EE 315: Electrical and Electronic Circuits III

EE 315-02 - Winter 2024

California Polytechnic State University – San Luis Obispo
Electrical Engineering Department

Frequency-Triggered Lock: Enhancing Security with Frequency-Based Access Control

Instructor: Dr. Siddarth Vyas
Robin Simpson
Sam Solano

Contents

1 Objective of the Project

The primary objective of this project is to design and implement a frequency-triggered lock system. This locking mechanism enhances security by utilizing unique frequency signatures as keys. Diverging from traditional locks that rely on physical or digital keys, this system employs a combination of acoustic signals and electronic components to control access, effectively reducing the risk of unauthorized entry by requiring a specific frequency for unlocking.

Design Explanation

The design of the frequency-triggered lock system is centered around a sophisticated assembly of electronic components tasked with processing acoustic signals and actuating a physical lock. The system operates through the following stages:

- A **microphone** (CMA-4544PF) captures ambient sounds, converting acoustic waves into electrical signals. The sensitivity of this component is critical for detecting the nuances of the desired frequency amidst ambient noise.
- An **amplifier** boosts the microphone's signal to a usable level, ensuring the system can accurately process it. This amplified signal is then fed into a comparator.
- The **comparator** serves a vital role in shaping the signal into a clean square wave representative of the frequency detected by the microphone. It is not the decision-making unit but a signal-processing stage that prepares the electrical signal for analysis by delineating a clear distinction between the high and low states of the frequency waveforms.
- The **Arduino Uno R3** microcontroller is the system's decision-maker. It is programmed to analyze the square wave from the comparator. If the wave corresponds to the target frequency, the Arduino executes a command to change the state of the lock mechanism.
- The **actuation** stage is managed by a solenoid and a MOSFET actuator, which function together as the lock. When the Arduino determines that the target frequency has been reached, it signals the power MOSFET (AOI4286), which in turn energizes the solenoid. The solenoid, acting as a physical locking pin, retracts to unlock. If the frequency does not match, the solenoid remains in its default state, keeping the lock secured. A diode (1N4004) is incorporated in parallel with the solenoid to protect against voltage spikes caused by the inductive load.

This frequency-triggered lock system not only exemplifies security mechanism through its reliance on a unique acoustic signature but also showcases the integration of basic electronic components to realize a secure locking system. It demonstrates the potential for simple electronic principles to be applied in practical applications.

2 Schematic and Design

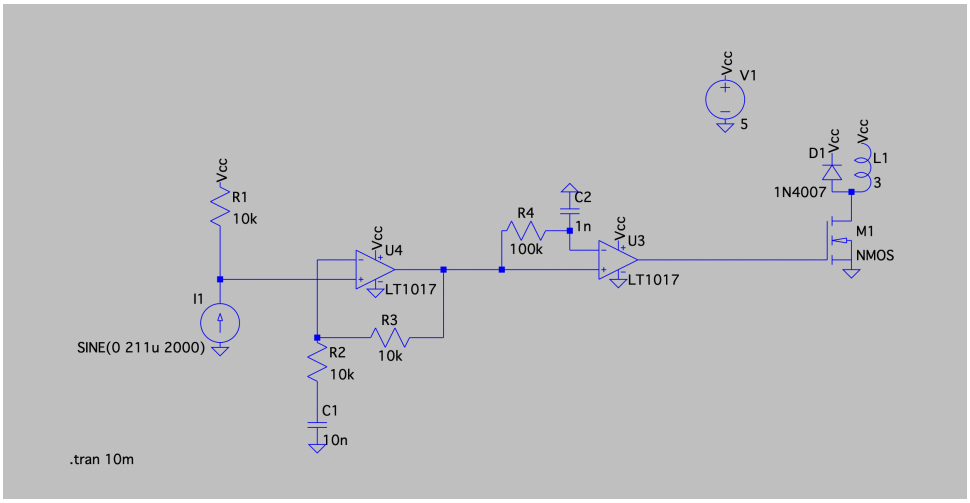

Figure 1: Amplifier + Comparator connected to the Actuator

The schematic depicts an electronic circuit designed to actuate a lock mechanism through frequency detection. The circuit encompasses three principal stages: frequency detection, signal amplification and comparison, and actuation.

