Assignment — RF Remote Signal Analysis Decoding Challenge

Digital Communications Systems - Advanced Telecommunication Technologies

Student 1:	**Student 2:**
Name: soheib abdessamad miloudi	Name: ahmed fekkai

Date: January 29, 2026

Contents

1 Introduction

This technical report documents the complete analysis of a garage door keyfob RF signal using software-defined radio (SDR) techniques. The objective was to analyze a prerecorded RF transmission from a simple ISM-band remote, identify its modulation scheme, recover the baseband signal, determine timing parameters, and extract the transmitted bit sequence.

The analysis follows a practical approach using GNU Radio Companion for signal processing and Python for detailed analysis. All work was conducted in an educational context using prerecorded signals to ensure compliance with ethical and legal guidelines regarding RF signal interception.

2 Equipment and Software

2.1 Hardware

- USRP Software Defined Radio (used for original signal capture)
- Prerecorded IQ samples of garage door keyfob transmission (ditec)

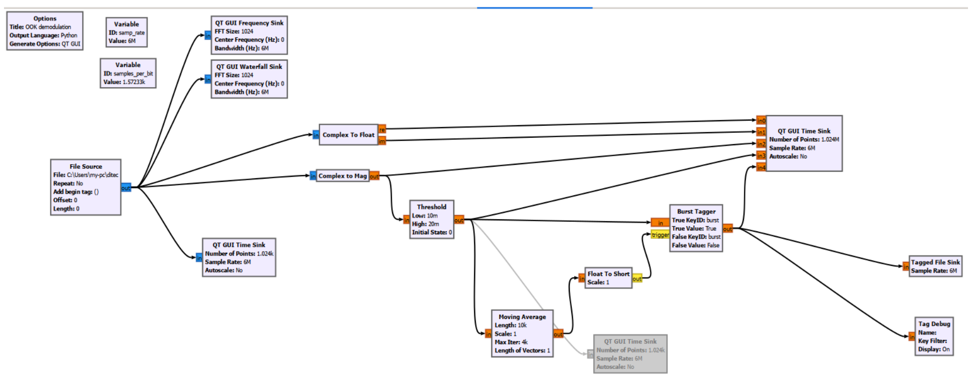
2.2 Software

- GNU Radio Companion 3.10
- Python 3.13.9+ with libraries: NumPy,glob, Matplotlib

3 Methodology

3.1 Signal Acquisition and Initial Inspection

The analysis began with a prerecorded IQ file containing multiple transmissions from a garage door keyfob. The initial step involved loading the file into GNU Radio for spectral and temporal analysis.



Options
Title: OOK demodulation
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 6M

Variable
ID: samples_per_bit
Value: 1.57233k

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

QT GUI Waterfall Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

File Source
File: C:\Users\my-pc\ditec
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

Complex To Float

Complex to Mag

QT GUI Time Sink
Number of Points: 1.024M
Sample Rate: 6M
Autoscale: No

Threshold
Low: 10m
High: 20m
Initial State: 0

Burst Tagger
True KeyID: burst
True Value: True
False KeyID: burst
False Value: False

Float To Short
Scale: 1

Moving Average
Length: 10k
Scale: 1
Max Iter: 4k
Length of Vectors: 1

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

Tagged File Sink
Sample Rate: 6M

Tag Debug
Name:
Key Filter:
Display: On

Figure 1: GNU Radio flowgraph for signal analysis

3.2 Frequency and power Analysis

The signal's spectral characteristics were examined using a QT GUI Frequency Sink. The center frequency was confirmed to be in the 433.920 MHZ ±0.05 , which is typical for garage door.

as we can see the peak appearing around the center and maximum power appearing near the center (0.05 MHz to the right) using a QT GUI waterfall sink

