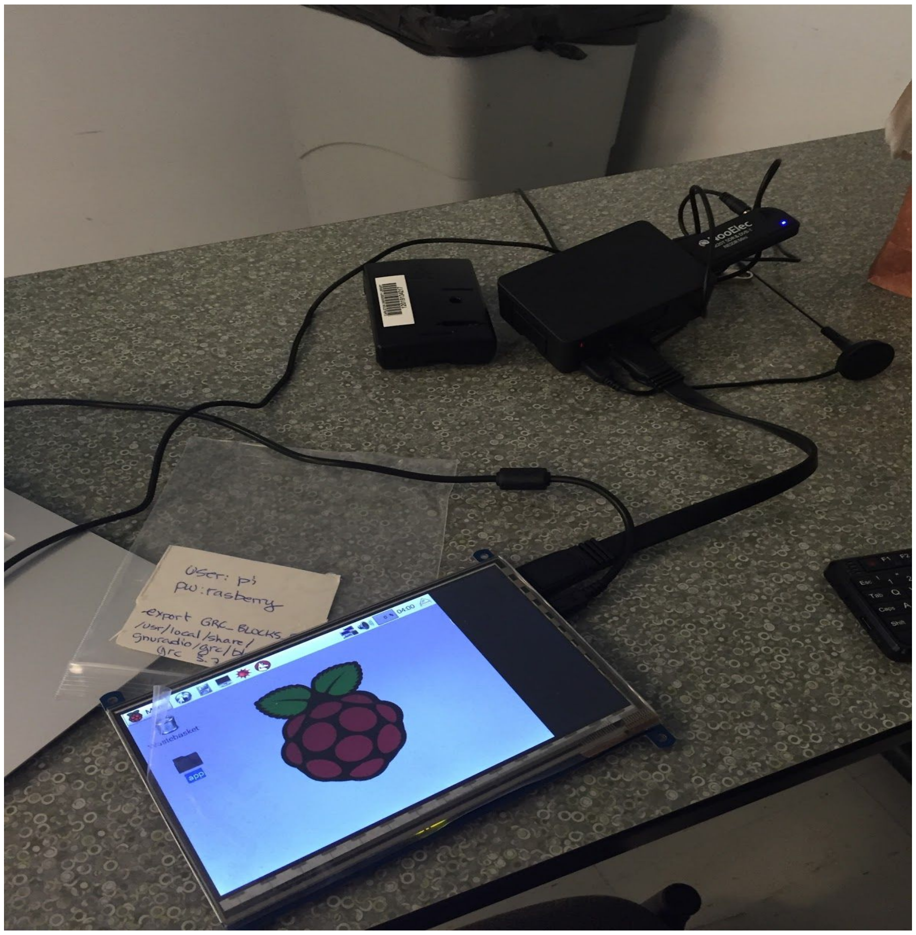


Portie

Project report

Margarita Otochkina

1. Introduction

1.1 Context

Portie is a Software Defined Radio (SDR) Frequency Modulation(FM) Receiver application for Raspberry Pi.

The formal definition of SDR is “*Radio in which some or all of the physical layer functions are software defined"*[1]. Our SDR app is developed using USB receiver with antenna and Raspberry Pi 3 as a hardware part and GNU Radio and GNU Radio Companion(GRC) as a software tool for writing some of the physical layer functions.

The project is designed for Raspberry Pi but the idea is to make the app light enough to run on microcomputers so that it could be extended for a specific purposes (military, expeditions).

1.2 Problem statement

The idea of a project is to create a portable radio using as less hardware and spending as less money for it as possible. USB TV receiver, which price is in average 20$, and Raspberry Pi, microcomputer of a price 50$, are the perfect tools.

The application for the radio is supposed to have a simple interface, which includes an option to choose a frequency. The feature of an application is to save and remove the station they like or want to use in the future.

At the same time it should be extendable so that users familiar with the software and SDR would be able to take the application’s source code as a base and use it for their own purposes.

1.3 Result

The result is a small and simple FM Receiver app that runs on Raspberry Pi. The applications, as it was described in a previous section, allows the user to choose a frequency in a certain range and save it or remove from saved.

1.4 Outline

The rest of the report is structured as follows:

Section 2 describes the background information and preparation process for the project. The preparation process includes a guide describing how to install all of the required software for the project on Raspberry Pi.

Section 3 describes the obtained result in details: how the final version of application works, what are the weak and strong sides, which problems have been faced and solved during the development.

Section 4 includes the evaluation of the result.

We conclude in Section 5.

2. Background Information

2.1 SDR - Software Defined Radio

As it was stated in Introduction section, SDR is a collection of hardware and software technologies where some of the radio functions are implemented through software technologies.

Comparing to a traditional hardware radio, where the components can be modified only through physical intervention, SDR is an efficient and relatively low-cost solution.

Some of the examples of radio's operation functions that has been implemented through software are:
1. Field Programmable Gate Arrays (FPGA)
2. Digital Signal Processor(DSP)
3. General Purpose Processors (GPP)

and many more.

