Reverse Engineering A 433MHz Remote Control

Max-Felix Müller

2021
May

Contents

List of Figures

List of Tables

Abstract

1 Introduction

Reverse engineering something yields interesting inights in the technology and the thought process behind products. In the time of rising popularity of IoT applications and products, the radio frequency world is coming to the masses.

There is also the possibility of finding weaknesses or loop holes in the security.

2 Target

The target for reverse engineering the wireless protocol is the 433MHz remote control known from the project "Replay Attack On A Remote Controlled Relay". Instead of replaying, in this project the goal is to find out as much as possible about the protocol the remote uses.

It is known already, from the last project, that the remote uses a rather simple ASK (amplitude shift keying) or more specific OOK (on off keying). Some guesses have been made about the bits in the transmitted data, but this time more clarification is required.

3 Universal Radio Hacker

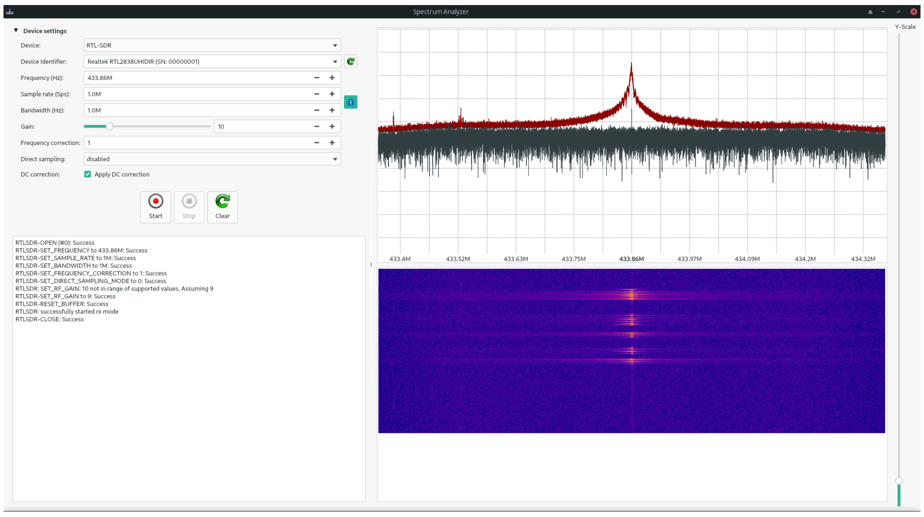
The software used in this project is the Universal Radio Hacker, sometimes called URH for short. It is free software, available for example on Linux.

This software can be used in combination with a SDR (software defined radio) like the RTL-SDR, to record, reverse engineer and even generate new data. When used in conjunction with a sdr that has transmit capabilities, such as the Hack-RF or Pluto SDR, URH can also transmit the generated signal.

3.1 Scanning the Spectrum

URH has a spectrum analyzer build in. It can be used to get an overview of the frequency spectrum to find the targeted signal in the first place.

For the remote control this step can be skipped, since the operating frequency is already known to be 433.86MHz. For demonstration purposes here is a screenshot.



Device:	RTL-SDR
Device Identifier:	Realtek RTL2838UHIDIR (SN: 00000001)
Frequency (Hz):	433.86M
Sample rate (Sps):	1.0M
Bandwidth (Hz):	1.0M
Gain:	10
Frequency correction:	1
Direct sampling:	disabled
DC correction:	✓ Apply DC correction

```
RTLSDR-OPEN (#0): Success
RTLSDR-SET_FREQUENCY to 433.86M: Success
RTLSDR-SET_SAMPLE_RATE to 1M: Success
RTLSDR-SET_BANDWIDTH to 1M: Success
RTLSDR-SET_FREQUENCY_CORRECTION to 1: Success
RTLSDR-SET_DIRECT_SAMPLING_MODE to 0: Success
RTLSDR: SET_RF_GAIN: 10 not in range of supported values. Assuming 9
RTLSDR-SET_RF_GAIN to 9: Success
RTLSDR-RESET_BUFFER: Success
RTLSDR: successfully started rx mode
RTLSDR-CLOSE: Success
```

Figure 1: Spectrum of the Remote Control

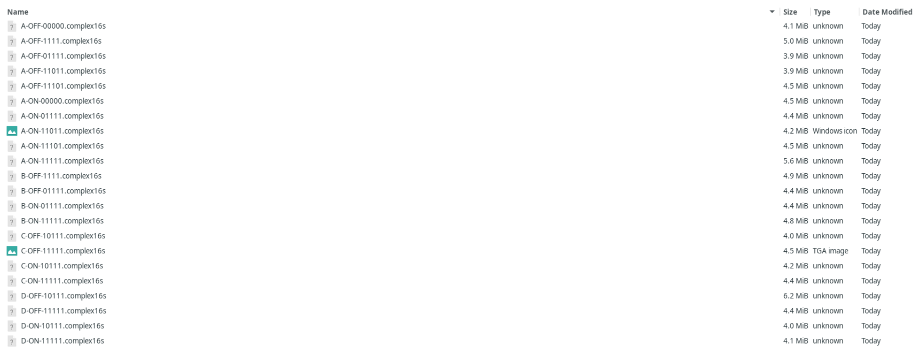
3.2 Recording the Signal

The recording window looks very similar to the spectrum analysis window. It is possible to set the center frequency, bandwidth, sampling rate, offset and gain. The options depend on the SDR used for recording.

