PreTalen
innovative efficiency

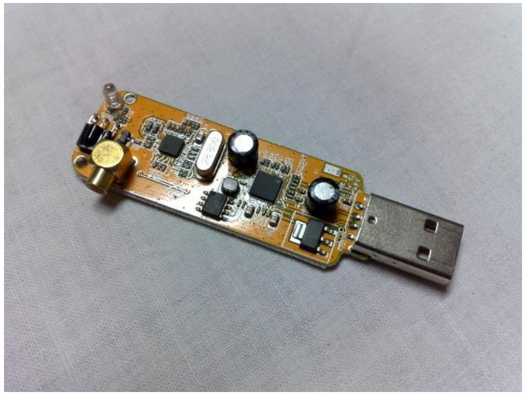
# Software Defined Radio Demonstration

Bruce Hart

November 16, 2015

# RTL-SDR Chipset

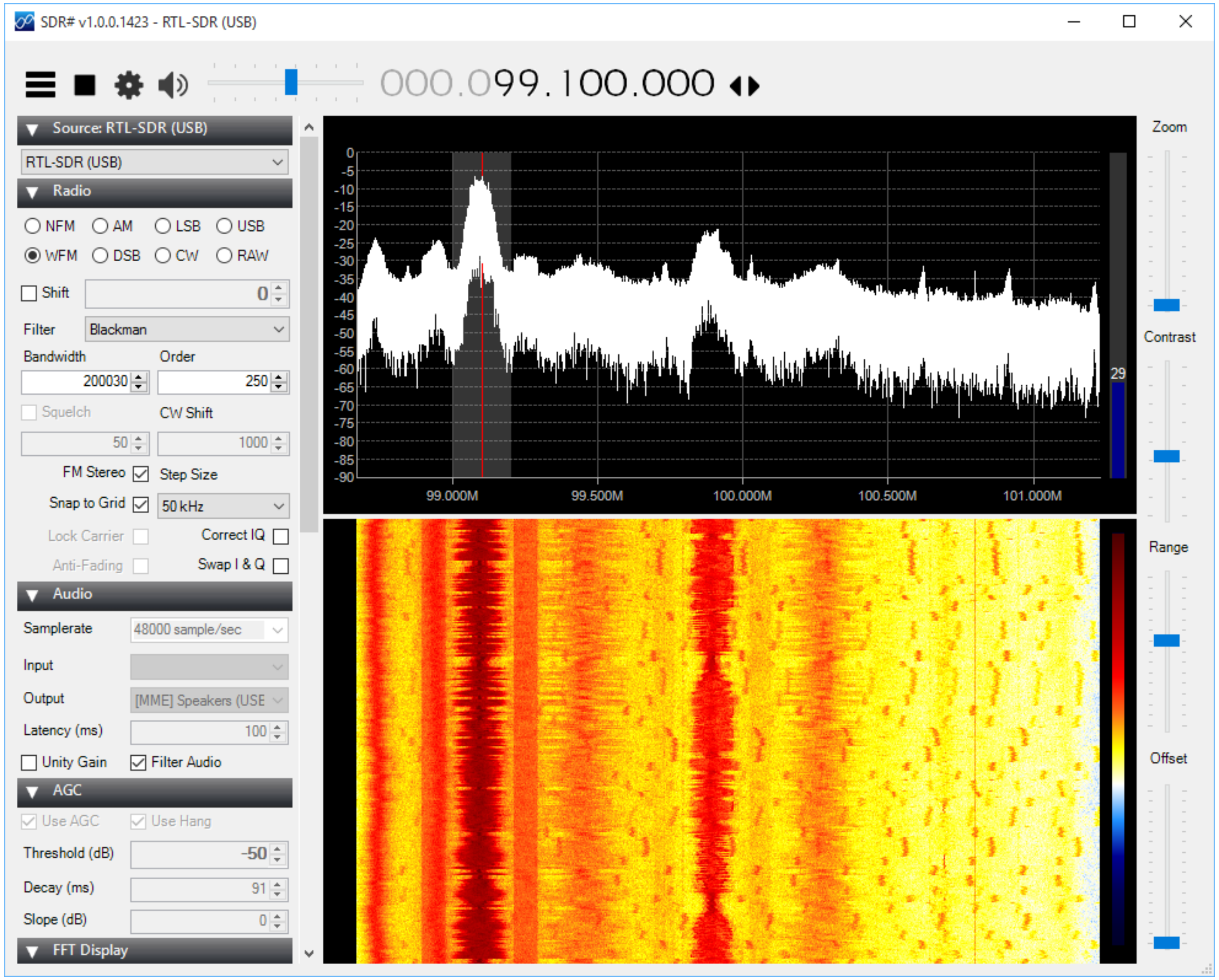
- Realtek RTL-2832U controller and tuner
- Originally created as a low cost DVB-T receiver in a USB device
- Discovered that the chipset can be used as a wideband SDR receiver
- Available for less than $20
- Tuning range from 24 MHz to 1.7 GHz
    - Lower tuning ranges possible with hardware HW modification
- Instantaneous bandwidth of up to 3.2 MHz