SDR allows to reduce the cost in providing a wireless communication to users.

2.2 GNU Radio and GNURadio-Companion

2.2.1 GNU Radio

From the official GNU Radio website [2], "Gnu radio is a free and open-source software development toolkit that provides signal-processing blocks to implement SDR."

It allows to develop SDR apps without any hardware for a simulation purposes or using affordable hardware like dongles(USB receiver and antenna) or more advanced ones like USPR.

GNU Radio performs all the signal processing. It could be used for writing a FM receiver that uses hardware as a source and a computer's sound card as an output or FM Transmitter than will push the data kept in a computer to digital streams and then transmits it using hardware.

GNU Radio has different elements(blocks) including: filters, channel codes, equalizers, demodulators, decoders.

Applications that are developed using GNU Radio are primarily written in Python. However, if the person wants to write his own block or modify the source code, it has to be done using C++.

2.2.2 GNU Radio Companion

GNU Radio Companion(GRC) is a graphical tool for creating signal flow graphs and generating flow-graph source code. [3]

GRC allows to create SDR applications without a knowledge of programming. It has a list of blocks which you can drag into your diagram, connect to other blocks and modify its settings.

Once the user run the diagram, the program generates a Python source code with could be modified to add functionality.

2.3 Hardware

The project requires a number of hardware pieces which are:

1. Raspberry Pi (B+) with SD Card and Screen
2. USB Receiver and antenna (NooElec NESDR Mini USB RTL-SDR & ADS-B Receiver Set is used for the project)

2.3.1 Raspberry Pi and parts

Raspberry Pi is a series of credit-card sized and single board computers developed in England, UK [4]. They are designed to be used for an education purposes promoting computer science basics for school students but also used for different kinds of projects.

Several generations of Raspberry Pi have been released: Pi 1, Pi 1 A+, B+, Pi 2 and Pi 3. The version used for a project is Pi 3.

Raspberry Pi goes with an 8GB SD Card that is used as a storage for an operating system (OS). The OS used for a project will be mentioned in the next section.

Also, to add portability, we use 7’’ touch screen. Since touch feature is not important in our case, it will not be mentioned anymore.

2.3.2 USB Receiver and Antenna

The possibility to use Realtek 2832U tuners has been discovered by Eric Fry, who used to to sniff the USB packets from Windows application in FB and DAB mode in March, 2010. [5]

Realtek 2832U tuners are the cheapest devices available for SDR. The receiver used for this project is NooElec NESDR Mini USB RTL-SDR & ADS-B Receiver Set with R820T Tuner.

2.4 Other tools and software

The OS used for the project is Raspbian Jessie version March 2016 with Kernel version 4.1. A clear image was used for a project to which all of the required software has been installed. The guide describing the process will be given in Appendix.

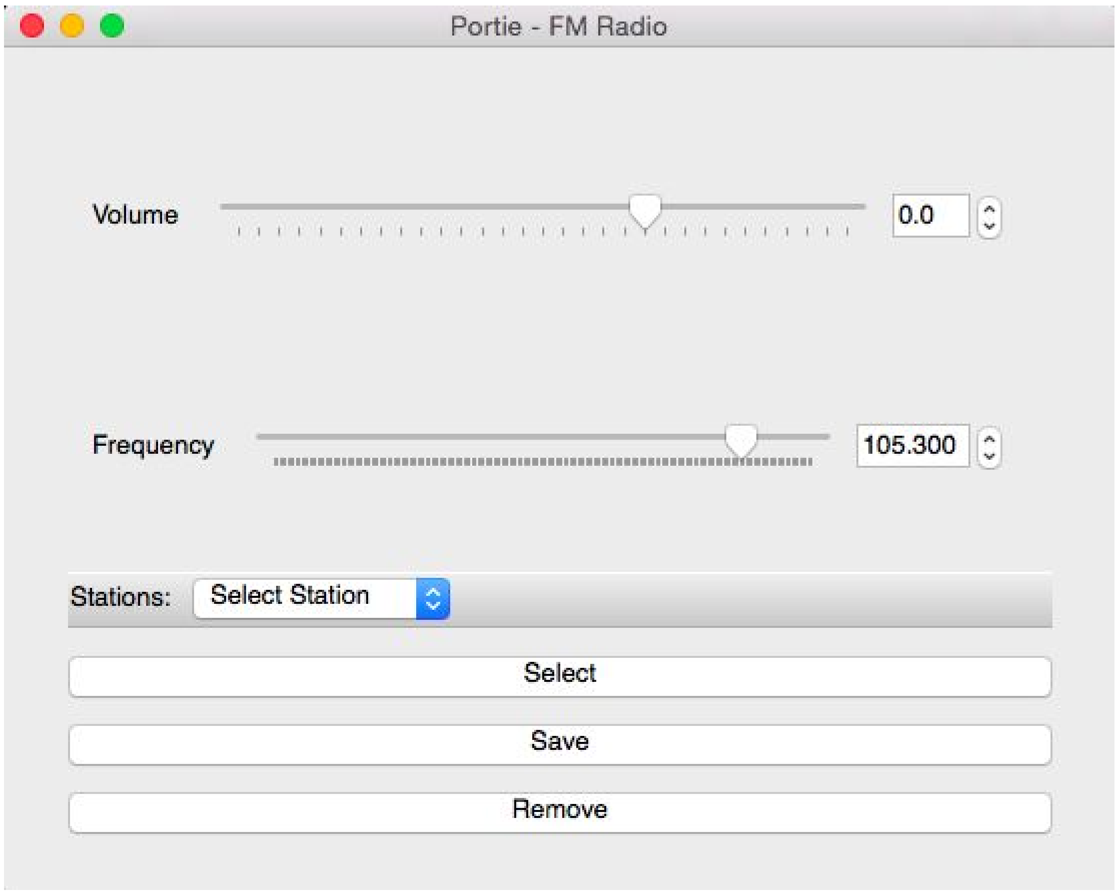
In order for our app to use a tuner with GNU Radio, rtl_sdr package provided by OsmoSDR has to be installed. The package includes librtlsdr library, command line tools and GNU Radio collection of tools. [6]
Rtl_sdr requires a libusb to be installed. Libusb is a C library that allows the applications to easily access USB devices on many different OS [7].

