

T126

# Rádio Definido por Software

Aula #01  
Introdução

# O que é Rádio Definido por Software?



Rádio Definido por Software (RDS) é um sistema de radiocomunicação onde componentes tipicamente implementados em hardware (mixers, filtros, amplificadores, moduladores/demoduladores, detectores, etc) são implementados em software, em um PC ou sistema embarcado.

Fonte: [Wikipedia - Software-Defined Radio](#)

# Vantagens

- Habilidade de receber e transmitir sinais com várias modulações usando um mesmo conjunto de hardware;
- Habilidade de alterar a funcionalidade do rádio fazendo o download e execução de um novo software;
- Possibilidade de escolher adaptativamente uma frequência de operação ou modo de operação de acordo com as condições do sistema;
- Oportunidade de reconhecer e evitar interferências de outros canais de comunicação;
- Eliminação de hardware analógico, reduzindo custos e simplificando a arquitetura do sistema;
- Facilidade e flexibilidade de prototipação e realização de experimentos.

# Desvantagens

- Dificuldade na escrita de software para várias arquiteturas e plataformas;
- Necessidade de interfaces para sinais digitais e algoritmos;
- Faixa dinâmica reduzida para a grande gama de frequências de interesse em alguns projetos de dispositivos para RDS;
- Alta complexidade dos projetos de software e hardware é o preço pago pela grande flexibilidade no uso do RDS.

# Histórico



1991 - Joseph Mitola III

Primeiro uso do termo Rádio  
Definido por Software.

Fonte: [Software Radios: Survey, Critical Evaluation and Future Directions](#)

# Histórico



1990 até 1995

Projeto do rádio SpeakEasy II  
(US Air Force Research Laboratory)



Primeiro projeto a usar um dispositivo [FPGA](#) para processamento digital de dados de rádio.

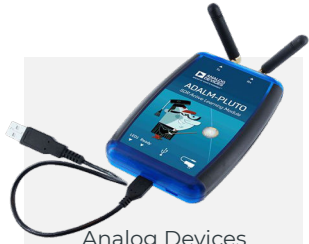
Emulava mais de 15 rádios militares existentes na época.

Fonte: [SpeakEasy: The Military Software Radio](#)

# O uso de RDS

- Sistemas de radiocomunicação
- Sistemas de celular [2G](#), 3G, [4G e 5G](#)
- Sistemas de TV Digital ([ISDB-T](#)): [Link](#)
- Sistemas de satélite: [Link](#)
- Sistemas de rádio amador: [Link](#)
- Sistemas de rádio digital ([DAB](#)): [Link](#)
- Receptor de GPS: [Link](#)
- Repositório de módulos não oficiais do GNU Radio: [Link](#)
- Sistemas de monitoramento aéreo ([ADS-B](#)): [Link 01](#), [Link 02](#)
- Wi-Fi: [Link](#)
- Rádio AM/FM: [Link](#)
- LoRa: [Link](#)
- Emulação de instrumentos de medida
- Radar doppler: [Link](#)

# Modelos de RDS

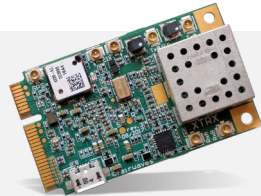


Analog Devices

[ADALM-PLUTO SDR](#)

325 MHz até 3,8 GHz  
20 MHz BW (USB 2.0)

US\$ 149



Fairwaves

[XTRX Pro](#)

30 MHz até 3,7 GHz  
120 MHz BW (mPCIe)

US\$ 599



Myriad RF

[LimeSDR PCIe](#)

100 kHz até 3,8 GHz  
61,44 MHz BW (PCIe)

US\$ 799



Ettus Research

[USRP B210](#)

70 MHz até 6 GHz  
56 MHz BW (USB 3.0)

US\$ 1.259



NI (Ettus)

[USRP 2954 \(X310\)](#)

10 MHz até 6 GHz  
160 MHz BW (PCIe)

US\$ 5.612



NI (Ettus)

[USRP 2974 \(USRP 2974\)](#)

10 MHz até 6 GHz  
160 MHz BW (integrado)

US\$ 13.455

# Ferramentas e linguagens



VHDL



C/C++





# Ferramentas e linguagens



# GNU Radio e sua interface



test\_01.grc - /home/alexandre/Programs - GNU Radio Companion

File Edit View Run Tools Help

**Options**  
ID: top\_block  
Generate Options: QT GUI

**Variable**  
ID: samp\_rate  
Value: 2.084M

**QT GUI Range**  
ID: fc  
Default Value: 0  
Start: 0  
Stop: 1M  
Step: 1k

**Signal Source**  
Sample Rate: 2.084M  
Waveform: Cosine  
Frequency: 0  
Amplitude: 1  
Offset: 0

**PlutoSDR Sink**  
IIO context URI:  
LO Frequency: 2.4G  
Sample rate: 2.084M  
RF bandwidth: 20M  
Buffer size: 32.768k  
Cyclic: False  
Attenuation (dB): 10  
Filter:  
Filter auto: True

**PlutoSDR Source**  
Device URI:  
LO Frequency: 2.4G  
Sample rate: 2.084M  
RF bandwidth: 20M  
Buffer size: 32.768k  
Quadrature: True  
RF DC: True  
BB DC: True  
Gain Mode: Manual  
Manual Gain (dB): 64  
Filter:  
Filter auto: True

**QT GUI Frequency Sink**  
FFT Size: 1.024k  
Center Frequency (Hz): 2.4G  
Bandwidth (Hz): 2.084M

<<< Welcome to GNU Radio Companion 3.7.11 >>>

Block paths:  
/usr/share/gnuradio/grc/blocks  
/usr/local/share/gnuradio/grc/blocks

Loading: "/home/alexandre/Programs/test\_01.grc"  
>>> Done

Generating: '/home/alexandre/Programs/top\_block.py'

Generating: '/home/alexandre/Programs/top\_block.py'

Executing: /usr/bin/python -u /home/alexandre/Programs/top\_block.py

Warning: failed to XinitThreads()

