

# OHI/O Makeathon Spring 2016

## RTL-SDR radio with hardware controls

Team members: Aaron Maharry, Aaron Pycraft, Erica Boyer

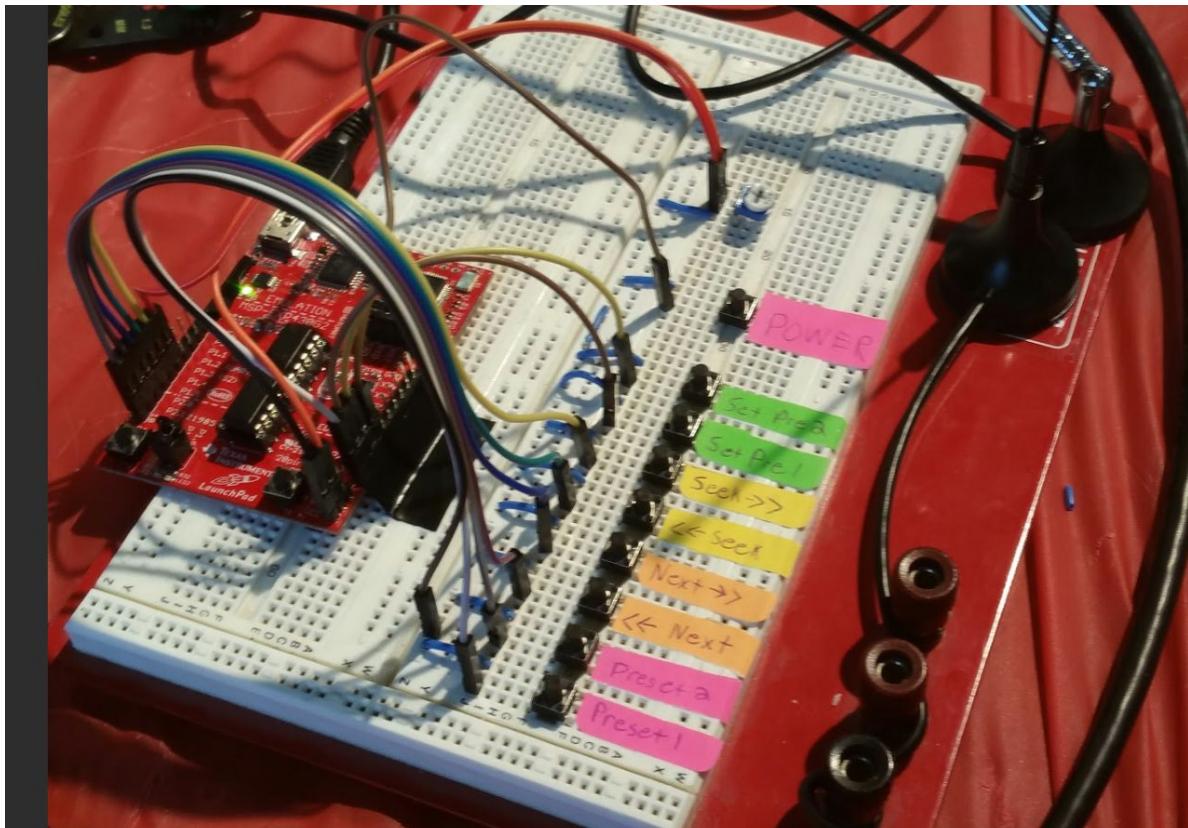


Figure 1 - Closeup view of radio user interface. Hardware controls and labels. Controls from top to bottom: power, set preset 1, set preset 2, seek forward, seek reverse, next higher freq., next lower freq., preset 2, preset 1.

## Project Description

A software defined radio (SDR) with a set of buttons and dials to tune into FM radio stations, set and tune into “favorite” radio station frequencies, and adjust radio volume. The entire project was designed and implemented within 24 hours at the Ohio State University Spring 2016 Makeathon.