Frequency	433.86MHz
Sample Rate	1Msps
Bandwidth	1MHz
Gain	10dB

Table 1: Settings for Recording the Signals

With this settins, a lot of different combinations have been tested. Each button A to D was pressed, held for a short amount of time and then released twice. This is done for the ON and the OFF button. On the back of the remote there are five dip switches that can be used to select a channel. Tested were the combinations (switch 1 to 5, 1 means switch is in the ON position): 11111, 01111, 10111, 11011, 11101, 00000. The rest of the possible settings will result from the reverse engineering and can be used to prove that the protocol was understood.



Name	Size	Type	Date Modified
A-OFF-00000.complex16s	4.1 MiB	unknown	Today
A-OFF-1111.complex16s	5.0 MiB	unknown	Today
A-OFF-01111.complex16s	3.9 MiB	unknown	Today
A-OFF-11011.complex16s	3.9 MiB	unknown	Today
A-OFF-11101.complex16s	4.5 MiB	unknown	Today
A-ON-00000.complex16s	4.5 MiB	unknown	Today
A-ON-01111.complex16s	4.4 MiB	unknown	Today
A-ON-11011.complex16s	4.2 MiB	Windows icon	Today
A-ON-11101.complex16s	4.5 MiB	unknown	Today
A-ON-11111.complex16s	5.6 MiB	unknown	Today
B-OFF-1111.complex16s	4.9 MiB	unknown	Today
B-OFF-01111.complex16s	4.4 MiB	unknown	Today
B-ON-01111.complex16s	4.4 MiB	unknown	Today
B-ON-11111.complex16s	4.8 MiB	unknown	Today
C-OFF-10111.complex16s	4.0 MiB	unknown	Today
C-OFF-11111.complex16s	4.5 MiB	TGA image	Today
C-ON-10111.complex16s	4.2 MiB	unknown	Today
C-ON-11111.complex16s	4.4 MiB	unknown	Today
D-OFF-10111.complex16s	6.2 MiB	unknown	Today
D-OFF-11111.complex16s	4.4 MiB	unknown	Today
D-ON-10111.complex16s	4.0 MiB	unknown	Today
D-ON-11111.complex16s	4.1 MiB	unknown	Today

Figure 2: List of the Captured Signals

If not happened yet, a project in URH should be created. Otherwise furhter progress will get lost when the program is closed.

4 Interpretation

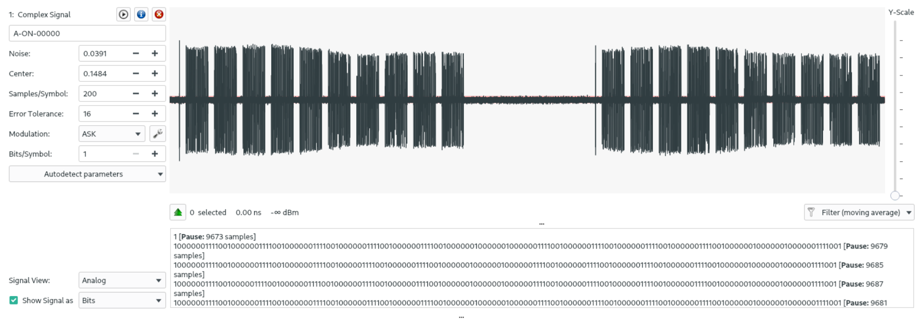
4.1 Loading the Signals

Once a project was created, the recorded signal files can be imported using "open". If there are a lot of files, the whole folder can be imported at once.

4.2 Refining the Signals

To make the signals easier to work with, they were all captured at the same sample rate.

Next the long empty sections in the beginning and end of a recording can be removed. This step has to be done only once per file. The file can then be overwritten.