>>> Done

Id	Value
Imports	
Variables	
fc	<Open Properties>
samp_rate	2084000

- Core
  - Audio
  - Boolean Operators
  - Byte Operators
  - Channelizers
  - Channel Models
  - Coding
  - Control Port
  - Debug Tools
  - Deprecated
  - Digital Television
  - Equalizers
  - Error Coding
  - FCD
  - File Operators
  - Filters
  - Fourier Analysis
  - GUI Widgets
  - Impairment Models
  - Instrumentation
  - Level Controllers
  - Math Operators
  - Measurement Tools
  - Message Tools
  - Misc
  - Modulators
  - Networking Tools
  - NOAA
  - OFDM
  - Packet Operators
  - Pager
  - Peak Detectors

# GNU Radio e sua interface

Flow graph

The screenshot displays the GNU Radio Companion (GRC) interface. The main window shows a flow graph with the following components and connections:

- Options** (ID: top\_block, Generate Options: QT GUI)
- Variable** (ID: samp\_rate, Value: 2.084M)
- QT GUI Range** (ID: fc, Default Value: 0, Start: 0, Stop: 1M, Step: 1k)
- Signal Source** (Sample Rate: 2.084M, Waveform: Cosine, Frequency: 0, Amplitude: 1, Offset: 0)
- PlutoSDR Sink** (IIO context URI, LO Frequency: 2.4G, Sample rate: 2.084M, RF bandwidth: 20M, Buffer size: 32.768k, Cyclic: False, Attenuation (dB): 10, Filter: Filter auto: True)
- PlutoSDR Source** (Device URI, LO Frequency: 2.4G, Sample rate: 2.084M, RF bandwidth: 20M, Buffer size: 32.768k, Quadrature: True, RF DC: True, BB DC: True, Gain Mode: Manual, Manual Gain (dB): 64, Filter: Filter auto: True)
- QT GUI Frequency Sink** (FFT Size: 1.024k, Center Frequency (Hz): 2.4G, Bandwidth (Hz): 2.084M)

The flow graph is connected as follows: Signal Source → PlutoSDR Sink → PlutoSDR Source → QT GUI Frequency Sink.

The terminal window at the bottom shows the following output:

```
<<< Welcome to GNU Radio Companion 3.7.11 >>>

Block paths:
/usr/share/gnuradio/grc/blocks
/usr/local/share/gnuradio/grc/blocks

Loading: "/home/alexandre/Programs/test_01.grc"
>>> Done

Generating: '/home/alexandre/Programs/top_block.py'
Generating: '/home/alexandre/Programs/top_block.py'

Executing: /usr/bin/python -u /home/alexandre/Programs/
top_block.py

Warning: failed to XinitThreads()

>>> Done
```

The right sidebar shows a list of blocks categorized by type:

- Core
- Audio
- Boolean Operators
- Byte Operators
- Channelizers
- Channel Models
- Coding
- Control Port
- Debug Tools
- Deprecated
- Digital Television
- Equalizers
- Error Coding
- FCD
- File Operators
- Filters
- Fourier Analysis
- GUI Widgets
- Impairment Models
- Instrumentation
- Level Controllers
- Math Operators
- Measurement Tools
- Message Tools
- Misc
- Modulators
- Networking Tools
- NOAA
- OFDM
- Packet Operators
- Pager
- Peak Detectors

Id	Value
Imports	
Variables	
fc	<Open Properties>
samp_rate	2084000

# GNU Radio e sua interface

Console Panel

The screenshot displays the GNU Radio Companion (GRC) interface. The main workspace shows a signal flow graph with the following components and connections:

- Signal Source**: Sample Rate: 2.084M, Waveform: Cosine, Frequency: 0, Amplitude: 1, Offset: 0.
- PlutoSDR Sink**: IIO context URI, LO Frequency: 2.4G, Sample rate: 2.084M, RF bandwidth: 20M, Buffer size: 32.768k, Cyclc: False, Attenuation (dB): 10, Filter: Filter auto: True.
- PlutoSDR Source**: Device URI, LO Frequency: 2.4G, Sample rate: 2.084M, RF bandwidth: 20M, Buffer size: 32.768k, Quadrature: True, RF DC: True, BB DC: True, Gain Mode: Manual, Manual Gain (dB): 64, Filter: Filter auto: True.
- QT GUI Frequency Sink**: FFT Size: 1.024k, Center Frequency (Hz): 2.4G, Bandwidth (Hz): 2.084M.

The console panel at the bottom, outlined in red, shows the following output:

```
<<< Welcome to GNU Radio Companion 3.7.11 >>>

Block paths:
/usr/share/gnuradio/grc/blocks
/usr/local/share/gnuradio/grc/blocks

Loading: "/home/alexandre/Programs/test_01.grc"
>>> Done

Generating: '/home/alexandre/Programs/top_block.py'
Generating: '/home/alexandre/Programs/top_block.py'

Executing: /usr/bin/python -u /home/alexandre/Programs/
top_block.py

Warning: failed to XinitThreads()

>>> Done
```

The right sidebar lists various tool categories, including Core, Audio, Boolean Operators, Byte Operators, Channelizers, Channel Models, Coding, Control Port, Debug Tools, Deprecated, Digital Television, Equalizers, Error Coding, FCD, File Operators, Filters, Fourier Analysis, GUI Widgets, Impairment Models, Instrumentation, Level Controllers, Math Operators, Measurement Tools, Message Tools, Misc, Modulators, Networking Tools, NOAA, OFDM, Packet Operators, Pager, and Peak Detectors.

# GNU Radio e sua interface

Variable Editor

The screenshot displays the GNU Radio Companion (GRC) interface. The main window shows a flow graph titled 'test\_01.grc' with the following components and connections:

- Signal Source**: Sample Rate: 2.084M, Waveform: Cosine, Frequency: 0, Amplitude: 1, Offset: 0.
- PlutoSDR Sink**: IIO context URI, LO Frequency: 2.4G, Sample rate: 2.084M, RF bandwidth: 20M, Buffer size: 32.768k, Cyclc: False, Attenuation (dB): 10, Filter: Filter auto: True.
- PlutoSDR Source**: Device URI, LO Frequency: 2.4G, Sample rate: 2.084M, RF bandwidth: 20M, Buffer size: 32.768k, Quadrature: True, RF DC: True, BB DC: True, Gain Mode: Manual, Manual Gain (dB): 64, Filter: Filter auto: True.
- QT GUI Frequency Sink**: FFT Size: 1.024k, Center Frequency (Hz): 2.4G, Bandwidth (Hz): 2.084M.

