UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UTILIZAÇÃO DE RÁDIO DEFINIDO POR SOFTWARE PARA ANÁLISES DE SINAIS APLICADOS AO ENSINO EM ENGENHARIA

PRISCILA CRISFIR ALMEIDA DINIZ

MARÇO
2013

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UTILIZAÇÃO DE RÁDIO DEFINIDO POR SOFTWARE PARA ANÁLISES DE SINAIS APLICADOS AO ENSINO EM ENGENHARIA

Texto da dissertação apresentada à Universidade Federal de Uberlândia, perante a banca de examinadores abaixo, como parte dos requisitos necessários para obtenção do título de Mestre em Ciências. Aprovada em 12 de Março de 2013.

Área de concentração: Processamento da Informação

Banca examinadora:

Prof. Antônio Cláudio Paschoarelli Veiga, Dr. (Orientador)

Prof. Paulo Sérgio Caparelli, Dr. (UFU)

Prof. Luciano Xavier Medeiros, Dr. (UFTM)

UTILIZAÇÃO DE RÁDIO DEFINIDO POR SOFTWARE PARA ANÁLISES DE SINAIS APLICADOS AO ENSINO EM ENGENHARIA

PRISCILA CRISFIR ALMEIDA DINIZ

_____	_____
Prof. Antônio Cláudio Paschoarelli Veiga, Dr.	Edgard A. Lamounier Júnior, PhD.
Orientador	Coordenador do curso de Pós-Graduação

Agradecimentos

Agradeço primeiramente a Deus por sempre guiar meus passos, iluminar meu caminho e sempre ser fonte de forças em momentos difíceis.

Aos meus pais Firmo e Izabel Cristina pela estrutura familiar e apoio emocional. À minha avó Ilda pelas orações e aos meus irmãos Débora, Fabrício e Rafael por sempre acreditarem em meu potencial e me darem força. Agradeço aos meus sobrinhos Melissa, João Pedro, Mateus e Giovanna por sempre me trazerem alegrias e agradeço a todos os meus familiares por me incluírem em suas orações.

Agradeço em especial ao Professor Paschoarelli pela orientação, exemplo profissional e pelas palavras de conforto durante todo o trabalho de mestrado. Agradeço a Professora Edna pela amizade e conselhos em diversos momentos e não poderia esquecer dos parceiros de laboratório Gustavo Nozella, Thiago Henrique e Paulo Victor que estiveram presentes em momentos frustrantes e momentos vitoriosos com meus testes.

Agradeço a Cecília Helena pela amizade e companheirismo dedicado, por sempre estar disposta a ouvir sobre minha dissertação e pelas correções de texto, aos amigos Fábio Henrique, Daniel, Saulo, Joyce, Pedro, e Edylara pelas horas de conversas, troca de ideias e ainda pelos momentos festivos e esportivos que passamos nesse período.

Por fim agradeço a todos que fizeram parte dessa jornada e quero dizer que sonhos são pra serem realizados e que este trabalho é o resultado de um sonho, a todos vocês o meu muito obrigado!

RESUMO

A tecnologia de Rádio Definido por *Software* está induzindo uma revolução nos conceitos de dispositivos para processamento de sinais. O Rádio Definido por *Software* fornece uma arquitetura de rádio flexível que permite a mudança da função do rádio, possivelmente, em tempo real, e, com um processo que garante uma qualidade de serviço desejado. Essa flexibilidade na arquitetura de *hardware* combinada com a flexibilidade em arquitetura de *software* possibilita uma integração com redes e com interfaces totalmente diferentes. A partir deste conceito, este trabalho apresenta aplicações utilizando o *Hardware Universal Software Radio Peripheral* e o *Software GNU Radio* como uma proposta para equipar laboratórios nas áreas de sistemas de telecomunicações, sistemas biomédicos bem como sistemas elétricos e eletrônicos. E ainda, utilizando o *GNU Radio*, este trabalho apresenta a criação de novos blocos de processamento com o objetivo de aplicar a Transformada Wavelet Daubechies para analisar sinais cardíacos e também para processar sinais de voz a fim de realizar a compressão de arquivos.

Palavras – chave: Rádio Definido por Software, USRP, GNU Radio, Análise espectral, Transformada Wavelet Daubechies.

ABSTRACT

The Software Defined Radio is revolutionizing the concept of signal processing devices. The Software Defined Radio provides a radio architecture that allows flexible switching of the radio function, possibly in real time, with a process that ensures a desired quality of service. This flexible hardware architecture combined with flexibility in software architecture allows integration with networks and with totally different interfaces. From this concept, this work presents applications using the Universal Software Radio Peripheral and the GNU Radio software as a proposal to equip telecommunications, biomedical, electrical and electronic laboratories. Also, this work presents the creation of new processing blocks using the GNU Radio, with the goal of applying the Daubechies Wavelet Transform to analyze cardiac signals and also for voice signals processing in order to perform file compression.

Keyword:. *Software Defined Radio, USRP, GNU Radio, Spectrum analyzer, Daubechies Wavelet Transform.*

ÍNDICE

LISTA DE FIGURAS

CAPÍTULO 1

1 INTRODUÇÃO

1.1 Introdução

A tecnologia de *software radio defined* (SDR) está induzindo uma revolução nos conceitos de dispositivos para processamento de sinais. Com o avanço nas tecnologias de comunicação móvel, os projetos de transceptores de rádio foram inovados. Os transceptores tradicionais são muito dependentes do seu *hardware*, executando funções especificas que seguem um determinado protocolo.

Nas últimas décadas foram feitos progressos na área de *hardware* de rádio frequência (RF) e com a difusão dos processadores digitais de sinais surgiram os transceptores digitais [1]. Um transceptor digital é dividido em duas partes: a primeira chamada de *front-end* (FE) que limita o sinal de entrada em uma faixa estreita e o translada para uma frequência menor, seguido de um conversor analógico digital (A/D), e a segunda parte denominada de *back-end* (BE) que é responsável por continuar o processamento do sinal.

O Rádio Definido por *Software* fornece uma arquitetura de rádio flexível que permite a mudança da função do rádio, possivelmente, em tempo real, e, com um processo que garante uma qualidade de serviço desejada. Essa flexibilidade na arquitetura de *hardware* combinada com a flexibilidade da arquitetura de *software*, por meio de técnicas de programação orientada a objeto, permite ao *software* de rádio a capacidade de integração com redes e com interfaces totalmente diferentes.

A Figura 1.1 ilustra as partes que compõe o rádio definido por software usado neste trabalho.

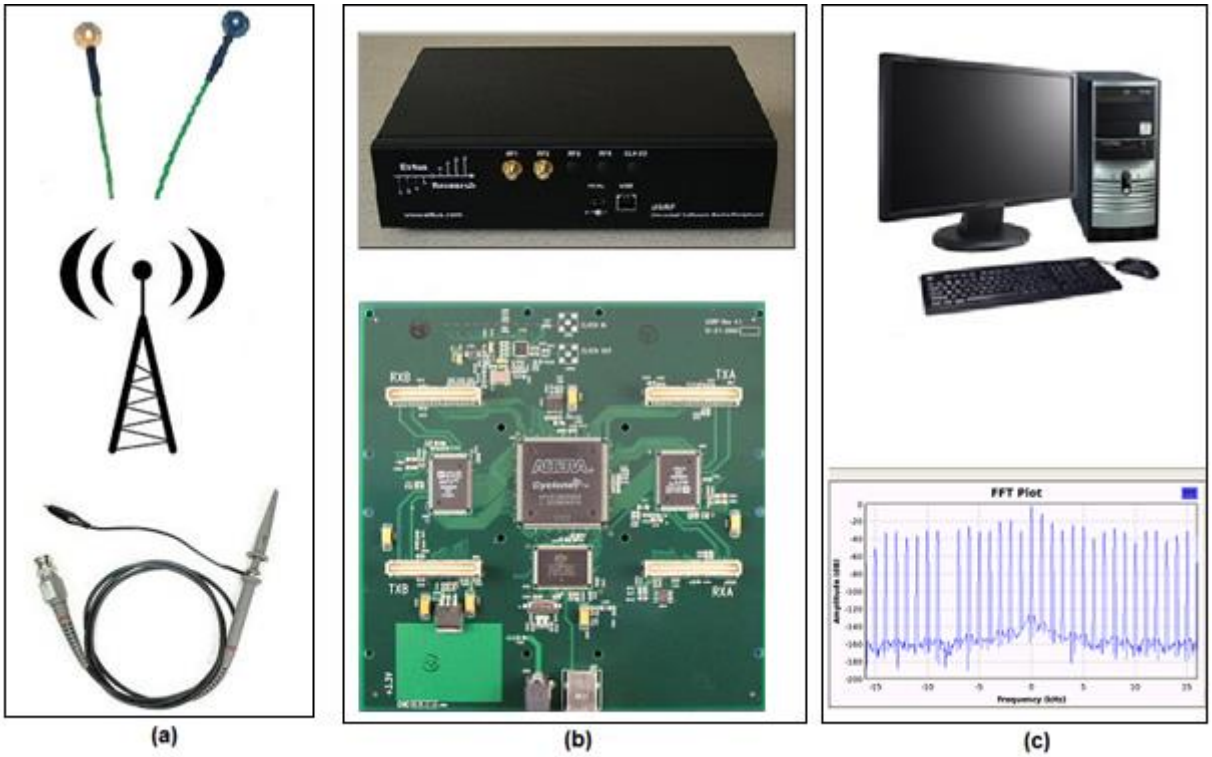

Figura 1.1 – Conjunto que compõe o SDR usado neste trabalho.

A Figura 1.1 (a) mostra que uma antena ou uma ponteira podem ser utilizados como dispositivos de entrada/saída de sinais. De acordo com a aplicação, outros dispositivos de entrada/saída podem ser utilizados. Como por exemplo, eletrodos e microfones. A Figura 1.1 (b) ilustra o *hardware* que é o módulo *Universal Software Radio Peripheral* (USRP) e a Figura 1.1 (c) representa o *software*, com um computador de uso geral, preferencialmente, com o sistema operacional Linux e com o *software* GNU Radio instalado. Este conjunto foi utilizado nos testes realizados neste trabalho.

1.2 Motivação Deste Trabalho

Existe um grande déficit de equipamentos para análise de sinais nas instituições de ensino. Este problema acontece principalmente devido ao alto custo de cada equipamento. Pode-se citar como exemplo de equipamentos: analisadores de espectro, processadores de sinais biológicos, osciloscópios, geradores de ondas, dentre outros.

A partir do conceito de Rádio Definido por *Software*, este trabalho foi motivado pelo baixo custo do conjunto *hardware + software* e por comporem uma ferramenta altamente flexível, capaz de processar qualquer tipo de sinal em uma ampla faixa de frequência permitindo que uma nova proposta para equipar laboratórios seja apresentada neste trabalho.

1.3 Objetivos Deste Trabalho

Para receber, transmitir, modular e demodular ou fazer uma análise espectral de um sinal são necessários diversos equipamentos, no qual, cada um possui uma função especifica. A partir do conceito de SDR e o uso deste conjunto em inúmeras aplicações, diversos equipamentos com *hardware* dedicado podem ser substituídos por um único conjunto SDR.

O objetivo deste trabalho é apresentar uma proposta para equipar laboratórios nas áreas de sistemas de telecomunicações, sistemas biomédicos bem como sistemas elétricos e eletrônicos. Utilizando o conceito da tecnologia de Rádio Definido por *Software* para mostrar aplicações práticas para auxiliar o aprendizado no ensino superior, além de abordar as facilidades de criar novos blocos funcionais para aplicações em processamentos de sinais de diversas naturezas.

1.4 Estrutura Deste Trabalho

O Capítulo 1 apresenta a motivação, o objetivo e a estrutura deste trabalho.

O Capítulo 2 mostra alguns conceitos fundamentais de rádio definido por software, o *hardware* USRP e o *software* GNU Radio.

O Capítulo 3 apresenta estudos tanto da parte do *software* quanto da parte do *hardware*, que compõem o SDR e ainda mostra algumas aplicações que podem ser utilizadas como práticas de laboratório no ensino superior.

O Capítulo 4 mostra a estrutura da árvore de construção do gnuradio e como criar novos blocos no GNU Radio e GRC.

O Capítulo 5 aborda conceitos teóricos, testes e aplicações com a transformada wavelet Daubechies a partir do *software* GNU Radio.

O Capítulo 6 mostra os resultados obtidos nas aplicações e testes realizados neste trabalho.

O Capítulo 7 apresenta as conclusões e contribuições deste trabalho e os trabalhos futuros que poderão ser desenvolvidos a partir desta dissertação.

CAPÍTULO 2

2 FUNDAMENTOS DE SDR

2.1 Introdução

O conceito de Rádio Definido por Software originou-se da necessidade do Departamento de Defesa Americano de integrar diversas interfaces aéreas existentes nos sistemas de comando e controle, assim, tornou-se necessário desenvolver rádios que interagissem simultaneamente com duas ou mais interfaces aéreas e em várias faixas de frequência, executando a função de um *gateway*, e que pudesse ser modificado por uma simples troca de *software*, sem a necessidade de maiores ajustes em *hardware*, aumentando a vida útil dos equipamentos e resultando em uma maior relação custo/benefício [2].

O Termo *Software Defined Radio* (SDR) foi usado por Joe Mitola em 1991 para referir a classe reprogramável ou reconfigurável de rádios. Em outras palavras, uma mesma peça de *hardware* atuando com diferentes funções em diferentes momentos. O *SDR Forum* define a *Ultimate Software Radio* (USR) como o rádio que aceita o tráfego totalmente programável, além de deter o controle da informação e proporcionar suporte a ampla gama de frequência, através de uma interface para aplicações de *software* [3].

A exata definição de SDR é controvérsia, e não existe um consenso sobre o nível de reconfigurabilidade para classificar a qualidade do rádio para um SDR. Um rádio que possui um microprocessador ou um processador digital de sinal (DSP) não necessariamente é qualificado como SDR. Entretanto, um rádio que define, por *software*, modulações, erros de correção e

processos de criptografia por meio de um mesmo *hardware* controlador de rádio frequência (RF), podendo ser reprogramável claramente por *software*, é um SDR. Assim, a definição mais adequada utilizada é que "*um software radio é um rádio substancialmente definido por software, cujo comportamento da camada física pode ser significativamente alterado por meio de um software*" [3]. Portanto, o termo *software radio* refere-se genericamente ao rádio flexível por meio de *software* enquanto é usado uma plataforma estática de *hardware*.

Nas últimas décadas foram feitos progressos na área de *hardware* de RF e com a difusão dos DSP's surgiram os transceptores digitais [1]. Um transceptor digital é dividido em duas partes: uma denominada de *front-end* (FE) que limita o sinal a uma faixa estreita e o translada para uma frequência menor, seguido de um conversor analógico digital (A/D), e a outra denominada de *back-end* (BE) que é responsável por continuar o processamento do sinal, como, por exemplo, a modulação e a demodulação.

Os transceptores tradicionais são muito dependentes do seu *hardware* executando funções especificas que seguem um determinado protocolo. O *hardware* de um SDR precisa ser flexível, assim, a BE do transceptor pode ser feita por *software*.

Um modelo de Rádio Definido por *Software* é mostrado na Figura 2.1. Esse é o único modelo de SDR essencialmente baseado em *hardware*.

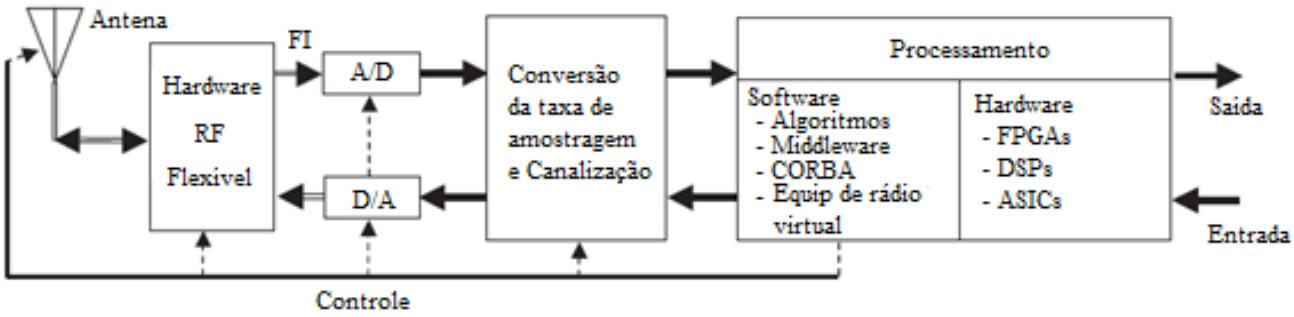

Figura 2.1 – Diagrama de blocos de um modelo de SDR [3].

Na Figura 2.1 as setas para direita indicam o fluxo do receptor e as setas para a esquerda mostra o fluxo do transmissor. O módulo *front-end* RF é composto pela antena inteligente, o *hardware* flexível e os conversores Analógico-Digital e Digital-Analógico. Observa-se que a digitalização do sinal foi deslocada da banda base para o estágio em RF de frequência intermediária (FI). A *back-end* é composta pelo bloco de filtragem digital e conversão da taxa de amostragem e pelo bloco de processamento. Por ser um modelo essencialmente baseado em *hardware* ele não é tão flexível.

O diagrama de blocos dos modelos utilizados atualmente é mostrado na Figura 2.2.

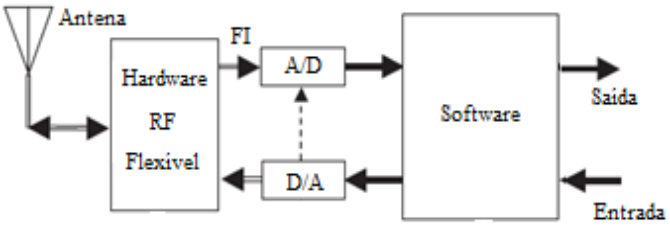

Figura 2.2 – Diagrama de blocos de SDR.

Na Figura 2.2 a FE é composta pela antena, *o hardware* flexível e os conversores Analógico-Digital e Digital-Analógico. Já a BE é composta apenas pelo bloco do *software*.

O rádio definido por *software* fornece uma arquitetura de rádio flexível que permite a mudança da função do rádio e, com um processo que garante uma qualidade de serviço desejada. Essa flexibilidade na arquitetura de *hardware* combinada com a flexibilidade em arquitetura de *software*, por meio da implementação de técnicas de programação orientada a objeto, permite ao *software* de rádio a capacidade de integração com redes e com interfaces totalmente diferentes.

Há uma demanda no setor de telecomunicações para substituição de rádios tradicionais por uma solução consolidada e programável [4]. Dentre as diversas soluções adotadas no mercado a escolhida para este trabalho é o módulo de *hardware Universal Software Radio Peripheral* e o

software GNU Radio. Esse conjunto está sendo o mais utilizado em aplicações de SDR, principalmente, pelo *software* possuir código aberto, e o *hardware* ser de custo baixo.

Este capítulo apresenta os conceitos fundamentais de um SDR e as características técnicas do *hardware* e do *software* utilizados neste trabalho. Finalmente, são realizadas considerações finais deste capitulo.

2.2 USRP

O módulo *Universal Software Radio Peripheral* ou simplesmente USRP é um dispositivo de *hardware* desenvolvido pela Ettus Research [5] e em conjunto com um computador de uso geral compõe uma plataforma flexível de SDR. As principais vantagens desse conjunto é o baixo custo e alto desempenho para recepção, transmissão e processamento de RF. Para realização deste trabalho foi utilizado a versão 1 do USRP, também conhecida como USRP1. Esse módulo custa $700 e suas principais características são apresentadas nesta seção. A Figura 2.3 mostra o USRP usado neste trabalho.

Figura 2.3 – Frente do módulo USRP.

O USRP em sua essência implementa as seções de processamento digital em banda base e frequência intermediária (FI) de um sistema de comunicação de rádio. O seu projeto básico consiste

no processamento da forma de onda, como a modulação e a demodulação, pelo computador que está conectado. Porém, todas as operações de alta velocidade como as conversões digitais para cima ou para baixo, decimação e interpolação são realizadas e gerenciadas pela *Field Programming Gate Array* (FPGA) [6].

A arquitetura do USRP é dividida em duas partes, a primeira é a placa-mãe (*motherboard*) responsável pelas funções mais complexas, como, por exemplo, a modulação. A segunda são as placas-filhas (*daughterboards*) que contém o módulo de RF. A Figura 2.4mostra a arquitetura básica do USRP.

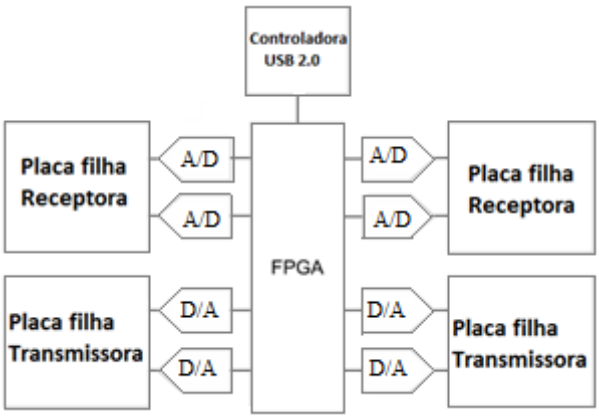

Figura 2.4 – Diagrama de blocos do USRP [7].

As características técnicas e funcionalidades de cada item do diagrama da Figura 2.4 são descritos a seguir.

2.2.1 Placa mãe

A *motherboard* possui quatro conversores A/D e quatro D/A de alta velocidade e um FPGA modelo Altera Cyclone EP1C12. Os conversores analógico/digital de alta velocidade são conectados as *daughterboards*, enquanto a FPGA utiliza uma interface USB 2.0 para conexão com o computador. A Figura 2.5 mostra a placa mãe do USRP [6].

Figura 2.5 – Placa mãe do USRP [6].

Na Figura 2.5 pode ser observado que a placa mãe possui quatro *slots* brancos para a conexão das quatro placas filhas, no qual dois *slots* são para transmissão (TXA e TXB) e dois *slots* são para recepção (RXA e RXB). Os sinais provenientes das placas filhas passam pelos conversores (*chips* pretos entre os *slots* brancos), no centro da placa mãe encontra-se o FPGA Altera Cyclone e no centro inferior da placa encontram-se a entrada da alimentação de 6 V e uma porta USB 2.0 usada para conexão com o computador.

2.2.2 Conversores A/D e D/A

Para a recepção a placa mãe contém quatro conversores A/D de alta velocidade com 12 bits por amostra e uma taxa de amostragem de 64M amostras por segundo. Para evitar o efeito de *aliasing* apenas sinais com largura de banda de até 32 MHz podem ser digitalizados [6].

Os conversores A/D possuem uma faixa de tensão de 2 V_{pp}, e impedância de entrada de 50 ohms diferencial, ou seja, 40 mW ou 16 dBm. Antes do conversor A/D há um amplificador de ganho programável (PGA), usado para amplificar o sinal de modo que se possa utilizar a faixa de tensão de entrada do conversor por completo. O ganho do PGA é programável por *software* e pode atingir até 20 dB. Com ganho zero, a escala total é de 2 V_{pp} diferencial. Se o ganho for programado para seu máximo, ou seja, 20 dB, é necessário um sinal de 0,2 V_{pp} para atingir a escala total [6].

Para a transmissão o USRP utiliza quatro conversores D/A de alta velocidade com 14 bits por amostra e taxa de amostragem de 128 M amostras por segundo. Portanto, a frequência de Nyquist é de 64 MHz. Contudo, deve-se manter esta taxa um pouco mais baixa, a fim de facilitar a filtragem. Uma faixa de frequência recomendada é de DC até 44 MHz. Os conversores D/A conseguem fornecer até 1 V_p para uma carga de 50 ohms, ou 10 mW (10 dBm). Assim, como na recepção há também um PGA logo após o conversor D/A para amplificar o sinal de saída. O PGA pode trabalhar com um ganho de até 20 dB e é programável via *software*. Os sinais de saída são sinais de corrente que variam de 0 até 20 mA. Esses sinais podem ser convertidos em tensão diferencial com um resistor.

2.2.3 FPGA

A FPGA é a "*peça chave*" do conjunto SDR e nele ocorre todo processamento digital do sinal anterior ao envio, ou para o computador com GNU Radio ou para as placas filhas. A FPGA limita a quantidade de dados de tráfego, pois o meio de conexão do USRP1 e o computador acontece pela porta USB.

A interface USB 2.0 possui uma taxa máxima de transferência total de 32 MBps. Os dados transmitidos pela USRP estão no formato complexo, em que cada amostra é composta pela parte real de 16 bits e pela parte imaginária também de 16 bits resultando em 4 bytes por amostra complexa. Assim, a taxa real total a ser recebida ou transmitida pelo USRP via interface USB é de 32 MBps/(4 bytes) = 8 M amostras por segundo. A taxa de amostragem mantém uma relação direta com a largura de banda do espectro do sinal, portanto, a largura de banda máxima do sistema é 8 MHz [6]. O USRP consegue operar em modo *full duplex*, sendo a transmissão independente da recepção, porém a banda máxima (soma das bandas dos canais em operação) não pode ultrapassar os 8 MHz.

A FPGA é reprogramável por meio da porta USB do computador e nela são alteradas informações de decimação e dos conversores para baixo (*Digital down converters – DDC*) para a recepção e a interpolação e dos conversores para cima (*Digital up converters – DUC*) para a transmissão. A Figura 2.6 mostra o fluxograma para a transmissão e para a recepção da USRP [8].

Na Figura 2.6 observa-se nos limites da FPGA, a presença do DDC e a decimação para diminuir a taxa de recepção e o DUC e a interpolação para aumentar a taxa de transmissão e, isto é necessário porque os conversores D/A e A/D do módulo operam com as taxas fixas de amostragem já descritas acima.

A função do DDC é converter o sinal centrado na banda intermediária para a banda base e fazer a decimação para ser transmitido pela interface USB. O sinal de entrada é complexo, e para transladar esse sinal para banda base, o sinal de entrada é multiplicado por um sinal exponencial de frequência constante, usualmente centrada na IF, o sinal resultante é um sinal complexo centrado em zero [6].

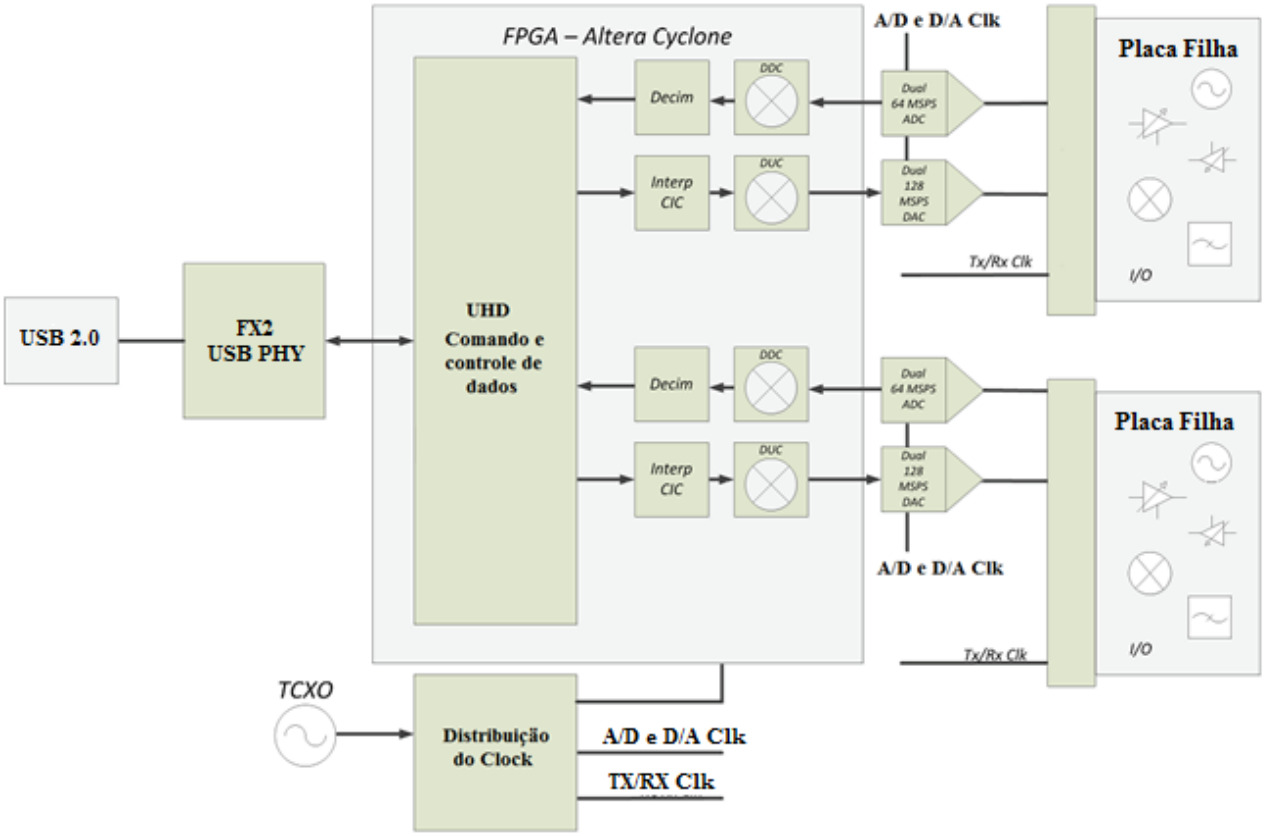

Figura 2.6 – Fluxograma da USRP [8].

Na transmissão é usado o DUC para que possa ser feita a interpolação do sinal, ocorrendo o caminho inverso da recepção. A FPGA recebe o sinal em amostras complexas no mesmo formato do que o da recepção, então o DUC interpola o sinal e o mesmo é transladado para uma frequência intermediária IF e enviado ao conversor analógico digital (A/D) [6]. A Figura 2.7 ilustra as *daughterboards* conectadas a *motherboard*.

A Figura 2.7 mostra os canais de interface RF para recepção e transmissão das quatro placas filhas que estão conectadas à placa-mãe, na qual é visto a FPGA, os conversores AD/DA, a porta USB e a entrada da alimentação DC.