1: Complex Signal

A-ON-00000

Noise: 0.0391

Center: 0.1484

Samples/Symbol: 200

Error Tolerance: 16

Modulation: ASK

Bits/Symbol: 1

Signal View: Analog

Show Signal as Bits

0 selected 0.00 ns -∞ dBm

```
1 [Pause: 9673 samples]
1000000111100100000011110010000001111001000000111100100000011110010000001000000100000011110010000001111001000000111100100000011110010000001111001000000100000010000001111001 [Pause: 9679 samples]
1000000111100100000011110010000001111001000000111100100000011110010000001000000100000011110010000001111001000000111100100000011110010000001111001000000100000010000001111001 [Pause: 9685 samples]
1000000111100100000011110010000001111001000000111100100000011110010000001000000100000011110010000001111001000000111100100000011110010000001111001000000100000010000001111001 [Pause: 9687 samples]
1000000111100100000011110010000001111001000000111100100000011110010000001000000100000011110010000001111001000000111100100000011110010000001111001000000100000010000001111001 [Pause: 9681
```

Figure 3: Removing Start and Stop Noise of the Captured Signals

4.3 Demodulation

As previously discovered, the signal is OOK ASK modulated. There is a small caveat in that an on or off is not 1 and 0, but the time in a given interval that the signal is on determines wether the bit is a 1 or a 0.

In the modulation settings it is possible to select the lengths of pauses. If the pauses are defined improperly, the interpreted signal will be very long and does not show clear individual messages. In this case the pause is the duration of 5 symbols.

With the URH it is possible to select a piece of the signal and have the samples counted. This way, the length of a bit is determined.

In this case, one symbol equals one bit and has a length of about 1350 samples. At 1Msps that is 1.35ms which in turn is 740 bits per second.

The tolerance can be used to accomodate for small deviations in the timing. A tolerance of 20 samples was used, but lower tolerances already would have worked.

Switching the signal view to the spectrum, a bandpass filter can be introduced by dragging over the waterfall image. The bandpass filter will reduce noise from nearby channels. After the noise has been reduced, the noise level can be set for the time domain signal.

4.4 Pre Analysis

Before moving to the actual analysis, the URH already shows the decoded signal in the interpretation tab. There is one interesting thing to notice in this particular case.

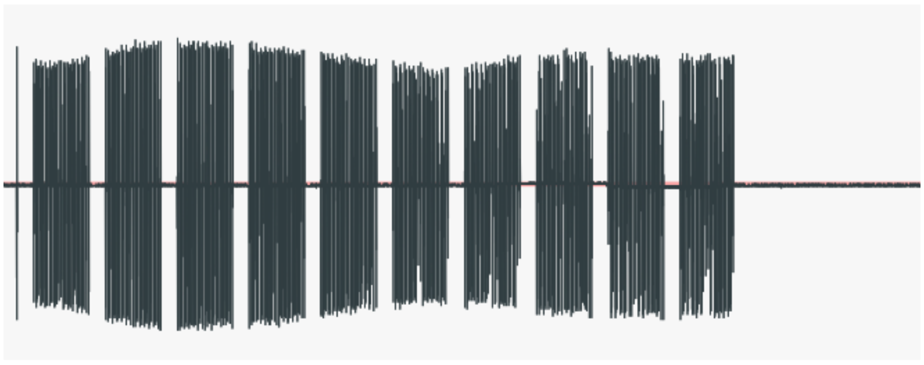
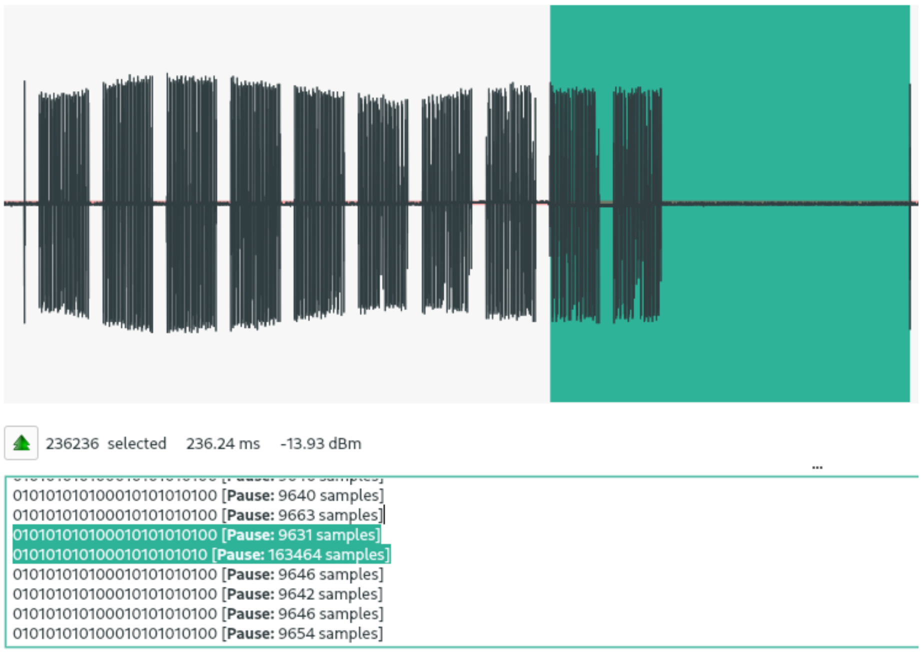


Figure 4: The Signal of One Button Press

There is a signal repeating for the whole duration the button is pressed. In the interpreted data, it can be seen that the last repetition is shorter than the previous ones. The analysis that was done manually in the last project concluded that the remote sends an additional bit to signal the receiver that the code will be repeated.

