

Jefferson da Silva Cândido

# **Implementações práticas de DSP e RF com GNURadio e HackRF One**

Uberlândia, Minas Gerais

2020



Jefferson da Silva Cândido

# **Implementações práticas de DSP e RF com GNURadio e HackRF One**

Trabalho apresentado na Universidade Federal de Uberlândia como requisito para conclusão do curso de graduação em Engenharia Eletrônica e de Telecomunicações.

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Elétrica

Graduação em Engenharia Eletrônica e de Telecomunicações

Orientador: Dr. Antônio Cláudio Paschoarelli Veiga

Uberlândia, Minas Gerais

2020



JEFFERSON DA SILVA CÂNDIDO  
**Implementações práticas de DSP e RF com GNURadio e  
HackRF One**

Trabalho apresentado na Universidade Federal de Uberlândia como requisito para conclusão do curso de graduação em Engenharia Eletrônica e de Telecomunicações.

Trabalho aprovado. Uberlândia, 09 de dezembro de 2020

---

**Dr. Antônio Cláudio Paschoarelli  
Veiga**  
Orientador

---

**Dr. Gilberto Arantes Carrijo**  
Convidado 1

---

**Dr. Éderson Rosa da Silva**  
Convidado 2

Uberlândia, Minas Gerais  
2020



Este trabalho é dedicado à minha mãe, Maria Aparecida da Silva, que sempre me foi exemplo de obstinação, diligência e honradez.





# Agradecimentos

Agradeço primeiramente a minha família por ter apoiado e viabilizado todo esse processo de aprendizado.

Sou grato pela liberdade e confiança dispensada pelo meu orientador, professor Dr. Antônio Cláudio Paschoarelli Veiga.

Aos professores que contribuíram para o cumprimento dessa jornada.

À Universidade Federal de Uberlândia por cumprir veementemente com o seu papel de formação de cidadãos.

Aos colegas do laboratório de Redes de Computadores e Telecomunicações, William, Daniel e Caio por todo apoio e amizade.

Agradeço também a todas as entidades que estiveram presentes durante minha formação, com destaques para o Diretório Acadêmico da Faculdade de Engenharia Elétrica e ao Laboratório de Automação, Sistemas Eletrônicos e Controle (LASEC), que muito auxiliaram no meu desenvolvimento profissional e de liderança.



*“Messages and the corresponding signals are points in two "function spaces", and the modulation process is a mapping of one space into other.”*  
*(Claude E. Shannon)*



# Resumo

Este é um trabalho introdutório e prático de pesquisa direcionado às áreas de processamento digital de sinais e transmissão e recepção de sinais de radiofrequência. O objetivo é que ele se torne um pontapé inicial para que estudantes de engenharia eletrônica e de telecomunicações possam ter um primeiro contato prático com técnicas de processamento digital de sinais aplicadas a partir de um sistema de rádio definido por software.

A proposta de realização dos experimentos conta com a utilização de métodos e técnicas de programação em C++/Python para a construção de módulos e blocos de processamento de sinais em conjunto ao uso de um *hardware* (o SDR *HackRF One*) dedicado ao propósito de ser a interface entre um computador e o *frontend* de radiofrequência.

No decorrer do trabalho são demonstrados detalhes de implementação, comunicação entre o *hardware* e provisionamento do ambiente e das ferramentas de *software* necessárias para a sua realização. Todos os esforços direcionados resultaram no êxito da demonstração do uso prático da ferramenta GNURadio, assim como a possibilidade de que o engenheiro também atue como um desenvolvedor, o que traz ganhos extremamente benéficos ao seu aprendizado e especialização profissional, além da possibilidade de aplicação dos conceitos teóricos adquiridos sobre sinais e sistemas, processamento digital de sinais, programação de sistemas de alta criticidade, sistemas de comunicações, modulação e demodulação e etc ...

**Palavras-chaves:** rádio, software, gnuradio, hackrf, sdr.



# Abstract

This is an introductory and practical research work aimed at the areas of digital signal processing and the transmission and reception of radio frequency signals. The goal is that it becomes a starting point for students of electronic and telecommunications engineering to have a first practical contact with digital signal processing techniques applied from a software-defined radio system.

The proposal for carrying out the experiments relies on the use of programming methods and techniques in C++/Python for the construction of modules and signal processing blocks together with the use of a *hardware* (SDR *HackRF One*) dedicated to the purpose of being the interface between a computer and the radio frequency *frontend*.

During the work, details of implementation, communication between *hardware* and provisioning of the environment and the *software* tools necessary for its realization are demonstrated. All directed efforts resulted in the successful demonstration of the practical use of the GNURadio tool, as well as the possibility that the engineer also acts as a developer, which brings extremely beneficial gains to his learning and professional specialization, in addition to the possibility of applying of the theoretical concepts acquired about signals and systems, digital signal processing, programming of high criticality systems, communication systems, modulation and demodulation and etc . . .

**Keywords:** radio, software, gnuradio, hackrf, sdr.





# Lista de ilustrações

Figura 1 – Configurações de CPU e memória do computador utilizado. . . . .	29
Figura 2 – Diagrama de blocos simplificado do <i>HackRF One</i> . . . . .	32
Figura 3 – <i>Kit</i> de desenvolvimento de aplicações de rádio definido por software. . .	33
Figura 4 – Teste de verificação de instalação do <i>Docker Engine</i> . . . . .	39
Figura 5 – Criação do <i>container</i> de desenvolvimento a partir de uma imagem Ubuntu. .	41
Figura 6 – Detalhamento de informações sobre <i>container's Docker</i> . . . . .	44
Figura 7 – Conferência de versão, <i>path</i> e componentes habilitados do GNURadio. .	45
Figura 8 – <i>GUI</i> do <i>GNURadio Companion</i> . . . . .	46
Figura 9 – Fazendo o <i>commit</i> da imagem base GNURadio. . . . .	46
Figura 10 – Submetendo a imagem base ao repositório remoto. . . . .	47
Figura 11 – Interface <i>web</i> do <i>DockerHub</i> com a imagem <i>Docker</i> criada. . . . .	48
Figura 12 – Verificação de versão do <i>firmware</i> e taxa de transmissão do <i>HackRF One</i> . .	49
Figura 13 – <i>Flowgraph</i> da soma de dois sinais. . . . .	54
Figura 14 – Gráficos da soma de dois sinais no GNURadio. . . . .	55
Figura 15 – <i>Flowgraph</i> do produto de dois sinais. . . . .	56
Figura 16 – Gráficos do produto de dois sinais no GNURadio. . . . .	57
Figura 17 – <i>Flowgraph</i> para geração de dois sinais defasados no tempo. . . . .	58
Figura 18 – Sinais defasados no domínio do tempo. . . . .	59
Figura 19 – <i>Flowgraph</i> para simulação de um comparador de sinais com histerése. .	60
Figura 20 – Comparação de sinais com histerése. . . . .	61
Figura 21 – <i>Flowgraph</i> para simulação de um receptor de rádio FM. . . . .	64
Figura 22 – <i>FFT plot</i> extraída durante simulação do receptor FM. . . . .	65
Figura 23 – Geração de um módulo <i>OOT</i> do GNURadio e sua estrutura de arquivos e pastas. . . . .	67
Figura 24 – Criação dos arquivos de um bloco customizado usando <b>gr_modtool</b> . .	68
Figura 25 – Prática do desenvolvimento guiado por testes. . . . .	69
Figura 26 – Classe do bloco de processamento digital de sinais customizado. . . . .	70
Figura 27 – Geração do arquivo de descrição do bloco na GUI. . . . .	71
Figura 28 – Geração dos pacotes <b>.deb</b> e <b>.rpm</b> . . . . .	72
Figura 29 – <i>Flowgraph</i> de um sinal em banda base e de uma portadora. . . . .	75
Figura 30 – Sinal mensagem e da portadora no domínio do tempo e da frequência. .	76
Figura 31 – <i>Flowgraph</i> para modulação de um sinal. . . . .	77
Figura 32 – Modulação do sinal mensagem, curvas no tempo e na frequência. . . .	78
Figura 33 – Utilização do bloco customizado em um <i>flowgraph</i> . . . . .	80



# Lista de tabelas

Tabela 1 – Comparativo de SDR's a nível de <i>hardware</i> e custo de investimento. . .	31
---	----



# Lista de abreviaturas e siglas

ADC	<i>Analog to Digital Converter</i>
ARM	<i>Advanced RISC</i>
BW	<i>Bandwidth</i>
CLI	<i>Command Line Interface</i>
DAC	<i>Digital to Analog Converter</i>
DSP	<i>Digital Signal Processing</i>
FFT	<i>Fast Fourier Transform</i>
FIR	<i>Finite Impulse Response</i>
FM	<i>Frequency Modulation</i>
FSF	<i>Free Software Foundation</i>
GNU	<i>GNU's Not Unix</i>
GPG	<i>GNU Privacy Guard</i>
GPLv3	<i>General Public License version 3</i>
GPSDO	<i>GPS Disciplined Oscillator</i>
GRC	<i>GNURadio Companion</i>
GUI	<i>Graphical User Interface</i>
LPF	<i>Low-Pass Filter</i>
OCXO	<i>Oven Controlled Crystal Oscillator</i>
OOT	<i>Out of Tree</i>
SDK	<i>Software Development Kit</i>
SDR	<i>Software-defined Radio</i>
SELinux	<i>Security-Enhanced Linux</i>
SPC	<i>Super Privileged Container</i>

STDIN	<i>Standard Input</i>
TCXO	<i>Temperature Compensated Crystal Oscillator</i>
TDD	<i>Test Driven Development</i>
TTY	<i>TeleTYpewriter</i>

# Lista de símbolos

$R_b$	Taxa de bits [bits/s]
$R_s$	Taxa de símbolos [amostras/s]
$dB$	Decibel
$P$	Potência [W]
$f$	Frequência [Hz]
$\omega$	Frequência angular [rad/s]
$f_s$	Frequência de amostragem [Hz]





# Sumário

	Introdução . . . . .	25
<b>I</b>	<b>PREPARAÇÕES</b>	<b>27</b>
1	HARDWARE . . . . .	29
2	SOFTWARE . . . . .	35
<b>II</b>	<b>DESENVOLVIMENTO</b>	<b>51</b>
3	CRIAÇÃO DE <i>FLOWGRAPH'S</i> NO GNURADIO . . . . .	53
4	O <i>HELLO-WORLD</i> DO GNURADIO . . . . .	63
5	CRIANDO BLOCOS CUSTOMIZADOS . . . . .	67
<b>III</b>	<b>RESULTADOS</b>	<b>73</b>
6	REVISÃO LITERÁRIA E ANÁLISE DE RESULTADOS . . . . .	75
7	CONSIDERAÇÕES FINAIS . . . . .	81
	REFERÊNCIAS . . . . .	83
	<b>ANEXOS</b>	<b>85</b>
	ANEXO A – ATRIBUIÇÃO DE FAIXAS DE FREQUÊNCIAS NO BRASIL . . . . .	87
	ANEXO B – DIAGRAMA ESQUEMÁTICO DO <i>HACKRF ONE</i> - PÁGINA 1. . . . .	89
	ANEXO C – DIAGRAMA ESQUEMÁTICO DO <i>HACKRF ONE</i> - PÁGINA 2. . . . .	91

**ANEXO D – DIAGRAMA ESQUEMÁTICO DO *HACKRF ONE* -  
PÁGINA 3. . . . . 93**

**ANEXO E – DIAGRAMA PARA MONTAGEM DOS COMPONENTES  
ELETRÔNICOS NO PCB DO *HACKRF ONE*. . 95**

# Introdução

O processamento digital de sinais que é conhecido por engenheiros atualmente começou a florescer por volta dos anos 1960 [1] passando por uma evolução dramática até alcançar solidez em técnicas e implementações. O cenário atual que o mundo se encontra é altamente favorável para que estudantes de engenharia direcionem seus esforços para esta área, pois o processo de transformação digital da indústria e da sociedade está ocorrendo de forma acelerada [2] mantendo alta a demanda por profissionais com habilidades técnicas correlatas.

Os grandes aliados do engenheiro na execução de seus trabalhos estão diretamente relacionados com a sua capacidade de descobrir e aplicar ferramentas que facilitem a execução de suas tarefas e a geração de relatórios e métricas que auxiliem em diagnósticos e na tomada de decisões que impactam os negócios que ele está envolvido, seja como funcionário ou como empreendedor. Dentre as principais ferramentas que o engenheiro pode ter ao seu lado, está o uso de boas práticas no que tange o desenvolvimento de *software* e, mais especificamente para este trabalho, está a ferramenta GNURadio que vem como uma proposta livre de integração e multi-disciplinaridade dentro das áreas estudadas por engenheiros em eletrônica e telecomunicações.

Para utilizar o *software* escrito utilizando o GNURadio além das simulações, o engenheiro precisa de um *hardware*, chamado SDR (*Software-defined Radio*), capaz de receber este *software* fazer as devidas conversões de frequência e transmitir/receber o sinal de radiofrequência desejado. As vantagens na utilização do SDR como o reuso de *hardware* e a utilização de níveis mais altos de abstração do *software* como também algumas desvantagens (desempenho inferior se comparado a *hardware's* dedicados, problemas de interoperabilidade e segurança) devem sempre ser avaliados para cada projeto [3]

Um dos diferenciais alcançados durante o desenvolvimento dessa pesquisa foi a utilização de *container's* para o isolamento do ambiente de desenvolvimento de *software*, o que flexibiliza o processo de criação, sendo que o impacto esperado é incentivar que outros estudantes também apliquem seus conhecimentos teóricos sobre processamento digital de sinais utilizando as ferramentas e boas práticas que aqui serão mencionadas.

O objetivo principal deste trabalho será trazer a visão de um estudante de engenharia eletrônica e de telecomunicações sobre o contato inicial com o desenvolvimento de aplicações de rádio definido por *software* utilizando o GNURadio para a realização do processamento digital de sinais.



Parte I

Preparações



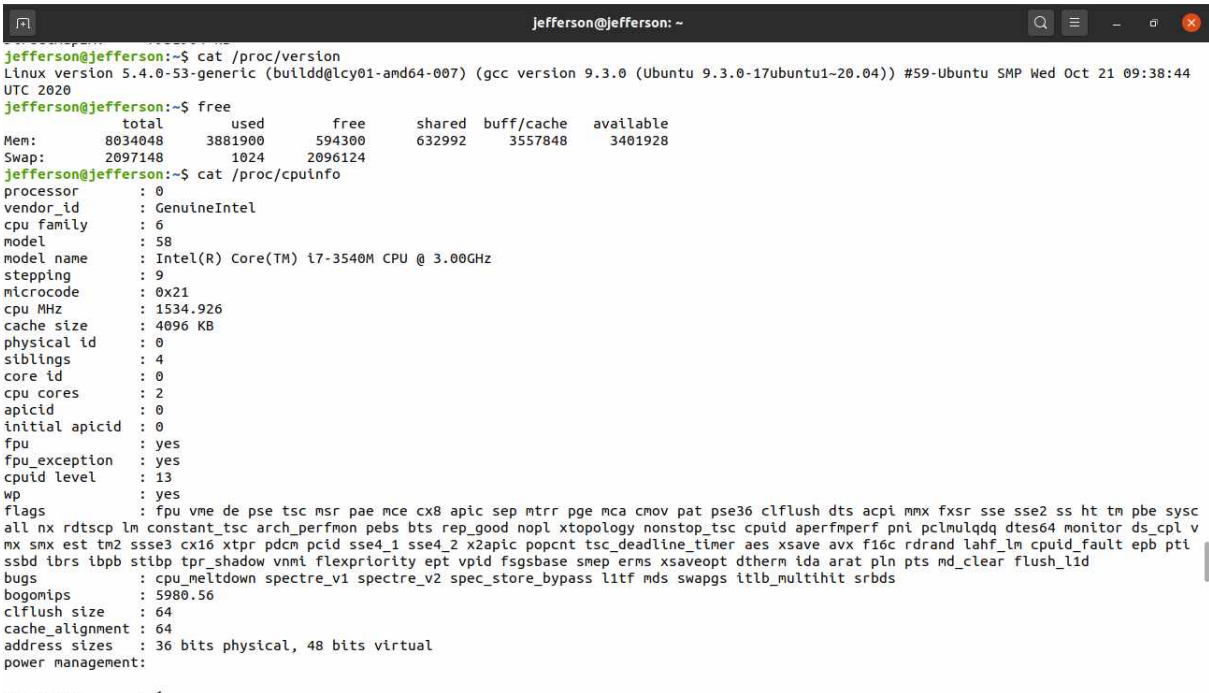
# 1 Hardware

Neste capítulo são abordados os dispositivos de *hardware* utilizados no desenvolvimento deste trabalho, trazendo brevemente um comparativo de quais podem ser encontrados no mercado atualmente e quais as especificações técnicas disponibilizadas por alguns fabricantes.

Basicamente, é necessário um dispositivo *host* (arquiteturas **x86\_64** ou **amd64**, **armhf** e **arm64/aarch64**, por exemplo), preferencialmente com alguma distribuição *Linux* e o *GNURadio-companion* instalados, uma placa de circuito impresso do sistema de rádio definido por *software* contendo algum cabo para comunicação com o computador *host* (normalmente USB 2.0/3.0), as antenas para transmissão e/ou recepção e algum oscilador (TCXO (*Temperature Compensated Crystal Oscillator*), OCXO (*Oven Controlled Crystal Oscillator*), GPSDO (*GPS Disciplined Oscillator*), etc...) caso a aplicação exija sincronismo por *clock* externo.

Com relação ao computador utilizado, trata-se de um notebook de arquitetura **amd64** com 8 GB de memória DDR3 e processador Intel® Core™ i7 de terceira geração com quatro núcleos e 3.0 GHz de *clock* conforme estatísticas retiradas do próprio sistema operacional e mostradas na Fig. 1.

Figura 1 – Configurações de CPU e memória do computador utilizado.



```

jefferson@jefferson: ~
jefferson@jefferson:~$ cat /proc/version
Linux version 5.4.0-53-generic (buildd@lcy01-amd64-007) (gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)) #59-Ubuntu SMP Wed Oct 21 09:38:44
UTC 2020
jefferson@jefferson:~$ free
              total        used        free     shared  buff/cache   available
Mem:           8034048    3881900     594300      632992     3557848     3401928
Swap:          2097148           1024     2096124
jefferson@jefferson:~$ cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 58
model name    : Intel(R) Core(TM) i7-3540M CPU @ 3.00GHz
stepping     : 9
microcode    : 0x21
cpu MHz      : 1534.926
cache size   : 4096 KB
physical id  : 0
siblings     : 4
core id      : 0
cpu cores    : 2
apicid       : 0
initial apicid : 0
fpu          : yes
fpu_exception : yes
cpuid level  : 13
wp           : yes
Flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl v
mx smx est tm2 sse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm cpuid_fault epb pti
ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase smep erms xsaveopt dtherm ida arat pln pts md_clear flush_lid
bugs         : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit srbds
bogomips     : 5980.56
clflush size : 64
cache_alignm : 64
address sizes : 36 bits physical, 48 bits virtual
power management:

```

Fonte: Elaborado pelo autor.

## Avaliação de SDR's

Para o sistema de rádio definido por *software* primeiramente foi realizada uma pesquisa de mercado para avaliar quais os SDR's disponíveis e quais *feature's* cada um oferece. Pontos importantes a serem observados antes de efetuar a compra de um SDR são: se ele pode atuar apenas como receptor (RX) ou como receptor e transmissor (RX/TX), qual a faixa de frequência de atuação, a largura de banda máxima e a resolução do ADC/DAC (número de *bits* por amostra), qual o suporte a nível de *software* e qual o custo do investimento. Entre os dispositivos apenas receptores, destacaram-se o RTL-SDR, o AirSpy Mini e o SDRplay RSPduo, os quais atuam até a casa de 2 GHz aproximadamente com larguras de banda máximas variando entre 3 MHz e 10 MHz. Com relação aos SDR's RX/TX foram considerados o LimeSDR Mini, BladeRF, *HackRF One*, Ettus USRP B210 e o Adalm PlutoSDR e todos podem ser utilizados com largura de banda de pelo menos 20 MHz e podendo variar entre 1 MHz e 6 GHz no domínio da frequência.

O resumo dos principais SDR's que podem ser encontrados no mercado é mostrado na Tabela 1, com preços em dólares cotados *online* através do site <https://pt.aliexpress.com> e também dos próprios fabricantes SDRplay, Analog Devices (Adalm Pluto), Nuand (BladeRF) e Ettus (USRP B210). O SDR escolhido para este trabalho foi o *HackRF One*, que será abordado com mais detalhes na próxima seção.

### HackRF One

O *HackRF One* é um SDR com *hardware* de código aberto que foi desenvolvido pelo pesquisador em segurança de redes sem fio, Michael Ossmann. Trata-se de um transceptor *half-duplex* com largura de banda máxima de 20 MHz (BW - *Bandwidth*) preparado para atuar em uma grande faixa de frequências que vão de 1 MHz a 6 GHz, possibilitando o processamento de até 20 milhões de amostras por segundo (tendo 8 bits por amostra em quadratura - sinais I/Q com 8 bits para o I e 8 bits para o Q) recebidas ou transmitidas por meio de uma antena fixada em seu conector SMA fêmea, de forma que o ganho - de recepção (RX) ou de transmissão (TX) - é configurável pelo *software* utilizado, que neste caso é o GNURadio. O *HackRF One* é alimentado pela tensão VCC fornecida pela porta de comunicação USB 2.0, a qual também possibilita que o SDR transmita até 40 MB por segundo (20 milhões de amostras de 2 bytes a cada segundo).