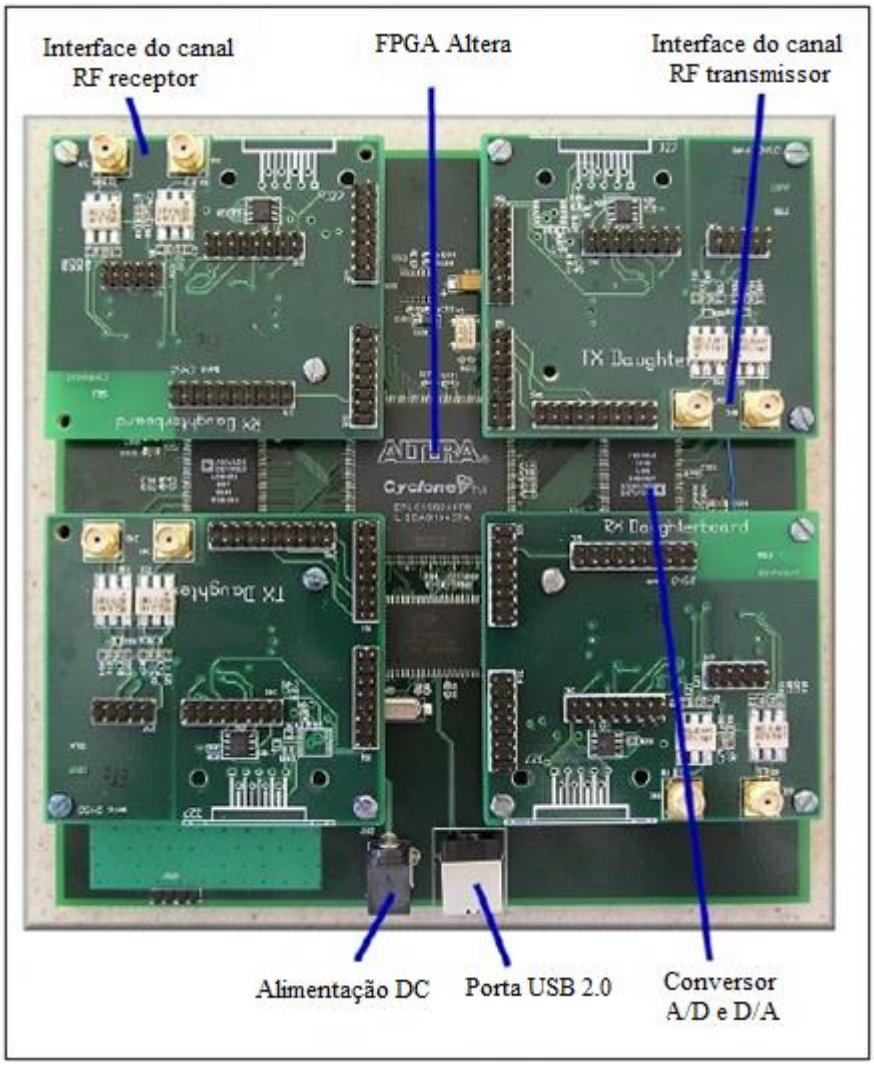

Figura 2.7–*Motherboard da* USRP.

A Ettus Research produz várias *daughterboards* que podem ser classificadas em placas transmissoras, receptoras ou transceptoras, cada uma abrange uma certa faixa de frequência e possui características próprias, as principais *daughterboards* são:

BasicRX, Receptor 1 - 250 MHz.

BasicTX, Transmissor 1 - 250 MHz.

LFRX, Receptor DC - 30 MHz.

LFTX, Transmissor DC - 30 MHz.

TVRX, Receptor 50 - 860 MHz.

DBSRX, Receptor 800 - 2400 MHz.

RFX400, Transceptor 400 - 500 MHz.

RFX900, Transceptor 800 - 1000 MHz.

RFX1200, Transceptor 1150 - 1400 MHz.

RFX1800, Transceptor 1500 - 2100 MHz.

RFX2400, Transceptor 2300 - 2900 MHz.

As placas filhas usadas neste trabalho são descritas a seguir.

2.2.4 BasicRX e BasicTX

As placas BasicTX e BasicRX são placas filhas de baixo custo, que possuem dois conectores SMA com impedância de 50 ohms. A BasicTX é usada para geradores de sinais com frequência que variam de 1 a 250 MHz e a BasicRX é usada para recepção com capacidade de 1 a 250 MHz. É aconselhável o uso de um *front end* externo, pois estas placas não possuem misturadores, filtros, ou amplificadores entre as entradas e saídas dos conversores A/D e D/A e os conectores SMA [8]. A Figura 2.8 mostra as placas BasicTX e Basic RX [8].

Figura 2.8 – Placas filhas (a) BasicTX e (b) BasicRX [8].

2.2.5 LFTX e LFRX

As placas LFTX e LFRX, assim como a BasicTX e BasicRX, são placas filhas de baixo custo, que possui dois conectores SMA com impedância de 50 ohms, além disso a LFTX e a LFRX possuem um filtro passa baixa de 30 MHz para *anti-aliasing* e usam amplificadores diferenciais ao invés de transformadores, possibilitando uma resposta em frequência até o nível DC [8]. A Figura 2.9 mostra as placas filhas LFTX e LFRX [8].

Figura 2.9 – Placas filhas (a) LFTX e em (b) LFRX [8].

A Figura 2.10 mostra um quadro de especificações que resume as características do USRP.

O USRP também oferece suporte ao recurso *multiple-input and multiple-output* (MIMO), ou seja, ele pode usar múltiplas antenas simultaneamente para a recepção e a transmissão de sinais. Se os sinais digitalizados utilizarem a amostragem real, cada *daughterboard* terá dois canais independentes e duas antenas. Se for usada a amostragem complexa, cada placa suporta somente um canal. O software escolhido para os testes deste trabalho foi o *software open source* GNU Radio que tem algumas funções descritas na próxima seção.

Especificações					
Espec.	Tip.	Unid.	Espec.	Tip.	Unid.
Energia			**Performance RF (W/WBX)**		
Entrada DC	6	V	SSB/LO Supressão	35/50	dBc
Consumo de corrente	0.7	A	Ruido de fase (1.8 GHz)		
w/WBX	1.7	A	10 KHz	-80	dBc/Hz
Performance dos conversores e Clocks			100 KHz	-100	dBc/Hz
Taxa de amostragem A/D	64	MS/s	1 MHz	-137	dBc/Hz
Resolução A/D	12	bits	Potência de saida	15	dBm
Largura de banda SFDR A/D	85	dBc	IIP3	0	dBm
Taxa de amostragem D/A	128	MS/s	Figura do ruido recebido	5	dB
Resolução D/A	14	bits	**Físico**		
Largura de banda SFDR D/A	83	dBc	Temperatura de operação	0 a 55°	C
Taxa de amostragem do hospedeiro	16/8	MS/s	Dimensões	18x21x5.5	cm
Precisão de Frequência	25	ppm	Peso	0.7	kg

Figura 2.10 – Especificações do USRP [8].

2.3 GNU Radio

O GNU Radio é uma ferramenta de *software* livre e código aberto que fornece blocos de processamento de sinais utilizados nas aplicações de Rádios Definidos por *Software*. O GNU Radio representa a central de desenvolvimento do SDR e o uso dele junto com uma *front-end* compõe um kit completo de SDR. A *front-end* usada neste trabalho foi a USRP descrita na seção anterior.

O GNU Radio está licenciado sob a *GNU General Public License* (GPL) e isso significa que qualquer pessoa tem o direito de usar, copiar e modificar o GNU Radio sem limites, desde que as extensões sejam disponibilizadas sob a mesma licença [9].

Segundo Eric Blossom, o fundador do GNU Radio, o objetivo desta plataforma é “trazer o código o mais próximo possível da antena” e, portanto “transformar os problemas de *hardware* em problemas de *software*”.

GNU Radio é uma plataforma que provê o processamento de sinais em tempo real. Ele é estruturado em blocos de processamento de sinais para aplicações de SDR utilizando o *hardware* externo como o USRP [5].

Aplicações em GNU Radio são escritas na linguagem de programação Pyhton, enquanto as funções de processamento de sinais de desempenho crítico são escritas em C++, usando extensões de processamento de ponto flutuante, quando disponível. Assim, o desenvolvedor pode implementar sistemas de rádio em tempo real e alta vazão em um ambiente de desenvolvimento rápido de aplicações de uso simples [10].

É recomendável que o GNU Radio seja instalado em um sistema operacional Linux. Para uma adequada instalação é aconselhável seguir os passos encontrados na Wiki do GNU Radio que pode ser acessado em [10]. Após instalado o GNU Radio inúmeros blocos de construção e processamento ficam disponíveis para uso no computador local. A Figura 2.11 mostra algumas classes de blocos de processamento de sinais, implementados em linguagem C++, e disponíveis na biblioteca do GNU Radio 3.3.0 [11].

Para projetar transceptores o GNU Radio faz o uso da teoria dos grafos, na qual os vértices são blocos de processamento de sinal e as bordas representam o fluxo de dados entre eles [10]. Os grafos são construídos e executados na linguagem Python, enquanto os blocos são implementados na linguagem C++. A integração entre as linguagens C++ e Python é feita por meio do *Simplifed Wrapper and Interface Generator* (SWIG). Assim, tem-se um sistema de rádio de alta capacidade e desempenho, utilizando a eficiência e a rapidez da linguagem C++ aliados à simplicidade para desenvolvimento de aplicações com a linguagem Python. A Figura 2.12 foi construída para mostra como deve ser a estrutura de um projeto SDR desenvolvido no GNU Radio.

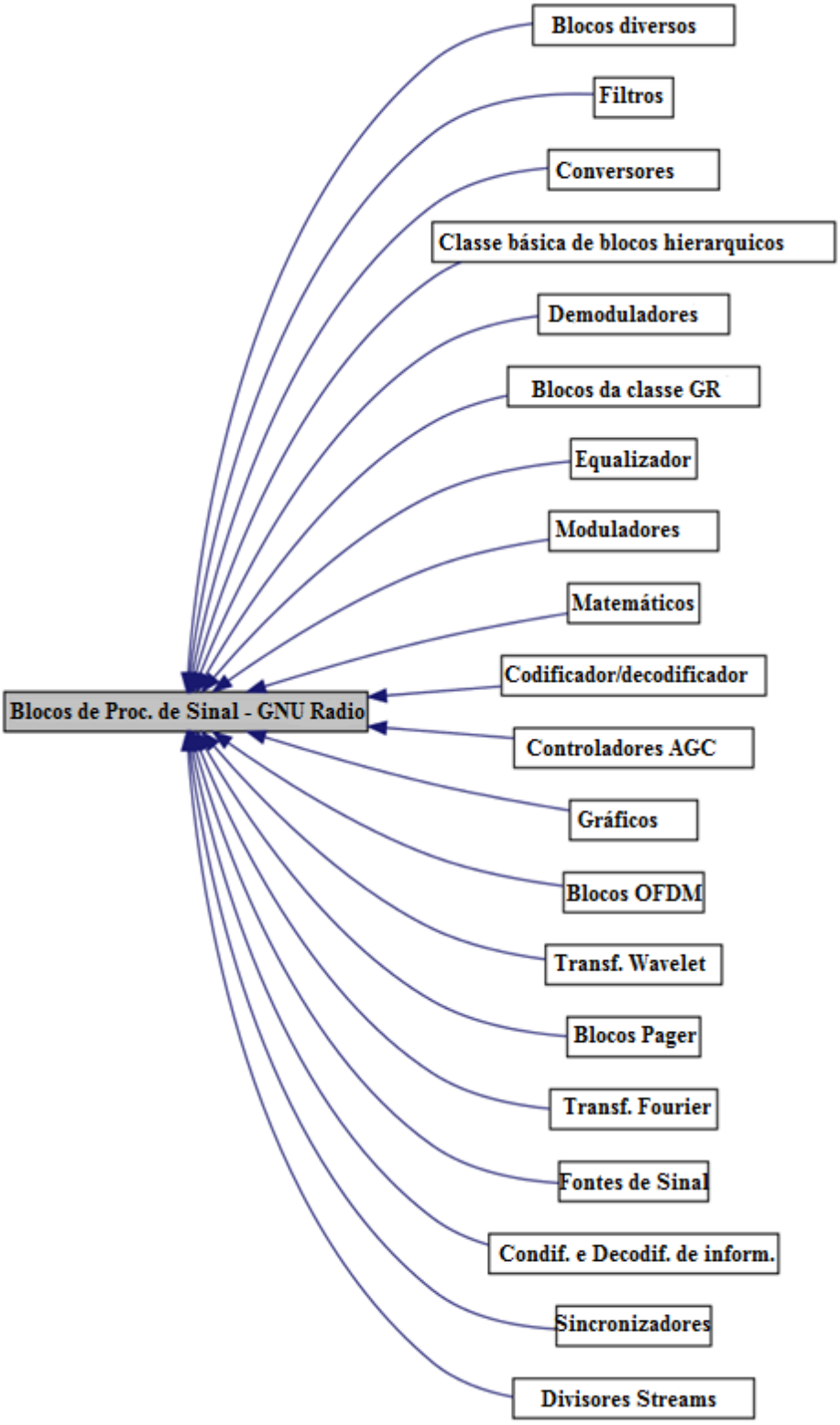

Figura 2.11 – Conjuntos de blocos de processamento de sinal do GNU Radio [11].

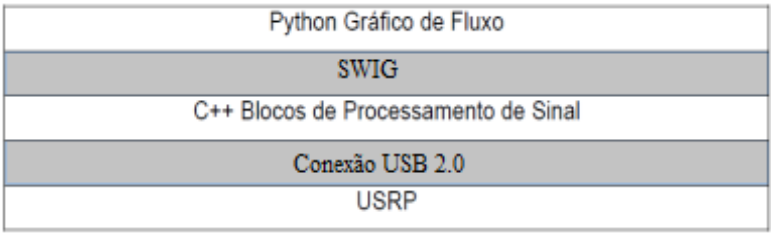

Figura 2.12 – Estrutura de um projeto SDR Utilizando o GNU Radio.

Para aqueles que não estão habilitados a desenvolver programas utilizando linguagem de programação ou querem usar de maneira prática os blocos disponibilizados nesta plataforma, é distribuído uma interface gráfica junto com o GNU Radio, denominada de *GNU Radio Companion* (GRC), que permite criar aplicações de processamento de sinais usando o "arrastar e soltar", que é uma técnica de programação similar à aplicada no Simulink do MATLAB e no LabVeiw. Assim, o projetista pode priorizar e focar o processamento dos sinais e o fluxo da informação a ser processada, tornando secundário o conhecimento da linguagem de programação ou a construção de novos códigos. A Figura 2.13 mostra a tela inicial do GRC versão 3.3.0 que foi instalado no sistema operacional Linux (Ubuntu 12.04).

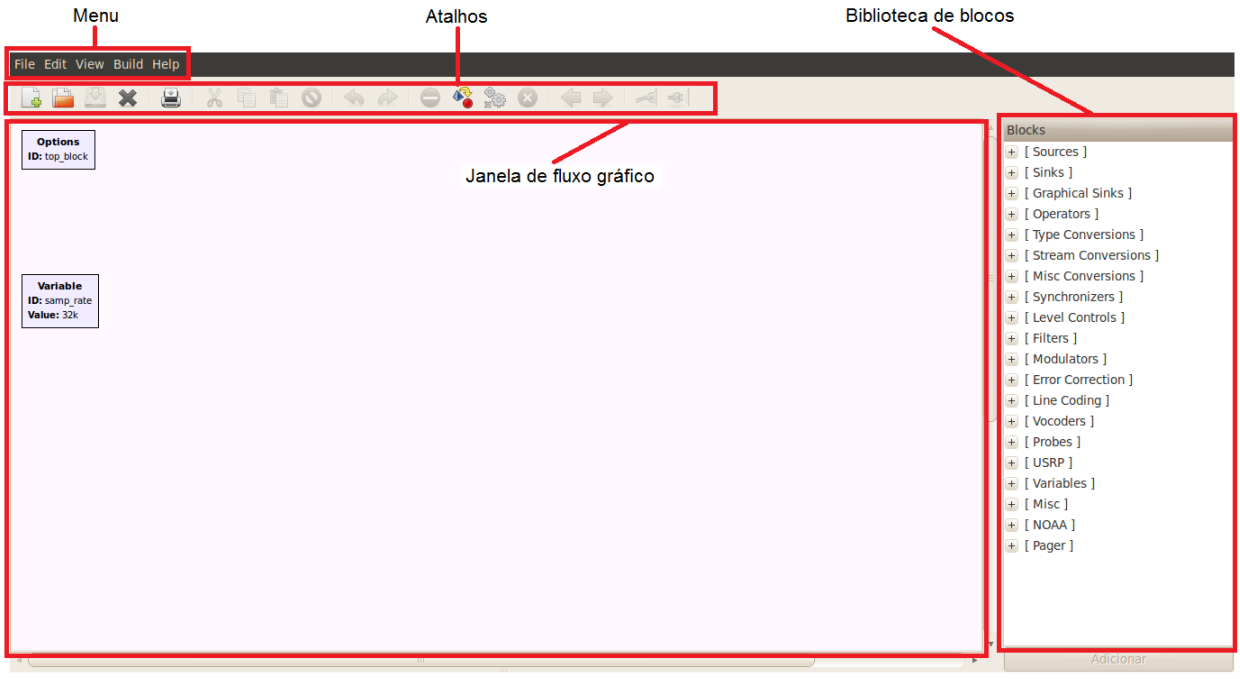

Figura 2.13 – Tela inicial do GRC.

Na Figura 2.13 é possível observar a barra menu, os atalhos disponíveis no GRC, a biblioteca de blocos disponíveis e uma janela de fluxo gráfico, na qual pode se criar um novo projeto SDR de maneira intuitiva movendo os blocos e conectando-os indicando o fluxo da informação de acordo com a necessidade do projetista.

Os sistemas de comunicações são criados graficamente de tal modo que os blocos de processamento de sinal são ligados como em um diagrama de blocos comum. Esses blocos são implementados em C++ e são integrados por meio da linguagem de programação Python [7]. Assim, o desenvolvedor é capaz de implementar sistemas de rádio de alta capacidade, simples de usar, rápidos e com um ambiente de desenvolvimento de aplicações.

O GRC possui algumas limitação, pois nem todos os blocos do GNU Radio estão disponíveis, caso o projetista necessite de um bloco não disponível no GRC é possível apenas incluí-lo criando um gráfico de fluxo em *Python*, ou caso ele necessite de um bloco que não existe no GNU Radio é possível criá-lo utilizando as linguagens de programação C++ e *Python* seguidos de passos que serão descritos no capítulo 4.

Embora não seja essencialmente uma ferramenta de simulação o GNU Radio oferece suporte ao desenvolvimento de algoritmos de processamento de sinal utilizando dados pré-gravados ou gerados por *software*, permitindo o uso sem a necessidade de um *hardware* de RF.

2.4 Considerações Finais Deste Capítulo

Este capítulo apresentou os fundamentos de SDR e ainda mostrou algumas características do *hardware* e do *software* que compõem o conjunto SDR usado neste trabalho.

O próximo capítulo explora os estudos experimentais e mostra algumas aplicações SDR em laboratórios de engenharia.

CAPÍTULO 3

3 ESTUDOS EXPERIMENTAIS E APLICAÇÕES SDR EM LABORATÓRIOS DE ENGENHARIA

3.1 Introdução

As pesquisas e aplicações utilizando SDR vêm aumentando exponencialmente, conquistando espaço em laboratórios e se mostrando uma tendência no mercado de telecomunicações e processamento de sinais.

A fim de buscar familiarização das funcionalidades do GNU Radio e do USRP, este capítulo apresenta estudos experimentais, tanto da parte do *software* quanto da parte do *hardware* que compõem o SDR, e ainda mostra algumas aplicações que podem ser utilizadas como práticas de laboratório no ensino superior.

3.2 Estudo funcional do GNU Radio e GRC

Como introdução das funcionalidades do *software* GNU Radio, um exemplo simples foi desenvolvido, conhecido como "*hello world*" do GNU Radio [12]. Esse projeto gera o tom de discagem do telefone e é reproduzido a partir da placa de som do computador.

O tom de discagem do *hello world* consiste na soma de duas senóides com frequências diferentes, uma com 350 Hz e a outra com 440 Hz. O Código 3.1 criado em linguagem *Python* [12] é mostrado a seguir.

```
1 #!/usr/bin/env python
2
3 from gnuradio import gr
4 from gnuradio import audio
5
6 class my_top_block(gr.top_block):
7     def __init__(self):
8         gr.top_block.__init__(self)
9
10         sample_rate = 48000
11         ampl = 0.4
12
13         src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
14         src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
15         dst = audio.sink (sample_rate, "")
16         self.connect (src0, (dst, 0))
17         self.connect (src1, (dst, 1))
18
19 if __name__ == '__main__':
20     try:
21         my_top_block().run()
22     except [[KeyboardInterrupt]]:
23         pass
```

Código 3.1 – *Hello world* em Python [12].

Ao analisar as linhas deste código é possível visualizar um padrão, a primeira linha é sempre necessária para rodar a rotina por meio do terminal do Linux e ela indica ao *Shell* que o arquivo está em linguagem *Python* que é necessário um interpretador *Python* para executá-lo. As linhas 3 e 4 indicam as bibliotecas que as funções dessa rotina precisam, neste exemplo, usou, o módulo *gr* que é o módulo básico e o módulo *áudio* para ativar a função de áudio. Entre as linhas 6 e 17 é definido a classe *my_top_block*no qual o *gr.top_block* é o encapsulador do fluxo gráfico contendo todas as funções de adição e conexão dos blocos. Duas variáveis de controle são vistas na sequência *sample_rate e ampl* controlando, respectivamente, a taxa de amostragem e a amplitude do sinal gerado. As linhas 13 e 14 mostram as duas fontes de sinais com suas características descritas dentro dos parênteses, sendo a primeira a taxa de amostragem (48000), o tipo de sinal (senoidal), a frequência da onda gerada (350 Hz e 440 Hz) e por fim a amplitude do sinal gerado. A linha 15 indica o bloco de controle de áudio da placa de som, dentro dos parênteses foi preenchida a taxa de amostragem de saída. As linhas 16 e 17 indicam as conexões dos blocos. As linhas 19 a 23 iniciam e executam o fluxo gráfico criado.

Também foi construído neste trabalho o mesmo exemplo, porém usando a interface gráfica GRC, assim os blocos e as conexões são mostradas na Figura 3.1.

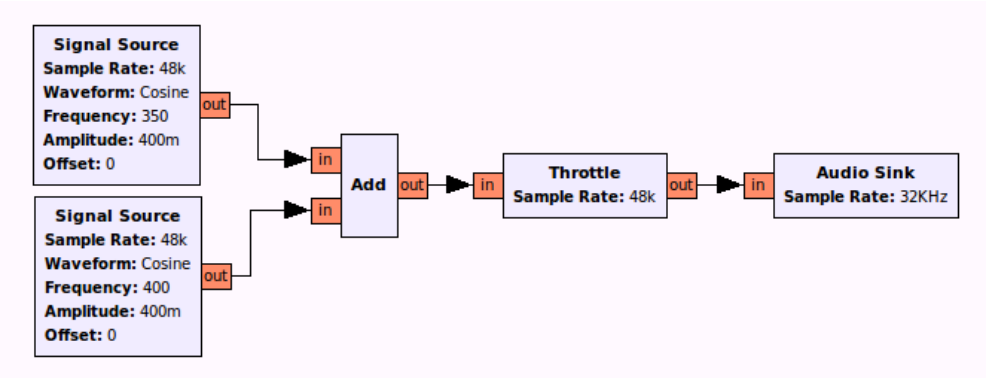

Figura 3.1 – Tom de discagem no GRC.

As características de cada bloco podem ser editadas com um clique duplo em cima do bloco, para então inserir os parâmetros desejados na caixa de propriedades, como mostra a Figura 3.2.

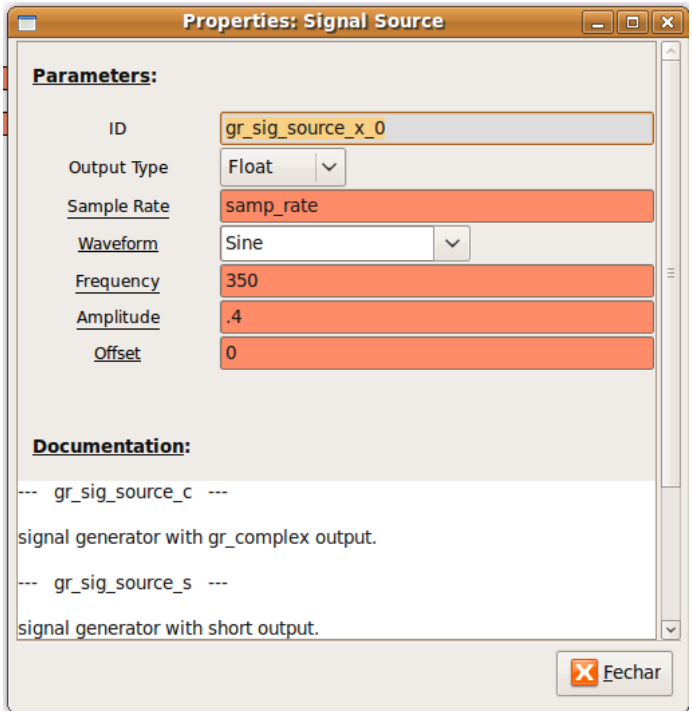

Figura 3.2 – Caixa de propriedades da fonte do sinal no GRC.

Após o estudo e desenvolvimento deste exemplo, a construção de novas aplicações são desenvolvidas utilizando o GRC e o USRP. O próximo item apresenta os fundamentos da modulação em amplitude, os gráficos gerados a partir do fluxo desenvolvido no GRC e a análise espectral do sinal de saída.

3.3 Modulação em Amplitude

A modulação em amplitude (*Amplitude modulation* – AM) tem como principal característica a variação da amplitude da portadora proporcional ao sinal modulante $m(t)$. A Equação (3.1) representa a equação da portadora.

$$S_c = A\cos(\varpi_c t + \theta_c) \tag{3.1}$$

onde: S_c - Sinal da portadora;

A - É a amplitude da portadora;

ϖ_c - É a frequência da portadora; e

θ_c - É a fase da portadora.

Na modulação em amplitude a frequência e a fase da portadora são constantes. Sem afetar os resultados e as conclusões suponha que a fase da onda portadora seja zero. Se a amplitude da portadora for diretamente proporcional ao sinal modulante, o sinal modulado será $m(t)\cos(\varpi_c t)$. A Figura 3.3 mostra o diagrama do modulador AM com banda lateral dupla com portadora suprimida (DSB – SC) [13].

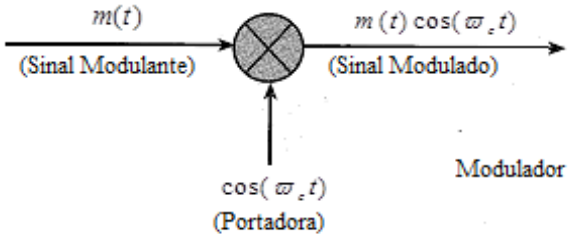

Figura 3.3 – Diagrama do modulador AM DSB – SC.

A modulação AMDSB – SC apenas desloca o espectro da mensagem $m(t)$ para a frequência da portadora ϖ_c. Logo fazendo a transformada de Fourier tem-se o resultado mostrado nas Equações (3.2) e (3.3)

$$m(t) \Leftrightarrow M(\varpi) \tag{3.2}$$

$$m(t)\cos(\varpi_c t) \Leftrightarrow \frac{1}{2}\left[M(\varpi+\varpi_c)+M(\varpi-\varpi_c)\right] \tag{3.3}$$

Se o sinal modulante também conhecido como mensagem possuir uma largura de banda de B Hz o sinal modulado possuirá uma largura de banda de $2B$ Hz em torno da frequência da portadora [13]. A Figura 3.4 mostra a mensagem $m(t)$ e seu espectro obtido a partir da transformada de Fourier.

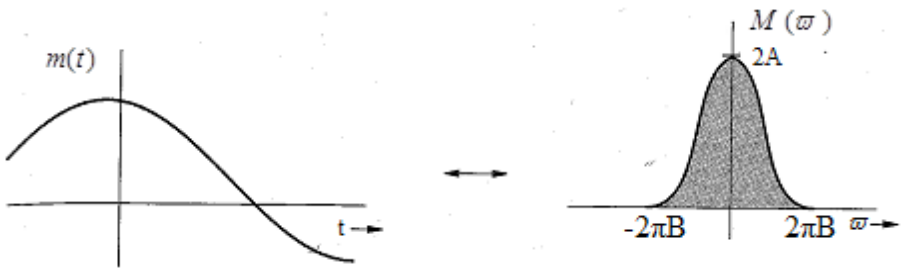

Figura 3.4 – Mensagem $m(t)$ e espectro em frequência $M(\varpi)$.

Para evitar interferência espectral entre as bandas lateral superior (USB) e inferior (LSB), a relação entre B e ϖ_c é dada pela Equação (3.4). A Figura 3.5 mostra o sinal modulado e o espectro do sinal modulado.

$$\varpi_c \geq 2\pi B \tag{3.4}$$

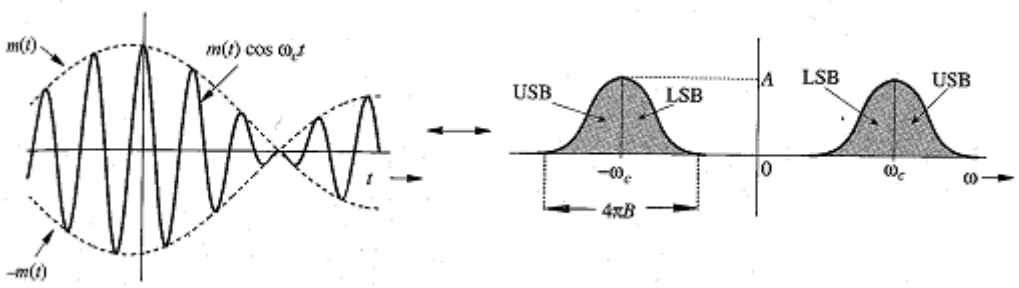

Figura 3.5 – Sinal modulado e seu espectro em frequência.

Para criar o projeto de um modulador no GRC, inicialmente foi simulado a mensagem como sendo uma senoide de 1 kHz e a portadora uma outra senoide na frequência de 10 kHz. O gráfico da mensagem e a transformada de Fourier da mesma são mostrados na Figura 3.6.

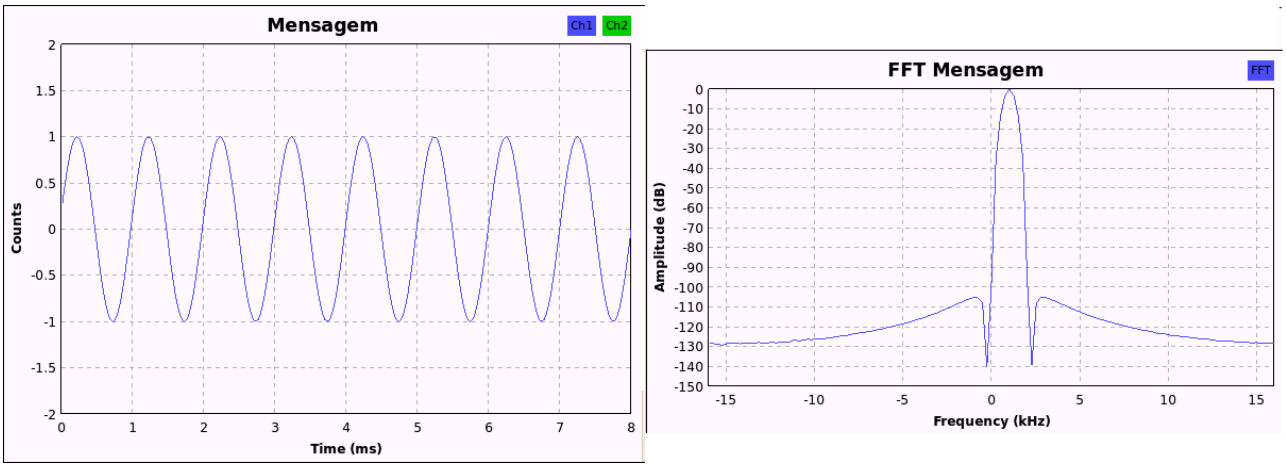

Figura 3.6 – Senoide e FFT.

