SMART CLOSED-LOOP JAMMING SYSTEM

Peter Joseph

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Congdon School of Supply Chain, Business Analytics, and Information Systems

University of North Carolina Wilmington

2023

Approved by

Advisory Committee

Dr. Ronald Vetter _____

Dr. Jeffery Cummings _____

Dr. Hosam Alamleh, Chair
_____

Accepted By

_____
Dean, Graduate School

TABLE OF CONTENTS

ABSTRACT

Smart Closed-Loop Jamming System. Joseph, Peter, 2023. Capstone Paper, University of North Carolina Wilmington.

The reliance on wireless technology and mobiles devices within the United States has created several vulnerabilities in the realm of cyber-security. Wireless devices are inherently vulnerable to interception attacks since they are required to connect to another device due to their functionality, but there is no real method to combat these attacks besides encryption. The Smart Closed-Loop Jamming System provides a possible solution by detecting the attack in its initial stage, and subsequently jamming the point of origin. The prototype of this system is tested through utilizing a series of devices and networks including home Wi-Fi, Bluetooth, Verizon cell phones, and HAM radios.

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Wireless Technology

There are over 400 million wireless networked devices in use with the United States today, such as: cell phones, Bluetooth headsets, unmanned aerial drones, wireless mouses and keyboards, radios, garage door openers, and many more. Wireless technology has evolved to accommodate the convenience of everyday life, but it has also increased the surface area for cyber threat actors. In the Mobile Security Report of 2021 published by Check Point Software Technologies, it is estimated that over 40% of these wireless devices are prone to cyber-attacks due to lackadaisical security measures and lack of user awareness (Check Point Software, 2021). Cyber threat actors have found success against wireless devices through several different methods, including Man-in-the-Middle (MITM) Attacks, Wi-Fi jamming, Spoofing attacks, Distributed-Denial-of-Service (DDOS), packet sniffing, rogue access points, interception attacks, and several others (Check Point Software, 2021). The COVID-19 pandemic in 2020 caused a significant increase in attacks against wireless devices due to the increase in remote work. Since 2020, 93% of businesses in the United States have suffered from at least one cyber-attack that originated from a mobile device (Check Point Software, 2021). While some of these attacks did originate from company owned devices, most originated from personal devices that served as an entry point to company networks. Because personal devices are inherently vulnerable due to their lack of security measures, threat actors specifically began to develop malware targeting mobile devices (Check Point Software, 2021). In 2022, the average cost of cyber-attacks against wireless devices in the United States was $4.35 million. In contrast, interception attacks such as MITM attacks cost over $5 million, and had more devastating effects due to the confidential information that was

compromised (James, 2022). Interception attacks are difficult to detect if they are properly executed, and they are effective whether they are employed against data at rest or data in motion.

Interception Attacks

Interception attacks allow unauthorized users to access confidential data, applications, or environments (McClanahan, 2022). Since wireless devices inherently require communication with a distant end to work, they are prone to the various methods of interception attacks, and attackers are beginning to take note. The Verizon Mobile Security Index of 2019 states, “Forty-eight percent of the sophisticated cyber actors identified by Lookout Mobile Security in the past year were found to have the tools and techniques for attacking both mobile and desktop devices” (Verizon Mobile, 2020). This was an 18% increase from 2018, and with the current trend the cost of such attacks is predicted to reach an annual cost of $1.25 billion by 2031 (Verizon Mobile, 2020). These interception attacks can lead to attackers eavesdropping or hijacking devices, copying files or programs for future use, storing messages for a replay attack later on, key logging, and even wiretapping the network to monitor further communications (McClanahan, 2022). Currently, the only defense against interception attacks is to encrypt the traffic, or to employ traffic padding, which is to add unnecessary ciphertext to the message to deceive any unwanted viewers (McClanahan, 2022). These are proactive measures to guard against interception attacks, but encryption can be broken and leave the user vulnerable to attack. What other methods do we have to guard against interception attacks? The Smart Closed-Loop Jamming System is an additional protection measure that would not only defend wireless devices, but also provide a limited response to the attacker.

Smart Closed-Loop Jamming System

The Smart-Closed Loop Jamming System will be a protective measure to any kind of interception attack. The Smart-Closed Loop Jamming System is built to detect a transmission spike from an unauthorized device and then transmit a jamming signal to the attacking device. The prototype of this system has two main features: the transmission spike detection feature, and the jamming signal emission feature. The transmission spike detection feature is based on the timing and decibel of the transmission. If there is a sustained transmission of longer than a few seconds at a high decibel rate on the authorized devices uplink frequency, the system interprets that as a possible interception. At this point, the transmission jamming feature begins. The system emits a high-decibel transmission to the downlink frequency of the attacker, effectively cutting off their ability to transmit. This allows the user to continue operating their wireless device without interruption.

CHAPTER 2: REVIEW OF LITERATURE REVIEW AND ANALYSIS

Electromagnetic Interference

Electromagnetic interference, or Radio Frequency Interference (RFI) occurs when unwanted radio frequency signals disrupt the use of a mobile device. It is common due to the number of devices that operate across the radio spectrum. There are two main categories of radio interference, broadband and narrowband. Broadband interference commonly occurs at the 2.4 GHz and 5Ghz frequency levels, which are the frequencies approved for Wi-Fi internet connections. It occurs when the unshielded copper wires of broadband devices pick up radio waves and convert them to an electrical signal, thus generating extra unwanted transmissions (Promptlink Communications, 2019). When observed with spectrum analyzer or other frequency monitoring technology, they appear to be increase in the decibel over a wide range (Wyatt, 2019). Broadband interference is typically caused by sources such as power line noise, or an interference with the device power supply (Wyatt, 2019). Narrowband Interference (NBI) can occur when a frequency has a bandwidth between 30 – 3400Hz (Wyatt, 2019). Most Ultra-High Frequencies (UHF) typically operate within this bandwidth range. NBI occurs when a relatively high interference power level is concentrated at a specific frequency (ISCO International, 2021). When observed by a spectrum analyzer, NBI is represented by narrow vertical lines or a slightly wider modulated vertical band that is associated with a specific frequency (Wyatt, 2019). NBI is has been known to be caused by radar installations and television broadcasts (ISCO International, 2021). In addition to the two main forms of interference, there is also Wideband Interference (WBI), which is characterized by lower power levels that is spread out over a frequency range (ISCO International, 2021). It can occur when a frequency has a bandwidth between 50 – 7000hz. Most VHF frequencies

are subject to WBI due to their wavelength thickness. When observed by a spectrum analyzer, WBI is represented by lower decibel levels spread over a wide frequency range (ISCO International, 2021). WBI can be caused by co-channel interference, such as when a foreign mobile network is operating on the same channel near the border. It can also be caused by mobile networks operating on a relatively adjacent channel (ISCO International, 2021).

Radio Transmission & Jamming Legal Repercussions

Radio broadcasts without a license issued by the Federal Communications Commission (FCC) is prohibited by The Communications Act of 1934, section 301 (Federal Communications Commission, 2021). Anyone found operating any radio station without authorization can face civil and criminal penalties, as well as have their equipment seized (Federal Communications Commission, 2021). The FCC enforces these laws by employing spectrum scanning to detect unauthorized radio operation and encouraging citizens to report any suspicion of unauthorized radio operations (Federal Communications Commission, 2021). Moving on, the FCC considers jamming devices to be a threat to the safety of the American people because it can interfere with emergency calls and interfere with public safety communications (Federal Communications Commission, 2020). Section 302b of the Communications Act also severely prohibits the operation, marketing, or sale of any kind of jamming device. The use of a phone jammer, GPS blocker, or any other signal jamming device designed to intentionally interfere with authorized radio communications violates federal law and will carry a significant criminal penalty (Federal Communications Commission, 2020). As such, the FCC advises consideration of outside factors if experiencing a bad connection including faulty equipment, any physical obstruction, and possible co-channel interference from devices

on the same operating frequency (Federal Communications Commission, 2020). They also advise users to troubleshoot their own equipment and connectivity before filing a report. Despite the severe restriction on jamming and unauthorized radio operation, there are limited exceptions for radio operations that can be authorized by the FCC, such as operating domestic ships, aircraft, and utilizing radios for personal use in line with the FCC consumer guide for personal radio services (Federal Communications Commission, 2019). This consumer guide lists several types of personal radio services that are available for public use, including: citizen band radios, family radio services, general mobile radio services, low-power radio services and multi-use radio service (Federal Communications Commission, 2019). These radio services allow the public to use several waveforms to address any private needs, but also provide backup emergency operations channels, specific waveforms that are reserved for disabled-persons, and allowing for basic two-way voice communication over short distances (Federal Communications Commission, 2019). The FCC limits personal devices to less than half of a mile on .5-watt channels and up to two miles on 2-watt channels (Federal Communications Commission, 2019). The General Mobile Radio Service is the only personal service that requires a license to operate so that the FCC can track the station if it relocates (Federal Communications Commission, 2019). The FCC encourages users of personal radio services to be inherently familiar with this guide, as any accidental breach of regulation can still incur legal repercussions.

Definition of Terms

This section will cover terminology that will be common to radio technology and signal analysis. Those terms will be defined here as they apply to audio engineering.

Gain. The definition of gain that will be used throughout this report comes from

Wiktionary.org. Gain is a point during an audio signal flow that the engineer can make amplification adjustments to the signal by any process to increase its strength.

Sampling Rate. The definition of sampling rate that will be used throughout this report comes from digitizationguidelines.gov. Sampling rate or sampling frequency defines the number of samples per second taken from a continuous signal to make a discrete or digital signal.

Decibel. The definition of decibel that will be used throughout this report comes from Wiktionary.org. A common measure of sound intensity ratio that is one tenth of a bel on the logarithmic intensity scale.

Wavelength. The definition of wavelength that will be used throughout this report comes from Wiktionary.org. The length of a single cycle of a wave, as measured by the distance between one peak or trough of a wave and the next.

Frequency. The definition of wavelength that will be used throughout this report comes from Wiktionary.org. A range of sound waves that are audible.