Portie, the application itself, is developed in Python using GUI libraries like QT.

SQLite database is used as a storage for the application. “SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed database engine in the world. The source code for SQLite is in the public domain. “ - according to SQLite official website[8].

3. Result

Pic 1: Portie app - interface



The result of the work is an FM receiver application called Portie. It allows the user to pick a station in a range from 88.1 to 107.9, save the station they like, select the station from a list of saved ones, remove a station they don't need anymore and change the volume.

The application's parts and details will be discussed below.

3.1 GRC - Flow Diagram

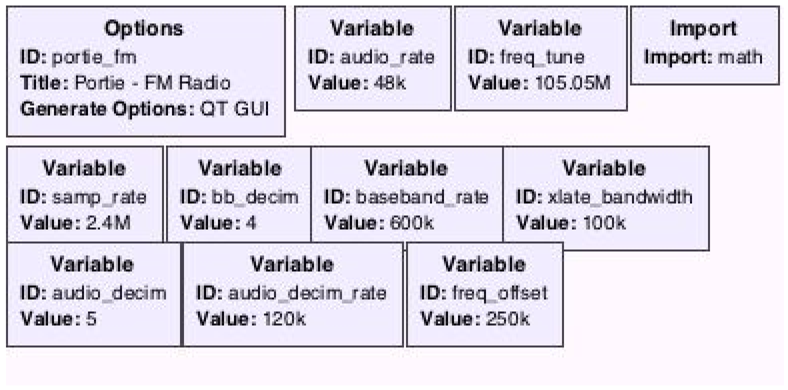
The base of the application is a python code generated by GRC.

In GRC, we create a flow diagram using blocks provided by the application and once the blocks are connected and the diagram has no errors, we execute it. During the execution, GRC generates a python source code which has been taken as a base for the application.

Variables

First, we start with declaring variables we need for the diagram.

Pic 2: Variables



Options
ID: portie_fm
Title: Portie - FM Radio
Generate Options: QT GUI

Variable
ID: audio_rate
Value: 48k

Variable
ID: freq_tune
Value: 105.05M

Import
Import: math

Variable
ID: samp_rate
Value: 2.4M

Variable
ID: bb_decim
Value: 4

Variable
ID: baseband_rate
Value: 600k

Variable
ID: xlate_bandwidth
Value: 100k

Variable
ID: audio_decim
Value: 5

Variable
ID: audio_decim_rate
Value: 120k

Variable
ID: freq_offset
Value: 250k

Options is the first and main block we need. In a source code it is our class that will contain all of the code for the application.

samp_rate is a sample rate, which is the number of samples of audio carried per second, measured in Hz or kHz, according to Audacity.[9] For the source(which is RTL_SDR receiver in our case), it set to 2.4M for Mac OS and 1M for Raspbian due to the fact that with sample rate higher that 1M the sound quality decreases and the sound turns into noise.

audio_decim is an audio decimation, which is a factor by which the sample rate is reduced.

bb_decim is a baseband decimation factor.

audio_rate is an audio sampling rate. According to Wiki, the sampling rate recommended by Audio Engineering Society is 48 kHz.[10]

Freq_tune is a default frequency to which our application is tuned. It is defined by a formula: frequency * 1M - frequency offset.

Baseband_rate is calculated by the formula: sample rate / baseband decimation factor.