Frequency Detection Stage

The input, denoted as I1, is set to produce a sine wave with an amplitude of 211 microvolts at a frequency of 2000 Hz, which simulates the acoustic frequency to be detected. Resistors R1 and R2, each of 10kΩ, along with capacitor C1 of 10nF, form a biasing network that establishes the operational point for the input signal.

Signal Amplification and Comparison Stage

Operational amplifiers U4 and U3, both LT1017 models, are employed to amplify the frequency signal and perform comparison. U4, configured as a non-inverting amplifier, utilizes R3 (10kΩ) in its feedback loop for stability. The non-inverting input is biased by the aforementioned resistive-capacitive network. U3 functions as a comparator, with R4 (100kΩ) and C2 (1nF) incorporated to provide hysteresis and noise immunity, crucial for translating the analog input into a clean digital square wave output.

Actuation Stage

The comparator's output is connected to an NMOS power transistor M1, which operates as the switch for the actuator. Diode D1, a 1N4007, is connected in parallel with the inductive load of solenoid L1, serving as protection against inductive spikes. Solenoid L1, modeled as a 3Ω coil, is the actuator for the locking mechanism. When M1 is activated by the comparator, it energizes the solenoid to retract the lock bolt if the target frequency is present, otherwise keeping the lock secured.

The system is powered by a 5V source, V1, which supplies the necessary operational voltage to both the op-amps and the solenoid through Vcc connections.