## Summary

Project uses MSP 430 launchpad and a USB serial connection to PC to receive user input from hardware controls. The PC runs a python program which inputs signals from MSP430 launchpad. The program uses this data to control the SDR. The SDR controls are implemented via python code, and several scripts are generated from a GNU Radio control diagram.

## Progress and Challenges

The group originally tried to host the gnuradio and python control software on raspberry pi B+. There was difficulty getting the Pi OS to compile and run the python code. There were some issues using Raspbian/Debian, and the group did not have the expertise or the time to resolve these issues. A PC was used in replacement of the pi.

## Conclusions

The group spent a lot of time testing and discovering how to use GNU Radio. The makeathon was the first time any of the group members had used the software or done something similar.

## Abstract

The purpose of this project is to build a software defined radio (SDR) using physical analog inputs for the Ohio State University Makeathon 2016. The goal was to build a functional hardware device (versus software like a hackathon) in 24 hours. This SDR will have basic features of a functioning radio, to include seeking, presets, and volume controls. The project will only have frequency modulation (FM) reception in scope, but can pick up all ranges, however for the sake of this Makeathon, the signal range has been damped to only receive signals between 88.0 MHz and 108.0 MHz (the “music” frequencies).

Table 1 - Team members and responsibility

Member	Responsibilities
Aaron Pycraft	Wiring and provided some materials
Aaron Maharry	Coding and provided some materials
Erica Boyer	Documentation and provided some materials

## RTL-SDR BASED RADIO WITH ENHANCEMENTS

Table 2 - List of Materials

Item	Quantity	Description of use
TI MSP430 Launchpad	1	Main microcontroller used for input and output
Breadboard	1	Assembly of circuits
NooElec R820T SDR & DVB-T	1	RTL-SDR receiver dongle for RF signals
NooElec Antenna	1	Signal antenna
Push Buttons	9	Toggle pushbuttons for various controls
200 Ohm Potentiometer	1	Volume control knob
Various Solid-stranded wires	~2 ft	For hardware control circuit
USB Blaster Cable for TI MSP430 Launchpad	1	For serial communication between PC and Launchpad

Table 3 - List of Software used

SOFTWARE	DESCRIPTION
GNU Radio	Graphical programming interface, to generate python code used to control SDR (change frequency, apply filters, etc)
Energia	IDE for MSP Launchpad programming
Python	Programming Language, To program the GNU Radio
C++	Programming Language, To program the Launchpad



Figure 2 - Overview of system. The breadboard contains the radio user's interface, and the MSP launchpad which interprets the user's input and transmits the data to the PC. The PC runs the python code which controls the SDR dongle (blue USB stick), which receives the RF signals (via the antenna setup on the breadboard).

## Instructions

1. Gather all materials as listed above on a clean, clear workspace.
2. Install software (gnuradio, python) on personal computer.
3. Setup grid of switches, buttons, and dials for user controls.
4. Map each hardware component to the GPIO pins on the MSP launchpad, the dial must be an analog input, not GPIO.
5. Use Energia to write the C code used to receive inputs from the user controls.
6. Setup serial connection between PC and MSP launchpad. Verify user input can be displayed using the serial monitor.
7. Use GNU radio, use the built in RTL-SDR libraries to display FFT of FM signal received, and add an audio output sink.
8. Use GNU radio to implement basic bandpass and lowpass filter, and any other desired filters.
9. Use any text editor to design a python program to input the data from the serial connection to MSP launchpad. Map each button input event to update some SDR variable, such as volume or frequency.
10. Execute the python code and use the buttons to control the radio.