Em quinze de maio de dois mil e quinze, Michael Ossmann publicou que seu SDR é capaz de operar em frequências menores que 10 MHz [4], reafirmando a faixa de operação do *HackRF One*. A potência máxima de transmissão do *HackRF One* varia de acordo com a faixa de frequência de operação e é suficiente para a realização de testes de curto alcance. Para recepção, o máximo de potência fica na ordem de -5 dBm e exceder este valor pode resultar em danos permanentes ao *HackRF One* [5].



Tabela 1 – Comparativo de SDR's a nível de *hardware* e custo de investimento.

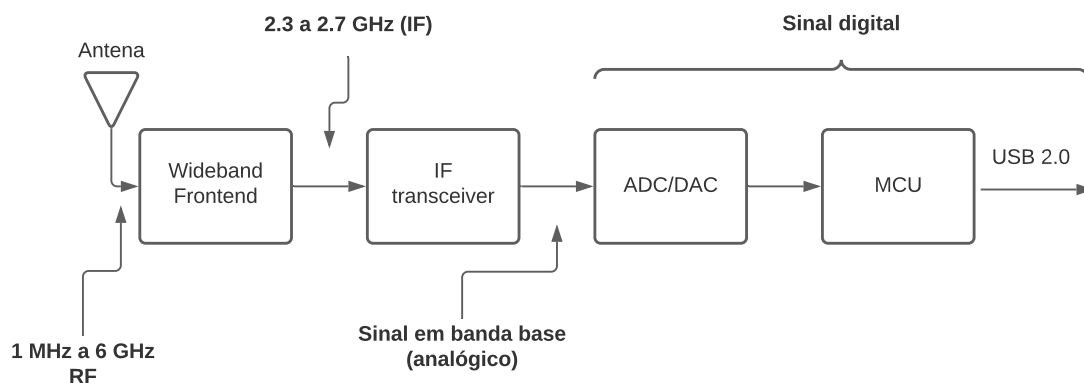
	Faixa de frequência	Largura de banda máxima	RX/TX	Half / Full Duplex	Resolução do ADC	Preço (US\$)
<b>RTL-SDR</b>	500 kHz a 1766 MHz	3 MHz	RX	-	8 bits	~25,00
<b>RSPduo</b>	1 kHz a 2 GHz	10 MHz	RX	-	14 bits	~260,00
<b>AirSpy Mini</b>	24 Hz a 1.7 GHz	6 MHz	RX	-	até 16 bits	~150,00
<b>USRP B200/B210</b>	70 MHz a 6 GHz	56 MHz	RX/TX	Full Duplex	12 bits	a partir de 890,00
<b>PlutoSDR</b>	325 MHz a 3.8 GHz	20 MHz	RX/TX	Full Duplex	12 bits	249,00
<b>LimeSDR Mini</b>	10 MHz a 3.5 GHz	30 MHz	RX/TX	Full Duplex	12 bits	de 300,00 a 350,00
<b>BladeRF</b>	300 MHz a 3.8 GHz	40MHz	RX/TX	Full Duplex	12 bits	a partir de 720,00
<b>HackRF</b>	1 MHz a 6 GHz	20 MHz	RX/TX	Half Duplex	8 bits	a partir de 70,00

Os valores apresentados a seguir, aproximados, estão relacionados à potência de transmissão do *HackRF One* e foram coletados e disponibilizados pelo fabricante [6]:

- 1 MHz a 10 MHz: 5 dBm a 15 dBm (3 mW a 30 mW) - crescente com a frequência;
- 10 MHz a 2150 MHz: 15 dBm a 5 dBm (30 mW a 3 mW) - decrescente com a frequência;
- 2150 MHz a 2750 MHz: 13 dBm a 15 dBm (20 mW a 30 mW) - tendência de crescimento/decrescimento não informada pelo fabricante;
- 2750 MHz a 4000 MHz: de 5 dBm a 0 dBm (3 mW a 1 mW) - decrescente com a frequência;
- 4000 MHz a 6000 MHz: de 0 dBm a -10 dBm (1 mW a 0.1 mW) - decrescente com a frequência.

A Fig. 2 ilustra o diagrama de blocos simplificado do *HackRF One*. Por se tratar de um equipamento que lida com radiofrequência, o uso do *HackRF One* deve estar dentro das observações legais da região em que for aplicado. No Brasil é importante que o engenheiro esteja atento às regras reguladas pela Agência Nacional de Telecomunicações, ANATEL, como por exemplo a Resolução nº 716, de 31 de outubro de 2019 em que foi aprovado o plano de atribuição, destinação e distribuição de faixas de frequências no Brasil (PDF). O Anexo A contém o quadro de atribuição de faixas de frequências no Brasil, edição de 2019.

Figura 2 – Diagrama de blocos simplificado do *HackRF One*.



Fonte: Elaborado pelo autor com base na documentação fornecida pelo fabricante [7].

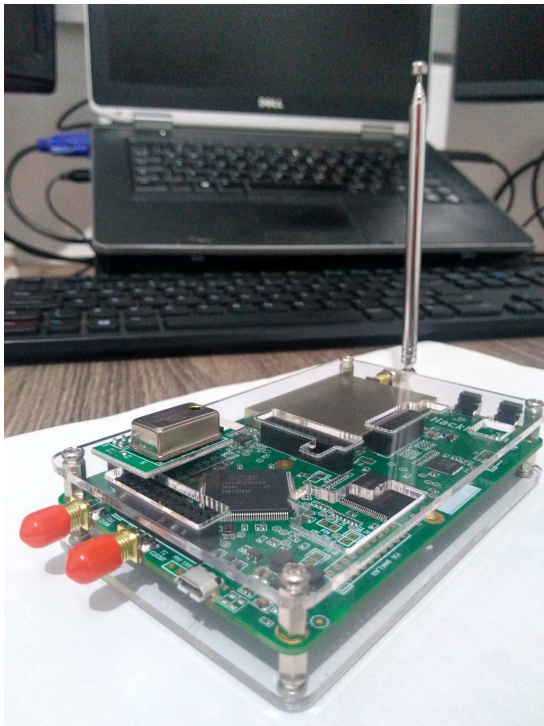
O *kit* que foi adquirido no site <https://pt.aliexpress.com> veio com uma placa de circuito impresso do SDR *HackRF One*, um oscilador TCXO, uma antena telescópica (banda passante de 40 MHz a 6 GHz), uma antena omnidirecional (para recepção de sinais GSM, 3G e 4G), um cabo USB e peças de acrílico para montagem do *kit*. As fotos da Fig. 3 foram tiradas após a montagem do *kit* e os diagramas esquemático e de montagem referentes ao projeto foram anexados a este trabalho (Anexos B a E).



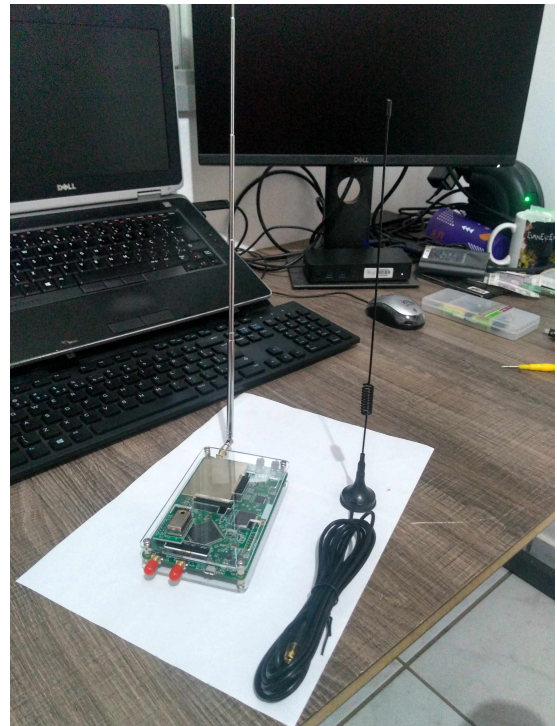
(a) Cabo USB e antena telescópica.



(b) Antena Omnidirecional.



(c) HackRF com TCXO.



(d) Kit completo.

Figura 3 – *Kit* de desenvolvimento de aplicações de rádio definido por software.



## 2 Software

O conceito e as boas práticas relacionadas ao desenvolvimento de *software* evoluiu bastante desde os primórdios da computação e algo que sempre esteve em pauta é o problema de incompatibilidade de ambientes de desenvolvimento, o que acabou dando origem ao jargão: “na minha máquina funciona”. Configurar o ambiente de desenvolvimento e depois ter incompatibilidade de versão de alguma biblioteca ou então não ter a facilidade de replicar o ambiente que foi utilizado durante o desenvolvimento é um grande problema enfrentado por engenheiros e/ou programadores [8], principalmente os que estão em início de carreira. Nesta parte deste trabalho será demonstrada a criação de um ambiente para desenvolvimento de aplicações de rádio definido por *software* utilizando o **GNURadio** instalado em um *container* **Docker**. Os conceitos de *container's* e do *Docker* com certeza já são amplamente abordados na atualidade e então, apenas alguns pontos relacionados ao tema serão tratados aqui.

### GNURadio

O GNURadio é um SDK (*software development kit*) de um projeto *OpenSource* criado com o intuito de auxiliar no desenvolvimento de aplicações de rádio definido por *software* inicialmente publicado no ano de 2001 como um pacote oficial GNU. Este *software* pode ser utilizado tanto por *hobbyistas*, como também em meio acadêmico ou para suprir necessidades do mercado no que tange comunicações sem fio ou qualquer tipo de sistema de rádio digital do mundo real. O GNURadio está sob uma licença pública geral — GNU GPLv3 — da *Free Software Foundation* (FSF).

Por ser um projeto de *software* agnóstico ao *hardware* das plataformas de desenvolvimento, o GNURadio é idealizado para trabalhar apenas com dados digitalizados e a partir disso executar um processamento digital de sinais (DSP — *Digital Signal Processing*) definido por aplicações escritas para receber ou enviar dados de sistemas de *streaming* digital programaticamente, utilizando a linguagem de programação Python — que é considerado mais fácil — ou C++ — para escrita de códigos de desempenho crítico — além de também ser possível através de uma interface gráfica de usuário (**GNURadio Companion** — GRC) por diagramas de blocos.

## Docker

O *Docker* é uma unidade padrão de *software* da empresa *Docker Inc.*, que fornece uma camada de abstração e automação em containers, ou seja, grupos isolados de processos do *kernel* Linux sendo executados e compartilhando os recursos de um mesmo *host*. Um bom entendimento sobre os *namespaces* do sistema operacional Linux (**mnt**, **pid**, **net**, **ipc**, **uts**, **user** e **cgroups**) pode ajudar na compreensão de quais soluções os *containers* podem oferecer.

Um questionamento bastante pertinente que pode vir à tona neste momento é: Seria mesmo necessário utilizar *container's* para criação do ambiente de desenvolvimento, sendo que o próprio GNURadio já é distribuído oficialmente em pacotes compatíveis com os principais sistemas operacionais encontrados no mercado? Com o decorrer do próximo capítulo o porquê se torna mais claro e por agora a ideia principal é de que quanto mais isolado possível for o ambiente de desenvolvimento, mais fácil será para testar hipóteses, criar diferentes soluções e tornar possível que elas sejam reproduzidas com facilidade independentemente se o código que foi desenvolvido em um sistema *host* está sendo executado em outro.

Muitas ferramentas que podem ser úteis para engenheiros e desenvolvedores de *software* para gerenciar ou solucionar problemas podem não estar incluídas no sistema *host* de suas máquinas por padrão e a melhor maneira de adicionar ferramentas a um *host* seria incluindo-as em um *container* possibilitando que o *host* seja o mais "enxuto" possível.

Os *container's* são projetados para manter suas próprias visualizações contidas de *namespaces* e têm acesso limitado aos *hosts* nos quais são executados. Por padrão, os *container's* têm uma tabela de processos, interfaces de rede, sistemas de arquivos e recursos IPC (*Inter-Process Communication*) separados do *host*. Muitos recursos de segurança, como por exemplo o SELinux (*Security-Enhanced Linux*), são colocados em *container's* para controlar o acesso ao sistema *host* e outros *container's*. Embora os *container's* possam usar recursos do *host*, os comandos executados a partir de um *container* têm uma capacidade muito limitada de interagir diretamente com o *host*.

Alguns *container's*, entretanto, têm como objetivo acessar, monitorar e, possivelmente, alterar recursos no sistema *host* diretamente [9]. Eles são chamados de *container's* super privilegiados (SPC - *Super Privileged Container*). Um desenvolvedor pode *subir* um SPC em um *host*, solucionar um problema e removê-lo quando não for mais necessário para liberar recursos.

No próximo capítulo é mostrado como utilizar dos *container's* super privilegiados para a criação do ambiente de desenvolvimento de aplicações de *software* e como os recursos de um sistema *host* são acessados a partir desse SPC.

## Criação do ambiente de desenvolvimento

Primeiramente é necessário ter a CLI (*Command Line Interface*) do **docker engine** instalada (na máquina *host*) e tal procedimento de instalação se torna extremamente simples bastando seguir o passo-a-passo fornecido pela [documentação oficial](#) do *Docker* [10]. Esta CLI está disponível para várias distribuições Linux na versão *Server* e também para Windows e Mac na versão *Desktop*. Neste trabalho será demonstrado o procedimento de instalação do **docker engine** em uma distribuição Linux, de forma que este procedimento é semelhante para demais distribuições e arquiteturas.

### Instalação do *Docker Engine*

O *Docker* fornece pacotes prontos (**.deb** e **.rpm**) para instalação dessa CLI em distribuições Linux como *CentOS*, *Debian*, *Fedora*, *Raspbian*, *Ubuntu* e derivadas (*LMDE*, *BunsenLabs Linux*, *Kali Linux*, *Kubuntu*, *Lubuntu* e *Xubuntu*, por exemplo) para arquiteturas **x86\_64/amd64**, **ARM (Advanced RISC Machine)** e **ARM64/AARCH64**. Outra opção de instalação seria utilizando os binários pré-compilados que são fornecidos no site do *Docker* e essa forma faz bastante sentido quando o sistema operacional utilizado não fizer parte de algum dentre todos os suportados pelo *Docker*.

Antes de iniciar o procedimento de instalação é necessário verificar se o sistema operacional do *host* utilizado está instalado em uma versão de hardware de arquitetura 64-bit (**x86\_64** ou **amd64**, **armhf** e **arm64/aarch64**) e não possui instalações de versões antigas do *software* (**docker**, **docker.io**, ou **docker-engine**) e, caso hajam, é necessário removê-las. No sistema operacional Ubuntu basta utilizar o gerenciador de pacotes **apt/apt-get** para fazê-lo, conforme é exemplificado a seguir:

```
$ sudo apt-get remove docker-engine \
    docker docker.io containerd runc
```

### Instalação utilizando o repositório oficial

Utilizar o gerenciador de pacotes presente na distribuição Linux facilita o processo de instalação e remoção de *software* do sistema operacional e esta instalação será feita utilizando o pacote fornecido no repositório oficial do *Docker*. O primeiro passo é atualizar a lista de pacotes através do comando **apt-get update** e instalar algumas dependências que permitem que o gerenciador de pacotes (**apt**) use um repositório sobre HTTPS. Os comandos são exemplificados como segue:

```
$ sudo apt-get update

$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
```

Após isso é necessário adicionar a chave GPG (*GNU Privacy Guard*) oficial do *Docker* através do comando:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  | sudo apt-key add -
```

Para verificar se o sistema já possui a chave com o *fingerprint Docker* (no momento em que este texto foi escrito, era **9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88**), o usuário deve pesquisar os últimos 8 caracteres desse *fingerprint*. Neste caso, basta utilizar o comando:

```
$ sudo apt-key fingerprint 0EBFCD88
```

E o retorno do terminal ficará:

```
pub  rsa4096 2017-02-22 [SCEA]
9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid  [unknown] Docker Release (CE deb) <docker@docker.com>
sub  rsa4096 2017-02-22 [S]
```

Feito isso, o próximo passo será configurar que o repositório mais estável ("*stable*") do *Docker* seja indexado ao gerenciador de pacotes do sistema, o **apt**:

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

Finalmente, as versões mais recentes do **containerd** e do *Engine* do *Docker* podem ser instaladas após atualizar os índices do gerenciador de pacotes **apt**.

```
$ sudo apt-get update
$ sudo apt-get install docker-ce \
  docker-ce-cli containerd.io
```

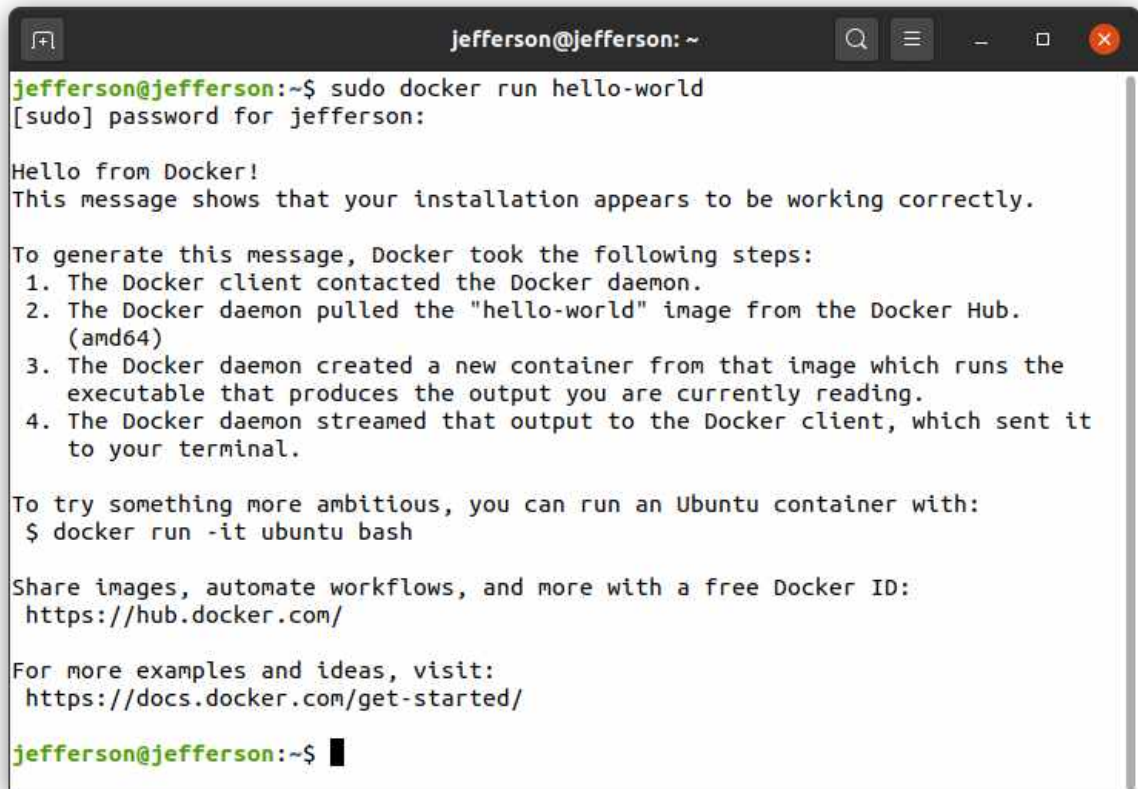
Para verificar se a instalação foi bem-sucedida, basta executar o comando de testes a seguir que é o "*hello-world*" do *Docker*. Se tudo ocorrer bem, o terminal responderá com



uma saída semelhante ao que é mostrado na Fig. 4.

```
$ sudo docker run hello-world
```

Figura 4 – Teste de verificação de instalação do *Docker Engine*.



```
jefferson@jefferson: ~  
jefferson@jefferson:~$ sudo docker run hello-world  
[sudo] password for jefferson:  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
 1. The Docker client contacted the Docker daemon.  
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
    (amd64)  
 3. The Docker daemon created a new container from that image which runs the  
    executable that produces the output you are currently reading.  
 4. The Docker daemon streamed that output to the Docker client, which sent it  
    to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/  
  
jefferson@jefferson:~$ █
```

Fonte: Elaborado pelo autor.

Neste ponto, o *Docker* encontra-se devidamente instalado no sistema operacional e o usuário pode optar por executar alguns comandos de pós-instalação para, por exemplo, possibilitar a execução do *Docker* sem a necessidade de ter privilégios de usuário *root*, iniciar o serviço *Docker* na inicialização do sistema operacional, usar uma *engine* de armazenamento diferente da que vem como padrão de instalação (*overlay2*) dentre outras coisas descritas na seção de [pós-instalação](#) da documentação oficial [11].

## Criação do container

Com o *Docker* instalado, para iniciar o procedimento de criação do *container* do ambiente de desenvolvimento é preciso executar o comando **xhost +** para fornecer acesso ao servidor gráfico do *host* a aplicações “externas” ao *host* (também conhecido como sessão **X** ou a tela do computador), afinal um dos objetivos é utilizar a interface gráfica do *GNURadio Companion* que estará instalado dentro do *container*.

Agora, executando o seguinte comando será criado um *container* a partir de uma

imagem base da distribuição *Ubuntu*. A Fig. 5 exemplifica o uso do comando e as saídas retornadas pelo terminal e é dentro do *container* criado que será feita a instalação do GNURadio.