The flow graph is connected as follows: Signal Source → PlutoSDR Sink → PlutoSDR Source → QT GUI Frequency Sink.

The **Variable Editor** is open at the bottom, showing a table of variables:

Id	Value
Imports	
Variables	
fc	<Open Properties>
samp_rate	2084000

The console output at the bottom shows the following messages:

```
<<< Welcome to GNU Radio Companion 3.7.11 >>>

Block paths:
/usr/share/gnuradio/grc/blocks
/usr/local/share/gnuradio/grc/blocks

Loading: "/home/alexandre/Programs/test_01.grc"
>>> Done

Generating: '/home/alexandre/Programs/top_block.py'

Generating: '/home/alexandre/Programs/top_block.py'

Executing: /usr/bin/python -u /home/alexandre/Programs/
top_block.py

Warning: failed to XinitThreads()

>>> Done
```

# GNU Radio e sua interface



## Block Tree Panel

The screenshot displays the GNU Radio Companion (GRC) interface. The main workspace shows a block diagram with the following components and connections:

- Signal Source** (ID: top\_block) connected to **PlutoSDR Sink** (ID: fc).
- PlutoSDR Sink** connected to **PlutoSDR Source** (ID: samp\_rate).
- PlutoSDR Source** connected to **QT GUI Frequency Sink** (ID: 1.024k).

The **Block Tree Panel** on the right side of the interface is highlighted with a red border. It lists various block categories and their sub-items:

- Core
  - Audio
  - Boolean Operators
  - Byte Operators
  - Channelizers
  - Channel Models
  - Coding
  - Control Port
  - Debug Tools
  - Deprecated
  - Digital Television
  - Equalizers
  - Error Coding
  - FCD
  - File Operators
  - Filters
  - Fourier Analysis
  - GUI Widgets
  - Impairment Models
  - Instrumentation
  - Level Controllers
  - Math Operators
  - Measurement Tools
  - Message Tools
  - Misc
  - Modulators
  - Networking Tools
  - NOAA
  - OFDM
  - Packet Operators
  - Pager
  - Peak Detectors

The bottom panel shows the command line output for the current session:

```
<<< Welcome to GNU Radio Companion 3.7.11 >>>

Block paths:
/usr/share/gnuradio/grc/blocks
/usr/local/share/gnuradio/grc/blocks

Loading: "/home/alexandre/Programs/test_01.grc"
>>> Done

Generating: '/home/alexandre/Programs/top_block.py'

Generating: '/home/alexandre/Programs/top_block.py'

Executing: /usr/bin/python -u /home/alexandre/Programs/
top_block.py

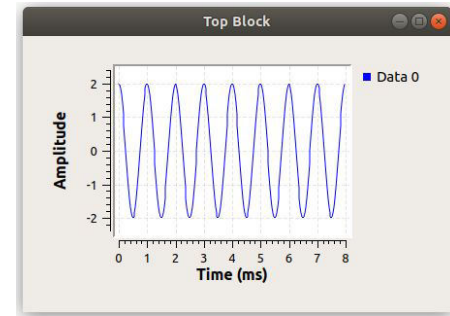
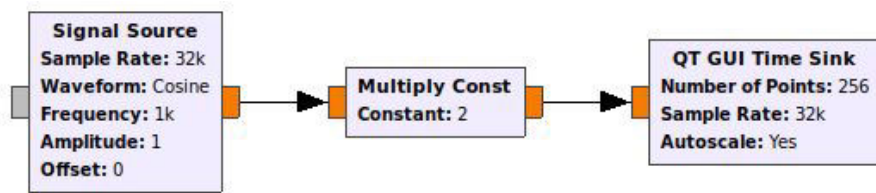
Warning: failed to XinitThreads()

>>> Done
```

Id	Value
Imports	
Variables	
fc	<Open Properties>
samp_rate	2084000



# Fluxo de dados entre blocos



- Cada bloco no GNU Radio tem sua própria [thread](#) no sistema operacional. Nesse caso, os blocos se comportam como pequenos programas sendo executados de forma independente no sistema operacional;
- Cada bloco no GNU Radio possui um [buffer circular](#) de saída onde são armazenados os dados gerados ou processados pelo bloco. No caso do projeto acima, os blocos Signal Source e Multiply Const possuem esse *buffer* de saída;
- Ao executar um *flow graph*, o [scheduler](#) do GNU Radio informa continuamente a cada uma das *threads* quanto há de espaço nos *buffers* de saída;
- Enquanto houver espaço nos *buffers* de saída, os blocos produzem ou processam dados para completar o *buffer*. O bloco subsequente na cadeia é notificado quando há dados a serem consumidos no *buffer* de saída do bloco anterior.

Fonte: [Behind the Veil: A Peek at GNU Radio's Buffer Architecture](#)

# Biblioteca de blocos do GNU Radio



- A instalação padrão do GNU Radio já vem com uma série de blocos pré instalados e oficialmente suportados. Estes blocos podem ser encontrados no *Block Tree Panel*, conforme a figura ao lado;
- A documentação para estes blocos pode ser encontrada no [site do GNU Radio](#);
- A lista de blocos pré instalados varia de acordo com a versão do GNU Radio instalada.

▼ Core

- ▶ Audio
- ▶ Boolean Operators
- ▶ Byte Operators
- ▶ Channelizers
- ▶ Channel Models
- ▶ Coding
- ▶ Control Port
- ▶ Debug Tools
- ▶ Deprecated
- ▶ Digital Television
- ▶ Equalizers
- ▶ Error Coding
- ▶ FCD
- ▶ File Operators
- ▶ Filters
- ▶ Fourier Analysis
- ▶ GUI Widgets
- ▶ Impairment Models
- ▶ Instrumentation
- ▶ Level Controllers
- ▶ Math Operators
- ▶ Measurement Tools
- ▶ Message Tools
- ▶ Misc
- ▶ Modulators
- ▶ Networking Tools
- ▶ NOAA
- ▶ OFDM
- ▶ Packet Operators
- ▶ Pager
- ▶ Peak Detectors
- ▶ Resamplers
- ▶ Stream Operators
- ▶ Stream Tag Tools
- ▶ Symbol Coding
- ▶ Synchronizers
- ▶ Trellis Coding
- ▶ Type Converters
- ▶ UHD
- ▶ Variables
- ▶ Video



# Adição de novos blocos à biblioteca



- O GNU Radio permite a adição de novos blocos ou módulos de blocos à sua biblioteca. Eles são chamados de módulos *Out-Of-Tree* (OOT);
- É possível importar um módulo OOT já pronto, não oficialmente suportado pelo GNU Radio, obtido na internet;
- Existe também a possibilidade de criação do seu próprio módulo customizado que poderá ser incorporado à biblioteca do GNU Radio e distribuído para outros usuários;
- Os blocos para o módulo OOT customizado podem ser construídos usando duas linguagens de programação: Python e C/C++. As vantagens e desvantagens no uso destas linguagens serão discutidas mais à frente na disciplina.