CHAPTER 3: METHODOLOGY

Technology Overview

The project was completed using the BladeRF 2.0 Micro xA5 through the Linux Dragon Operating System. The BladeRF xA5 is a superspeed Software Defined Radio (SDR) that has dual input and output modes and is fully programmable through its field-programmable gate array (Nuand, 2023). It has a frequency range of 47MHz to 6Ghz, with a sampling rate of 61.44MHz (Nuand, 2023). The Linux Dragon Operating system is designed specifically for Software Defined Radio functionality. It contains built in open-source software radio programs such as: SDR++, Baudline, GNU Octave, and GNU Radio Companion (SourceForge, 2020). It also contains the entire BladeRF library package, allowing the user to implement the device from the command line interface (SourceForge, 2020). SDR++ is a bloat-free SDR software that contains a frequency manager to create lists of pre-defined frequencies and display them for live testing (Ryzerth, 2022). It has the capability to transmit and receive over both radio and network waveforms. It also contains a module manager that allows the user to manually configure their specifications such as the gain, decibel, and sample rate (Ryzerth, 2022). GNU-Octave is an interpreted programming language that was developed for numerical computations (Eaton, 2022). The GNU Octave platform has the functionality to build 2D and 3D models, which can be used to represent the flowgraphs needed for radio transmissions. The octave interpreter can be run through shell scripts, through the Graphical User Interface (GUI) function, or called through the console (Eaton, 2022). Octave was developed through C++, C, and Fortran code with the purpose of solving mathematical equations. As such, it is highly compatible with MATLAB scripts, which

can be run through the Octave interpreter (Eaton, 2022). Moving on, the GNU Radio Companion will also be highly useful for this project. The GNU Radio Companion is a Software Development Framework that provides signal processing functions for implementing software defined radios (GNU Radio, 2022). It contains a drag and drop library to dictate specific actions and processes that will simulate a highly capable real-world radio system (GNU Radio, 2022). It is a flexible and scalable program in that you can design smaller functions or create massive programs that run features. To prove the concept that the Smart Closed-Loop Jamming System can be accomplished, there are three primary test cases that will need to be completed: the home Wi-Fi test case, the Verizon cell phone test case, and the HAM Radio test case. There will be a specific set of steps for the Transmission Spike Detection feature, as well as the Jamming Signal emission feature.

Wi-Fi Test Case Procedure

The first test case completed was the home Wi-Fi test. Home Wi-Fi routers are run at a frequency of 2.4GHz, or a frequency of 5Ghz if using a 5G connection. The first feature to test will be the Transmission Spike Detection. The BladeRF xA5 will be put in receive mode at a frequency of 2.4Ghz using the BladeRF command client. With the antenna in the receive port, the BladeRF will be ran through terminal to scan for network spikes as various devices (cell phones, computers, Xbox) are connecting to the Wi-Fi and passing traffic. Additionally, a speed test of the connection was conducted to determine the effects on the network. The intent is to be able to identify a sustained spike in transmission at a high decibel rate. For each device that is connected, the results were recorded and saved to an excel spreadsheet that was used to recreate the signal flowgraph. Moving on, the jamming signal feature was then tested using home Wi-Fi.

The BladeRF device was then be programmed with a transmit frequency of 2.4Ghz. While transmitting continuously, the same devices were connected to the internet, passed some form of traffic, and then a speed test was conducted to determine if the upload and download speeds of the network have been affected. The speed test results of the PC and cell phones, as well as the ping rate of the Xbox during gameplay was observed, and any notable service interruptions were recorded.

Cell Phone Test Case Procedure

The next text case was conducted using a pair of cell phones operating on Verizon's frequency range. Figure 1 below details the uplink and downlink frequencies for every major cell carrier within the United States.

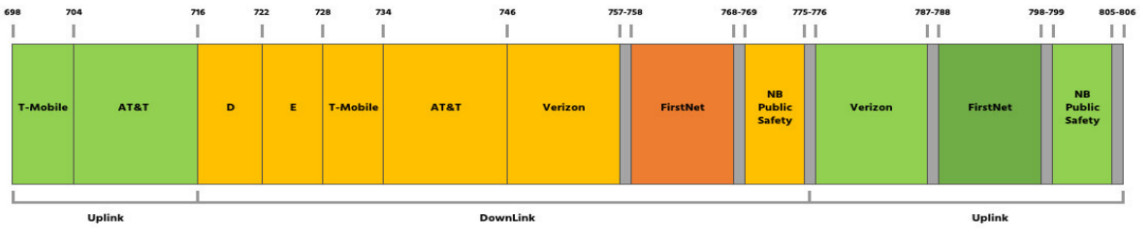


Figure 1: Cell Carrier Spectrum Band

The downlink frequency for Verizon ranges from 746MHz to 757MHz, and the uplink frequency ranges from 776MHz to 787MHz (Amplifiers, 2022). To conduct the test for the transmission spike detection, the BladeRF was programmed to monitor the uplink frequency range. A call was then made by two Verizon cell phones, and the transmission wavelength will be viewed in the Baudline spectrum viewer while the call is being placed, when the call is accepted and the connection is established, and when the call ends. The receiving cell phone waited for 10 seconds before accepting the call to observe any transmission spikes within the uplink frequency range. These results were saved to an excel spreadsheet for analysis at a later date. Next, the jamming signal emission was

tested. The BladeRF was programmed within Verizon's downlink frequency range. It then broadcasted continuously over that frequency range, and another call will be made using the two Verizon devices. Any service interruptions of the call to include static, fuzzing, and any issues accepting or declining the call were noted. This required coordination with Verizon to have the phones programmed at specific frequencies for a limited time. They were unwilling to accommodate this, so the X-Talk and SINOX applications that are available for Android phones were used to replicate the phone call.

Radio Test Case Procedure

The final test conducted was utilizing a pair of BaoFeng UV-5R HAM radios. The UV-5R radio has a programmable frequency range from 47MHz to 520 MHz, which was used to test both features (BaoFeng Radios, 2020). The transmission spike detection feature was the first to be tested. The radios will be programmed at both Very High Frequency (VHF) low, VHF high, and UHF frequencies. The BladeRF will be programmed to monitor the same frequency as the HAM radios. Using baudline or SDR++, the BladeRF detected voice traffic during the transmissions, taking note of any significant spike while radio transmissions are in progress. The test was conducted through a sustained radio check of 10 seconds, with specific attention being paid to the decibel rate during the initial radio key and throughout the sustained radio transmission. To test the jamming signal feature, the radios were again be programmed at VHF low, VHF high, and UHF frequencies. The BladeRF was programmed at those same frequencies but was now placed in transmit mode at a high decibel rate. The BladeRF then broadcasted continuously over the frequencies that the HAM radio is programmed at. Radio transmissions were attempted from the HAM radios during this broadcast, and

any service interruptions that are observed were recorded. This includes any static, fuzzing, or broken voice traffic during the transmissions.

CHAPTER 4: COMPLETED PROJECT AND TEST RESULTS

Transmission Spike Detection GRC Flowgraph Overview

To develop the transmission detection feature, a signal flowgraph was built in GNU Radio Companion to generate python code. This flowgraph uses a First-In, First Out (FIFO) file as an audio source to process the samples received by the BladeRF. This allows the samples, which are taken in 16-bit integer format, to be saved for later analysis, as well as be written to a GUI that can be visualized in real time. Once the samples are taken in from the FIFO, they are converted into complex signals so that they can be displayed in real-time through a GUI for the user to visualize the audio signals. The samples are also compared to a noise threshold that is established by the user. The created flowgraph is displayed in figure 2 below.

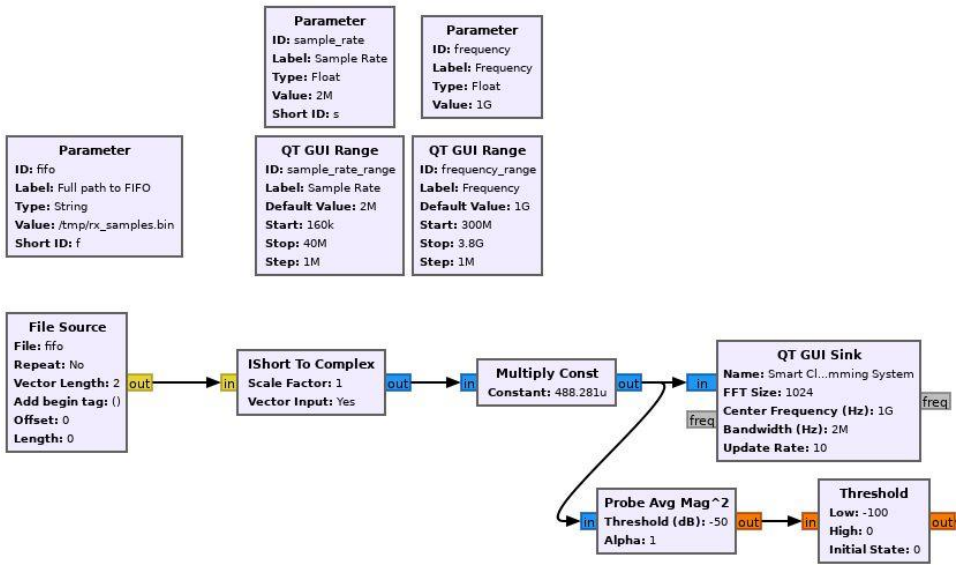


Figure 2: Transmission Spike Detection Flowgraph

The user is able to pass the sample rate and frequency in as parameters when they invoke the program. The FIFO file is also a parameter, but since the File Source block pulls the file path directly from the ID of the FIFO parameter block, the file path must be established in the FIFO parameter block and can't be passed through by the user (GNU

Radio, *File source* 2022). The file source block passes the 16-bit integer values of the audio samples to an IShort-to-Complex block. The IShort-to-Complex block converts the audio samples from their short integer format to complex signals so that they can be displayed by the GUI, as well as calculate the average magnitude of the signal so that the program can be compared to the threshold value (GNU Radio, *Ishort to complex* 2022). The IShort-to-Complex block passes the complex signal to the Multiply Const block, which adjusts the amplitude of the signal (GNU Radio, *Multiply const* 2021). The Multiply Const block is set to 488.21u, which will reduce the signal strength to .0488281% of its original strength. This value is set to significantly reduce the strength of the incoming signal so as not to harm the BladeRF device, as well as bring the signals down to reasonable value ranges that can be displayed by the GUI. The Multiply Const block passes the complex signal to the QT GUI Sink Box, and the Probe Avg Mag^2 block. The QT GUI Sink block displays the plot of the audio signals in different graphs (GNU Radio, *Qt Gui Sink* 2019). This GUI allows the user to adjust the sample rate and frequency of the program from the initial user-input values. The Probe Avg Mag^2 block computes the average magnitude squared of the incoming signal. This is used to measure and monitor the signals power level during processing, as well as set the pace for how often the averaging process is completed (GNU Radio, *Probe avg mag^2* 2019). The last block is the Threshold Block, which will compare the input to the specified threshold value. It will output a 1 if the input signal is greater than or equal to the threshold, or a 0 if the input signal is less than the threshold (GNU Radio, *Threshold* 2022).