However, it can be noticed that there is a small pulse before the large signal junks start. This would mean, that the additional bit actually is in front of the "pause". The first bit in this case is ignored, as the interpretation uses the longer pauses as start and stop of one burst.



```
236236  selected    236.24 ms    -13.93 dBm

010101010100010101010100 [Pause: 9640 samples]
010101010100010101010100 [Pause: 9663 samples]
010101010100010101010100 [Pause: 9631 samples]
01010101010001010101010 [Pause: 163464 samples]
010101010100010101010100 [Pause: 9646 samples]
010101010100010101010100 [Pause: 9642 samples]
010101010100010101010100 [Pause: 9646 samples]
010101010100010101010100 [Pause: 9654 samples]
```

Figure 5: The Last Code sent is Shorter

The selected part of the signal is also selected in the interpreted view.

This means, that actually the first bit is sent, then a pause, and then the rest of the code. To confirm this, a single pulse would have to be captured.

5 Analysis

The analysis is all about finding meaning in the data. Patterns can help greatly with this.

One thing to keep in mind for this particular project is that the last bit might in reality be the first and then a pause is included in each packet.

In this document the switches will have the "classes" A, B, C or D and the "channel" is the combination of the dip switches.

5.1 On or Off

There are two different buttons on the remote. One for on and one for off. On a simple protocol, which is expected in this case, there should be one bit representing the on or off command.

Using the difference between two singals that should be the same (same channel and same class) except for the on or off state, the bits that change can be found. In this case there are two bits that change: bit 22 and bit 24. Note once again, that bit 24 might in fact be bit 0.

In this case, bit 22 is on if the switch shall turn off. Bit 24 is reversed (or correct) in that if it is 1, the switch is on also.

5.2 Class

Comparing signals of channel 11111 and all off buttons, the difference in class can be observed. For the direct comparison, the OFF button is used for each class. In this case, it seems like the bits 12, 14, 16, and 18 are responsible for this setting.

A	0111
B	1011
C	1101
D	1110

Table 2: A, B, C or D

As can be seen, the position of the 0 in these 4 bits is the class that is selected. This holds true even for the ON signal.

5.3 Channel

The last setting that can be made is the channel, by setting the dip switches on the back of the remote. Again, the difference was investigated on signals that were otherwise the same: A - OFF.

The bits that changed are: 2, 4, 6, 8, 10. Interestingly enough, once again the number corresponds to the number of dip switches: 5.

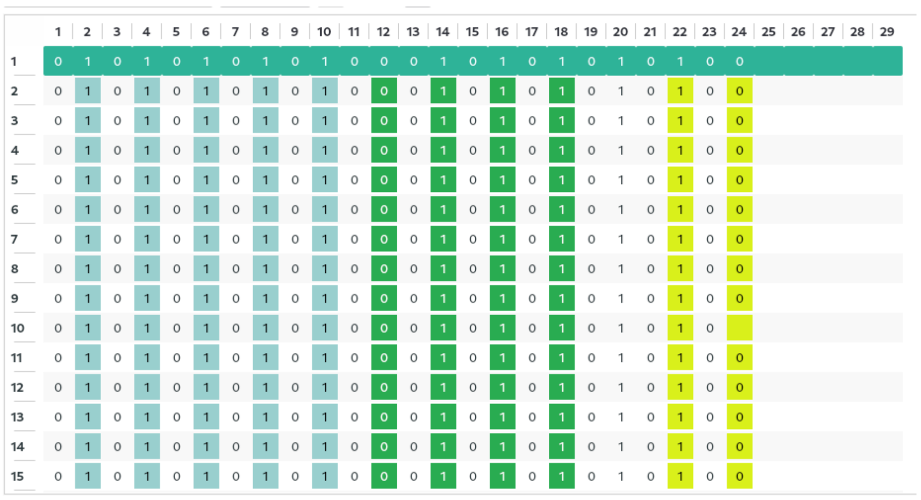
In this case not all possible combinations of dip switch configurations were tested, but it should be enough to understand the protocol.

00000	11111
01111	10000
11011	00100
11101	00010
11111	00000

Table 3: A, B, C or D

In this case, the bits are the inversion of the switch setting.

