

# **Samostalni praktični projekat**

## **Slanje i dekodovanje NB-IoT signala upotrebom SDR uređaja**

### **1. Uvod**

Narrowband Internet of Things (NB-IoT) je 3GPP standard koji spada u Low Power Wide Area Network (LPWAN) tehnologije, tj. uređaji mogu međusobno da komuniciraju preko velikih razdaljina upotrebom relativno malo energije. Na fizičkom sloju NB-IoT koristi istu resursnu matricu kao i LTE standard, ali *bandwidth* je značajno redukovana na samo jedan *Physical resource block (PRB)* od 180 kHz da bi se postigla niska kompleksnost (*low-complexity*), niska cena uređaja (*low-cost*) i dug vek trajanja baterije (*long battery life*). NB-IoT je prvi put ušao u standard kroz LTE Release 13.

Ovaj primer objašnjava kako uz pomoć softverskih alata i upotrebom softverski definisanog radija (SDR) generisati, ali i dekodovati jedan NB-IoT *downlink* signal. Cilj praktičnog projekta jeste razumeti raspored i ulogu različitih signala (MIB, SIB) pri NB-IoT komunikaciji. Dodatni teorijski resursi <sup>1</sup>.

### **2. Priprema okruženja**

U narednim primerima kao softverski alat koristiće se *open-source* srsRAN softverski paket koji služi za izvođenje eksperimenata i emulaciju 4G LTE i 5G NR end-to-end sistema<sup>2</sup>.

srsRAN trenutno omogućava simulaciju/emulaciju sledećih komponenti:

- srsUE - *full-stack 4G - 5G Non-standalone UE (User Equipment)*.
- srsENB - *full-stack 4G eNodeB (Baznu stanicu)*
- srsEPC - *4G EPC (Core Network – Jezgro mreže) uključujući sve komponente (MME, HSS and S/P-GW)*

Sve pomenute komponente mogu da se koriste na Linux operativnim sistemima i uz pomoć *off-the-shelf* SDR uređaja.

Pre instalacije softverskog paketa, potrebno je instalirati neophodne biblioteke.

Na Ubuntu sistemima neophodne biblioteke mogu da se instaliraju sledećom komandom:

```
$ sudo apt-get install build-essential cmake libfftw3-dev libmbedtls-dev libboost-program-options-dev libconfig++-dev libsctp-dev
```

Ako se koristi Fedora:

```
dnf install cmake fftw3-devel mbedtls-devel lksctp-tools-devel libconfig-devel boost-devel
```

<sup>1</sup> [https://cdn.rohde-schwarz.com/pws/dl\\_downloads/dl\\_application/application\\_notes/1ma266/1MA266\\_0e\\_NB\\_IoT.pdf](https://cdn.rohde-schwarz.com/pws/dl_downloads/dl_application/application_notes/1ma266/1MA266_0e_NB_IoT.pdf)

<sup>2</sup> <https://www.srsran.com/>

Za centOS koristiti istu komandu kao i za Fedora, ali umesto *libconfig-devel* koristiti *libconfig*.

Download i build srsRAN softvera:

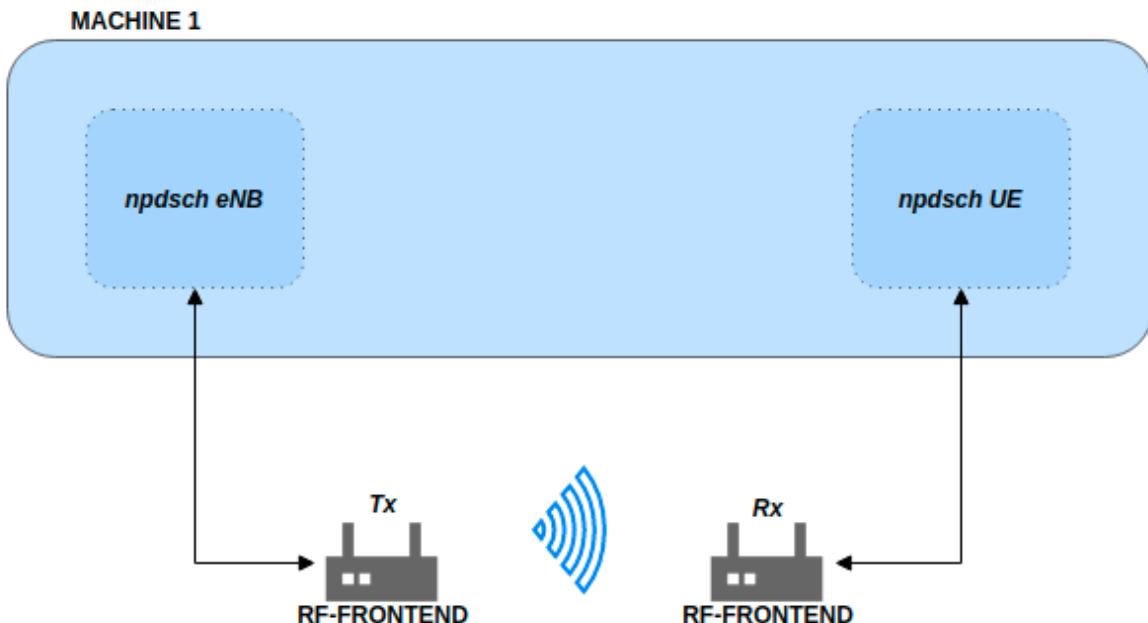
```
git clone https://github.com/srsRAN/srsRAN.git
cd srsRAN
mkdir build
cd build
cmake ../
make
make test
```

Proizvoljno mogu da se pozovu i sledeće komande:

```
sudo make install
sudo srsran_install_configs.sh user
```

### 3. Slanje i dekodovanje NB-IoT signala

Na slici 1 prikazana je arhitektura sistema potrebna za slanje i primanje NB-IoT *downlink* signala.



Slika 1, Arhitektura sistema<sup>3</sup>

Jedan SDR uređaj predstavlja NB-IoT baznu stanicu (eNB) koja će da generiše *downlink* signal, dok drugi SDR uređaj predstavlja UE (korisnički uređaj) koji će da primi i dekoduje takav signal.

Svi primeri u sklopu srsRAN softverskog paketa nalaze se u `srsRAN/build/lib/examples` direktorijumu. Za pokretanje bazne stanice potrebno je pozvati sledeću komandu:

<sup>3</sup> [https://docs.srsran.com/en/latest/\\_images/nbIoT\\_TxRx.png](https://docs.srsran.com/en/latest/_images/nbIoT_TxRx.png)

```

$ ./lib/examples/npdsch_enodeb -f 868e6
Opening RF device...
[INFO] [UHD] linux; GNU C++ version 8.3.0; Boost_106700; UHD_3.13.1.0-3build1
[INFO] [LOGGING] Fastpath logging disabled at runtime.
[INFO] [B200] Loading firmware image: /usr/share/uhd/images/usrp_b200_fw.hex...
Opening USRP channels=1, args: type=b200,master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B200mini
[INFO] [B200] Loading FPGA image: /usr/share/uhd/images/usrp_b200mini_fpga.bin...
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting sampling rate 1.92 MHz
Set TX gain: 70.0 dB
Set TX freq: 868.00 MHz
NB-IoT DL DCI:
- Format flag: 1
+ FormatN1 DCI: Downlink
- PDCCH Order: 0
- Scheduling delay: 0 (0 subframes)
- Resource assignment: 0
+ Number of subframes: 1
- Modulation and coding scheme index: 1
- Repetition number: 0
+ Number of repetitions: 1
- New data indicator: 0
- HARQ-ACK resource: 1
- DCI subframe repetition number: 0
DL grant config:
- Number of subframes: 1
- Number of repetitions: 1
- Total number of subframes: 1
- Starting SFN: 0
- Starting SF index: 6
- Modulation type: QPSK
- Transport block size: 24

```

Parametar **-f** služi da definišemo frekvenciju na kojoj će *downlink* signal biti emitovan, u ovom primeru na 868 MHz. Bazna stanica (eNB) će u ovom primeru da odašilje standardnom definisan *downlink* signal sa MIB-NB i SIB-NB1 signalima.