Na Figura 3.6 pode ser observado que a FFT da mensagem tem o eixo y sendo a amplitude dado em dB e o eixo x sendo a frequência dado em kHz, o pico da curva está em 1 kHz e esse traçado acontece devido as ligação entre os pontos.

Para criar o modulador AM foi utilizado como referência a Figura 3.3, e, então, foi desenvolvido neste trabalho o diagrama da Figura 3.7. Esse fluxograma foi construído no GRC e a partir do osciloscópio e do analisador de espectro virtual, foram obtidos os gráficos mostrados na Figura 3.8.

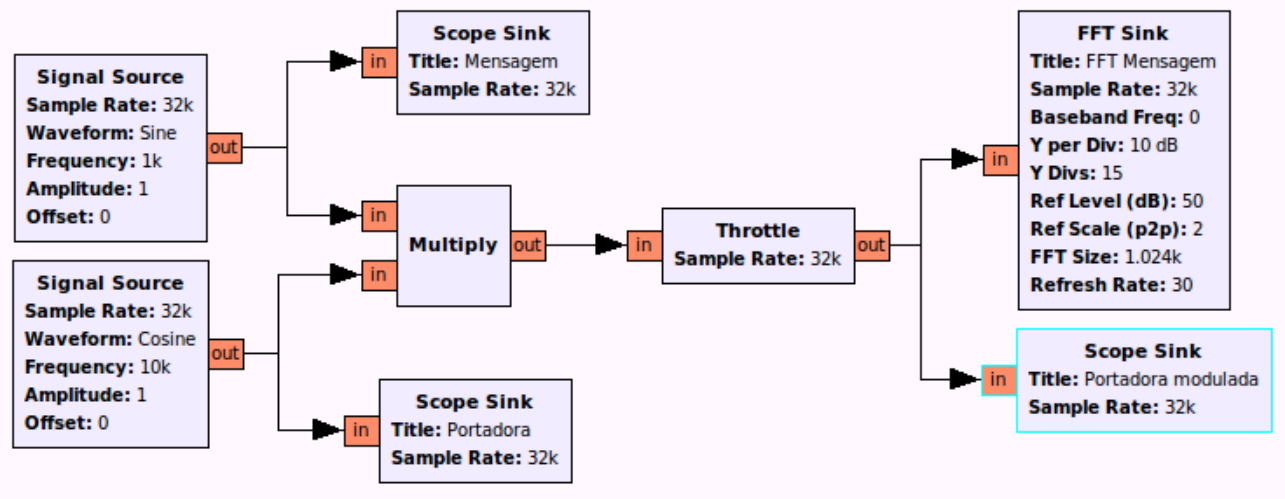

Figura 3.7 – Diagrama de blocos do modulador AM contruído.

A Figura 3.8 mostra o sinal de saída do modulador AM, ou seja, o sinal modulado e o seu respectivo espectro em frequência com os picos da FFT em 9 kHz e 11 kHz.

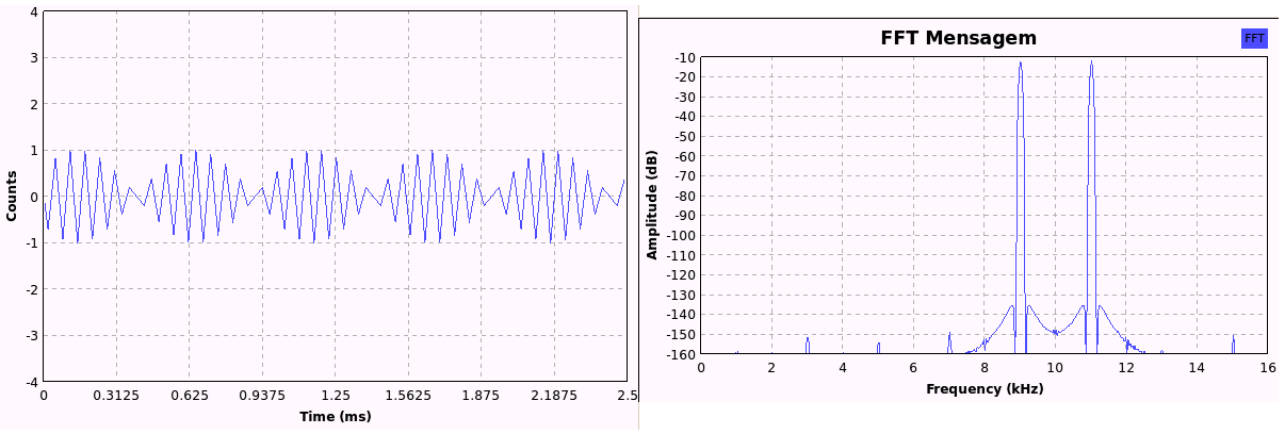

Figura 3.8 – Sinal modulado e FFT.

3.4 Receptor AM DSB – SC

Um dispositivo de rádio receptor tem o objetivo de sintonizar uma onda dentre as captadas pela antena receptora e reproduzir com fidelidade as informações contidas nela. A maioria dos receptores usados em AM são super-heteródinos que possuem melhor desempenho em sensibilidade e seletividade. O diagrama de blocos deste receptor pode ser visto na Figura 3.9.

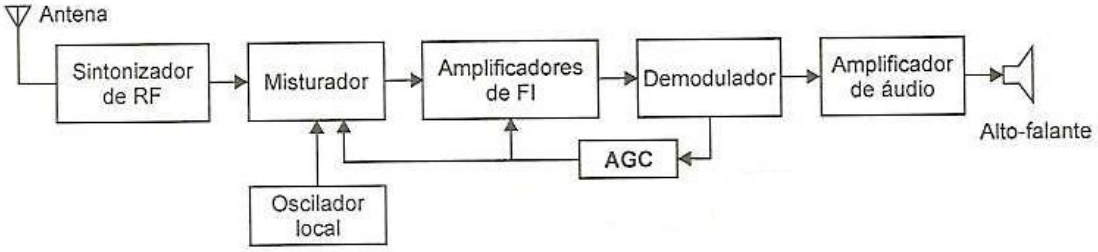

Figura 3.9 – Receptor super heteródino.

Os sinais capturados pela antena produzem corrente elétrica muito fraca que são amplificadas na primeira etapa que é composta por um filtro para sintonizar a frequência desejada e um amplificador. O sinal amplificado é aplicado ao misturador que translada a portadora ϖ_c para uma frequência intermediária (FI) gerada pelo oscilador local. A próxima etapa é amplificar o sinal de FI para então prosseguir com a demodulação e por fim acontecer a conversão do sinal em áudio.

A Figura 3.10 mostra o diagrama do demodulador AM *Double Side Band Suppressed Carrier (DSB – SC)* [13].

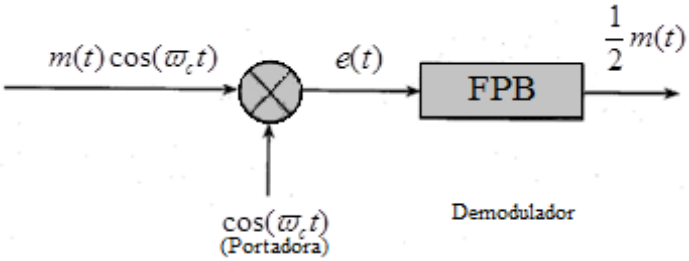

Figura 3.10 – Demodulador AM DSB – SC.

Para montar um receptor AM comumente são usados na construção do *hardware* diversos componentes, dentre eles bobinas, capacitores, resistores e transistores, e organizados de tal forma que execute apenas a função de receptor. Utilizando o conceito de SDR, é descrito abaixo como construir um receptor AM por meio do *software* GRC e o *hardware* USRP.

Para desenvolver o receptor AM no GRC foram utilizados diversos blocos oriundos do GNU Radio. A Figura 3.11 mostra o diagrama de fluxo do receptor AM construído no GRC.

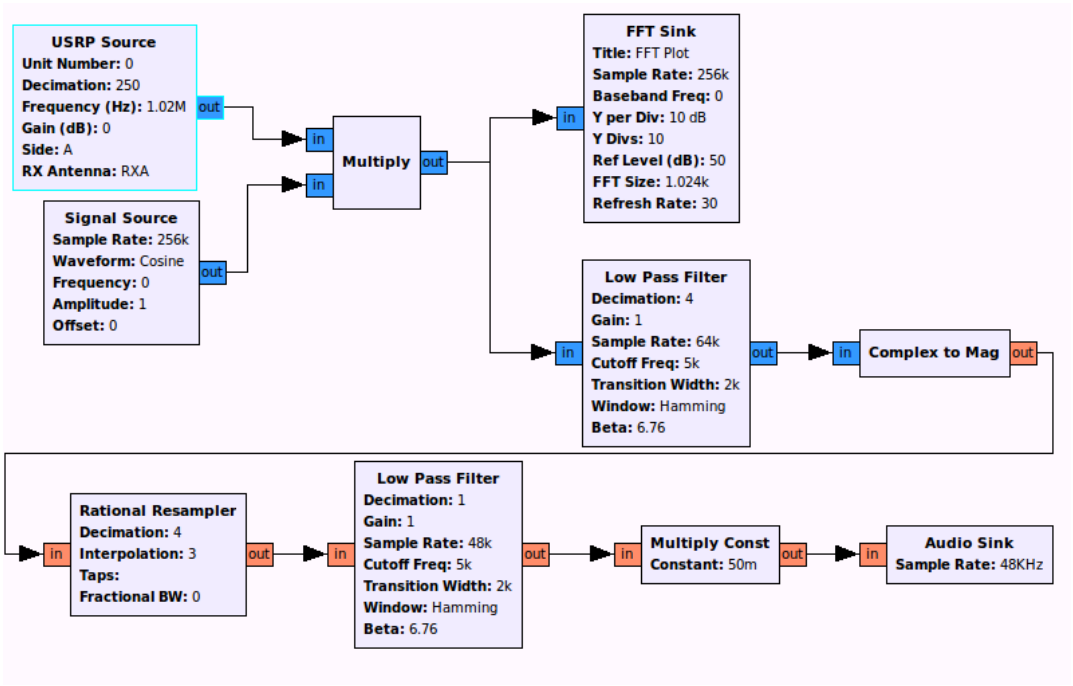

Figura 3.11 – Diagrama de blocos do receptor AM no GRC.

O sinal AM é captado por uma antena e absorvido pelo sistema por meio da placa filha LFRX do USRP. Na Figura 3.11 pode ser observado que o primeiro bloco é a fonte do sinal e tem origem no USRP, como o conversor A/D do USRP amostra o sinal a 64 M amostras por segundo (aps), foi realizado uma decimação de 250 deixando a saída do bloco "USRP *source*" com 256 kaps. Há um misturador de frequência que deve ser ajustado na frequência da estação AM. Após passar por

algumas etapas de filtragem, o sinal de saída é entregue a placa de som do computador e pode ser ouvido em tempo real.

Os resultados e as análises dos sinais obtidos a partir da construção do receptor AM no SDR mostrado nesta seção serão apresentados no capítulo 6.

3.5 Modulação Angular

Outro método de modular uma onda é pela modulação angular, na qual o ângulo da portadora é variado de acordo com o sinal de banda base. Neste método a amplitude é mantida constante. Uma característica importante da modulação angular é que ela pode oferecer uma melhor discriminação contra ruído e interferência, do que a modulação em amplitude, entretanto essa melhoria é obtida a partir do aumento da largura de banda da transmissão [14].

Admitindo que $\theta(t)$ indica o ângulo de uma portadora senoidal modulada, que supõe ser uma função do sinal de mensagem. A Equação (3.5) representa a resultante da onda modulada em ângulo.

$$s(t) = Acos[\theta(t)] \tag{3.5}$$

onde: A é a amplitude da portadora.

Se $\theta(t)$ aumentar com o tempo, a frequência média ao longo de um intervalo que varia de $t\ a\ t - \Delta t$ será dada pela Equação (3.6).

$$f_{\Delta t}(t) = \frac{\theta(t+\Delta t)-\theta(t)}{2\pi\Delta t} \tag{3.6}$$

Assim, a frequência instantânea do sinal com modulação angular é definida pela Equação (3.7).

$$\begin{aligned} f_i(t) &= \lim_{\Delta t \to 0} f_{\Delta t}(t) \\ &= \lim_{\Delta t \to 0} \left[\frac{\theta_i(t+\Delta t)-\theta_i(t)}{2\pi \Delta t}\right] \\ &= \frac{d\theta_i(t)}{2\pi dt} \end{aligned} \tag{3.7}$$

A modulação em fase e a modulação em frequência são os dois casos, em que o ângulo é variado de alguma forma com o sinal da mensagem.

3.5.1 Modulação em Fase

Na modulação em fase (PM) o ângulo $\theta_i(t)$ é variado linearmente com o sinal da mensagem$m(t)$ como mostrado na Equação (3.8) [14].

$$\theta_i(t) = 2\pi f_c t + k_p m(t) \tag{3.8}$$

onde: $2\pi f_c t$ representa o ângulo da portadora não modulada; e

k_p é uma constante de sensibilidade à fase do modulador.

Partindo da Equação (3.8) e supondo que o ângulo da portadora não modulada seja 0 em $t = 0$, então o sinal modulado em fase é descrito no domínio do tempo conforme mostra a Equação (3.9).

$$\varphi_{PM}(t) = Acos[\omega_c t + k_p m(t)] \tag{3.9}$$

Para visualizar um sinal PM foi utilizado o GRC para desenvolver o diagrama de blocos apresentado na Figura 3.12, no qual a partir de uma mensagem, que para este exemplo foi usado uma onda senoidal, a modulação em fase é obtida e mostrada na Figura 3.13.

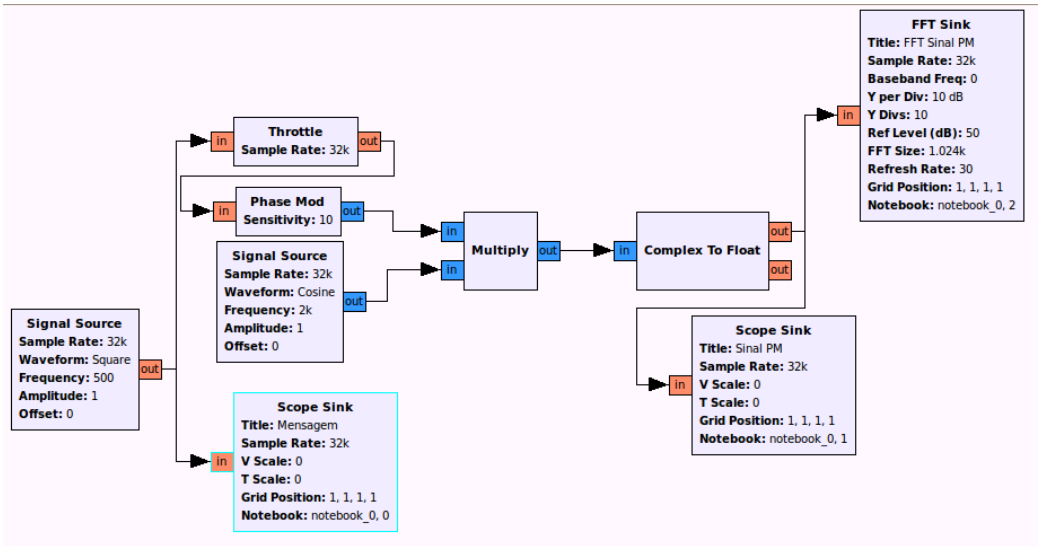

Figura 3.12 – Diagrama de blocos do GRC para o sinal PM.

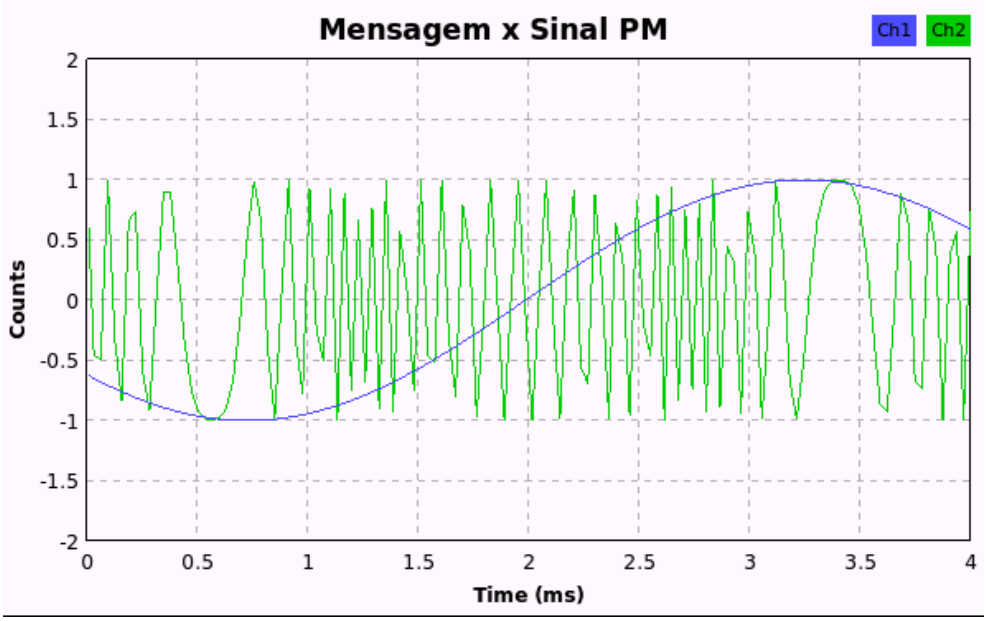

Figura 3.13 – Mensagem (Ch1) e sinal PM (Ch2).

3.5.2 Modulação em Frequência

A modulação em frequência (FM) é a forma de modulação angular na qual a frequência instantânea, $f_i(t)$, é variada linearmente com o sinal da mensagem$m(t)$ como mostra a Equação (3.10).

$$\omega_i(t) = \omega_c + k_f m(t) \tag{3.10}$$

onde: ω_c é a frequência da portadora não modulada; e

k_f é uma constante sensível a frequência do modulador.

Integrando a Equação (3.10) em relação ao tempo é obtida a Equação (3.11).

$$\theta_i(t) = \omega_c t + k_f \int_{-\infty}^{t} m(\alpha) d\alpha \tag{3.11}$$

Supondo que o ângulo da portadora não modulada seja zero em $t = 0$, o sinal modulado em frequência é descrito no domínio do tempo pela Equação (3.12).

$$\varphi_{FM}(t) = A\cos\left[\omega_c t + k_f \int_{-\infty}^{t} m(\alpha) d\alpha\right] \tag{3.12}$$

onde: A é a amplitude da portadora;

ω_c é a frequência da portadora em radianos por segundo;

k_f é a constante do modulador; e

$m(\alpha)$ é a mensagem a ser transmitida.

A fim de obter um sinal FM foi utilizado o GRC para desenvolver o diagrama de blocos apresentado na Figura 3.14, no qual a partir de uma mensagem, que para este exemplo também foi usado uma onda senoidal, a modulação em frequência foi obtida e mostrada na Figura 3.15.

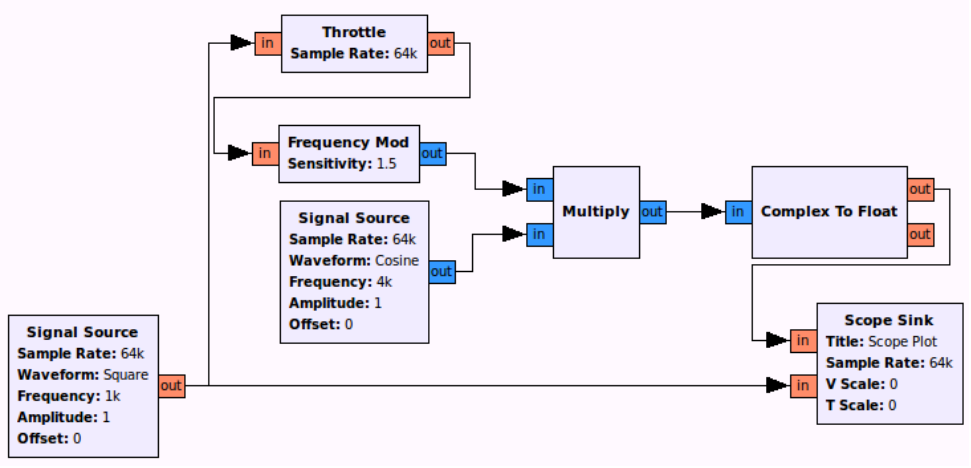

Figura 3.14 – Diagrama de blocos do GRC para gerar o sinal FM.

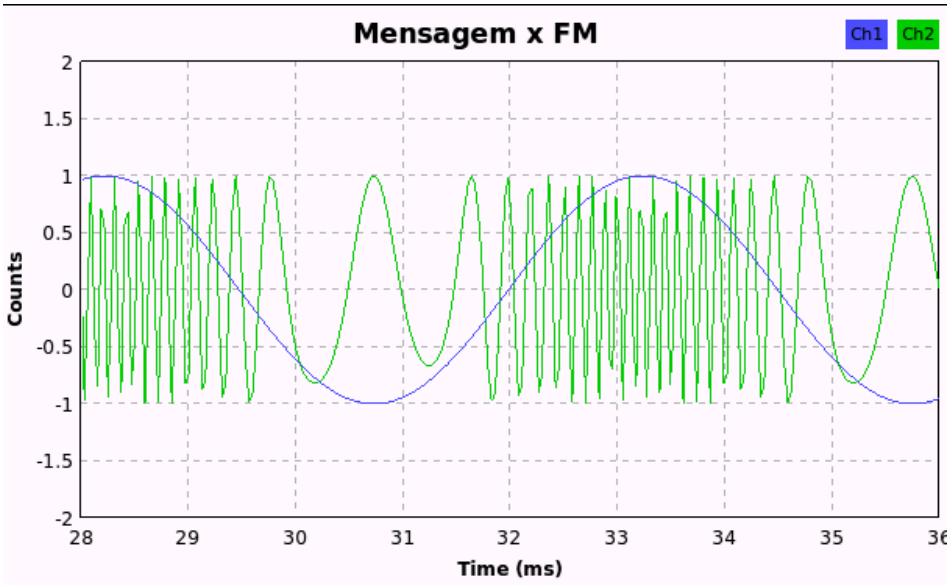

Figura 3.15 – Mensagem (Ch2) e sinal FM (Ch1).

Comparando a Equação (3.9) com a (3.12), verifica-se que o sinal FM pode ser considerado um sinal PM se a onda modulante $m(t)$ for igual a $\int_{-\infty}^{t} m(\alpha)d\alpha$, ou seja, o sinal FM pode ser gerado integrando-se $m(t)$e usando depois o resultado como a entrada para um modulador de fase. Com o mesmo raciocínio pode-se gerar um sinal PM diferenciando-se primeiro $m(t)$ e usando depois o resultado como entrada para um modulador de frequência [14]. Portanto as propriedades dos sinais

PM e FM estão relacionadas, podendo ser concentrado os estudos em apenas uma delas. A Figura 3.16 ilustra a relação dos sinais PM e FM.

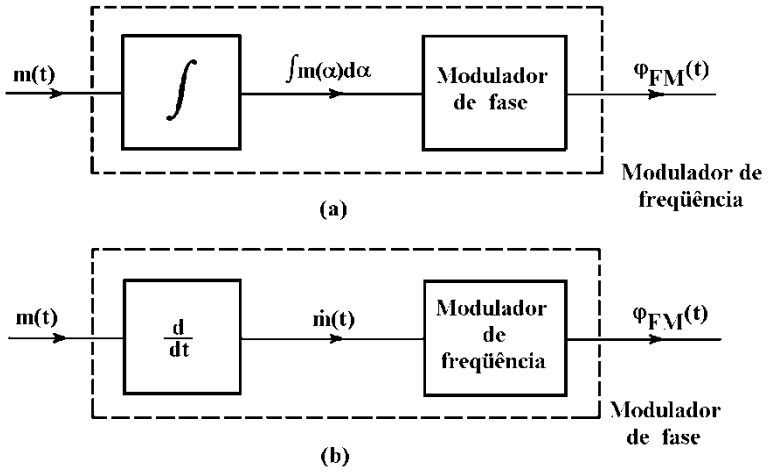

Figura 3.16 – Relação dos sinais PM e FM.

O sinal FM definido pela Equação (3.12) é uma função não linear do sinal modulante de $m(t)$, o que torna a FM um processo de modulação não linear. Em consequência, o espectro do sinal FM não está relacionado de maneira simples com o sinal modulante [14].

A modulação é realizada dentro de uma faixa fixa de frequência denominada de desvio de frequência da portadora, Δf , e representa o afastamento máximo da frequência instantânea do sinal FM com relação à frequência da portadora ω_c , e pode ser calculada conforme mostra a Equação (3.13) [13].

$$\Delta f = \frac{k_f m_p}{2\pi} \tag{3.13}$$

A razão do desvio de frequência, Δf, com a frequência modulante, f_m, é denominada de índice de modulação do sinal FM e indicado por β como mostra a Equação (3.14).

$$\beta = \frac{\Delta f}{f_m} \tag{3.14}$$

De acordo com o valor do índice de modulação, pode-se distinguir dois casos de modulação em frequência: FM faixa estreita (NBFM) para um índice de modulação muito pequeno comparado a um radiano e FM faixa larga (WBFM) para um índice de modulação muito largo em comparação com um radiano.

A seção 3.6 apresenta conceitos de um modulador WBFM e o desenvolvimento de um transmissor WBFM usando SDR para análise do seu espectro.

3.6 Transmissor WBFM

Ao determinar o espectro de um sinal FM para um valor arbitrário de índice de modulação, a partir da Equação (3.15), em geral o sinal FM produzido por um sinal modulante senoidal é ele próprio, não periódico, a menos que a frequência da portadora, ω_c, seja um múltiplo inteiro da frequência de modulação, ω_m, para simplificar essa questão pode ser feito a representação complexa de sinais como mostra a Equação (3.16).

$$\varphi_{FM}(t) = A\, cos[\omega_c t + \beta sen(\omega_m t) \tag{3.15}$$

$$\hat{\varphi}_{FM}(t) = A\, e^{j[\omega_c t + k_f \int_{-\infty}^{t} m(\alpha) d\alpha]} \tag{3.16}$$

Define-se $a(t)$ conforme a Equação (3.17).

$$a(t) = \int_{-\infty}^{t} m(\alpha) d\alpha \tag{3.17}$$

Expandindo a exponencial $e^{[jk_f a(t)]}$ na Equação (3.16), em série de potência, obtém-se a Equação (3.18).

$$\hat{\varphi}_{FM}(t) = A\left[1 + jk_f a(t) - \frac{k_f^2}{2!}a^2(t) + \cdots + j^n \frac{k_f^n}{n!}a^n(t)\right] e^{j\omega_c t} \tag{3.18}$$

Extraindo a parte real obtém-se a Equação (3.19).

$$\mathrm{Re}[\hat{\varphi}_{FM}(t)] = A\begin{bmatrix} \cos\omega_c t - k_f a(t) sen\omega_c t \\ -\frac{k_f^2}{2!}a^2(t)\cos\omega_c t \\ \frac{k_f^3}{n!}a(t) sen\omega_c t + \cdots \end{bmatrix} \tag{3.19}$$

Pela regra de Carson a largura de banda *Wide band Frequency Modulation*(WBFM) pode ser aproximada por $2\Delta f$.

Para desenvolver um transmissor FM algumas etapas devem ser construídas, a fim de ilustrar estas etapas, o diagrama de blocos de um transmissor FM básico é mostrado na Figura 3.17.

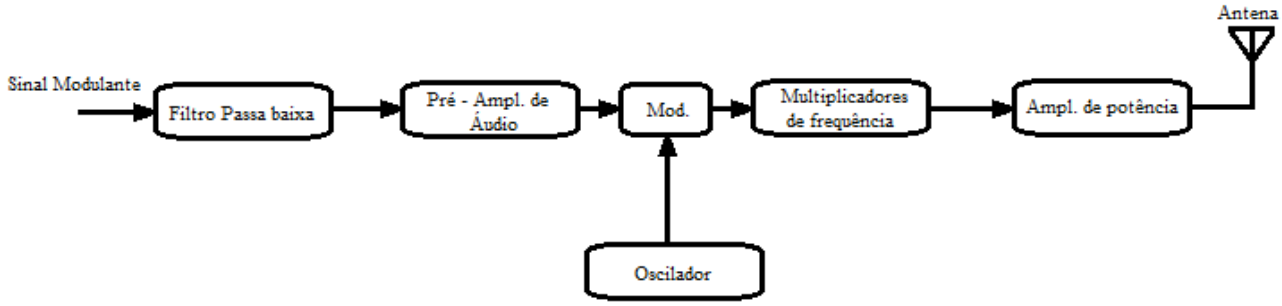

Figura 3.17 – Diagrada de blocos de um transmissor FM básico.

Na Figura 3.17 o sinal modulante passa por um filtro passa baixa antes do pré-amplificador de áudio, isso ocorre para limitar a frequência em 15 kHz. Com o objetivo de reduzir a distorção do sinal na recepção, faz-se o tratamento de pré-ênfase no sinal modulante, logo após o estágio do pré-amplificador do áudio. O pré-ênfase é a primeira parte do processo para robustecer as componentes

de frequência mais altas do sinal de áudio, para diminuir o efeito do ruído, principal causa da distorção. O circuito de pré-ênfase pode ser montado apenas com componentes passivos como mostra a Figura 3.18.

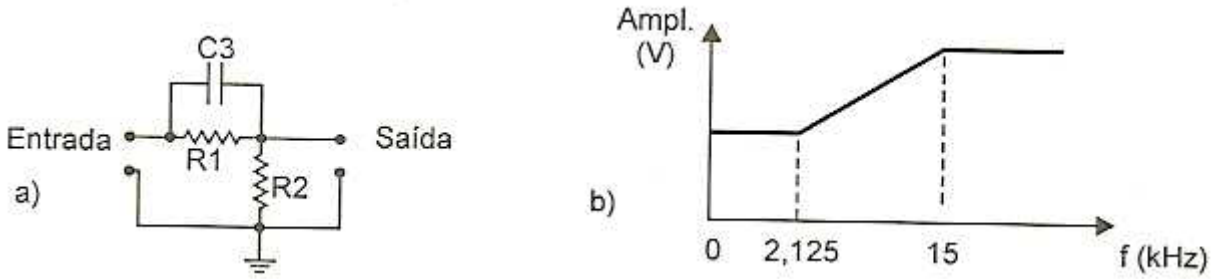

Figura 3.18 – Circuito de pré-ênfase em (a) e curva de resposta em (b).