5.4 Combining the Parts

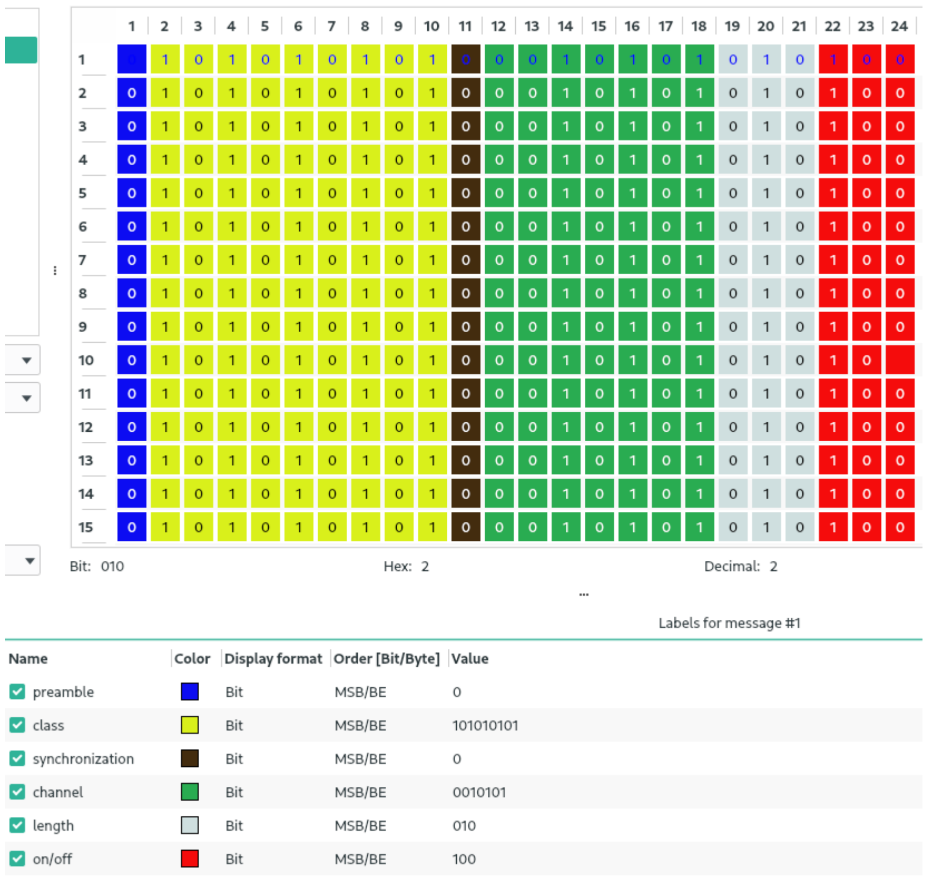


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
2	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
3	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
4	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
5	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
6	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
7	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
8	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
9	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
10	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0						
11	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
12	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
13	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
14	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					
15	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0					

Figure 6: Labeled Messages

Note, that there are unmarked bits. Bit 20 is always 1 while all the odd bits are always 0. This holds true for every captured signal.

URH does not really support labels that have "0" bits in between. As a workaround, the 0s have been added to the labels. The 0 in front was called preamble, because it comes first, the 0 between the class and the channel is called synchronization and the 010 between the channel and the on/off data is the length now, since there are 2 bits representing the state. The naming is somewhat arbitrary, since these bits do not hold any information.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
2	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
3	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
4	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
5	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
6	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
7	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
8	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
9	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
10	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	
11	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
12	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
13	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
14	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0
15	0	1	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0

Bit: 010 Hex: 2 Decimal: 2

Labels for message #1

Name	Color	Display format	Order [Bit/Byte]	Value
✓ preamble	🟦	Bit	MSB/BE	0
✓ class	🟨	Bit	MSB/BE	101010101
✓ synchronization	🟫	Bit	MSB/BE	0
✓ channel	🟩	Bit	MSB/BE	0010101
✓ length	⬜	Bit	MSB/BE	010
✓ on/off	🟥	Bit	MSB/BE	100

Figure 7: Completely Labeled Messages

Since this is a very simple protocol, the data is not encoded any further.

6 Testing the Protocol

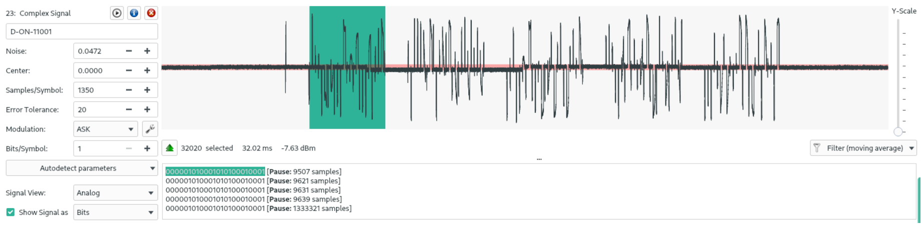
Since not all possible combinations were recorded previously, now a new combination can be chosen. It should be possible to guess the resulting data, based on the protocol that was analyzed.