## Results and Figures

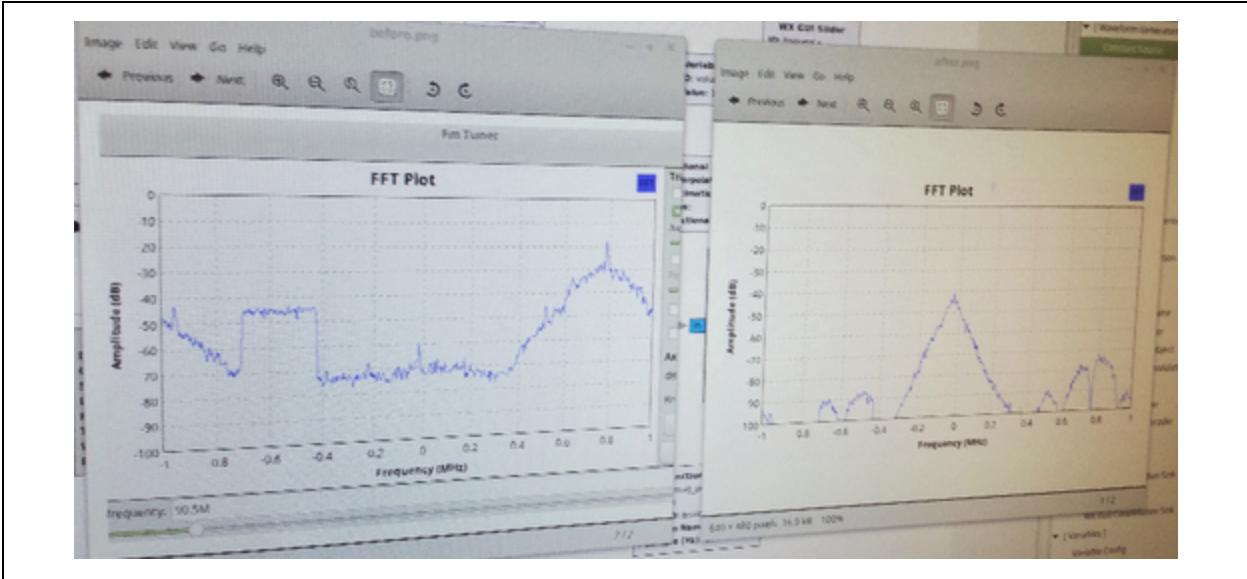


Figure 3 - Fast Fourier Transform plots of FM signal, before and after filtering noise.