Para desenvolver um transmissor WBFM estéreo o espectro da banda base do sinal deve ser como mostra a Figura 3.19. Os canais de áudio *Left* (L) e *Right* (R) tem como referência para a sintonia do receptor um sinal piloto, que é uma subportadora em 19 KHz. Para formar o sinal banda base do FM estéreo é necessário um processamento dos sinais para a formação dos canais de áudio L+R, L-R e o sinal piloto.

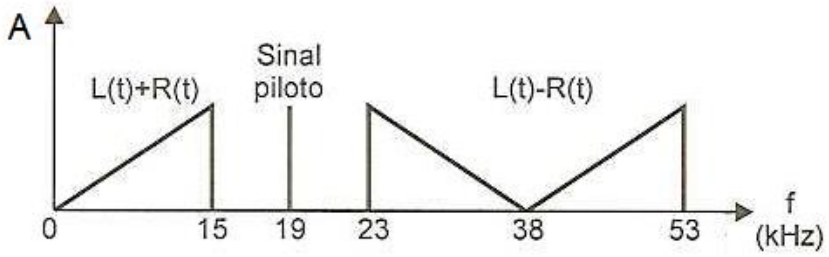

Figura 3.19 – Espectro da banda base do sinal FM.

Os três sinais são adicionados e enviados ao filtro de pré-ênfase para proporcionar um ganho nas altas frequências do sinal banda base. Em seguida, é realizada a modulação em frequência do sinal, utilizando um modulador com desvio de frequência padrão WBFM de 75 kHz. Finalmente, o

sinal é enviado ao transmissor para a multiplexação em uma determinada frequência que é a mesma a ser sintonizada nos receptores.

A Figura 3.20 mostra o fluxo do diagrama de blocos da construção do WBFM estéreo montado no GRC. A fonte de áudio pode ser um arquivo com extensão MP3 ou extensão FIFO. Cada canal é amostrado a uma taxa de 32 k amostras por segundo. Em seguida, o fluxo de cada canal é inserido nas funções de soma e subtração. A saída do somador passa por um filtro de pré-ênfase, e posteriormente é filtrada por um filtro passa baixa que realiza também a interpolação por um fator igual a oito, resultando em uma taxa de 256 k amostras por segundo [15].

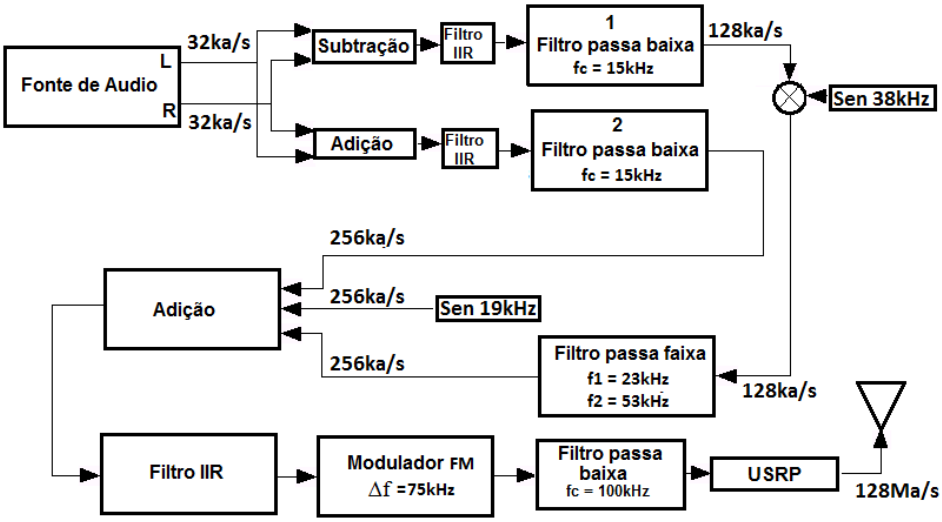

Figura 3.20 – Diagrama de blocos do transmissor WBFM.

O filtro de pré-ênfase foi implementado utilizando o filtro de resposta infinita ao impulso (IIR) existente no GNU Radio. O objetivo dessa filtragem é fornecer um ganho nas altas frequências do sinal em banda base para compensar as futuras atenuações que o sinal sofrerá devido as interferências de ruído do canal [13].

A saída do subtrator passa por um filtro de pré-ênfase e posteriormente é filtrada por um filtro passa baixa que também realiza uma interpolação, porém de fator quatro, resultando em uma taxa

de 128 k amostras por segundo. O sinal da saída do filtro passa baixa passa por um multiplicador que o multiplica por uma senóide de 38 kHz. O sinal resultante transladado de 38 kHz é filtrado pelo filtro passa faixa que além disso, realiza uma interpolação de fator dois, gerando também, um sinal de 256 k amostras por segundo [15].

O sinal piloto, uma senóide com frequência igual a 19 kHz, é amostrado diretamente a uma taxa de 256 k amostras por segundo. Os três sinais de 256 k amostras por segundo são adicionados e filtrados pelo filtro de pré-ênfase que antecede o modulador em frequência. A Figura 3.21 mostra o diagrama de blocos da construção do transmissor WBFM estéreo desenvolvido no GRC.

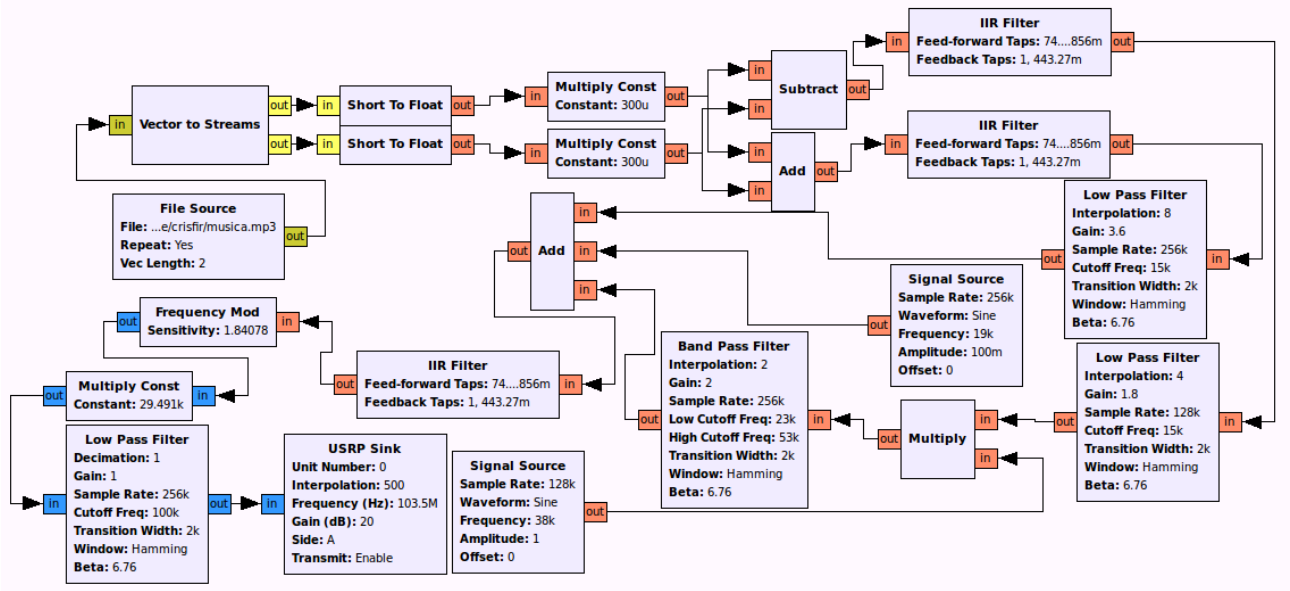

Figura 3.21–Transmissor WBFM desenvolvido no GRC.

O sinal de saída do filtro de pré-ênfase é modulado em frequência com desvio máximo de frequência da portadora de 75 kHz. Esse sinal modulado passa por um filtro passa baixa, com frequência de corte de 100 kHz, que é então transmitido ao módulo USRP, por meio de um cabo USB. Esse sinal é recebido pelo conversor D/A onde é realizada a interpolação de fator 500 resultando em um sinal com taxa de saída de 128 M amostras por segundo. Finalmente, o sinal é multiplexado em uma determinada frequência a ser sintonizada pelos receptores.

A análise do sinal em cada etapa do transmissor WBFM construído usando o SDR será apresentada no capítulo 6.

O próximo item apresenta os moduladores ASK, PSK e FSK construídos no GRC.

3.7 Moduladores Digitais ASK, FSK e PSK

A modulação digital é obtida a partir de um sinal de entrada digital que pode ser binário ou um código de vários níveis, a Figura 3.22 [16] ilustra as técnicas de modulações digitais mais comuns utilizando, na entrada, dados binários.

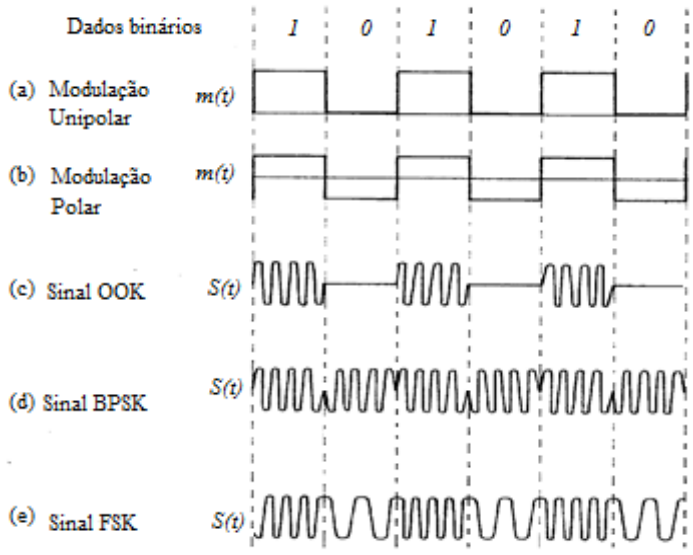

Figura 3.22 – Sinal de entrada binário e modulações por amplitude, fase e frequência [16].

On-Off Keying (OOK) ou também conhecida como *Amplitude Shift Keying* (ASK), consiste no chaveamento de uma portadora senoidal ligado ou desligado de acordo com o sinal binário unipolar [16]. A Figura 3.23 mostra o sinal ASK gerado a partir do GNU Radio. Sinal de entrada em verde e sinal de saída em azul.

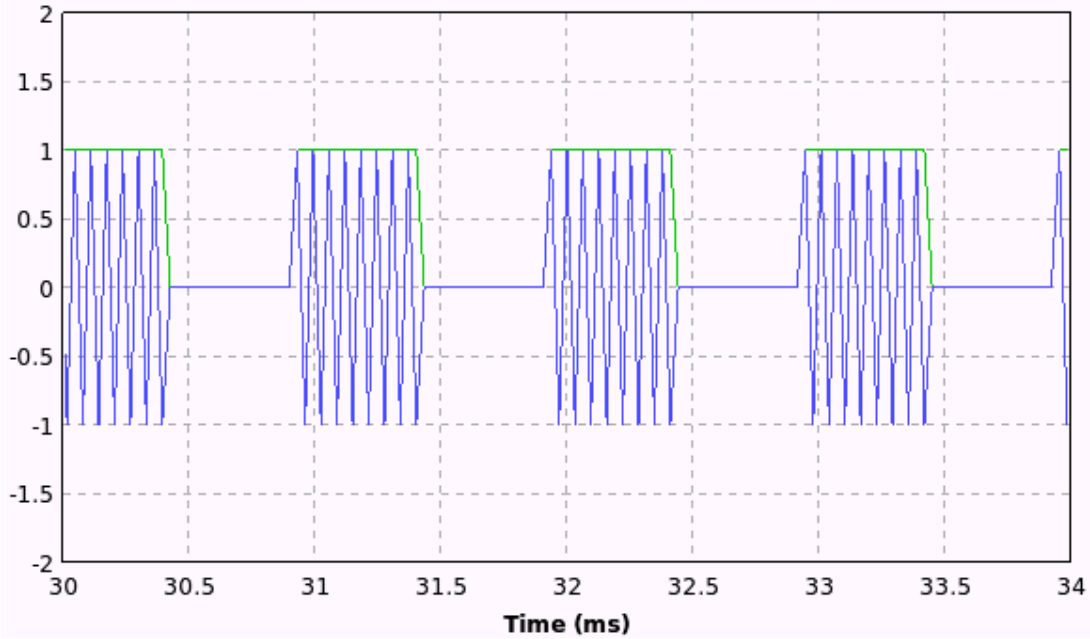

Figura 3.23 – Entrada binária e modulação ASK.

Binary Phase Shift keying (BPSK), no qual consiste na alteração da fase da portadora senoidal de 0° ou 180° de acordo com o sinal binário unipolar [16]. A Figura 3.24 mostra o sinal BPSK gerado a partir do GNU Radio, sinal de entrada em azul e sinal de saída em verde.

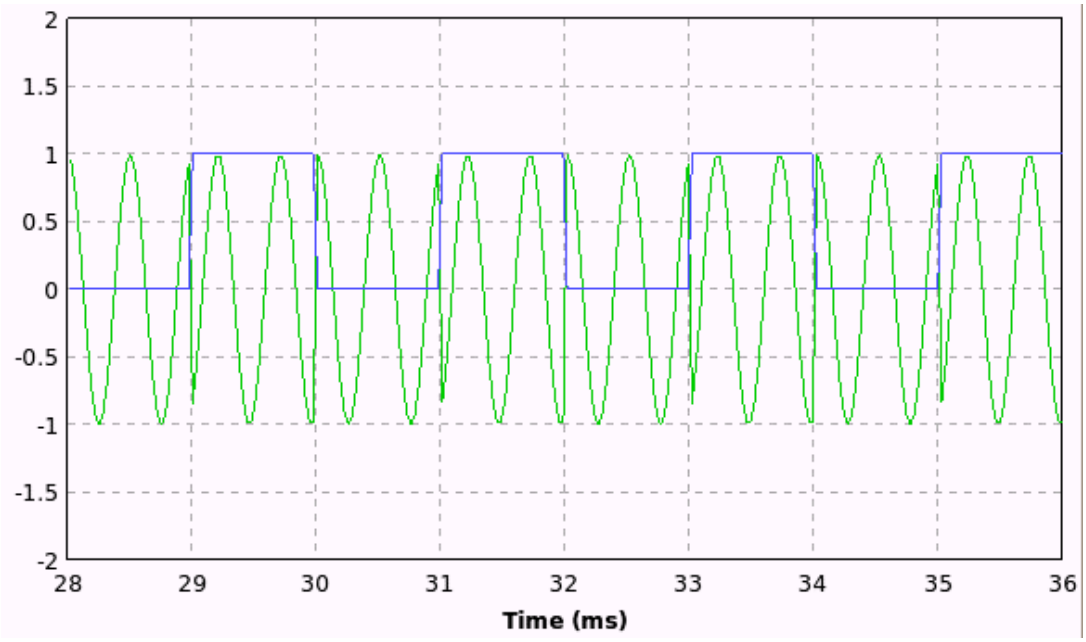

Figura 3.24 – Entrada binária e modulação BPSK.

Frequency Shift Keying (FSK), que consiste na alteração da frequência da portadora senoidal a partir do sinal digital de entrada [16]. A Figura 3.25 mostra o sinal FSK gerado a partir do GNU Radio, sinal de entrada verde e sinal de saída em azul.

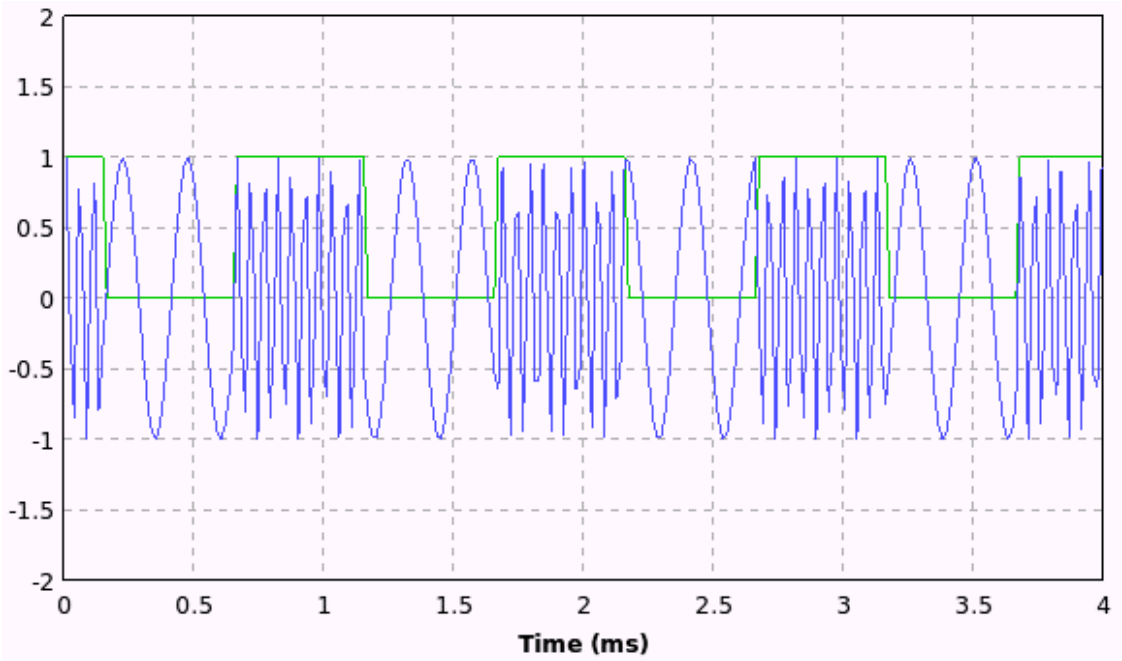

Figura 3.25 – Entrada binária e modulação FSK.

Os fluxogramas de blocos do GRC utilizados para obter as modulações ASK, PSK e FSK são os mesmos das Figuras 3.7, 3.12 e 3.14 respectivamente.

3.8 Considerações Finais Deste Capítulo

Este capítulo apresentou estudos experimentais e mostrou algumas aplicações SDR que podem ser usadas em laboratórios para práticas em engenharia.

O próximo capítulo explora o *software* do SDR mostrando a estrutura da árvore de construção do gnuradio e ainda apresenta como criar novos blocos e funções no GNU Radio e no GRC.

CAPÍTULO 4

4 CONSTRUÇÃO DE NOVOS BLOCOS

4.1 Introdução

Após a familiarização do projetista com o *software* do SDR, pode-se ter a necessidade de algumas novas funcionalidades de processamento de sinais que não estão disponíveis no GNU Radio e ou na versão gráfica GRC.

Este capítulo apresenta aspectos importantes de como inserir um novo bloco de processamento no GNU Radio e como disponibilizar a versão gráfica no GRC. Finalmente, são realizadas as considerações finais deste capítulo.

4.2 Inserindo blocos no GNU Radio e GRC

Para criar um novo bloco no GNU Radio que possa ser usado no GRC é preciso compreender a estrutura de combinações de códigos *Python*, que fornece uma organização de alto nível, com blocos de processamento em C++, que oferecem alto desempenho.

Para iniciar a criação do novo bloco, primeiramente é necessário compreender o fluxo de dados fornecido pelo GNU Radio que é intuitivo e possui dois conceitos fundamentais: os blocos de processamento de sinal e as conexões entre eles. Cada bloco possui um conjunto de portas que pode ser conectadas. As portas representam as conexões entre os blocos e determinam o tipo de fluxo dos dados. Um bloco pode ter várias portas de entrada e de saída. Blocos sem porta de entrada são

fontes e blocos sem portas de saídas são coletores. A Figura 4.1 mostra alguns tipos de blocos presentes no GRC.

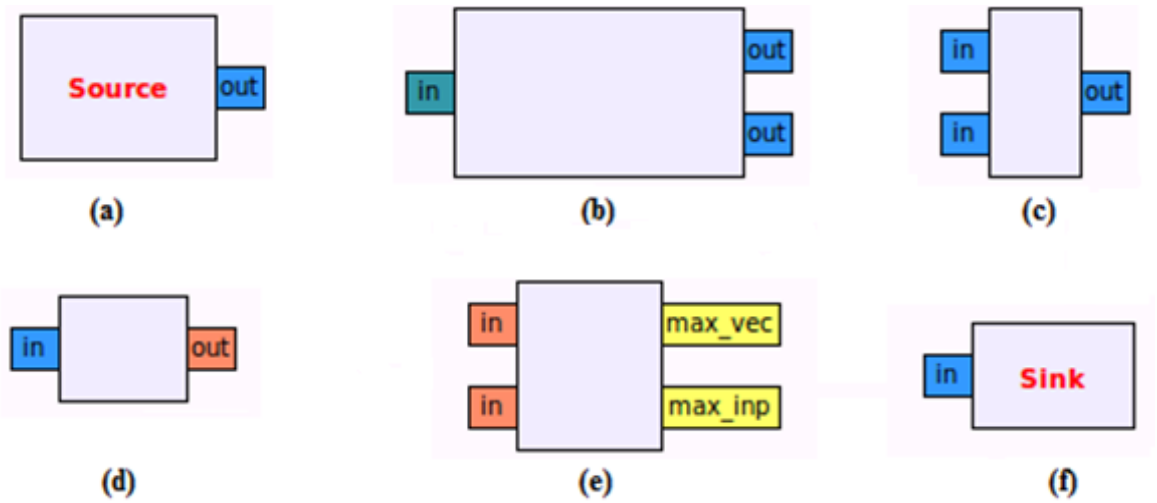

Figura 4.1 – Tipos de blocos do GRC em (a) bloco fontes, (b) bloco com uma entrada e duas saídas, (c) bloco com duas entradas e uma saída, (d) bloco com uma entrada e uma saída, (e) bloco com duas entradas e duas saídas e (f) bloco coletor.

Após o entendimento dos conceitos fundamentais que envolvem um bloco a próxima etapa é conhecer a estrutura da árvore de construção do gnuradio-core, pois o novo bloco é construído fora da árvore e posteriormente é carregado para a biblioteca do gnuradio-core de maneira dinâmica, utilizando o mecanismo de importação em *Python* por meio do SWIG e seguindo toda a estrutura lógica da árvore de construção do gnuradio-core.

Para auxiliar na construção de um novo bloco Asier Alonso, um colaborador da Wiki do GNU Radio, disponibilizou em [17] uma pasta com a estrutura da árvore e alguns arquivos que podem ser usados como modelo. A Figura 4.2 mostra a estrutura da árvore de arquivos que deve conter a pasta com o novo bloco a ser sincronizado com a biblioteca.

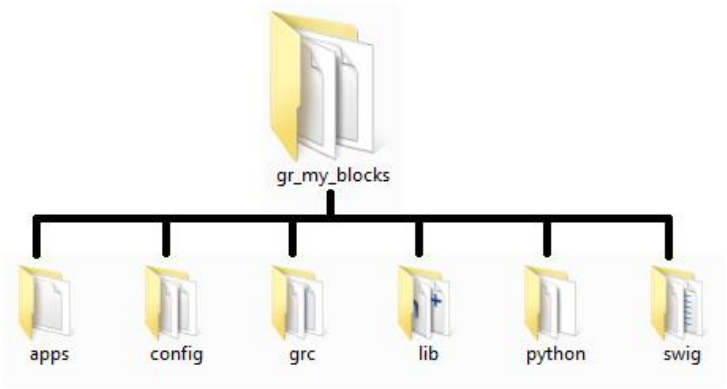

Figura 4.2 – Estrutura da árvore de arquivos de um novo bloco.

A Figura 4.2 possui em sua pasta raiz ‘*gr_my_blocks*’ outras seis pastas, na qual a pasta ‘*apps’* contém aplicações de testes e exemplos, a pasta ‘*config’* contém diferentes configurações de arquivos que não podem ser modificados, a pasta ‘*grc’* contém o arquivo com extensão .xml usado para acrescentar o bloco no GRC, a pasta ‘l*ib’* contém os arquivos fontes do novo bloco com extensão .cc e .h, a pasta ‘*python’* contém *scripts* em linguagem *Python* e a pasta ‘*swig’* possui arquivos que geram automaticamente uma interface *Python* para o bloco desenvolvido em C++.

Na Tabela 4.1 a primeira coluna mostra cada pasta da árvore de construção, a segunda coluna descreve os arquivos contidos em cada pasta da árvore que devem ser alterados para a criação de um novo bloco e na última coluna apresenta a função de cada pasta.

Para exemplificar a criação de um novo bloco foi desenvolvido um bloco com a função de amplificador. Os arquivos criados para este novo bloco seguem a estrutura da árvore mostrados na Figura 4.2 e Tabela 4.1. Após construir todos os arquivos observando cada um dos parâmetros, o próximo passo é inserir o novo módulo na árvore do gnuradio. Assim, por meio do terminal do Linux deve ser acessada a pasta raiz da árvore de construção do novo bloco, e, então, os comandos listados Tabela 4.2 devem ser executados.

Diretório	Arquivos que devem ser modificados	Função do diretório
gr_my_block	configure.ac makefile.am makefile.common	Pasta do novo módulo que contém arquivos e diretórios que devem ser alterados para acrescentar o novo bloco de processamento.
config	—	Contém arquivos de configuração autoconf que auxiliam na execução do scripts configure.
lib	novo_bloco.cc novo_bloco.h makefile.am	Contém arquivos fontes C++ que geram o novo bloco de processamento de sinal no GNU Radio.
apps	—	Contém os programas, aplicações grc, scripts e/ou outros executáveis instalados na bin do diretório do sistema gnuradio.
grc	arquivo.xml makefile.am	Contém os arquivos.xml que são os envelopes que descrevem os blocos para o grc.
python	o_init_py makefile.am	Contém módulos em python que são instalados no sistema do diretório python lib. Este é o local apropriado para colocar a hierarquia de blocos e as classes uteis.
Swig	arquivo.i makefile.am	Contém arquivos gerenciados pela ferramenta SWIG com função de criar uma interface python para as classes C++.

Tabela 4.1 – Árvore dos arquivos para um novo bloco.

```
$ sudo ./bootstrap
$ sudo ./configure
$ cd swig
$ sudo make generate-makefile-swig
$ cd ..
$ sudo make
$ sudo make install
$ sudoldconfig
```

Tabela 4.2 – Sequência de comandos para inserir novo bloco.

Uma vez que foram executados esses comandos com sucesso o bloco criado pode ser usado no GNU Radio por meio de comandos Python e no GRC utilizando os blocos. Alguns materiais de apoio para a construção do novo bloco podem ser encontrados na Wiki do GNU Radio em [18]. A Figura 4.3 mostra a tela principal do GRC após a inserção do novo bloco.

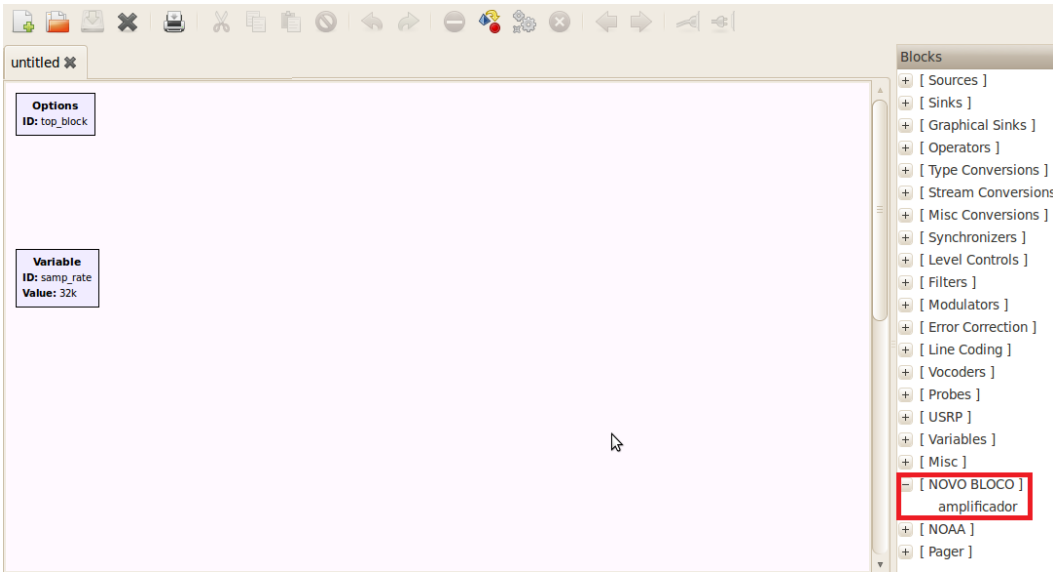

Figura 4.3 – Novo bloco no GRC.

Para testar se a função implementada no novo bloco está funcionando no GRC, foi criado o diagrama de blocos, mostrado na Figura 4.4, no qual a partir de um sinal de entrada senoidal de amplitude 1 V é aplicado ao bloco ‘amplificador’ com ganho de 3 e o gráfico deste sinal é apresentado na Figura 4.5.

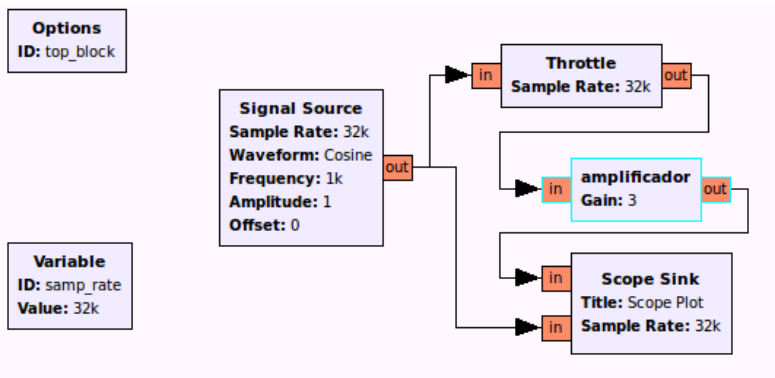

Figura 4.4 – Diagrama de blocos do amplificador.

