado para realizar a FFT. Finalmente, o prefixo cíclico é adicionado aos quadros OFDM no módulo posterior *cp_adder*, definido no arquivo C++ *gr_ofdm_cyclic_prefixer.cc*.

Os símbolos, antes de serem transmitidos, ainda são multiplicados por uma constante que introduz um ganho no sinal a ser transmitido pelo canal.

Cabe ressaltar ainda que o transmissor OFDM implementado neste caso é iniciado com a execução do script *python* do arquivo *benchmark_tx.py* que recebe como entrada todos os parâmetros do sistema de transmissão OFDM e instancia o objeto da classe *ofdm_mod*, responsável pela transmissão OFDM propriamente dita.

3.4 RECEPTOR OFDM

Assim como na transmissão, a demodulação OFDM é feita a partir de códigos implementados em C++ e *python*. Toda a abstração está implementada na classe *ofdm_demod* do arquivo *ofdm.py*. Os pacotes OFDM oriundos do canal servem de entrada para o demodulador. Como no caso do transmissor, o arquivo *benchmark_rx.py* é o responsável por iniciar o processo de recepção e recebe os parâmetros do sistema OFDM para a recepção. Além disso, neste arquivo é definida a função *rx_callback*, análoga a função *send_pkt* do transmissor. Tal função serve para propagar o *payload* pelos módulos da recepção e para levar o sinal demultiplexado para os níveis mais altos da hierarquia. Antes dos símbolos recebidos serem de fato demodulados, estes passam pela sequência de módulos abaixo:

No arquivo *receive_path.py* as seguintes funções são definidas para validar uma recepção:

a) **carrier_sensed**: Retorna o valor booleano *true* ao detectar a presença de portadoras no sinal.

b) **carrier_threshold**: Retorna o limiar em db.

c) **set_carrier_threshold**: Configura o limiar de detecção de portadora em db.

Após as portadoras serem detectadas o arquivo *receive_path.py* instancia a classe *ofdm_demod* e, através da função *callback*, carrega-a com o pacote OFDM recebido

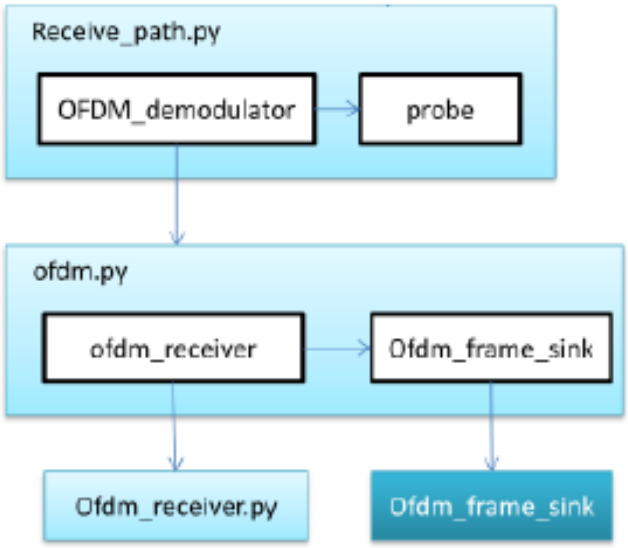

FIG. 3.11: Módulos utilizados em *receive_path.py*. Fonte: Nguyen (2013).

para iniciar o processo de demultiplexação. Primeiramente a função *watcher* da classe *queue_watcher_thread*, que é definida também no arquivo *ofdm.py*, testa a validade dos pacotes OFDM utilizando o código corretor de erro, CRC.

Para a recepção ser implementada duas importantes etapas devem ser realizadas: a sincronização e equalização do sinal e, a demodulação do sinal (decisão e mapeamento inverso de símbolos em bits). Estas funçãos são desempenhadas pelos módulos *ofdm_receiver* e o *ofdm_frame_sink*, presentes no arquivo *python ofdm.py*.