3 Implementation

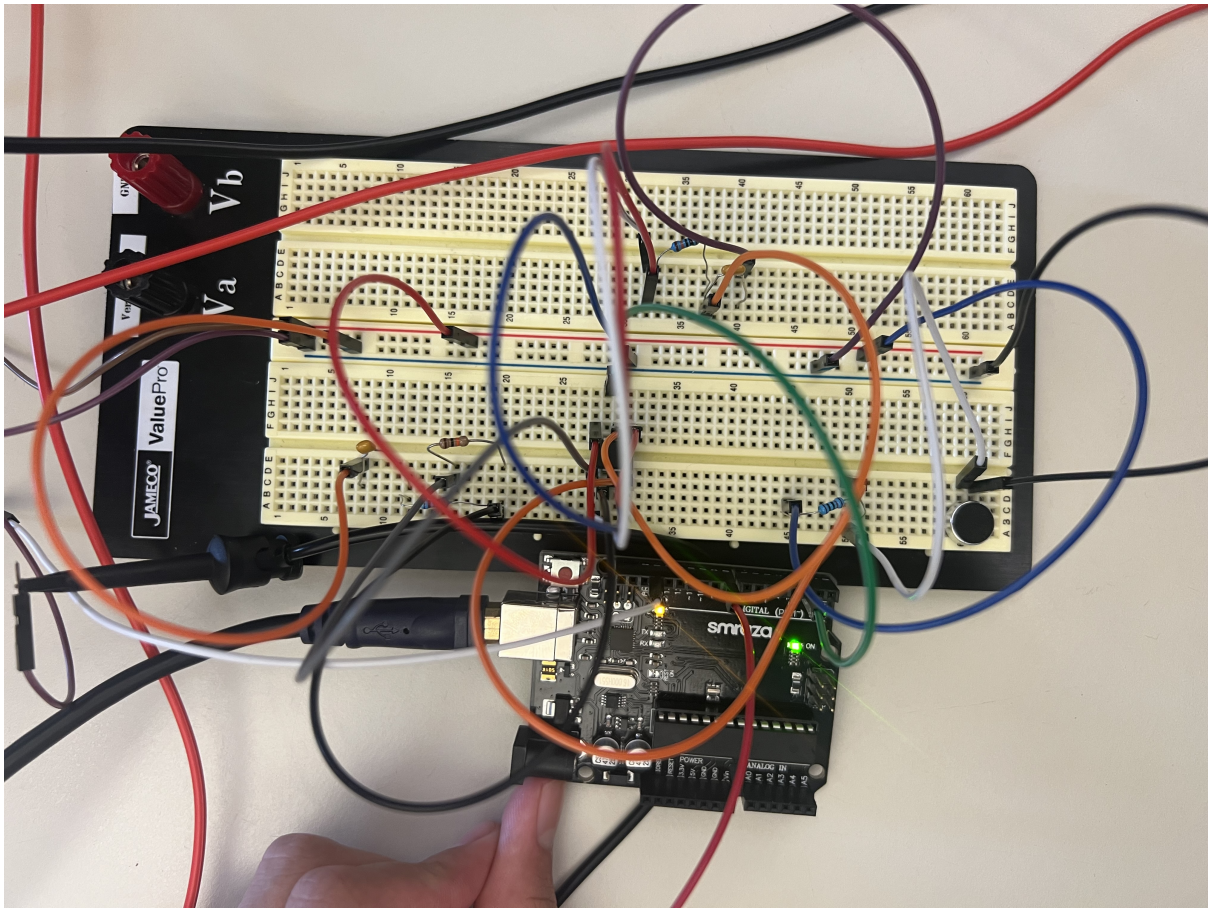

Figure 2: This circuit diagram illustrates the amplifier and comparator stages integrated with a microcontroller. The amplifier enhances the signal received from the input device, while the comparator standardizes the signal into a binary output for the microcontroller to process.

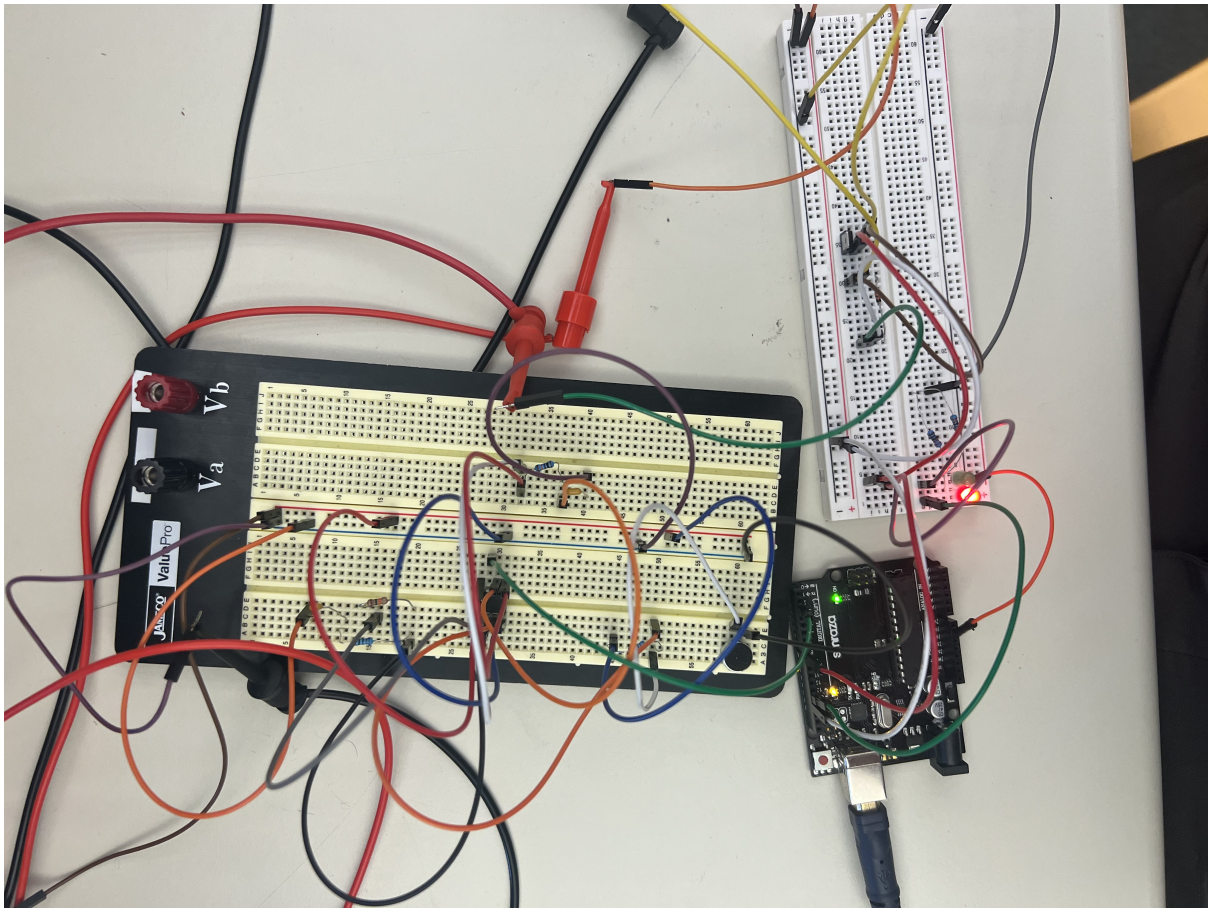

Figure 3: This schematic extends the previous figure by adding an actuator and LEDs to the system. The microcontroller, after processing the comparator's output, activates the actuator, which serves as the physical locking mechanism in the frequency-triggered lock system.