As a demonstration, two signals will be used.

6.1 D - ON - 11001

The result for D - ON - 11001 should, according to the protocol, be 0 000010100 0 1010100 010 001. For easier reading, sections have been split according to the labels.

Here is the captured signal with the interpreted data in the bottom of the screenshot.



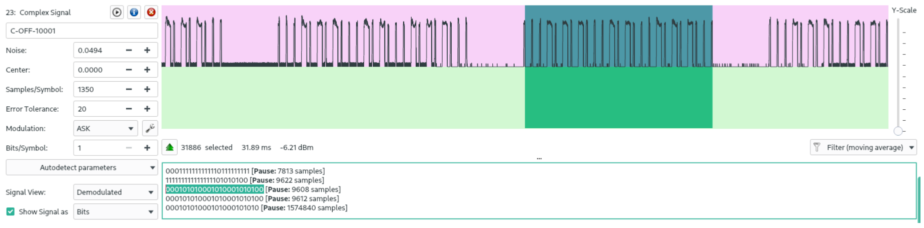
```
D-ON-11001
Noise: 0.0472
Center: 0.0000
Samples/Symbol: 1350
Error Tolerance: 20
Modulation: ASK
Bits/Symbol: 1
Autodetect parameters
Signal View: Analog
Show Signal as: Bits

32020  selected    32.02 ms    -7.63 dBm

0000010100010101000010001 [Pause: 9507 samples]
0000010100010101000010001 [Pause: 9621 samples]
0000010100010101000010001 [Pause: 9631 samples]
0000010100010101000010001 [Pause: 9639 samples]
0000010100010101000010001 [Pause: 1333321 samples]
```

Figure 8: D - ON - 11001

The protocol holds true.

6.2 C - OFF - 10001

The result for C - OFF - 10001 should, according to the protocol, be 0 001010100 0 1010001 010 100.



```
C-OFF-10001
Noise: 0.0494
Center: 0.0000
Samples/Symbol: 1350
Error Tolerance: 20
Modulation: ASK
Bits/Symbol: 1
Autodetect parameters
Signal View: Demodulated
Show Signal as: Bits

31886  selected    31.89 ms    -6.21 dBm

0001111111111110111111111 [Pause: 7813 samples]
1111111111111111101010100 [Pause: 9622 samples]
0001010100010100001010100 [Pause: 9608 samples]
0001010100010100001010100 [Pause: 9612 samples]
0001010100010100010101010 [Pause: 1574840 samples]
```

Figure 9: C - OFF - 10001

The protocol holds true.

7 Generating new Commands

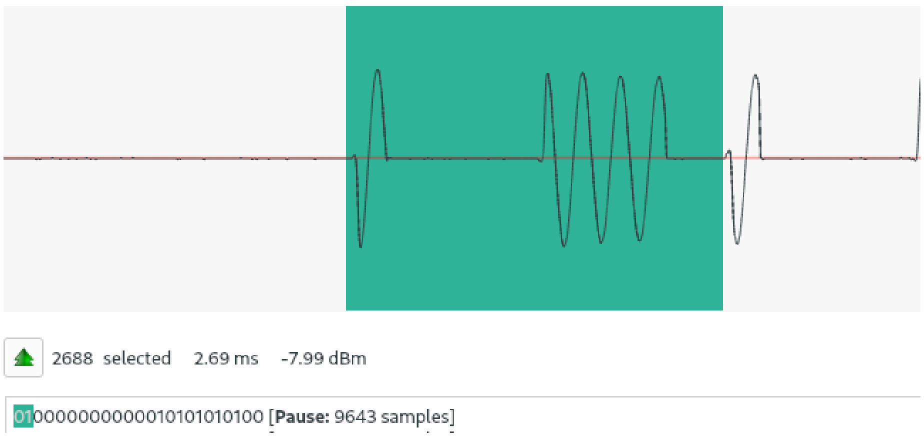
The URH could be used to continue generating and simulating new commands. Instead, because the protocol is quite simple, it shall be implemented as a GNU Radio Companion flowgraph.

In a first step, the signal can be reproduced as analyzed, meaning with bit 24 in the last position. Either the text ”000101010001010001010100” (from the protocol test) shall be used as an input, or a 3 byte string (3 byte are 24 bit).

Then, as a test, the signal shall be reproduced in such a way, that the 24th bit is placed in front of the first bit with the pause. This signal shall be sent only once, as if the button was pressed for a very short amount of time. It is unclear if this will work, since in practice it is not possible to press the button for this short amount of time and the receiver may miss a single command.