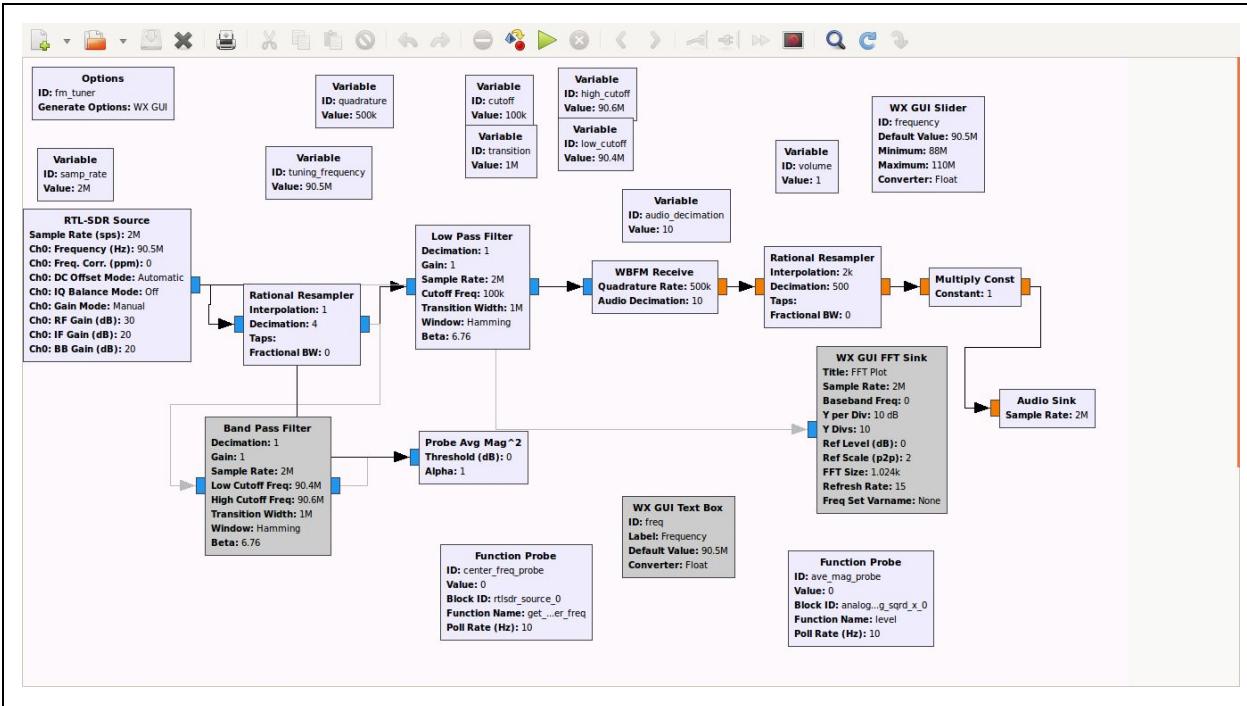


Figure 4 - GNU Radio Flow diagram.

Figure 5 - Python code generated via GNU Radio

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: Fm Tuner
# Generated: Sun Mar  6 09:18:23 2016
#####
from gnuradio import analog
from gnuradio import audio
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import osmosdr
import threading
import time
import wx

class fm_tuner(grc_wxgui.top_block_gui):
    def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Fm Tuner")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.tuning_frequency = tuning_frequency = 90.5e6
self.volume = volume = 1
self.transition = transition = 1000000
self.samp_rate = samp_rate = 2000000
self.quadrature = quadrature = 500000
self.low_cutoff = low_cutoff = tuning_frequency-0.1e6
self.high_cutoff = high_cutoff = tuning_frequency+0.1e6
self.frequency = frequency = tuning_frequency
self.cutoff = cutoff = 100000
self.center_freq_probe = center_freq_probe = 0
self.ave_mag_probe = ave_mag_probe = 0
self.audio_decimation = audio_decimation = 10

#####
# Blocks
#####
self.rtlsdr_source_0 = osmosdr.source( args="numchan=" + str(1) + " " + "" )
self.rtlsdr_source_0.set_sample_rate(samp_rate)
self.rtlsdr_source_0.set_center_freq(tuning_frequency, 0)
self.rtlsdr_source_0.set_freq_corr(0, 0)
self.rtlsdr_source_0.set_dc_offset_mode(2, 0)
self.rtlsdr_source_0.set_iq_balance_mode(0, 0)
```

```

self.rtlsdr_source_0.set_gain_mode(False, 0)
self.rtlsdr_source_0.set_gain(30, 0)
self.rtlsdr_source_0.set_if_gain(20, 0)
self.rtlsdr_source_0.set_bb_gain(20, 0)
self.rtlsdr_source_0.set_antenna("", 0)
self.rtlsdr_source_0.set_bandwidth(0, 0)

self.analog_probe_avg_mag_sqrd_x_0 = analog.probe_avg_mag_sqrd_c(0, 1)
self.rational_resampler_xxx_1 = filter.rational_resampler_fff(
    interpolation=2000,
    decimation=500,
    taps=None,
    fractional_bw=None,
)
self.rational_resampler_xxx_0 = filter.rational_resampler_ccc(
    interpolation=1,
    decimation=4,
    taps=None,
    fractional_bw=None,
)
self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
    1, samp_rate, cutoff, transition, firdes.WIN_HAMMING, 6.76))
_frequency_sizer = wx.BoxSizer(wx.VERTICAL)
self._frequency_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_frequency_sizer,
    value=self.frequency,
    callback=self.set_frequency,
    label='frequency',
    converter=forms.float_converter(),
    proportion=0,
)
self._frequency_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_frequency_sizer,
    value=self.frequency,
    callback=self.set_frequency,
    minimum=88e6,
    maximum=110e6,
    num_steps=100,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_frequency_sizer)
def _center_freq_probe_probe():
    while True:
        val = self.rtlsdr_source_0.get_center_freq()
        try: self.set_center_freq_probe(val)
        except AttributeError, e: pass
        time.sleep(1.0/(10))
_center_freq_probe_thread = threading.Thread(target=_center_freq_probe_probe)
_center_freq_probe_thread.daemon = True
_center_freq_probe_thread.start()
self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vff((volume, ))

```