3.4.1 OS MÓDULOS *OFDM_RECEIVER* E *OFDM_FRAME_SINK*

A Figura 3.12 detalha a implementação da classe *ofdm_receiver*. Nesta figura é possível verificar a sequência de operações realizadas por meio dos módulos utilizados e suas conexões:

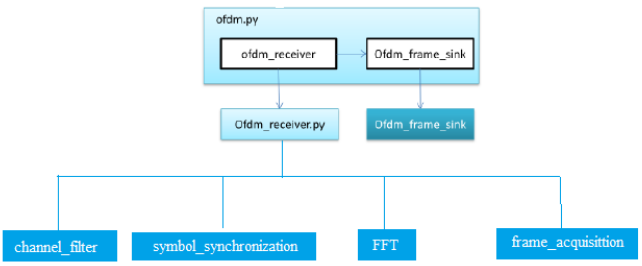

FIG. 3.12: Módulos utilizados em *ofdm_receiver.py*. Fonte: Nguyen (2013).

Primeiramente o sinal recebido é filtrado a fim de se eliminar o ruído fora da faixa,

sendo a saída do filtro constituída apenas pelo sinal dentro da largura de banda referente às portadoras ocupadas. Tal filtro está implementado no arquivo *gr_fft_filter_ccc.c*. Após a filtragem, a sincronização é realizada pelo módulo implementado no arquivo *python ofdm_sync_pn.py* cuja função é determinar o correto deslocamento de frequência e iniciar a recepção dos quadros. Este deslocamento de frequência serve como parâmetro para o gerador de sinal que será empregado na efetiva correção em frequência do sinal recebido no módulo *sigmix*. Apos a heterodinização é esperado que o deslocamento de frequência seja corrigido e o sinal devidamente sincronizado.

Finalmente o sinal, na forma de vetor de tamanho igual ao número de portadoras empregadas, serve de entrada para o módulo que executa a FFT, levando-o sinal para o domínio da frequência, onde cada subportadora contém um símbolo de informação modulada. O último módulo da classe *ofdm_receiver*,*frame_acquisition*, detecta o inicio de cada quadro e equaliza cada subportadora. Este módulo está definido no arquivo C++ *digital_ofdm_frame_acquisition.cc*.

O segundo módulo da recepção, *frame_sink*, é implementado segundo uma máquina de estados definido no arquivo C++ *digital_ofdm_frame_sink.cc*. Sua função é validar os quadros sincronizados do módulo *ofdm_receiver* e enfileirar os quadros validados para a demultilexação. A máquina de estados possui três estados: *"sync search"*, *"have sync"* e *"have header"*. A Figura 3.11 a seguir ilustra a sua implementação:

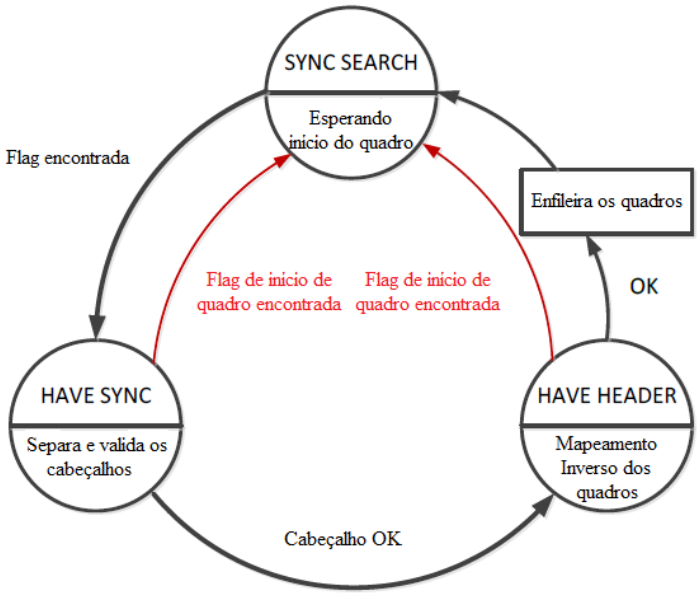

FIG. 3.13: Máquina de estados do módulo *ofdm_frame_sink*. Fonte: Youssef et al. (2012).

4 SIMULAÇÕES REALIZADAS E RESULTADOS OBTIDOS

No capítulo anterior foram discutidas duas implementações de sistemas OFDM no software GNU Radio, na abstração puramente a nível dos arquivos *python* e C++ e na abstração de blocos do GNU Radio Companion. Nesse capítulo serão discutidos os resultados obtidos ao se utilizar essas duas implementações.