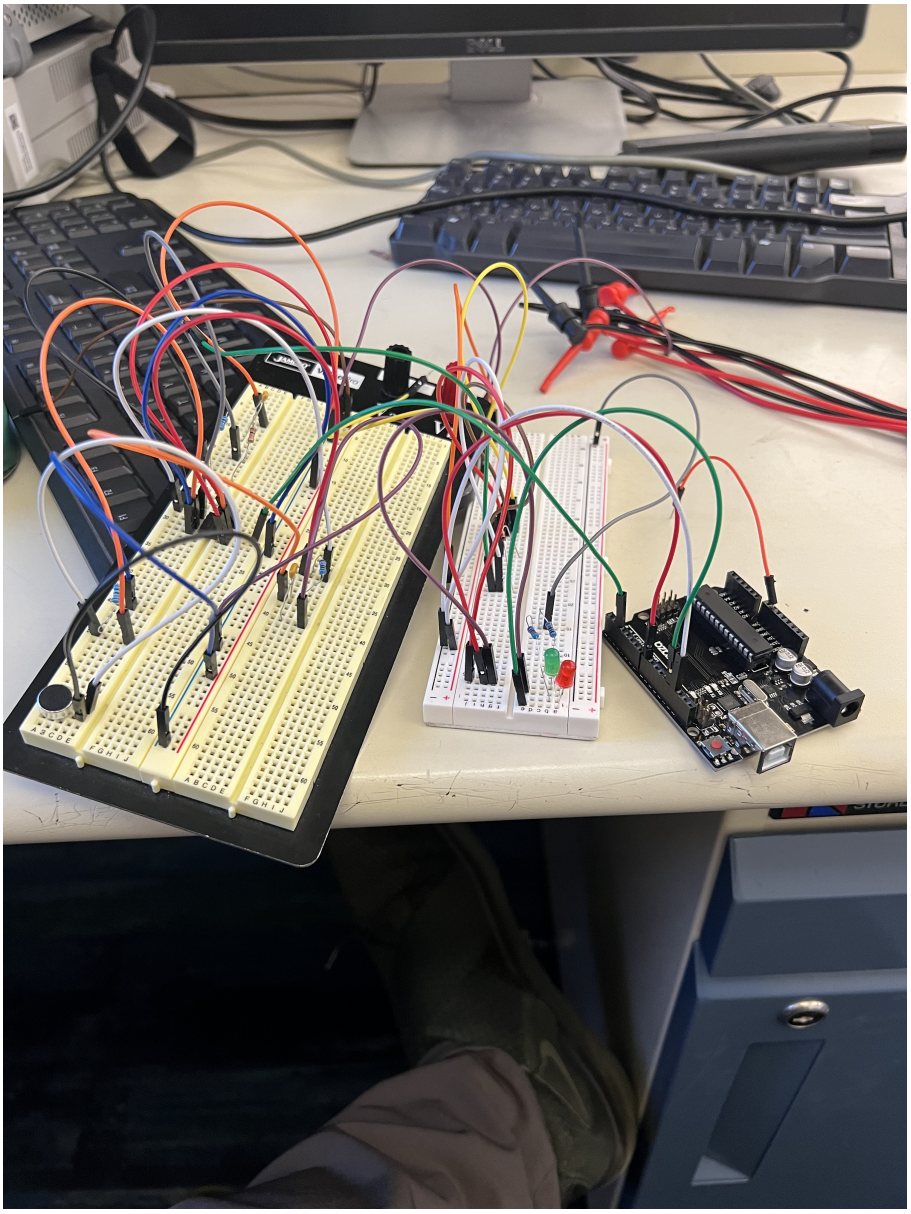

Figure 4: Presented here is the complete system layout, including the amp, comparator, controller, and actuator,. The orientation of the diagram facilitates a clear view of the component layout for assembly and troubleshooting purposes.