```

def _ave_mag_probe():
    while True:
        val = self.analog_probe_avg_mag_sqrd_x_0.level()
        try: self.set_ave_mag_probe(val)
        except AttributeError, e: pass
        time.sleep(1.0/(10))
_ave_mag_probe_thread = threading.Thread(target=_ave_mag_probe)
_ave_mag_probe_thread.daemon = True
_ave_mag_probe_thread.start()
self.audio_sink_0 = audio.sink(samp_rate, "", True)
self.analog_wfm_rcv_0 = analog.wfm_rcv(
    quad_rate=quadrature,
    audio_decimation=audio_decimation,
)

#####
# Connections
#####
self.connect((self.analog_wfm_rcv_0, 0), (self.rational_resampler_xxx_1, 0))
self.connect((self.blocks_multiply_const_vxx_0, 0), (self.audio_sink_0, 0))
self.connect((self.low_pass_filter_0, 0), (self.analog_wfm_rcv_0, 0))
self.connect((self.rational_resampler_xxx_1, 0), (self.blocks_multiply_const_vxx_0, 0))
self.connect((self.rational_resampler_xxx_0, 0), (self.low_pass_filter_0, 0))
self.connect((self.rtlsdr_source_0, 0), (self.rational_resampler_xxx_0, 0))
self.connect((self.rtlsdr_source_0, 0), (self.analog_probe_avg_mag_sqrd_x_0, 0))

# QT sink close method reimplementation

def get_tuning_frequency(self):
    return self.tuning_frequency

def set_tuning_frequency(self, tuning_frequency):
    self.tuning_frequency = tuning_frequency
    self.set_high_cutoff(self.tuning_frequency+0.1e6)
    self.set_low_cutoff(self.tuning_frequency-0.1e6)
    self.set_frequency(self.tuning_frequency)
    self.rtlsdr_source_0.set_center_freq(self.tuning_frequency, 0)
    self.rtlsdr_source_0.set_center_freq(self.tuning_frequency, 1)

def get_volume(self):
    return self.volume

def set_volume(self, volume):
    self.volume = volume
    self.blocks_multiply_const_vxx_0.set_k((self.volume,))

def get_transition(self):
    return self.transition

def set_transition(self, transition):
    self.transition = transition
    self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, self.cutoff,
self.transition, firdes.WIN_HAMMING, 6.76))

def get_samp_rate(self):

```

```

    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, self.cutoff,
self.transition, firdes.WIN_HAMMING, 6.76))
    self.rtlsdr_source_0.set_sample_rate(self.samp_rate)

def get_quadrature(self):
    return self.quadrature

def set_quadrature(self, quadrature):
    self.quadrature = quadrature

def get_low_cutoff(self):
    return self.low_cutoff

def set_low_cutoff(self, low_cutoff):
    self.low_cutoff = low_cutoff

def get_high_cutoff(self):
    return self.high_cutoff

def set_high_cutoff(self, high_cutoff):
    self.high_cutoff = high_cutoff

def get_frequency(self):
    return self.frequency

def set_frequency(self, frequency):
    self.frequency = frequency
    self._frequency_slider.set_value(self.frequency)
    self._frequency_text_box.set_value(self.frequency)

def get_cutoff(self):
    return self.cutoff

def set_cutoff(self, cutoff):
    self.cutoff = cutoff
    self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, self.cutoff,
self.transition, firdes.WIN_HAMMING, 6.76))

def get_center_freq_probe(self):
    return self.center_freq_probe

def set_center_freq_probe(self, center_freq_probe):
    self.center_freq_probe = center_freq_probe

def get_ave_mag_probe(self):
    return self.ave_mag_probe
def set_ave_mag_probe(self, ave_mag_probe):
    self.ave_mag_probe = ave_mag_probe
def get_audio_decimation(self):
    return self.audio_decimation
def set_audio_decimation(self, audio_decimation):

```