4.1 SIMULAÇÕES NA IMPLEMENTAÇÃO GNU RADIO

Como foi explicado na seção 3.2 o software GNU Radio, contém três arquivos *python* associados à implementação a nível de arquivos *python* e C++ de um sistema OFDM. São eles : *benchmarck_ tx.py*[2], *benchmarck_ add_ channel.py* e *benchmarck_ rx.py*. As simulações realizadas envolveram apenas os arquivos de transmissão e recepção, ou seja, o sistema foi simulado em *software* sem efeito de canal adicionado.

O sinal de informação transmitido foi escolhido, por simplicidade, como um arquivo binário de zeros. Este arquivo foi criado via terminal Ubuntu utilizando-se o comando *dd* ilustrado abaixo.

dd if=/dev/zero of=dados_ TX bs =1000003 count=1

Foram realizadas duas simulações com o mesmo arquivo de entrada, variando-se a modulação digital empregada. A primeira simulação empregava a modulação BPSK e na segunda empregou-se a modulação QPSK.

4.1.1 EXECUTANDO OS ARQUIVOS *BENCHMARK_ TX.PY* E *BENCHMARK_ RX.PY*

Através do comando *python benchmark_ tx - - help* é possível configurar vários parâmetros para a transmissão, tais como arquivo de informação a ser transmitido, tamanho de pacotes, modulação digital a ser empregada, etc. O apêndice 7.1 apresenta a tela de configuração *python benchmark_ tx - - help*, para uma melhor visualização de seus parâmetros. Neste momento nos importa saber os parâmetros utilizados para configurar as simulações realizadas:

a) **- -from-file:** Habilita a opção de carregar um arquivo de entrada.

[2]Vale destacar que foram feitas algumas modificações no arquivo original *benchmark_ tx.py*, presente no apêndice 7.2, a fim de se armazenar o sinal de informação que seria transmitido em um arquivo.

b) **- -m:** Configura a modulação digital que será empregada.

c) **- -to-file:** Define o arquivo de destino, ou seja, o arquivo que conterá os símbolos OFDM resultantes do processo de multiplexação.

d) **- - logg:** Habilita que os arquivos intermediários gerados no processo de multiplexação, descritos na seção 3.3, sejam armazenados em arquivos externos.

Assim, considerando que a modulação QPSK foi empregada, o arquivo de origem foi o "dados_TX", o arquivo de saída o "sinal_RX"e a opção - -log foi habilitada a fim de registrar os arquivos intermediários, os comandos executados para as simulações seguiram o seguinte padrão:

python benchmark_tx.py –from-file=dados_TX –m=qpsk –to-file=sinal_RX –logg

Da mesma forma que na transmissão, através do comando *python benchmark_rx.py - - help* tem-se acesso a diversas opções de configuração da recepção.

Para recepção foi executado o seguinte comando, onde as opções têm significado equivalente aos descritos para a transmissão.

python benchmark_rx.py –from-file=sinal_RX –m=qpsk –to-file=sinal_recebido –logg

4.1.2 RESULTADOS OBTIDOS

Com objetivo de aplicar o conhecimento adquirido sobre a implementação OFDM estudada, foram analisados os dados gerados pelas simulações utilizando-se a ferramenta MATLab para gerar figuras.

As Figuras 4.1 e 4.2 ilustram a saída do módulo *gr_ofdm_mapper_bcv.cc*, para os casos da modulação BPSK e QPSK, corroborando a explicação descrita na seção 3.3 onde se afirmou que o sinal de informação é modulado digitalmente pelo módulo *gr_ofdm_mapper_bcv.cc*.

Observa-se nestas figuras a presença de um símbolo nulo, além dos símbolos esperados. Este fato se deve a presença de portadoras nulas inseridas na transmissão.

A saída do módulo *frame_acquisition* definido no arquivo C++ *digital_ofdm_frame_sink.cc* são os símbolos OFDM a serem detectados (após os processos de sincronização e equalização descritos brevemente na subseção 3.4). Um exemplo de saída desse módulo é apresentada abaixo, para o caso das simulações realizadas.

Estas figuras podem causar confusão, indicando a falsa ideia de que o resultado seria incoerente com as condições da transmissão, devido a dispersão das amostras.