- ▼ Core
- ▶ Audio
- ▶ Boolean Operators
- ▶ Byte Operators
- ▶ Channelizers
- ▶ Channel Models
- ▶ Coding
- ▶ Control Port
- ▶ Debug Tools
- ▶ Deprecated
- ▶ Digital Television
- ▶ Equalizers
- ▶ Error Coding
- ▶ FCD
- ▶ File Operators
- ▶ Filters
- ▶ Fourier Analysis
- ▶ GUI Widgets
- ▶ Impairment Models
- ▶ Instrumentation
- ▶ Level Controllers
- ▶ Math Operators
- ▶ Measurement Tools
- ▶ Message Tools
- ▶ Misc
- ▶ Modulators
- ▶ Networking Tools
- ▶ NOAA
- ▶ OFDM
- ▶ Packet Operators
- ▶ Pager
- ▶ Peak Detectors
- ▶ Resamplers
- ▶ Stream Operators
- ▶ Stream Tag Tools
- ▶ Symbol Coding
- ▶ Synchronizers
- ▶ Trellis Coding
- ▶ Type Converters
- ▶ UHD
- ▶ Variables
- ▶ Video

# Analog Devices Pluto SDR



Specifications	Typical
<i>Power</i>	
DC Input (USB)	4.5 V to 5.5 V
<i>Conversion Performance and Clocks</i>	
ADC and DAC Sample Rate	65.2 kSPS to 61.44 MSPS
ADC and DAC Resolution	12 bits
Frequency Accuracy	±25 ppm
<i>RF Performance</i>	
Tuning Range	325 MHz to 3800 MHz
Tx Power Output	7 dBm
Rx Noise Figure	<3.5 dB
Rx and Tx Modulation Accuracy (EVM)	−34 dB (2%)
RF Shielding	None
<i>Digital</i>	
USB	2.0 On-the-Go
Core	Single ARM Cortex®-A9 @ 667 MHz
FPGA Logic Cells	28k
DSP Slices	80
DDR3L	4 Gb (512 MB)
QSPI Flash	256 Mb (32 MB)
<i>Physical</i>	
Dimensions	117 mm × 79 mm × 24 mm 4.62" × 3.11" × 0.95"
Weight	114 g
Temperature	10°C to 40°C

Fonte: [ADALM-PLUTO SDR - Product Highlight](#)

- Qual o porquê do nome Pluto? É uma referência ao corpo celestial [Plutão](#), um [planeta anão](#), que não possui certos critérios técnicos para ser considerado um planeta. Da mesma maneira, o PLUTO é um dispositivo destinado ao aprendizado, não possuindo alguns critérios técnicos e de desempenho para ser equiparado a um RDS comercial;

Fonte: [Analog Devices - Wiki](#)

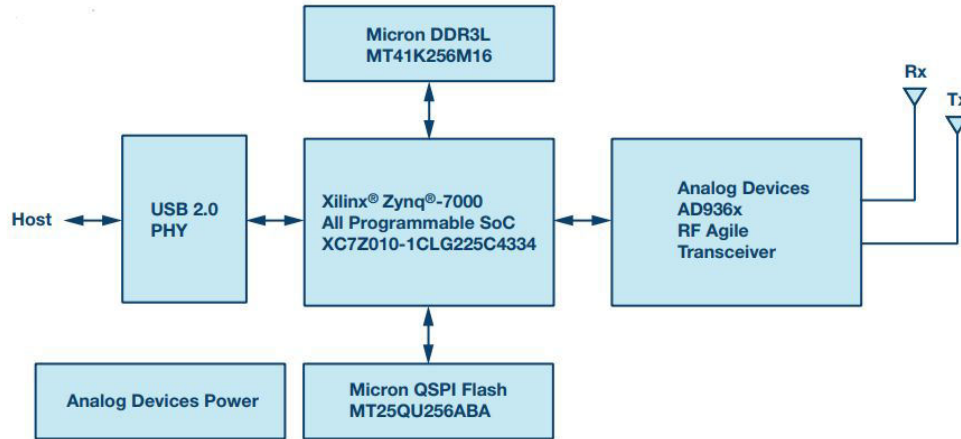
## AD9363

- Transceptor 2x2
- 12-bit DACs and ADCs
- Banda: até 20 MHz
- Frequência: 325 MHz até 3,8 GHz

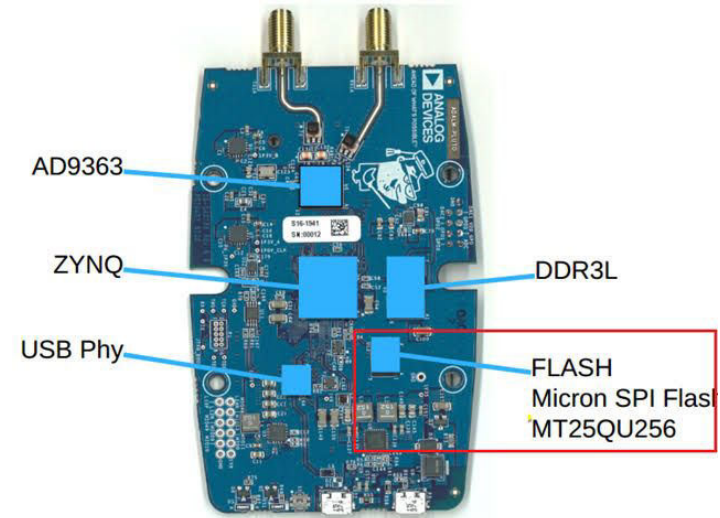
## AD9361

- Transceptor 2x2
- 12-bit DACs and ADCs
- Banda: até 56 MHz
- Frequência: 70 MHz até 6 GHz

