Morse Code Project
Final Report
Team Wind

ECE 103 Engineering Programming

Philip Nevins | pnevins@pdx.edu
Andrew Stanton | stan35@pdx.edu
Aziz Alshaaban | alshaab@pdx.edu
Luis Poroj Tepetlapa | poroj2@pdx.edu
Paul Krueger | paul.r.krueger@pdx.edu

Introduction

Our team decided to choose the Morse Code project for our ECE 103 Engineering Programming term project. The idea behind this project is that we are using a laser diode and photocell, to simulate a traditional Morse Code machine. The goal is to be able to input a series of words into the serial monitor of the Arduino IDE on the transmitter side, have the laser diode flash on and off to send these words to the photocell, and display them on the second Arduino IDE serial monitor that is hooked up to the receiver side. Each flash will also trigger a buzzer to play a sound, just like the traditional Morse Code machines do.

Final Design

The key features of the project is that you can input a series of words, decode those into a series of laser flashes, transmit this series to a photocell, and decode the response on the photocell back into a series of words. The UI includes a scrolling menu to choose between sending a message or several diagnostic options. It also has an info panel, and an activity log. The schematic and final design can be seen in the Appendix, Figure 4 and 3 respectively.

Hardware

For the hardware, we used the ESP32 board to run the code, and a laser diode that was bought from the EPL. We designed a stand to hold the laser and photocell, to ensure a direct connection to the photocell can be achieved. We cut a 24 inch board to fasten the transmitter and receiver microcontrollers to. Then we drilled a hole in two pieces of wood, to ensure the laser would make the direct connection to the photocell. Once everything was tested and confirmed to work, it is fastened to the base board. In Figure 2, we can see the initial iteration of the stand. In Figure 3, we can see the final version. Both of these Figures are in the Appendix.

Bill of Materials

Laser Diode
Lumber
ESP32 Huzzah x2
Breadboard x2
Photocell (CdS photoresistor)
Piezoelectric buzzer
Red LED

Theory of Circuit Operation

The idea behind the circuit is, one of the ESP32s takes in text from the UI, translates it to a series of dots and dashes, then this is converted into a series of flashing a laser on and off for specific times based on the dots and dashes. The laser is directly connected to a photocell that picks up the laser on/off series, converts it to a series of dots and dashes, and then converts that back into a series of words. This is accomplished by using the standard Morse Code library, seen in Figure 1. The photocell also triggers a buzzer, to play a series of tones based on the series of dots and dashes, exactly like how a traditional Morse Code machine operates.

Software

Transmitter

The transmitter code should take in a series of words, decode them into a series of dots, dashes and spaces. These dots and dashes are then decoded into a series of laser blinks of different lengths, and the spaces are decoded into a specified “laser off” length. This will differentiate between the dots, dashes and spaces.

The transmitter code uses two keys that are declared as global array pointers, that are defined in terms of dots and dashes. These definitions are from the standard Morse Code Translation table, found in Appendix, Figure 1.

After the global array declarations, the Serial Monitor is initialized, along with the Laser pin on the HUZZAH32. We type out a message into the UI, and send it. Then each letter is converted into a series dots and dashes, and based on the series, the laser is turned on and off at different rates.

Inside the Arduino loop() function, we declare the input letter variable, which will hold the letter being translated into dots and dashes. After this declaration, we use an if else if statement to check that there is a letter, number or space in the Serial Monitor. The if-else-if statement inside of the loop() function has four parts: lower case, upper case, number and space. The space case will just put out a specific delay that can be read by the photocell. The other three cases are where the decoder function is called.

Depending what the input is, it will trigger a different part of the if-else-if statement, which depends if we have a lower case, upper case or number. Then we subtract a or A from it, to convert a to 0 or A to 0, b to 1 or B to 1, etc. This is done because the ASCII value for a is not 0, so we can set the input letter to 0-25 by subtracting a from the input value. This value (0-25) is then used to pull the desired series of dots and dashes from the global array pointer, giving us the desired series of dots and dashes.. This series is then passed into the decoder function, which enters the series of dots and dashes into a new array, and sends the new array of dots and dashes to the second function. The second function turns the laser on, then has a delay based on if there is a dot or dash, then turns the laser off. It ends when the NULL character is found within the decoder function, because this signifies the end of the array.

Receiver

The receiver code is split into 3 main sections: recording the input time signatures, translating the time signatures into Morse code and then into Latin characters, and finally resetting the receiver in preparation for upcoming messages. A basic state machine is used to determine which mode the receiver is currently operating in.