```
$ docker run -i -t --privileged \
  --ipc=host --net=host --pid=host -e HOST=/host \
  -e DISPLAY=$DISPLAY \
  -e DCONF_PROFILE=/etc/dconf/profile/ \
  -e XDG_DATA_HOME=/config/xdg/data \
  -e XDG_CONFIG_HOME=/config/xdg/config \
  -e XDG_CACHE_HOME=/config/xdg/cache \
  -e XDG_RUNTIME_DIR=/tmp/runtime-root \
  -e DBUS_SESSION_BUS_ADDRESS="$DBUS_SESSION_BUS_ADDRESS" \
  -e DEBIAN_FRONTEND="noninteractive" \
  -e IMAGE=jeffcandido/gnuradio \
  -v /run:/run -v /var/log:/var/log \
  -v /etc/localtime:/etc/localtime -v /:/host \
  -v /etc/dconf/profile:/etc/dconf/profile/ \
  -v /tmp/runtime-root:/tmp/runtime-root/ \
  -v /tmp/.X11-unix:/tmp/.X11-unix \
  -v /dev/usb:/dev/usb \
  -v /dev/snd:/dev/snd \
  -v /home/jefferson/gnuradio:/home/jefferson/gnuradio \
  --name gnuradio \
  ubuntu:20.04
```

Para melhorar o entendimento, essa linha vai ser dividida e explicada por partes. O comando **docker run** primeiro cria uma camada de *container* gravável sobre a imagem especificada (*Ubuntu*) e, em seguida, o inicia usando o comando especificado. Ou seja, **docker run** é equivalente a utilizar **docker container create** e depois **docker container start**. Um *container* interrompido pode ser reiniciado com todas as suas alterações anteriores intactas usando **docker start**. Agora, sobre as *flags* passadas:

- **i (interactive)**: manter o **STDIN** (*standard input* — fluxo de entrada padrão) aberto, mesmo se não estiver conectado;
- **t (tty)**: alocar um pseudo-**TTY** (*TeleTYpewriter* — simplesmente um terminal ao qual o usuário está conectado);
- **privileged**: dar privilégios estendidos a este *container* (desativa a separação de segurança entre o *host* e o *container*, o que significa que um processo executado

Figura 5 – Criação do *container* de desenvolvimento a partir de uma imagem Ubuntu.

```

root@jefferson: /
jefferson@jefferson:~$ docker run -i -t --privileged \
> --ipc=host --net=host --pid=host -e HOST=/host \
> -e DISPLAY=$DISPLAY \
> -e DCONF_PROFILE=/etc/dconf/profile/ \
> -e XDG_DATA_HOME=/config/xdg/data \
> -e XDG_CONFIG_HOME=/config/xdg/config \
> -e XDG_CACHE_HOME=/config/xdg/cache \
> -e XDG_RUNTIME_DIR=/tmp/runtime-root \
> -e DBUS_SESSION_BUS_ADDRESS="$DBUS_SESSION_BUS_ADDRESS" \
> -e DEBIAN_FRONTEND="noninteractive" \
> -e NAME=gnuradio -e IMAGE=jeffcandido/gnuradio \
> -v /run:/run -v /var/log:/var/log \
> -v /etc/localtime:/etc/localtime -v /:/host \
> -v /etc/dconf/profile:/etc/dconf/profile/ \
> -v /tmp/runtime-root:/tmp/runtime-root/ \
> -v /tmp/.X11-unix:/tmp/.X11-unix \
> -v /dev/usb:/dev/usb \
> -v /dev/snd:/dev/snd \
> -v /home/jefferson/gnuradio:/home/jefferson/gnuradio \
> --name gnuradio \
> ubuntu:20.04
Unable to find image 'ubuntu:20.04' locally
20.04: Pulling from library/ubuntu
e6ca3592b144: Pull complete
534a5505201d: Pull complete
990916bd23bb: Pull complete
Digest: sha256:cbcf86d7781dbb3a6aa2bcea25403f6b0b443e20b9959165cf52d2cc9608e4b9
Status: Downloaded newer image for ubuntu:20.04
root@jefferson:/#

```

Fonte: Elaborado pelo autor.

como *root* dentro do *container* tem o mesmo acesso ao *host* que ele poderia também ter se fosse executado de fora do *container*.);

- **e (env)**: definir variáveis de ambiente;
- **v (volume)**: vincular a montagem de um volume dentro do *container*;
- **name**: o nome que se dará ao *container* criado.
- As *flags* **-ipc=host**, **-net=host** e **-pid=host** desligam os *namespaces* **ipc**, **net** e **pid** dentro do *container*. Isso significa que os processos dentro do *container* veem a mesma rede e tabela de processos, bem como compartilham quaisquer IPCs [12] com os processos do *host*.

Indo um pouco mais a fundo, tem-se a configuração de algumas variáveis de ambiente e a definição dos pontos de montagem e vinculação de alguns *volumes* ao *host*. Em tese, estas variáveis de ambiente foram configuradas porque foram observadas algumas necessidades durante o desenvolvimento deste trabalho, tais quais:

- **HOST=/host**: definir uma variável que possa ser usada dentro do *container* para acessar arquivos e diretórios a partir da raiz do *filesystem* do *host*;
- **DISPLAY=\$DISPLAY**: definir uma variável que identifique onde serão exibidos recursos pelo servidor gráfico **X**;
- **DCONF\_PROFILE=/etc/dconf/profile/**: definir a variável relacionada a um sistema de armazenamento das preferências de usuário [13];
- **XDG\_DATA\_HOME=/config/xdg/data**: definir qual será o único diretório base relativo ao qual os arquivos de dados específicos do usuário devem ser gravados [14];
- **XDG\_CONFIG\_HOME=/config/xdg/config**: definir qual será o único diretório base relativo ao qual os arquivos de configuração específicos do usuário devem ser gravados [14];
- **XDG\_CACHE\_HOME=/config/xdg/cache**: definir qual será o único diretório base relativo ao qual os dados não essenciais específicos do usuário (em cache) devem ser gravados [14];
- **XDG\_RUNTIME\_DIR=/tmp/runtime-root**: definir qual será o único diretório base relativo ao qual os arquivos de tempo de execução específicos do usuário e outros objetos de arquivo devem ser colocados;
- **DBUS\_SESSION\_BUS\_ADDRESS=**  
**"\$DBUS\_SESSION\_BUS\_ADDRESS"**: possibilitar a inicialização de uma sessão de barramento usando o utilitário D-Bus [15];
- **DEBIAN\_FRONTEND="noninteractive"**: definir variável para silenciar os *prompt's* de configuração do *container* [16];
- **IMAGE=jeffcandido/gnuradio**: definir a variável para identificar o nome da imagem.

Com relação aos *volumes* que foram montados, seguem suas definições:

- **/run:/run**: montar o diretório **/run** do *host* no diretório **/run** dentro do *container*. Isso permite que os processos dentro do *container* falem com o serviço **dbus** do *host* e falem diretamente com o serviço **systemd**;
- **/var/log:/var/log**: permitir que comandos sejam executados dentro do *container* para ler e gravar arquivos de log no diretório **/var/log** do *host*;

- `/etc/localtime:/etc/localtime`: fazer com que o fuso horário do sistema *host* seja usado também no *container*;
- `/:/host`: montar a raiz da árvore de diretórios do *host* (mais conhecida como `/`) no ponto de montagem `/host` para permitir que processos dentro do *container* consigam modificar facilmente o conteúdo no *host*;
- `/etc/dconf/profile/:/etc/dconf/profile/`: vincular o sistema de armazenamento de preferências do usuário ao mesmo do *host* [17];
- `/tmp/runtime-root/:/tmp/runtime-root/`: montar e vincular o diretório base relativo aos arquivos de tempo de execução específicos do usuário;
- `/tmp/.X11-unix/:/tmp/.X11-unix/`: montar um volume para o *unix socket* X11 (servidor gráfico);
- `/dev/usb:/dev/usb`: vincular o volume que possibilita a utilização de recursos da porta USB do *host*;
- `/dev/snd:/dev/snd`: vincular o volume que possibilita a utilização de recursos de áudio do *host*;
- `/home/jefferson/gnuradio:/home/jefferson/gnuradio`: montar um diretório que será utilizado no desenvolvimento deste trabalho e que poderá ser acessado tanto pelo *host* como pelo *container*.

Resumindo, foi utilizada uma linha de comando para criar um *container*, o qual foi nomeado **gnuradio**, a partir de uma imagem base (Ubuntu) com privilégios para acessar recursos da máquina (o *host*), onde mais precisamente será possível utilizar os recursos de áudio, da porta USB e da interface visual com uma vinculação de volumes montados no *container* a partir da máquina *host*.

Algo importante a se notar na Fig. 5 é que depois da criação do *container* o terminal “mudou” do usuário **jefferson** para **root** e manteve o *hostname*. Isso ocorre porque a partir desse momento o que aparece na tela é o terminal visto de “dentro” do **container**, ou seja, com o usuário **root** em um *container* que está enxergando o mesmo *namespace net* que o sistema *host*.

Em outro terminal é possível obter mais informações como, por exemplo, quais *container*’s estão sendo executados no momento ou inspecionar algum *container* específico em busca de maiores detalhes, como é mostrado na Fig. 6.

Figura 6 – Detalhamento de informações sobre *container's Docker*.

```

jefferson@jefferson:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
8acc6e3cc0b9   ubuntu:20.04   "/bin/bash"            3 hours ago   Up 3 hours   -             gnuradio
jefferson@jefferson:~$ docker inspect gnuradio
[
  {
    "Id": "8acc6e3cc0b9ca6daab7466362d856c483a3f201b6318e8f79e23a17787093cf",
    "Created": "2020-09-19T05:03:06.776842569Z",
    "Path": "/bin/bash",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 102292,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2020-09-19T05:03:07.677766992Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "sha256:bb0eaf4eee00c28cb8ffd54e571dd225f1dd2ed8d8751b2835c31e84188bf2de",
    "ResolvConfPath": "/var/lib/docker/containers/8acc6e3cc0b9ca6daab7466362d856c483a3f201b6318e8f79e23a17787093cf/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/8acc6e3cc0b9ca6daab7466362d856c483a3f201b6318e8f79e23a17787093cf/hostname",
    "HostsPath": "/var/lib/docker/containers/8acc6e3cc0b9ca6daab7466362d856c483a3f201b6318e8f79e23a17787093cf/hosts",
    "LogPath": "/var/lib/docker/containers/8acc6e3cc0b9ca6daab7466362d856c483a3f201b6318e8f79e23a17787093cf/8acc6e3cc0b9ca6d",
    "Name": "/gnuradio",
    "RestartCount": 0,
    "Driver": "overlay2",
    "Platform": "linux",
    "MountLabel": "",
    "ProcessLabel": "",
    "AppArmorProfile": "unconfined",
    "ExecIDs": null,
    "HostConfig": {
      "Binds": [
        "/etc/localtime:/etc/localtime",
        "/etc/dconf/profile:/etc/dconf/profile",
        "/tmp/runtime-root:/tmp/runtime-root",
        "/tmp/.X11-unix:/tmp/.X11-unix",
        "/run:/run",
        "/var/log:/var/log",
        ":/host",
        "/dev/usb:/dev/usb",
        "/dev/snd:/dev/snd",
        "/home/jefferson/gnuradio:/home/jefferson/gnuradio"
      ],
      "ContainerIDFile": "",
      "LogConfig": {
        "Type": "json-file",
        "Config": {}
      },
      "NetworkMode": "host",
      "PortBindings": {},
      "RestartPolicy": {

```

Fonte: Elaborado pelo autor.

## Instalação do GNURadio

Até o momento apenas foi feito o provisionamento de um *container Docker* executando uma imagem Ubuntu e então o GNURadio será instalado utilizando o gerenciador de pacotes **apt** disponível na distribuição Ubuntu deste *container*. É recomendável atualizar a lista dos repositórios e verificar as dependências para esta instalação bastando executar os seguintes comandos no terminal do *container* criado:

```

$ apt-get update && apt-get upgrade
$ apt-get install gir1.2-gtk-3.0 libx11-dev
$ apt-get install gnuradio

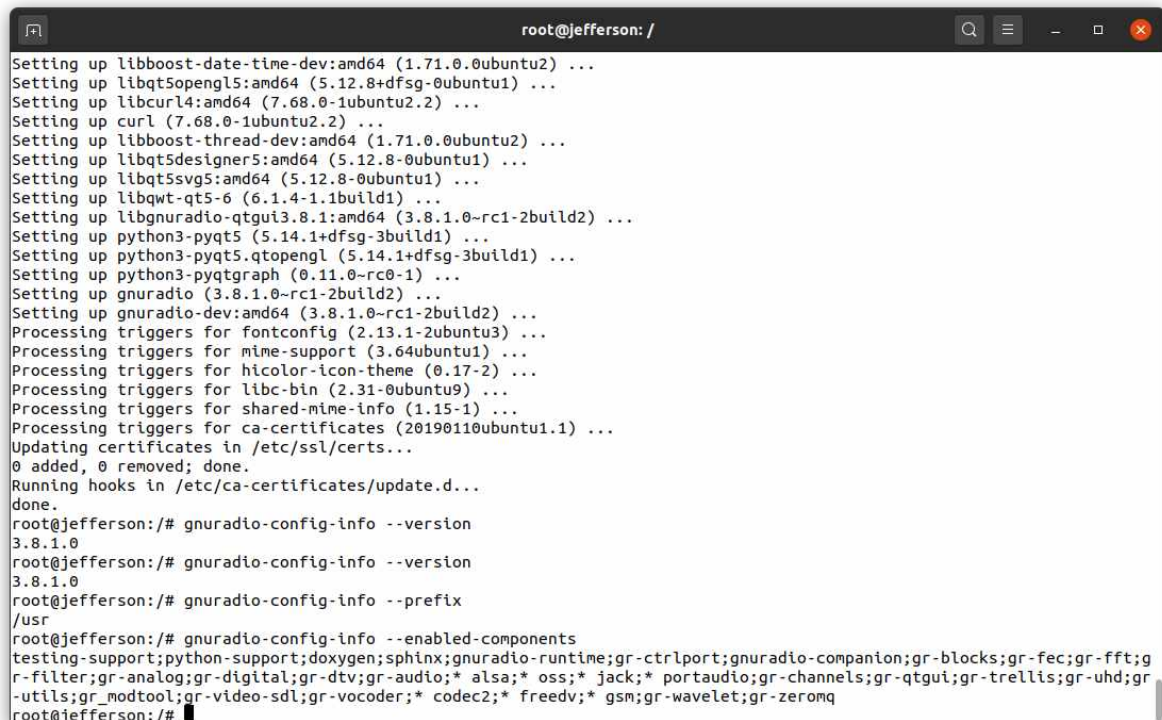
```

A partir desse ponto, o GNURadio estará instalado no *container Docker* e é possível verificar a versão, o local de instalação e os componentes já habilitados através dos seguintes

comandos que foram exemplificados na Fig. 7:

```
$ gnuradio-config-info --version
$ gnuradio-config-info --prefix
$ gnuradio-config-info --enabled-components
```

Figura 7 – Conferência de versão, *path* e componentes habilitados do GNURadio.



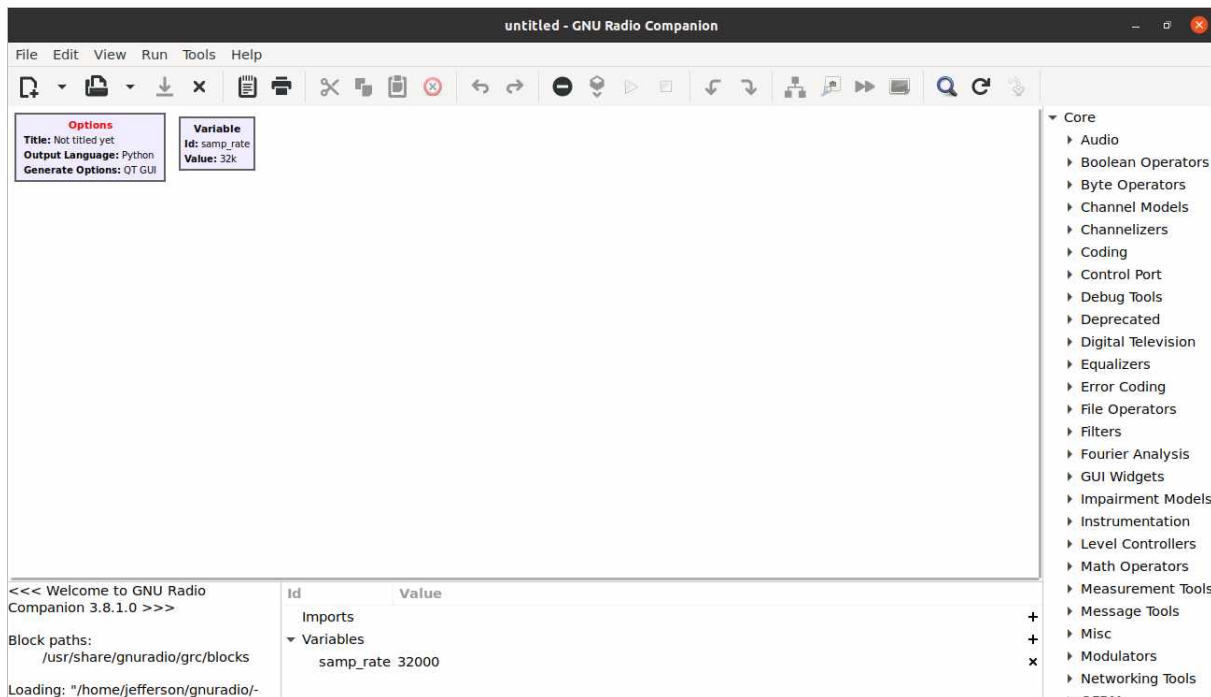
```
root@jefferson: /
Setting up libboost-date-time-dev:amd64 (1.71.0.0ubuntu2) ...
Setting up libqt5opengl5:amd64 (5.12.8+dfsg-0ubuntu1) ...
Setting up libcurl4:amd64 (7.68.0-1ubuntu2.2) ...
Setting up curl (7.68.0-1ubuntu2.2) ...
Setting up libboost-thread-dev:amd64 (1.71.0.0ubuntu2) ...
Setting up libqt5designer5:amd64 (5.12.8-0ubuntu1) ...
Setting up libqt5svg5:amd64 (5.12.8-0ubuntu1) ...
Setting up libqwt-qt5-6 (6.1.4-1.1build1) ...
Setting up libgnuradio-qtgui3.8.1:amd64 (3.8.1.0-rc1-2build2) ...
Setting up python3-pyqt5 (5.14.1+dfsg-3build1) ...
Setting up python3-pyqt5.qtopengl (5.14.1+dfsg-3build1) ...
Setting up python3-pyqtgraph (0.11.0-rc0-1) ...
Setting up gnuradio (3.8.1.0-rc1-2build2) ...
Setting up gnuradio-dev:amd64 (3.8.1.0-rc1-2build2) ...
Processing triggers for fontconfig (2.13.1-2ubuntu3) ...
Processing triggers for mime-support (3.64ubuntu1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for libc-bin (2.31-0ubuntu9) ...
Processing triggers for shared-mime-info (1.15-1) ...
Processing triggers for ca-certificates (20190110ubuntu1.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
root@jefferson:/# gnuradio-config-info --version
3.8.1.0
root@jefferson:/# gnuradio-config-info --version
3.8.1.0
root@jefferson:/# gnuradio-config-info --prefix
/usr
root@jefferson:/# gnuradio-config-info --enabled-components
testing-support;python-support;doxygen;sphinx;gnuradio-runtime;gr-ctrlport;gnuradio-companion;gr-blocks;gr-fec;gr-fft;g
r-filter;gr-analog;gr-digital;gr-dtv;gr-audio;* alsa;* oss;* jack;* portaudio;gr-channels;gr-qtgui;gr-trellis;gr-uhd;gr
-utils;gr_modtool;gr-video-sdl;gr-vocoder;* codec2;* freedv;* gsm;gr-wavelet;gr-zeromq
root@jefferson:/#
```

Fonte: Elaborado pelo autor.

Conforme ilustrado na Fig. 8, após todos os procedimentos descritos neste capítulo, o GNURadio na versão **3.8.1.0** estará disponível para o desenvolvimento de aplicações de rádio definido por *software* e de processamento digital de sinais utilizando as bibliotecas que foram instaladas no *container* ou utilizando a GUI — *Graphical User Interface*, interface gráfica de usuário — através do comando **gnuradio-companion** (GRC).

## Criação da imagem *Docker*

O *container* criado na seção anterior pode ser disponibilizado para outros desenvolvedores por meio da criação de uma imagem base. A imagem base deve receber uma *tag*, que normalmente está ligada à sua versão e, por fim, esta imagem pode ser "empurrada" para um repositório de imagens *Docker*. Neste caso, foi utilizado o *DockerHub* que é um repositório de imagens *Docker* público e hospedado e fornecido pela própria empresa *Docker* Inc. Qualquer empresa pode hospedar, manter e disponibilizar um repositório de imagens *Docker* por meio da implantação de um *Docker Registry* próprio.

Figura 8 – GUI do *GNURadio Companion*.

Fonte: Elaborado pelo autor.

Na Fig. 9, ao executar o comando **docker images** apenas a imagem **ubuntu** foi listada, pois ainda não foi criada uma imagem do container **gnuradio**, o qual pode ser conferido com o comando **docker ps -a**. Utilizando o commando **docker commit** referenciando para o container **gnuradio**, uma imagem *Docker* é criada localmente, onde as *flags* utilizadas, **-a** e **-m**, são uma descrição sobre o autor e uma breve mensagem sobre a imagem criada, respectivamente.

Figura 9 – Fazendo o *commit* da imagem base GNURadio.

```

jefferson@jefferson: ~
jefferson@jefferson:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              20.04              bb0eaf4eee00      5 days ago        72.9MB
jefferson@jefferson:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
8acc6e3cc0b9      ubuntu:20.04      "/bin/bash"        2 days ago         Exited (129) 2 days ago           gnuradio
jefferson@jefferson:~$ docker commit -a "Jefferson Cândia <jefinstz@gmail.com>" -m "GNURadio 3.8.1.0 base image" gnuradio
sha256:1c66799cf78068d50ae8707d2459ccec90b433c8e49861535d56504142637ccf
jefferson@jefferson:~$

```

Fonte: Elaborado pelo autor.

Após o *commit* da imagem, ao executar o comando **docker images** novamente, observa-se que a imagem foi criada com um **IMAGE ID** igual a **1c66799cf780** faltando apenas que seja associada a um repositório e que tenha uma *tag* de versionamento. Como exemplo foi criada a *tag* para a imagem e associada a um repositório público através do comando **docker tag 1c66799cf780 jeffcandido/gnuradio**. Ao executar **docker images** novamente, a imagem do *container* contendo a instalação do GNURadio na versão 3.8.1.0 está pronta para ser levada ao repositório de imagens *Docker* e se tornar acessível



a outros desenvolvedores. Para fazer esse procedimento, foi necessário criar uma conta (gratuita) no *DockerHub* e realizar o *login* via linha de comando e finalmente executar o *push* da imagem para levá-la até o repositório remoto, como é demonstrado nos passos executados na Fig. 10.

Figura 10 – Submetendo a imagem base ao repositório remoto.