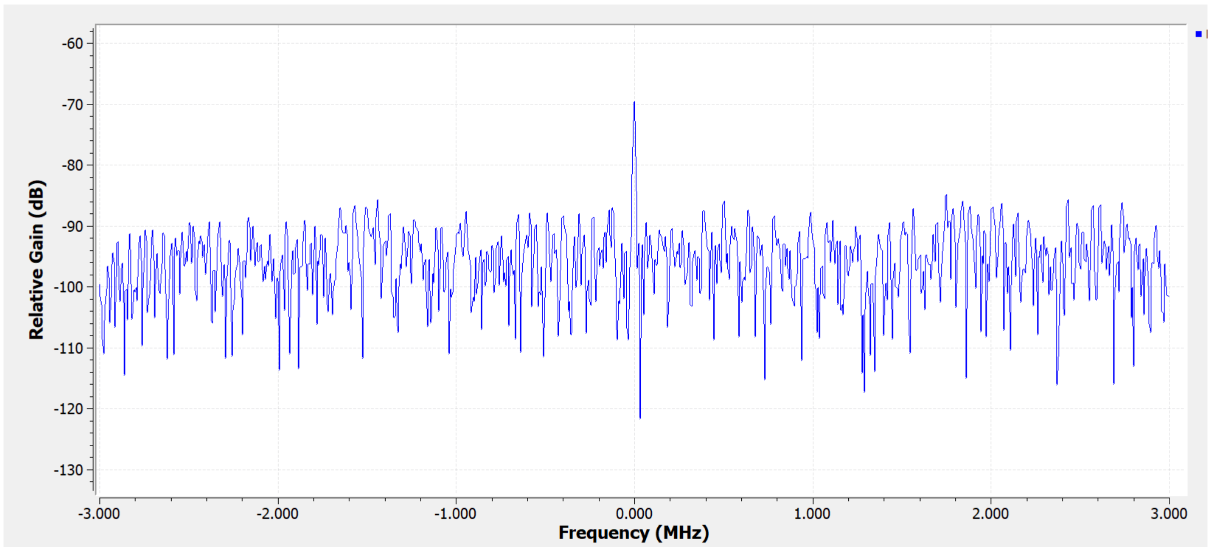


Figure 2: Spectral analysis showing signal at 433.92 MHz

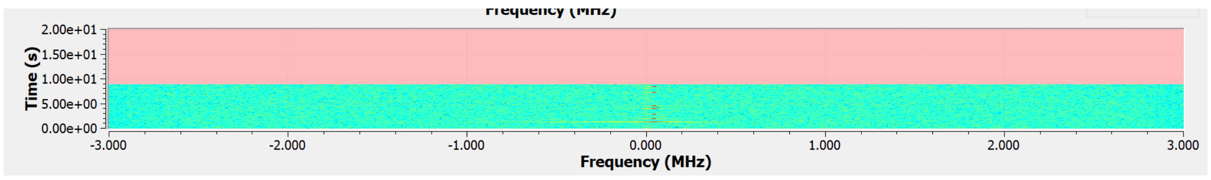


Figure 3: power spectrum shows energy concentration around the carrier

3.3 Modulation Identification

The time-domain waveform revealed clear On-Off Keying (OOK) modulation characteristics. The signal envelope showed distinct high and low states corresponding to binary 1 and 0.

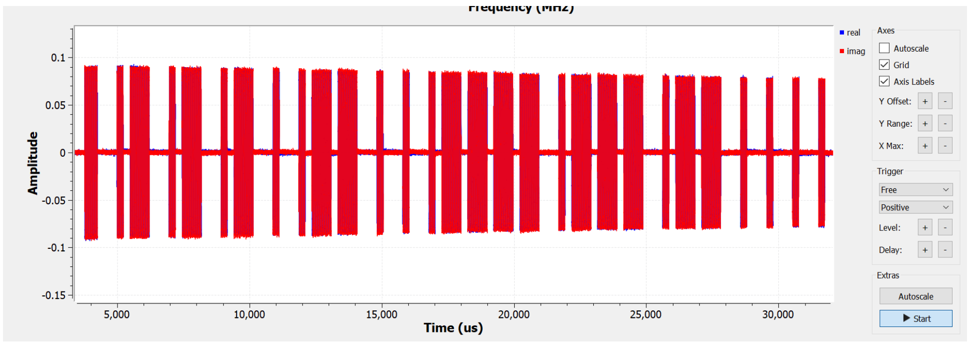


Figure 4: OOK modulation visible in time-domain signal

3.4 Bit Rate Determination

The symbol rate was calculated by measuring the duration of individual bits in the time-domain display QT GUI time sink :

$$T_{bit} \approx 262\mu s \quad \Rightarrow \quad R_b = \frac{1}{T_{bit}} = 3816 \text{ bps}$$

3.5 Frame Structure Analysis

we then extract the bit frame structure using OOK decoding , we can create a simple decoder in GNURadio obtaining the magnitud the the signal received.

We will modify the flowgraph by adding a Complex to Mag block, and also, since the GUI TIME Sink will work this time with float32 data, we can add also a Complex to Float to convert the IQ data from the source to a complex float32 data.