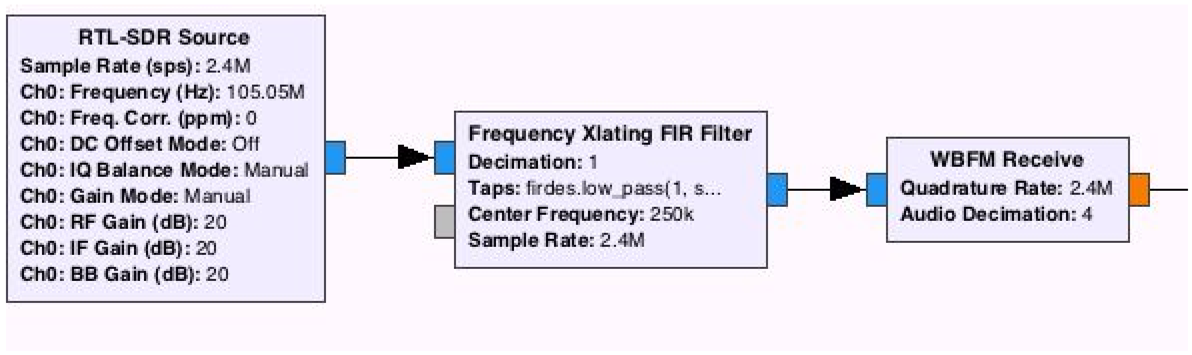
Audio_decim_rate is an audio decimation rate that is calculated by the formula: baseband rate / audio decimation factor.

Xlate_bandwidth is a bandwidth value used for frequency-translating finite impulse response(FIR) filter.

Freq_offset is the difference between receive and transmit frequencies of a radio channel.

Receiving the signal

Pic 3: source


RTL-SDR Source
Sample Rate (sps): 2.4M
Ch0: Frequency (Hz): 105.05M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Manual
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 20
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Frequency Xlating FIR Filter
Decimation: 1
Taps: firdes.low_pass(1, s...
Center Frequency: 250k
Sample Rate: 2.4M

WBFM Receive
Quadrature Rate: 2.4M
Audio Decimation: 4

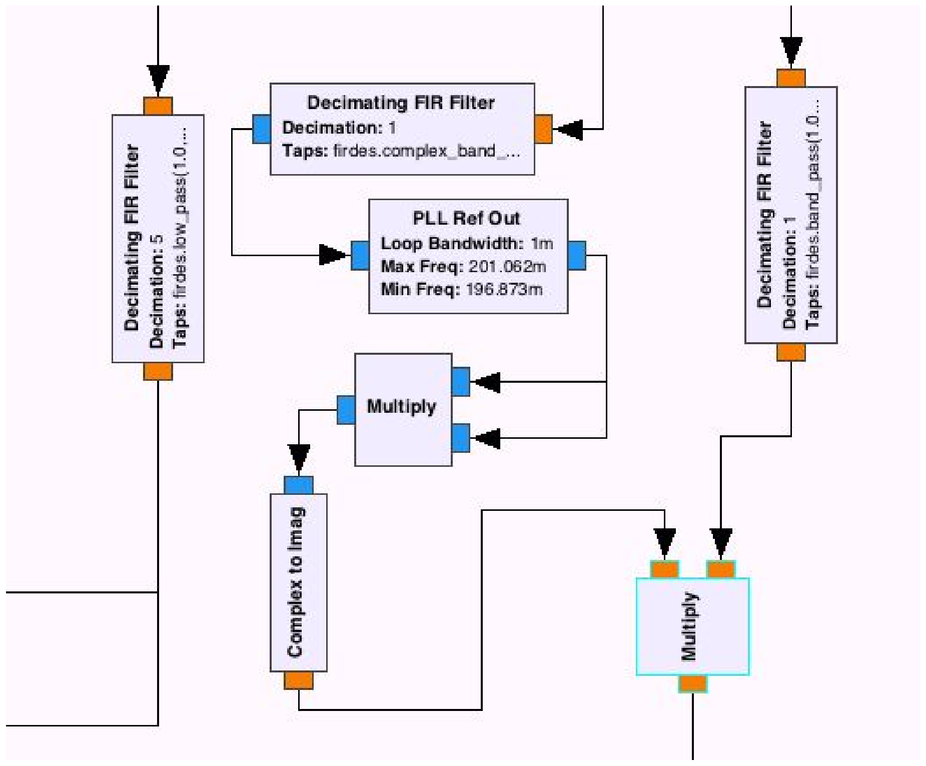
RTL-SDR Source is a source block that uses a hardware such as USB receiver as an input device for receiving signals. The input parameters are sample rate, frequency and gain.