```

jefferson@jefferson: ~
jefferson@jefferson:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              20.04              bb0eaf4eee00       5 days ago         72.9MB
jefferson@jefferson:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
8acc6e3cc0b9       ubuntu:20.04       "/bin/bash"        2 days ago         Exited (129) 2 days ago
jefferson@jefferson:~$ docker commit -a "Jefferson Cândido <jefinstz@gmail.com>" -m "GNURadio 3.8.1.0 base image" gnuradio
sha256:1c66799cf78068d56ae8707d2459ccec90b433c8e49861535d56504142637ccf
jefferson@jefferson:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>             1c66799cf780       About a minute ago 1.33GB
ubuntu              20.04              bb0eaf4eee00       5 days ago         72.9MB
jefferson@jefferson:~$ docker tag 1c66799cf780 jeffcandido/gnuradio
jefferson@jefferson:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jeffcandido/gnuradio latest             1c66799cf780       5 minutes ago     1.33GB
ubuntu              20.04              bb0eaf4eee00       5 days ago         72.9MB
jefferson@jefferson:~$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: jeffcandido
Password:
WARNING! Your password will be stored unencrypted in /home/jefferson/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
jefferson@jefferson:~$ docker push jeffcandido/gnuradio:latest
The push refers to repository [docker.io/jeffcandido/gnuradio]
6ac5ff72b0f: Pushed
128fa0b0fb81: Mounted from library/ubuntu
c0151ca45f27: Mounted from library/ubuntu
b2fd17df2071: Mounted from library/ubuntu
latest: digest: sha256:b40adbdb4f51c18d1b69c22157807a0073797f4fa6dff5b8a3126a3698142a96 size: 1156
jefferson@jefferson:~$

```

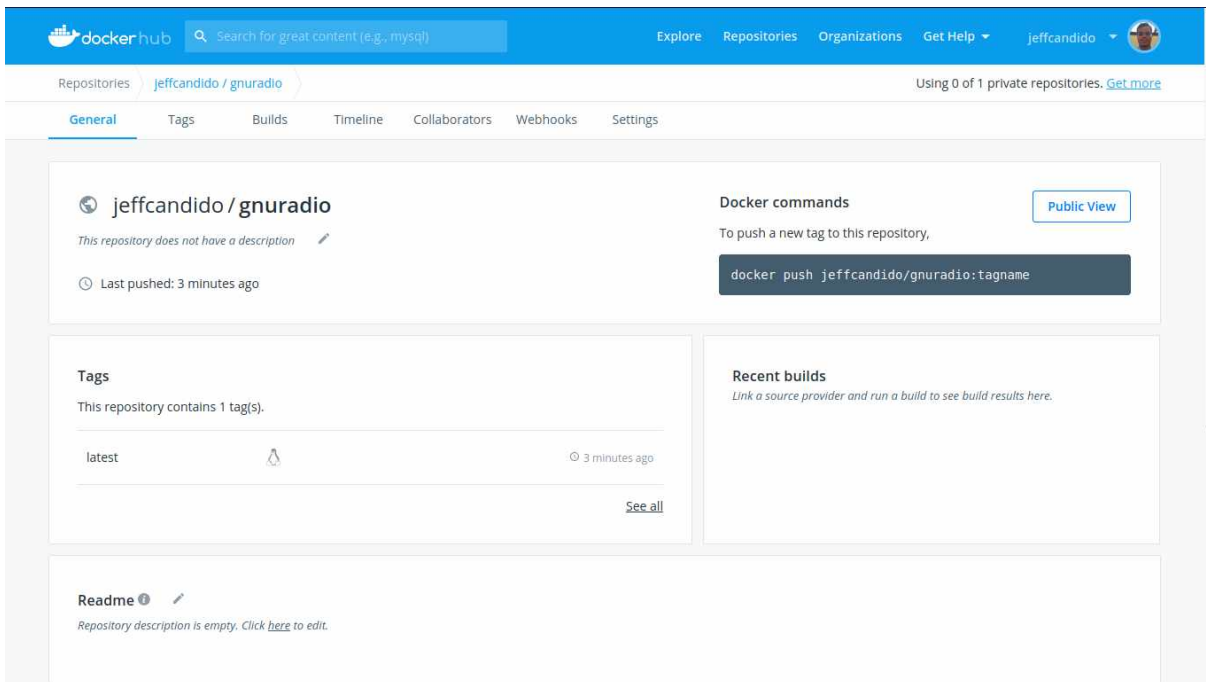
Fonte: Elaborado pelo autor.

A imagem que foi criada e submetida ao repositório remoto agora pode ser visualizada pela interface *web* do *DockerHub* (Fig. 11) e qualquer usuário conectado à *Internet* poderá buscar e utilizar essa imagem.

Algumas otimizações poderiam ser feitas com relação ao processo de criação do *container* e geração de imagem *Docker*, como por exemplo a criação de arquivos para composição do *container* (*Dockerfile*, *docker-compose.yaml* e *.dockerignore*). Entretanto, isto não adicionaria vigor e relevância ao propósito deste trabalho, de forma que tais otimizações podem ser realizadas em trabalhos futuros nessa mesma área de estudo.

## Instalação do firmware do *HackRF One*

A forma de instalação do *firmware* e utilitários do *HackRF One* recomendada pelo fabricante é utilizando o gerenciador de pacotes disponibilizado pelo sistema operacional que neste caso é o *apt*, porém o *software* a ser executado na máquina *host* também pode ser compilado através do código-fonte [18]. Foi necessário instalar o *HackRF One* como um periférico de rádio definido por *software* com o apoio dos pacotes **hackrf** (utilitários), **libhackrf-dev** (biblioteca para desenvolvimento), **libhackrf0** (biblioteca para tempo

Figura 11 – Interface *web* do *DockerHub* com a imagem *Docker* criada.

Fonte: Elaborado pelo autor.

de execução), **gr-osmosdr** (Blocos do GNURadio do projeto OsmoSDR) e **gqrx-sdr** (receptor de rádio definido por *software*).

```
$ apt-get install hackrf libhackrf-dev libhackrf0 gqrx-sdr gr-osmosdr
```

Para atualizar o firmware do *HackRF One* basta baixar a versão desejada diretamente do repositório oficial do projeto e seguir os passos definidos na documentação [19] que nada mais é que escrever o executável na memória *flash* da placa (*hackrf\_spiflash -w hackrf\_one\_usb.bin*). Realizado este procedimento, é possível verificar que o dispositivo foi instalado corretamente utilizando os comandos **hackrf\_info** e **hackrf\_transfer** [20], conforme mostra a Fig. 12.

Figura 12 – Verificação de versão do *firmware* e taxa de transmissão do *HackRF One*.

```
root@jefferson: /
jefferson@jefferson:~$ docker exec -it gnuradio /bin/bash
root@jefferson:/# hackrf_info
hackrf_info version: git-52851ee
libhackrf version: git-52851ee (0.5)
Found HackRF
Index: 0
Serial number: 000000000000000026b468dc2e9d768f
Board ID Number: 2 (HackRF One)
Firmware Version: 2017.02.1 (API:1.02)
Part ID Number: 0xa000cb3c 0x00544368
root@jefferson:/# hackrf_transfer -r /dev/null -s 20000000
call hackrf_set_sample_rate(20000000 Hz/20.000 MHz)
call hackrf_set_hw_sync_mode(0)
call hackrf_set_freq(900000000 Hz/900.000 MHz)
Stop with Ctrl-C
39.8 MiB / 1.000 sec = 39.8 MiB/second
40.1 MiB / 1.000 sec = 40.1 MiB/second
39.8 MiB / 1.000 sec = 39.8 MiB/second
40.1 MiB / 1.000 sec = 40.1 MiB/second
40.1 MiB / 1.000 sec = 40.1 MiB/second
39.8 MiB / 1.000 sec = 39.8 MiB/second
40.1 MiB / 1.000 sec = 40.1 MiB/second
40.1 MiB / 1.000 sec = 40.1 MiB/second
39.8 MiB / 1.000 sec = 39.8 MiB/second
^Ccaught signal 2
 4.2 MiB / 0.104 sec = 40.4 MiB/second