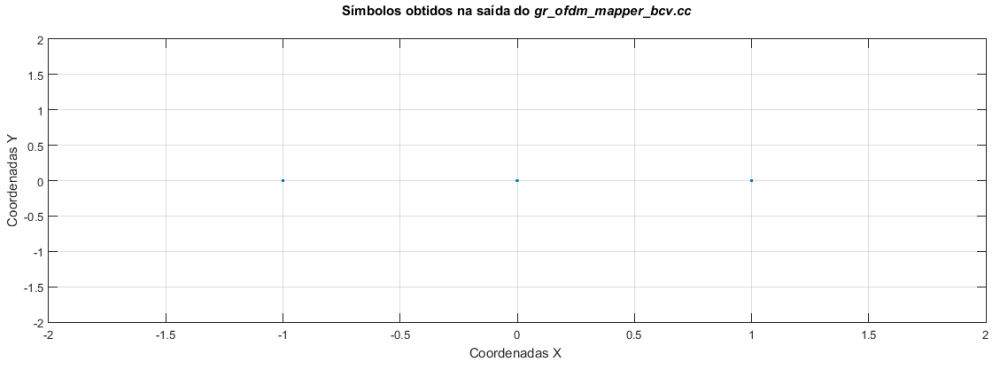

FIG. 4.1: Sinal modulado BPSK

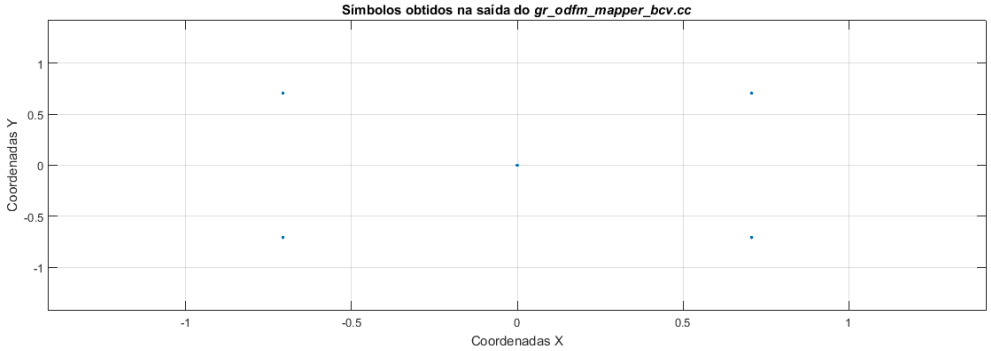

FIG. 4.2: Sinal modulado QPSK

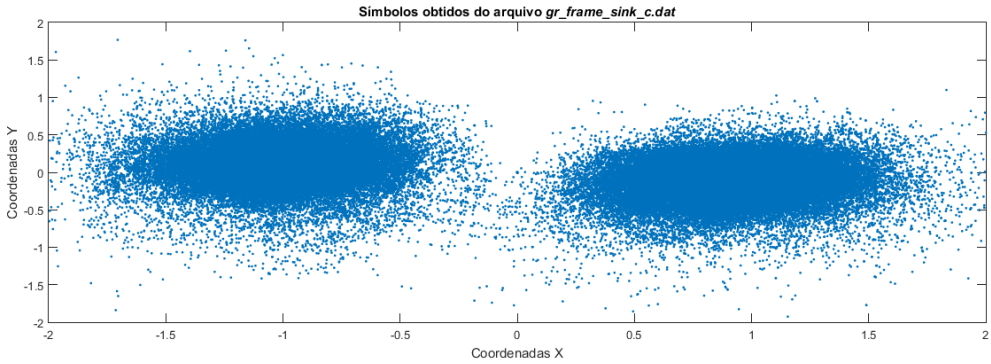

FIG. 4.3: Símbolos BPSK na recepção.

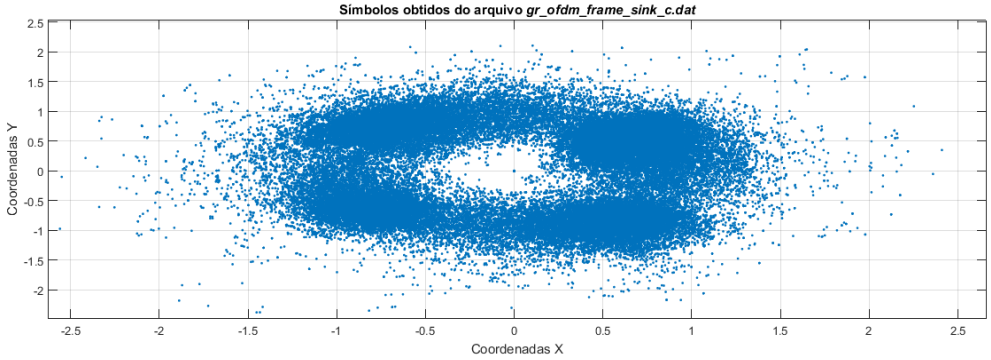

FIG. 4.4: Símbolos QPSK na recepção.