# GNURadio Background

- Open source software development kit for software defined radio and digital signal processing
- Operations are contained into blocks that are connected together to form a system flow.
- GNURadio Companion provides a graphical user interface to connect blocks and plot data
- Wide variety of blocks come preinstalled:
  - Signal processing – filtering, gain adjustment, demod
  - Data sources – generated, RTL-SDR, USRP
  - Data output – GUI plot, binary file, Tx device, network
- Users can also create custom-programmed blocks

# SDRSharp for Windows



SDR# v1.0.0.1423 - RTL-SDR (USB)

000.099.100.000

Source: RTL-SDR (USB)

RTL-SDR (USB)

Radio

○ NFM ○ AM ○ LSB ○ USB
● WFM ○ DSB ○ CW ○ RAW

☐ Shift 0

Filter: Blackman

Bandwidth: 200030 Order: 250

☐ Squelch 50 CW Shift: 1000

FM Stereo ☑ Step Size: 50 kHz

Snap to Grid ☑

Lock Carrier ☐ Correct IQ ☐

Anti-Fading ☐ Swap I & Q ☐

Audio

Samplerate: 48000 sample/sec

Input:

Output: [MME] Speakers (USE

Latency (ms): 100

☐ Unity Gain ☑ Filter Audio

AGC

☑ Use AGC ☑ Use Hang

Threshold (dB): -50

Decay (ms): 91

Slope (dB): 0

FFT Display

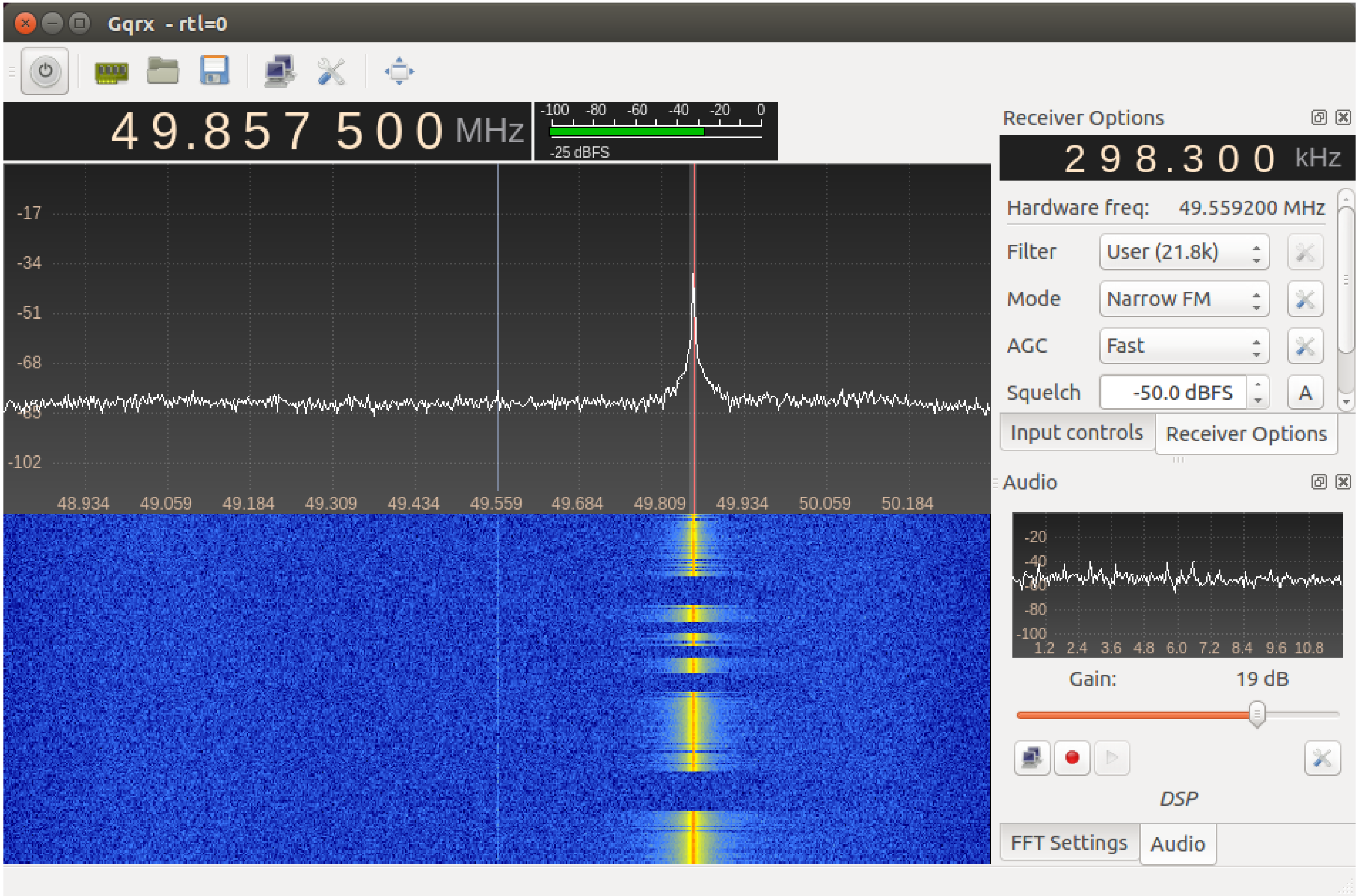
# Setting Up the RTL-SDR on Linux

- Setup is simple in Ubuntu: package manager does most of the work and installs the latest versions
- Set up Ubuntu on separate hard drive partition
- Install GNURadio:
  - `sudo apt-get install gnuradio-dev`
- Install GQRX:
  - `sudo apt-get install gqrx-sdr`
- Install Qt Creator and build tools:
  - `sudo apt-get install build-essential`
  - `wget http://download.qt.io/official_releases/qt/5.0/5.0.2/qt-linux-opensource-5.0.2-x86_64-offline.run`
  - `chmod +x qt-linux-opensource-5.0.2-x86_64-offline.run`
  - `./qt-linux-opensource-5.0.2-x86_64-offline.run`

# Demonstration Background

- Goal: Demonstrate decoding an RF protocol
- More interesting than broadcast AM/FM station
- Something original that does not replicate demonstration already online
- Selected an RC car controller:
  - 49 MHz center frequency
  - RF protocol is unknown – need to use GNU radio to analyze and build decoder
  - 8 different commands : up, down, left, right, up+left, up+right, down+left, down+right

# Examining the Signal in GQRX



Gqrx - rtl=0

49.857 500 MHz

-25 dBFS

Receiver Options

298.300 kHz

Hardware freq: 49.559200 MHz

Filter: User (21.8k)

Mode: Narrow FM

AGC: Fast

Squelch: -50.0 dBFS

Audio

Gain: 19 dB

# RC Signal Analysis

- Plotted data -- hard to see in real time
- Used GNURadio to write samples to a file
- Discovered the signal is pulsed CW
  - Also called on-off keying (OOK) or binary keying
- Simple signal – easy to generate with cheap electronics
- Single frequency – avoids intermod products on cheap amplifier