Ovako emitovan signal može da se primi i dekoduje uz pomoć drugog SDR uređaja, i to sledećom komandom:

```
$ ./lib/examples/npdsch_ue -f 868e6 -r 0x1234 -s
```

```
Opening RF device...
Found Rafael Micro R820T tuner
Soapy has found device #0: driver=rtlsdr, label=Generic RTL2832U
OEM :: 00000001, manufacturer=Realtek, product=RTL2838UHIDIR,
serial=00000001, tuner=Rafael Micro R820T,
Selecting Soapy device: 0
[INFO] Opening Generic RTL2832U OEM :: 00000001...
Found Rafael Micro R820T tuner
Setting up Rx stream with 1 channel(s)
[INFO] Using format CF32.
[R82XX] PLL not locked!
Available device sensors:
Available sensors for Rx channel 0:
State of gain elements for Rx channel 0 (AGC supported):
- TUNER: 0.00 dB
State of gain elements for Tx channel 0 (AGC supported):
- TUNER: 0.00 dB
Rx antenna set to RX
Tx antenna set to RX
Set RX gain: 40.0 dB
Set RX freq: 868.000000 MHz
Setting sampling rate 1.92 MHz
NSSS with peak=24.363365, cell-id: 0, partial SFN: 0
*Found n_id_ncell: 0 DetectRatio= 0% PSR=8.66, Power=86.4 dBm
MIB received (CFO: -1,55 kHz) FrameCnt: 0, State: 10
SIB1 received
CFO: -1,41 kHz, RSRP: 12,0 dBm SNR: 19,0 dB, RSRQ: -3,7 dB, NPDCCH
detected: 510, NPDSCH-BLER: 0,20% (1 of total 511), NPDSCH-Rate:
10,36 kbit/s
```

## **Samostalni praktični projekat**

### **Slanje i prijem signala upotrebom SDR uređaja i GNU Radio softvera**

According to the definition given by Federal Communications Commission (FCC): “Cognitive radio: A radio or system that senses its operational electromagnetic environment and can dynamically and autonomously adjust its radio operating parameters to modify system operation, such as maximize throughput, mitigate interference, facilitate interoperability and access secondary markets.” A CR is an SDR that is aware of its environment, internal state, and location, and autonomously adjusts its operations to achieve designated objectives.”

With today's ambition for more wireless devices and systems, the need for spectral resources to accommodate these systems grows every day, but the remaining spectrum resources seems to be running out. Cognitive radio technology offers a revolutionary solution to this crisis by looking at the spectrum in an unconventional manner as a multi-dimensional space. The current wireless communication systems are governed by fixed spectrum assignment policy, which was proven by measurement to be inefficient. Cognitive radio enables maximum spectrum utilization efficiency through opportunistic access to temporarily unutilized portions of spectrum. Cognitive radios have the ability to analyze the surrounding radio spectrum to locate spectrum access opportunities, use a smart technique based on cumulative learning to decide which opportunity to use, in a manner that does not cause the surrounding systems any harm. A cognitive radio is highly flexible in a manner that allows it to coexist with any communication system, adapt to any channel condition by varying its operation parameters such as modulation scheme and channel coding to achieve the highest possible quality of service.