# Analog Devices Pluto SDR

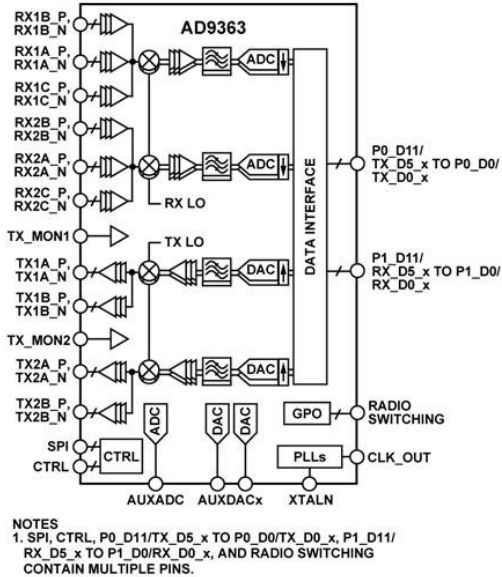


Fonte: [ADALM-PLUTO SDR - Product Highlight](#)

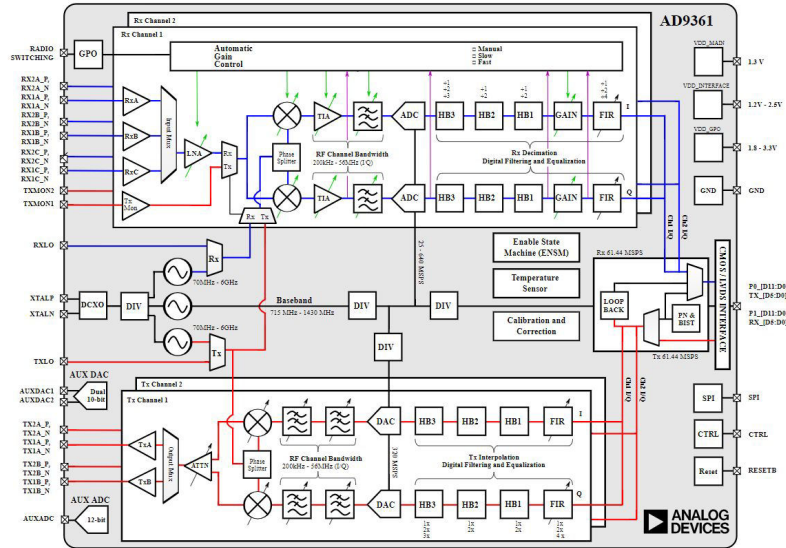


Fonte: [Analog Devices - Engineer Zone](#)

# Analog Devices Pluto SDR



Fonte: [Analog Devices - AD9363 Data Sheet](#)



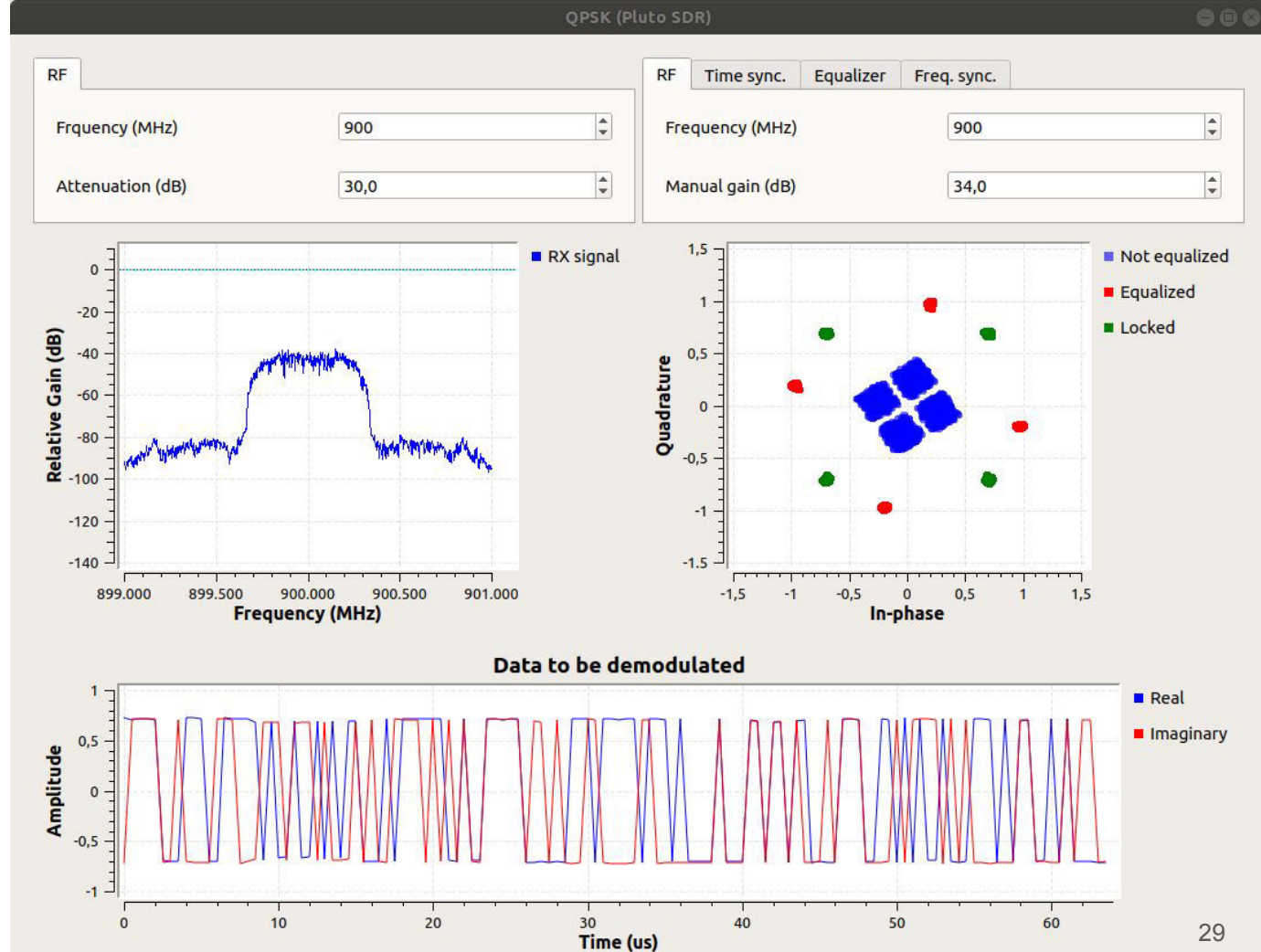
Fonte: [Analog Devices - Wiki AD9361, AD9364 and AD9363](#)

# Transceptor QPSK

- Transmissor e receptor em portadora única com modulação QPSK;
- Projeto a ser desenvolvido no curso.