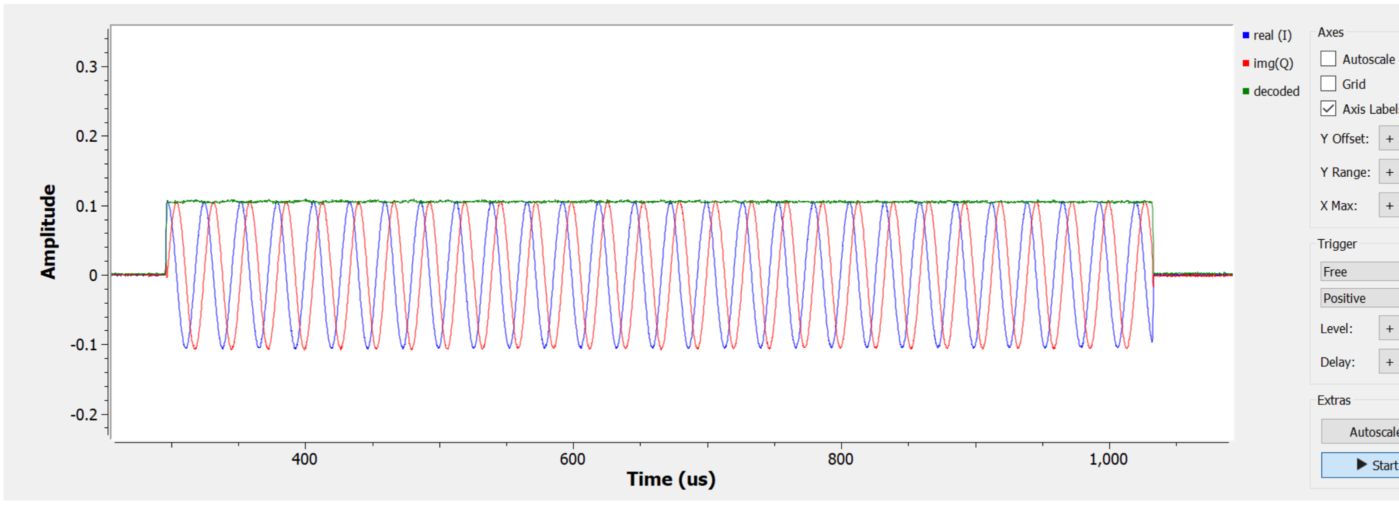
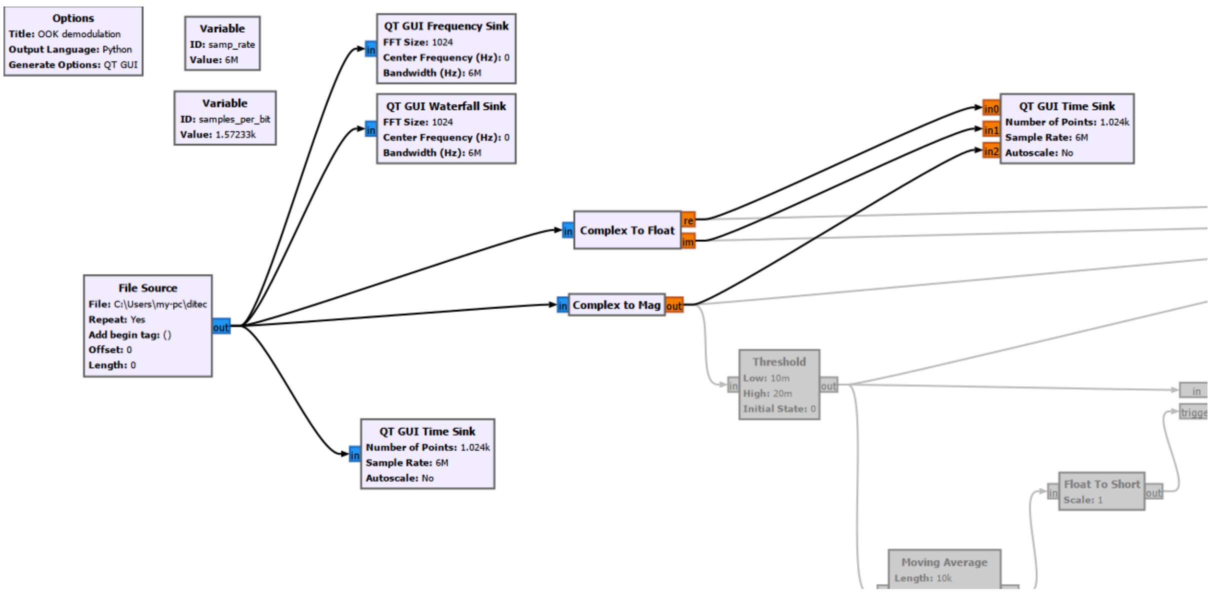


Figure 5: Complete frame structure showing distinct segments



Options
Title: OOK demodulation
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 6M

Variable
ID: samples_per_bit
Value: 1.57233k

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

QT GUI Waterfall Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

File Source
File: C:\Users\my-pc\ditec
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

Threshold
Low: 10m
High: 20m
Initial State: 0

Float To Short
Scale: 1

Moving Average
Length: 10k

Figure 6: the flowgraph untill now

4 Signal Decoding Process

4.1 Threshold Detection

A threshold detector was implemented to convert the analog envelope to digital bits:

$$\text{Binary output} = \begin{cases} 1 & \text{if } x(t) > V_{thresh-high} \\ 0 & \text{if } x(t) < V_{thresh-low} \end{cases}$$

where $V_{thresh-high} = 0.02$ and $V_{thresh-low} = 0.010$.