Transmission Spike Detection Python Code Overview

GNU Radio Companion has a feature that will allow the user to generate python code based on the Flowgraph. The code generated from the flowgraph represented in

figure 2 is depicted in several figures below. Some python code was added to implement the functionality required by the Transmission Detection feature.

```
from packaging.version import Version as StrictVersion
import math

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print("Warning: failed to XInitThreads()")

from PyQt5 import Qt
from gnuradio import qtgui
from gnuradio.filter import firdes
import sip
from gnuradio import blocks
import pmt
from gnuradio import gr
from gnuradio.fft import window
import sys
import signal
from argparse import ArgumentParser
from gnuradio.eng_arg import eng_float, intx
from gnuradio import eng_notation
from gnuradio.qtgui import Range, GrRangeWidget
from PyQt5 import QtCore
from gnuradio import analog
```

Figure 3: Python Imports from GRC Flowgraph for Audio Detection Feature

Figure 3 represents the import packages that are required for the program to function. The program has several imports from the GNU Radio package such as the blocks, analog, qtgui, QtCore, and eng_Notation. The Qt and QtCore are also imported from the PyQt5 application framework in order to create the GUI. The sys, sip, pmt, and signal packages are also imported for the python environment to manipulate the signals into the GUI.

```
class audioDetect(gr.top_block, Qt.QWidget):

    def __init__(self, fifo='/tmp/rx_samples.bin', frequency=1e9, sample_rate=2000000):
        gr.top_block.__init__(self, "Smart Closed Loop Jamming System", catch_exceptions=True)
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Smart Closed Loop Jamming System")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "bladeRF_fifo_rx")

        try:
            if StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
                self.restoreGeometry(self.settings.value("geometry").toByteArray())
            else:
                self.restoreGeometry(self.settings.value("geometry"))
        except:
            pass

        ##################################################
        # Parameters
        ##################################################
        self.fifo = fifo
        self.frequency = frequency
        self.sample_rate = sample_rate
        self.threshold_db = -25

        ##################################################
        # Variables
        ##################################################
        self.sample_rate_range = sample_rate_range = sample_rate
        self.frequency_range = frequency_range = frequency
```

Figure 4: Constructor, parameters, and variables for the audioDetect class

Figure 4 depicts the constructor of the audioDetect class with the FIFO file path, frequency, and sample rate as the variables. The parameter values for the FIFO, frequency, sample rate and threshold are defined. The FIFO file path is determined by the FIFO parameter block value, while the sample rate and frequency are provided by user input when they run the program. The threshold decibel value is established at this point (-25), while the sample rate and frequency ranges for the GUI are also established here.

```
##################################################
# Blocks
##################################################
self._sample_rate_range_range = Range(160e3, 40e6, 1e6, sample_rate, 200)
self._sample_rate_range_win = GrRangeWidget(self._sample_rate_range_range, self.set_sample_rate_range, "Sample Rate", "counter", float, QtCore.Qt.Horizontal, "value")

self.top_grid_layout.addWidget(self._sample_rate_range_win, 0, 0, 1, 1)
for r in range(0, 1):
    self.top_grid_layout.setRowStretch(r, 1)
for c in range(0, 1):
    self.top_grid_layout.setColumnStretch(c, 1)
self._frequency_range_range = Range(300e6, 3.8e9, 1e6, frequency, 200)
self._frequency_range_win = GrRangeWidget(self._frequency_range_range, self.set_frequency_range, "Frequency", "counter", float, QtCore.Qt.Horizontal, "value")

self.avg_power_db = analog.probe_avg_mag_sqrd_c(self.threshold_db, 0.0001)


self.top_grid_layout.addWidget(self._frequency_range_win, 0, 1, 1, 1)
for r in range(0, 1):
    self.top_grid_layout.setRowStretch(r, 1)
for c in range(1, 2):
    self.top_grid_layout.setColumnStretch(c, 1)
self.qtgui_sink_x_0 = qtgui.sink_c(
    1024, #fftsize
    window.WIN_BLACKMAN_hARRIS, #wintype
    frequency_range, #fc
    sample_rate_range, #bw
    "", #name
    True, #plotfreq
    True, #plotwaterfall
    True, #plottime
    True, #plotconst
    None # parent
)
self.qtgui_sink_x_0.set_update_time(1.0/10)
self._qtgui_sink_x_0_win = sip.wrapinstance(self.qtgui_sink_x_0.qwidget(), Qt.QWidget)

self.qtgui_sink_x_0.enable_rf_freq(False)

self.top_grid_layout.addWidget(self._qtgui_sink_x_0_win, 1, 0, 1, 8)
for r in range(1, 2):
    self.top_grid_layout.setRowStretch(r, 1)
for c in range(0, 8):
    self.top_grid_layout.setColumnStretch(c, 1)
self.blocks_multiply_const_vxx_0 = blocks.multiply_const_cc((1.0 / 2048.0))
self.blocks_interleaved_short_to_complex_0 = blocks.interleaved_short_to_complex(True, False,1.0)
self.blocks_file_source_0 = blocks.file_source(gr.sizeof_short*2, fifo, False, 0, 0)
self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)
```

Figure 5: Block Code for the Audio Detect class

The code represented in Figure 5 corresponds to the blocks in the GRC Flowgraph. Each block is defined with their respective attributes, and the corresponding signals they will need to pass the audio signal to the next block. The blocks and widgets for the GUI functionality to include the counters for the sample rate and frequency are established. Additionally, the Probe Avg Mag^2 block uses the threshold value to create an object called avg_power_dB. This object will be used to compare the power of each signal to the threshold value and trigger an alert later in the program.

```
##################################################
# Connections
##################################################
self.connect((self.blocks_file_source_0, 0), (self.blocks_interleaved_short_to_complex_0, 0))
self.connect((self.blocks_interleaved_short_to_complex_0, 0), (self.blocks_multiply_const_vxx_0, 0))
self.connect((self.blocks_multiply_const_vxx_0, 0), (self.qtgui_sink_x_0, 0))
self.connect((self.blocks_multiply_const_vxx_0, 0), (self.avg_power_db, 0))
```

Figure 6: Connection Code for Audio Detect class

Figure 6 depicts the connections between each block. The file source output leads to the input for the IShort-to-Complex input. The IShort-to-Complex output leads to the Multiply Constant input. The Multiply Const block output leads to the input of the GUI sink block, and the input of the avg_power_dB block that was created earlier.

```
def closeEvent(self, event):
    self.settings = Qt.QSettings("GNU Radio", "bladeRF_fifo_rx")
    self.settings.setValue("geometry", self.saveGeometry())
    self.stop()
    self.wait()

    event.accept()

def get_fifo(self):
    return self.fifo

def set_fifo(self, fifo):
    self.fifo = fifo
    self.blocks_file_source_0.open(self.fifo, False)

def get_frequency(self):
    return self.frequency

def set_frequency(self, frequency):
    self.frequency = frequency
    self.set_frequency_range(self.frequency)

def get_sample_rate(self):
    return self.sample_rate

def set_sample_rate(self, sample_rate):
    self.sample_rate = sample_rate
    self.set_sample_rate_range(self.sample_rate)

def get_sample_rate_range(self):
    return self.sample_rate_range

def set_sample_rate_range(self, sample_rate_range):
    self.sample_rate_range = sample_rate_range
    self.qtgui_sink_x_0.set_frequency_range(self.frequency_range, self.sample_rate_range)

def get_frequency_range(self):
    return self.frequency_range

def set_frequency_range(self, frequency_range):
    self.frequency_range = frequency_range
    self.qtgui_sink_x_0.set_frequency_range(self.frequency_range, self.sample_rate_range)
```

Figure 7: Getters and Setters for Audio Detect parameters

Figure 7 depicts the getter and setter functions for every variable and parameter, including the frequency and sample rate ranges for the GUI. These functions allow the user to set and retrieve the values of each variable. Figure 8 depicts the main function and argument parser function of the program. The argument parser function sets the FIFO file path, the frequency, and the sample rate to their default values. The main method invokes the argument parser, creates the GUI, and starts the SDR device with the tb.start()

command. This represents the last bit of code that the flowgraph generated. The main method also creates a timer that checks if the threshold is crossed.

```
def argument_parser():
    description = 'RX bladeRF binary samples from a FIFO, convert them to GR Complex values, and write them to a GUI sink. Additionally, it detects if the signal power crosses a threshold and alerts the user.'
    parser = ArgumentParser(description=description)
    parser.add_argument(
        "-f", "--fifo", dest="fifo", type=str, default='/tmp/rx_samples.bin',
        help="Set Full path to FIFO [default=%(default)r]")
    parser.add_argument(
        "--frequency", dest="frequency", type=eng_float, default=eng_notation.num_to_str(float(1e9)),
        help="Set Frequency [default=%(default)r]")
    parser.add_argument(
        "-s", "--sample-rate", dest="sample_rate", type=eng_float, default=eng_notation.num_to_str(float(2000000)),
        help="Set Sample Rate [default=%(default)r]")
    return parser


def main(top_block_cls=audioDetect, options=None):
    if options is None:
        options = argument_parser().parse_args()

    if StrictVersion("4.5.0") <= StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls(fifo=options.fifo, frequency=options.frequency, sample_rate=options.sample_rate)

    tb.start()

    tb.show()

    def sig_handler(sig=None, frame=None):
        tb.stop()
        tb.wait()

        Qt.QApplication.quit()

    signal.signal(signal.SIGINT, sig_handler)
    signal.signal(signal.SIGTERM, sig_handler)

    timer = Qt.QTimer()
    timer.start(500)
    timer.timeout.connect(lambda: None)

    qapp.exec_()

if __name__ == '__main__':
    main()
```

Figure 8: Argument Parser and Main Method of Audio Detect class

```
def check_threshold(self):
    avg_power_linear = self.avg_power_db.level()
    avg_power_db = 10 * math.log10(avg_power_linear)
    if avg_power_db > self.threshold_db:
        print(f"Threshold exceeded! Average power: {avg_power_db:.2f} dB. Downlink frequency: {self.frequency_range + 5e6} Hz")
```

Figure 9: Check Threshold Function

Figure 9 depicts the check threshold function, which checks if the average power of the signal exceeds the threshold value. It does this by calling the level() method on the avg_power_dB object and storing it into avg_power_linear variable. It then converts the linear scale of the average power value into a decibel scale and stores that value. The

converted value is then compared to the predefined threshold value and alerts the user if the threshold has been crossed. Additionally, the downlink frequency is calculated here by adding 5MHz to the center frequency, as most VHF and UHF signals have a downlink frequency within a 5MHz range.

```
# Add a QTimer to periodically check if the threshold is exceeded
self.check_threshold_timer = QtCore.QTimer()
self.check_threshold_timer.timeout.connect(self.check_threshold)

# Start the timer with a 1000 ms interval
self.check_threshold_timer.start(1000)
```

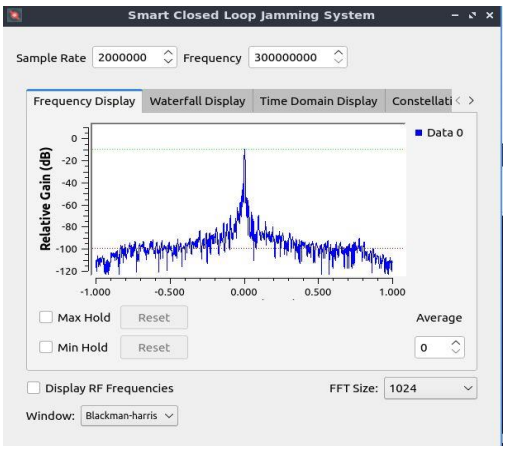
Figure 10: Timer activation code for the Check Threshold function

Figure 10 depicts the timer that actively invokes the check_threshold function. The timer is set off every 1000 milliseconds (1 second) to determine if the threshold has been crossed. If the threshold is exceeded when the timer performs the check, the print statement in the check threshold function will execute. Performing the check every second provides a reasonably quick pace to ensure that every spike is detected.