A fim de se ter uma outra apreciação deste resultado optou-se por construir histogramas, uma vez que neste tipo de gráfico é de simples visualização a ocorrência dos casos

mais frequentes.

Estes diagramas são mostrados nas figuras 4.3 e 4.4. Pode-se ver nestas figuras que de fato a grande maioria das amostras se concentra muito próximo de valores correspondentes aos símbolos das modulações empregadas.

A presença de amostras diferentes de tais valores nas figuras acima pode à primeira vista parecer estranha, por se tratar de uma transmissão sem ruido, mas deve-se ter em mente que a geração do sinal OFDM envolve operações tais como o emprego de janelas temporais para suavização de espectro que podem resultar neste tipo de dispersão. Uma investigação mais detalhada poderia ser feita a fim de tentar confirmar esta conjectura mas, diante das limitações de tempo a que esteve sujeita a execução deste trabalho optou-se por não realizar este tipo de aprofundamento, em prol de outros objetivos que se desejava atingir.

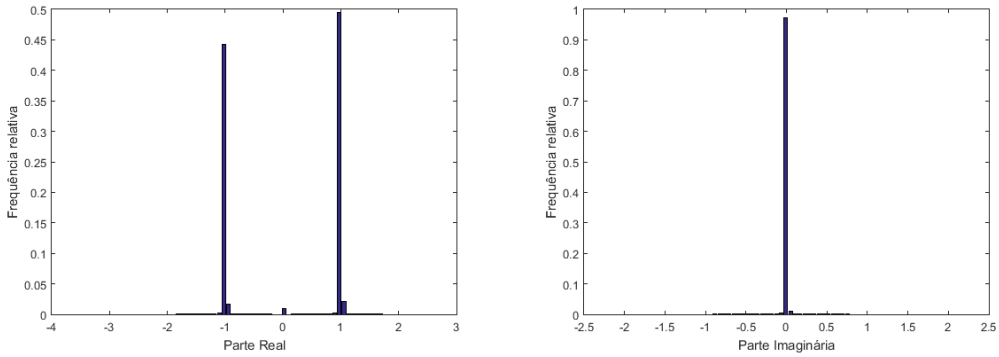

FIG. 4.5: Símbolos BPSK na recepção.

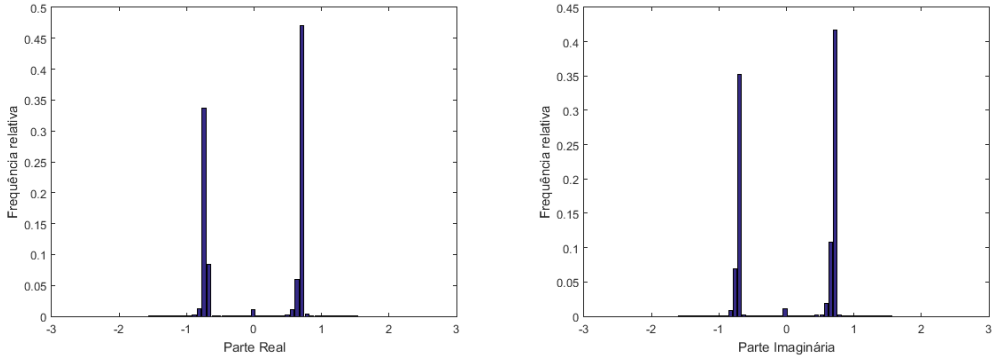

FIG. 4.6: Símbolos QPSK na recepção.

4.2 SIMULAÇÕES NA IMPLEMENTAÇÃO GNU RADIO COMPANION

Foi realizada a transmissão OFDM utilizando-se blocos default do GNU Radio Companion, TX_OFDM e RX_OFDM, a fim de avaliar o potencial desta ferramenta.