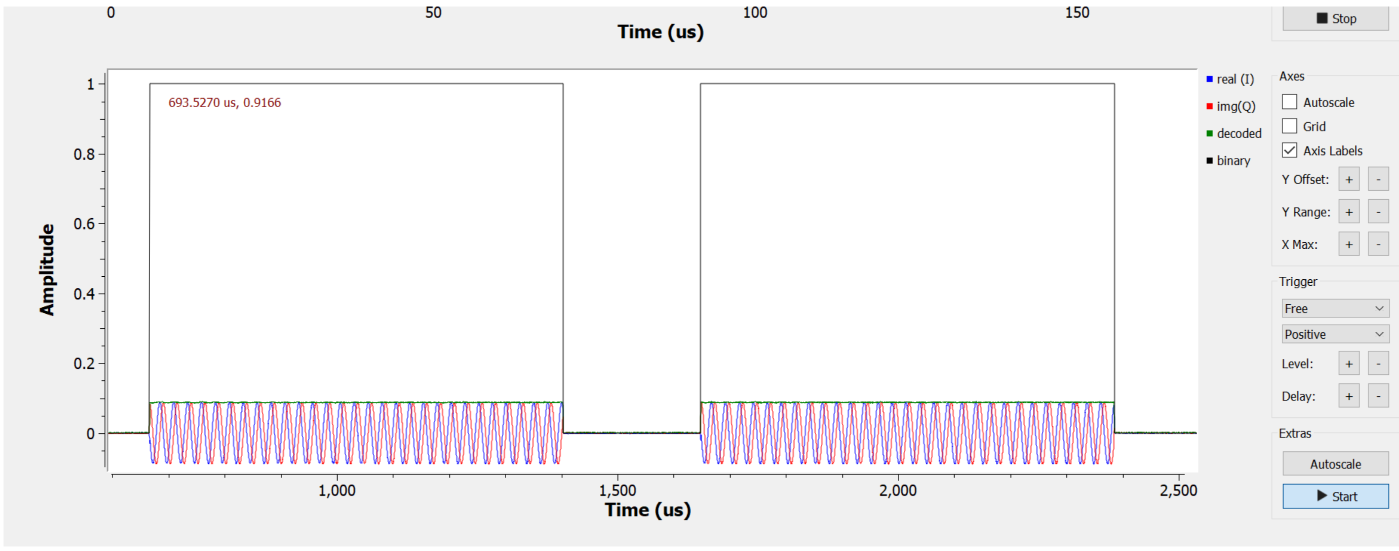
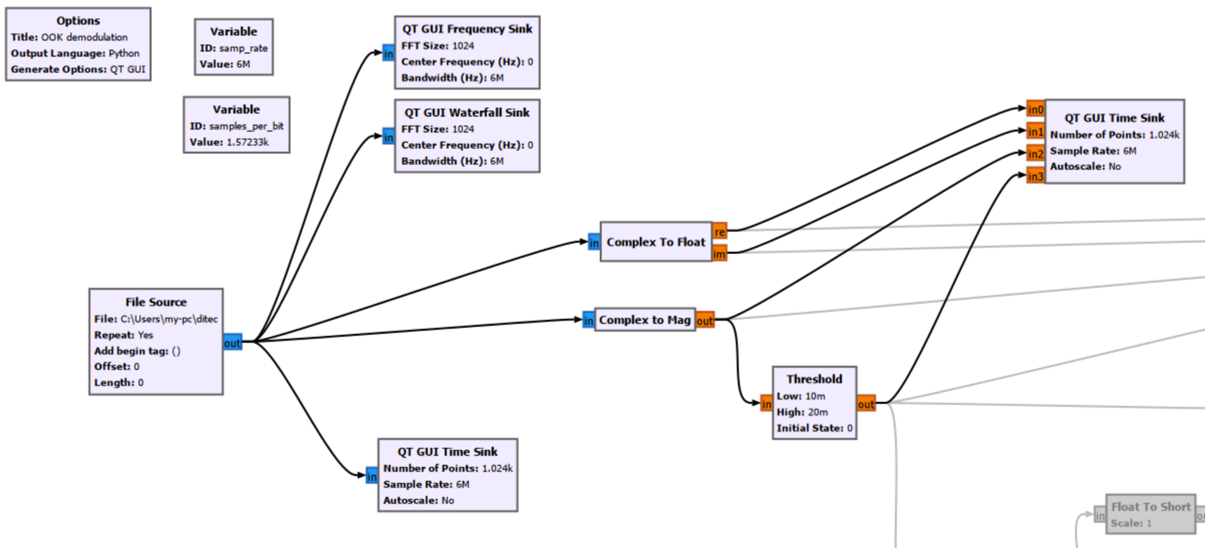