Transmission Spike Detection VHF HAM Radio Test

The first HAM radio test was conducted using the BaoFeng UV-5R radios, set to a frequency of 150.325MHz, with a sample rate of 2MHz. The frequency was well within the VHF frequency range of 30-300MHz (Midland Radio, 2021), and with the test being conducted in the mid-afternoon timeframe, the frequency was fairly active. VHF radio waves have a shorter range, and thus require more power for them to travel further (Midland Radio, 2021). The threshold value was initially set for -25dB, but as soon as the program started it activated the threshold regardless of whether the radio was keyed or not. The alert provided by the program stated that the average power when it crossed the threshold was between -15dB and -8dB, the threshold was set to -5dB for the test. A spectrum viewer was opened in Baudline, and the audio detection program was

established through the terminal. With both programs running, the BladeRF was then set to the established frequency, bandwidth, and sample rate before the transmission key was pressed on the radio. The key was held down for 10 seconds to conduct the radio check between the two radios. Figures 11 and 12 below display the spectrum viewers as the radio check was conducted.



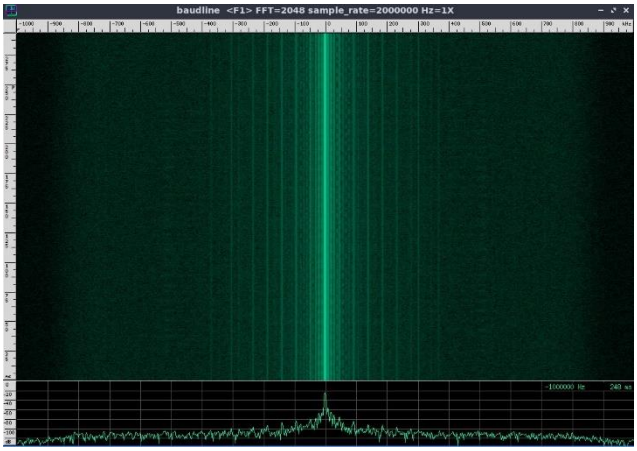


Figure 11: GRC Spectrum Viewer – VHF Test Figure 12: Baudline Spectrum Viewer – VHF Test

Both spectrum viewers were able to accurately represent the transmission spike during the radio check. At the specific frequency of 150.325MHz, there is a massive increase in the relative gain in comparison to the signal flow within 1MHz. The radio waves remained in a similar position until the transmission key was released.

```
live@live:~/Desktop$ ./audioDetect.py --sample-rate=2e6 --frequency=150.325e6
Threshold exceeded! Average power: -2.69 dB. Downlink frequency: 155325000.0 Hz
Threshold exceeded! Average power: -3.30 dB. Downlink frequency: 155325000.0 Hz
Threshold exceeded! Average power: -2.15 dB. Downlink frequency: 155325000.0 Hz
Threshold exceeded! Average power: -2.84 dB. Downlink frequency: 155325000.0 Hz
Threshold exceeded! Average power: -4.22 dB. Downlink frequency: 155325000.0 Hz
Threshold exceeded! Average power: -4.70 dB. Downlink frequency: 155325000.0 Hz
Threshold exceeded! Average power: -3.79 dB. Downlink frequency: 155325000.0 Hz
Threshold exceeded! Average power: -3.96 dB. Downlink frequency: 155325000.0 Hz
```

Figure 13: Threshold being crossed during VHF radio check

While the radio check was being conducted, the terminal running the audio detect program was printing the alert message that the threshold had been exceeded. As depicted

in Figure 13, the program alerts the user that the threshold was crossed, prints the offset downlink frequency, and calculates the average power at the time of detection.

Transmission Spike Detection UHF HAM Radio Test

The next test to be conducted was the UHF radio test. The UHF frequency range spans from 300MHz to 3GHz (Midland Radio, 2021), so the BaoFeng UV-5R radios were set to a frequency of 440.125MHz with a sample rate of 2MHz. UHF radio waves are thinner than VHF and don't require as much power to transmit further distances, so the threshold was established at -15dB. The frequency of the BladeRF was reprogrammed to 440.125MHz, and the spectrum viewer in baudline was started before activating the audio detection program. The transmission key on the radios was then held for 10 seconds to conduct a sustained radio check. Figures 14 and 15 below display the spectrum viewers as the radio check was conducted.

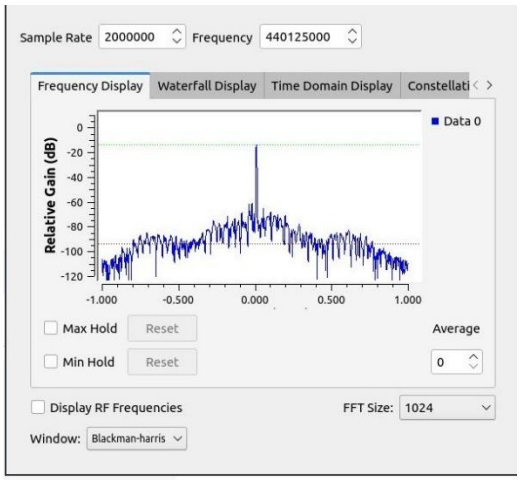


Figure 14: GRC Spectrum Viewer – UHF Test

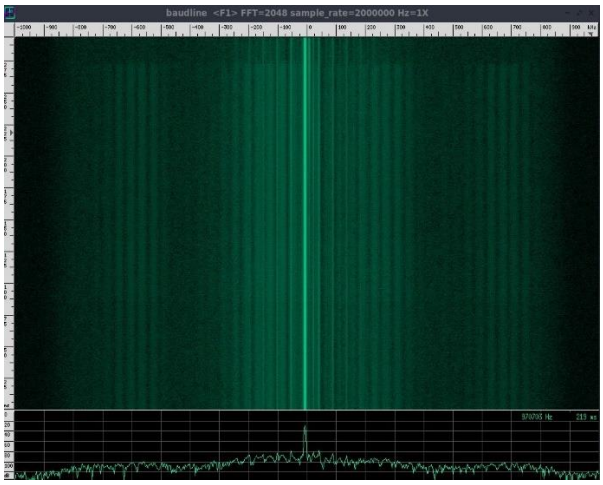


Figure 15: Baudline Spectrum Viewer – UHF Test

Again, both spectrum viewers were able to accurately capture the transmission spike during the radio check. At the specific frequency of 440.125MHz, there is a massive increase in the relative gain in comparison to the signal flow within the established sample rate. The radio waves again remained in a similar position until the transmission

key was released. While the radio check was being conducted, the terminal running the audio detect program was printing the alert message that the threshold had been exceeded as it had with the VHF test.

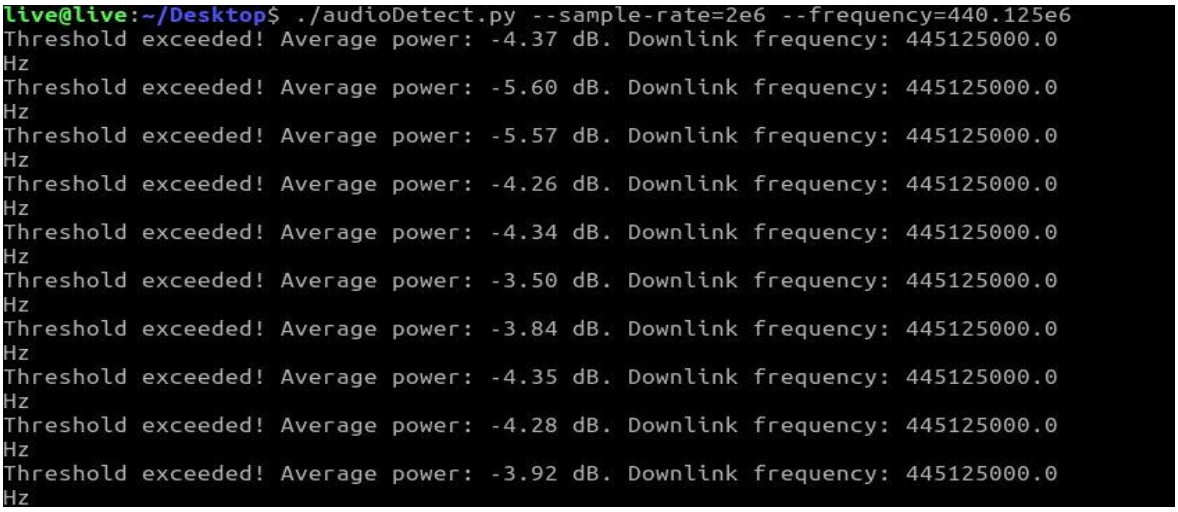


Figure 16: Threshold being crossed during UHF radio check

As depicted in Figure 16, the program alerts the user that the threshold was crossed, prints the offset downlink frequency, and calculates the average power at the time of detection.