The state machine includes 4 states: idle, recording, translating, and resetting. During the idle stage, the receiver constantly checks whether a signal is being received by the photocell. If no signal is present, nothing happens. Once a signal is received, the state machine automatically transitions into the recording state. During this state, the receiver records the timing of incoming inputs and stores them for later translation. Once no signal has been received for 5 seconds, the receiver transitions into the translating state. This state first translates the time signatures to Morse sequences, and then translates the sequences into Latin characters. The translation occurs in a concurrent manner that is detailed below. Once the

message has been translated and printed to the serial monitor, the machine transitions into the reset state. This state is short-lived, as it involves resetting the appropriate data structures to a functionally similar state that they were in when the machine first booted. Once this is complete, the receiver transitions back to the idle state, waiting for more input.

The input recording is accomplished by monitoring when the signal from the photocell changes. Two arrays are used for the task. The first array includes entries of 0s and 1s, indicating whether the photocell is currently reading a signal (1) or is not currently reading a signal (0). The second array stores the length of time for which that signal was held. This entry is stored as a double, with each second being saved as 1.0, allowing for partial seconds to be monitored. The two arrays are kept in sync by their index number. For example, the entry at index 0 in both arrays are related to the same event. The first array will show that the signal was on with a 1, and the second array will hold the length of time that the signal was received for. This continues for both the on and the off signals within the log. There are some helper variables, such as the last time the signal changed from one state to the other.

The translation portion of the receiver has two separate tasks that it constantly switches back and forth between. The first step that is taken is to begin translating the time signatures into their appropriate Morse characters. Since the transmitter and the receiver are working together, hard-coded time signatures are checked to determine what the signal is. Whichever symbol the time signature is closest to will be what the output is from this step, and it is stored within an array. Once either a space is received, 5 Morse characters are queued up within the array, or the end of the message is received, the stored sequence will be translated to the appropriate letter or number. The sequence is compared with a table that includes all Latin letters and Arabic numerals to determine which sequence fits. If the signal received does not match a character, then a question mark is saved to the message output. Once the final character is translated, it is printed out to the console.

Resetting the data structures is the final state of the machine. This resets the strings used for output to being empty, and resets the input log count to 0. The arrays are not zeroed out or cleaned up in any manner, as the log count and null-characters will indicate where new data ends when it is viewed during the next cycle through. Once these are reset, the receiver returns to its idle state and awaits new input.

The data structures used for the receiver are standard arrays. After setting a maximum length for the received messages, the other arrays are set to be 5 times as large due to letters being at most 4 dashes and/or dots. With spaces included, this puts required array space needed at 5 for each Latin character.

There are two main improvements that could be made to future iterations of this design. The first is to use a linked list in place of the static arrays, assuming the environment was not embedded. This would allow a more dynamic memory allocation that can match memory usage to each individual message being received. The second improvement would be to use a tree-structure for decoding the Morse code. The current structure of using string comparisons can be computationally heavy. Using a tree which includes all of the Morse code sequences and storing values as enums instead of strings would be faster, at the cost of setting up a more complicated decoding process.

UI

The user interface is implemented using the NCURSES library. It uses several windows to allow for several functions. The main window, titled “Send,” first allows the user to choose from a list of options. This is referred to as the “Action Menu.” The user can choose to send a message, to exit, or from a few diagnostic options such as turning the laser on or off, or sending a default test message.. The messages are sent to the microcontroller over USB serial, using only C code and the necessary POSIX headers such as `termios.h` and `unistd.h`.

Diagnostic Panel

The Diagnostic mode is used to test the hardware using signals from the microcontroller. It will test the laser, the buzzer and the red LED, It will also send a message over serial using a USB. It will inform the user on whether the hardware is working or not. If it is working then it will send “Laser ON”, if not it will send “Laser OFF”. This diagnostic mode uses device control characters (DC1, DC2, DC3, DC4) to represent the tests.

Pseudocode

Transmitter Side

The delay(200) {which means laser is on for 200, signifies .
The delay(600) {which means laser is on for 600, signifies -
The delay(1000) {which means laser is off for 1000, signifies a space

Declare global array pointers MCLib and MCNumbers that hold the morse code key
Declare Laser pin number

```
setup()
        Initialize laser output pin
        Initialize serial window at 115200

loop()
        Declare inputLetter char, which clears previous letter each iteration
        If there is a letter sent to the serial monitor
        Read serial input, save as inputLetter
                If letter is lowercase, subtract a from it
                Call MCLib[inputLetter - a]
                Send this array to MCDecoder
                Else If letter is upper case, subtract A from it
                Call MCLib[inputLetter - A]
                Send this array to MCDecoder
                Else If letter is a number, subtract 0 from it
                Call MCNumbers[inputLetter - 0]
                Send this array to MCDecoder
                Else if letter is a space, delay 1000
```

```
MCDecoder(char* x)  (array x points to MCNumbers / MCLib)
        Initialize dummy variable i = 0
        While x[i] does not equal NULL
                Call dashOrDot function using x[i] as input
                Increment i to go to next entry in array x

dashOrDot(char data)
        Turn laser on
                If dot, delay 200
                If dash, delay 600
        Turn laser off
Delay 200 to allow photocell to clear previous letter
```