Exiting...
Total time: 9.10532 s
hackrf_stop_rx() done
hackrf_close() done
hackrf_exit() done
fclose(fd) done
exit
root@jefferson:/#
```

Fonte: Elaborado pelo autor.



Parte II

Desenvolvimento



## 3 Criação de *flowgraph*'s no GNURadio

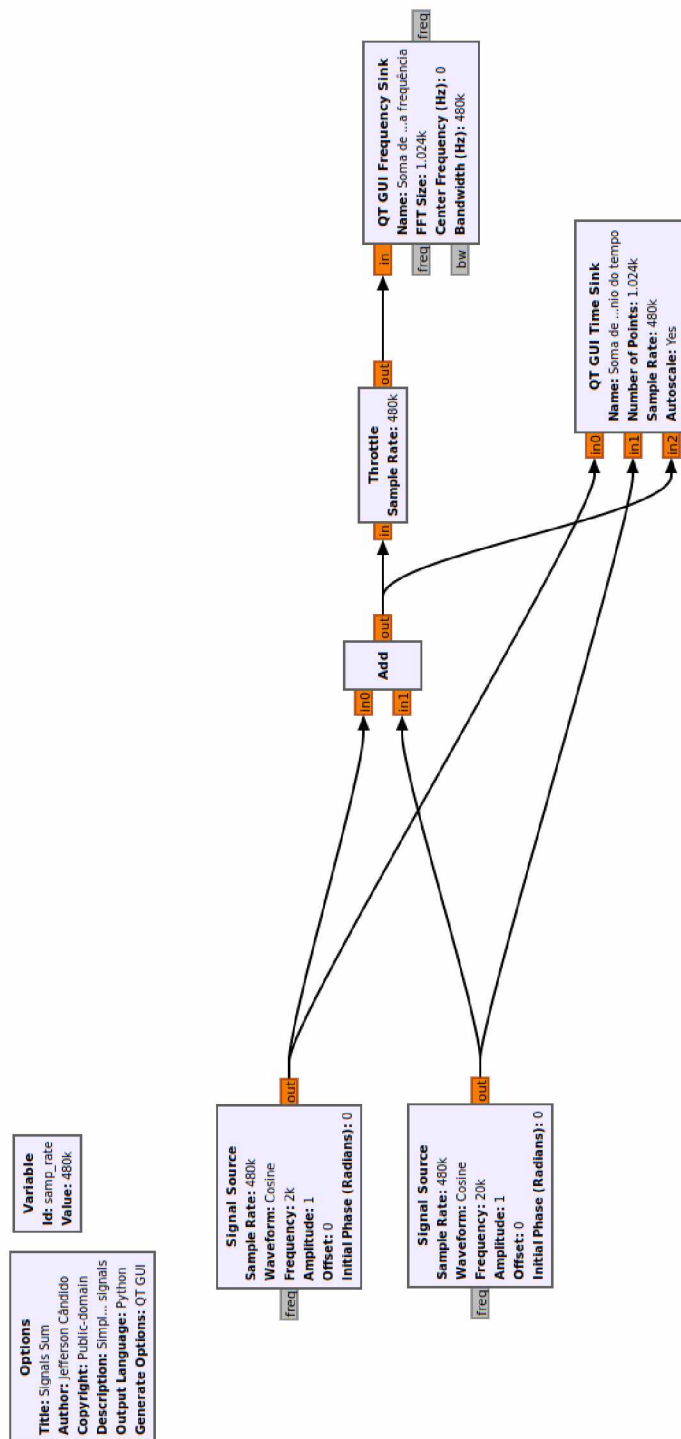
Neste ponto o objetivo é demonstrar fluxos simples de processamento digital de sinais utilizando os GNURadio e nada mais justo que iniciar esta parte do trabalho desmonstrando *flowgraph*'s de operações matemáticas básicas aplicadas a sinais digitais, assim como utilizações convenientes do processamento de sinais.

Para o primeiro *flowgraph* foram posicionadas duas fontes de sinal de frequências distintas (2 kHz e 20 kHz) e suas saídas foram agregadas em um bloco de soma de sinais. Após isso, para que as formas de onda dos sinais possam ser visualizadas, foram adicionados os blocos *QT GUI* para apresentação dos sinais nos domínios do tempo (*Time Sink*) e da frequência (*Frequency Sink*). O bloco *Throttle* normalmente é utilizado na saída de blocos que não são de origem diretamente de *hardware*, com o objetivo de limitar a taxa na qual esse bloco cria as amostras.

Ao executar o código gerado pelo *flowgraph* da Fig. 13, a interface do *GNURadio-companion* constrói na Fig. 14 os gráficos relacionados à soma dos sinais, tanto no domínio do tempo como no domínio da frequência. De forma semelhante, na Fig. 15 está o *flowgraph* para operação do produto de dois sinais (de frequências 10 kHz e 100 kHz) que podem ser visualizados pela interface gráfica na Fig. 16.

Foram criados também dois *flowgraph*'s, apresentados nas Figs. 17 e 19, que têm objetivos bastante conhecidos por engenheiros em eletrônica e telecomunicações, que são o entendimento sobre a defasagem entre sinais (Figs. 17 e 18) e também o conceito de histerése (Figs. 19 20) utilizado na comparação de sinais, principalmente aqueles sobre o efeito de ruído.

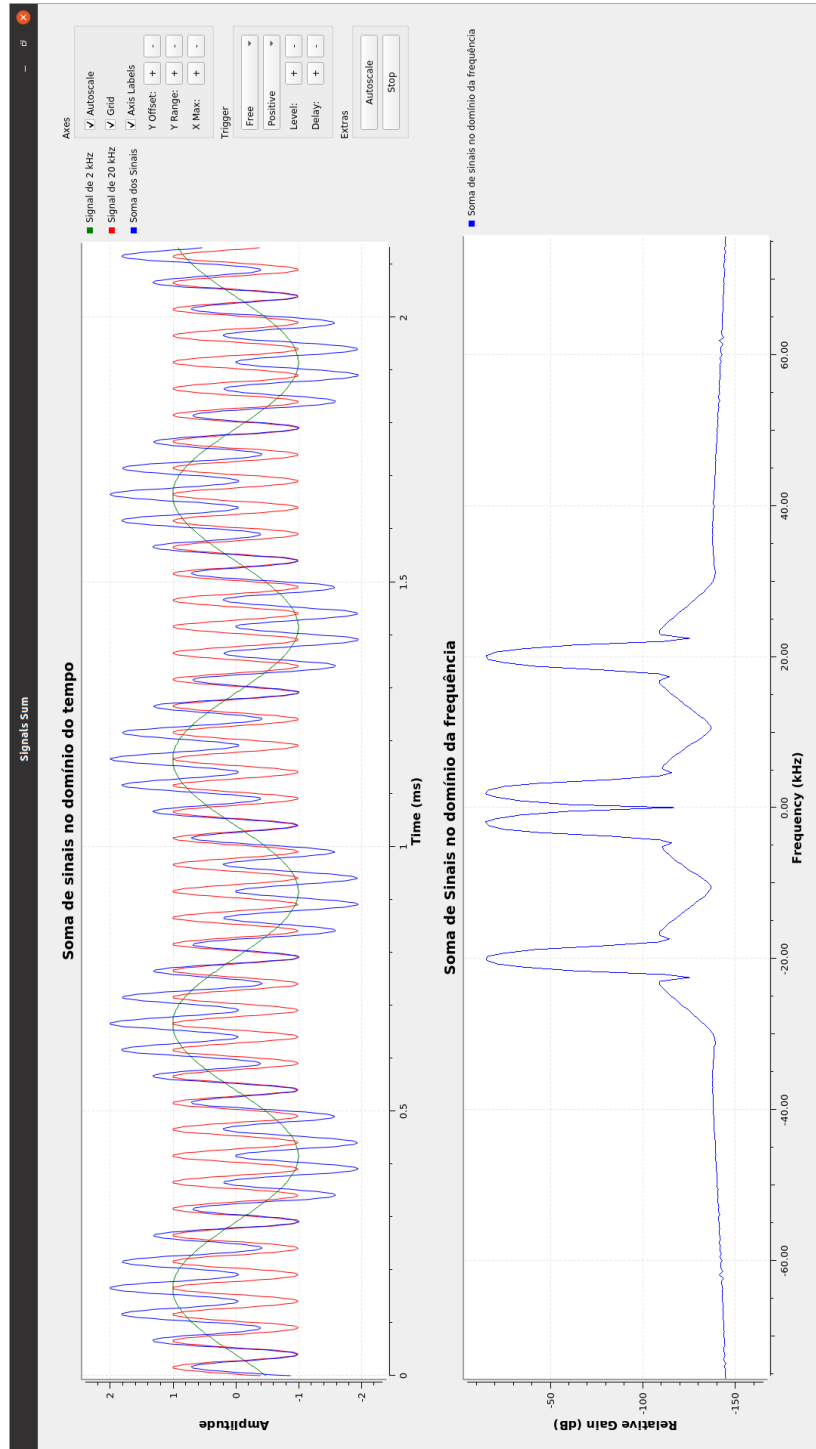
Figura 13 – Flowgraph da soma de dois sinais.



Fonte: Elaborado pelo autor.

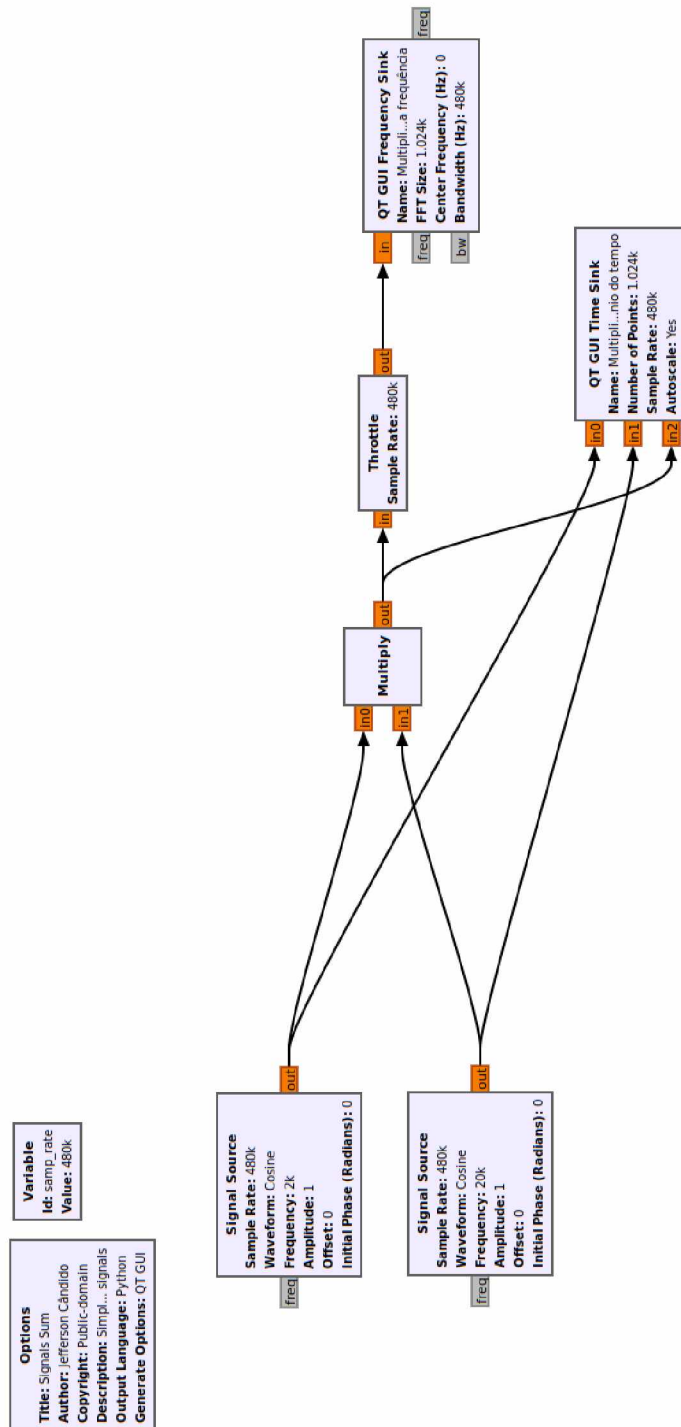


Figura 14 – Gráficos da soma de dois sinais no GNURadio.



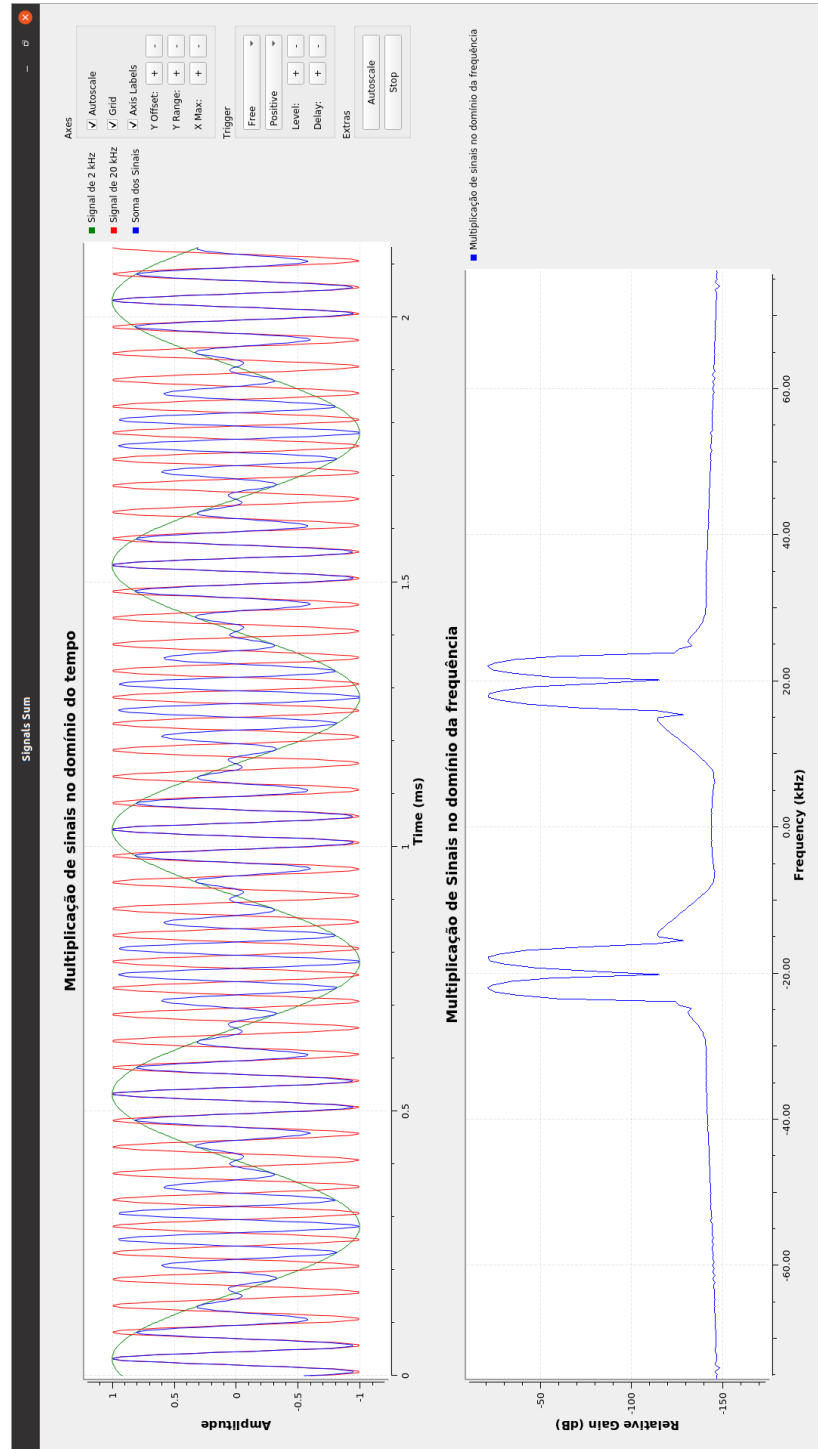
Fonte: Elaborado pelo autor.

Figura 15 – Flowgraph do produto de dois sinais.

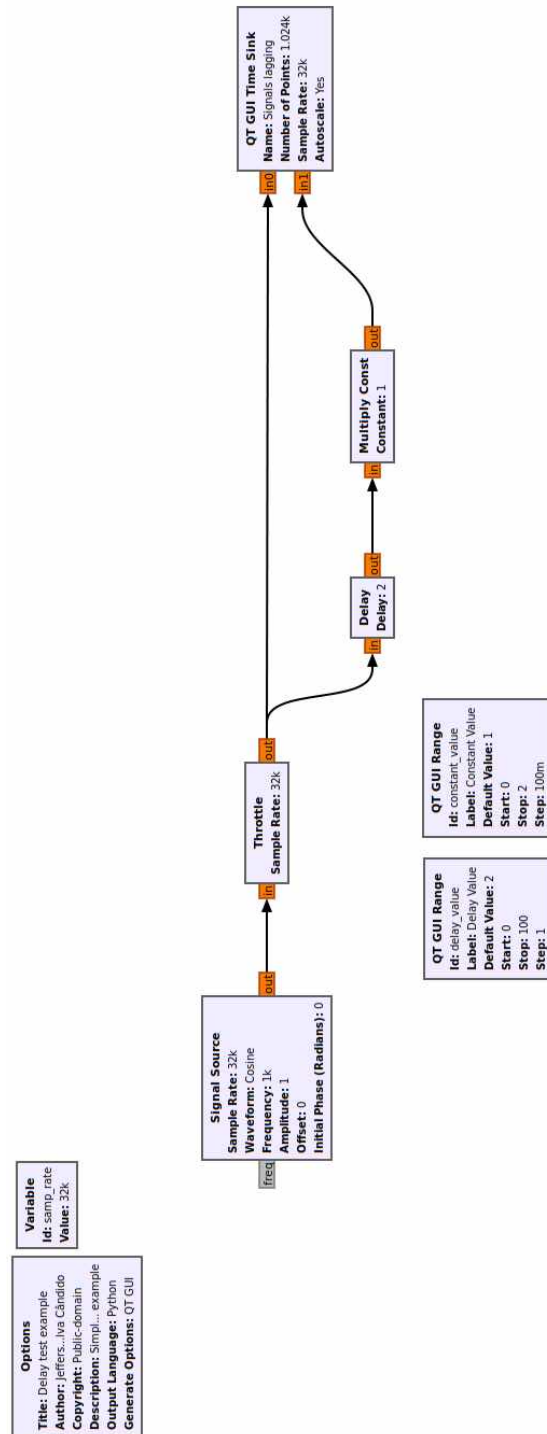


Fonte: Elaborado pelo autor.

Figura 16 – Gráficos do produto de dois sinais no GNURadio.

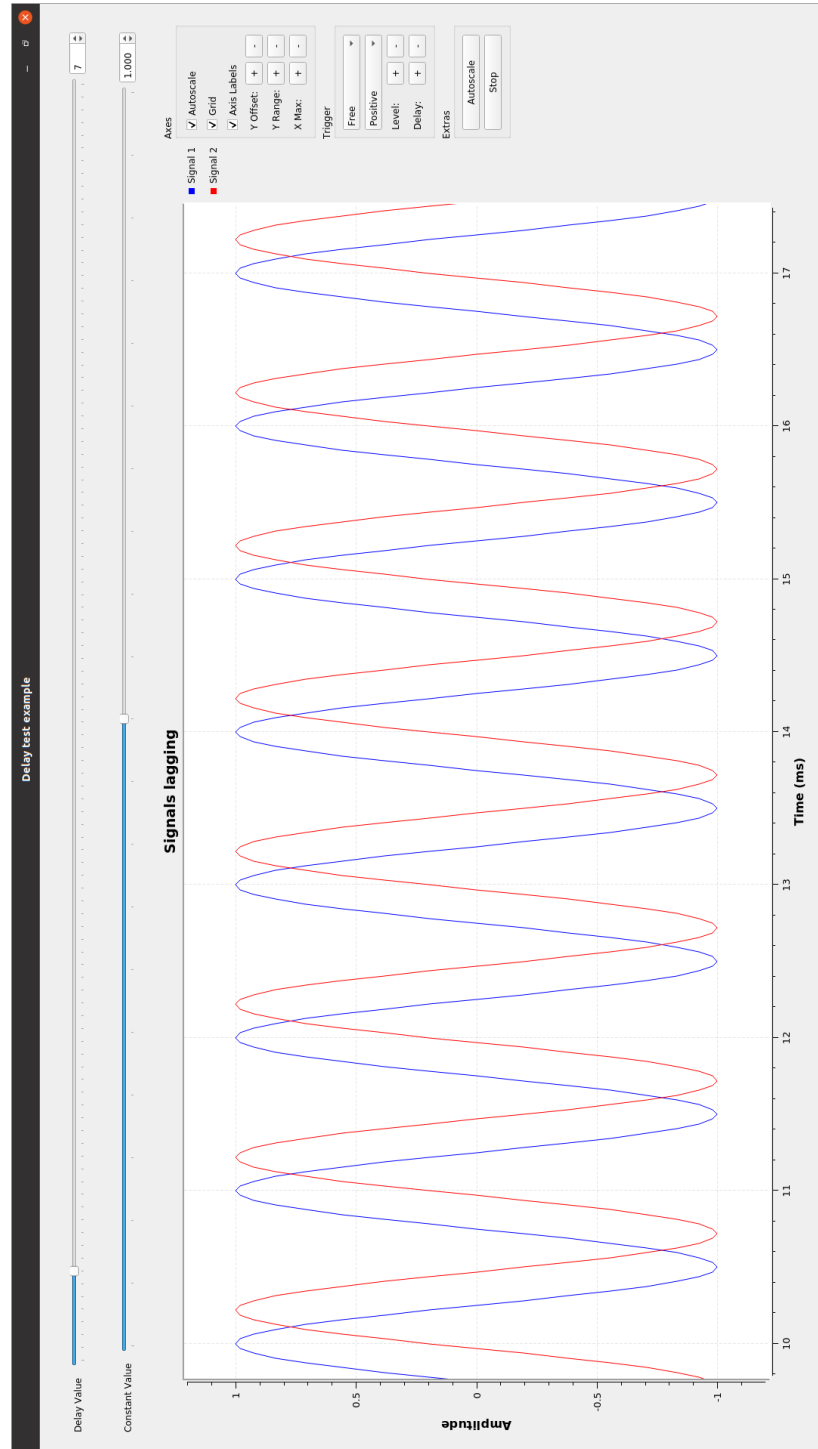


Fonte: Elaborado pelo autor.

Figura 17 – *Flowgraph* para geração de dois sinais defasados no tempo.

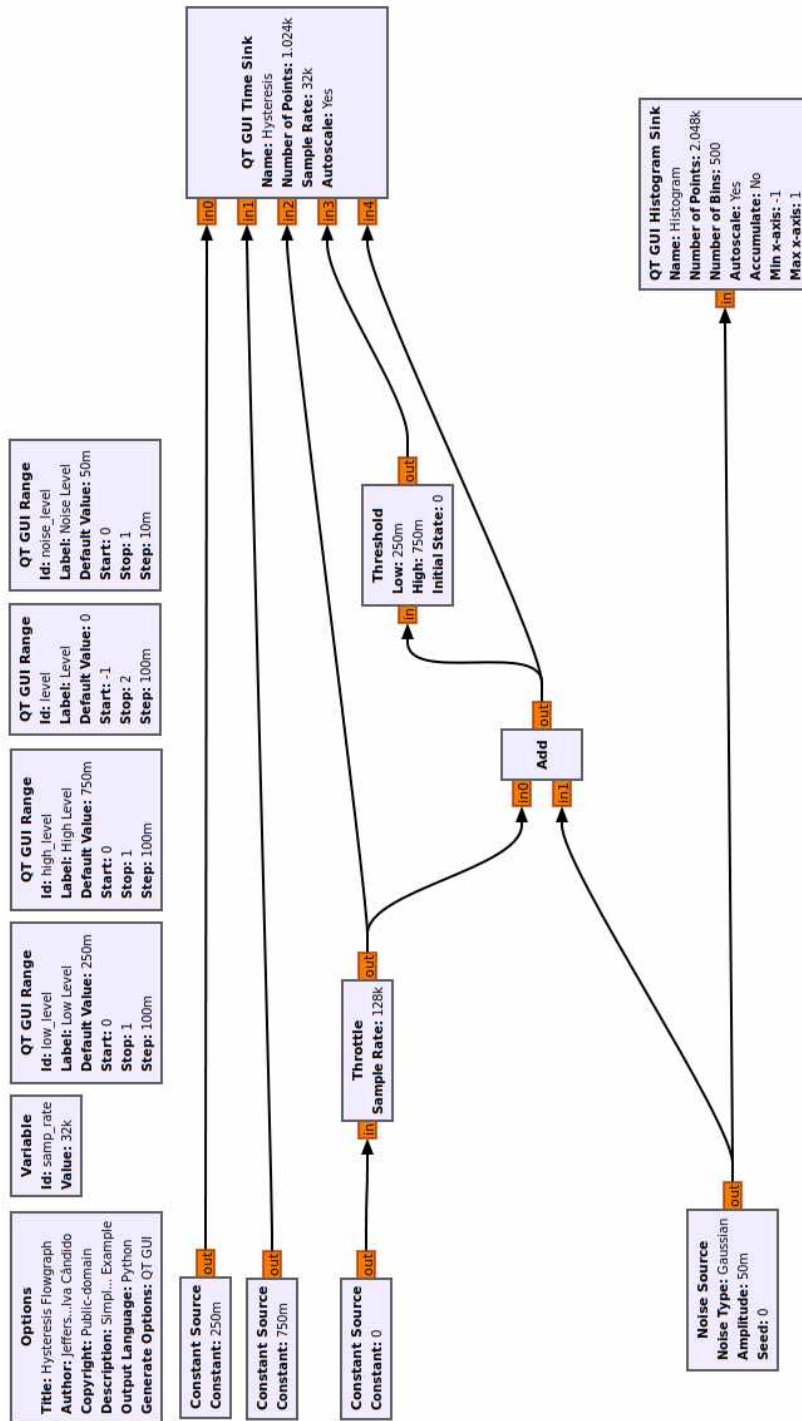
Fonte: Elaborado pelo autor.

Figura 18 – Sinais defasados no domínio do tempo.



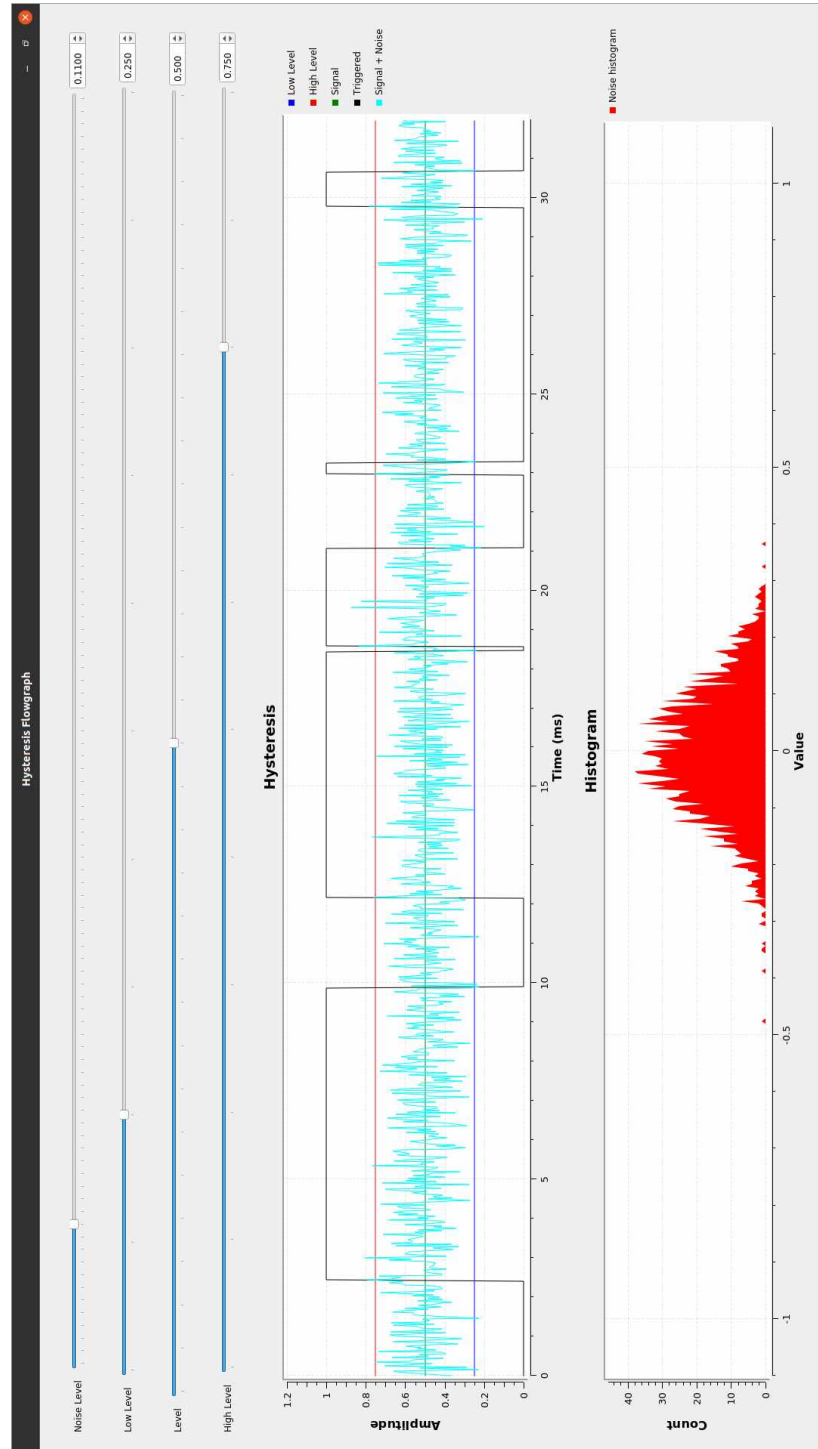
Fonte: Elaborado pelo autor.

Figura 19 – Flowgraph para simulação de um comparador de sinais com histerése.



Fonte: Elaborado pelo autor.

Figura 20 – Comparação de sinais com histerese.



Fonte: Elaborado pelo autor.



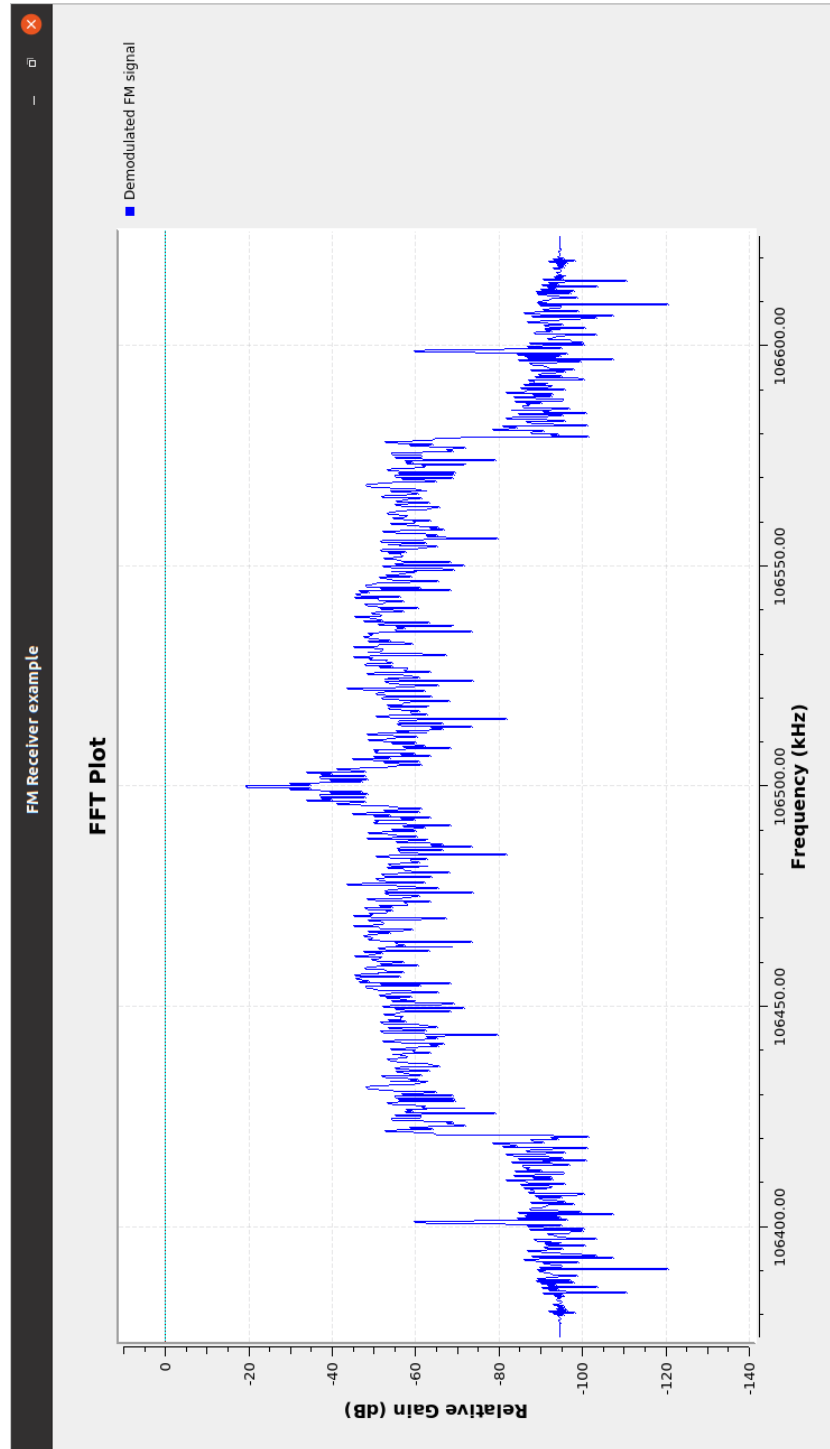


## 4 O *Hello-World* do GNURadio

Neste capítulo será reproduzido o *flowgraph* que é considerado o "*hello-world*" do GNURadio com o hardware do SDR: o receptor de rádio FM. Trata-se de um *flowgraph* bastante simples, pois todos os blocos necessários já foram disponibilizados anteriormente nos procedimentos de instalação do GNURadio e do *firmware* do *HackRF One*.

O sinal de rádio FM recebido pelo *hardware* do *HackRF One* sintonizado na frequência portadora de 106.5 MHz (um canal de rádio local com programação ativa na cidade de Uberlândia, Minas Gerais) passa por um filtro passa-baixas (LPF - *Low-Pass Filter*) de ganho unitário com resposta finita ao impulso unitário (FIR - *Finite Impulse Response*), frequência de corte de 75 kHz, largura de banda de transição de 25 kHz (com janela de Hamming) e fator de redução da frequência de amostragem (decimação) de 20:1. Em cascata o sinal é convoluído com um filtro FIR polifásico de reamostragem racional (Rational Resampler) com fator da razão entre as constantes de interpolação e decimação do sinal igual a  $\frac{12}{5}$  e, por fim, é demodulado e novamente decimado para que possa ser consumido pelo bloco *Audio Sink* (reprodução do áudio nos alto-falantes do computador) e que seu espectro seja apresentado utilizando um bloco coletor gráfico para exibição de sinais em frequência (via FFT - *Fast Fourier Transform*), o *QT GUI Frequency Sink*.



Figura 22 – *FFT plot* extraída durante simulação do receptor FM.

Fonte: Elaborado pelo autor.

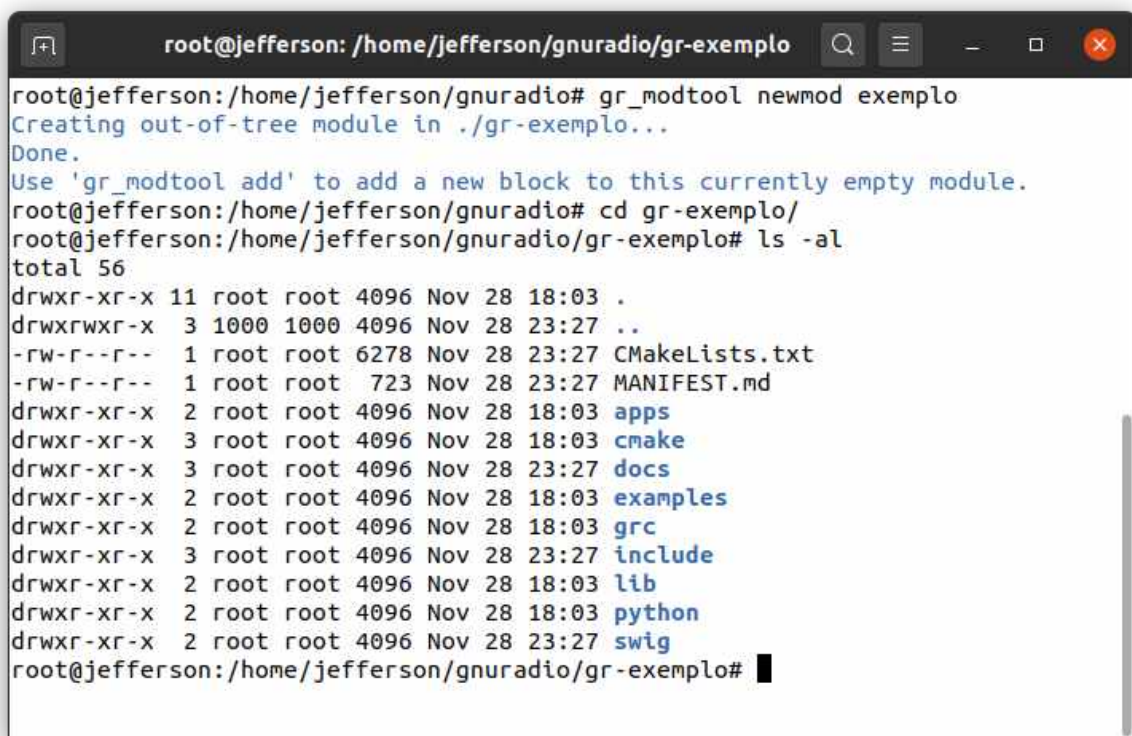


## 5 Criando blocos customizados

O objetivo deste capítulo é disponibilizar uma visão geral de como é possível que qualquer um seja também um desenvolvedor de módulos (os quais são chamados de módulos *OOT - Out of Tree* [21]) de processamento digital de sinais no GNURadio e não apenas usuários da ferramenta *GNURadio-companion*. Para auxiliar no entendimento, será mostrado um passo a passo de como criar um módulo do GNURadio, adicionar blocos a ele e depois disponibilizar pacotes de instalação que podem ser aproveitados por outros desenvolvedores que estejam direcionados a esse mesmo propósito.

### Módulos *OutOfTree*

Figura 23 – Geração de um módulo *OOT* do GNURadio e sua estrutura de arquivos e pastas.