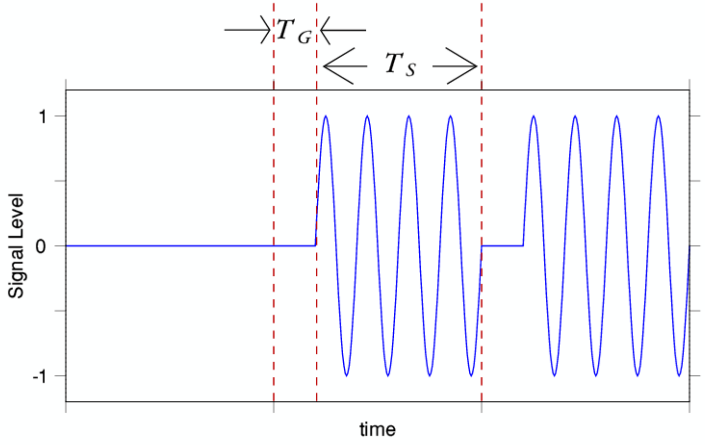
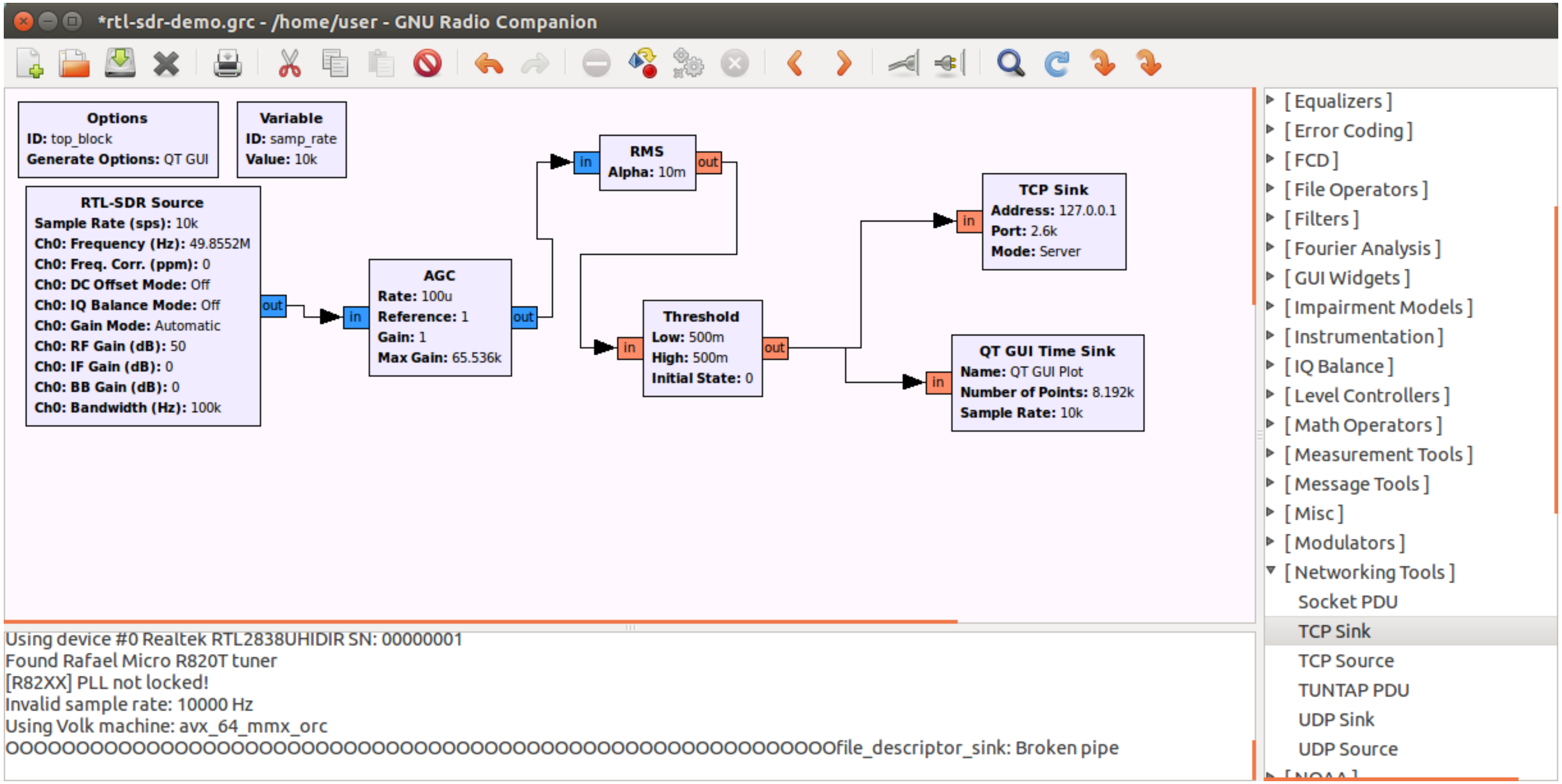