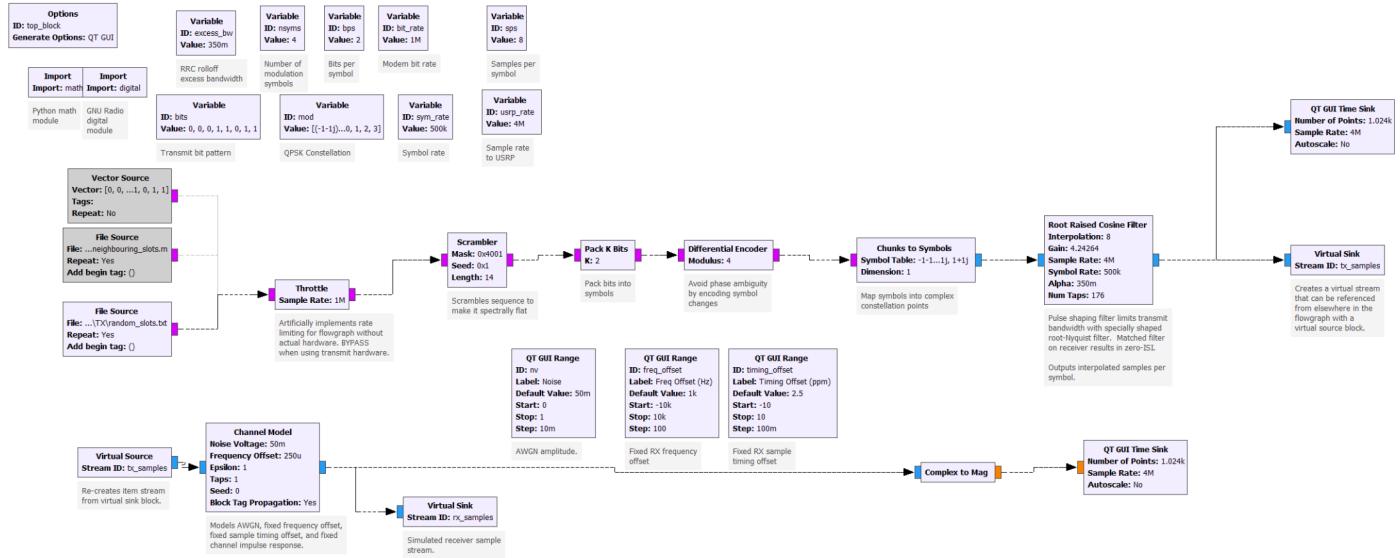
First class introduces the student to GNU radio and Matlab for cognitive radio experimentation. The Introduction section provides an overview of software defined radio and the GNU Radio toolkit. The hardware commonly used with this software, the Universal Software Radio Peripheral (USRP), is also introduced. The analog communication section is focused on the Python level, introducing some Python basics and how Python is used in GNU Radio to connect signal-processing blocks and control the flow of the digital data. The most common type of modern communication is digital. Building upon the analog communication section, a digital radio is developed to transmit a randomly generated bitstream. Last part shows how we can use Matlab with USRP devices.

#### **4. What is GNU Radio?**

GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in research, industry, academia, government, and hobbyist environments to support both wireless communications research and real-world radio systems.

It is a software toolkit for signal processing

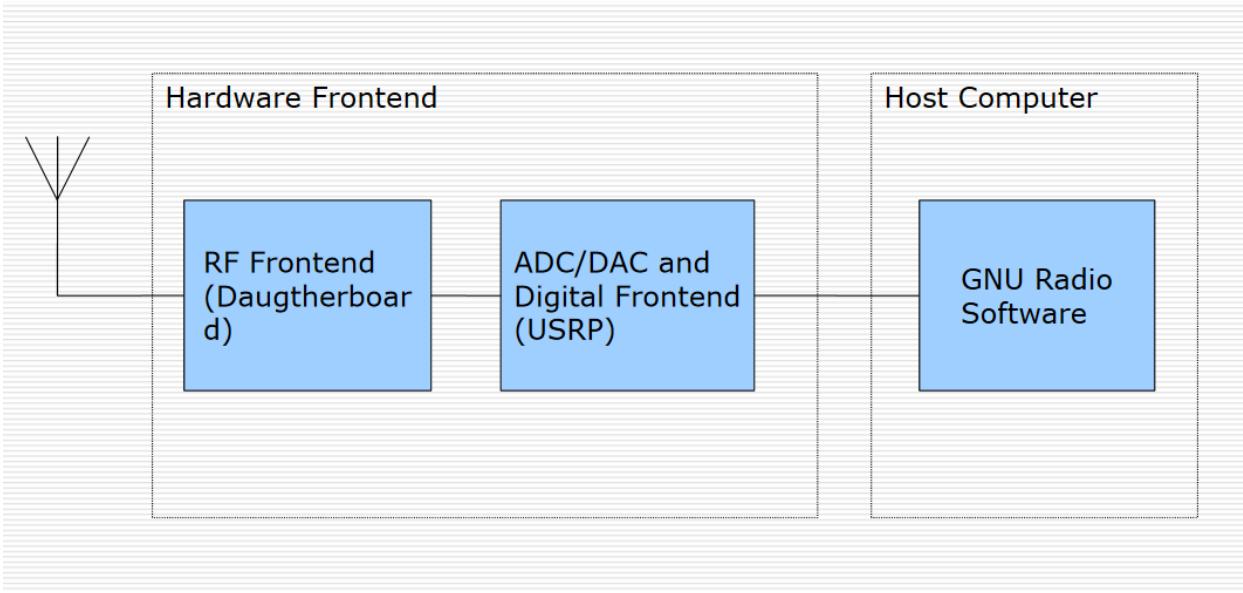
- Software radio construction
- Rapid development
- Cognitive radio



USRP (Universal Software Radio Peripheral) is a hardware frontend for sending and receiving waveforms.