```

root@jefferson: /home/jefferson/gnuradio/gr-exemplo
root@jefferson:/home/jefferson/gnuradio# gr_modtool newmod exemplo
Creating out-of-tree module in ./gr-exemplo...
Done.
Use 'gr_modtool add' to add a new block to this currently empty module.
root@jefferson:/home/jefferson/gnuradio# cd gr-exemplo/
root@jefferson:/home/jefferson/gnuradio/gr-exemplo# ls -al
total 56
drwxr-xr-x 11 root root 4096 Nov 28 18:03 .
drwxrwxr-x  3 1000 1000 4096 Nov 28 23:27 ..
-rw-r--r--  1 root root 6278 Nov 28 23:27 CMakeLists.txt
-rw-r--r--  1 root root  723 Nov 28 23:27 MANIFEST.md
drwxr-xr-x  2 root root 4096 Nov 28 18:03 apps
drwxr-xr-x  3 root root 4096 Nov 28 18:03 cmake
drwxr-xr-x  3 root root 4096 Nov 28 23:27 docs
drwxr-xr-x  2 root root 4096 Nov 28 18:03 examples
drwxr-xr-x  2 root root 4096 Nov 28 18:03 grc
drwxr-xr-x  3 root root 4096 Nov 28 23:27 include
drwxr-xr-x  2 root root 4096 Nov 28 18:03 lib
drwxr-xr-x  2 root root 4096 Nov 28 18:03 python
drwxr-xr-x  2 root root 4096 Nov 28 23:27 swig
root@jefferson:/home/jefferson/gnuradio/gr-exemplo#

```

Fonte: Elaborado pelo autor.

Primeiramente, um módulo fora da árvore (*OOT module*) recebe esse nome por se tratar de um "pedaço" de *software* que pode fazer uso das bibliotecas do GNURadio, mas separado dos módulos que já foram disponibilizados em uma versão oficial e já se encontram na árvore do GNURadio. Neste módulo o engenheiro pode desenvolver blocos

com objetivos específicos sem haver a necessidade de alterar o que já existe na raiz do projeto do GNURadio, de forma que esse tipo de ramificação permite que o próprio desenvolvedor mantenha o código e disponibilize-o da forma que achar conveniente.

A Fig. 23 mostra a execução da ferramenta `gr_modtool` para a criação de um módulo de exemplo (chamado de *exemplo*), o qual conterà um bloco de processamento digital de sinais que será responsável por simplesmente receber amostras do tipo ponto flutuante e entregar em sua saída o valor da entrada elevado à segunda potência. Nesta mesma figura é mostrada a estrutura de pastas criada pelo comando `gr_modtool`.

Uma linha de comando simples utilizando a ferramenta `gr_modtool` cria toda a estrutura de arquivos e pastas de um novo módulo do GNURadio. Os arquivos dos códigos escritos em *python* devem ir para a pasta `python/`, os arquivos escritos em C++ devem ser colocados na pasta `lib/` e os seus arquivos de cabeçalho (`*.h`) devem ir para a pasta `include/`. Também é criada uma pasta relacionada ao `swig`, que é um *wrapper* (um empacotador) simplificado e gerador de interfaces que cria automaticamente o código em *python* dos blocos, mesmo que eles tenham sido escritos em C++.

Figura 24 – Criação dos arquivos de um bloco customizado usando `gr_modtool`.



```

root@jefferson: /home/jefferson/gnuradio/gr-exemplo
root@jefferson:/home/jefferson/gnuradio/gr-exemplo# gr_modtool add -t general
-l cpp ao_quadrado_ff
GNU Radio module name identified: exemplo
Language: C++
Block/code identifier: ao_quadrado_ff
Please specify the copyright holder: Jefferson da Silva Cândido
Enter valid argument list, including default arguments:

Add Python QA code? [Y/n] Y
Add C++ QA code? [y/N] Y
Adding file 'lib/ao_quadrado_ff_impl.h'...
Adding file 'lib/ao_quadrado_ff_impl.cc'...
Adding file 'include/exemplo/ao_quadrado_ff.h'...
Adding file 'lib/qa_ao_quadrado_ff.cc'...
Editing swig/exemplo_swig.i...
Adding file 'python/qa_ao_quadrado_ff.py'...
Editing python/CMakeLists.txt...
Adding file 'grc/exemplo_ao_quadrado_ff.block.yml'...
Editing grc/CMakeLists.txt...
root@jefferson:/home/jefferson/gnuradio/gr-exemplo#

```

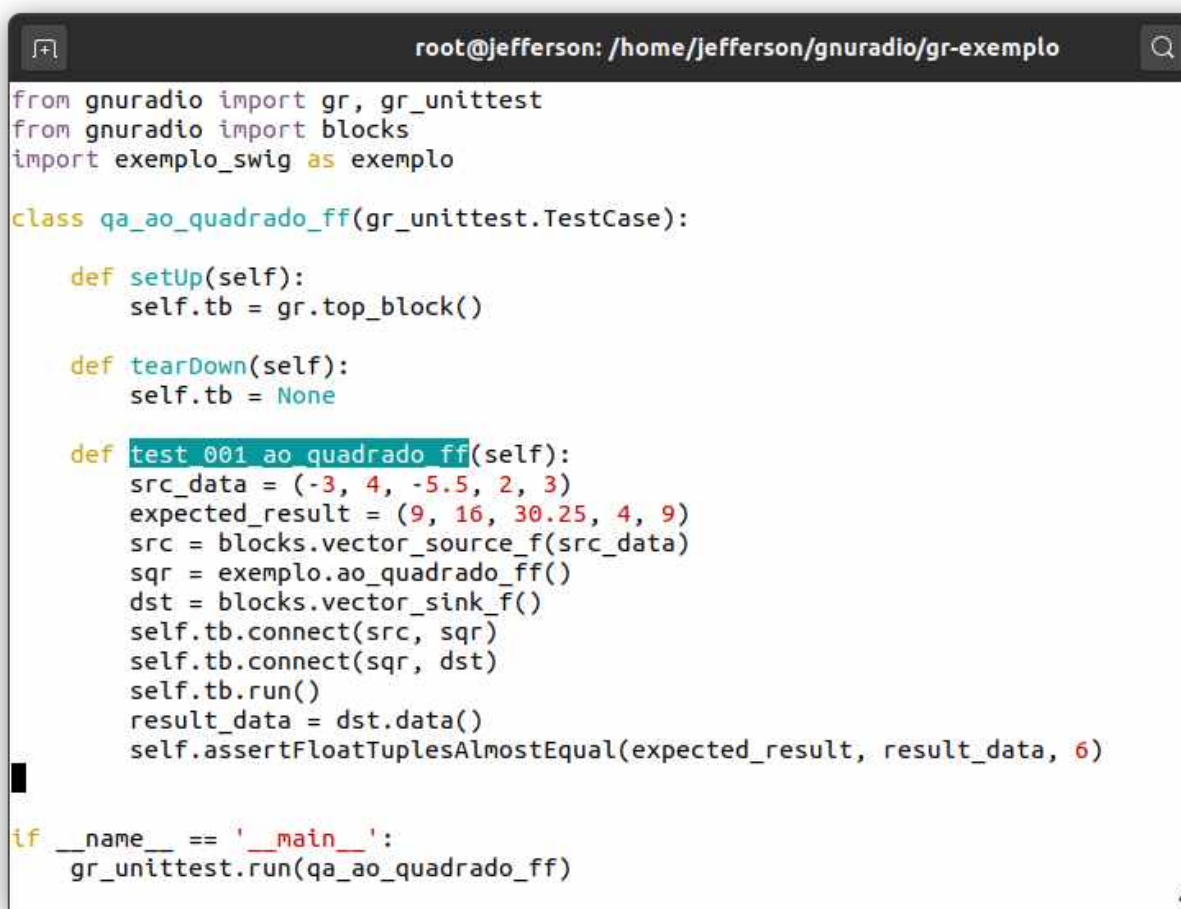
Fonte: Elaborado pelo autor.

Para disponibilizar os blocos via interface gráfica do *GNURadio-companion*, basta adicionar suas descrições na pasta `grc/` (na versão 3.8 são arquivos `.yaml` e antes da versão 3.8 eram arquivos `.xml`). A pasta `docs/` contém, intuitivamente, as instruções sobre

como as documentações devem ser extraídas dos arquivos C++ e Python utilizando a bibliotecas *Doxygen* e *Sphinx*. Para compilação do código, o desenvolvedor deve se atentar ao arquivo *CMakeLists.txt* e à pasta **cmake/** para garantir as instruções a serem passadas para o programa *Cmake* sobre como encontrar as bibliotecas necessárias e também garantir que o módulo seja compilado corretamente.

O passo seguinte (ilustrado na Fig. 24) foi criar os arquivos relacionados a um bloco de exemplo, o qual foi nomeado *ao\_quadrado\_ff*, também através de uma linha de comando simples utilizando a ferramenta **gr\_modtool**. A nomeação do bloco foi escolhida seguindo uma convenção de nomenclatura dos blocos que consiste na concatenação do nome customizado do bloco com os indicadores de quais são os tipos de que serão recebidos nas entradas e quais serão entregues na saída após serem processados ("*ao\_quadrado\_*" + "*float float*"). As flags passadas estão relacionadas ao tipo do bloco (-t *general*, bloco de uso geral) e o tipo de linguagem (-l *cpp*) a ser usada para escrever suas funções.

Figura 25 – Prática do desenvolvimento guiado por testes.



```

root@jefferson: /home/jefferson/gnuradio/gr-exemplo
from gnuradio import gr, gr_unittest
from gnuradio import blocks
import exemplo_swig as exemplo

class qa_ao_quadrado_ff(gr_unittest.TestCase):

    def setUp(self):
        self.tb = gr.top_block()

    def tearDown(self):
        self.tb = None

    def test_001_ao_quadrado_ff(self):
        src_data = (-3, 4, -5.5, 2, 3)
        expected_result = (9, 16, 30.25, 4, 9)
        src = blocks.vector_source_f(src_data)
        sqr = exemplo.ao_quadrado_ff()
        dst = blocks.vector_sink_f()
        self.tb.connect(src, sqr)
        self.tb.connect(sqr, dst)
        self.tb.run()
        result_data = dst.data()
        self.assertFloatTuplesAlmostEqual(expected_result, result_data, 6)

if __name__ == '__main__':
    gr_unittest.run(qa_ao_quadrado_ff)

```

Fonte: Elaborado pelo autor.

Criar blocos customizados do GNURadio é uma ótima oportunidade para colocar em prática os passos gerais de uma metodologia de desenvolvimento guiada por testes

(TDD - *Test Driven Development*) que são: escrever um teste, compilá-lo e fazê-lo rodar e depois realizar correções para tornar o código correto [22]. A Fig. 25 traz o arquivo `python/qa_ao_quadrado_ff.py` onde foi adicionada a função `test_001_ao_quadrado_ff` que possui amostras de dados de entrada e dados que são esperados na saída para realizar comparações com os resultados gerados pelo código escrito.

Com relação aos arquivos do código em C++, basta editar o conteúdo da classe de implementação do bloco (`lib/ao_quadrado_ff_impl.cc`) e seus arquivos de cabeçalho (`lib/ao_quadrado_ff_impl.h` `include/exemplo/ao_quadrado_ff.h`) quando necessário. De forma resumida, na implementação do código em C++ o desenvolvedor terá que descrever no construtor da classe a "assinatura" das entradas e saídas do bloco, ou seja, quantas entradas de um determinado tipo e quantas saídas, também com a definição de seus tipos, serão reservadas conforme ilustra a Fig. 26. Outro ponto importante é que na função `forecast` deve ser informada a relação entre quantos itens são necessários na entrada para gerar uma quantidade específica de itens na saída.

Figura 26 – Classe do bloco de processamento digital de sinais customizado.

```

root@jefferson: /home/jefferson/gnuradio/gr-exemplo
/* The private constructor
 */
ao_quadrado_ff_impl::ao_quadrado_ff_impl()
: gr::block("ao_quadrado_ff",
           gr::io_signature::make(1, 1, sizeof(float)),
           gr::io_signature::make(1, 1, sizeof(float)))
{}

/*
 * Our virtual destructor.
 */
ao_quadrado_ff_impl::~ao_quadrado_ff_impl()
{
}

void
ao_quadrado_ff_impl::forecast (int noutput_items, gr_vector_int &ninput_items_required)
{
    ninput_items_required[0] = noutput_items;
}

int
ao_quadrado_ff_impl::general_work (int noutput_items,
                                   gr_vector_int &ninput_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 i = 0; i < noutput_items; i++) {
        out[i] = in[i] * in[i];
    }

    // Do <+signal processing+>
    // Tell runtime system how many input items we consumed on
    // each input stream.
    consume_each (noutput_items);
}

-- INSERT --


```

Fonte: Elaborado pelo autor.



Agora, para tornar o bloco disponível através da interface gráfica do *GNURadio-companion* é necessário gerar o arquivo **.yaml** que contém as descrições necessárias do bloco e este procedimento também é realizável utilizando a ferramenta *gr\_modtool*.

Figura 27 – Geração do arquivo de descrição do bloco na GUI.



```

root@jefferson: /home/jefferson/gnuradio/gr-exemplo
root@jefferson:/home/jefferson/gnuradio/gr-exemplo# gr_modtool makeyaml ao_quadrado_ff
GNU Radio module name identified: exemplo
Warning: This is an experimental feature. Don't expect any magic.
Searching for matching files in lib/:
Making GRC bindings for lib/ao_quadrado_ff_impl.cc...
Overwrite existing GRC file? [y/N] y
root@jefferson:/home/jefferson/gnuradio/gr-exemplo# cat grc/exemplo_ao_quadrado_ff.block.yaml
id: exemplo_ao_quadrado_ff
label: ao quadrado ff
category: '[Exemplo]'
templates:
  imports: import exemplo
  make: exemplo.ao_quadrado_ff()
inputs:
- label: in
  domain: stream
  dtype: float
outputs:
- label: out
  domain: stream
  dtype: float
file_format: 1
root@jefferson:/home/jefferson/gnuradio/gr-exemplo#

```

Fonte: Elaborado pelo autor.

Para finalizar, é necessário realizar alguns procedimentos para compilar o código, gerar os arquivos binários executáveis e também os pacotes (**.deb** e **.rpm**) que tornam possível a disponibilização do trabalho realizado para outros desenvolvedores através dos gerenciadores de pacotes dos sistemas operacionais. Estes procedimentos são muito importantes, pois um grande diferencial do engenheiro é ir além da função de programar as aplicações, estando preparado para atuar em todos os passos do processo de criação, implantação e suporte do *software*, o qual pode ser comercializado ou também uma contribuição de *software* livre.

Para tornar possível a geração de um pacote **.deb** do bloco customizado, foi necessário adicionar ao final do arquivo *CMakeLists.txt* que fica na raiz do módulo (**gr-exemplo**) as três seguintes linhas:

```

SET(CPACK_GENERATOR "DEB")
SET(CPACK_DEBIAN_PACKAGE_MAINTAINER "Jefferson da Silva Candido")
INCLUDE(CPack)

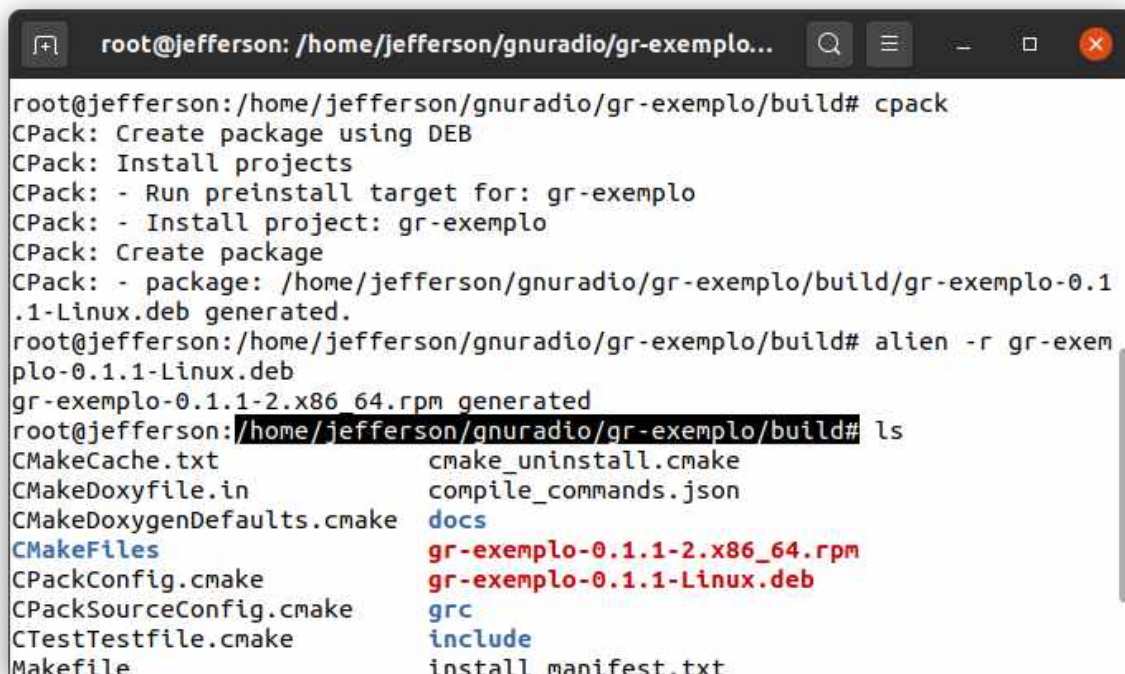
```

O procedimento de geração de uma *release* do *software*, quando configurado corretamente, segue um conjunto de passos simples de serem executados, os quais consistem em criar um diretório dentro da raiz do projeto e dentro dele realizar a construção dos

arquivos para produção, a execução dos testes unitários e de integração do código e a instalação ao final.