Transmission Spike Detection Wi-Fi Laptop Test Case

The next set of test cases are regarding using frequency around home Wi-Fi on the 2.4Ghz frequency range. This feature was tested using a Dell Latitude E5420 operating on a 2G internet connection, which uses a frequency of 2.4GHz. The audio detection program was activated in terminal with the threshold set to -50dB to begin the test. Figure 17 below, depicts the baseline threshold for the internet connectivity at around -33dB. This threshold alert was consistent for the entire 15 seconds the program was running.

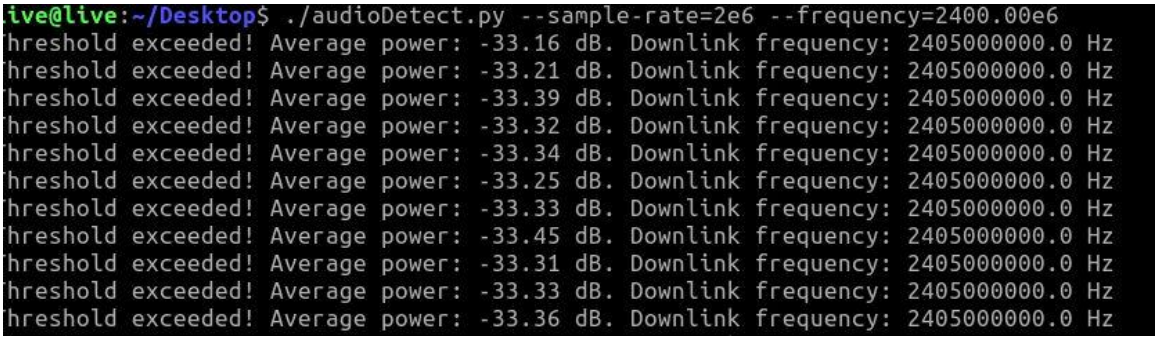


Figure 17: Baseline Threshold for Laptop Connection Test

The laptop was then disconnected from the 2G internet connection, and the audio detection threshold was set to -30dB. Even when the Dell laptop was re-connecting to the internet, the threshold alert was not triggered. A Verizon Samsung S21 5G was also connect to the 2G internet, as well as an Xbox Series X, but still no alerts were triggered. Figure 18 below depicts the terminal when the test was ongoing.

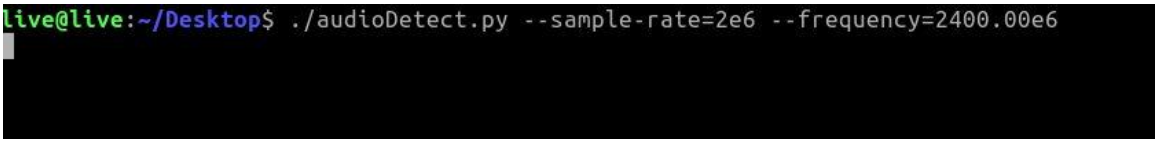


Figure 18: Wi-Fi Connectivity Terminal Test Result

Even when multiple devices were trying to attempt to connect, the decibel rate did not exceed the -30dB. Rather than repeat the test using the 5G connection, a similar test was conducted using a wireless mouse.

Transmission Spike Detection Bluetooth Mouse Test Case

The next test case to be conducted was the Bluetooth mouse test using a Microsoft Bluetooth Ergonomic Mouse. This mouse operates on a Bluetooth connection on a 2.45Ghz frequency with a range of about 30 feet (Microsoft Accessories, 2023). The antenna was placed between the mouse and the Dell computer. The threshold for the audio detection program was set for -25dB, but no alerts were triggered. The spectrum viewer that is displayed in figure 19 below shows that the maximum gain for the

2.45GHz frequency range had a baseline between -45dB and -50dB. Upon discovering the baseline gain for the 2.45Ghz frequency range, the threshold value was switched to -40dB. The audio detection program was started, and the mouse was clicked repeatedly while next to the BladeRF antenna. The audio detection program processed several threshold alerts with the average gain around the -35dB mark. In order to further test the receiving strength, the threshold value was then set to -30dB, and the mouse click button was held down for a longer period of time.

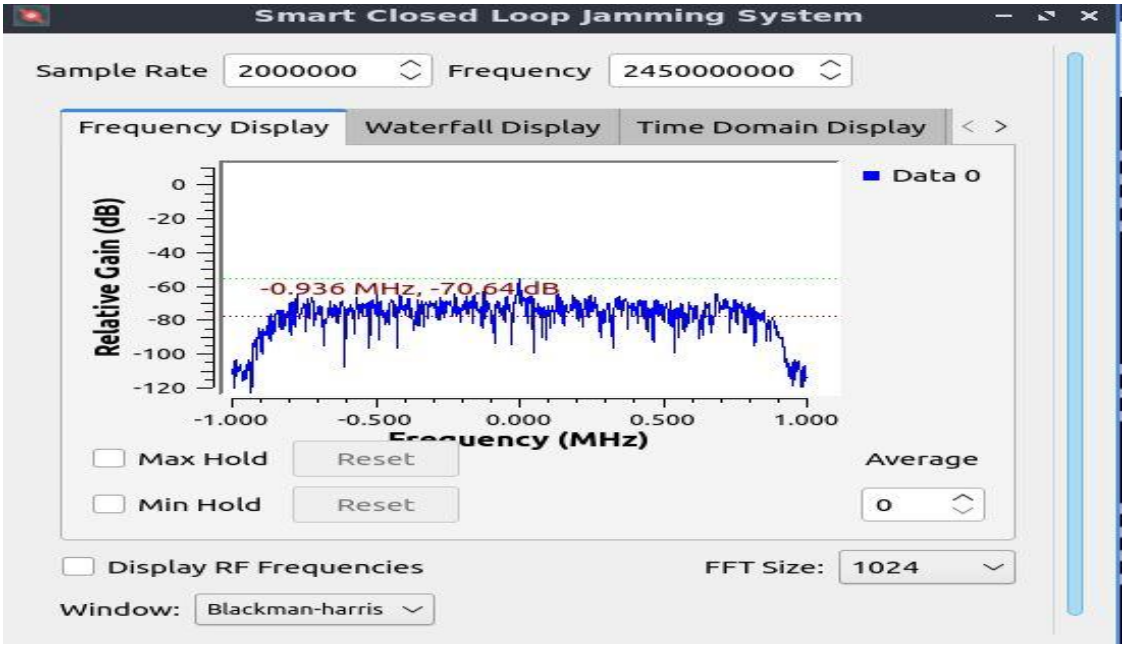


Figure 19: Baseline for 2.45GHz Frequency Range

Far fewer alerts were received this with this test, but the threshold was crossed three times at around -29dB. Figure 20 below displays the test results in the terminal running the audio detection program. UHF waves are thin, so they do not require as much power to be transmitted. Additionally, they also have excellent reception when they have line of sight communication, so they do not require as much power especially at closer ranges. Both reasons are possible These are contributing factors as to why the threshold was not crossed when the decibel rate was raised.

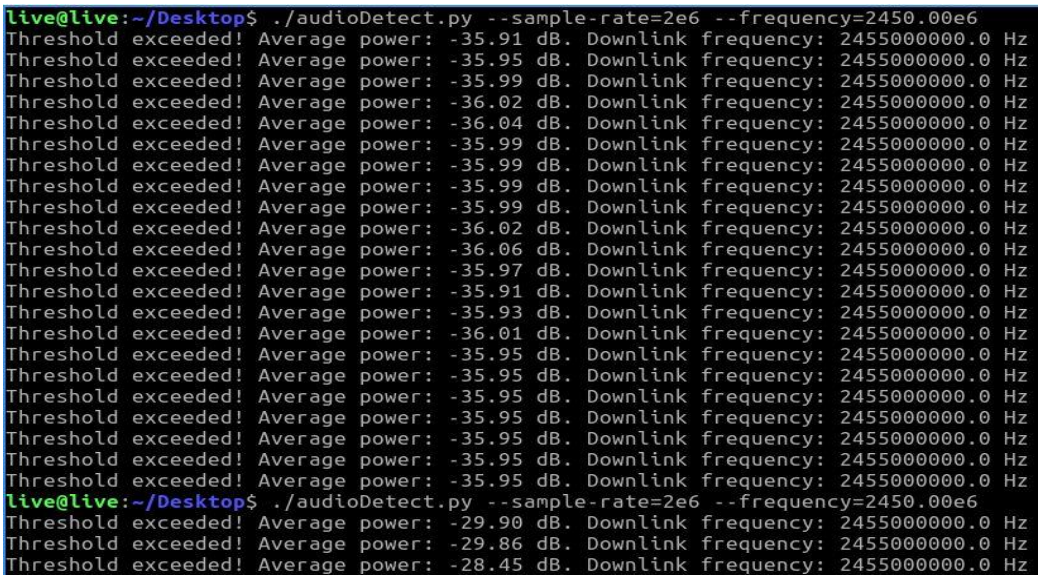


Figure 20: Mouse Audio Detection Threshold Alert

Transmission Spike Detection Cell Phone Test

The last test case for the transmission spike detection feature is the Verizon cell phone test case. As displayed in figure 1, the uplink frequency range for Verizon cell phones is between 776MHz and 787MHz (Amplifiers, 2022). To ensure that this spectrum range was covered for the test call, the BladeRF was set to a receiving frequency of 781.5MHz, with a sample rate of 12MHz. This allows the program to cover from 775.5MHz to 787.5MHz. With the audio detection program running under the same parameters, the BladeRF was given the command to start receiving samples. The baseline gain for the 781.5MHz frequency range was approximately -50dB per the spectrum viewer, so the threshold value was set to -40dB and the audio detection program was activated through the terminal. The cell phone was then placed next to the receiving antenna to ensure that the signal was being captured. Figure 21 below shows the spike when the phone call was initiated.