1.3 Aplicação motivacional:  
demonstração de projetos

Inatel 2020



# Transmissor de TV Digital ISDB-Tb

- Link do projeto: [Link](#)

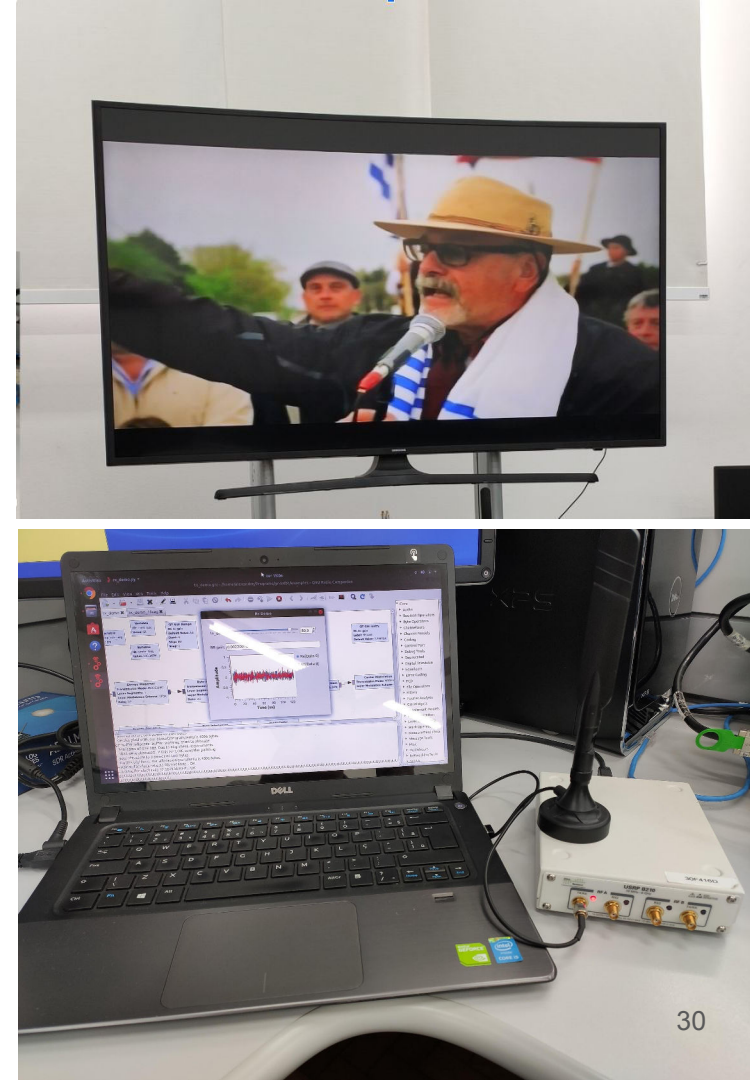
- Instalação:

```
git clone https://github.com/git-artes/gr-isdbt.git
cd gr-isdbt
mkdir build
cd build
cmake ../
make && sudo make install
sudo ldconfig
```

- Para usar o [VOLK](#), executar:

```
volk_profile
```

- Download de arquivos de vídeo (.ts): [Link](#)
- Lista de canais de TV Digital no Brasil: [Link](#)