```
$ mkdir build
$ cd build/
$ cmake ../
$ make
$ make test
$ make install
```

Figura 28 – Geração dos pacotes `.deb` e `.rpm`.



```
root@jefferson: /home/jefferson/gnuradio/gr-exemplo...
root@jefferson:/home/jefferson/gnuradio/gr-exemplo/build# cpack
CPack: Create package using DEB
CPack: Install projects
CPack: - Run preinstall target for: gr-exemplo
CPack: - Install project: gr-exemplo
CPack: Create package
CPack: - package: /home/jefferson/gnuradio/gr-exemplo/build/gr-exemplo-0.1.1-Linux.deb generated.
root@jefferson:/home/jefferson/gnuradio/gr-exemplo/build# alien -r gr-exemplo-0.1.1-Linux.deb
gr-exemplo-0.1.1-2.x86_64.rpm generated
root@jefferson:/home/jefferson/gnuradio/gr-exemplo/build# ls
CMakeCache.txt                cmake_uninstall.cmake
CMakeDoxyfile.in             compile_commands.json
CMakeDoxygenDefaults.cmake   docs
CMakeFiles                   gr-exemplo-0.1.1-2.x86_64.rpm
CPackConfig.cmake            gr-exemplo-0.1.1-Linux.deb
CPackSourceConfig.cmake      grc
CTestTestfile.cmake          include
Makefile                      install_manifest.txt
```

Fonte: Elaborado pelo autor.

Os pacotes que podem ser instalados em sistemas compatíveis com as distribuições *Debian* e *CentOS* foram gerados com o auxílio dos programas `cpack` e `alien` como mostra a Fig. 28.

Parte III

Resultados

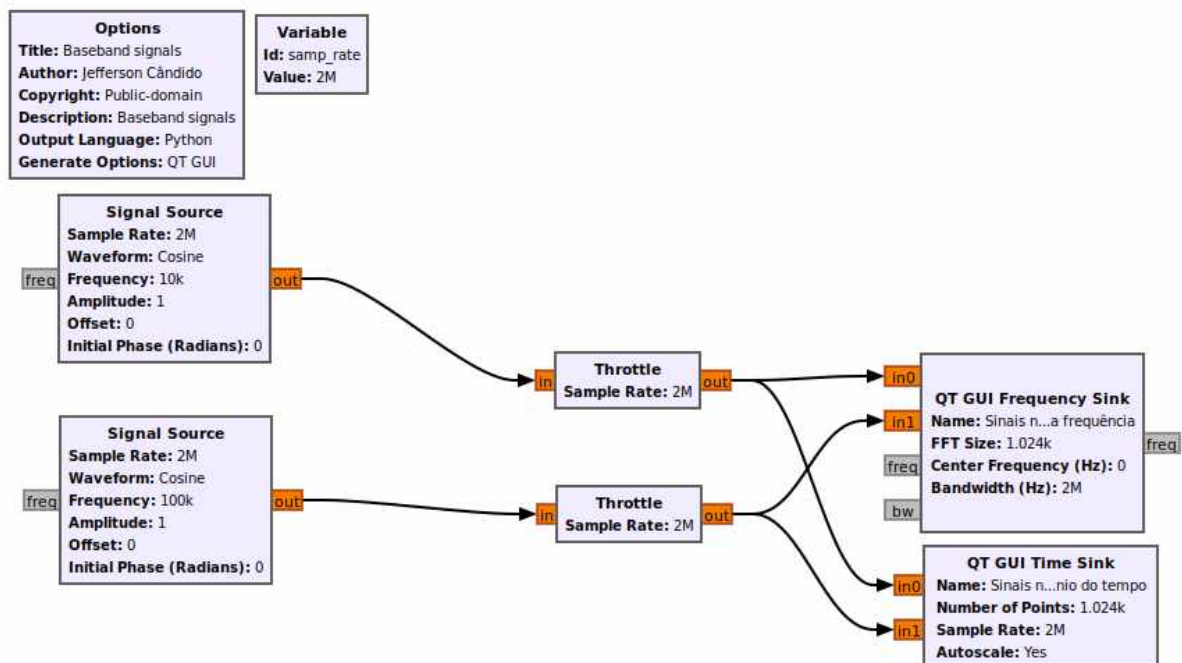


## 6 Revisão literária e análise de resultados

Com o intuito de reforçar a consistência prática deste trabalho, é necessário fazer a ligação entre os passos que foram executados até agora e suas fundamentações teóricas. Tópicos essenciais de telecomunicações que não podem deixar de ser analisados é o Teorema de Modulação e a aplicação de filtros digitais, visto que os blocos do GNURadio necessitam de realizar operações de conversão de taxa a todo momento. Em todos os *flowgraph's* também foi necessário passar o parâmetro **samp\_rate** que é a taxa de amostragem (dada em Hz), a qual sempre deve ser observada, para que esteja de acordo com o Teorema da amostragem (ou Teorema de Nyquist), pois a utilização de valores que sejam inferiores ao dobro da largura de banda (ou máximo valor de frequência) do sinal em banda-base resulta em efeitos indesejados caso fosse necessário recuperar este mesmo sinal após a recepção, transmissão e/ou processamento.

### Teorema da Modulação

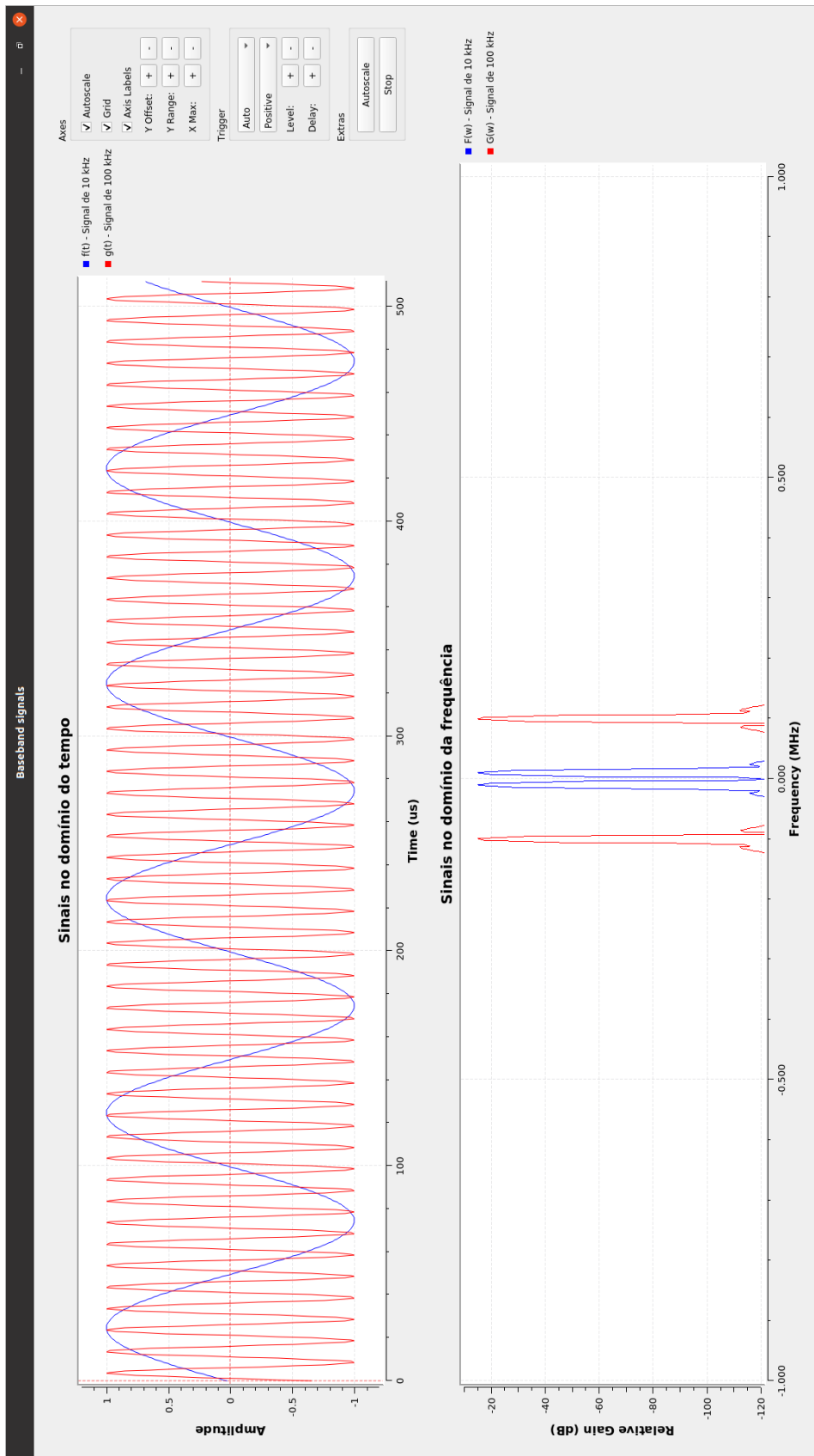
Figura 29 – *Flowgraph* de um sinal em banda base e de uma portadora.



Fonte: Elaborado pelo autor.

Para conferir o Teorema da modulação, foi montado o *flowgraph* apresentado na Fig. 29 que ao ser executado mostra as curvas de dois sinais (Fig. 30), sendo que o primeiro

Figura 30 – Sinal mensagem e da portadora no domínio do tempo e da frequência.



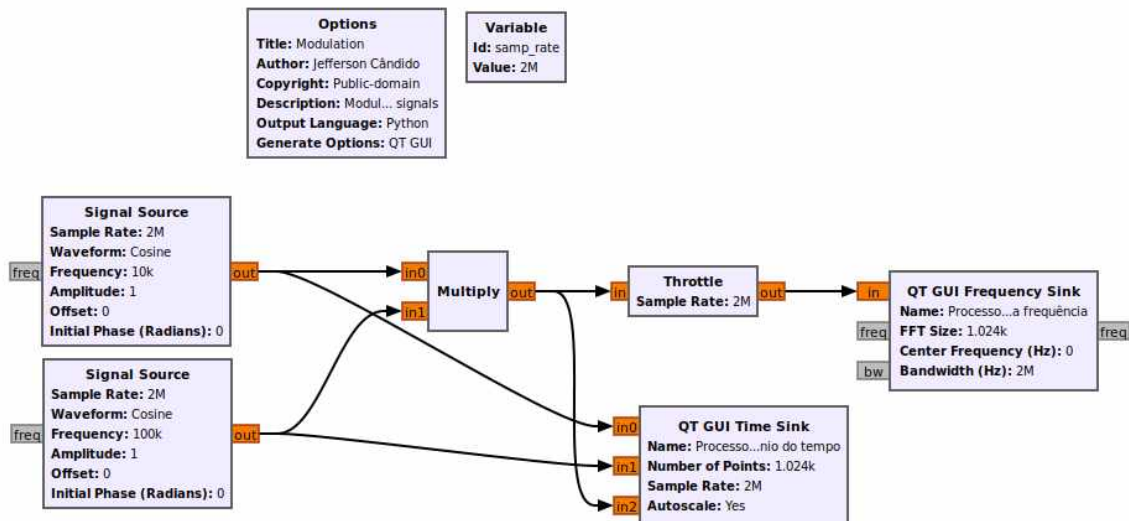
Fonte: Elaborado pelo autor.

( $f(t) = \cos(\omega_m t)$ , onde  $\omega_m = 2\pi f_m$ ) é um sinal em banda base (fonte de sinal senoidal, tom de frequência de  $f_m = 10$  kHz), aqui chamado de mensagem e, o segundo é o sinal de uma portadora ( $c(t) = \cos(\omega_c t)$ , onde  $\omega_c = 2\pi f_c$ ) com  $f_c = 100$  kHz. Considerando  $f(t)$  e  $c(t)$  e:

- sabendo que são sinais lineares e invariantes no tempo;
- suas transformadas de Fourier são  $F(\omega)$  e  $C(\omega)$ , respectivamente;
- sabendo que o Teorema da Modulação [23] determina que ao multiplicar um sinal mensagem (em banda base) por outro sinal com uma portadora de frequência (*Flowgraph* da Fig. 31);

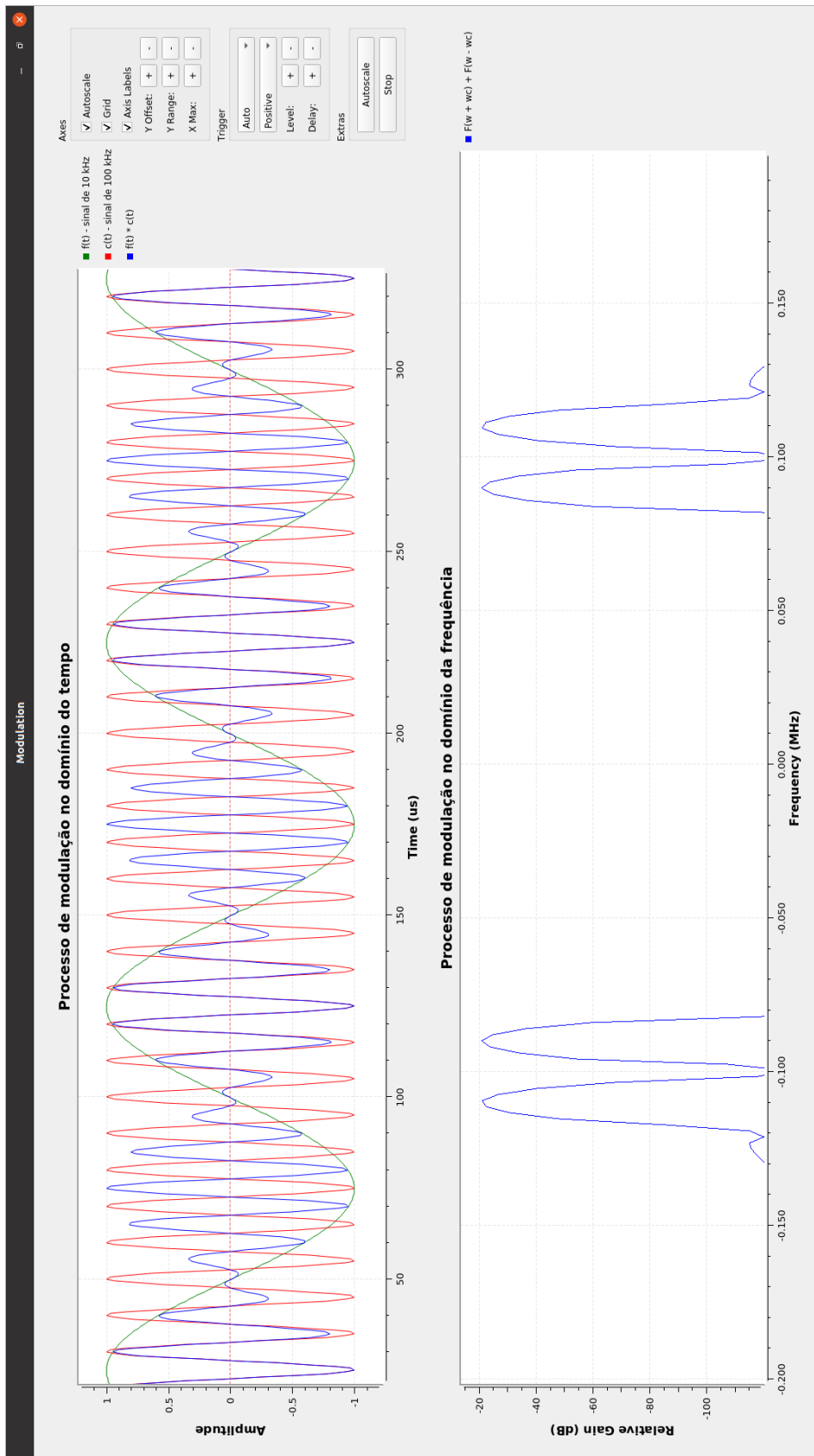
tem-se que o reflexo no espectro em frequência deste produto é entendido como o sinal original apenas deslocado exatamente do valor da frequência da onda portadora. Em resumo, o sinal que antes encontrava-se em torno da frequência 0 Hz ( $F(\omega)$ ), passa a ficar em torno da frequência da portadora (sinal modulado  $F(\omega + \omega_c) + F(\omega - \omega_c)$ ). A Fig. 32 mostra esse deslocamento em frequência como um exemplo prático de demonstração do Teorema da Modulação.

Figura 31 – *Flowgraph* para modulação de um sinal.



Fonte: Elaborado pelo autor.

Figura 32 – Modulação do sinal mensagem, curvas no tempo e na frequência.



Fonte: Elaborado pelo autor.



## Decimação e Interpolação

Decimação e interpolação são outros dois conceitos bastante importantes e que demandam do desenvolvedor alguma afinidade teórica com o processamento de sinais, pois a qualquer momento pode ser necessário fazer a conversão da frequência de amostragem do sinal por algum fator [1], aqui denominados  $L$  (para interpolação) e  $M$  (para decimação). Basicamente, dizer que está sendo realizado o processo de decimação de um sinal significa que sua taxa de amostragem foi reduzida por um fator  $M$  inteiro e para a interpolação seria o caso de ocorrer um aumento dessa taxa por um fator  $L$ . Às vezes é necessário fazer uma conversão de taxa que não pode ser usando um fator inteiro e é aí que entra uma conversão fracionário (ou racional) dessa taxa.

Matematicamente falando, dado o sinal discreto  $x[n]$  que possui transformada de Fourier  $X(e^{j\omega})$ , o qual deseja-se elevar ou diminuir a frequência de amostragem por um fator  $L$  - para interpolação - ou  $M$  - para decimação - constante (e inteiro por se tratar de sinais discretos), gerando um novo sinal discreto ( $x_L[n]$  ou  $x_M[n]$ ), basta recordar a propriedade de escalonamento aplicada à transformada de Fourier [23], onde:

$$x_L[n] = x\left[\frac{n}{L}\right] \iff LX(e^{j\omega L})$$

sendo  $x_L[n]$  o sinal após o procedimento de interpolação e,

$$x_M[n] = x[nM] \iff \frac{1}{M}X(e^{j\frac{\omega}{M}})$$

com  $x_M[n]$  o sinal em que ocorreu a decimação.

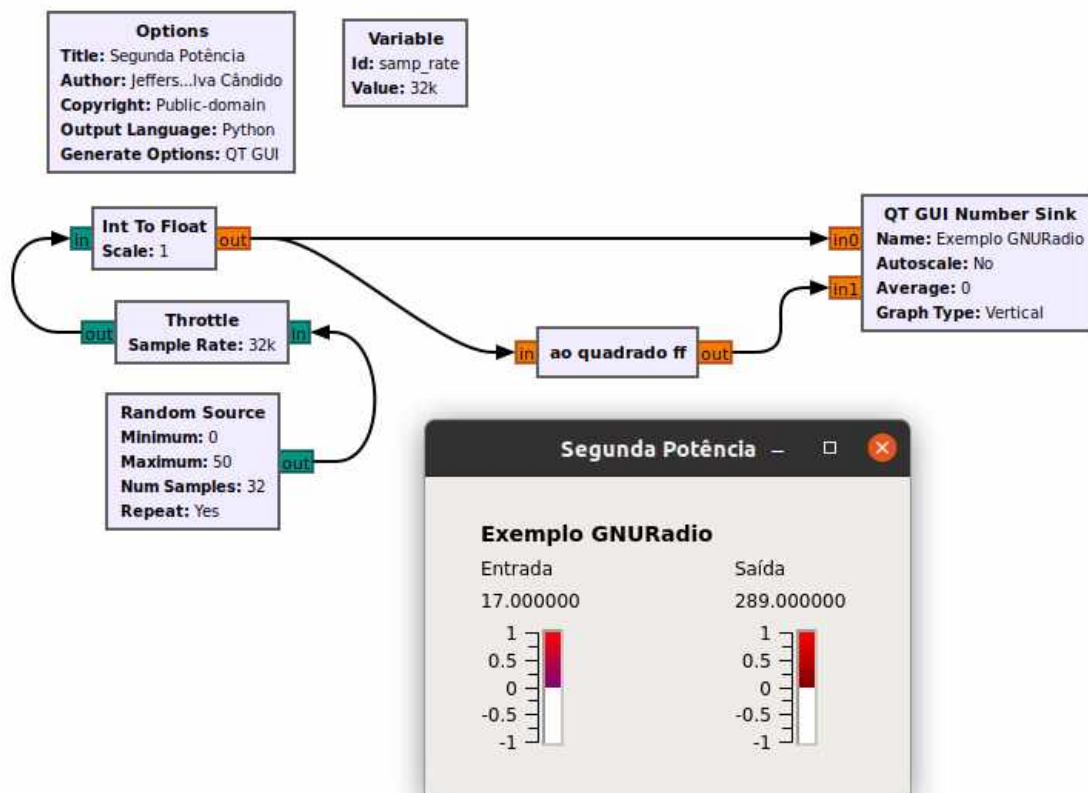
Quando deseja-se fazer um procedimento de reamostragem por um fator não inteiro, então entra uma relação entre os fatores de interpolação e decimação ao mesmo tempo fazendo com que o sinal reamostrado racionalmente seja:

$$x_{L/M}[n] = x\left[\frac{nM}{L}\right] \iff \frac{L}{M}X(e^{j\frac{\omega L}{M}})$$

sendo  $L$  e  $M$  valores inteiros.

O bloco OOT que foi criado na seção 5 foi testado conforme mostra a Fig. 33, sendo criado um *flowgraph* propriamente para verificar sua funcionalidade de receber amostras do tipo ponto flutuante, elevar seus valores ao quadrado e apresentá-los na saída.

Figura 33 – Utilização do bloco customizado em um *flowgraph*.



Fonte: Elaborado pelo autor.

## 7 Considerações Finais

Todos os objetivos previstos para este trabalho foram alcançados e felizmente foi possível trazer em uma abordagem ampla, mesmo que introdutória, alguns conceitos importantes e essenciais para auxiliar engenheiros e desenvolvedores de aplicações de rádio definido por software. Trabalhos futuros a partir deste podem ser realizados de diversas maneiras, como por exemplo:

- utilização do SDR para levantamento da resposta em frequência de dispositivos transmissores;
- construção de diferentes tipos de moduladores de sinais digitais (BPSK, QPSK, QAM, etc ...);
- criação de transmissores dentro dos padrões do mercado (DVB, ISDB, ATSC, NTSC, etc ...);
- implementação prática de filtros digitais customizados.