## 5. Gnu Radio Components

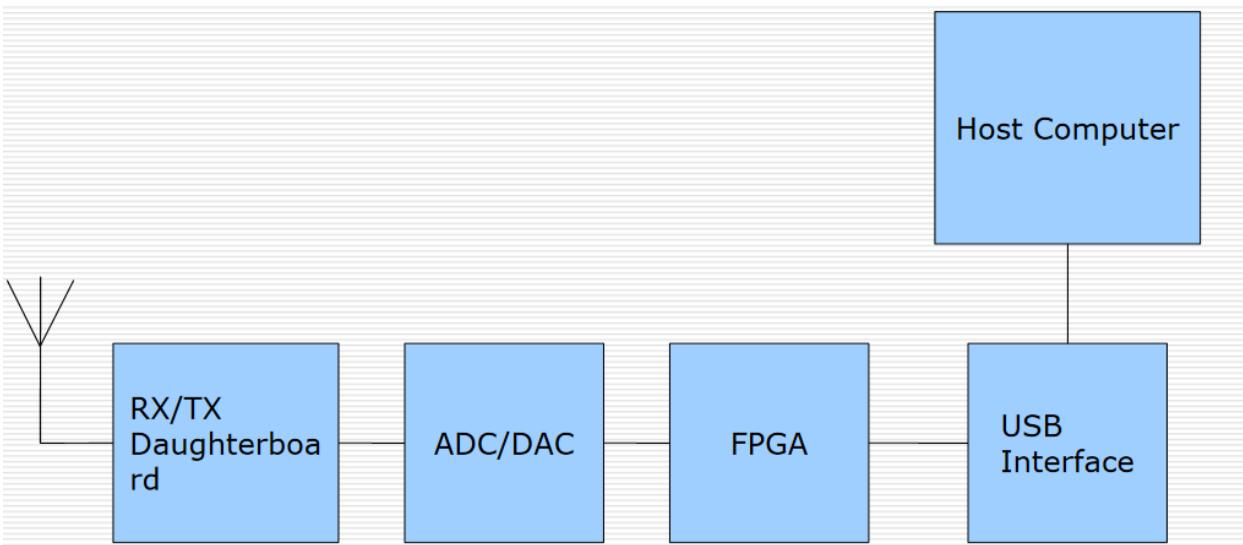


Gnu Radio Software is an opensource software with existing examples like 802.11b, Zigbee, ATSC, OFDM, DBPSK, DQPSK.

Features of GNU Radio:

- Extensive library of signal processing blocks(C++/ and assembly)
- Python environment for composing blocks (flow graph)

## 6. GNU Radio hardware schematic



## 7. Blocks

Signal Processing Block

- Accepts 0 or more input streams
- Produces 0 or more output streams

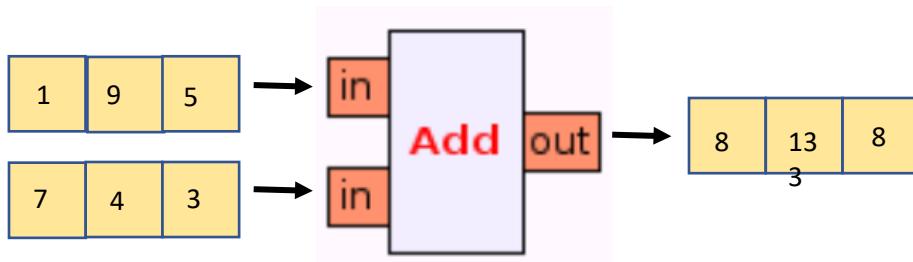
Source: No input

- Noise source, signal source, USRP source

Sink: No outputs:

- Audio sink, USRP sink

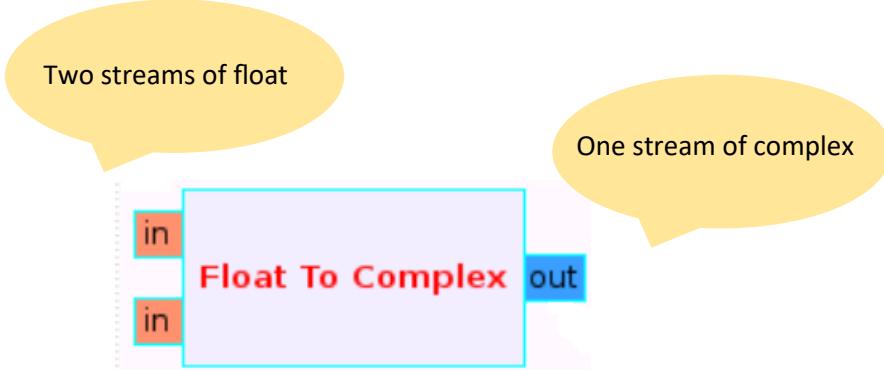
Blocks operate on streams of data



Blocks operate on certain data types :

- Char, short, int, float, complex
- Vectors

We have Input and output signatures which are data types for input and output streams.

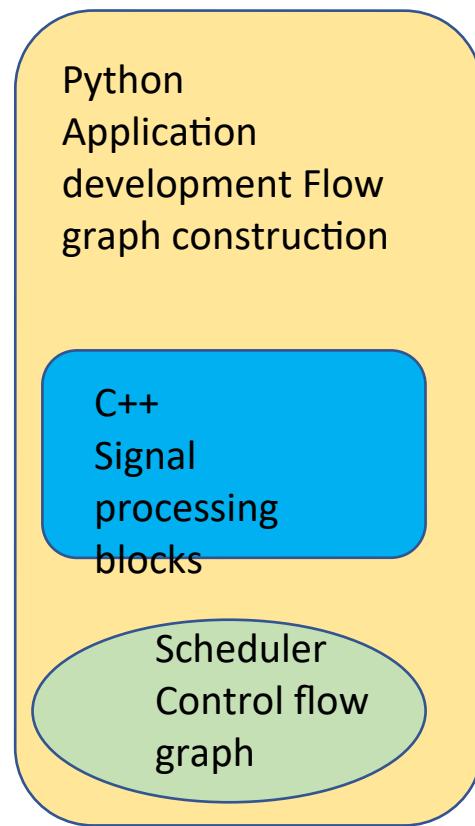


Data stream flow from source to sink

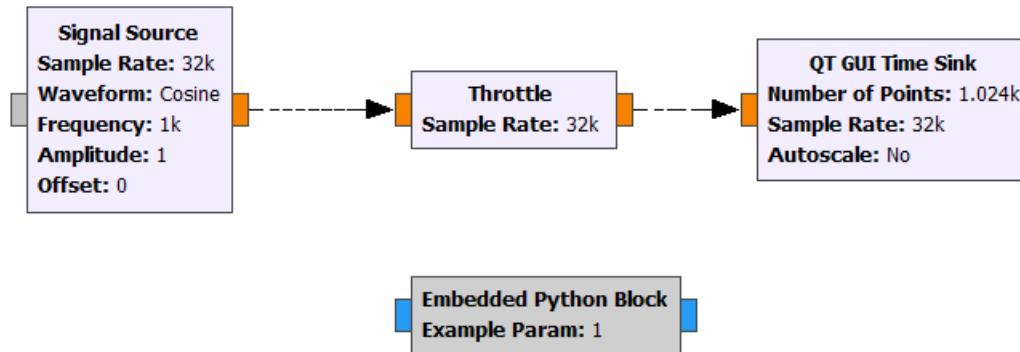
What we are interested in the first part is how to use the existing modules that has been provided in GNU radio, to show how communicate two end devices.