Figure 7: showcase of the thresholding binarisation



Options
Title: OOK demodulation
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 6M

Variable
ID: samples_per_bit
Value: 1.57233k

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

QT GUI Waterfall Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

File Source
File: C:\Users\my-pc\ditec
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

Complex To Float

Complex to Mag

Threshold
Low: 10m
High: 20m
Initial State: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

Float To Short
Scale: 1

Figure 8: flowgraph of said thresholding

4.2 PWM Decoding

Analysis revealed PWM encoding, where each bit is represented by a different periods of HIGH state , **Howver** before that we must extract our signal threshold data , note that we avoided just using a file sink block as it would consume insane amount of ram resources , and their prices are extensively higher

thus leading us to To : Analysis revealed that the signal uses PWM encoding, where each bit is represented by a different HIGH-state duration (period).

However, before decoding, we first need to extract the thresholded signal data. It is important to note that we intentionally avoided using a simple File Sink block, since it would consume an excessive amount of RAM resources, which are costly and inefficient for long captures.

This constraint led us to the following approach:

First, we need to detect when a frame is received. This is not difficult, as we can use the output of the Threshold block to detect the first '1' of a frame. Once this initial '1' is detected, we must keep a control signal at '1' for the entire duration of the frame.

To achieve this, we use a Moving Average filter. While the received data remains at '1', the filter output stays at least equal to 1 over the filter length. After the last '1' of the frame is received, the filter output remains non-zero for a duration equal to:

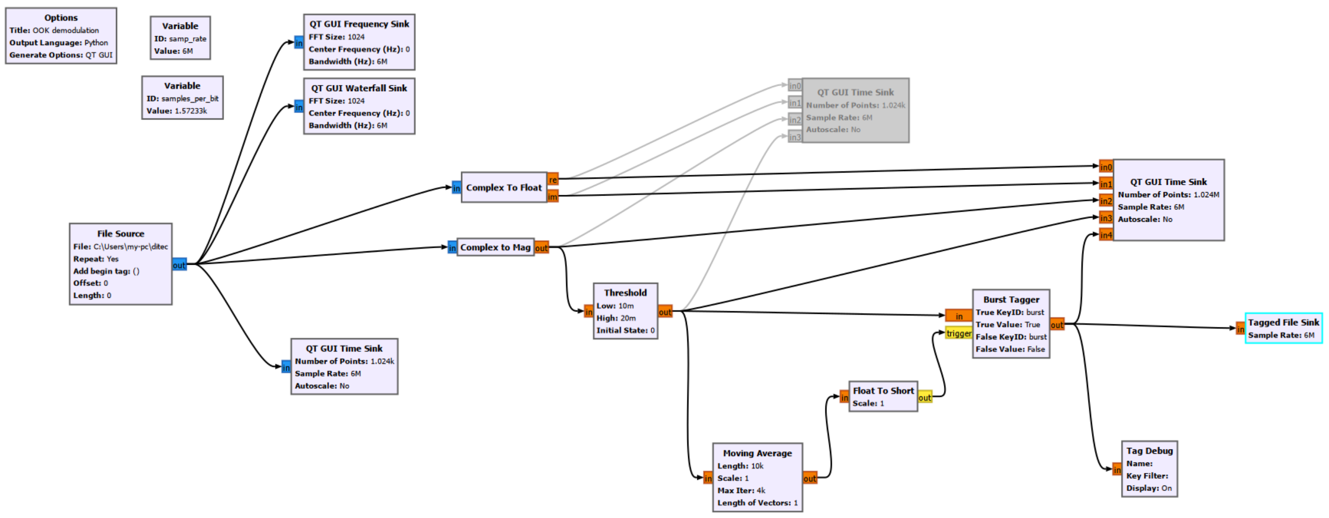
(filter length) $\times$ (sampling period)

In our case, we configured a filter length of 10,000 samples (10k), which provided reliable results.

Finally, the output of this filter is used as a control signal (switch) to manage file writing. A new file is created when the filter output goes to '1', and data is stored in this file for as long as the filter output remains '1'. When the signal returns to '0', the file is closed, and the system waits for the next frame.