Foi possível abordar diferentes temáticas que estão relacionadas ao desenvolvimento de aplicações com GNURadio, passando desde questões relacionadas ao *hardware* necessário até detalhes de implementação e geração de pacotes do *software*. O foco dado ao provisionamento do ambiente de desenvolvimento se deu ao fato de que essa não é diretamente uma das áreas de estudo dos estudantes de engenharia eletrônica e de telecomunicações da UFU e o conhecimento sobre servidores Linux, utilização de linhas de comando, conectividade entre sistemas e etc ... têm grande chances de serem úteis e necessários na bagagem de um profissional formado neste contexto.

Um ponto importante a ser observado que a visão trazida neste trabalho é despertar no engenheiro que desejar ter contato com o GNURadio a vontade de ser um desenvolvedor e não apenas um usuário da ferramenta. Na visão do autor os passos considerados mais onerosos são os que foram realizados neste trabalho, visto que os estudantes de engenharia eletrônica e de telecomunicações da UFU possuem o foco mais voltado para a área de processamento digital de sinais utilizando ferramentas como MATLAB® e/ou Simulink®.

O esperado é que este seja um ponto de partida e todo o material aqui apresentado ficará disponível na conta do github do autor para que possa ser reproduzido posteriormente.



# Referências

- 1 HAYES, M. H. Schaum's outline of theory and problems of digital signal processing. In: THE MCGRAW-HILL COMPANIES, INC. *Schaum's outline of theory and problems of digital signal processing*. United States of America, 1999. p. 110–114. Citado 2 vezes nas páginas 25 e 79.
- 2 Ebert, C.; Duarte, C. H. C. Requirements engineering for the digital transformation: Industry panel. In: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. [S.l.: s.n.], 2016. p. 4–5. Citado na página 25.
- 3 CORDEIRO, J. R. S. et al. Introdução a rádios definidos por software com aplicações em gnradio. In: *Minicursos do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos — SBRC 2015*. Porto Alegre, RS, Brasil: SBC, 2015. ISSN 2177-4987. Citado na página 25.
- 4 OSSMANN, M. *HackRF One at 1 MHz*. 2015. Disponível em: <<https://greatscottgadgets.com/2015/05-15-hackrf-one-at-1-mhz/>>. Acesso em: 09 de dezembro de 2020. Citado na página 30.
- 5 OSSMANN, M.; SPILL, D. *HackRF One - Receive Power*. 2014–2020. Disponível em: <<https://github.com/mossmann/hackrf/wiki/HackRF-One#receive-power>>. Acesso em: 09 de dezembro de 2020. Citado na página 30.
- 6 OSSMANN, M.; SPILL, D. *HackRF One - Transmit Power*. 2014–2020. Disponível em: <<https://github.com/mossmann/hackrf/wiki/HackRF-One#transmit-power>>. Acesso em: 09 de dezembro de 2020. Citado na página 31.
- 7 OSSMANN, M. et al. *HackRF - Hardware Components*. 2012–2020. Disponível em: <<https://github.com/mossmann/hackrf/wiki/Hardware-Components>>. Acesso em: 09 de dezembro de 2020. Citado na página 32.
- 8 MENG, H.; THAIN, D. Facilitating the reproducibility of scientific workflows with execution environment specifications. *Procedia Computer Science*, Elsevier, v. 108, p. 705–714, 2017. Citado na página 35.
- 9 HAT, R. *Chapter 5. Running Super-Privileged Containers*. [S.l.], 2020. Disponível em: <[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux\\_atomic\\_host/7/html/managing\\_containers/running\\_super\\_privileged\\_containers](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/managing_containers/running_super_privileged_containers)>. Acesso em: 09 de dezembro de 2020. Citado na página 36.
- 10 DOCKER. *Install Docker Engine*. 2020. Disponível em: <<https://docs.docker.com/engine/install/>>. Acesso em: 09 de dezembro de 2020. Citado na página 37.
- 11 DOCKER. *Post-installation steps for Linux*. 2020. Disponível em: <<https://docs.docker.com/engine/install/linux-postinstall/>>. Acesso em: 09 de dezembro de 2020. Citado na página 39.
- 12 RUSLING, D. A. Chapter 5 - interprocess communication mechanisms. In: THE LINUX DOCUMENTATION PROJECT. *The Linux Kernel*. 3 Foxglove

- Close, Wokingham, Berkshire RG41 3NF, UK, 1996–1999. p. 51–60. Disponível em: <<https://tldp.org/LDP/tlk/ipc/ipc.html>>. Acesso em: 09 de dezembro de 2020. Citado na página 41.
- 13 PROJECT, G. *Dconf overview*. [S.l.], 2005–2014. Disponível em: <<https://developer.gnome.org/dconf/unstable/dconf-overview.html>>. Acesso em: 09 de dezembro de 2020. Citado na página 42.
- 14 BASTIAN, W.; LORTIE, R.; POETTERING, L. *XDG Base Directory Specification*. 2010. Disponível em: <<https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html>>. Acesso em: 09 de dezembro de 2020. Citado na página 42.
- 15 PENNINGTON, H. et al. *D-Bus Specification*. 2020. Revisão 0.36. Disponível em: <<https://dbus.freedesktop.org/doc/dbus-specification.html>>. Acesso em: 09 de dezembro de 2020. Citado na página 42.
- 16 TEAM, D. I. *Debian Installer Parameters*. 2004–2019. Disponível em: <<https://www.debian.org/releases/stable/amd64/ch05s03.en.html#installer-args>>. Acesso em: 09 de dezembro de 2020. Citado na página 42.
- 17 HAT, R. *What Are dconf Profiles?* [S.l.], 2020. Disponível em: <[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/desktop\\_migration\\_and\\_administration\\_guide/profiles](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/desktop_migration_and_administration_guide/profiles)>. Acesso em: 09 de dezembro de 2020. Citado na página 43.
- 18 OSSMANN, M.; BOONE, J. *HackRF - How to build the host software on Linux*. 2017–2020. Disponível em: <<https://github.com/mossmann/hackrf/tree/master/host#how-to-build-the-host-software-on-linux>>. Acesso em: 09 de dezembro de 2020. Citado na página 47.
- 19 OSSMANN, M.; BOONE, J. *HackRF - Updating Firmware*. 2013–2020. Disponível em: <<https://github.com/mossmann/hackrf/wiki/Updating-Firmware>>. Acesso em: 09 de dezembro de 2020. Citado na página 48.
- 20 SPILL, D. *HackRF - HackRF Tools*. 2013–2020. Disponível em: <<https://github.com/mossmann/hackrf/wiki/Software-Support#hackrf-tools>>. Acesso em: 09 de dezembro de 2020. Citado na página 48.
- 21 WIKI, G. *OutOfTreeModules*. 2020. Disponível em: <<https://wiki.gnuradio.org/index.php/OutOfTreeModules>>. Acesso em: 09 de dezembro de 2020. Citado na página 67.
- 22 BECK, K. In: THREE RIVERS INSTITUTE. *Test-Driven Development By Example*. [S.l.], 2002. p. 28. Citado na página 70.
- 23 LATHI, B. P. *Modern Digital and Analog Communication Systems 3e Osece*. 3rd. ed. USA: Oxford University Press, Inc., 1998. 84-90 p. ISBN 0195110099. Citado 2 vezes nas páginas 77 e 79.
- 24 OSSMANN, M. et al. *HackRF - Hardware Documentation*. 2012–2020. Disponível em: <<https://github.com/mossmann/hackrf/tree/master/doc/hardware>>. Acesso em: 09 de dezembro de 2020. Citado 4 vezes nas páginas 89, 91, 93 e 95.

# Anexos

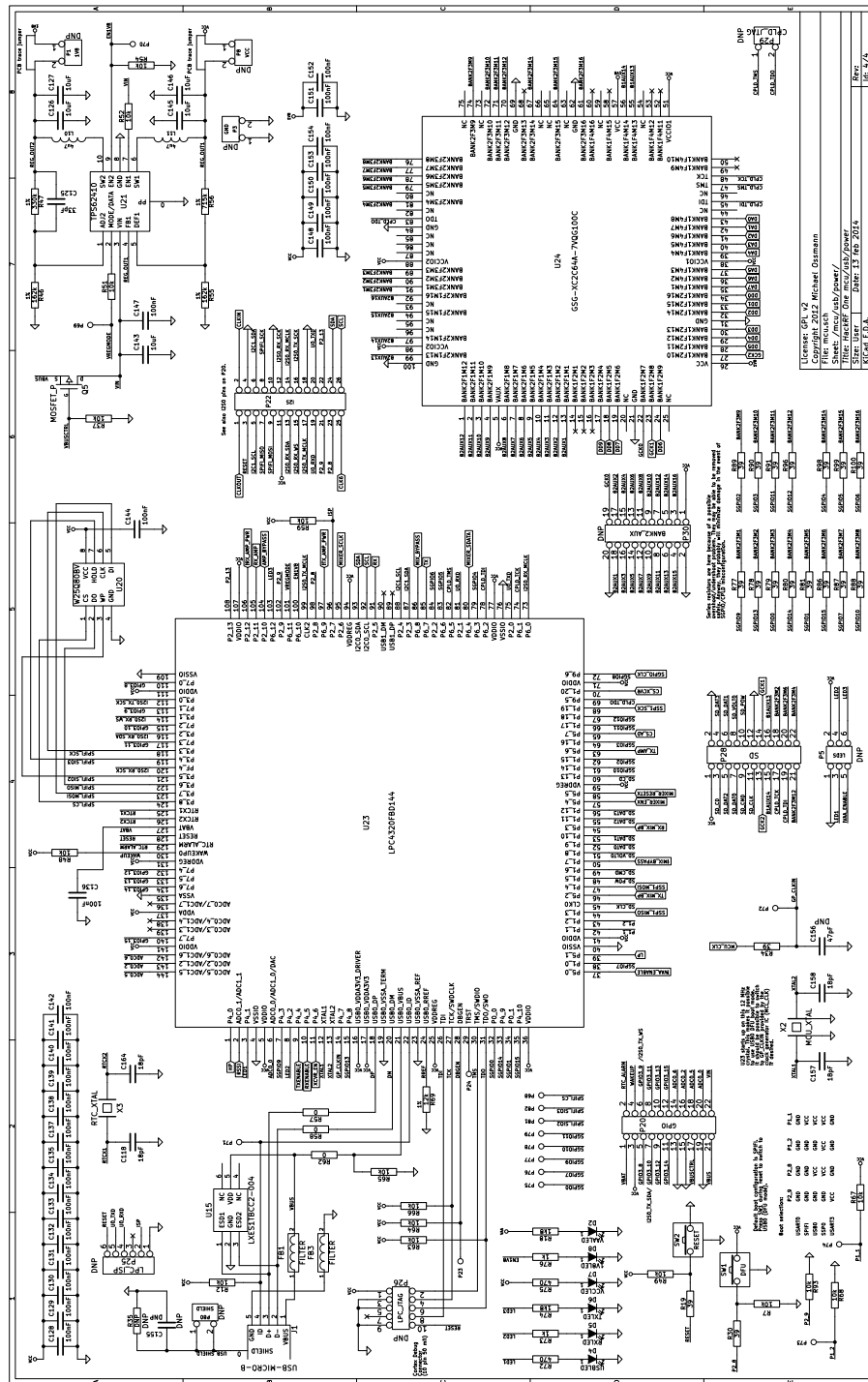






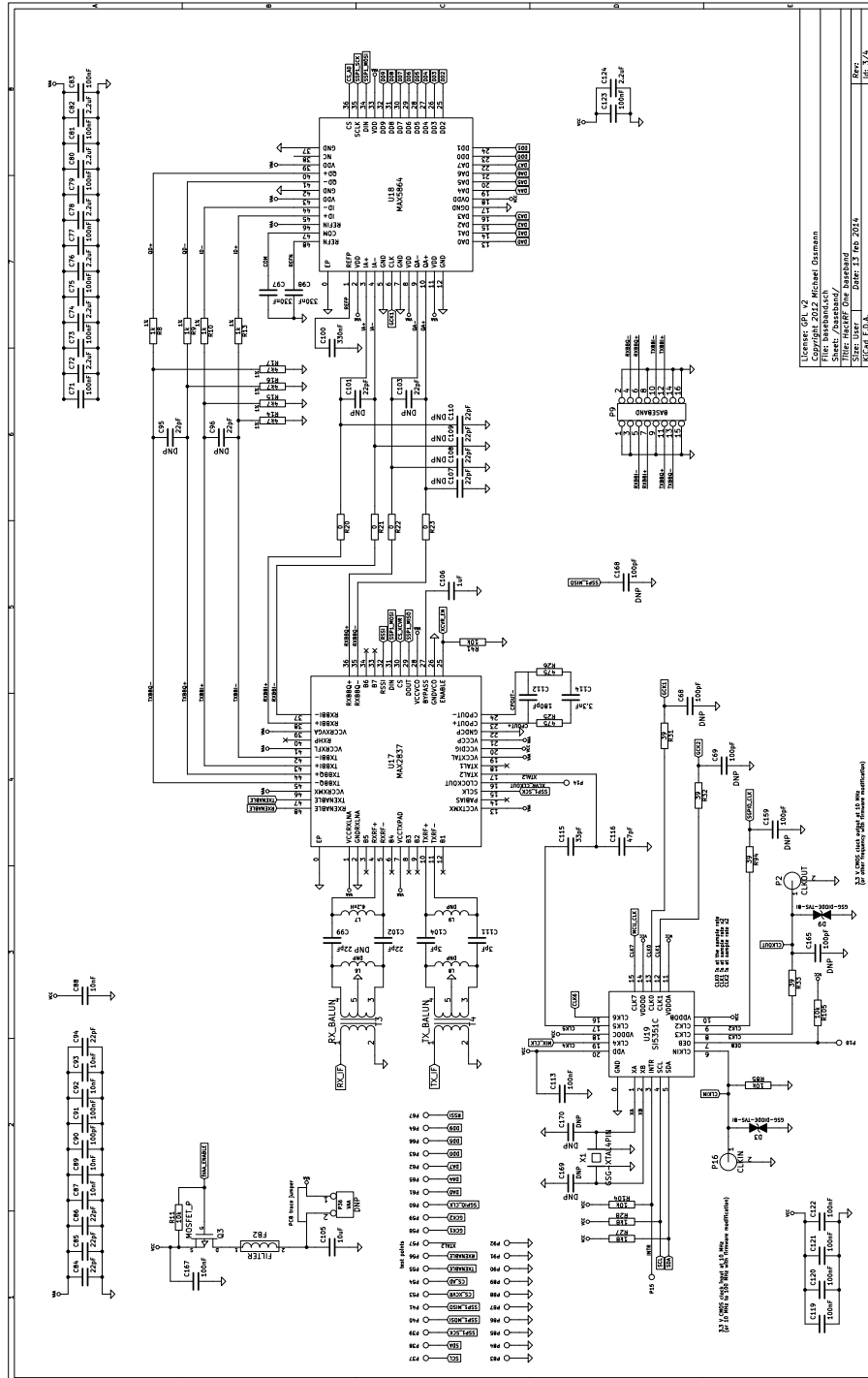


# ANEXO B – Diagrama esquemático do HackRF One - página 1.



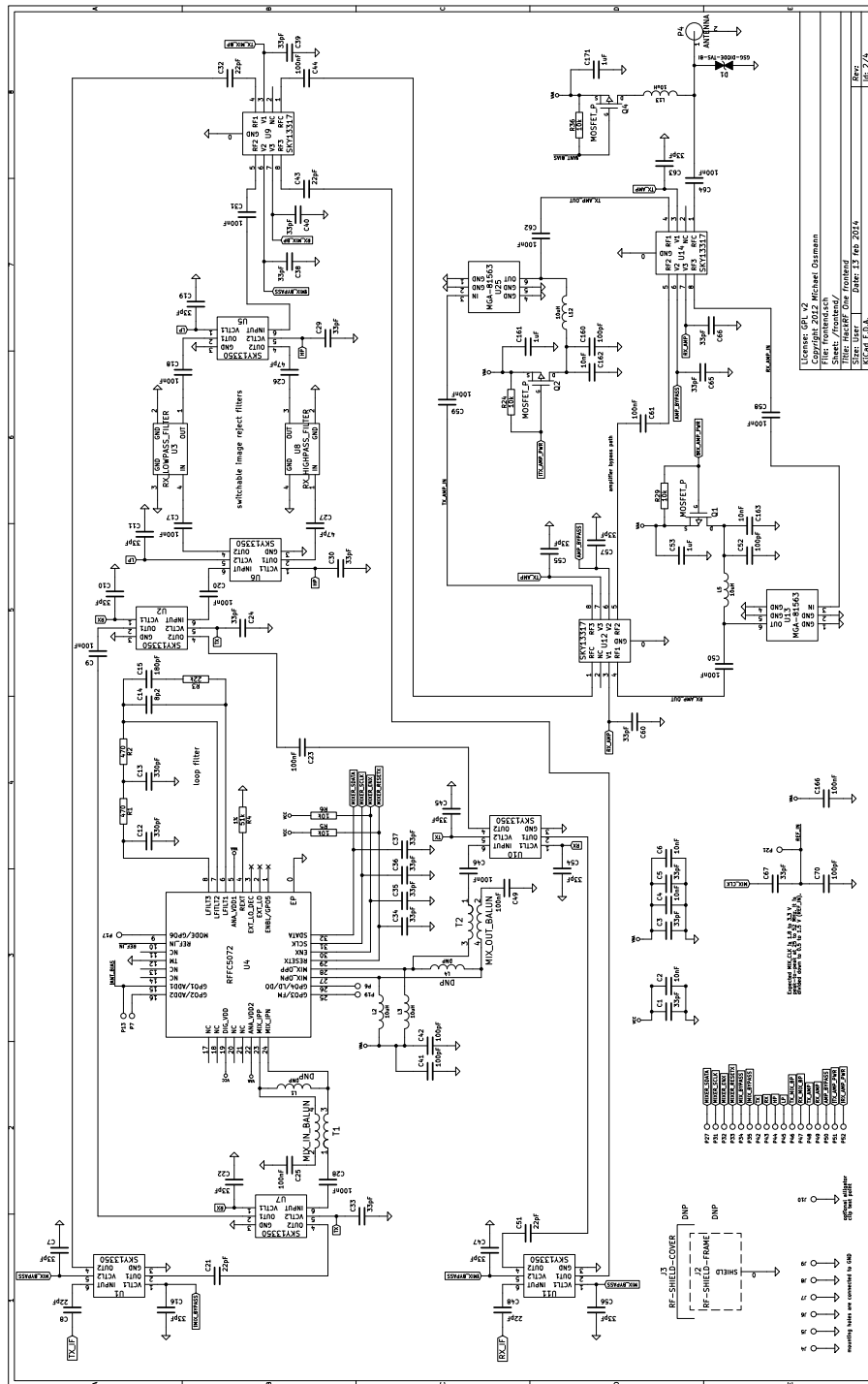


# ANEXO C – Diagrama esquemático do HackRF One - página 2.





# ANEXO D – Diagrama esquemático do HackRF One - página 3.

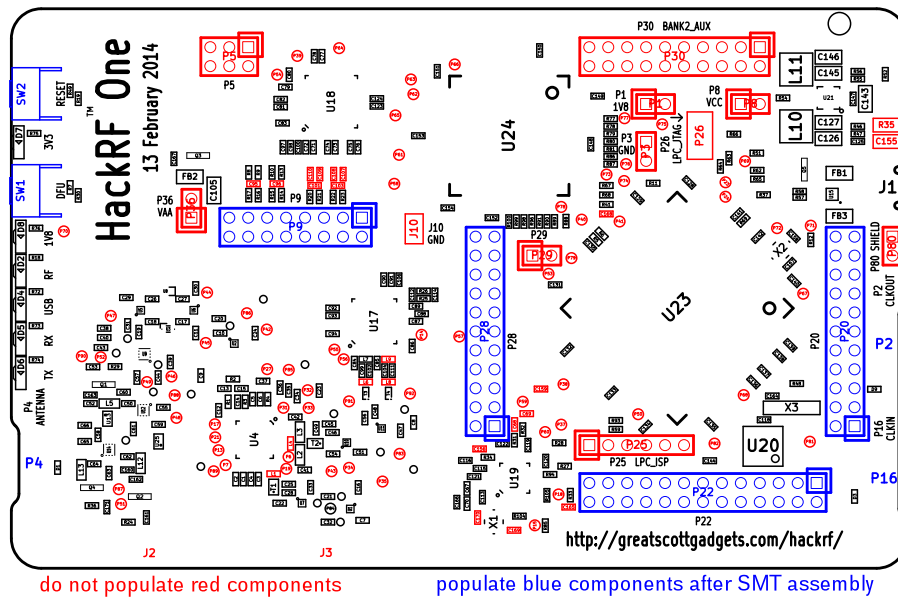


Fonte: GitHub [24].





# ANEXO E – Diagrama para montagem dos componentes eletrônicos no PCB do *HackRF One*.



Fonte: GitHub [24].