Receiver Side

```
Start Serial Output
Attach photocell for input
Attach buzzer and LED pins for output

Initialize RecordingDataStructure
Initialize Message as empty string

Main Loop
  Switch Current State
    Case Idle
      If photocell input > 0
        Get current ESP time since boot, store in Timer
        Set State to Recording
      End If
    End Case
    Case Recording
      Call RecordInput, passing RecordingDataStructure
      If time since last input > 5 seconds
        Set State to Translating
      End If
    End Case
    Case Translating
      Call MorseTranslate, passing RecordingDataStructure and Message
      Print Message
      Set State to Reset
    End Case
    Case Reset
      Set Message to an empty string
      Set State to Idle
```

```
        End Case
    End Switch

    If input is being received on photocell
        Turn on buzzer
        Light up LED
    Else
        Turn off buzzer
        Turn off LED
    End If
End Main Loop

Function RecordInput
    Initialize Input with type of signal from photocell

    If Input is not equal to Recording Current
        Initialize Double TotalTime with current time - Recording Timer

        Set Recording Time for Recording Count to TotalTime
        Set Recording Type for Recording Count to Recording Current
        Add 1 to Recording Count

        Set Recording Current to Input
        Set Recording Timer to ESP library time
End Function

Function MorseTranslate
    Initialize Sequence as 5 character string
    Initialize Translated as 1 character string
    Initialize Count to 0

    For i in 0 -> Input Count
        Call TranslateTime, passing Input Type[i] and Input Time[i], set as Type

        Switch Type
            Case Dash
                Append - to Sequence
                Add 1 to Count
            End Case
            Case Dot
                Append . to Sequence
                Add 1 to Count
            End Case
            Case Separator
```

```
            Print the Sequence to Serial
            Call SequenceToCharacter passing Sequence, set to Translated
            Append Translated to Output
            Set Sequence to empty string
            Set Count to 0
        End Case
        Case Space
            Print the Sequence to Serial
            Call SequenceToCharacter passing Sequence, set to Translated
            Append Translated to Output
            Append a space to output
            Set Sequence to empty string
            Set Count to 0
        End Case
    End Switch

    If Count Is 5
        Print the Sequence to Serial
        Call SequenceToCharacter passing Sequence, set to Translated
        Append Translated to Output
        Set Sequence to empty string
        Set Count to 0
    End If
  End For
  Print Sequence to Serial with a newline
  Call SequenceToCharacter passing Sequence, set to Translated
  Append Translated to Output
  Append . to Output
End MorseTranslate

Function SequenceToCharacter
  For i in 0 -> 25
    If Sequence equals MorseCodeLib[i]
      Return MorseCodeLib[i]
    End If
  End For
  For i in 0 -> 9
    If Sequence equals MorseNumberLib[i]
      Return MorseNumberLib[i]
    End If
  End For

  Return null character
End Function
```

```
Function TranslateTime
   Initialize Difference as Double
   Initialize Result as enum None

   Switch TypeCode
      Case Off
         Set Difference to Time - NoTime

         If (Time - SeparatorTime) < Difference
            Set Difference to Time - SeparatorTime
            Set Result to enum Separate
         End If
         If (Time - SpaceTime) < Difference
            Set Result to enum Space
         End If
      End Case
      Case On
         Set Difference to Time - DotTime
         Set Result to enum Dot

         If (Time - DashTime) < Difference
            Set Result to enum Dash
         EndIf
      End Case
   End Switch

   Return Result
End Function
```

UI

```
Initialize NCURSES
Create windows
Initialize serial port
Bool loop = true
current _choice = 0
While loop
        For each option
                If current_choice == option
                        Display option
                Endif
        Endfor
        Key = getch()
        If key == up
```

```
            Increment current_choice
        Else if key == down
            Decrement current_choice
        Endif
        If key == enter
            Switch (current_choice)
                MESSAGE:
                    Str = get string from user
                    sendMessage(str)
                LASER_ON:
                    Enable laser
                LASER_OFF:
                    Disable laser
                TEST:
                    sendMessage(test)
                EXIT:
                    Exit UI
            Endswitch
        Endif
Endwhile
Close serial port
End NCURSES
```