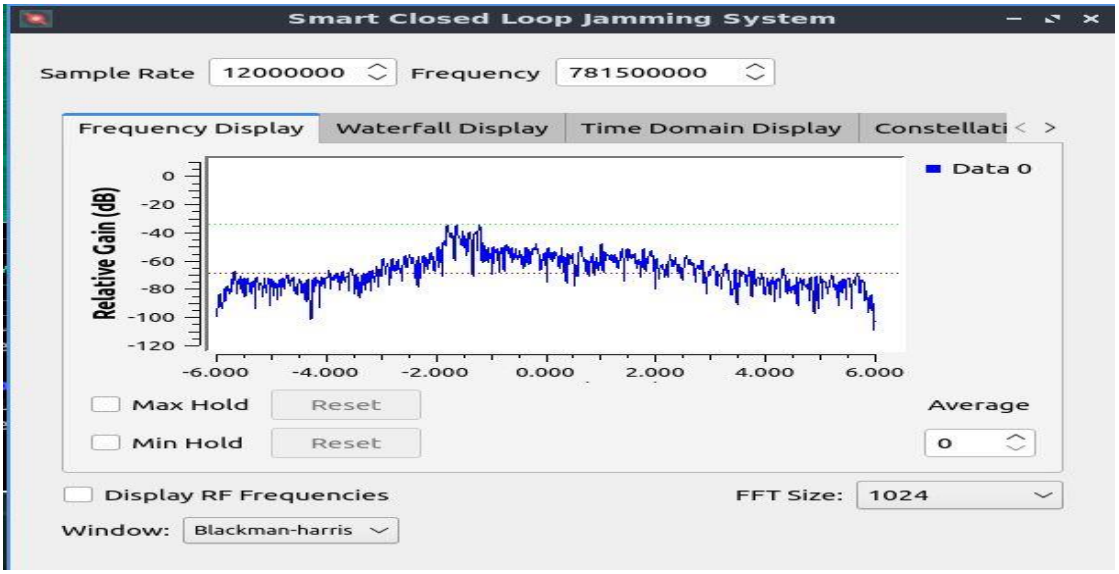


Figure 21: GRC Spectrum Viewer for Verizon Test Case

Unlike the other test cases, the spike does not occur at the center frequency, but roughly 2MHz below it. This has to do with cell phone carriers being authorized to use frequency division duplex bands, which separate the uplink and downlink frequencies by a larger margin than normal. These blocks are reserved for the cell carriers to use, so the phone will search for an open frequency within the reserved block before the call is officially established. Figure 22 below displays the spectrum viewer in baudline captured a few moments after the phone call was initiated. In this spectrum viewer, the transmission spike has occurred approximately 1MHz above the center frequency. This is because the signal fluctuates while the call is connected. Regardless of which spectrum viewer was utilized, the frequency of the transmission spike shifted several times throughout the call. The natural offset of calculating the downlink frequency was removed for this test since Verizon has a specified carrier block of 746Mhz to 757Mhz for their downlink frequencies (Amplifiers, 2022). The audio detection program triggered several alerts once the call was in place, with an average gain between -19dB and -24dB.

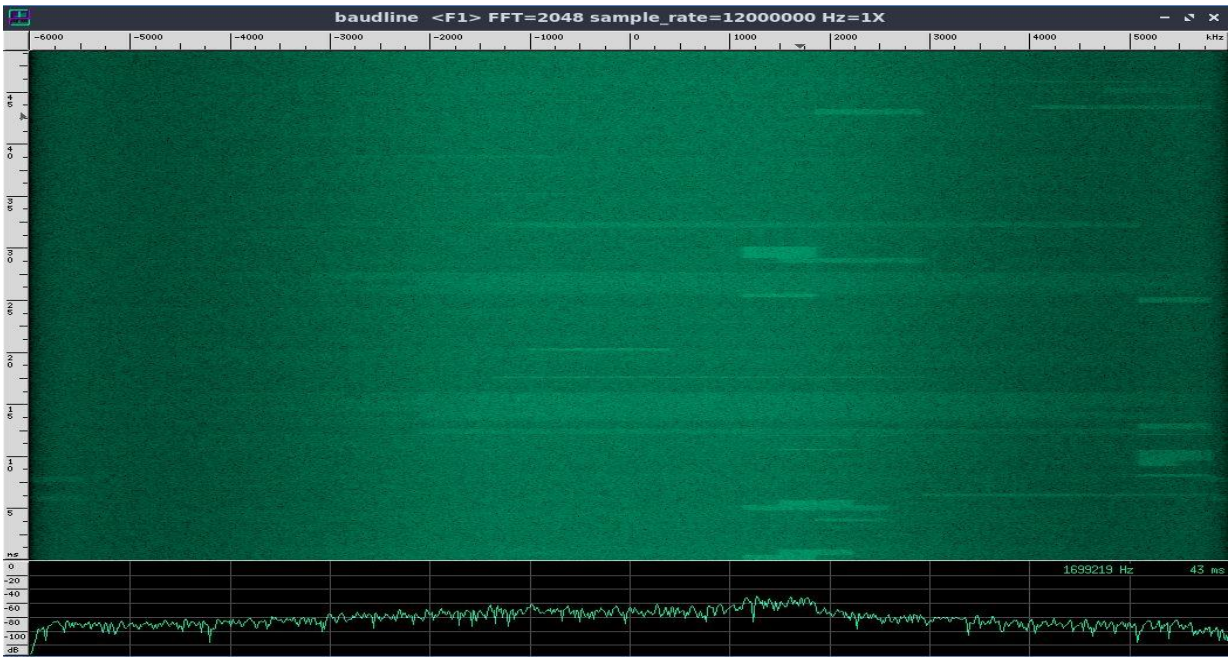


Figure 22: Baudline Spectrum Viewer for Verizon Test Case

Figure 23 below shows the audio detection program running in the terminal while the call was in place for 15 seconds.

```
live@live:~/Desktop$ ./audioDetect.py --sample-rate=12e6 --frequency=781.500e6
Threshold exceeded! Average power: -24.27 dB.
Threshold exceeded! Average power: -24.11 dB.
Threshold exceeded! Average power: -22.93 dB.
Threshold exceeded! Average power: -24.10 dB.
Threshold exceeded! Average power: -23.35 dB.
Threshold exceeded! Average power: -24.27 dB.
Threshold exceeded! Average power: -23.37 dB.
Threshold exceeded! Average power: -24.75 dB.
Threshold exceeded! Average power: -22.01 dB.
Threshold exceeded! Average power: -20.63 dB.
Threshold exceeded! Average power: -21.89 dB.
Threshold exceeded! Average power: -18.99 dB.
```

Figure 23: Verizon Call Audio Detection Threshold Alert

When the call was ended, the receiver was left running for a few minutes to observe if other transmissions would trigger an alert. There were a few more threshold alerts due to the receiver picking up stray signals, but there were no continuous alerts for any sustained periods.

Jamming Signal GNU Radio Companion Flowgraph

To develop the jamming signal feature, another signal flowgraph was built in GNU Radio Companion to generate python code. This flowgraph generates an audio

sample and applies frequency modulation to it. The flowgraph then amplifies the gain and transmits the signal using a device defined by the user.

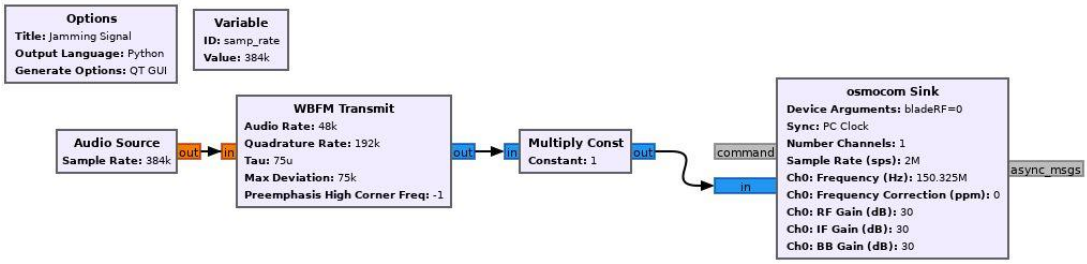


Figure 24: Jamming Signal GRC Flowgraph

This flowgraph first uses an audio source block, which generates the audio signal from the input source and passes it on as a continuous stream of audio samples to the Wideband Frequency Modulation (WBFM) Transmit block (GNU Radio, *Audio source* 2022). The WBFM Transmit block takes the audio samples and transmit them using WBFM (WBFM Transmit, *WBFM transmit* 2019). The modulated samples are the passed through a Multiply Const block to amplify the signal by a set value, thus affecting the gain. The set value for this Multiply Const block is set to 1 to ensure that there is no unwanted interference with outside receivers and to comply with local transmission laws established by the FCC. The final block is the Osmocom Sink block, which translates the modulated signals through an SDR that is specified by the user (Forge, 2022). The gain, sample rate and frequency can also be adjusted in this block. The Osmocom Sink block is compatible with several different devices, including the BladeRF, which makes it feasible to develop future capabilities using other transmissions sources (Forge, 2022).

Jamming Signal Python Code

Once the flowgraph was complete, the python code was generated from GNU Radio Companion. This program was much smaller than the transmission spike detection

feature since all it does is transmit a generated signal. The code that was developed from the flowgraph in figure 24 is depicted in the figures below.

```
from packaging.version import Version as StrictVersion

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print("Warning: failed to XInitThreads()")

from gnuradio import analog
from gnuradio import audio
from gnuradio import blocks
from gnuradio import gr
from gnuradio.filter import firdes
from gnuradio.fft import window
import sys
import signal
from PyQt5 import Qt
from argparse import ArgumentParser
from gnuradio.eng_arg import eng_float, intx
from gnuradio import eng_notation
import osmosdr
import time
```

Figure 25: Python Imports from GRC Flowgraph for Jamming Signal Feature

Similar to the audio detection feature, the program has several imports from the GNU Radio package such as the blocks, analog, and eng_Notation, as well as the audio package needed to generate a signal. The Qt package is imported from the PyQt5 application framework in order to create a GUI that can be used to close the program. The sys, sip, and signal packages are also imported for the python environment to manipulate the signals generated for the transmission. Finally, the osmosdr package is imported, which is required for the Osmocom Sink block to interact with the various different SDR devices. Figure 26 depicts the constructor and variables for the jamming_TX class. The constructor contains the code to open a small GUI that will close the program when exited. The only variable taken by this class is the sample rate, which is established at 384 kilohertz. 384 kilohertz is the maximum sample rate when the BladeRF device is transmitting so that it doesn't interfere with nearby frequencies.

```
class jamming_TX(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Jamming Signal", catch_exceptions=True)
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Jamming Signal")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "bladeRF_TX")

        try:
            if StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
                self.restoreGeometry(self.settings.value("geometry").toByteArray())
            else:
                self.restoreGeometry(self.settings.value("geometry"))
        except:
            pass

        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 384000
```