4.2.1 TRANSMISSÃO OFDM BPSK E QPSK

A fim de se fazer uma primeira abordagem do funcionamento do sistema na presença de ruído, foi realizada uma adaptação nos arquivos TX_OFMD e RX_OFMD, de modo a unificá-los, inserindo-se entre eles um bloco de modelo de canal com ruído aditivo Gaussiano branco.

Introduziu-se ainda um bloco de comparação dos bits transmitidos e recebidos, bem como o processamento adicional para se obter uma estimativa da frequência (taxa) de ocorrência de discrepâncias nesta comparação, a qual será nomeada aqui de taxa de erro de bits.

Cabe observar de antemão que o indicador assim obtido só refletirá o funcionamento da detecção dos símbolos de informação se as etapas de recepção (de sincronização e detecção de cabeçalho, em particular, como mostrado na Figura 3.13) possibilitarem o correto alinhamento entre o fluxo de bits transmitidos e o recebido. Noutras palavras, se houver desalinhamento entre estes fluxos, e tendo em conta as propriedades probabilísticas da fonte de dados, será obtida um indicador de taxa de erro com valor próximo de 0,5, que servirá apenas como indicador da ocorrência de perda de sincronismo ou recebimento incorreto de cabeçalho.

Uma ilustração deste fato é mostrada na Figura **??**, onde se observa que a taxa de erro salta bruscamente para um valor de aproximadamente 0,5, depois de permanecer próxima de zero por um intervalo relativamente longo. Esta variação brusca indica a ocorrência de uma das falhas acima mencionadas e o consequente desalinhamento dos fluxos de bits comparados. Uma vez ocorrido tal desalinhamento, mesmo que seja por um quadro apenas, o mesmo permanecerá.

Para se ter uma estimativa da taxa de erro de recepção de informação útil seria necessário implementar um processo mais elaborado de comparação dos bits transmitidos com os recebidos, de modo a manter alinhados os fluxos de bits comparados, mesmo diante da perda de sincronismo de quadro ou perda de cabeçalho. Tendo em vista os objetivos e as limitações de tempo do presente trabalho, optou-se por deixar esta tarefa para possíveis continuações do mesmo.

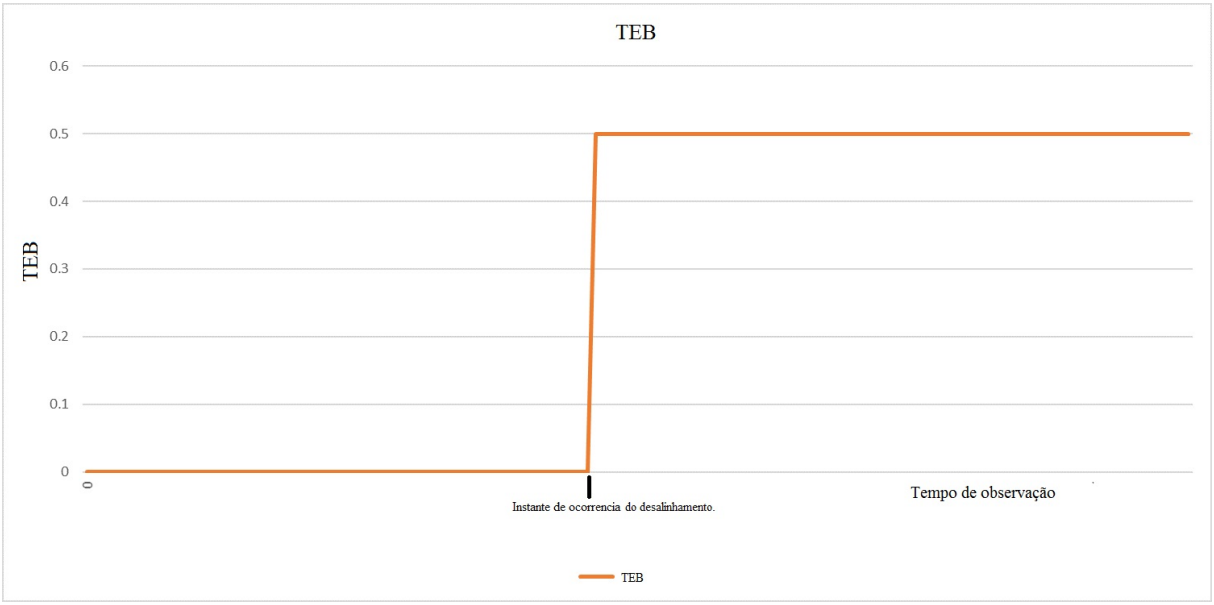

FIG. 4.7: Comportamento da TEB medida.