This functionality is implemented using the Burst Tagger and Tagged File Sink blocks. The Burst Tagger generates a tag while its Trigger input is TRUE, and the Tagged File Sink writes data to a file only when the burst tag value is TRUE. Therefore, we only need to ensure that the Burst Tagger sets the burst tag to TRUE during the time interval in which we want to store the data.

resulting in the next gnuradio flowgraph :



Options
Title: OOK demodulation
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 6M

Variable
ID: samples_per_bit
Value: 1.57233k

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

QT GUI Waterfall Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 6M

File Source
File: C:\Users\my-pc\ditec
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

Complex To Float

Complex to Mag

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 6M
Autoscale: No

Threshold
Low: 10m
High: 20m
Initial State: 0

Moving Average
Length: 10k
Scale: 1
Max Iter: 4k
Length of Vectors: 1

Float To Short
Scale: 1

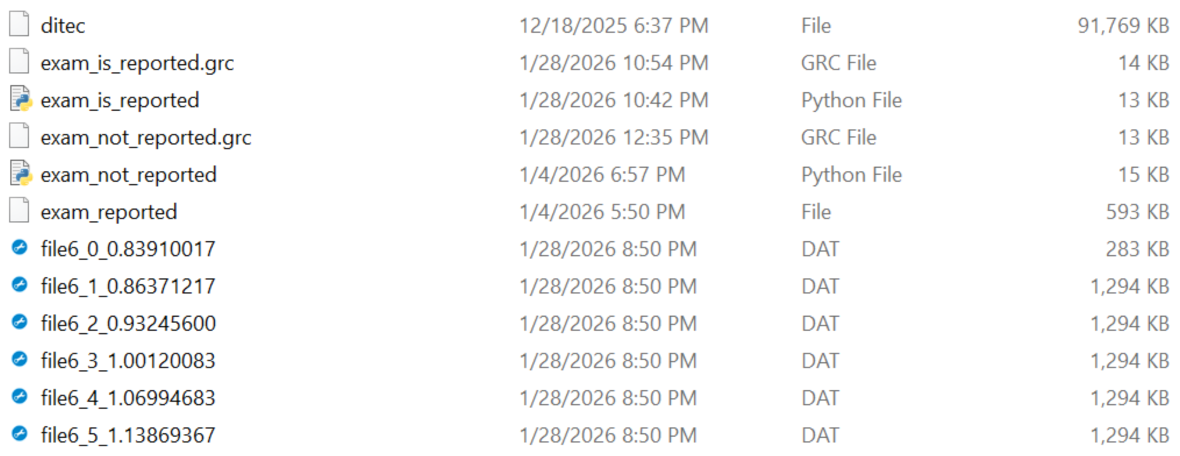
Burst Tagger
True KeyID: burst
True Value: True
False KeyID: burst
False Value: False

QT GUI Time Sink
Number of Points: 1.024M
Sample Rate: 6M
Autoscale: No

Tagged File Sink
Sample Rate: 6M

Tag Debug
Name:
Key Filter:
Display: On

we then recieve these 6 frame files



ditec	12/18/2025 6:37 PM	File	91,769 KB
exam_is_reported.grc	1/28/2026 10:54 PM	GRC File	14 KB
exam_is_reported	1/28/2026 10:42 PM	Python File	13 KB
exam_not_reported.grc	1/28/2026 12:35 PM	GRC File	13 KB
exam_not_reported	1/4/2026 6:57 PM	Python File	15 KB
exam_reported	1/4/2026 5:50 PM	File	593 KB
file6_0_0.83910017	1/28/2026 8:50 PM	DAT	283 KB
file6_1_0.86371217	1/28/2026 8:50 PM	DAT	1,294 KB
file6_2_0.93245600	1/28/2026 8:50 PM	DAT	1,294 KB
file6_3_1.00120083	1/28/2026 8:50 PM	DAT	1,294 KB
file6_4_1.06994683	1/28/2026 8:50 PM	DAT	1,294 KB
file6_5_1.13869367	1/28/2026 8:50 PM	DAT	1,294 KB

Figure 9: the 6 .dat files in chronological order

and using python as an analysis tool we can plot the interesting data from the frames