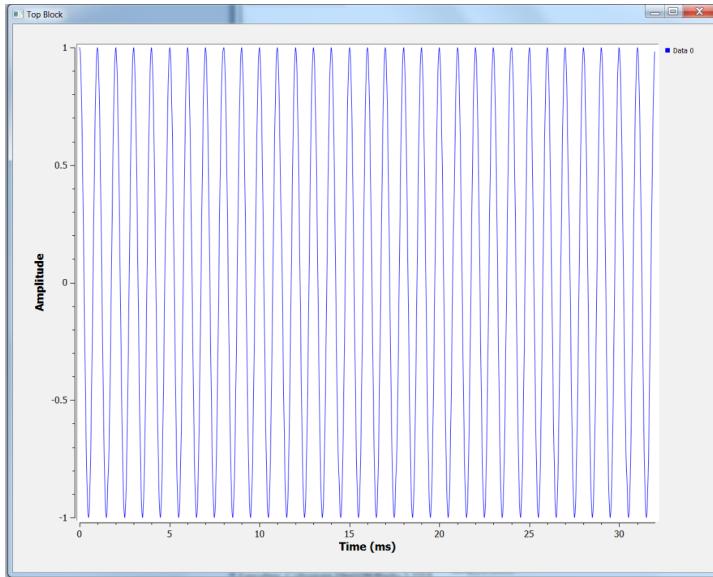
Python scripting language is used for creating “signal flow graphs”. C++ is used for creating signal processing blocks (we already have an existing library of signaling blocks).

The scheduler is using Python’s built-in module threading, to control the starting, stopping or waiting operations of the signal flow graph.



## 8. Digital Communications: Transmitter





### Original code of Python block:

```

import numpy as np

from gnuradio import gr

class blk(gr.sync_block): # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, example_param=1.0): # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block', # will show up in GRC
            in_sig=[np.complex64],
            out_sig=[np.complex64]
        )
        # if an attribute with the same name as a parameter is found,
        # a callback is registered (properties work, too).
        self.example_param = example_param

```

```

def work(self, input_items, output_items):
    """example: multiply with constant"""

    output_items[0][:] = input_items[0] * self.example_param

    return len(output_items[0])

```

**After modification Python block:**

```

import numpy as np

from gnuradio import gr

from gnuradio import audio, analog

```

```

class blk(gr.sync_block): # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

```

```

def __init__(self, example_param=1.0): # only default arguments here

    """arguments to this function show up as parameters in GRC"""

    gr.sync_block.__init__(
        self,
        name='Thingy', # will show up in GRC
        in_sig=[np.float32],
        out_sig=[np.float32]
    )

    # if an attribute with the same name as a parameter is found,
    # a callback is registered (properties work, too).

    self.example_param = example_param

```

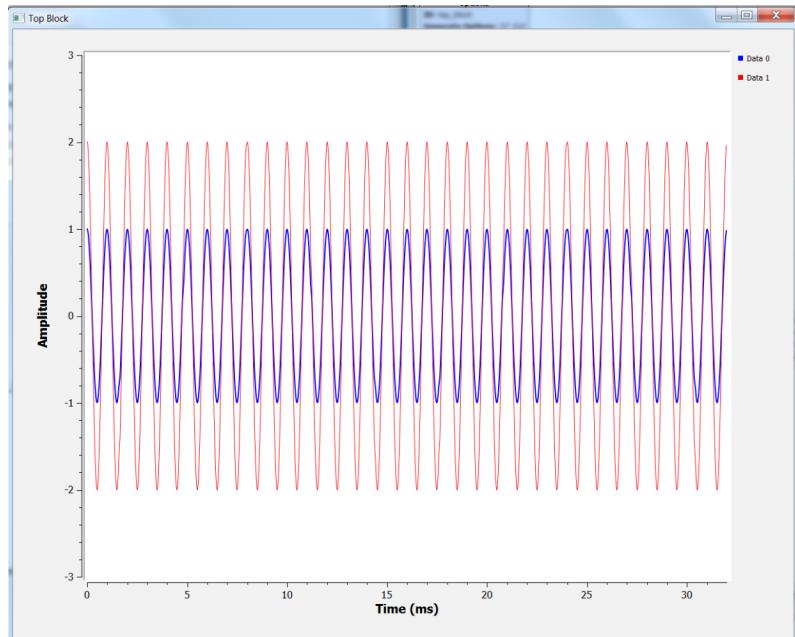
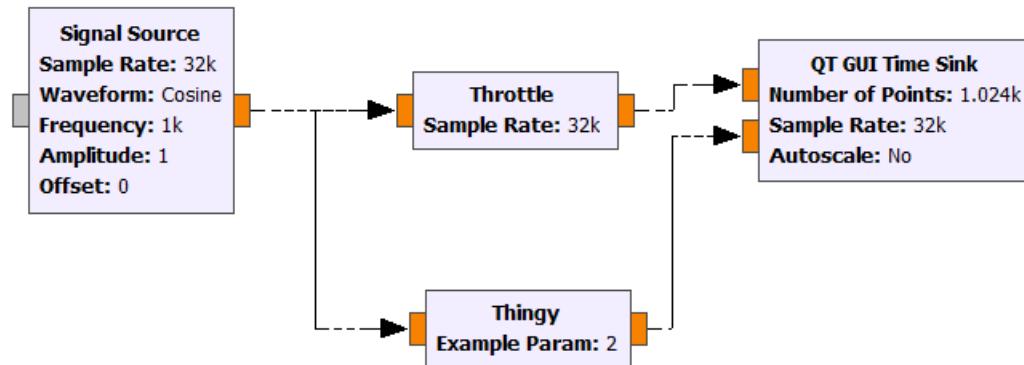
```

def work(self, input_items, output_items):
    """example: multiply with constant"""

    output_items[0][:] = input_items[0] * self.example_param