Fig 19.2 Symbol and Guard durations.

# Signal Decoder

- Demodulation is pretty simple: the goal is to get the envelope of the function and then decode the bit sequence
- Process: filter on the center frequency, take the root mean squared of the signal
- Signal can only take on two values: on or off
- Use thresholding to convert to on-off values
- How to decode the bit sequence?
  - Captured all commands to data files
  - Wrote a C++ program to analyze the data

# Block Diagram of Receiver



**Options**
**ID:** top_block
**Generate Options:** QT GUI

**Variable**
**ID:** samp_rate
**Value:** 10k

**RTL-SDR Source**
**Sample Rate (sps):** 10k
**Ch0: Frequency (Hz):** 49.8552M
**Ch0: Freq. Corr. (ppm):** 0
**Ch0: DC Offset Mode:** Off
**Ch0: IQ Balance Mode:** Off
**Ch0: Gain Mode:** Automatic
**Ch0: RF Gain (dB):** 50
**Ch0: IF Gain (dB):** 0
**Ch0: BB Gain (dB):** 0
**Ch0: Bandwidth (Hz):** 100k

**AGC**
**Rate:** 100u
**Reference:** 1
**Gain:** 1
**Max Gain:** 65.536k

**RMS**
**Alpha:** 10m

**Threshold**
**Low:** 500m
**High:** 500m
**Initial State:** 0

**TCP Sink**
**Address:** 127.0.0.1
**Port:** 2.6k
**Mode:** Server

**QT GUI Time Sink**
**Name:** QT GUI Plot
**Number of Points:** 8.192k
**Sample Rate:** 10k

```
Using device #0 Realtek RTL2838UHIDIR SN: 00000001
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
Invalid sample rate: 10000 Hz
Using Volk machine: avx_64_mmx_orc
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOfile_descriptor_sink: Broken pipe
```