Figure 26: Jamming Signal Constructor and Variables

Figure 27 depicts the blocks and their respective variables developed by the program. The transmission frequency, sample rate, gain, and most importantly the transmitting device are established here. In the WBFM Transmit Block, the audio rate, quadrature rate, and signal deviation variables are established. These variables allow the program to amplify the signal.

```
        ##################################################
        # Blocks
        ##################################################
        self.osmosdr_sink_0 = osmosdr.sink(
            args="numchan=" + str(1) + " " + 'bladeRF=0'
        )
        self.osmosdr_sink_0.set_time_now(osmosdr.time_spec_t(time.time()), osmosdr.ALL_MBOARDS)
        self.osmosdr_sink_0.set_sample_rate(2e6)
        self.osmosdr_sink_0.set_center_freq(440.125e6, 0)
        self.osmosdr_sink_0.set_freq_corr(0, 0)
        self.osmosdr_sink_0.set_gain(60, 0)
        self.osmosdr_sink_0.set_if_gain(60, 0)
        self.osmosdr_sink_0.set_bb_gain(60, 0)
        self.osmosdr_sink_0.set_antenna('', 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)
        self.blocks_multiply_const_vxx_0 = blocks.multiply_const_cc(1)
        self.audio_source_0 = audio.source(samp_rate, '', True)
        self.analog_wfm_tx_0 = analog.wfm_tx(
            audio_rate=48000,
            quad_rate=192000,
            tau=(75e-6),
            max_dev=75e3,
            fh=(-1.0),
        )

        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_wfm_tx_0, 0), (self.blocks_multiply_const_vxx_0, 0))
        self.connect((self.audio_source_0, 0), (self.analog_wfm_tx_0, 0))
        self.connect((self.blocks_multiply_const_vxx_0, 0), (self.osmosdr_sink_0, 0))
```

Figure 27: Jamming Signal Blocks and Connections

Additionally, the connections from each block are represented. The audio source output leads to the input of the WBFM Transmit block. The WBFM Transmit block output then goes to the Multiply Const block input, which leads to the input of the Osmocom Sink.

```
    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "bladeRF_TX")
        self.settings.setValue("geometry", self.saveGeometry())
        self.stop()
        self.wait()

        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate



def main(top_block_cls=jamming_TX, options=None):

    if StrictVersion("4.5.0") <= StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()

    tb.start()

    tb.show()

    def sig_handler(sig=None, frame=None):
        tb.stop()
        tb.wait()

        Qt.QApplication.quit()

    signal.signal(signal.SIGINT, sig_handler)
    signal.signal(signal.SIGTERM, sig_handler)

    timer = Qt.QTimer()
    timer.start(500)
    timer.timeout.connect(lambda: None)

    qapp.exec_()

if __name__ == '__main__':
    main()
```

Figure 28: Jamming Signal Getters, Setters, and Main Function

Figure 28 depicts the getters and setter functions for the sample rate variable, as well as the main function for the jamming signal feature. The main function works by creating an instance of the jamming_TX class and starting the corresponding devices' transmitter. It does not stop transmitting until the user terminates the program in the terminal.

Jamming Signal Test

The first test for the jamming signal feature was the HAM Radio test utilizing the BaoFeng UV-5R radios on low power. The first channel to be tested was within the VHF frequency range at 150.325MHz. The jamming signal program was set to the same frequency, with a gain of 60dB. This is the maximum amount of gain that the BladeRF

can produce. The test was conducted with the radios placed next to the transmitter, across the room, and with them placed in another room. Figure 29 depicts the jamming signal being initiated in terminal for the test.

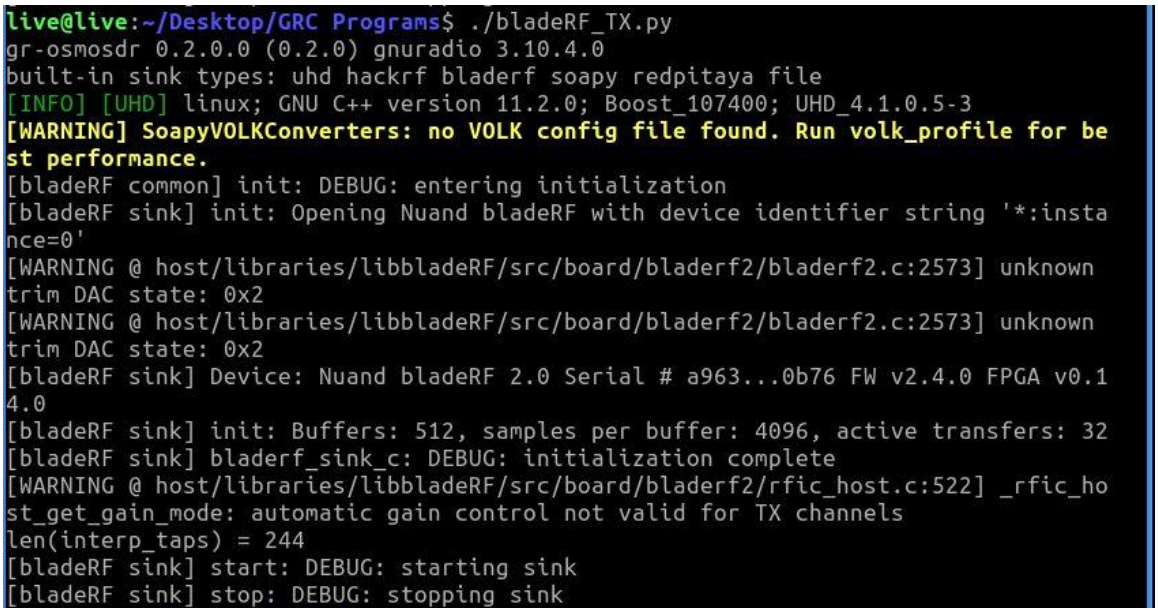


Figure 29: Jamming Signal program initiated in terminal

Once the program as initiated, there was a constant static that played across both radios while the transmitter was active. The static was a bit weaker when the radio was placed in a separate room, but each radio was still able to at least pick up some pieces of the transmission. Both radios were able to receive the signal being transmitted by the BladeRF device; however, even while receiving that signal, they were still able to transmit between each other. The signal being transmitted by the BladeRF at its maximum gain was not strong enough to block out the radio's transmission capability. The radios were then programmed to 440.125MHz to test the jamming feature for UHF voice signals. The jamming signal program was then switched to the same frequency and activated through the terminal while the radios were placed in the same positions as the VHF test. The static transmissions received by the radios were weak and broken when

they were closer to the BladeRF antenna, but the radio screen kept blinking red indicating that it was receiving a transmission. When there was a wall or object placed in between the radio and the transmitting antenna, the static audio was more constant, but the screen was no longer flashing red. This indicates that even though the radio could receive the audio samples from the BladeRF, it did not necessarily perceive that a device was trying to contact it. Regardless of the distance or whether the radio detected a transmission, both radios were still able to transmit out and conduct a successful radio check. Even with the thinner wavelengths from the UHF radio waves, the BladeRF was still not able to provide enough interference to prevent the radios from communicating to each other. The BladeRF was then set to 2.4GHz to conduct the jamming signal test on the router of a personal residence 2G Wi-Fi internet connection. The properties for this Wi-Fi connection are displayed in figure 30. The jamming signal program was then initiated in terminal but there were no noticeable effects to the phone or laptop when they established connection. The devices were still able to connect to web browsers and view different websites, as well as upload and download files without any noticeable drop in speed.

Figure 30: Personal Residence 2G Wi-Fi properties

While the low power output from the BladeRF was certainly a factor, a bigger contributing factor is that the transmission signal from the BladeRF is only using frequency modulation. Routers use Orthogonal Frequency Division Multiplexing

(OFDM) modulation schemes to communicate with other devices for Wi-Fi connections (Router-Switch.com, 2020). Figure 30 displays that the tested router is using the protocol 802.11n, which uses ODFM modulation for data transmission. OFDM works by splitting signals apart into several closely spaced subchannel frequencies instead of one single Wideband channel frequency (Router-Switch.com, 2020). OFDM is very resilient to signal interference due to the amount of bandwidth and channels that are allocated for use. This modulation scheme is especially effective against interference because it is designed to support multiple closed-space and overlapping channels (Router-Switch.com, 2020). The last jamming signal test is the Verizon cell phone test. This test was conducted using a Samsung S21 5G in addition to the BladeRF. The Samsung S21 5G in this test uses Quadrature Amplitude Modulation (QAM). QAM is a modulation scheme that works splitting the transmission into two different carrier waves and then changing the amplitudes of the two carrier waves (Frenzel, 2018). The cell phone was placed directly next to the transmission antenna, and a call was placed for about 10 minutes while the jamming signal program was initiated through the terminal. The jamming signal frequency was switched throughout the call to cover Verizon's downlink frequency range (746MHz – 757MHz). The frequency was increased by .5MHz every minute to identify any signal potential interference throughout the range. There was no significant interference on any of the frequencies within the downlink range during the call. Due to the difference in modulation schemes, the phone was not did not register the transmissions from the BladeRF. A Samsung S21 5G can use up to two watts of power to establish and maintain a phone call, which vastly outpowers the BladeRF Micro xA5 maximum power output of four milliwatts (Nuand, 2023). Even with the transmitting

antenna directly touching the phone during the call, this was not enough power to interfere with the call.

Limitations

Development of the Smart Closed-Loop Jamming system was hindered by several limitations imposed by the issued BladeRF device. The first limitation was that there is only one antenna with which to operate. Even though the BladeRF has two transmit and two receiver ports, only one port could be used at a time. This forced each feature to be developed individually so as not to cause a port error when trying to transmit or receive. The concept behind the Smart Closed-Loop Jamming System is that it automatically sends a jamming signal when it detects a transmission spike. This can only be implemented with a minimum of two antennas to allow the device to receive on the uplink frequency and transmit on the offset downlink frequency. The second limitation is that gain and maximum power output of the BladeRF are not strong enough to generate a signal that can block out other transmissions. The tests revealed that the program successfully made the BladeRF transmit to an outside sources, which satisfies the intent of the features' functionality. A more powerful device is required to generate a signal strong enough to block out other transmissions sources.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

The key deliverables for the Smart Closed-Loop Jamming System were the development of the two features, which are the transmission spike detection feature and the jamming signal emission feature. The completion of these features provides the foundation that the Smart Closed-Loop Jamming System needs to develop future capabilities.

Conclusion

While there have been some limiting factors, there is significant progress made towards the completion of both features. The transmission spike detection feature is able to accurately identify any spikes within the sample rate range of the center frequency. It is able to calculate the downlink frequency based on the offset defined by the user, which tells them the frequency they need to initate the jamming signal. The jamming signal feature is capable of transmitting a signal using frequency modulation that can cause minor interference to radio systems. With a more powerful emission source, the testing has proven that it is possible to generate interference that can significant interference with radio signals. While this did not work for the cell phone or Wi-Fi tests due to the modulation schemes and power output, it provides a backbone to upgrade this feature.