Frequency Xlating FIR Filter is a frequency-translating FIR filter whose impulse response is of finite duration, according to Wiki.[11] It performs frequency translation, channel selection and decimation in one step.[12]

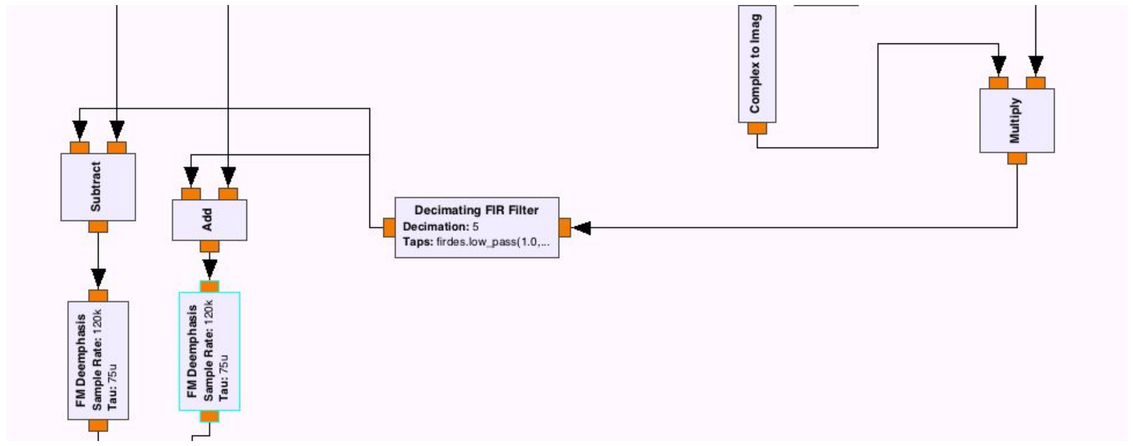
WBFM Receive is used to demodulate(extract the original information-bearing signal from a modulated carrier wave) the wide band frequency modulated signal.[13]

Filtering

Pic 4.1: Filters



Pic 4.2: Filters



The diagram contains three types of filters: low pass filter, bandpass filter and complex bandpass filter.

A **low-pass filter** is a “filter that passes signals with a frequency lower than a certain cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency”, according to Wiki.[15]

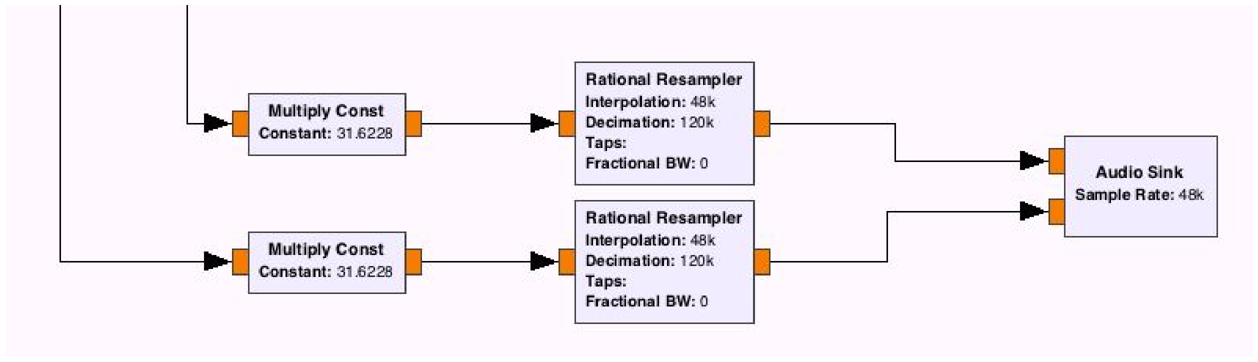
A **band-pass filter** is a device that passes frequencies within a certain range and rejects (attenuates) frequencies outside that range.[16]

PLL Ref Out(phase-locked loop) is a control system that generates an output signal whose phase is related to the phase of an input signal.[17]

De-emphasis is a system process designed to decrease the magnitude of some (usually higher) frequencies with respect to the magnitude of other (usually lower) frequencies in order to improve the overall signal-to-noise ratio by minimizing the adverse effects of such phenomena as attenuation differences or saturation of recording media in subsequent parts of the system. [18]

Output

Pic 5: Output



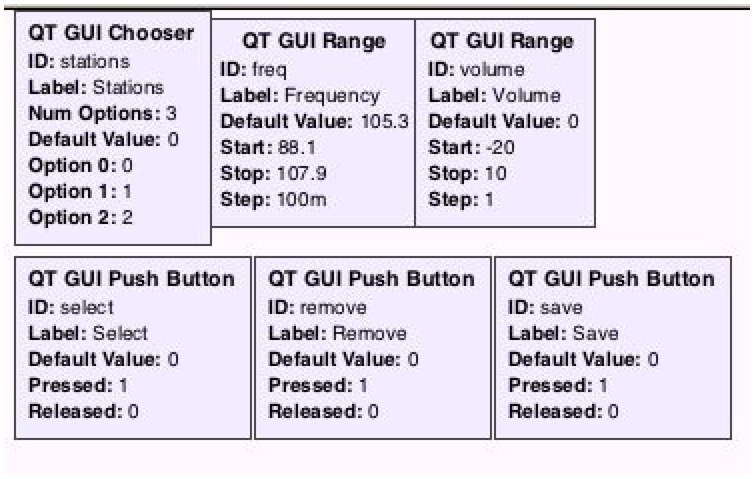
Multiply Const is a value by which a volume of an output sound depends.