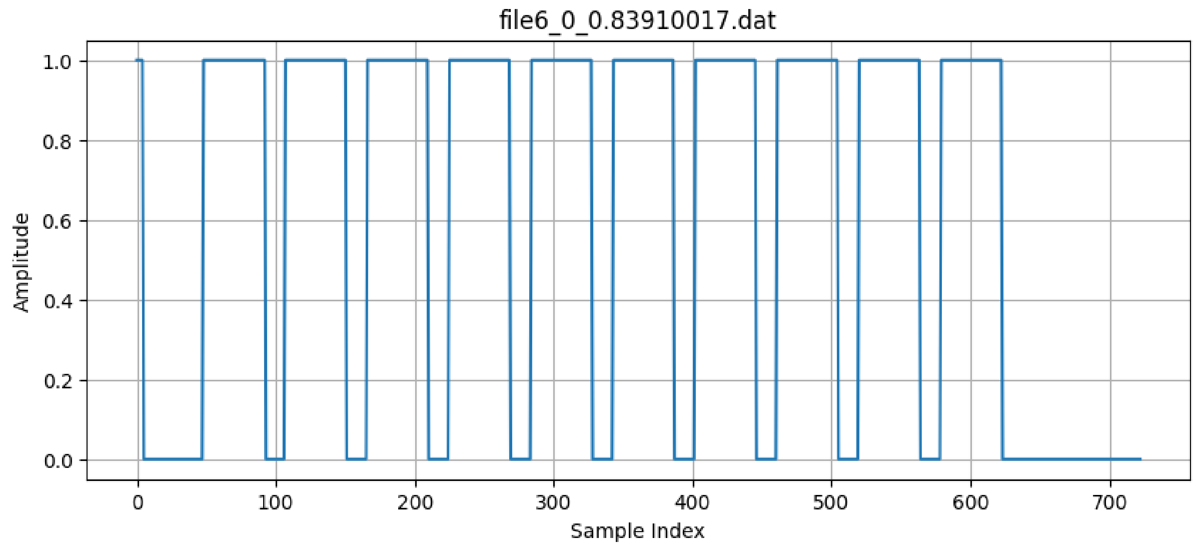


Figure 10: PWM encoding example showing the different levels (first frame)

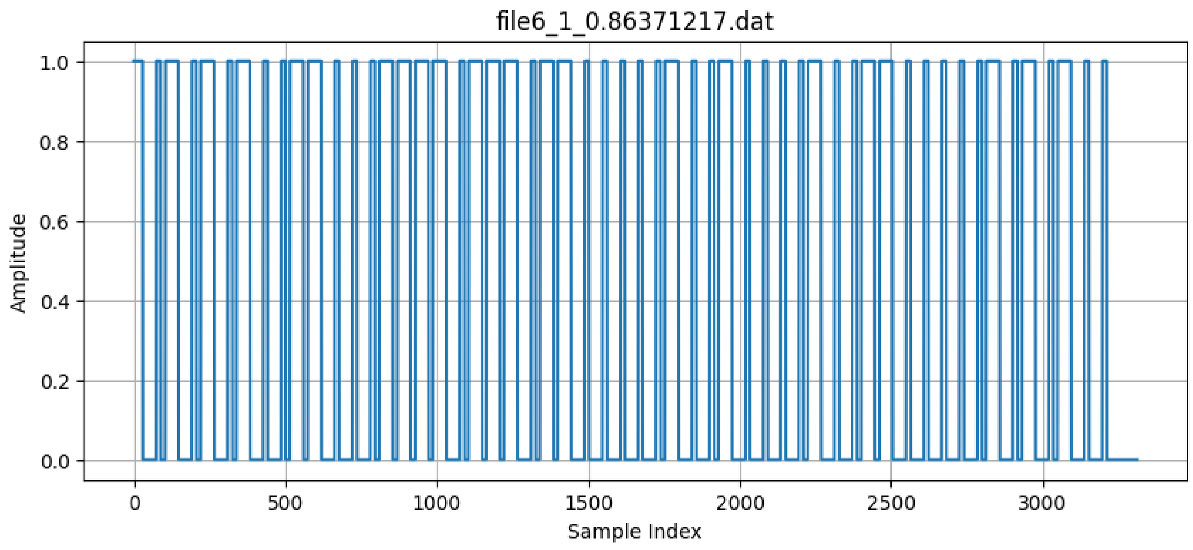


Figure 11: PWM encoding example showing the different levels (second frame)

4.3 Python Decoding Algorithm

The following Python code was used for Manchester decoding:

Listing 1: PWM decoding function

```python
import numpy as np
import glob
import os

def decode_file(file_path):
    # 1. Load the thresholded binary data
    # Standard GNU Radio .dat files are typically float32
    try:
        data = np.fromfile(file_path, dtype=np.float32)
    except Exception as e:
        return f"Error reading file: {e}"

    # 2. Find rising edges (0 to 1 transitions)
    # We look for where the difference between samples is
       positive
    diff = np.diff(data)
    rising_edges = np.where(diff > 0.5)[0]

    # 3. Calculate the time (in samples) between each rising edge
    periods = np.diff(rising_edges)

    if len(periods) == 0:
        return "No pulses found"

    # 4. Determine Threshold for Pulse Interval Encoding
    # Short periods = 0, Long periods = 1
    # We use the average of the min and max period found in this
       specific file
    threshold = (np.min(periods) + np.max(periods)) / 2

    # 5. Convert periods to a bitstring
    bits = "".join(['1' if p > threshold else '0' for p in periods])

    return bits

# --- Main Loop ---

# This pattern finds all files starting with 'file6_' and ending in '.dat'
file_pattern = "file6_*.dat"
data_files = sorted(glob.glob(file_pattern))

print(f"Found␣{len(data_files)}␣files␣to␣process.\n")

for f in data_files:
    # Get just the filename for cleaner printing
    short_name = os.path.basename(f)

    result = decode_file(f)

    print(f"---␣File:␣{short_name}␣---")

    # Check if it's a preamble (mostly the same bit repeating)
    if result.count(result[0]) == len(result):
        print(f"Type:␣Potential␣Preamble")
    else:
        print(f"Type:␣Data␣Segment")

    print(f"Decoded␣Bits:␣{result}\n")
```