7.1 What is a 0 and a 1

Before sending a command, note that previously it was discovered that a 0 and a 1 are not just on or off, but actually the duration of the pulse.



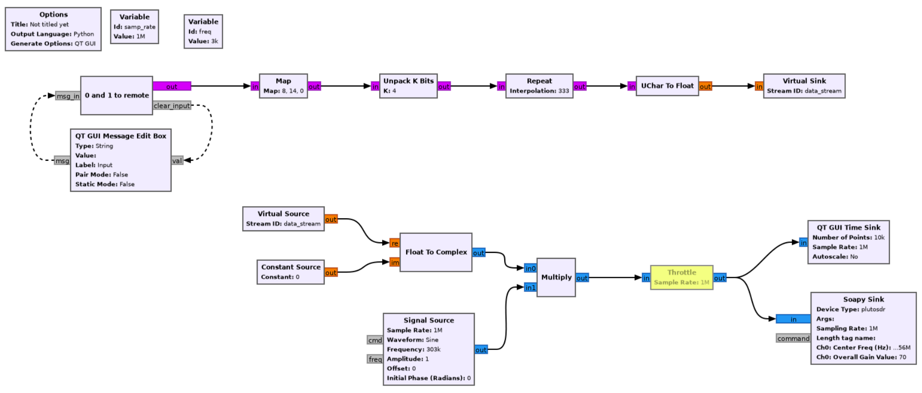
```
2688  selected    2.69 ms    -7.99 dBm

010000000000010101010100 [Pause: 9643 samples]
```

Figure 10: 0 and 1 as Pulse Duration

Upon closer inspection of this image, it becomes obvious that for a 0 the pulse is exactly one period of the carrier frequency and a 1 has 3 periods. Then there is a gap (carrier off time) to fill the symbol length to be 4 periods in total, so 3 missing periods for 0 and 1 missing for 1.

This is not exact, but the estimate should be good enough to recreate the signal.

7.2 Sending the Command via GRC


Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 1M

Variable
Id: freq
Value: 3k

0 and 1 to remote

QT GUI Message Edit Box
Type: String
Value:
Label: Input
Pair Mode: False
Static Mode: False

Map
Map: 8, 14, 0

Unpack K Bits
K: 4

Repeat
Interpolation: 333

UChar To Float

Virtual Sink
Stream ID: data_stream

Virtual Source
Stream ID: data_stream

Constant Source
Constant: 0

Float To Complex

Multiply

Throttle
Sample Rate: 1M

Signal Source
Sample Rate: 1M
Waveform: Sine
Frequency: 303k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

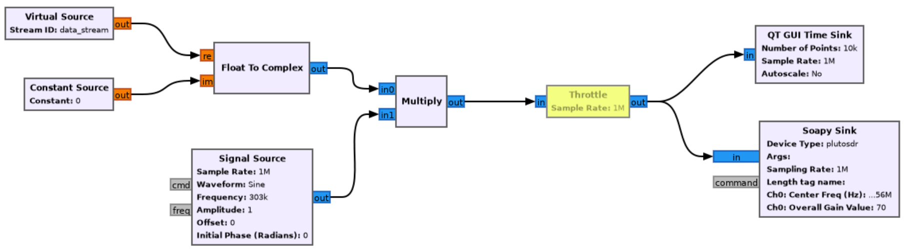
QT GUI Time Sink
Number of Points: 10k
Sample Rate: 1M
Autoscale: No

Soapy Sink
Device Type: plutosdr
Args:
Sampling Rate: 1M
Length tag name:
Ch0: Center Freq (Hz): ...56M
Ch0: Overall Gain Value: 70

Figure 11: The Flowgraph for Sending Commands via GRC

Explaining the gnuradio flowgraph works best from the back to the front, since the signal that has to be sent is already known in this case.

7.2.1 On Off Keying


Virtual Source
Stream ID: data_stream

Constant Source
Constant: 0

Float To Complex

Multiply

Throttle
Sample Rate: 1M

Signal Source
Sample Rate: 1M
Waveform: Sine
Frequency: 303k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

QT GUI Time Sink
Number of Points: 10k
Sample Rate: 1M
Autoscale: No

Soapy Sink
Device Type: plutosdr
Args:
Sampling Rate: 1M
Length tag name:
Ch0: Center Freq (Hz): ...56M
Ch0: Overall Gain Value: 70

Figure 12: On Off Keying in GRC

A signal source produces a cosine wave at the carrier frequency. In this case the frequency has been calculated from the number of samples within the original captured signal. The frequency is later offset by the SDR, when it is shifted by the output frequency. The signal source creates a 303 kHz signal which is then shifted by 433.56 MHz from the Soapy Sink block. The resulting output will be a signal at 433.863 MHz.