4 Microcontroller

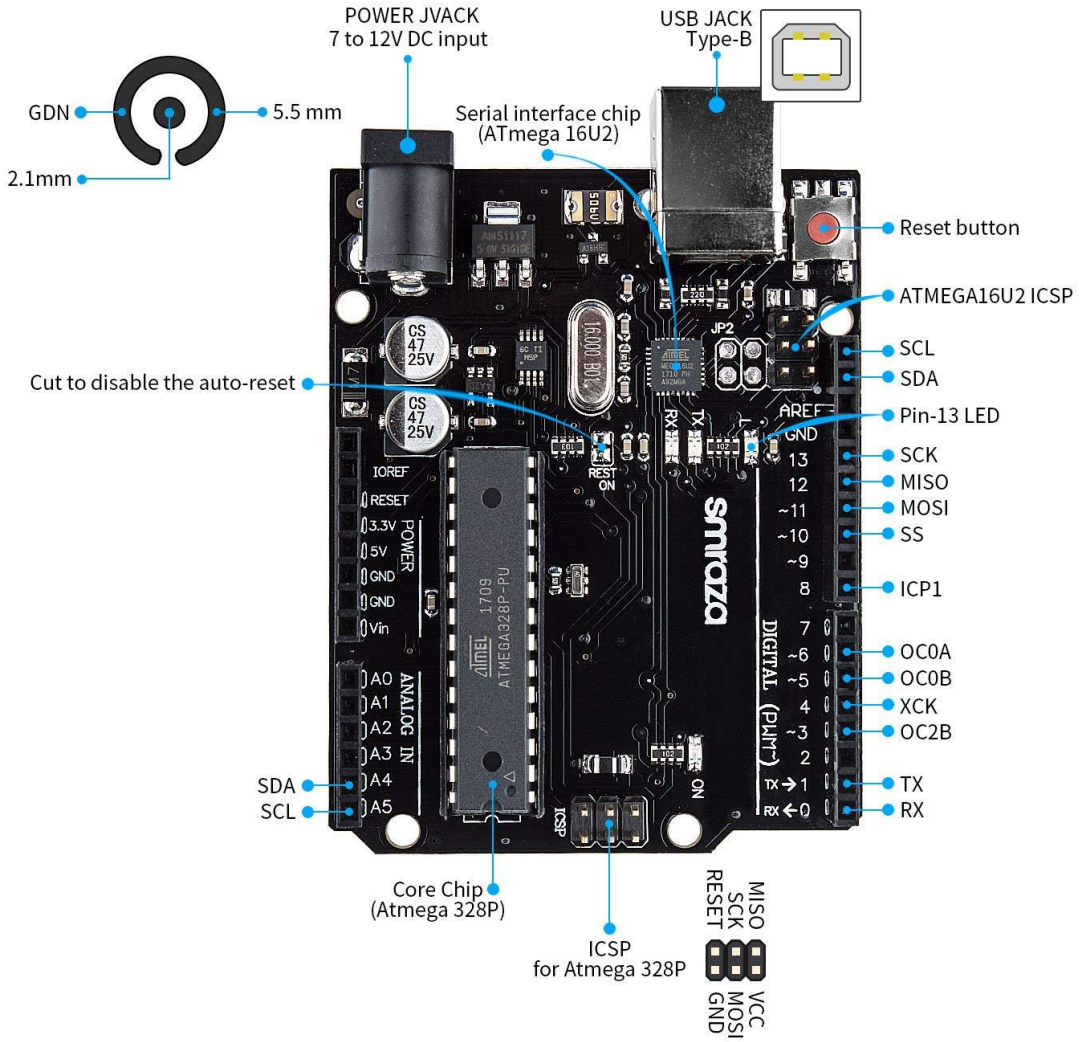

Figure 5: UNO R3

The Arduino Uno R3 was the microcontroller of choice for the frequency-triggered lock system due to its balance between performance, accessibility, and cost. The Uno R3's chip provides sufficient computational power for real-time processing. The implementation leveraged the Arduino's analog and digital I/O pins, clock speeds, and serial communication capabilities to create an efficient and responsive locking mechanism.

4.1 Programming the Arduino

The project's programming logic was developed in the IDE, utilizing C/C++ language to manipulate the microcontroller's hardware. The code was architected to continuously sample an input signal pin that connected to an amplifier and comparator circuit, which pre-processed the received audio signals. This preprocessing converted the analog audio frequencies into square wave signals that the Arduino could interpret with digital logic.