5 CONCLUSÃO E TRABALHOS FUTUROS

Uma das principais motivações para o desenvolvimento de implementações de sistemas OFDM é sua crescente aplicação em diversos sistemas de comunicação. Exemplos estão em versões mais recentes do padrão IEEE 802.11, em alguns sistemas de vídeos, como a Televisão Digital.

O presente trabalho consistiu num estudo inicial de alternativas disponíveis para implementação de sistemas OFDM em ambiente GNU Radio. Foram realizadas simulações desses sistemas nas duas implementações disponíveis do software. O ganho do presente trabalho é justamente fornecer para estudantes e pesquisadores futuros uma referência para o desenvolvimento desses sistemas, informações que atualmente encontram-se escassas. O fato do GNU Radio ser um ambiente colaborativo possui vantagens e desvantagens. A principal vantagem é de ser gratuito, ou seja, não há necessidade de se pagar, seja para desenvolver rádios, seja na forma de eventuais royalties para se ter acesso a uma tecnologia. Por outro lado, a principal desvantagem é a má documentação e a falta de suporte.

Como possíveis trabalhos futuros pode-se apontar, principalmente, testes de transmissão real utilizando-se placas USRP e modificações nos arquivos Python e C++ do ambiente GNU Radio. Após essas evoluções serem conquistadas, teria-se uma maior maturidade com o a plataforma GNU Radio, possibilitando a construção de novos fluxogramas para sistemas OFDM, visando a otimização dos processos descritos neste trabalho.

6 REFERÊNCIAS BIBLIOGRÁFICAS

GNU Radio Manual and C++ API Reference. Disponível em: <https://gnuradio.org/doc/doxygen/index.html>. Acesso em: 18 ago. de 2016.

WIRELESS COMMUNICATION. OFDM. Disponível em: <http://www.wirelesscommunication.nl/reference/chaptr05/ofdm/ofdmhist.html>. Acesso em: 21 set. de 2017.

NATIONAL INSTRUMENTS. OFDM and Multi-Channel Communication Systems. Disponível em: <http://www.ni.com/white-paper/3740/en/>. Acesso em: 21 set. de 2017.

NGUYEN, D. T. **Implementation of OFDM systems using GNU Radio and USRP**. 2013. 117 f. Dissertação (Master by Research - Engineering) – University of Wollongong, Wollongong, Austrália, 2013. Acesso em: Agosto de 2013.

GNU RADIO PROJECT. Packet Communications. Disponível em: <https://www.gnuradio.org/doc/doxygen/page_packet_comms.html>. Acesso em: 18 ago. de 2016.

GNU RADIO PROJECT. Guided Tutorials. Disponível em: <https://wiki.gnuradio.org/index.php/Guided_Tutorials>. Acesso em: 5 jul. de 2017.

MY COMPUTER TUTORS. Multiplexing: FDM, WDM and TDM. Disponível em: <http://comptutorkg2pg.blogspot.com.br/2012/05/multiplexing-fdm-wdm-and-tdm.html>. Acesso em: 21 set. de 2017.

YOUSSEF, A. Y. F.; HASSAN, K. M. H.-A.; MOSTAFA, M. G. ; MOSTAFA, M. G. **Implementation of a wireless OFDM system using USRP 2 and USRP N210 kits**. 2012. 90 f. Trabalho de Conclusão de Curso (Bachelor of Science in Electronics and Communications Engineering) – Faculty of Engineering, Cairo University, Giza, Egito, 2012.