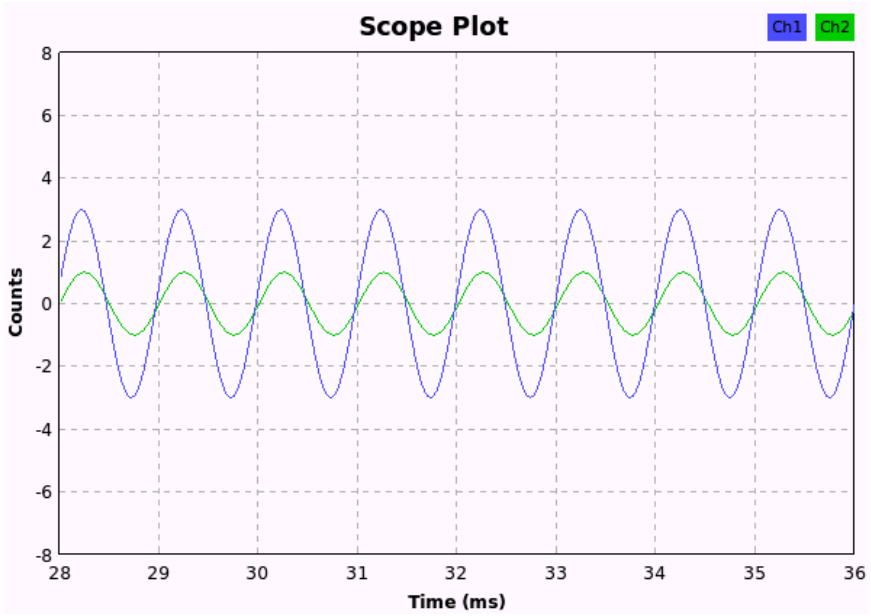

Figura 4.5 – Sinal de entrada (Ch2) e Sinal de Saída (Ch1) do amplificador.

Para utilizar a função do novo bloco apenas no GNU Radio sem a facilidade gráfica do GRC, foi usada a linguagem de programação *Python* para construir uma rotina contendo a nova função, a rotina obtida do fluxo gráfico da Figura 4.4, cujo resultado foi apresentado na Figura 4.5 é mostrada no Código 4.1.

```
#!/usr/bin/env python
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import scopesink2
fromgrc_gnuradioimportwxgui as grc_wxgui
from optparse import OptionParser
import gr_my
import wx
class Amplificador(grc_wxgui.top_block_gui):

        def __init__(self):
                grc_wxgui.top_block_gui.__init__(self, title="Amplificador")
self.samp_rate = samp_rate = 32000
                self.gr_my_amplifier_ff_0 = gr_my.gr_make_my_amplifier_ff(3)
                self.gr_sig_source_x_0 = gr.sig_source_f(samp_rate, gr.GR_COS_WAVE, 1000, 1, 0)
                self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, samp_rate)
                self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
                        self.GetWin(),
                        title="Scope Plot",
                        sample_rate=samp_rate,
                        v_scale=0,
```

```
                        v_offset=0,
                        t_scale=0,
                        ac_couple=False,
                        xy_mode=False,
                        num_inputs=2,
                )
                self.Add(self.wxgui_scopesink2_0.win)
self.connect((self.gr_sig_source_x_0, 0), (self.gr_throttle_0, 0))
                self.connect((self.gr_throttle_0, 0), (self.gr_my_amplifier_ff_0, 0))
                self.connect((self.gr_my_amplifier_ff_0, 0), (self.wxgui_scopesink2_0, 0))
                self.connect((self.gr_sig_source_x_0, 0), (self.wxgui_scopesink2_0, 1))

        defset_samp_rate(self, samp_rate):
                self.samp_rate = samp_rate
                self.gr_sig_source_x_0.set_sampling_freq(self.samp_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.samp_rate)

if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = Amplificador()
        tb.Run(True)
```

Código 4.1 – Amplificador de sinal em Python.

4.3 Construindo o bloco Wavelet no GRC

Analisando as funções de processamento do GNU Radio como mostra a Figura 2.11, verificou-se que o bloco da Transformada Wavelet (WT) não está disponível na versão gráfica do GRC. Para inserir o bloco da WT no GRC a primeira etapa é compreender a rotina em C++ da wavelet, as variáveis e as características dessa rotina.

O GNU Radio utiliza a *GNU Scientific Library* (GSL) que é uma biblioteca livre com programas desenvolvidos nas linguagens de programação C e C++. Essa biblioteca fornece uma ampla gama de rotinas matemáticas com mais de 1000 funções no total [19]. A WT desenvolvida pela GSL, e que é usada neste trabalho é a Transformada Wavelet Daubechies. A Figura 4.6 ilustra a relação de arquivos da rotina wavelet desenvolvida no GSL.

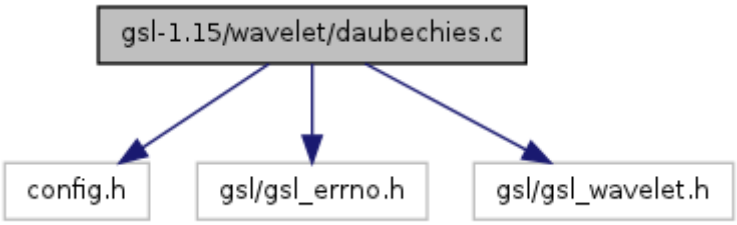

Figura 4.6 – Diagrama de blocos da relação de arquivos wavelet do GSL.

Para compreender a rotina do arquivo *gsl_wavelet* (ANEXO A) utilizou-se o manual de referências GSL disponível em [20]. A transformada wavelet contínua e sua inversa são definidas pelas Equações (4.1) e (4.2) [20].

$$W(s,\tau) = \int_{-\infty}^{\infty} f(t)\Psi_{s,\tau}(t)dt \tag{4.1}$$

$$f(t) = \int_{0}^{\infty} ds \int_{-\infty}^{\infty} w(s,\tau)\Psi_{s,\tau}(t)d\tau \tag{4.2}$$

onde a função base $\Psi_{s,\tau}$ é obtida por escala e translação de uma única função denominada de wavelet mãe.

A versão discreta da Transformada Wavelet atua igualmente no espaço amostral, com escala fixa (s,τ). Os eixos de tempo e frequência são amostrados por pares nas escalas 2^j através de um parâmetro de nível *j*. A família resultante de funções $\Psi_{j,n}$ constitui uma base ortonormal para sinais quadrados integráveis. A transformada wavelet discreta é também referida como Transformada Rápida Wavelet [20].

A estrutura *gsl_wavelet* contém os coeficientes do filtro que define a wavelet e quaisquer parâmetros *offset* associados. A partir da função:

*gsl_wavelet * gsl_wavelet_alloc (constgsl_wavelet_type * T, size_t k).*

Verificou-se que esta função aloca e inicializa um objeto wavelet do tipo T. O parâmetro *k* seleciona e especifica o membro da família wavelet. Um ponteiro nulo é retornado se avaliado memória insuficiente ou se um membro não disponível for selecionado [20].

Sabendo que para este trabalho foi utilizado a Wavelet Daubechies, o parâmetro *type, T,* foi fixado para *gsl_wavelet_daubechies*ou*gsl_wavelet_daubechies_centered.* Esta é a família Wavelet Daubechies de fase máxima com $\frac{k}{2}$momentos de fulga. Essa wavelet possui $k = 4, 6, \ldots, 20$ com k sendo um número par. As funções:

int gsl_wavelet_transform (*constgsl_wavelet * w, double * data, size_t stride, size_t n, gsl_wavelet_directiondir, gsl_wavelet_workspace * work*)

int gsl_wavelet_transform_forward (*constgsl_wavelet * w, double * data, size_t stride, size_t n, gsl_wavelet_workspace * work*)

int gsl_wavelet_transform_inverse (*constgsl_wavelet * w, double * data, size_t stride, size_t n, gsl_wavelet_workspace * work*)

Podem ser usadas com a transformada wavelet direta ou inversa de comprimento *n*, com passos *stride* do vetor *data*. O comprimento da transformada é *n* restrito a potências de dois. Para a versão da transformada da função o argumento *dir* pode ser tanto direto (1) quanto inverso (-1). Uma área de trabalho *work* de comprimento *n* deve ser fornecida [20].

Para a transformada direta os elementos da matriz original são passados pela transformada wavelet discreta,$f_i \rightarrow w_{j,k}$ e são armazenados por pacotes, onde *j* é o índice do nível de *j = 0 ... J-1* e *k* é o índice do coeficiente de dentro de cada nível, *k = 0 ... 2^j -1*. O número total de níveis são $J = \log_2 n$. Os dados de saída são $(s_{-1,0}, d_{0,0}, d_{1,0}, d_{1,1}, d_{2,0}, \ldots, d_{j,k}, \ldots, d_{j-1,2^{j-1}-1})$ onde o primeiro elemento é o coeficiente de aproximação$(s_{-1,0})$, seguido dos coeficientes de detalhe $d_{j,k}$, para cada nível j. A transformada inversa desses coeficientes geram os dados originais [20].

Esta função retorna o status de GSL_SUCCESS após a conclusão, e pode ter retorno GSL_EINVAL caso *n* não seja inteiro e uma potência de 2, ou se o vetor fornecido na entrada da função estiver adequado [20].

Após a análise detalhada do programa da Wavelet Daubechies, o próximo passo é conhecer a estrutura da árvore de construção do gnuradio-core. Navegando nas pastas do gnuradio-core pode ser encontrado os arquivos gr_wavelet.cc e gr_wavelet.h (ANEXO B) esses arquivos estão

relacionados ao arquivo gsl_wavelet descrito acima. Após análises desses arquivos foi possível verificar que os dados de entrada do bloco wavelet é um vetor, pois é uma transformada unidimensional e que três parâmetros devem ser configurados nesse bloco, o tamanho (*size*), a ordem (*order*) e se a transformada é direta ou inversa (*forward*). A relação de acesso aos blocos do GNU Radio no qual a função wavelet faz uso é mostrado na Figura 4.7 [21].

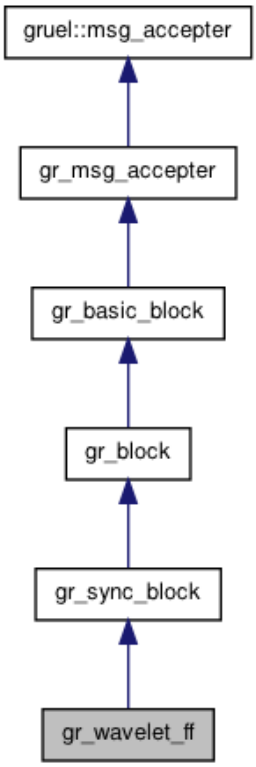

Figura 4.7 – Fluxo da Wavelet no GNU Radio.

O prefixo *gr* nos nomes das funções, da Figura 4.7, indica a origem de cada função como referência ao pacote ou biblioteca em que ela está localizada, o pacote *gr* é o módulo Python principal do gnuradio. A função *gr_wavelet_ff* executa as rotinas da transformada wavelet Daubechies unidimensional, a função *gr_sync_block* é um bloco de sincronia indicando que possui uma entrada para uma saída (1:1), a *gr_block* é uma classe base abstrata que indica que o bloco deve conter um conjunto de entradas e saídas *streams* para garantir que os dados de saída tenham a

mesma taxa. A função *gr_basic_block*é uma classe base abstrata para todos os blocos de processamento de sinal, todos os blocos que possuem um conjunto de entrada e saída devem instanciar esta classe. A *gr_msg_accepter* verifica se todos os dados exigidos estão corretos e dentro das restrições de execução.

Para criar o arquivo xml do bloco *Wavelet*, na pasta 'grc' do gnuradio-core foi necessário seguir alguns padrões essenciais, como por exemplo, quantidade de entradas e saídas, nome do bloco, identificação da chave (*key*) do bloco, a categoria que se enquadra o bloco, qual módulo contém o bloco (*import*), indicação de qual arquivo contém as funções do bloco (*make*), construção das características dos parâmetros do bloco, por fim os tipos de entrada e saída do bloco. A rotina criada neste trabalho para criar o bloco *Wavelet* é mostrada no Código 4.2 mostrado abaixo.

```
<?xml version=”1.0”?>
<block>
<name>Wavelet</name>
<key>gr_wavelet_ff</key>
<category>filters</category>
<import>import gr</import>
<make>gr.gr_wavelet_ff($d_size, $d_order, $d_forward)</make>
<param>
<name>Wavelet Size</name>
<key>d_size</key>
<value>1024</value>
<type>int</type>
</param>
<param>
<name>Wavelet Order</name>
<key>d_order</key>
<value>20</value>
<type>int</type>
</param>
<param>
<name>Wavelet Forward/Reverse</name>
<key>d_forward</key>
<type>enum</type>
<option>
<name>Forward</name>
<key>True</key>
</option> -<option>
<name>Reverse_wav</name>
<key>False</key>
</option>
</param>
<sink>
<name>in</name>
```

```
<type>double</type>
</sink>
<source>
<name>out</name>
<type>double</type>
</source>
</block>
```

Código 4.2 – Rotina xml do bloco *wavelet*.

Construído o arquivo, a próxima etapa é inseri-lo na pasta '*grc*' do gnuradio-core, o caminho dessa pasta é /usr/local/share/gnuradio/grc/blocks. Após realizada esta etapa, ao abrir o GRC o novo bloco pode ser encontrado. A Figura 4.8 mostra o bloco wavelet no GRC e a tela de edição dos parâmetros deste bloco.

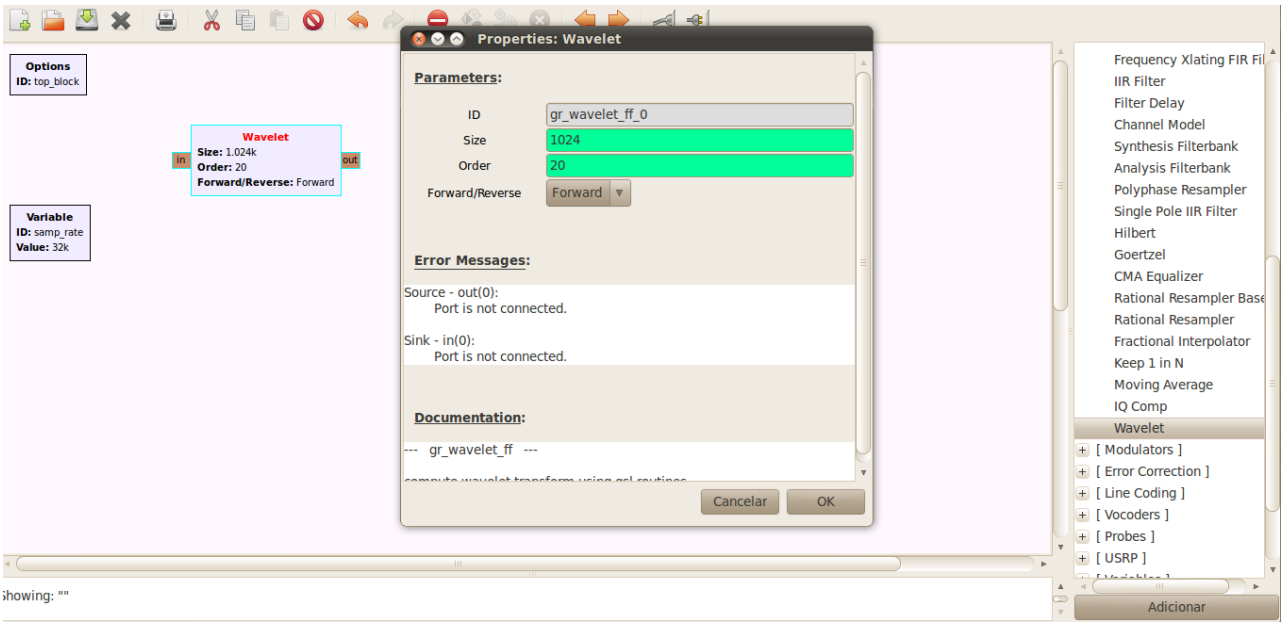

Figura 4.8 – Bloco Wavelet no GRC e parâmetros.

Na área de alteração de parâmetros do bloco *Wavelet* mostrado na Figura 4.8, a variável *Size* indica o tamanho do vetor de entrada e a quantidade de níveis para decomposição, ou seja, se o valor de *Size* é igual a 1024, os primeiros 1024 valores compõem o vetor de entrada e o número de níveis é igual 9 conforme descrito na seção 4.2 deste trabalho. O parâmetro *Order* indica a ordem

da Daubechies a ser executado, ou seja, se *Order* é igual a 20 o bloco *Wavelet* aplica a Db10 e o último parâmetro que pode ser alterado é a indicação de *Forward/Reverse* que permite que seja escolhida a transformada direta ou inversa.

4.4 Considerações Finais Deste Capítulo

Neste capítulo foram apresentadas as etapas utilizadas para inserir um bloco no GNU Radio e disponibilizar a versão gráfica no GRC para ser usada neste trabalho.

O capítulo 5 aborda fundamentos teóricos da Wavelet Daubechies, testa e aplica a transformada para ser usada no SDR.

CAPÍTULO 5

5 BLOCO WAVELET DAUBECHIES

5.1 Introdução

O objetivo de qualquer transformada é revelar características do sinal que não são aparentes no sinal original. A Transformada Wavelet (WT) é uma ferramenta que separa o sinal em diferentes frequências e divide cada componente com uma resolução combinada com sua escala [22]. As Wavelets são utilizadas para decomposição de sinais no domínio da frequência e do tempo, tornando possível visualizar em que momento determinadas frequências ocorrem.

Este capítulo apresenta os fundamentos da Transformada Wavelet Daubechies, mostra os testes utilizados para avaliar o funcionamento do bloco criado no GRC e aplica esta transformada no SDR. Finalmente, são realizadas as considerações finais deste capítulo.

5.2 Fundamentos da Transformada Wavelet Daubechies.

A Transformada Wavelet vem demonstrado ser muito útil na análise de sinais descritos como: aperiódicos, ruidosos, intermitentes, transiente e assim por diante [23]. A Wavelet permite uma análise multirresolução que é uma teoria que incorpora e unifica técnicas de várias áreas, incluindo a codificação de sub-bandas, o processamento de sinais, a filtragem de quadratura espelhada, o reconhecimento digital de voz e o processamento piramidal de imagens [24].

A WT é um método no qual a resolução tempo-frequência da transformada varia, permitindo uma descrição mais precisa do comportamento do sinal, ou seja, a WT fornece a localização da

frequência proporcionalmente ao seu nível e, consequentemente, a localização no tempo se torna mais precisa para altas frequências. Essa transformada é uma ferramenta que separa os dados em componentes diferenciados na frequência, e, então, estuda cada componente com a resolução semelhante a sua escala [25].

Para uma análise por meio da transformada wavelet o sinal a ser analisado é multiplicado por uma função denominada wavelet mãe e as transformadas são calculadas separadamente por segmentos diferentes do sinal no domínio do tempo. A transformada wavelet contínua é definida como a soma ao longo do tempo de sinais multiplicados por versões escaladas e deslocadas da função wavelet, $\Psi(t)$, mostrado na Equação (5.1).

Um conjunto completo de funções wavelets básicas, $\Psi_{s,\tau}(t)$, pode ser gerado pela translação e dilatação da função wavelet básica, $\Psi(t)$, como mostrado na Equação (5.1).

$$\Psi_{s,\tau}(t) = \frac{1}{\sqrt{|s|}}\,\Psi\left(\frac{t-\tau}{s}\right) \tag{5.1}$$

onde $s\ e\ \tau \in R\ e\ s\ \neq 0$

Portanto a função wavelet varia de escala e é deslocada ao longo no sinal analisado, gerando coeficientes $s\ e\ \tau$ que podem ser plotados em um gráfico no domínio tempo escala.

A partir da Equação (5.1) dado um sinal $f(t)$ a WT consiste na projeção de $f(t)$em um conjunto de funções de análise $\Psi_{s,\tau}(t)$, o resultado da WT são muitos coeficientes, C, que são funções de escala e posição como mostrado na Equação (5.2).

$$C_{escala,posição} = \int_{-\infty}^{\infty} f(t)\Psi_{escala,posição}(t)dt \tag{5.2}$$

Existe uma relação entre a escala da Wavelet e a frequência, ou seja, uma escala pequena resulta em uma Wavelet comprimida onde os detalhes mudam rapidamente em alta frequência,

enquanto que uma escala grande resulta em uma Wavelet expandida onde os detalhes mudam lentamente em baixa frequência [25].

Na análise Wavelet ocorre a decomposição em múltiplos níveis, em que, cada sinal é decomposto em uma *aproximação* e um *detalhe*. A *aproximação* é a componente de alta escala e baixa frequência do sinal. O *detalhe* é a componente de baixa escala e alta frequência do sinal. A *aproximação* pode ser decomposta em uma *aproximação* e um *detalhe*, para uma decomposição de$n$ níveis, existem $n+1$ possibilidades de decompor o sinal. A árvore de decomposição é mostrada na Figura 5.1. O sinal S pode ser reconstruído como mostrado na Equação(5.3).

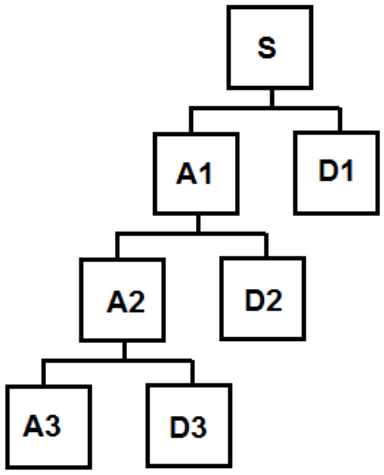

Figura 5.1 – Árvore de decomposição da WT.

$$S = A1 + D1 = A2 + D2 + D1 = A3 + D3 + D2 + D1 \tag{5.3}$$

Neste trabalho foi utilizada a Transformada Wavelet Discreta da família de Wavelets Daubechies (DbN onde *N* indica a ordem) que são wavelets ortogonais, essa família é composta por 9 membros, Db2 a Db10. A Figura 5.2 mostra as funções da família Wavelet Daubechies.

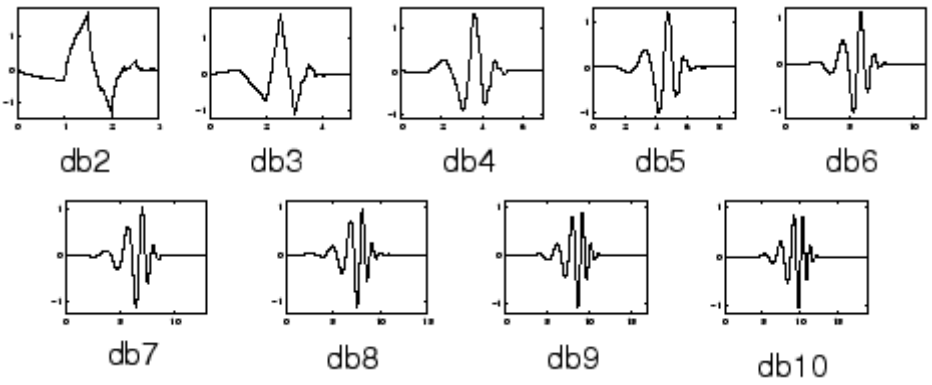

Figura 5.2 – Família de Wavelets Daubechies.

As wavelets discretas são utilizadas para decomposição e filtragem de séries temporal sem provocar redundâncias de coeficientes entre as escalas.

A Transformada Wavelet multirresolução pode ser melhor compreendida como um filtro passa baixa (FPB), que são os coeficientes de detalhes da decomposição, e outro filtro passa alta (FPA), que são os coeficientes de aproximação da decomposição da Wavelet. A Figura 5.3 ilustra a decomposição multirresolução.

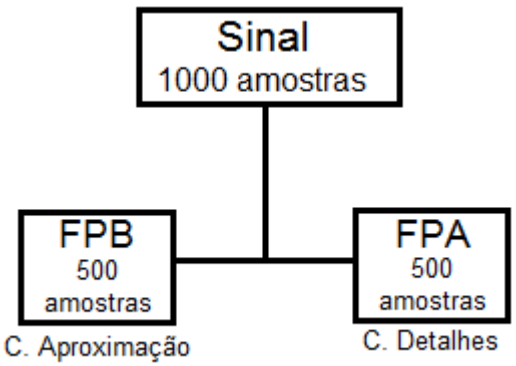

Figura 5.3 – Decomposição do sinal.

A seção 5.3 apresenta um teste usando a WT do GRC e compara com a decomposição wavelet do MATLAB.

5.3 Testando o bloco Wavelet Daubechies

Para verificar se a transformada tanto direta quanto a inversa do bloco criado estão funcionando adequadamente foi construído um fluxo gráfico no GRC, no qual após aplicado um sinal de entrada um primeiro bloco *wavelet* executa a WT direta e um segundo bloco aplica a WT inversa obtendo então o sinal original. A Figura 5.4 mostra o diagrama de blocos construído para este primeiro teste.

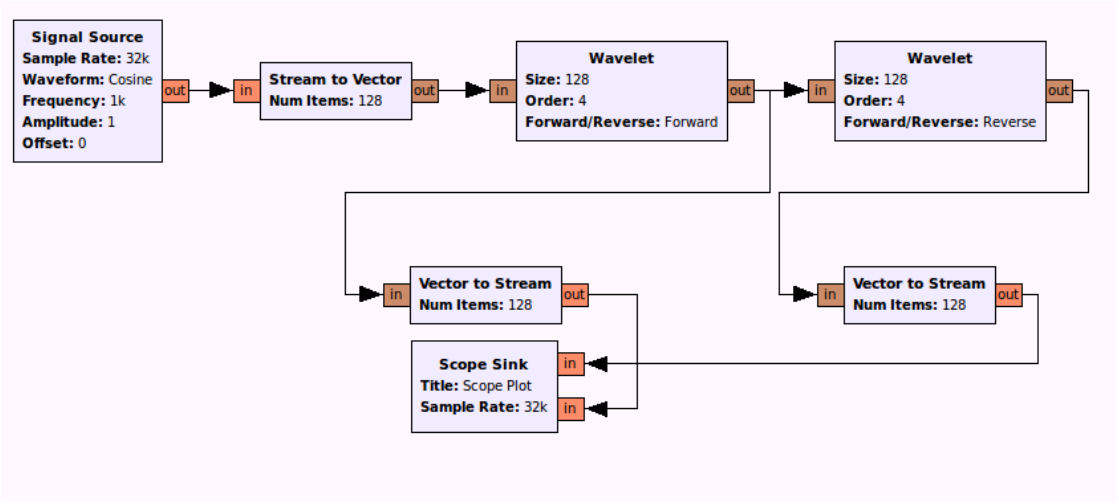

Figura 5.4 – Diagrama de blocos de teste da Wavelet Daubechies no GRC.

O primeiro bloco da Figura 5.4 mostra a fonte de sinal que é uma cossenoide, o segundo bloco converte um vetor e um *stream* para então aplicar a WT direta ao sinal. A saída do bloco *wavelet* é inserida em um osciloscópio virtual e na entrada de um outro bloco WT, porém, executando agora a função de transformada inversa. A saída deste segundo bloco também é mostrada em um osciloscópio virtual. A Figura 5.5 mostra no canal 1 (ch1) a WT direta e o no canal 2 (ch2) a WT inversa do sinal cossenoidal.

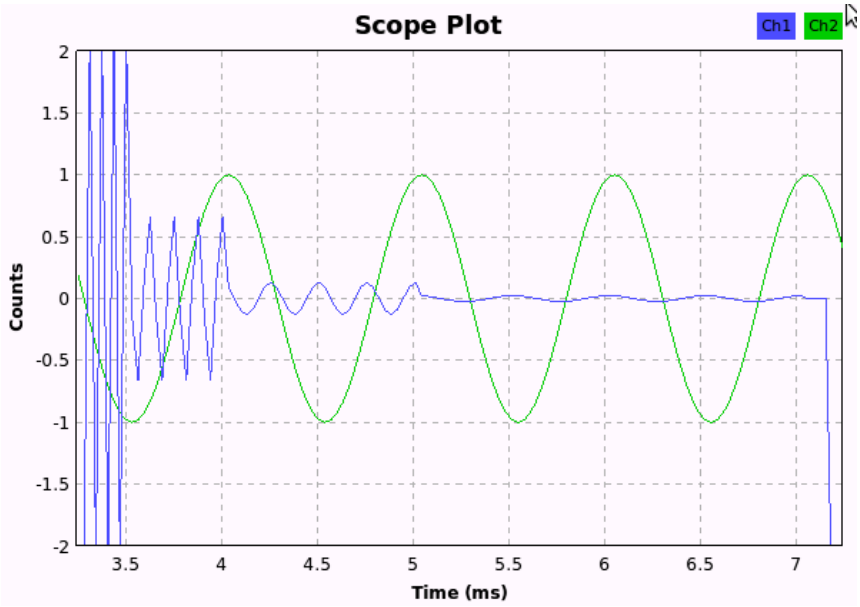

Figura 5.5 – Resultado do teste da Wavelet Daubechies no GRC.

Pode ser visto na Figura 5.5 que o sinal do canal 1 é um sinal decomposto em A3, D3, D2 e D1, assim como ilustra a árvore de decomposição da Figura 5.1. Já o sinal do canal 2 mostra que foi executado a operação inversa da transformada e, portanto, obtido o sinal cossenoidal tal como o sinal de entrada.

Um segundo teste é necessário para verificar se a decomposição realizada por meio do bloco *Wavelet* criado no GRC está processando de maneira correta, para esta verificação foi feito o mesmo processamento utilizando o MATLAB. Assim, como no GRC o sinal de entrada é uma fonte cossenoidal e a decomposição do sinal de saída, obtido do MATLAB, pode ser visto na Figura 5.6.

Assim, como o sinal do canal 1 mostrado na Figura 5.5, o sinal obtido pelo MATLAB e mostrado na Figura 5.6 é um sinal decomposto em A3, D3, D2 e D1 similar à árvore de decomposição mostrada na Figura 5.1.

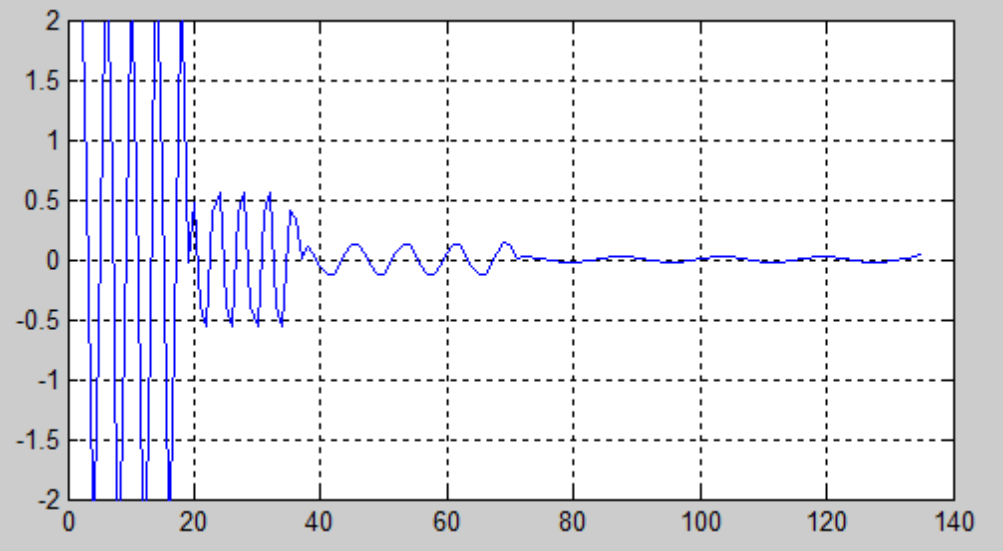

Figura 5.6 – Decomposição Wavelet Db2 pelo MATLAB.