```

```
return len(output_items[0])
```



## 9. Matlab and USRP

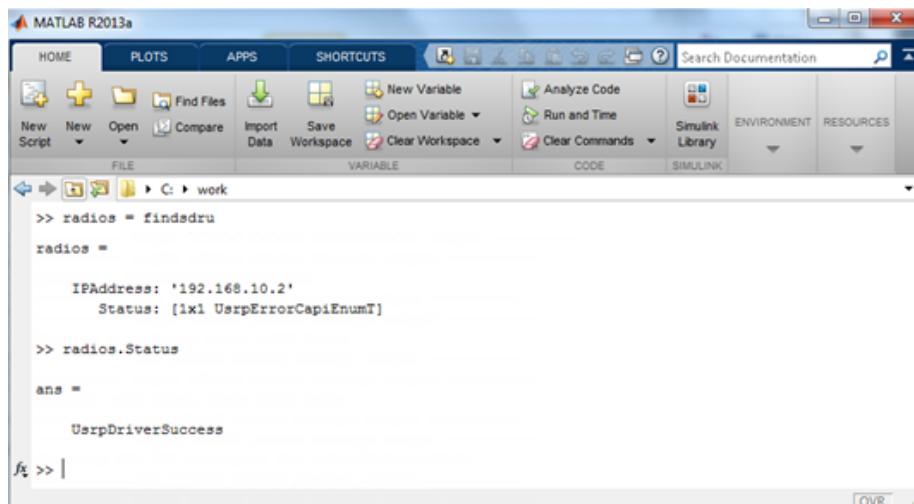
If you are not sure which USRP® [1] radio devices are connected to your computer, you can use a helper function to find them.

Type the following at the MATLAB® command line:

```
radios = findsdru
```

The variable, radios, is a structure that contains information on the USRP® radios connected to the host computer.

- If you get a successful status, it means that MATLAB can communicate with the USRP® radio and the radio is ready to be used.
- If the function cannot find a radio, MATLAB returns an empty IPAddress field or a status other than Success.
- If the function finds one or more radios, MATLAB displays a message similar to the following.



A screenshot of the MATLAB R2013a interface. The window title is "MATLAB R2013a". The toolbar includes icons for New Script, New, Open, Find Files, Import Data, Save Workspace, New Variable, Open Variable, Analyze Code, Run and Time, Clear Commands, Simulink Library, Environment, and Resources. The command window shows the following text:

```
>> radios = findsdru
radios =
    IPAddress: '192.168.10.2'
    Status: [1x1 UsrpErrorCapiEnumT]

>> radios.Status

ans =
    UsrpDriverSuccess
```

- The IPAddress field is the IP address of the USRP® radio.
- The Status field is the status of the radio. Type radios.Status in the command window and press enter to see status value.

In this example, the USRP® IP address is 192.168.10.2 and its status is USRPDriverSuccess. The successful status indicates that MATLAB can communicate with the USRP® radio and the radio is ready to be used.

After you have found radios attached to your computer, you can use the function [probesdru](#) to get detailed information about a particular USRP® radio.