Future Work

The first upgrade to the Smart-Closed Loop Jamming System is to combine functionality of both programs so that the system works as intended. The first step is to import the jamming signal class into the audio detection class so that it can be used when needed. When a spike is triggered in the audio detection class, the jamming signal class can be initatied with the gain and downlink frequency as parameters. This instance of the jamming signal will continue to run until closed out by the user, as it needs to be

continuous. This may lead to overlapping transmissions if multiple spikes are detected or to spikes being ignored if the transmitter is occupied when a new spike is detected. To prevent this, a queue or buffer system will have to be implemented to store information about the spikes, such as the timing and frequency. When the transmitter is free it can move through the queue and transmit as needed. Other future capabilities would be to improve the jamming signal feature to emit different modulation types. Some possible modulation types revolve around the Institute of Electrical and Electronic Engineers (IEEE) standards, which would allow the system to provide emissions to protect internet networks. Another future capability would be to implement an amplitude modulation for the jamming signal instead of frequency modulation. Figure 31 depicts a GRC flowgraph that can create an apmplitude modulated signal.

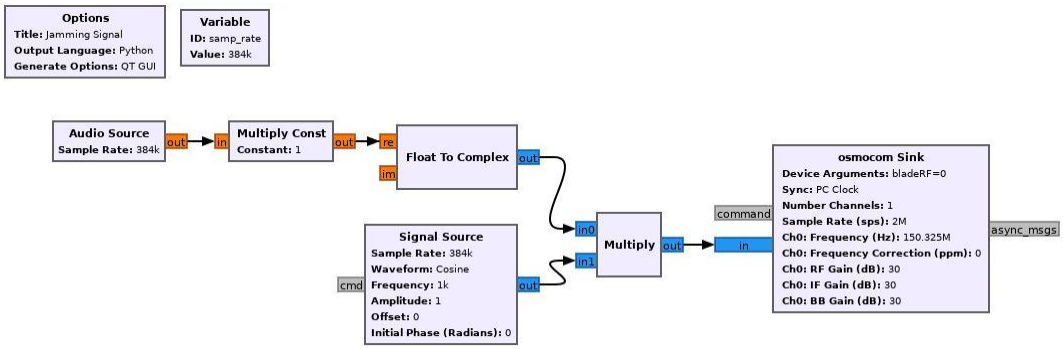


Figure 31: Amplitude Modulation Flowgraph

In the flowgraph provided in Figure 28, a Multiply Const block amplifies the original audio signal. The Float to Complex block and the Signal Source block replace the WBFM Transmit Block that was used for the FM transmitter. The combination of these two blocks generates a carrier signal, whose depth will be different from the original audio source that has been amplified. These two sources are then blended together using a Multiply block, before reaching the Osmocom Sink block, which will be used to transmit the signal. The device parameter can be changed in the Osmocom Sink block, so

the user is not limited to the BladeRF package. The additions made to the Smart-Closed Loop Jamming System have set the foundation for future development teams to complete these capabilities.

REFERENCES

1. Amplifiers, W. (2023, May 20). Cell phone frequency bands by provider. WilsonAmplifiers.com. Retrieved January 8, 2023, from https://www.wilsonamplifiers.com/blog/frequencies-by-provider/

2. Wireless intercept and "wiphishing". Wireless Intercept and "WiPhishing" | Information Technology | University of Pittsburgh. (2019, August 28). Retrieved January 15, 2023, from https://www.technology.pitt.edu/security/wireless-intercept-and-wiphishing

3. BladeRF 2.0 micro XA5. Nuand. (n.d.). Retrieved December 10, 2022, from https://www.nuand.com/product/bladeRF-xA5/

4. Ruth, D. (2012, January). AR.Drone: Security threat analysis and exemplary attack to track persons. Retrieved January 20, 2023, from https://www.researchgate.net/publication/258713432_ARDrone_Security_threat_analysis_and_exemplary_attack_to_track_persons

5. McClanahan, P. (2022, November 1). *1.4 attacks - types of attacks*. Engineering LibreTexts. Retrieved January 21, 2023, from https://eng.libretexts.org/Courses/Delta_College/Information_Security/01%3A_Information_Security_Defined/1.4_Attacks_-_Types_of_Attacks

6. Andress, J. (2015). The Basics of Information Security: Understanding the fundamentals of infosec in theory and Practice. Syngress.

7. Check point software's mobile security report 2021 shows almost every organization globally experienced a mobile malware attack during the past year. Check Point Software. (2021, April 12). Retrieved January 24, 2023, from https://www.checkpoint.com/press/2021/check-point-softwares-mobile-security-report-2021-shows-almost-every-organization-globally-experienced-a-mobile-malware-attack-during-the-past-year/

8. Berco, S. (2022, November 18). Managing vulnerabilities, from attack interception to result prevention. Industrial Cybersecurity Pulse. Retrieved January 28, 2023, from https://www.industrialcybersecuritypulse.com/threats-vulnerabilities/managing-vulnerabilities-from-attack-interception-to-result-prevention/

9. James, N. (2022, December 22). The Staggering Cost of Cyberattacks: How Much Money do Businesses Actually Lose? Retrieved January 28, 2023, from https://www.getastra.com/blog/security-audit/cost-of-cyberattacks/#:~:text=The%20average%20cost%20of%20a%20ransomware%20attack%20went%20down%20slightly,data%20breach%2C%20USD%204.35%20million.

10. International, I. S. C. O. (2021, October 25). What is narrowband and wideband interference?: ISCO International. ISCO. Retrieved January 30, 2023, from https://iscointl.com/narrowband-and-wideband-interference-cancellation/#:~:text=NBI%20is%20characterized%20by%20relatively,or%20from%20non%2Dcellular%20sources.

11. Unauthorized radio operation. Federal Communications Commission. (2021, January 14). Retrieved February 3, 2023, from https://www.fcc.gov/consumers/guides/unauthorized-radio-operation#:~:text=Federal%20law%20generally%20prohibits%20radio,other%20civil%20and%20criminal%20penalties.

12. Personal Radio Services. Federal Communications Commission. (2020, March 13). Retrieved February 3, 2023, from https://www.fcc.gov/consumers/guides/personal-radio-services-prs-keeping-touch

13. Jammer enforcement. Federal Communications Commission. (2020, May 3). Retrieved February 4, 2023, from https://www.fcc.gov/general/jammer-enforcement#:~:text=Jamming%20Prohibited&text=The%20use%20of%20a%20phone,classroom%2C%20residence%2C%20or%20vehicle.

14. *OFDM vs. OFDMA - router-switch.com*. (2020, December 31.). Retrieved March 13, 2023, from https://www.router-switch.com/faq/ofdm-vs-ofdma.html

15. Mobile, V. (2020, January 1). *Cyberattacks on mobile devices are on the rise*. Verizon Business. Retrieved January 16, 2023, from https://www.verizon.com/business/resources/whitepapers/cyberattacks-on-mobile-devices-are-on-the-rise/

16. Communications, P. (2019, November 11). *What is Broadband Network Noise and why is it difficult to find?* Broadband Network Noise and Ingress. Retrieved January 26, 2023, from https://www.promptlink.com/media-library/blog/what-is-broadband-network-noise-and-why-is-it-difficult-to-find.html#:~:text=Radiofrequency%20interference%20%E2%80%93%20Unshielded%20copper%20wires,noise%20within%20a%20broadband%20network.

17. SourceForge. (2020, July 4). *DragonOS_Focal*. SourceForge. Retrieved October 16, 2023, from https://sourceforge.net/projects/dragonos-focal/

18. Ryzerth. (2022, December). SDR++. Retrieved January 16, 2023, from https://www.sdrpp.org/

19. Eaton, J. (2022). *GNU octave*. GNU Octave. Retrieved January 28, 2023, from https://octave.org/

20. BaoFeng Radios. (2020, December 19). *Baofeng UV-5R*. BaoFeng Radios. Retrieved February 2, 2023, from https://baofengtech.com/product/uv-5r/

21. GNU Radio. (2022). *About GNU radio · Gnu Radio*. GNU Radio. Retrieved January 24, 2023, from https://www.gnuradio.org/about/

22. GNU Radio, W. (2022, July 24). *File source*. File Source - GNU Radio. Retrieved February 16, 2023, from https://wiki.gnuradio.org/index.php?title=File_Source

23. GNU Radio, W. (2022, August 5). *Ishort to complex*. IShort To Complex - GNU Radio. Retrieved February 16, 2023, from https://wiki.gnuradio.org/index.php?title=IShort_To_Complex

24. GNU Radio, W. (2021, October 24). *Multiply const*. Multiply Const - GNU Radio. Retrieved February 16, 2023, from https://wiki.gnuradio.org/index.php?title=Multiply_Const

25. GNU Radio, W. (2019, July 15). *Qt Gui Sink*. QT GUI Sink - GNU Radio. Retrieved February 16, 2023, from https://wiki.gnuradio.org/index.php?title=QT_GUI_Sink

26. GNU Radio, W. (2019, September 5). *Probe avg mag^2*. Probe Avg Mag^2 - GNU Radio. Retrieved February 16, 2023, from https://wiki.gnuradio.org/index.php?title=Probe_Avg_Mag%5E2

27. GNU Radio, W. (2022, January 7). *Threshold*. Threshold - GNU Radio. Retrieved April 16, 2023, from https://wiki.gnuradio.org/index.php?title=Threshold

28. Midland Radio. (2021, April 24). *UHF vs. VHF - understanding the differences in radio frequencies*. Midland Radio. Retrieved March 3, 2023, from https://midlandusa.com/blogs/blog/uhf-vs-vhf-understanding-the-differences-in-radio-frequencies

29. Accessories, M. (2023). *Microsoft bluetooth® Ergonomic Mouse*. Microsoft Accessories. Retrieved April 1, 2023, from https://www.microsoft.com/en/accessories/products/mice/microsoft-bluetooth-ergonomic-mouse?activetab=pivot%3Atechspecstab#tab1be9a1924-e8f5-469e-b86f-416b31d75785

30. Forge, L. (2022, July). *SDR (Software Defined Radio) " GR-OSMOSDR*. GrOsmoSDR - gr-osmosdr - Open Source Mobile Communications. Retrieved February 17, 2023, from https://osmocom.org/projects/gr-osmosdr/wiki

31. GNU Radio, W. (2022, January 13). *Audio source*. Audio Source - GNU Radio. Retrieved February 16, 2023, from https://wiki.gnuradio.org/index.php?title=Audio_Source

32. WBFM Transmit, W. (2019, August 10). *WBFM transmit*. WBFM Transmit - GNU Radio. Retrieved February 16, 2023, from https://wiki.gnuradio.org/index.php?title=WBFM_Transmit