A fim de aplicar o bloco criado neste trabalho, a próxima seção usa diferentes tipos de sinais para fazer o processamento da WT usando o GRC.

5.4 Aplicando o Bloco Wavelet Daubechies no SDR

Em 1987, pela primeira vez foi demonstrado por Mallat que as wavelets constituíam as bases para o processamento e análise de sinais utilizando a técnica de multirresolução, ou seja, representa o sinal processado em diversas resoluções, podendo extrair ou tornar perceptível características que no sinal original não é perceptível.

O bloco denominado de '*Wavelet'* abordado neste trabalho é um bloco de uma wavelet discreta que utiliza a técnica de multirresolução e contempla a família Daubechies unidimensional. A função do bloco '*Wavelet*' é decompor qualquer sinal em níveis que podem ser determinados de acordo com a aplicação que se queira.

Duas aplicações com tipos de sinais diferentes foram realizadas neste trabalho, a primeira utiliza sinais biomédicos e a segunda usa sinais de voz.

5.4.1 Aplicação com Sinais Cardíacos

O sinal de entrada usado no GRC foi um sinal de eletrocardiograma (ECG) com arritmia. O sinal ECG é obtido durante a atividade do músculo cardíaco e possui curvas com características bem definidas representadas pelas letras P, Q, R, S e T. Cada letra traduz uma atividade do coração, a onda P representa a contração atrial do coração, complexo QRS a contração ventricular do coração e a onda T traduz o retorno da massa ventricular ao estado de repouso elétrico. A forma de onda de um ECG normal pode ser visto na Figura 5.7 [26].

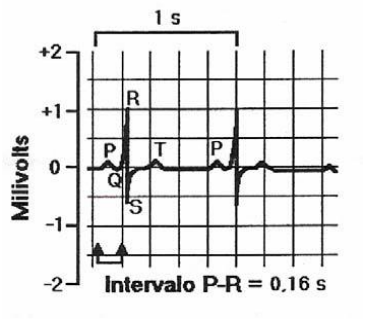

Figura 5.7 – Registro de um eletrocardiograma normal.

O exame ECG auxilia na detecção de doenças cardíacas. Dentre elas podem ser detectadas as arritmias cardíacas, que são alterações no ritmo cardíaco normal, ou seja, o coração bate numa velocidade diferente do padrão normal. Sinais de ECG com arritmia foram coletados da base de dados do *Massachutsetts Institute of Technology (MIT)* que juntamente com *Boston's Beth Israel Hospital (BHI)* formam um banco de dados com mais de 40 registros de ECG's que são disponibilizados gratuitamente e online [27]. O sinal ECG com arritmia usado no processamento é mostrado na Figura 5.8.

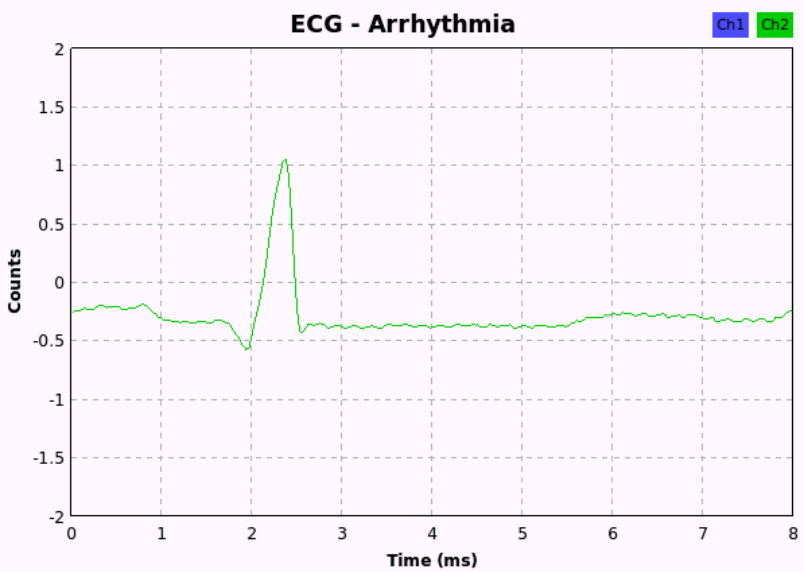

Figura 5.8 – ECG com arritmia.

Foi desenvolvido no GRC a sequência de blocos mostrado na Figura 5.9, esse conjunto processa o sinal de ECG com arritmia usando a WT Daubechies de ordem 2.

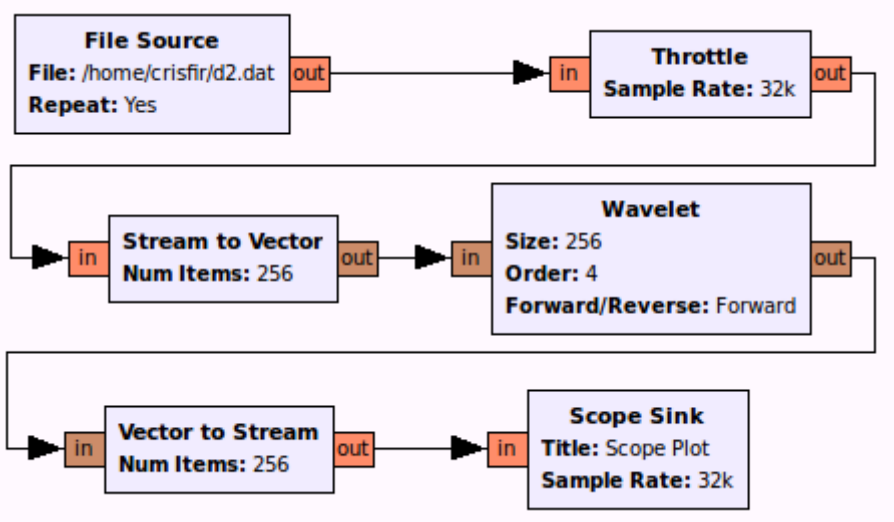

Figura 5.9 – Diagrama de blocos para WT DB2.

O arquivo do ECG com arritmia é colocado no primeiro bloco, *File Source*, e após o processamento pela WT o sinal é mostrado no osciloscópio virtual do GRC, *Scope Sink*. Os parâmetros utilizados para este teste foram *size* = 256, *order* = 4 e *forward*, portanto foi obtido a Db2. A Db2 com 7 níveis resulta em uma janela de aproximação, e 6 janelas de detalhes. O sinal de

saída do processamento é um vetor com todas as janelas mostradas em um único gráfico, para uma melhor visualização foi capturado a aproximação e cada detalhe em janelas separadas, a árvore de decomposição DB2 e suas respectivas janelas são mostradas nas Figuras 5.4.4.

A reconstrução do sinal ECG com arritmia, para obtenção do sinal filtrado, utiliza o mesmo fluxo, porém com os parâmetros alterados, para *size = 128, order = 4 e reverse*. Os resultados obtidos nos testes com os sinais cardíacos são mostrado no próximo capítulo.

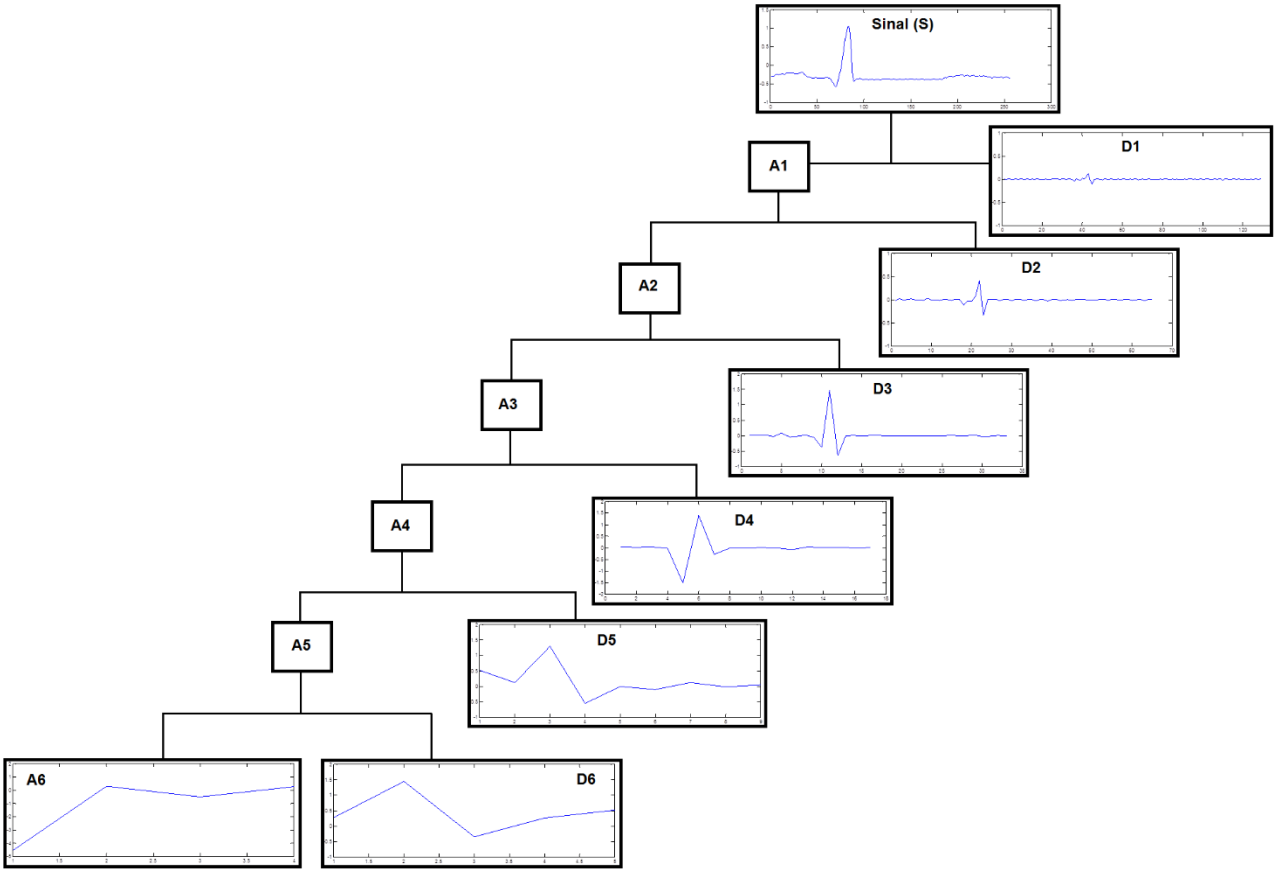

Figura 5.10 – Árvore de decomposição Db2.

O próximo item apresenta testes que também utilizam o bloco Wavelet e o GNU Radio, entretanto, o sinal de entrada é um sinal de voz que é processado para obtenção da compressão de palavras.

5.4.2 Aplicações com Sinais de Voz

Uma característica comum de sinais gerados por fontes físicas é que, em sua forma natural, eles contêm uma quantidade significativa de informação redundante, cuja transmissão, portanto, desperdiça importantes recursos de comunicação. Para uma transmissão eficiente de sinal, deve-se eliminar do sinal informações redundantes antes da transmissão. Essa operação, com nenhuma perda de informação, comumente é realizada no sinal na forma digital, e nesse caso a denominamos compactação de dados ou compressão de dados sem perdas [14].

Nos sinais de voz as componentes de baixa frequência contem maior informação e nas componentes de alta frequência são encontrados detalhes do sinal que se removidos alteram o som, mas ainda é possível compreender a mensagem, já se a remoção de dados acontecer nas baixas frequências a compreensão da informação é afetada.

O método de compactação usado neste trabalho foi do tipo multirresolução usando a Transformada Wavelet criada no GRC. O Sinal de voz foi capturado e gravado a partir de um microfone para que o mesmo pudesse ser processado utilizando todas as ordens da família Daubechies. O sinal capturado é mostrado na Figura 5.11 e contém a palavra *"teste"*.

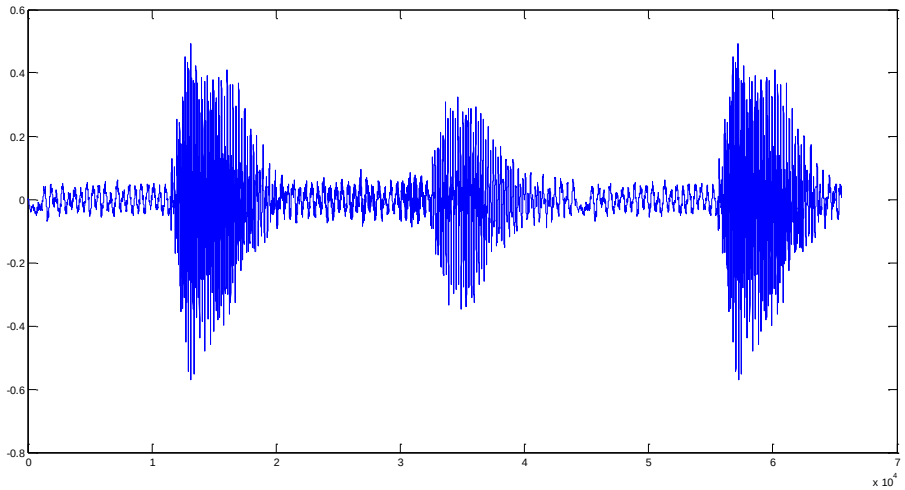

Figura 5.11 – Sinal de entrada com 65536 amostras.

A partir do sinal de entrada mostrado na Figura 5.11 foi feito o processamento utilizando a WT direta de Db2 a Db10, em todos os processamentos a parte de detalhes, D1, que possui 32768 amostras foi retirado antes da reconstrução do sinal pela WT inversa. A Figura 5.12 mostra a decomposição do sinal de voz.

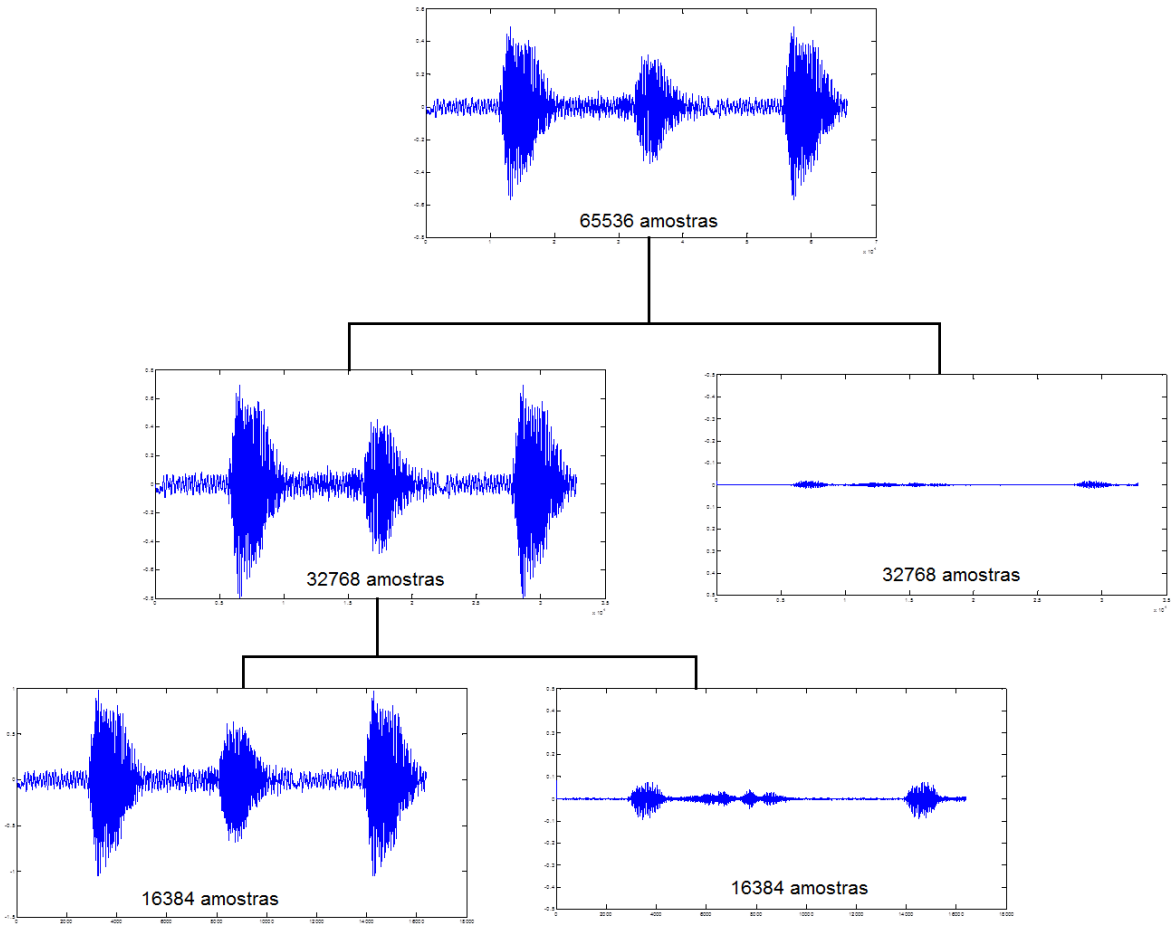

Figura 5.12 – Decomposição do sinal de voz.

Como pode ser visto na Figura 5.12 o sinal amostrado passa por um filtro passa baixa, que é a componente de aproximação, e por um filtro passa alta, que é a componente de detalhe, em cada nível acontece essa filtragem. Para a compactação, desenvolvida neste trabalho, a componente de detalhe do primeiro nível de cada ordem de Daubechies são descartadas, o que resulta em um sinal com a metade do número de amostras do sinal original. Então é realizada a recomposição do sinal

com a mesma ordem de decomposição inicial. Os resultados dos testes são apresentados no próximo capítulo.

5.5 Considerações Finais Deste Capítulo

Neste capítulo foram apresentados os fundamentos da Transformada Wavelet Daubechies, foram realizados testes utilizando o GNU Radio para executar a WT, e, ainda, este capítulo mostrou duas aplicações com sinais distintos utilizando a mesma ferramenta e o mesmo processamento.

O capítulo 6 mostra os resultados obtidos nos testes e aplicações realizados. E finalmente, são realizadas conclusões sobre esses resultados.

CAPÍTULO 6

6 RESULTADOS

6.1 Introdução

Este capítulo apresenta os resultados obtidos neste trabalho. Ele mostrada os espectros dos sinais do receptor AM, do transmissor FM e os espectros obtidos de um analisador de espectro portátil. Além disso, este capítulo mostra os resultados obtidos na decomposição do sinal cardíaco e a filtragem do mesmo. E, ainda, apresenta os resultados da compactação do sinal de voz por meio da Transformada Wavelet Daubechies.

6.2 Análise Espectral do Receptor

O sinal AM é recebido pela antena e absorvido pelo sistema por meio da placa filha LFRX do USRP. A partir do fluxo do receptor AM, que foi mostrado na Figura 3.11, para auxiliar a compreensão da Figura 3.11 foi criado o diagrama de blocos mostrado na Figura 6.1.

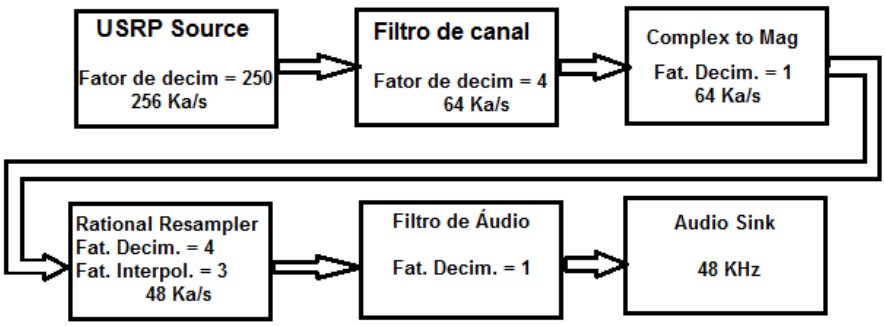

Figura 6.1 – Diagrama de blocos do receptor AM implementado no SDR.

Na Figura 6.1 o primeiro bloco, *USRP Source*, é uma fonte com sinais oriundos do módulo USRP. O sinal obtido da USRP possui 64 Ma/s assim já no primeiro bloco aplica um fator de decimação de 250, resultando em 256 Ka/s. Um misturador é aplicado juntamente com o sinal de entrada para que a estação desejada seja transladada pra banda base. O filtro de canal é aplicado para filtrar apenas uma estação AM, o fator de decimação do filtro é de 4, resultando em 64 Ka/s. O bloco *complex to Mag* é usado para calcular a informação do módulo do valor complexo, pois até então foram trabalhadas apenas com informações complexas. O bloco *rational Resampler* é usado para fazer operações de decimação e interpolação para que o sinal de saída tenha 48 Ka/s. Um filtro de áudio é aplicado a fim de eliminar alguns ruídos. O bloco *Audio Sink* tem função de enviar o sinal diretamente para a placa de som do computador.

Tanto o filtro de canal quanto o filtro de áudio são filtros passa baixa com frequência de corte de 5 KHz.

Um analisador de espectro foi posicionado logo após a translação do sinal para a banda base. O sinal obtido pelo analisador de espectro virtual do GRC é mostrado na Figura 6.2.

Na Figura 6.2 pode ser visto diversos picos, cada um deles indica uma estação de rádio AM. O sinal com o pico em 0 KHz é o que possui maior potência, e é o que se deseja demodular, este sinal representa o sinal da rádio AM de frequência 1020 KHz.

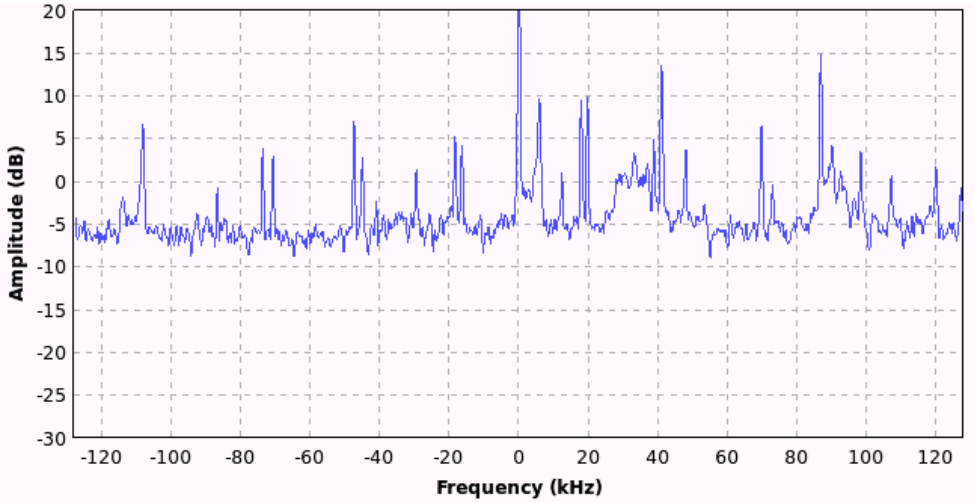

Figura 6.2 – Espectro do sinal AM recebido a 1020 kHz.

Algumas características, como o ajuste de volume e o controle na eliminação de ruídos, são evidências do quão flexível é o uso do SDR para aplicações como esta. Observando esta aplicação em um contexto acadêmico, pode se concluir que o SDR se diferencia de qualquer outro equipamento de análise de sinais, pois é possível tanto controlar quanto analisar os sinais em diferentes etapas do processamento.

O receptor AM desenvolvido neste trabalho apresentou resultados satisfatórios, não somente no que diz respeito à sintonia da estação desejada, mas também no controle e na análise do sinal.

6.3 Análise espectral do Transmissor

Após desenvolvido o transmissor WBFM no SDR alguns espectros da Transformada Rápida de Fourier, foram obtido de forma estratégica para análise em cada ponto. A Figura 6.4 [15] ilustra alguns pontos de captura da FFT obtida por meio do analisador de espectro virtual do GRC.

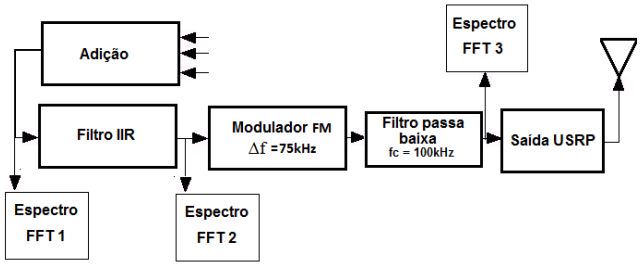

Figura 6.4 – Pontos de captura dos espectros FFT 1, FFT 2, FFT 3.

Pode-se observar na Figura 6.5que o espectro do sinal de áudio L+R em banda base até a frequência de 15 kHz, o sinal piloto em 19 kHz e o sinal L-R de 23 kHz a 53 kHz centrado em 38 kHz. Esse espectro está relacionado com o espectro teórico apresentado na Figura 3.19.

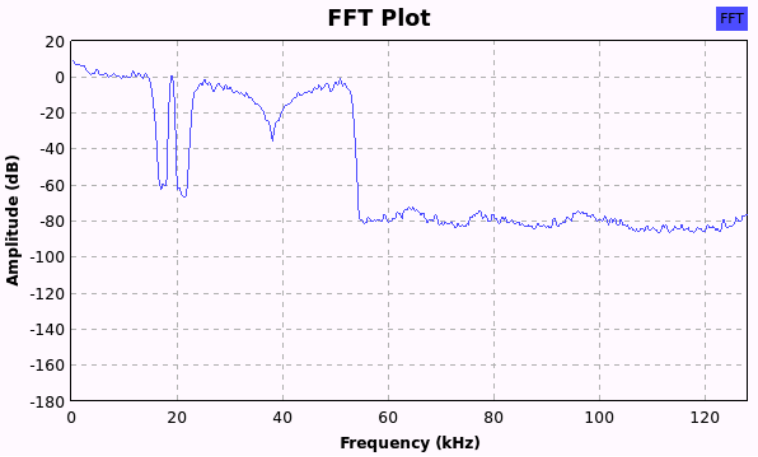

Figura 6.5 – Espectro FFT 1.

A Figura 6.6 mostra o espectro do sinal em banda base após a terceira filtragem de pré-ênfase, em que se observa o ganho a partir de 2,12 kHz, valor igual ao mostrado na curva de resposta mostrado na Figura 3.18 (b).

A inserção do terceiro filtro IIR que antecede o modulador FM foi necessária após a identificação da presença de ruído no áudio recebido por um receptor FM estéreo comum. Com o terceiro filtro IIR ocorreu uma melhora significativa na qualidade do áudio recebido, além de permitir a identificação do efeito da pré-ênfase no espectro.

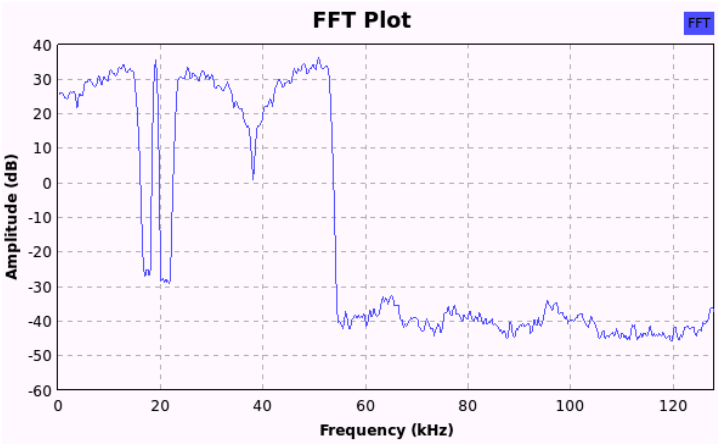

Figura 6.6 – Espectro FFT 2.

A Figura 6.7 mostra o espectro do sinal em banda base da Figura 6.6 modulado em frequência, após a filtragem passa baixa com frequência de corte em 100 kHz para manter a largura de banda total do espectro em 200 kHz.

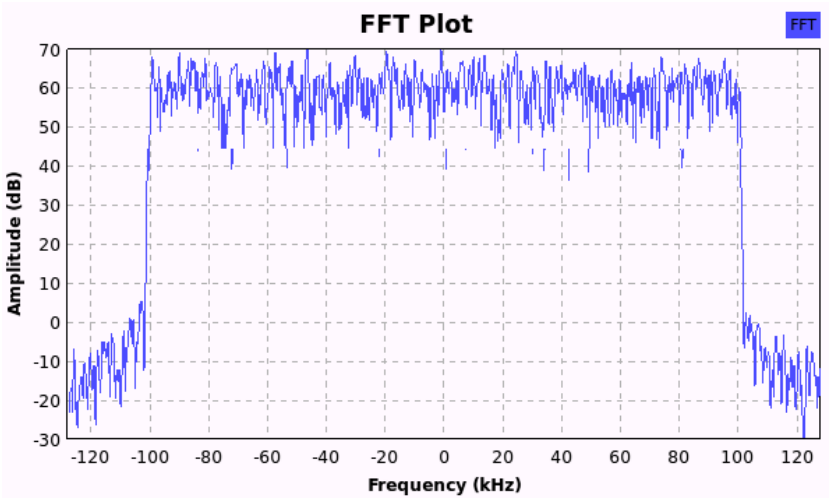

Figura 6.7 – Espectro FFT 3.

Para a transmissão foi utilizada a placa filha BasicTX, com faixa de frequência de 1 a 250 MHz e potência máxima de saída de 0.5 mW. O sinal resultante foi irradiado ao meio externo por meio de uma antena acoplada a uma saída de RF do módulo USRP, após aplicar um ganho de 20 dB.

De posse de um receptor FM estéreo comum, obteve-se sucesso na recepção do sinal transmitido localmente, em que foi possível ouvir claramente o áudio com a mesma qualidade quando comparada com a reprodução desse mesmo arquivo em um dispositivo compatível, como um MP3 player.

O analisador de espectro portátil, desenvolvido pela Agilent Technologies do modelo N9912A, foi utilizado para receber o sinal WBFM transmitido pelo SDR, o sinal recebido pelo analisador de espectro portátil foi capturado e é apresentado na Figura 6.8. O espectro é centrado na frequência de 103,5 MHz.