Rational Resampler is a block that converts the audio output we get from filters to the sampling rate of 48 kHz.

Audio Sink is a block that uses a hardware such as sound card to output the sound.

Gui Parts

Pic 6: GUI



QT GUI Chooser
ID: stations
Label: Stations
Num Options: 3
Default Value: 0
Option 0: 0
Option 1: 1
Option 2: 2

QT GUI Range
ID: freq
Label: Frequency
Default Value: 105.3
Start: 88.1
Stop: 107.9
Step: 100m

QT GUI Range
ID: volume
Label: Volume
Default Value: 0
Start: -20
Stop: 10
Step: 1

QT GUI Push Button
ID: select
Label: Select
Default Value: 0
Pressed: 1
Released: 0

QT GUI Push Button
ID: remove
Label: Remove
Default Value: 0
Pressed: 1
Released: 0

QT GUI Push Button
ID: save
Label: Save
Default Value: 0
Pressed: 1
Released: 0

WT GUI Chooser is a combo box that contains a list of saved stations.

QT GUI Range is a widget that contains both range with a tick and text box. It is used for frequency and volume.

QT GUI Push Button is a button. It is used for select, remove and save buttons.

3.2 Application

Once we have the source code, we can modify it to add our own features.

We create a sqlite database called "stations.db" that will keep the stations saved by users. Creating the database allows to keep the stations saved even when the user closes the application.

The database contains two tables:

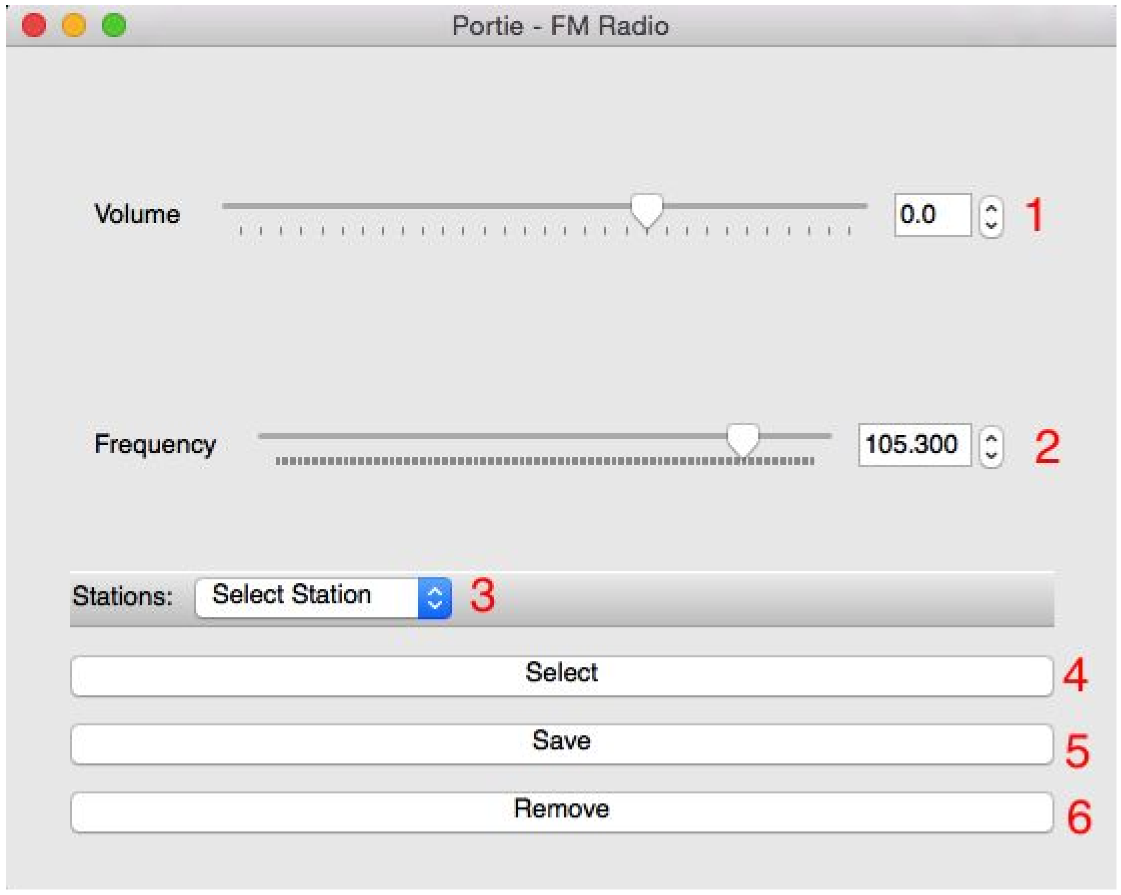
```
CREATE TABLE users(
        id INT PRIMARY KEY NOT NULL,
        username CHAR(20) UNIQUE NOT NULL,
        password CHAR(15)
);

CREATE TABLE stations (
        user_id int NOT NULL,
        station CHAR(10) NOT NULL,
        PRIMARY KEY(user_id, station),
        FOREIGN KEY(user_id) references users(id)
);
```

Where users table is the table that keeps the usernames and passwords of all of the users of the applications and stations tables keeps a list of stations saved by every user.
Then we connect to the database in the source code.