5 Results and Analysis

5.1 Decoded Bit Sequences

Multiple transmissions were captured and decoded, revealing the following patterns:

Filename	Type	Length (bits)	Decoded Binary Sequence
file6_0_0.dat	Preamble	9	010101010
file6_1_0.dat	Data Segment 1	53	01010100010000001101010011010010010010100010010010101 1
file6_2_.dat	Data Segment 2	53	01010100111000011000100110100100101001100101010101011
file6_3_1.dat	Data Segment 3	53	01010100011001011001101110100110101001101101010101010
file6_4_1.dat	Data Segment 4	53	01010110110100111001101101100110110101101101100101010
file6_5_1.dat	Data Segment 5	53	01010110111001101001101101100110111001000111010101011

5.2 Frame Structure Analysis

Analysis of the decoded sequences reveals:

5.2.1 Preamble Structure

The preamble (010101010) serves as a synchronization pattern. The alternating 0-1 pattern helps the receiver establish bit timing through zero-crossing detection.

5.2.2 Data Field Analysis

Comparing the data segments reveals both fixed and variable components:

Listing 2: Pattern analysis between frames

```
# Fixed header across all frames (bits 0-15):
0101010001000001  # Present in all data segments

# Variable payload (bits 16-52):
# Changes with each transmission, likely containing:
# - Device ID (fixed portion)
# - Rolling code (variable portion)
# - Command (open/close/stop)
# - Checksum or error correction
```

5.3 Security Analysis

The analysis indicates this keyfob uses a **fixed-code system** with the following characteristics:

- **Fixed Preamble**: Always identical, allowing easy detection
- **Partially Fixed Data**: First 16 bits remain constant
- **Variable Payload**: Last 37 bits change, possibly implementing simple rolling code
- **Vulnerability**: Susceptible to replay attacks if variable portion repeats

6 Technical Specifications

Based on the analysis, the keyfob specifications are:

Parameter	**Value**
Carrier Frequency	433.92 MHz
Modulation	OOK (On-Off Keying)
Bit or symbol Rate	3816 bps
Encoding	Pulse width modulation
Frame Length	62 bits (9 preamble + 53 data)
Transmission Type	Burst transmission (6 segments)
Security Level	Basic fixed-code with partial variability

Table 1: Technical specifications of analyzed keyfob

7 Conclusion

This analysis successfully demonstrated the complete decoding process of a garage door keyfob RF signal. Key findings include:

1. The signal uses OOK modulation at 433.92 MHz with Manchester encoding
2. Bit rate was determined to be 1908 bps
3. Frame structure consists of a 9-bit preamble followed by 53-bit data segments
4. The system appears to use a partially fixed code with some variability
5. Security analysis reveals vulnerability to replay attacks

7.1 Security Implications

The analyzed keyfob highlights fundamental security characteristics of basic RF remote systems:

- **Fixed-Code Vulnerabilities:** Transmit identical codes, enabling simple *replay attacks* through signal capture and retransmission.
- **Rolling-Code Advantages:** Generate unique, one-time codes for each transmission, preventing code reuse through cryptographic sequencing.
- **Modern Keyless Systems:** Employ proximity-based *challenge–response protocols* with stronger cryptography, yet remain susceptible to relay/signal amplification attacks.
- **Emerging Attack Surfaces:** Continuous wireless communication introduces new vulnerabilities despite cryptographic enhancements, necessitating both digital and physical security considerations.

8 Appendix

8.1 Complete GNU Radio Flowgraph

The complete GNU Radio flowgraph used for analysis is available in the `keyfob_analysis.grc` file.

8.2 Python Analysis Scripts

All Python scripts used for decoding and analysis are included in the submission archive.

8.3 Raw Data Files

The original `.dat` files containing IQ samples are preserved for verification purposes.

8.4 References

1. "Decoding a Car Key Fob with the USRP B206mini and GNU Radio" - Ettus Research
2. "RF Signal Analysis Fundamentals" - GNU Radio Documentation
3. "Wireless Security: Rolling Code Systems" - IEEE Security Papers
4. "Manchester Encoding Theory and Applications" - Digital Communications Textbook