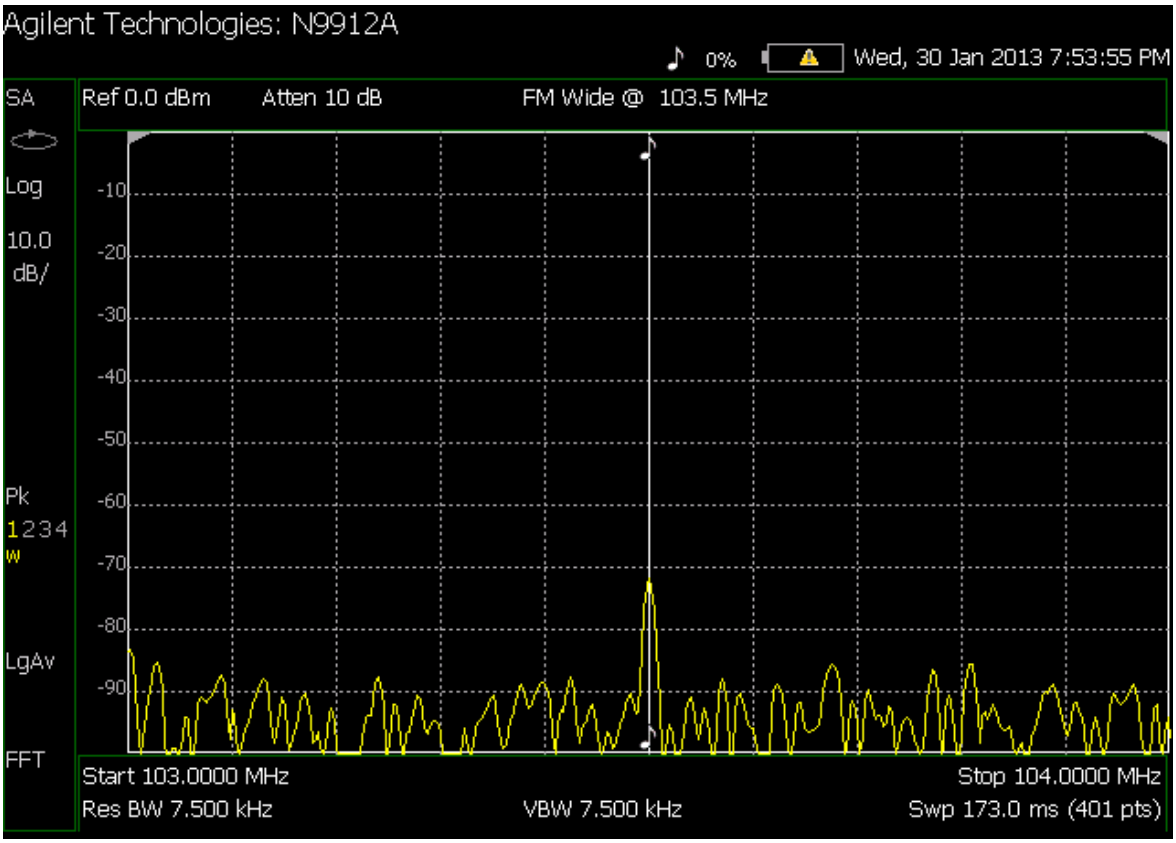

Figura 6.8 – Espectro do sinal FM recebido pelo analisador de espectro portátil.

A transmissão WBFM por meio do conjunto SDR se mostrou satisfatória e vantajosa por possibilitar a análise espectral em partes estratégicas do transmissor, sendo esta última significativa para o uso do SDR como ferramenta de auxílio ao aprendizado.

6.4 Decomposição e Filtragem do Sinal Cardíaco

O sinal de entrada usado para decomposição e filtragem é um sinal de ECG com arritmia, que foi coletado da base dados do *Massachutsetts Institute of Technology – MIT* [27]. A Figura 6.9 mostra o sinal de entrada.

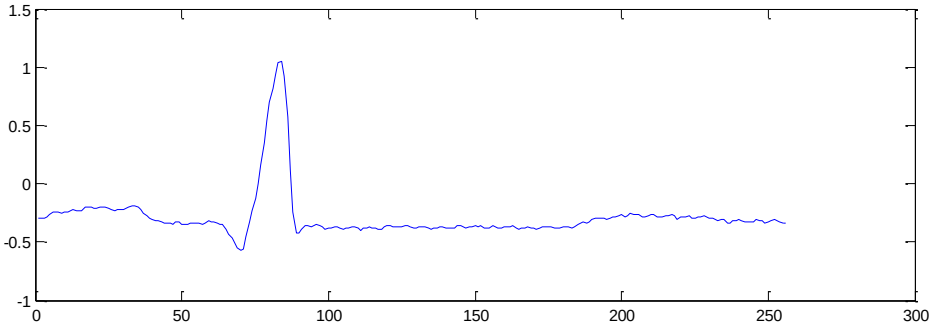

Figura 6.9 – Sinal ECG com arritmia.

O fluxo usado para o processamento do sinal ECG foi mostrado na Figura 5.9, de acordo com os parâmetros utilizados no bloco *Wavelet*, o sinal de saída é uma decomposição que resulta em uma aproximação e seis detalhes. A cada decomposição o número de amostras é distribuído em quantidades iguais para o detalhe e a aproximação. As Figuras 6.10 a 6.16 mostram as janelas em cada nível de decomposição para a Db2.

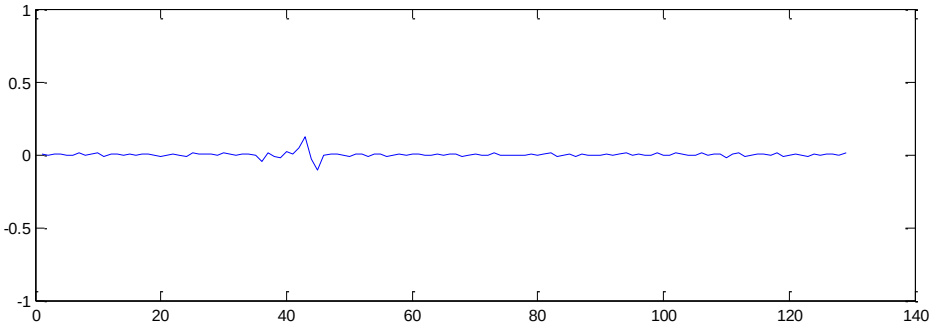

Figura 6.10 – Detalhe, D1, com 128 amostras.

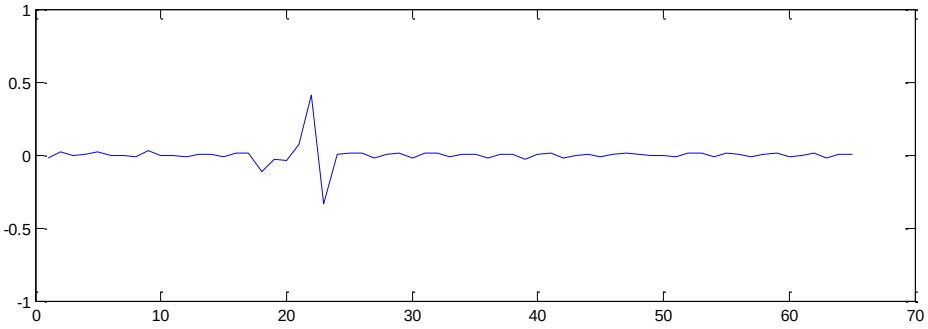

Figura 6.11 – Detalhe, D2, com 64 amostras.

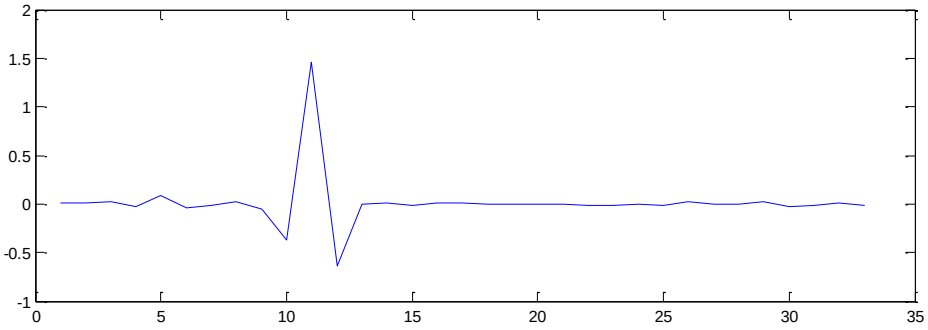

Figura 6.12 – Detlahe, D3, com 32 amostras.

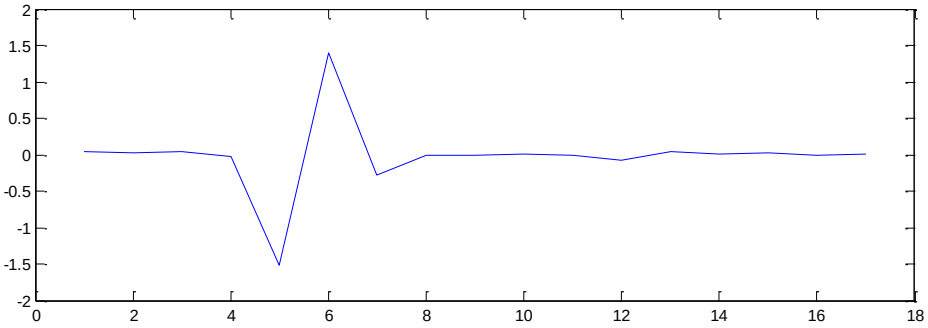

Figura 6.13 – Detalhe, D4, com 16 amostras.

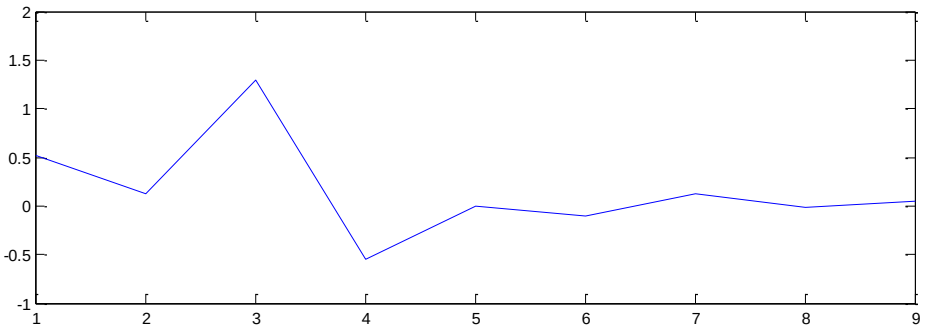

Figura 6.14 – Detalhe, D5, com 8 amostras.

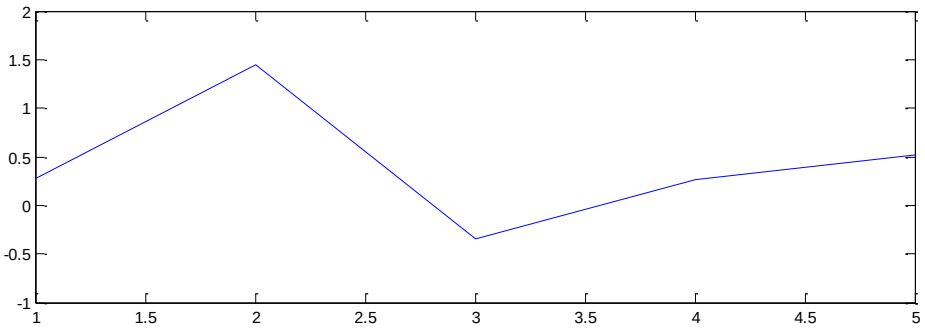

Figura 6.15 – Detalhe, D6, com 4 amostras.

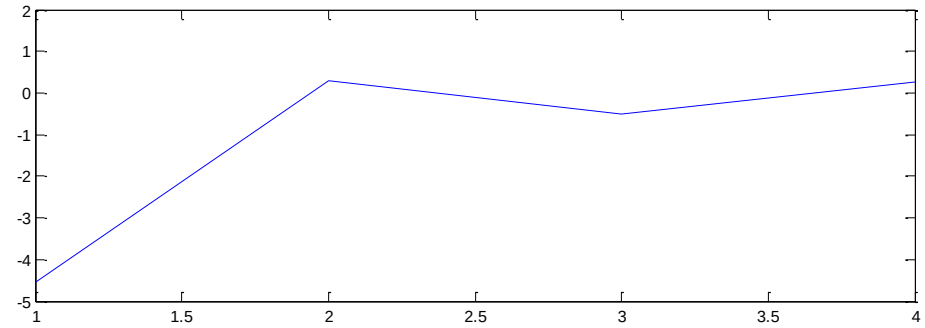

Figura 6.16 – Aproximação, A6, com 4 amostras.

Assim, como mostrado na árvore de decomposição da Figura 5.10 o número de amostras em cada janela das Figuras 6.10 a 6.16 vão diminuindo até chegar em 4 amostras como mostradas nas componentes de detalhe D6 e aproximação A6.

O nível de detalhe mostrado na Figura 6.10 possui potência muito pequena com relação ao sinal original. Como o objetivo é filtrar o sinal cardíaco, o sinal D1 pode ser extraído do sinal original e o sinal filtrado é equivalente ao sinal A1 da árvore de decomposição mostrada na Figura 5.10. Portanto, após o processamento utilizando a Transformada Wavelet Daubechies de ordem 2 o sinal filtrado A1 é mostrado na Figura 6.17.

Figura 6.17 – Sinal ECG após processamento com Db2.

A Figura 6.18 mostra o sinal de entrada, com 256 amostras, e o sinal de saída, com 128 amostras, após o processamento por meio da WT Db2. O sinal de saída é o sinal filtrado pelo filtro passa baixa.

Para uma visualização macro da Figura 6.18 o sinal de entrada equivale ao sinal S e o sinal de saída equivale ao sinal A1 da árvore de decomposição mostrado na Figura 5.10.

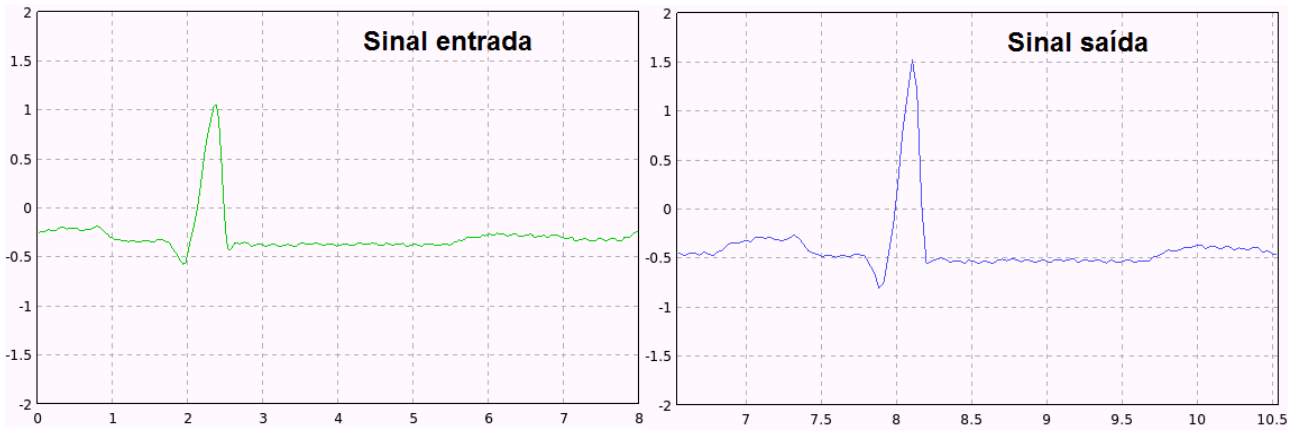

Figura 6.18 – Sinais de entrada e saída após Db2.

Os testes realizados com sinais cardíacos apresentaram resultados satisfatórios para a filtragem do sinal, e ainda mostrou que não somente o GNU Radio, mas o conjunto SDR pode ser uma ferramenta potencialmente poderosa na análise de sinais de natureza biológica.

6.5 Compactação do Sinal de Voz

O sinal de voz da palavra “teste” foi capturado utilizando uma amostragem de 44100 amostras. Para seguir os parâmetros de entrada do bloco wavelet foi necessário repetir o sinal e corta-lo em 65536 amostras, (2^{16}). A Figura 6.19 mostra o fluxo do processo de compactação do sinal de voz. A Figura 6.20 mostra o sinal de entrada.

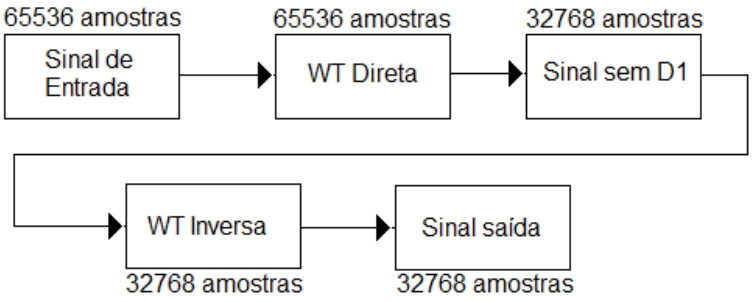

Figura 6.19 – Diagrama de blocos do processo de compactação do sinal de voz.

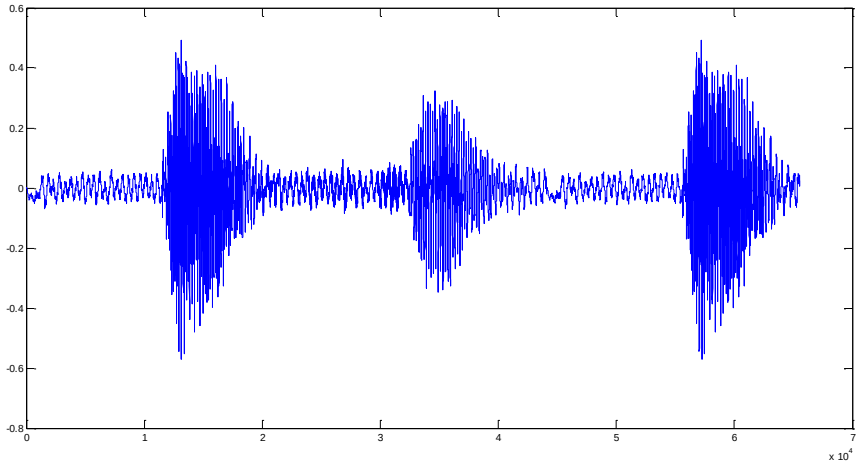

Figura 6.20 – Sinal de entrada com 65536 amostras.

O sinal foi processado utilizando a WT Daubechies de ordem Db2 a Db10, após este processamento o primeiro nível de detalhes que possui 32768 amostras foi descartado para cada ordem. As Figuras 6.21 a 6.29 mostram as primeiras 32768 amostras do processamento que representa o conjunto de aproximação e detalhes sem o primeiro nível de detalhe.

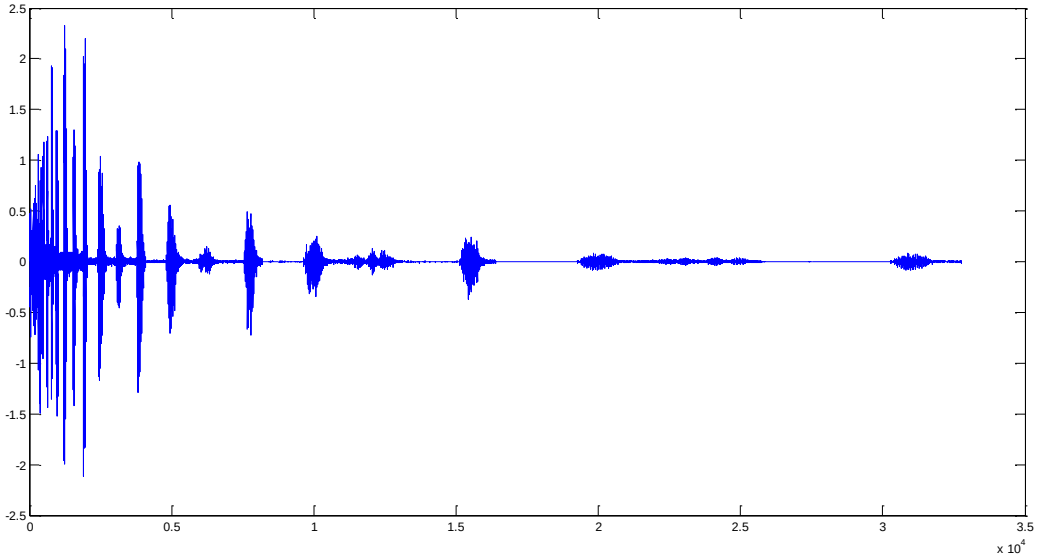

Figura 6.21 – 32768 amostras da WT Db2.

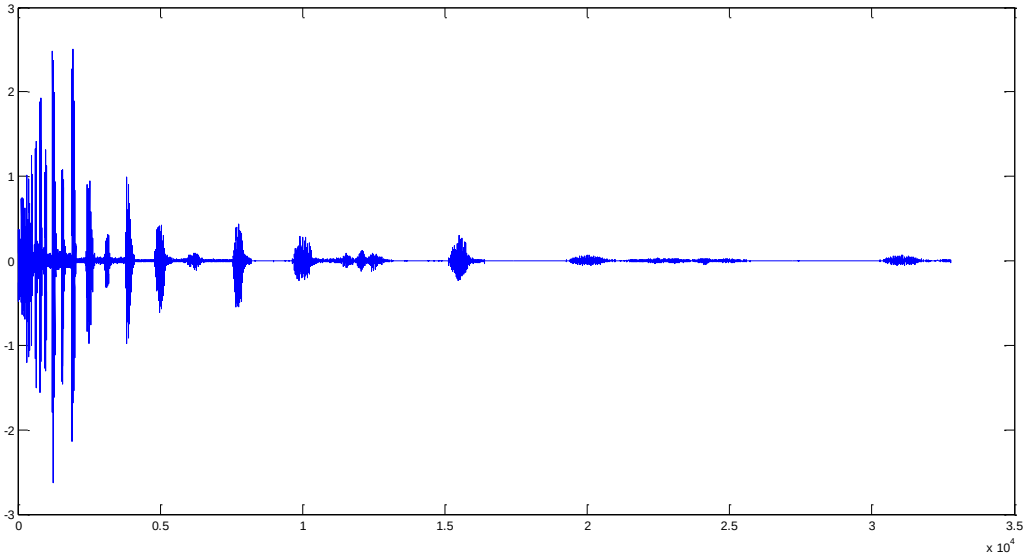

Figura 6.22 – 32768 amostras da WT Db3.

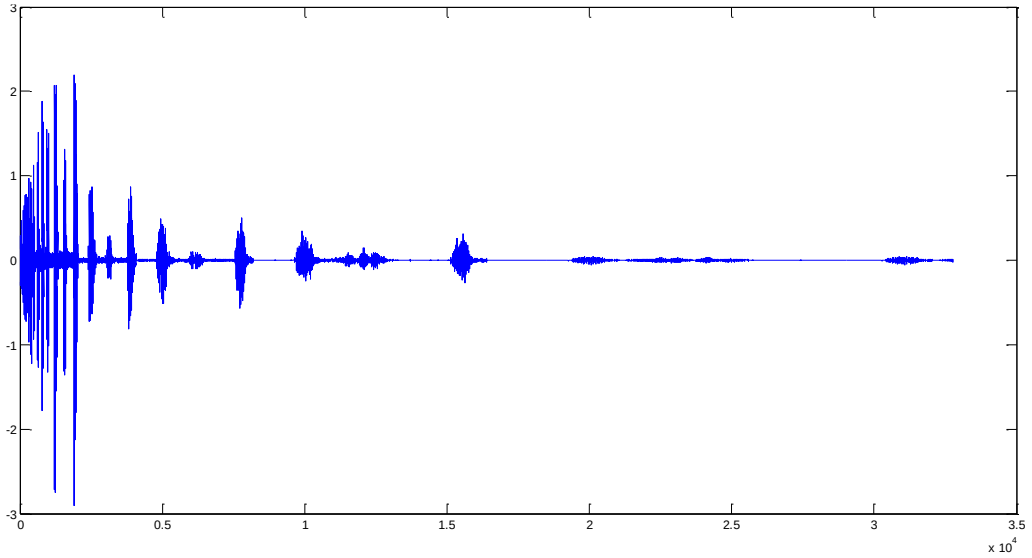

Figura 6.23 – 32768 amostras da WT Db4.

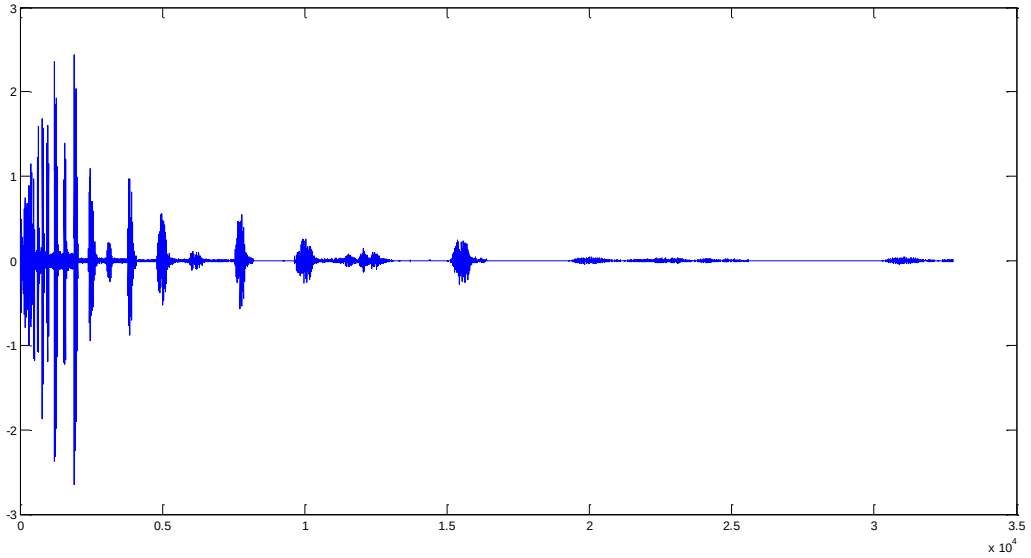

Figura 6.24 – 32768 amostras da WT Db5.

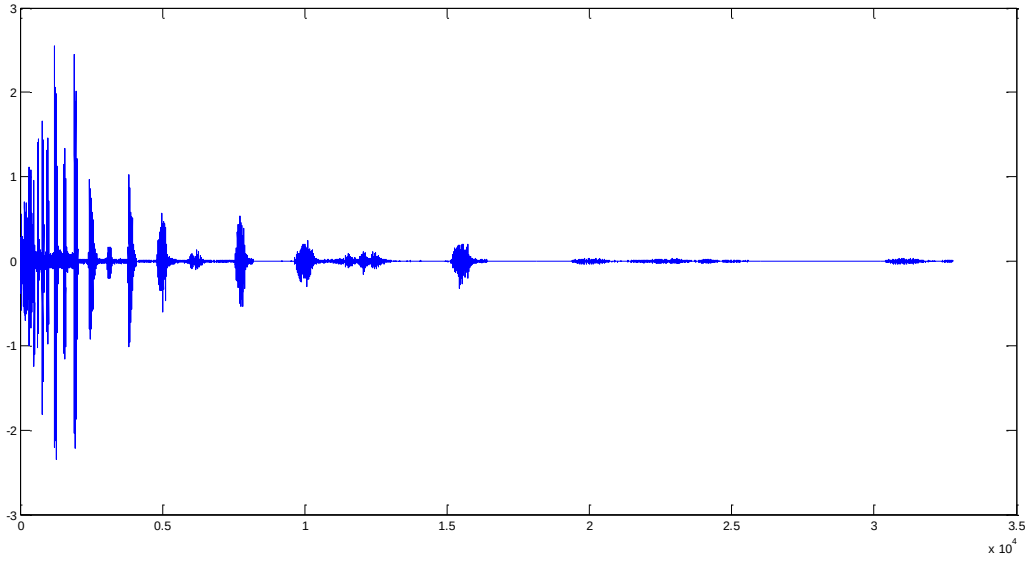

Figura 6.25 – 32768 amostras da WT Db6.

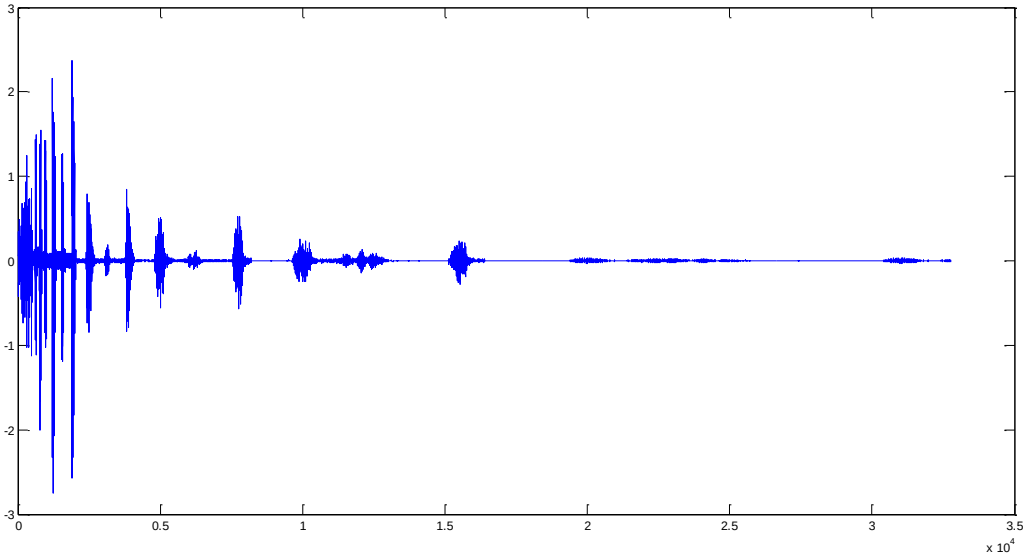

Figura 6.26 – 32768 amostras da WT Db7.

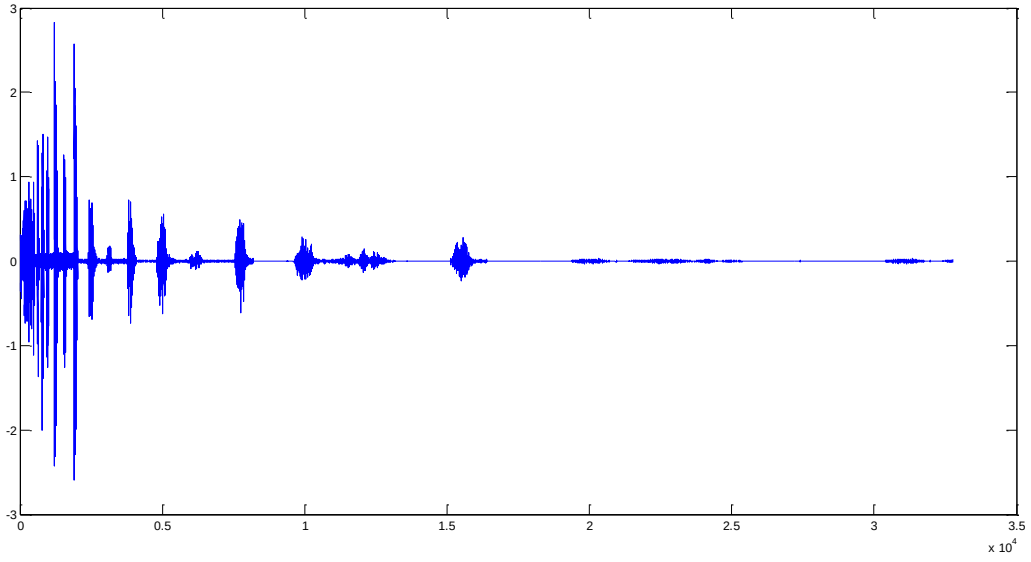

Figura 6.27 – 32768 amostras da WT Db8.