Second step is to add functionality to the buttons. Three functions are added to the source code: the function that saves the current station to the database, the function that removes currently selected station from the database and the function that tunes to the selected station from a list of saved stations.

Interface



1. Volume: The user can either drag the tick to the left to increase the volume and to the right to decrease it, or he/she can enter the volume value he/she wants.

2. Frequency: As well as with the volume, the user can change the station by moving the tick to left or right, or by entering the value of the station. The range given is from 88.1 to 107.9.

3. Stations: A list of stations the user saved and can select to tune to.

4. Select button: Tunes to the saved station the user selected. If the station is not selected(the list is at “Select station” value), the program does nothing.

5. Save button: Saves the station that is currently selected in a Frequency widget. If the station is already in a database it will not be saved.

6. Remove button: Remove the stations that is selected in a list of saved stations. If the station is not selected(the list is at “Select station” value), the program does nothing.

4. Evaluation

Two criterias are used for the evaluation of the project: usability and performance.

Usability is determined by the easiness for a person to use the application, whether it is a computer science student or not.

Performance is determined by Portie app performance comparing to Simple FM Receiver, an application I have developed using GRC, and Radio Receiver app for Chrome browser[14]. Radio Receiver is an application that uses USB receiver(the same kind I use for a project), allows the user to change the frequency, scan and save stations and and record.
Performance of the application is also evaluated according to the quality of sound in different environment.

Ten(10) people, including seven(7) computer science students, have been asked to evaluate the usability and performance of the Portie application.

Usability

Every person was instructed how to use the application. Then they've spend 5 to 10 minutes using the application.

Design

Every person find a design simple and clear. However, 2 people out of 10 say that the design is too simple and some features could have been added.

Easiness

10 out of 10 users find the applications easy to use with instructions given. Three(3) out of ten(10) people also say that the application is simple enough to use without the instructions given.

Interface

10 out of 10 people find the interface clear and easy to use, but 3 people expressed the opinion that "Select" button could be removed and instead, if the user select the station from a list of saved one, the app will automatically tune to it.

Performance

Performance evaluation is divided into two parts: with users - when polled users compare Portie app to two other apps - and without users - when the execution speed, performance in different environment and performance overall is tested.

Testing without users: sound quality

Three applications have been tested in my house, in a library building and Herzberg building of Carleton University, and outside.

In the house, the sound quality from all of the applications is relatively the same. The Simple Receiver has some noise on the background, but Radio Receiver and Portie have the same quality of sound.

In a library building, the sound quality of Portie and Simple receiver is bad. However, keeping the antenna under certain angle helps Portie to catch at least some stations, even though it still has a lot of noise. In case with Simple receiver, the only thing I am able to hear is noise no matter what I do. Radio Receiver works with some noise too, but still better that two other apps.

In a Herzberg building and outside the sound quality is the same as inside the house.

Also the sound quality varies from a platform. Simple receiver and Portie both have way worse sound quality running in Linux on Virtual Box. However, on Mac OS and Raspberry Pi’s Raspbian everything works fine.

Testing with users: overall performance check

5 out of 10 people have been polled in a library building of Carleton university where the signal is very bad so the sound performance could hardly been checked. They have only been able to check the overall performance of the app.

For all three apps the execution time is relatively the same. Portie and Simple FM receiver execute at the same time because the code for both of them is roughly the same. Radio Receiver executes in the same time if Chrome browser is not open(it needs to open Chrome first), but faster if Chrome is already open.

All of 5 people find the performance satisfying and don’t notice any delays.

Testing with users: sound quality

The users(5 out of 10 asked) were given a choice of testing the app on both Raspberry Pi and Macbook. They have compared the performance of Portie app to the performance of Radio Receiver and Simple FM Receiver.

According to the testers, the sound quality of Portie on both Raspberry Pi and Mac OS is the same.

The users also haven’t noticed the difference in sound between Portie, Radio Receiver and Simple Receiver. The sound quality was marked as high and the sound is clear, with some exceptions when the sound was noisy because the antenna placement.

The other 5 people, as it was mentioned before, weren’t able to check the sound quality however they noticed a difference between Portie and simpler FM receiver saying that the first one at least produces some sound comparing to the second one.

5. Conclusion

5.1 Summary

The result of the project shows that the idea of creating an affordable simple radio is possible and could be done in a short period of time.

Using Raspberry Pi 3, USB receiver and GNU Radio we were able to make a radio application with a quality of sound compatible to a regular radio in a period of less than one semester.

5.2 Relevance

SDR is one of the topics that has been discussed during the course.

5.3 Future work

I will continue to work with SDR in a future. My honours project is planned to be an SDR-related topic.