```
    self.audio_decimation = audio_decimation

if __name__ == '__main__':
    import ctypes
    import os
    if os.name == 'posix':
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"
    parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
    (options, args) = parser.parse_args()
    tb = fm_tuner()
    tb.Start(True)
    tb.Wait()
```

Figure 5 - Python code generated via GNU Radio (continued)

Figure 6 - Python code to receive user input and control SDR

```
import serial
import time
from fm_tuner import fm_tuner
from threading import Timer
from threading import Thread
import math

preset1 = 90.5E6
preset2 = 92.3E6

s = None
tb = fm_tuner()
tb.Start()
enabled = False
print "*****"
print enabled
curr_freq = preset1
ave_mag = 0.0
curr_volume = 1
def change_freq(new_freq):
    if enabled:
        tb.rtlsdr_source_0.set_center_freq(new_freq)
        curr_freq = new_freq

"""Used to analyze strength of signals across Entire FM radio band.
fp = open("station-magnitudes.csv", "w")
for _freq in xrange(879, 1079, 2):
    freq = _freq * 10**5
    change_freq(freq)
    time.sleep(4.0)
    ave_mag = 0
    for i in range(20):
        ave_mag += tb.get_ave_mag_probe()
        time.sleep(0.1)
    ave_mag = ave_mag / 20
    print "Center Freq: %f\t\tAve Mag: %f" %(freq, ave_mag)
    fp.write("%f,%f\n" %(freq, ave_mag))
fp.close()
exit()"""

def processInputs(values):
    global tb
    global enabled
    global preset1
    global preset2
    global curr_volume
    if len(values) != 10:
        return
    if values[1] == "1":
        change_freq(preset1)
    if values[2] == "1":
        change_freq(preset2)
    if values[3] == "1":
```

```

        change_freq(max(curr_freq - 0.2e6, 87.9e6))
if values[4] == "1":
    change_freq(min(curr_freq + 0.2e6, 107.9e6))
if values[5] == "1":
    change_freq(max(curr_freq - 0.2e6, 87.9e6))
if values[6] == "1":
    change_freq(min(curr_freq + 0.2e6, 107.9e6))
if values[7] == "1":
    preset1 = curr_freq
if values[8] == "1":
    preset2 = curr_freq
if values[9] == "1":
    if enabled:
        enabled = False
    else:
        enabled = True
#print(enabled)

new_volume = float(int(values[0])) * 2 / 1024
#print("vol: %f, %f" %(curr_volume, new_volume))
if not enabled and curr_volume != 0:
    tb.set_volume(0)
    curr_volume = 0
elif enabled and new_volume != curr_volume:
    tb.set_volume(new_volume)
    curr_volume = new_volume

try:
    while True:
        if enabled:
            curr_freq = tb.get_center_freq_probe()
            ave_mag = (9 * ave_mag + tb.get_ave_mag_probe())/10
            print "Center Freq: %f\tAve Mag: %f" %(curr_freq, ave_mag)
        if not s:
            try:
                s = serial.Serial("/dev/ttyACM0", 9600)
            except serial.SerialException:
                s = None
            time.sleep(0.05)
        else:
            try:
                if s.inWaiting() > 1:
                    value = s.readline().strip()
                    #print value
                    processInputs(value.split(","))
            except serial.SerialException:
                s = None
            time.sleep(0.05)
except KeyboardInterrupt:
    s.close()

```

Figure 6 - Python code to receive user input and control SDR (continued)