# Instrumento de medida: receptor Gqrx

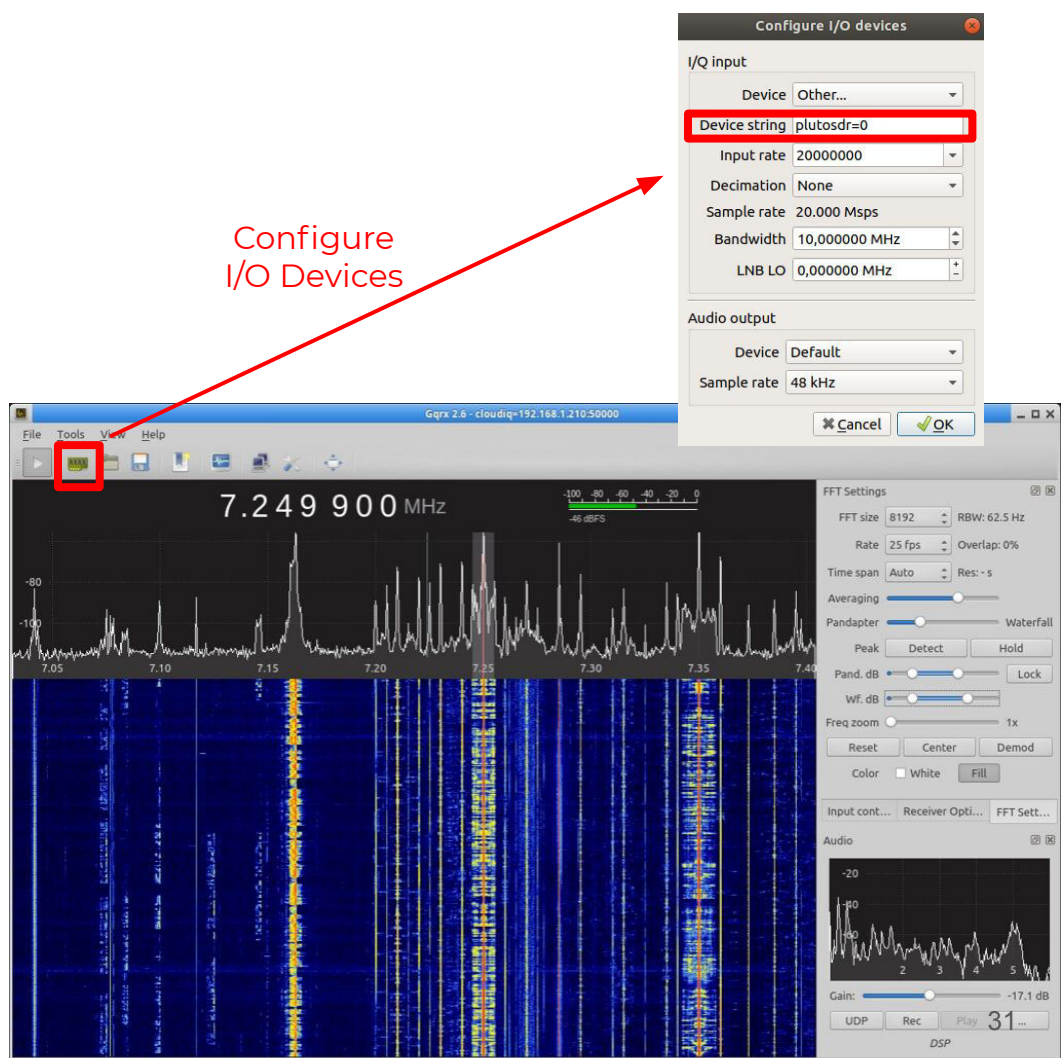
- Link do projeto: [Gqrx](https://github.com/csete/gr-osmosdr-gqrx)
- Instalação:

```
git clone https://github.com/csete/gr-osmosdr-gqrx
cd gr-osmosdr-gqrx/
git checkout plutosdr
mkdir build
cd build/
cmake ../
make
sudo make install
sudo ldconfig
```

```
git clone https://github.com/csete/gqrx.git gqrx.git
cd gqrx.git
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Inatel 2020

1.3 Aplicação motivacional:  
demonstração de projetos

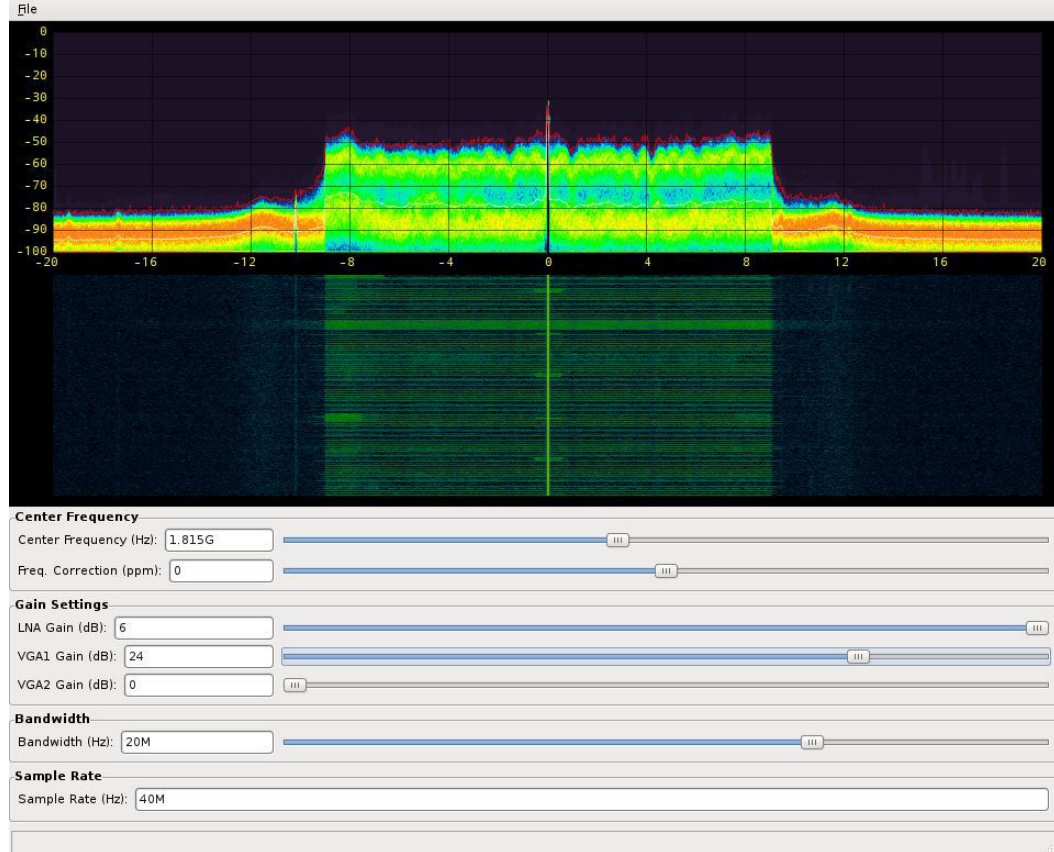


# Instrumento de medida: receptor Fospor



- Link do projeto: [Fosphor](#)
- Características
  - Alto desempenho (tempo real);
  - Utiliza a [GPU](#) do PC;
  - Baseado em [OpenCL](#) e [OpenGL](#);
- Verificar compatibilidade do PC com OpenCL:

```
lspci | grep VGA
```
- De acordo com a [Wikipedia](#), processadores Haswell (HSW-ULT) podem utilizar OpenCL versão 1.2.





# Instrumento de medida: receptor Fospor



- Instalação de dependências:

```
sudo apt install ocl-icd-* opencl-headers  
cmake xorg-dev libglul-mesa-dev
```

- Procurar por dispositivo OpenCL no PC:

```
sudo apt install clinfo  
clinfo
```

- Instalar pacotes para o OpenCL:

```
sudo apt install beignet
```

- Instalação e solução para problema de compartilhamento OpenCL e OpenGL ([link](#)):

```
sudo apt install qt4-default  
sudo apt install libglfw3-dev
```

```
git clone git://git.osmocom.org/gr-fospor  
cd gr-fospor  
git checkout 7b6b9961bc2d9b84daeb42a5c8f8aeba293d207c
```

Procurar pelo arquivo `lib/fospor/cl.c` e no início da função `cl_do_init` comentar a linha `self->flags |= FLG_FOSPHOR_USE_CLGL_SHARING ;`

```
mkdir build  
cd build  
cmake -DCMAKE_INSTALL_PREFIX=/usr ..
```

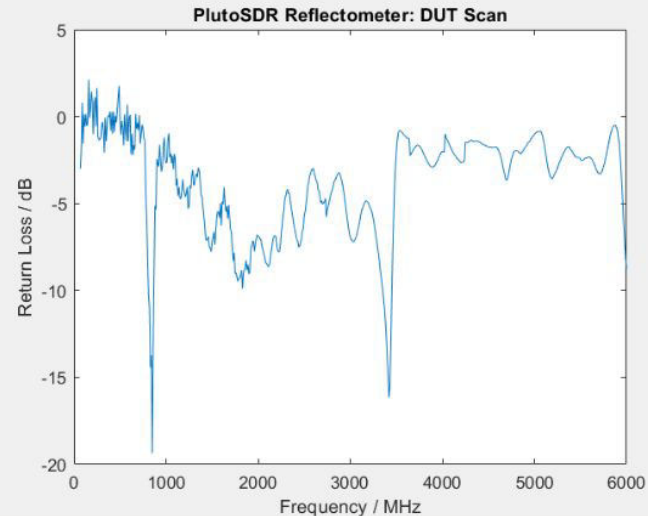
Confirmar que qt e glfw estejam adicionados na compilação.