I will also continue to develop the application by adding the features like frequency scanner.

Appendix

Installation Guide

The first step is to write an OS image to the SD card for Raspberry Pi. The process is described on raspberry pi website [20].

Once it's done, we turn on Raspberry Pi and open terminal since all of the software can be installed through it. The process of installation will be described below step by step.

1. Installing GNU Radio

To install GNU Radio runtime and development files:

```
$ sudo apt-get install gnuradio gnuradio-dev
```

2. Setting up RTL SDR

Since we use a TV dongle, we need to stop Linux Kernel from claiming it as a TV reception. This could be done by adding a line:

```
blacklist dvb_usb_rtl28xxu
```

To a blacklist file **/etc/modprobe.d/raspi-blacklist.conf**.

Then, in order to access a device as a non-root user, we need to set up a new udev rule. To get a USB ID we run a command:

```
$ lsusb
```

This gives us a list of USB devices where the Receiver is listed as:

```
Bus 001 Device 013: ID 0bda:2838 Realtek Semiconductor Corp.
RTL2838 DVB-T
```

This informations is needed for a new rule.

Then we create a new file **/etc/udev/rules.d/20.rtlsdr.rules** with the line:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="0bda",
ATTRS{idProduct}=="2838", GROUP="adm", MODE="0666",
SYMLINK+="rtl_sdr"
```

Where 0bda and 2838 are taken from USB ID.

After this actions the restart is required.

3. Rtl_sdr installation[6]

First, we need to install libusb-1.0. This can be done using the command:

```
sudo apt-get install libusb-1.0-0-dev
```

Now we can install rtl_sdr. First we pull the source code:

```
git clone git://git.osmocom.org/rtl-sdr.git
```

Then we build it with cmake:

```
   cd rtl-sdr/
   mkdir build
   cd build
   cmake ../
   make
   sudo make install
   sudo ldconfig
```

All of the required packages are installed.

Another way is to install rtl_sdr through apt-get:

```
sudo apt-get install rtl-sdr gr-osmosdr
```

The TV tuner can be tested using the command:

```
rtl_test -t
```

References

[1] - Wireless Innovations, documents, “What is Software defined radio?”:
http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf

[2] - GNU Radio official website, Wiki, “What is GNU Radio and why do I want it?:
http://gnuradio.org/redmine/projects/gnuradio/wiki/WhatIsGR

[3] - GNU Radio Official website, “GNU Radio Companion”:
http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion#GNU-Radio-Companion

[4] - Wiki Page, “Raspberry Pi”:
https://en.wikipedia.org/wiki/Raspberry_Pi

[5] - rtlsdr.org Wiki, “History and Discovery of RTLSDR”:
http://rtlsdr.org/#history_and_discovery_of_rtlsdr

[6] - Osmocom official website, rtlsdr wiki, “rtl_sdr”:
http://osmocom.org/projects/sdr/wiki/Rtl-sdr

[7] - libusb Official Website, “libusb Home Page”:
http://www.libusb.org

[8] - SQLite Official Website, Home Page:
https://www.sqlite.org

[9] - Audacity Wiki Page, “Sample Rate”:
http://wiki.audacityteam.org/wiki/Sample_Rates

[10] - Wiki Page, “Sampling rate”:
https://en.wikipedia.org/wiki/Sampling_(signal_processing)#Sampling_rate

[11] - Wiki Page, “Finite Impulse Response”:
https://en.wikipedia.org/wiki/Finite_impulse_response

[12] - GNU Radio Companion filters, “Frequency Xlating FIR Filter”:
http://www.ece.uvic.ca/~elec350/grc_doc/ar01s12s08.html

[13] - BladeRF, “Wide Band Frequency Modulation”:
https://sites.google.com/site/sdrbladerf/home/some-sample-flowgraphs/wide-band-frequency-modulation

[14] - Radio Receiver app page:
https://chrome.google.com/webstore/detail/radio-receiver/miieomcelenidlleokajkghmifldohpo?hl=en

[15] - Wiki, “Low pass filter”:
https://en.wikipedia.org/wiki/Low-pass_filter

[16] - Wiki, Band-Pass filter:
https://en.wikipedia.org/wiki/Band-pass_filter

[17] - Wiki, Phase-locked loop(PLL):
https://en.wikipedia.org/wiki/Phase-locked_loop

[18] - Wiki, Emphasis:
https://en.wikipedia.org/wiki/Emphasis_(telecommunications)#De-emphasis

[19] - Raspberry Pi official website, “Installing operating system images”:
https://www.raspberrypi.org/documentation/installation/installing-images/

[20] - Design Spark, “Taking the Raspberry Pi 2 for a Test Drive with GNU Radio”
http://www.rs-online.com/designspark/electronics/eng/blog/taking-the-raspberry-pi-2-for-a-test-drive-with-gnu-radio-2

[21] - Base diagram for the source code, “FM RDS Reception with GNURadio and RTL SDR”:
https://www.youtube.com/watch?v=05i9C5lhorY