4.2 Signal Processing

Through programming, the Arduino Uno R3 was instructed to measure the period of these square waves, which inversely correlates to their frequencies. The signal's high and low durations were timed using the `pulseIn` function, an accurate method to gauge pulse widths. By summing these durations, the Arduino calculated the total period of the waveform, from which the frequency was derived.

4.3 Frequency Analysis and Decision Making

The algorithm employed by the Arduino Uno R3 was designed to compare the calculated frequency with a pre-defined target frequency, which served as the acoustic 'key' to the lock. A tolerance threshold was set to account for minor variations and potential noise in the signal. If the sampled frequency fell within this tolerance window, the system deduced a valid 'key' had been presented and initiated the actuation sequence to unlock.

4.4 Actuation Sequence

Upon recognizing the correct frequency, the Arduino sent a digital signal to an external MOSFET, which then powered a solenoid acting as the physical locking mechanism. This actuation process transforms an abstract frequency signal into a mechanical response.

4.5 Feedback and Monitoring

For user feedback and system monitoring, an LED was incorporated into the design and controlled by the Arduino. It provided immediate visual feedback regarding the system's when the lock was engaged and disengaged.

4.6 Future Work

Looking ahead, the project opens avenues for further refinement. Enhancements could include optimizing the algorithm for frequency detection, improving the system's power efficiency, and exploring wireless communication options to enable remote actuation. The adaptability of the Arduino Uno R3 leaves room for these advancements, highlighting the potential for scalability and the introduction of additional features.

5 Arduino Code

```
1 // Definitions for pins and target frequency
2 const int inputPin = 2;
3 const int ledPin = 13;
4 const int controlPin = 8;
5 const float targetFrequency = 4000.0;  // Hz
6 const float frequencyTolerance = 50.0;  // Hz
7 const int sampleSize = 10;
8 const unsigned long timeoutDuration = 1000000;
9 const unsigned long microSecondsInOneSecond = 1000000;
10
11 // Setup function to initialize the pins and serial communication
12 void setup() {
13   pinMode(controlPin, OUTPUT);
14   pinMode(inputPin, INPUT_PULLUP);
15   pinMode(ledPin, OUTPUT);
16   Serial.begin(9600);
17 }
18
19 // Main program loop
20 void loop() {
21   unsigned long periodSum = 0;
22   int validSamples = 0;
23   float frequency = 0;
24
25   for (int i = 0; i < sampleSize; i++) {
26     unsigned long highTime = pulseIn(inputPin, HIGH,
     timeoutDuration);
27     unsigned long lowTime = pulseIn(inputPin, LOW,
     timeoutDuration);
28
29     if (highTime != 0 && lowTime != 0) {
30       unsigned long period = highTime + lowTime;
31       frequency = microSecondsInOneSecond / period;
32
33       if (abs(frequency - targetFrequency) <= frequencyTolerance)
     {
34         periodSum += period;
35         validSamples++;
36       }
37     }
38   }
39
40   processFrequency(validSamples, periodSum, frequency);
41   delay(500);  // Update every 1/2 second
42 }
43
44
45
46
```

```
47
48
49 // Helper function to process the frequency and actuate the lock
50 void processFrequency(int validSamples, unsigned long periodSum,
     float lastFrequency) {
51
52   if (validSamples > 0) {
53     float averageFrequency = microSecondsInOneSecond / (periodSum
     / validSamples);
54
55     Serial.println("Average Frequency: " + String(
     averageFrequency) + " Hz - Enabled");
56
57     digitalWrite(controlPin, HIGH);
58     digitalWrite(ledPin, LOW);
59
60   } else {
61     Serial.println("No valid signal. Current frequency: " +
62
63     String(lastFrequency) + " Hz is outside the range of " +
64
65     String(targetFrequency) + " Hz with a window of " +
66
67     String(frequencyTolerance) + " Hz - Disabled");
68
69     digitalWrite(controlPin, LOW);
70     digitalWrite(ledPin, HIGH);
71   }
72 }
```