The signal from the signal source is interrupted by multiplying it with the data. A multiplication with 1 will result in the signal being active, while a multiplication with 0 turns the signal off. This way the on off keying is achieved.

The ones and zeros for the modulation are inside the "data_stream" virtual source block. Because they are floating point values, they are converted to a complex number by adding a constant imaginary part of 0.

7.2.2 Creating the Data Stream

The protocol for the remote control has three distinct values. The pause between the first bit and the rest of the code, 0 and 1. These values are represented in text format as " " (space) for the pause, "1" for 1 and "0" for 0. This way, a string can be used to define the command.

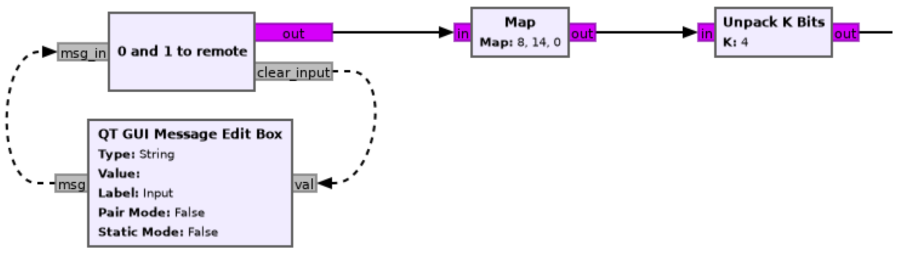


Figure 13: Inputting a Command in GRC

The string is entered in the flowgraph via the message edit block. When the input is confirmed, it is interpreted by a custom python module. Each individual character in the string is iterated over. The text characters for "0", "1" and " " are converted into the numbers 0, 1 and 2. Every other character is ignored.

With the map block, the three possibilities are mapped to three new values. Each of the new values is unpacked into a 4 bit value, with each bit representing one period of the signal. For example the 1 has three on periods followed by one off period. It is coded as 14, which is 0b1110 in binary.

The pause is coded as four 0 bits, but this does not cover the whole length. It was determined that the pause shall be stored in the string as seven consecutive spaces.

For completeness of documentation, the python code:

```
"""
0 and 1 to remote
"""

# epy_block_0.py
# created 10/17/2019

import numpy as np
from gnuradio import gr

import pmt

textboxValue = ""

class my_sync_block(gr.sync_block):
  """
  reads input from a message port
  if input is '0' or '1' it is converted to the signal for the remote
```

```
"""
def __init__(self):
  gr.sync_block.__init__(self,
    name = "0 and 1 to remote",
    in_sig = None,
    out_sig = [np.byte])
  self.message_port_register_in(pmt.intern('msg_in'))
  self.message_port_register_out(pmt.intern('clear_input'))
  self.set_msg_handler(pmt.intern('msg_in'), self.handle_msg)

def handle_msg(self, msg):
  global textboxValue

  textboxValue = pmt.symbol_to_string (msg)
  # print (textboxValue)

def work(self, input_items, output_items):
  global textboxValue

  # get length of string
  _len = len(textboxValue)
  if (_len > 0):
    # terminate with LF
    #textboxValue += "\n"
    #_len += 1
    # store elements in output array
    for x in range(_len):
      if textboxValue[x] == '0':
        output_items[0][x] = 0
      if textboxValue[x] == '1':
        output_items[0][x] = 1
      if textboxValue[x] == ' ':
        output_items[0][x] = 2
      #output_items[0][x] = ord(textboxValue[x])
    textboxValue = ""
    self.message_port_pub(pmt.intern('clear_input'), pmt.intern(''))
    return (_len)
  else:
    return (0)
```

7.2.3 Elongating the Bits

Each bit only represents one sample. This would not satisfy the targeted on and off times. Instead, based on the sample rate and the signal frequency, each bit is repeated a certain number of times.

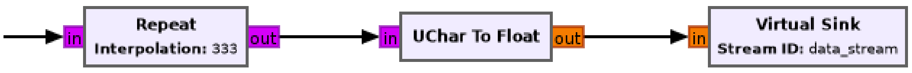


Figure 14: Elongate the Bits

7.3 Testing the Flowgraph

To test the flowgraph a remote controlled outlet was used. It was set to class D on channel 11111. The string command for turning the relay on is "0 0000000000001010100010001" (there are 7 spaces between the first bit and the rest of the command).

The command had to be sent at least three times in a row for the relay to recognize it. That is no problem for the real world, since the button can not be pressed any faster. It was not possible to switch the relay by sending the command only once.

8 Links

Documents and ressources used:

This tutorial proved very helpful to get started
https://hakin9.org/universal-radio-hacker-investigate-wireless-protocols-like-a-boss/