7 APÊNDICES

APÊNDICE 1: TELA DE CONFIGURAÇÃO DOS ARQUIVO *BENCHMARK*

Encontra-se a seguir uma imagem da captura de tela do arquivo *benchmark_ tx.py*, para melhor ilustração dos parâmetros configuráveis na transmissão.

```
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.11.0.git
-94-g5964adcd
Usage: benchmark_tx.py [options]

Options:
  -h, --help            show this help message and exit
  -s SIZE, --size=SIZE  set packet size [default=400]
  -M MEGABYTES, --megabytes=MEGABYTES
                        set megabytes to transmit [default=1.0]
  --discontinuous       enable discontinuous mode
  --from-file=FROM_FILE
                        use intput file for packet contents
  --to-file=TO_FILE     Output file for modulated samples
  --tx-amplitude=AMPL   set transmitter digital amplitude: 0 <= AMPL < 1.0
                        [default=0.1]
  -W BANDWIDTH, --bandwidth=BANDWIDTH
                        set symbol bandwidth [default=500000.0]
  -m MODULATION, --modulation=MODULATION
                        set modulation type (bpsk, qpsk, 8psk, qam{16,64})
                        [default=bpsk]
  -f FREQ, --freq=FREQ  set Tx and/or Rx frequency to FREQ [default=none]
  -a ARGS, --args=ARGS  UHD device address args [default=]
  --spec=SPEC           Subdevice of UHD device where appropriate
  -A ANTENNA, --antenna=ANTENNA
                        select Rx Antenna where appropriate
  --tx-freq=FREQ        set transmit frequency to FREQ [default=none]
  --lo-offset=LO_OFFSET
                        set local oscillator offset in Hz (default is 0)
  --tx-gain=TX_GAIN     set transmit gain in dB (default is midpoint)
  -C CLOCK_SOURCE, --clock-source=CLOCK_SOURCE
                        select clock source (e.g. 'external') [default=none]
  -v, --verbose

  Expert:
    --log               Log all parts of flow graph to file (CAUTION: lots of
                        data)
    --fft-length=FFT_LENGTH
                        set the number of FFT bins [default=512]
    --occupied-tones=OCCUPIED_TONES
                        set the number of occupied FFT bins [default=200]
    --cp-length=CP_LENGTH
                        set the number of bits in the cyclic prefix
                        [default=128]
```

FIG. 7.1: Captura de tela do comando *python benchmark_ tx.py - - help.*

APÊNDICE 2: CÓDIGO *BENCHMARK_TX.PY UTILIZADO*

Encontra-se a seguir o código *benchmark_tx.py* modificado que foi utilizado no presente trabalho.

```
from gnuradio import gr
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import time, struct, sys

from gnuradio import digital
from gnuradio import blocks

# from current dir
from transmit_path import transmit_path
from uhd_interface import uhd_transmitter

class my_top_block(gr.top_block):
def __init__(self, options):
gr.top_block.__init__(self)

if(options.tx_freq is not None):
self.sink = uhd_transmitter(options.args,
options.bandwidth, options.tx_freq,
options.lo_offset, options.tx_gain,
options.spec, options.antenna,
options.clock_source, options.verbose)
elif(options.to_file is not None):
self.sink = blocks.file_sink(gr.sizeof_gr_complex, options.to_file)
else:
self.sink = blocks.null_sink(gr.sizeof_gr_complex)
```

```
# do this after for any adjustments to the options that may
# occur in the sinks (specifically the UHD sink)
self.txpath = transmit_path(options)


self.connect(self.txpath, self.sink)

# ////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////

def main():

def send_pkt(payload='', eof=False):
return tb.txpath.send_pkt(payload, eof)

parser = OptionParser(option_class=eng_option, conflict_handler="resolve")
expert_grp = parser.add_option_group("Expert")
parser.add_option("-s", "--size", type="eng_float", default=400,
help="set packet size [default=%default]")
parser.add_option("-M", "--megabytes", type="eng_float", default=1.0,
help="set megabytes to transmit [default=%default]")
parser.add_option("","--discontinuous", action="store_true", default=False,
help="enable discontinuous mode")
parser.add_option("","--from-file", default=None,
help="use intput file for packet contents")
parser.add_option("","--to-file", default=None,
help="Output file for modulated samples")

transmit_path.add_options(parser, expert_grp)
digital.ofdm_mod.add_options(parser, expert_grp)
uhd_transmitter.add_options(parser)
```

```
(options, args) = parser.parse_args ()

# build the graph
tb = my_top_block(options)

r = gr.enable_realtime_scheduling()
if r != gr.RT_OK:
print "Warning: failed to enable realtime scheduling"

tb.start()                          # start flow graph

# generate and send packets
nbytes = int(1e6 * options.megabytes)
n = 0
pktno = 0
pkt_size = int(options.size)
if not options.from_file is None:
print "cria arquivo"
source_file = file(options.from_file)
```