6 Demo

Demo and explanation of the Frequency Lock

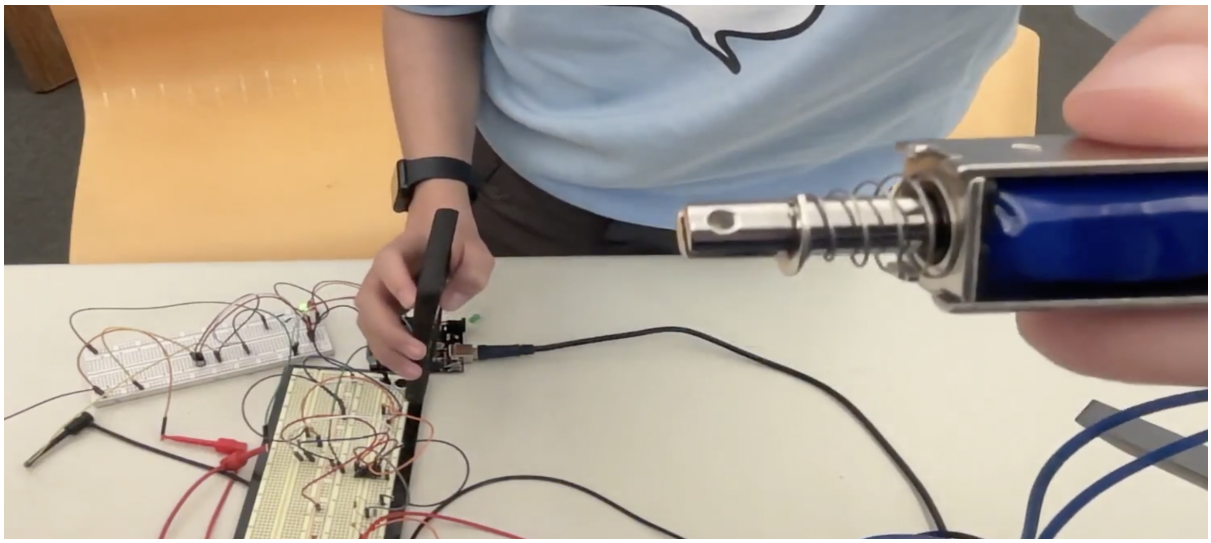

Figure 6: Click on the thumbnail to watch the video.

Extra: Doofensmirtz Explains the Frequency Lock

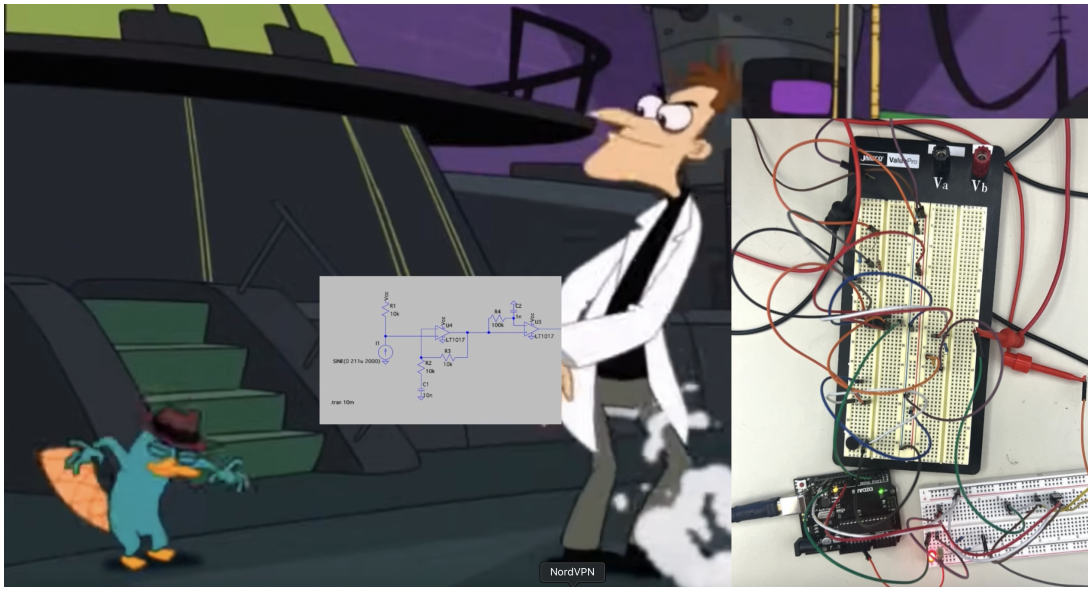

Figure 7: Click on the thumbnail to watch the video.