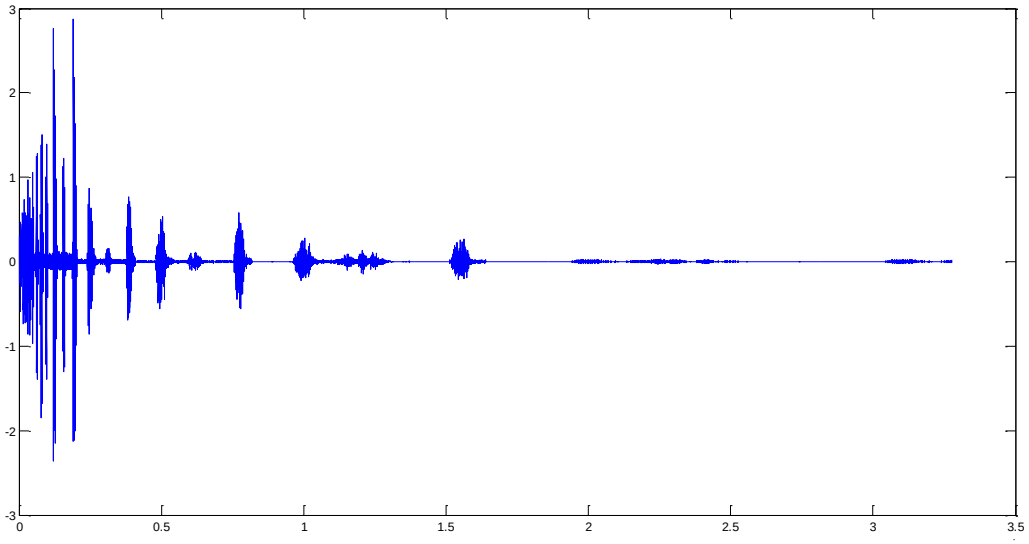

Figura 6.28 – 32768 amostras da WT Db9.

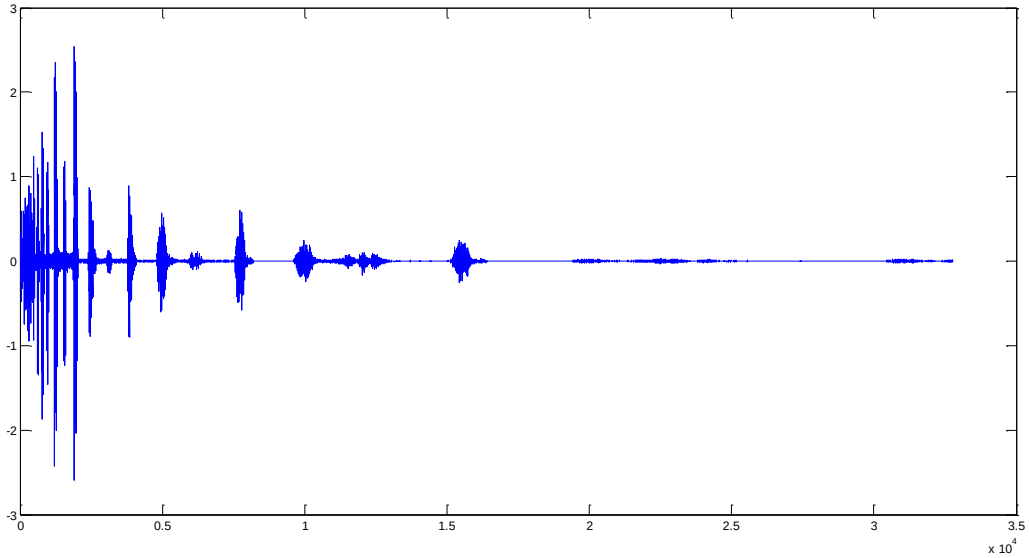

Figura 6.29 – 32768 amostras da WT Db10.

Após a reconstrução do sinal com 32768 amostras por meio da WT inversa em todas as ordens da Daubechies, o resultado foi satisfatório e o sinal de saída praticamente igual. A Figura 6.30 mostra o sinal da reconstrução Db2 em vermelho e o sinal da reconstrução Db10 em azul.

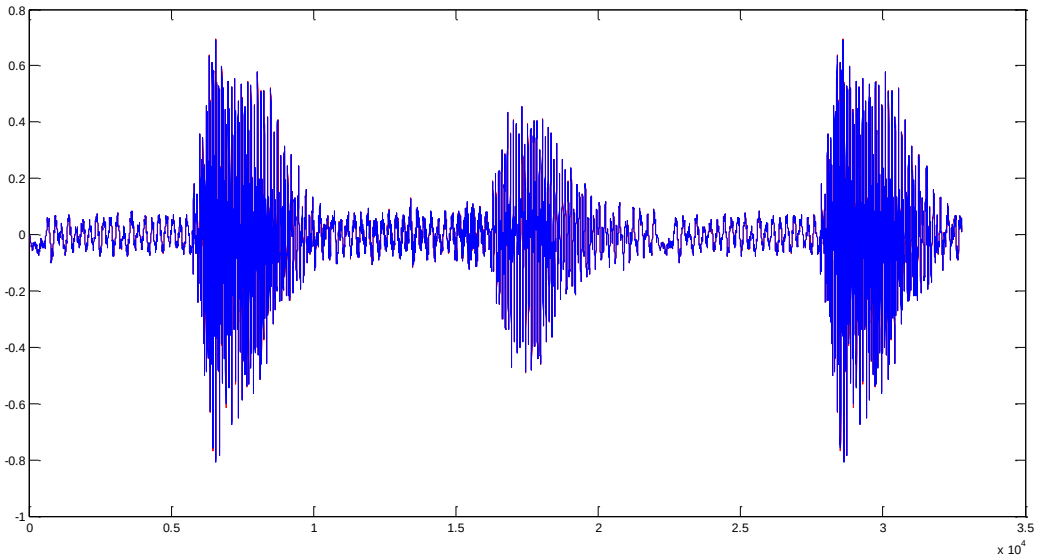

Figura 6.30 – Sinal de Saída Db2 em vermelho e Db10 em azul.

Observando a Figura 6.30 tanto o resultado obtido pela Db2 quanto o resultado obtido pela Db10 são bem próximos, para comprovar, optou-se pelo cálculo da diferença do sinal obtido pela Db10 e o sinal da Db2, resultando no sinal mostrado na Figura 6.31.

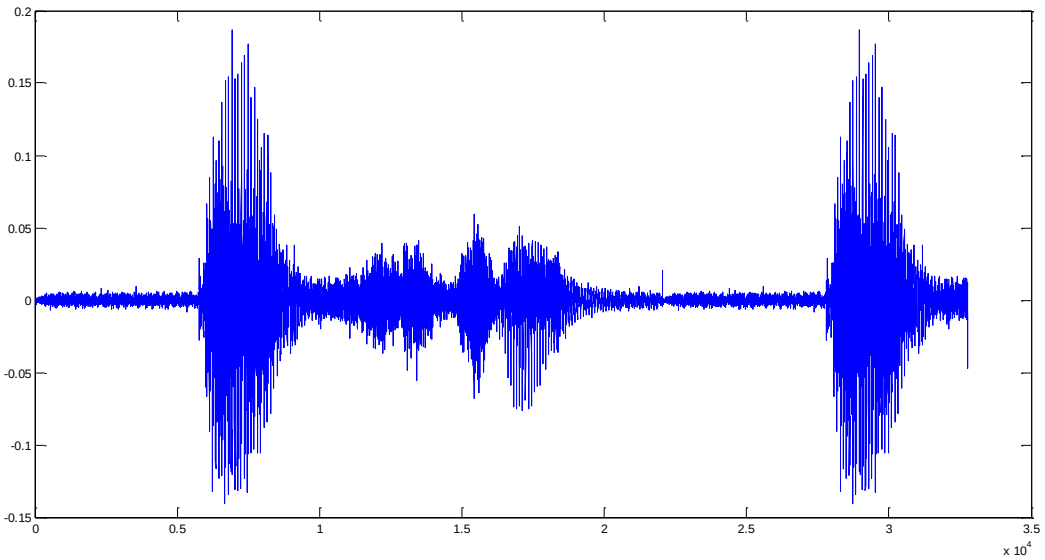

Figura 6.31 – Resultado da diferença do sinal da Db10 pela Db2.

A partir da análise da Figura 6.31 é possível concluir que o sinal “erro” entre a Db2 e a Db10 é bem pequeno. Para uma melhor visualização é mostrado na Figura 6.32 os três sinais em uma mesma escala.

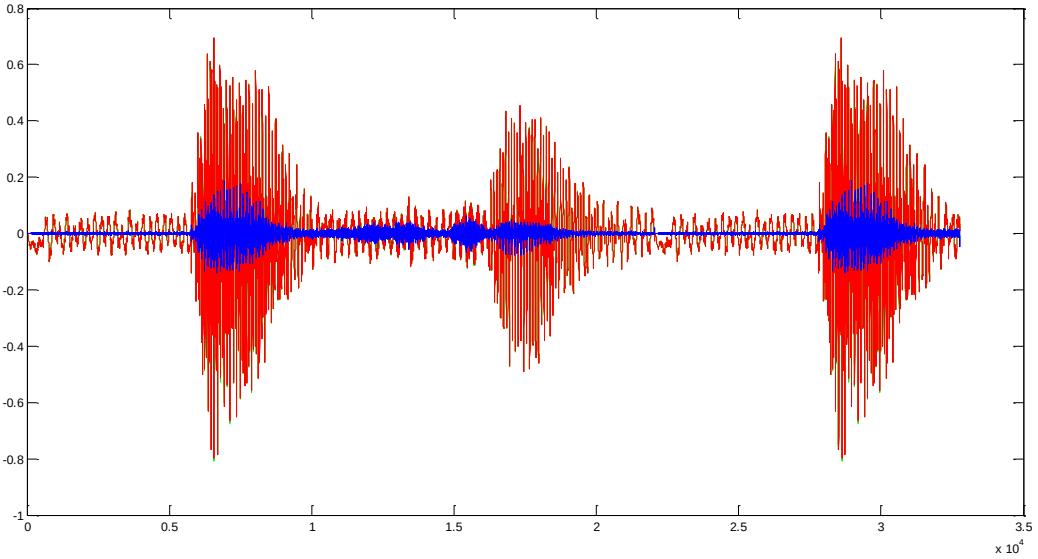

Figura 6.32 – Db2 em vermelho, Db10 em verde, Db10-Db2 em azul.

O sinal de entrada após o processamento da WT Daubechies resulta em dois sinais, um equivale ao sinal filtrado por um filtro passa baixa e outro equivalente ao sinal filtrado por um filtro passa alta como mostrado na Figura 5.3.

Usando todas as ordens da família Daubechies foram obtidos nove resultados de sinais filtrados pelo filtro passa baixa. Esses sinais são oriundos da reconstrução dos sinais mostrados nas Figuras 6.21 a 6.29.

De posse dos sinais compactados uma primeira análise para verificar a qualidade da informação foi feita por meio do áudio. Utilizando o ouvido humano não foi possível detectar nenhuma diferença entre as saídas obtidas do processamento da Db2 a Db10.

Uma segunda etapa de análise foi feita utilizando a comparação gráfica dos resultados. Assim, verificou-se que para cada ordem da WT existe uma diferença entre os sinais, e a que apresentou o maior “erro” foi a Db2 com relação a Db10, mostrado na Figura 6.32.

Depois das duas análises, pode-se concluir que, considerando o sinal de entrada e os resultados de Db2 a Db10, o conteúdo da informação original não apresentou perdas significativas após a compressão da informação. Portanto, o uso do SDR se mostrou eficiente para a compactação de sinais de voz.

6.6 Conclusões

O receptor AM utilizando GNU Radio e o USRP funcionou adequadamente e se mostrou eficiente para a compreensão do processo de demodulação, a fim de mostrar o espectro do sinal em pontos estratégicos do demodulador. Por utilizar *software* para a construção do receptor, não ocorreu descarte de peças, componentes, placas e equipamentos como ocorreria se a construção fosse a tradicional.

O transmissor WBFM utilizando GNU Radio e o USRP foi eficiente em seu funcionamento. O uso do SDR para aplicações como esta se mostrou uma ferramenta muito didática, possibilitando o detalhamento de todo o processo de modulação, e ainda permitiu a obtenção dos espectros em partes estratégicas do projeto para comparação com espectros teóricos.

Os testes para filtragem de sinais biológicos utilizando a decomposição por multirresolução se mostrou satisfatório e ainda permitiu uma abordagem para utilização do conjunto SDR ainda inexplorada na literatura.

Os testes de compressão do sinal foram feitos com toda a família Daubechies, e os resultados obtidos utilizando o GNU Radio se mostrou eficiente e apresentou compressão de 50% da mensagem original.

O próximo capítulo apresenta as conclusões, as contribuições deste trabalho e os trabalhos futuros que poderão ser desenvolvidos a partir desta dissertação.

CAPÍTULO7

7 CONCLUSÕES ETRABALHOS FUTUROS

7.1 Introdução

A tecnologia de rádio definido por software está induzindo uma revolução nos conceitos de dispositivos para processamento de sinais, com uma arquitetura de rádio flexível que permite a mudança da função do rádio, possivelmente, em tempo real, e, com um processo que garante uma qualidade de serviço.

Este trabalho foi motivado pelo baixo custo do conjunto SDR, *hardware + software*, e por comporem uma ferramenta altamente flexível capaz de processar qualquer tipo de sinal em uma ampla faixa de frequência, permitindo que uma nova proposta para equipar laboratórios fosse apresentada neste trabalho.

O objetivo deste trabalho foi apresentar os conceitos de SDR, aplicar algumas práticas utilizando USRP e GNU Radio para mostrar que este conjunto é uma ferramenta poderosa no auxílio ao ensino superior em engenharia. E, ainda, para garantir que o conjunto seja uma ferramenta muito flexível, foi mostrado neste trabalho, a criação de um bloco de processamento multirresolução para filtragem e compressão de sinais de diferentes naturezas, utilizando a Transformada Wavelet Daubechies.

Este capítulo apresenta as conclusões, em seguida são mencionadas as contribuições acadêmicas deste trabalho, e por fim são mostradas sugestões para os trabalhos futuros que poderão ser desenvolvidos a partir desta dissertação.

7.2 Conclusões

Este trabalho permitiu mostrar alguns conceitos fundamentais de rádio definido por software, descreveu especificações do *hardware* USRP e apresentou estudos do *software* GNU Radio. E ainda, mostrou algumas aplicações que podem ser utilizadas como práticas de laboratório no ensino superior. Após descrever a estrutura da árvore de construção do gnuradio e da criação do bloco wavelet no GRC, foi realizada uma abordagem dos conceitos teóricos da Transformada Wavelet Daubechies, para, então, desenvolver testes e aplicações utilizando o bloco wavelet, cujos resultados foram apresentados no Capítulo 6 desta dissertação.

O receptor AM, construído utilizando o GNU Radio e o USRP, funcionou adequadamente e se mostrou eficiente para a compreensão do processo de demodulação, a fim de mostrar o espectro do sinal em pontos estratégicos do demodulador, por utilizar *software* para a construção, não ocorreu descarte de peças, componentes, placas e equipamentos como ocorreria se a construção fosse a tradicional.

O transmissor WBFM construído utilizando o GNU Radio e o USRP foi eficiente em seu funcionamento. O uso do SDR para aplicações como esta se mostrou uma ferramenta muito didática, possibilitando o detalhamento de todo o processo de modulação, e ainda permitiu a obtenção dos espectros em partes estratégicas do projeto para comparação com espectros teóricos.

Os testes para filtragem do sinal cardíaco, utilizando o GNU Radio, para decomposição por multirresolução foi adequado e permitiu mostrar que o SDR é um conjunto com um potencial incalculável para processamento e aplicações com sinais de diversas naturezas inclusive a biológica.

Os testes para compressão do sinal foram feitos com toda a família Daubechies. Os resultados obtidos do processamento pelo GNU Radio se mostraram eficientes e apresentaram compressão de 50% da mensagem original.

Todas as aplicações apresentadas neste trabalho mostraram que é possível desenvolver inúmeros ensaios e práticas para aplicar diversas teorias de telecomunicações, bem como, processar

sinais de diferentes naturezas, além disso, mostrou que o uso do SDR é uma ferramenta útil também em ensaios de novos padrões para tecnologias em desenvolvimento, possibilitando construções de projetos de modo prático e simples.

Outra grande vantagem do uso do SDR em práticas laboratoriais é que além de não ser necessária a aquisição de componentes eletrônicos, existe o respaldo ambiental em que a tecnologia se mostra ecologicamente sustentável, não ocorrendo descarte de protótipos mal sucedidos ou qualquer outro resíduo no meio ambiente.

Conclui-se que uso do GNU Radio em conjunto com o módulo USRP permite que as instituições de ensino equipem seus laboratórios com uma ferramenta extremamente poderosa para aplicações práticas de engenharia a um baixo custo.

7.3 Contribuição Deste Trabalho

Até hoje nenhuma ferramenta ou equipamento usado em laboratórios se mostrou tão eficiente e flexível para a compreensão das diversas teorias de telecomunicações. Este trabalho apresentou as facilidades do uso do SDR e mostrou o quão flexível e prático é o uso desta ferramenta, sendo este um conjunto poderoso e muito didático para aplicações em laboratórios acadêmicos.

Esta dissertação buscou contribuir com a construção de aplicações para fim didático e ainda, contribuiu com a criação do bloco de processamento da transformada wavelet para o GRC, facilitando futuras aplicações com o uso deste bloco.

7.4 Trabalhos Futuros

A partir deste trabalho, podem ser citados inúmeros projetos e aplicações para uso como práticas de laboratório, por exemplo, moduladores e demoduladores QAM, DPSK, GMSK, OFDM

dentre outros, o GNU Radio ainda permite análises da constelação do sinal. Para cada aplicação a seleção do tipo de sinal real a ser capturado na entrada depende apenas da escolha adequada da placa filha do USRP.

Para citar aplicações do USRP e do GNU Radio com o bloco da Transformada Wavelet Daubechies criada neste trabalho pode ser considerado o desenvolvimento de um protótipo para reconhecimento de voz utilizando SDR, ou ainda pode ser feito um protótipo, para uso inicialmente em pesquisa, de um aparelho eletrocardiograma para o processamento de sinais cardíacos em tempo real, não obstante, pode ser citado o uso do conjunto em análises de sinais de 60 Hz para identificação do tipo de perturbações no domínio do tempo, bem como a análise e identificação de perturbações no sinal da rede ethernet.

Outros blocos podem ser incluídos no GRC seguindo os mesmos passos, possibilitando disponibilizar outros tipos da transformada wavelet, tanto unidimensional quanto bidimensional, para que trabalhos com processamento de imagens possam ser abordados com o uso do SDR.

BIBLIOGRAFIA

[1] D. Valerio. **Open source software-defined radio: A survey on gnu-radio and its applications**. Forschungszentrum Telekommunikation Wien, Vienna, Technical Report FTW-TR-2008-002, 2008.

[2] Lima A. G. M. **Rádio Definido por Software: O Próximo Salto no Mundo das Telecomunicações e Computação**. Revista Digital Online, 2004.

[3] Jeffrey H. Reed. **Software Radio - A Modern Approach to Radio Engineering**. Prentice Hall, 2002.

[4] D.C. Tucker and G.A. Tagliarini. **Prototyping with gnu radio and the usrp-where to begin**. In Southeastcon, 2009. SOUTHEASTCON'09. IEEE, pages 50-54. IEEE, 2009.

[5] Ettus Research, "**Support: Downloads**" disponível em http://ettus.com/support/downloads. Acessado em 03 de Março de 2012.

[6] Hamza, F. A.. "**The USRP Under 1.5X MagnifyingLens!**" GNURADIO Community, 2008.

[7] Blossom, E. "***Exploring GNU Radio***" disponível em http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html. Acessado em 03 de Março de 2012.

[8] Ettus Research, "**USRP1 Bus Series**" disponível em https://www.ettus.com/content/files/06983_Ettus_USRP1_DS_Flyer_HR.pdf. Acessado em 20 de agosto de 2012.

[9] Mueller, Andreas. DAB: **Software Receiver Implementation**. 2008. 88 f. Dissertação (Mestrado) - Swiss Federal Institute Of Technology Zurichh, Zurique, 2008.

[10] Página oficial do gnuradio "**GNU Radio**", disponível em http://gnuradio.org/redmine/projects/gnuradio/wiki. Acessado em 25 de agosto de 2012.

[11] Página oficial GNU Radio "**GNU Radio 3.3.0 C++ API**", disponível em http://gnuradio.org/doc/doxygen-3.3.0/group__block.html. Acessado em 10 de novembro de 2012.

[12] Página oficial do GNU Radio "**Tutorials**", disponível em http://gnuradio.org/redmine/projects/gnuradio/wiki/TutorialsWritePythonApplications. Acessado em 31 de Dezembro de 2012.

[13] Lathi, B. P. "**Modern Digital and Analog Communication Systems**" 3rd ed. Oxford University Press, 1998.

[14] Simon Haykin. "**Sistemas de Comunicação – Analógico e Digital**" 4ª Ed. Bookman. São Paulo, 2004.

[15] Diniz P. C. A., Rodrigues P. V. F., Rocha G. N., Veiga A. C. P. **Analisador de Espectro para Equipar Laboratórios de Telecomunicações a Baixo Custo Utilizando Rádio Definido por Software.** Conferência de Estudos em Engenharia Elétrica, 2012.

[16] Couch, L. W. II. **Digital and Analog Communication Systems.** 5th ed., Prentice-Hall, 1997.

[17] Página oficial do GNU Radio "**Tutorials**", disponível em http://gnuradio.org/redmine/projects/gnuradio/wiki/Tutorials. Acessado em 9 de novembro de 2012.

[18] Página oficial do GNU Radio "**Out-of-tree modules**", disponível em http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules#gr-howto-write-a-block. Acessado em 11 de novembro de 2012.

[19] Página oficial do **GSL – GNU Scientific Library**, disponível em http://www.gnu.org/software/gsl/. Acessado em 10 de Dezembro de 2012.

[20] Página oficial do GSL, **Wavelet Transforms**, disponível em http://www.gnu.org/software/gsl/manual/html_node/Wavelet-Transforms.html. Acessado em 10 de Dezembro de 2012.

[21] Pagina da API do GNU Radio **gr_wavelet_ff Class Reference,** disponível em http://gnuradio.org/doc/doxygen-3.3/classgr__wavelet__ff.html. Acessado em 13 de Dezembro de 2012.

[22] Daubechies, I. **Ten Lectures on Wavelets**. CBMS-NSF Series in Applied Mathematics n. 61, Philadelphia: SIAM, 1992.

[23] Addison P. S. **The Illustrated Wavelet Transform Handbook.** Introductory Theory and Applications in Science, Engineering, Medicine and Finance. Bristol and Philadelphia: IoP Publishing Ltd 2002.

[24] Gonzalez, R. C.; Woods, R. E.. **Processamento de Imagens Digitais**. São Paulo: Edgard Blucher, 2000.

[25] Lopez, L. A. N. M. **Transformada de wavelet e lógica fuzzy na inspeção por eddy-current em tubos de geradores de vapor de centrais nucleares**. 2002. 1 v. Tese (Doutorado) - Universidade São Paulo, 2002.Tese.

[26] Diniz P. C. A. Rodrigues P. V. F., Veiga A. C. P., Carneiro M. B. P. **Utilização de Rádio Definido por Software na Engenharia Biomédica.** XXIII Congresso Brasileiro em Engenharia Biomédica, p. 447 a 451, 2012.

[27] Physio Bank, MIT-BHI Arrhythmia Data base. Disponível em: http://www.physionet.org/physiobank/. Acessado em 19 de junho de 2012.

ANEXOS

ANEXO A

Rotina gsl_wavelet

```
/* wavelet/gsl_wavelet.h
 *
 * Copyright (C) 2004 Ivo Alxneit
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

#ifndef __GSL_WAVELET_H__
#define __GSL_WAVELET_H__
#include <stdlib.h>
#include <gsl/gsl_types.h>
#include <gsl/gsl_errno.h>

#undef __BEGIN_DECLS
#undef __END_DECLS
#ifdef __cplusplus
# define __BEGIN_DECLS extern "C" {
# define __END_DECLS }
#else
# define __BEGIN_DECLS         /* empty */
# define __END_DECLS           /* empty */
#endif

__BEGIN_DECLS

#ifndef GSL_DISABLE_DEPRECATED
typedefenum {
  forward = 1, backward = -1,
  gsl_wavelet_forward = 1, gsl_wavelet_backward = -1
}
gsl_wavelet_direction;
#else
typedefenum {
  gsl_wavelet_forward = 1, gsl_wavelet_backward = -1
}
```

```
 48 gsl_wavelet_direction;
 49 #endif
 50
 51 typedefstruct
 52 {
 53   const char *name;
 54   int (*init) (const double **h1, const double **g1,
 55                const double **h2, const double **g2, size_t * nc,
 56                size_t * offset, size_t member);
 57 }
 58 gsl_wavelet_type;
 59
 60 typedefstruct
 61 {
 62   constgsl_wavelet_type *type;
 63   const double *h1;
 64   const double *g1;
 65   const double *h2;
 66   const double *g2;
 67   size_tnc;
 68   size_t offset;
 69 }
 70 gsl_wavelet;
 71
 72 typedefstruct
 73 {
 74   double *scratch;
 75   size_t n;
 76 }
 77 gsl_wavelet_workspace;
 78
 79 GSL_VAR constgsl_wavelet_type *gsl_wavelet_daubechies;
 80 GSL_VAR constgsl_wavelet_type *gsl_wavelet_daubechies_centered;
 81 GSL_VAR constgsl_wavelet_type *gsl_wavelet_haar;
 82 GSL_VAR constgsl_wavelet_type *gsl_wavelet_haar_centered;
 83 GSL_VAR constgsl_wavelet_type *gsl_wavelet_bspline;
 84 GSL_VAR constgsl_wavelet_type *gsl_wavelet_bspline_centered;
 85
 86 gsl_wavelet *gsl_wavelet_alloc (constgsl_wavelet_type * T, size_t k);
 87 void gsl_wavelet_free (gsl_wavelet * w);
 88 const char *gsl_wavelet_name (constgsl_wavelet * w);
 89
 90 gsl_wavelet_workspace *gsl_wavelet_workspace_alloc (size_t n);
 91 void gsl_wavelet_workspace_free (gsl_wavelet_workspace * work);
 92
 93 intgsl_wavelet_transform (constgsl_wavelet * w,
 94                           double *data, size_t stride, size_t n,
95                          gsl_wavelet_directiondir,
 96                           gsl_wavelet_workspace * work);
 97
 98 intgsl_wavelet_transform_forward (constgsl_wavelet * w,
 99                                   double *data, size_t stride, size_t n,
100                                   gsl_wavelet_workspace * work);
101
102 intgsl_wavelet_transform_inverse (constgsl_wavelet * w,
103                                   double *data, size_t stride, size_t n,
104                                   gsl_wavelet_workspace * work);
105
106 __END_DECLS
107 #endif /* __GSL_WAVELET_H__ */
```

ANEXO B

Arquivo wavelet.cc

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <stdexcept>
#include <gr_wavelet_ff.h>
#include <gr_io_signature.h>

#include <stdio.h>

// NB in this version, only Daubechies wavelets
// order is wavelet length, even, 2...20

gr_wavelet_ff_sptr
gr_make_wavelet_ff(int size,
int order,
bool forward)
{
  return gr_wavelet_ff_sptr(new gr_wavelet_ff(size,
                                order,
                                forward));
}

gr_wavelet_ff::gr_wavelet_ff(int size,
int order,
bool forward)
  : gr_sync_block("wavelet_ff",
gr_make_io_signature(1, 1, size * sizeof(float)),
gr_make_io_signature(1, 1, size * sizeof(float))),
d_size(size),
d_order(order),
d_forward(forward)
{
d_wavelet = gsl_wavelet_alloc(gsl_wavelet_daubechies, d_order);
  if (d_wavelet == NULL)
    throw std::runtime_error("can't allocate wavelet");
d_workspace = gsl_wavelet_workspace_alloc(d_size);
  if (d_workspace == NULL)
    throw std::runtime_error("can't allocate wavelet workspace");
d_temp = (double *) malloc(d_size*sizeof(double));
  if (d_workspace == NULL)
    throw std::runtime_error("can't allocate wavelet double conversion temp");
}

gr_wavelet_ff::~gr_wavelet_ff()
{
gsl_wavelet_free(d_wavelet);
gsl_wavelet_workspace_free(d_workspace);
  free((char *) d_temp);
}

int
gr_wavelet_ff::work(intnoutput_items,
gr_vector_const_void_star&input_items,
```

```
gr_vector_void_star&output_items)
{
const float *in  = (const float *) input_items[0];
  float       *out = (float *) output_items[0];

  for (int count = 0; count <noutput_items; count++) {
    for (inti = 0; i<d_size; i++)
d_temp[i] = in[i];

    if (d_forward)
gsl_wavelet_transform_forward(d_wavelet,
d_temp,
                          1,
d_size,
d_workspace);
    else
gsl_wavelet_transform_inverse(d_wavelet,
d_temp,
                          1,
d_size,
d_workspace);

    for (inti = 0; i<d_size; i++)
      out[i] = d_temp[i];

    in  += d_size;
    out += d_size;
  }

  return noutput_items;
}
```

Arquivo wavelet.h

```
#ifndef INCLUDED_GR_WAVELET_FF_H
#define INCLUDED_GR_WAVELET_FF_H

#include <iostream>
#include <gr_sync_block.h>

#include <gsl/gsl_errno.h>
#include <gsl/gsl_wavelet.h>

class gr_wavelet_ff;
typedef boost::shared_ptr<gr_wavelet_ff>gr_wavelet_ff_sptr;

gr_wavelet_ff_sptr
gr_make_wavelet_ff(int size = 1024,
int order = 20,
bool forward = true);

/*!
 * \brief compute wavelet transform using gsl routines
 * \ingroupwavelet_blk
 */

class gr_wavelet_ff : public gr_sync_block
{
intd_size;
intd_order;
boold_forward;
gsl_wavelet          *d_wavelet;
gsl_wavelet_workspace *d_workspace;
  double               *d_temp;

  friend gr_wavelet_ff_sptr
gr_make_wavelet_ff(int size,
int order,
bool forward);

gr_wavelet_ff(int size,
int order,
bool forward);

public:
  ~gr_wavelet_ff();

int work (intnoutput_items,
gr_vector_const_void_star&input_items,
gr_vector_void_star&output_items);
};

#endif /* INCLUDED_GR_WAVELET_FF_H */
```