# Bit Sequence

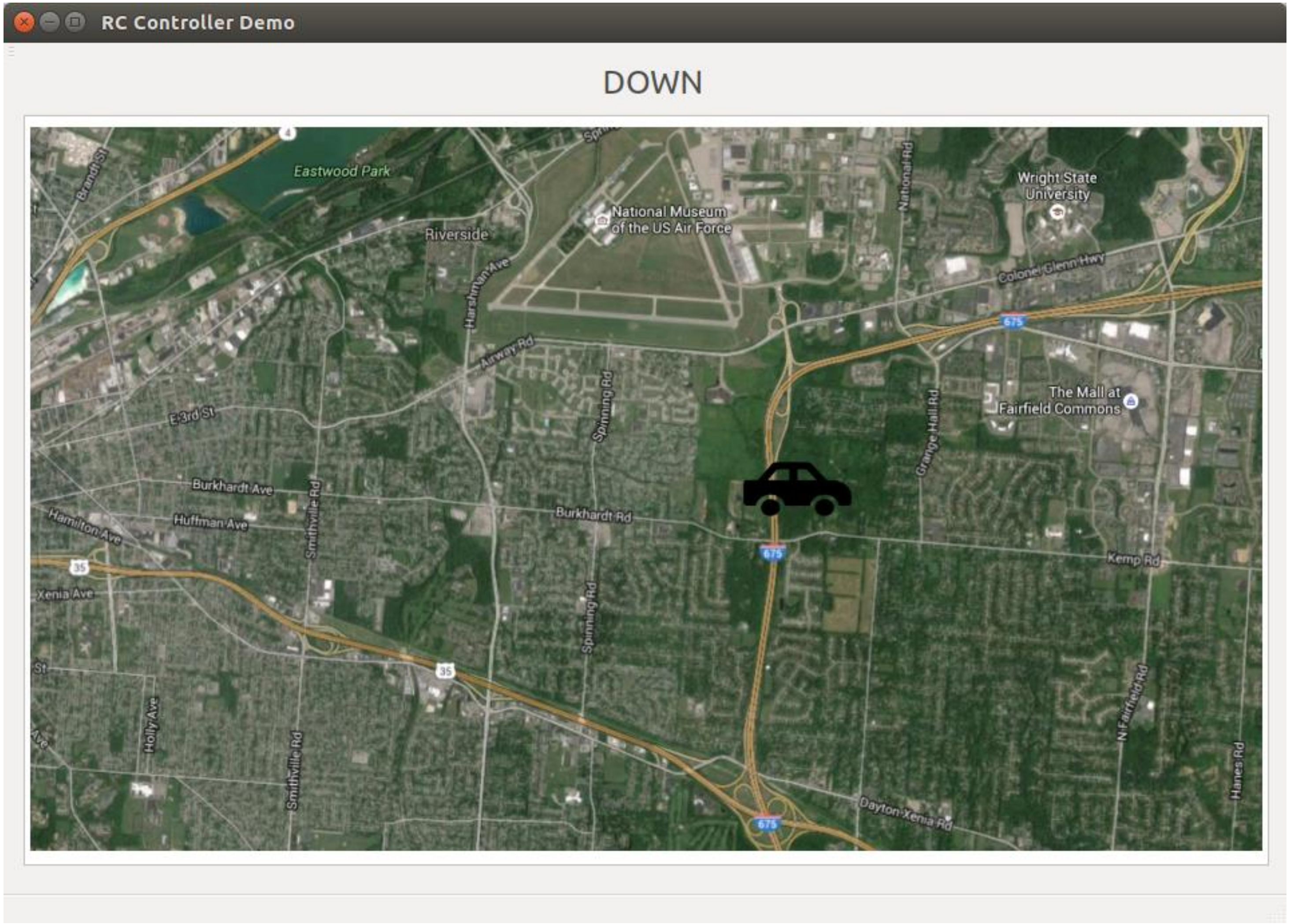
- Controller sends two types of pulses: short pulse (3 ms) and long pulse (8.25 ms)
- Used a 5 ms cutoff to separate short and long pulses
- Command always starts with four long pulses followed by a series of short pulses:

|  |  |  |  |
|---|---|---|---|
| UP | 10 pulses | DOWN | 40 pulses |
| LEFT | 58 pulses | RIGHT | 64 pulses |
| DOWN + LEFT | 52 pulses | UP + LEFT | 28 pulses |
| DOWN + RIGHT | 46 pulses | UP + RIGHT | 34 pulses |

# Bit Decoder

- GNURadio feeds output over a local TCP socket to a custom Qt application
- Application uses a simple state machine to determine command:
  - Short pulses increment a counter
  - Long pulses reset the counter, execute commands
- When command is decoded, car icon is moved around on the screen
- Qt signals and slots approach makes it easy to buffer up TCP data and process periodically

# Qt Software Demonstration



DOWN

# Project Summary

- Fun project – learned more about GNURadio’s capabilities
- Biggest challenges:
  - Figuring out the RF protocol
  - Windows driver issue
  - Using TCP vs. UDP
- Possibilities with more time:
  - Make a two player game with RC controllers at different frequencies
  - Code improvements: lower sampling rate/data transfer rate, custom GNURadio blocks, direct interface with device via USB driver
- Questions?