```
make  
sudo make install  
sudo ldconfig
```

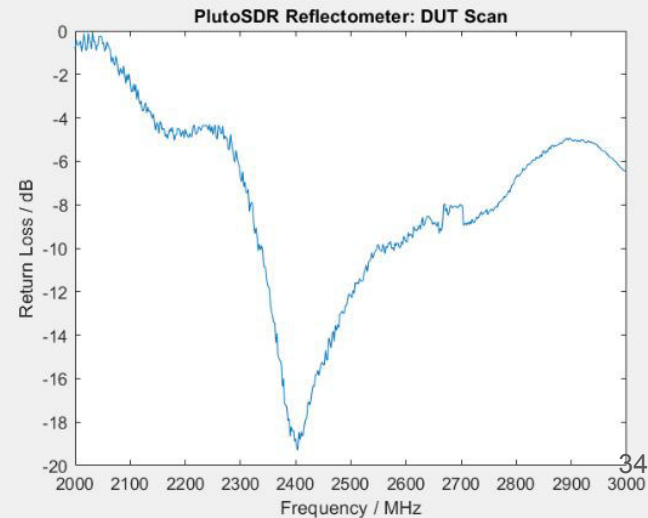
# Instrumento de medida: analisador de sinais

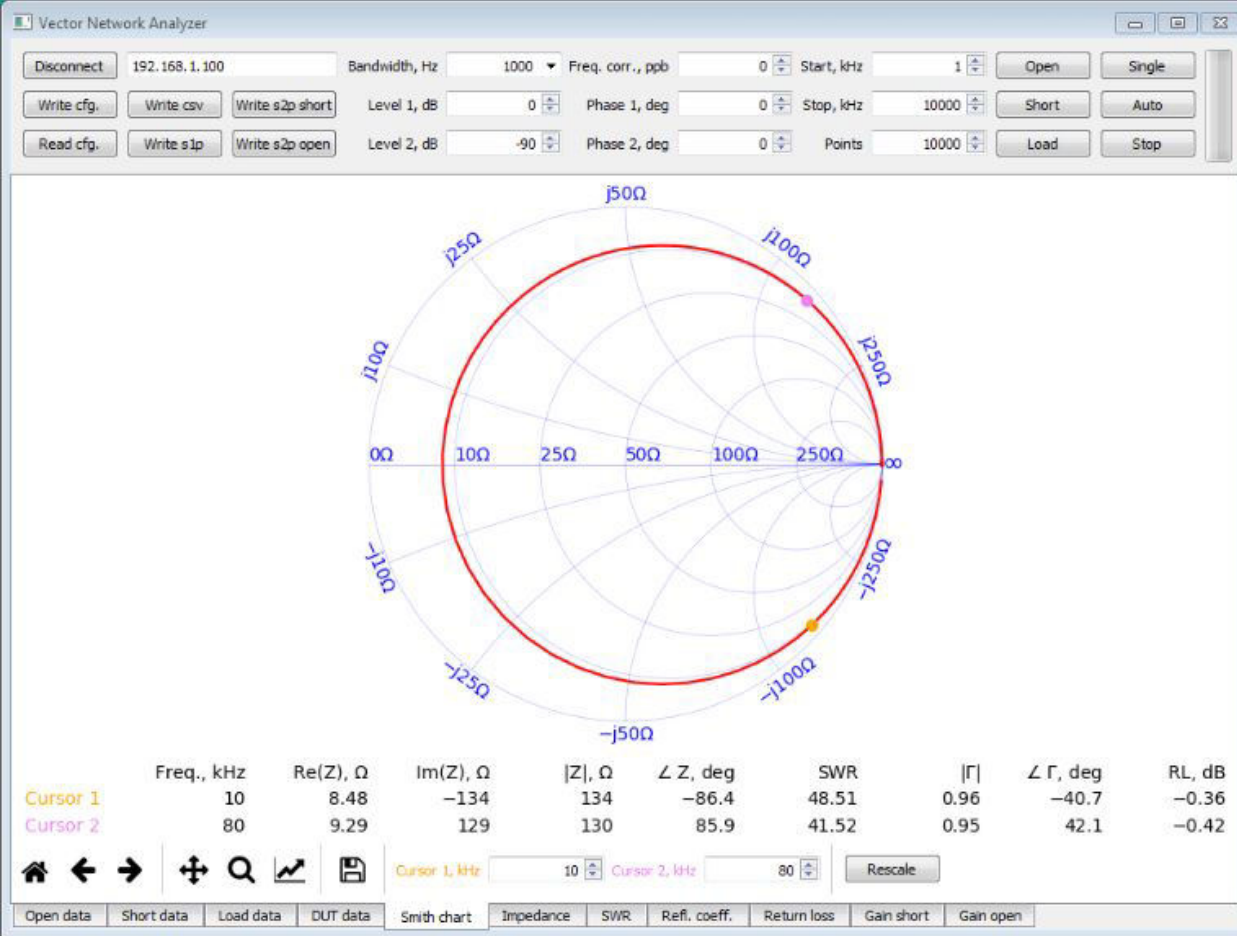
- Link do projeto: [Link](#)
- Medida de Return Loss vs Frequency

Antena do kit da Pluto SDR



Antena de Wi-Fi





# Instrumento de medida: analisador de sinais

- Link do projeto: [Link](#)
- Vector Network Analyzer

# Atividade extraclasses

- Familiarização com a linguagem Python e o pacote NumPy;
- Tutorial de Python: [Link](#);
- Tutoriais de NumPy: [Link 1](#), [link 2](#), [link 3](#).