Diagnostic Panel

```
Initialize laserPin out
Initialize BuzzerPin
Initialize resistorPin

Main loop
        If str == “LED ON”
            Turn LED On
        Else if str == “LED OFF”
            Turn LED Off
        Else if str == “Laser ON”
            Turn Laser ON
        Else if str == “Laser OFF”
            Turn Laser OFF
        Else if str == “Buzzer_TEST”
            Turn Buzzer on && off
            delay(50<=400)

        Else if str == “FULL”
            Turn LED on && off
            delay(1000)
```

Turn Buzzer on && off
delay(500)
Turn laser on && off
delay(1000)

Problems and Solutions

Philip Nevins: When I was first writing the code, I was having issues using a switch statement, so I went to the if-else-if statement and that worked better. Also, when testing, it wasn't producing the correct output, and I realized I did not subtract a/A from the input letter, because a/A is not 0 in ASCII. a/A is 32 in ASCII, so you have to subtract a/A from the input letter so it will pull the correct value from the key array.

Andrew Stanton: When designing the UI and learning NCURSES, I wanted the menu to also be accessible using the mouse. This proved difficult, as the mouse input is not consistent, and did not work well without a virtual terminal. I settled for keyboard input only, including (optional) vim style controls. Serial communication in C was also a lot to learn. Of all the things that could have gone wrong, luckily mismatching baud rates was the worst mistake I made.

Paul Krueger: The main problem I ran into was how to store the timing sequences from the photocell. My first couple of passes were over-engineered and too complicated for the task. I ended up using a pair of arrays kept in sync with each other through a shared counter. One array held whether the signal was on or off, and the other held how long that signal was in that state for. This was easier than any of my previous attempts, as a single iterator could read the appropriate values from both arrays. I also learned the importance of everyone on a team using the same compiler, as the version of the ESP fork of GCC included with the Arduino IDE is not the same on my Mac installation and Andrew's Linux installation, which led to us seeing dramatically different results when running the code.

Aziz Alshaaban: I had trouble connecting the diagnostic mode with the device control in the UI.because we don't have enough inputs for all the hardware. So, we ended up using two methods, for the laser we used the device controls and for the buzzer and LED we used basic string functions for each hardware.

Luis Poroj: I had trouble with the switch statement mainly because I was having trouble imagining how the computer would perceive each letter, I ended up using a for loop and used the null feature to make it work. An issue I had with the diagnostic code was that when I did my first initial research for it I used a string to alert the serial buffer if it was High or Low and display LASER ON/OFF which initial was wrong but I ended up using a snippet of code from the zy-book to help me.

Discussion

Philip Nevins: I learned that communication is important. SCRUM is an important aspect of working in an engineering team and environment. It can help keep you on track and make sure everything is being done in a timely manner. This project has taught me the skills that will be used in an engineering job. These skills are important to hone and practice, so you are prepared for your career when you graduate.

Andrew Stanton: There can be lots of ways to tackle a problem, you may not be able to solve it the way you want to. Keep it simple, don't over analyze, and have fun with it.

Paul Krueger: The solutions necessary for programming at a lower level with microcontrollers is different from the solutions one would use when programming apps at a higher level. Apps can be overengineered and inefficient, and modern computers and mobile devices will handle it perfectly, whereas microcontrollers require efficient solutions first. This required a change in approach from how I would typically handle a problem, but one that I have become more comfortable with through this project. Despite my implementation of it being very basic, I saw how effective state machines can be on microcontrollers that could potentially be running in a loop for months or years.

Aziz Alshaaban: I learned that programming can be frustrating but once you get everything to work it becomes really satisfying and opens up other ideas that could be fun to play around. I also learned that practice makes perfect, especially in programming.

Luis Poroj: I learned that communication is a key factor for the success and efficiency of the project. I also learned that if mistakes were made during the process of programming, its always good to pull yourself away for a breather and come back with a refreshed mind, sometimes new ideas come to you when procrastinating.

"Signature" Paragraph

Philip Nevins: I wrote the transmitter code and designed the hardware. I wrote the Hardware Transmitter, Software Transmitter, Theory of Circuit Operation, part of the Final Design and Introduction portion of the Final Report.

Paul Krueger: I wrote the code used by the receiver hardware. For the report, I wrote the section on the receiver software and the pseudocode outlining how the receiver operates. I worked with Andrew to make adjustments with the software to match the final hardware.

Andrew Stanton: I created the user interface and the serial communication code it uses. I worked with Aziz to integrate it into the diagnostic system. I was also responsible for the receiver circuit, including wiring up the photocell, buzzer, and LED. I then worked with both Phil and Paul in preliminary testing of the transmitter and receiver.

Aziz Alshaaban: I wrote the laser part of the diagnostic mode that connects with the UI that Andrew made and I worked with Luis's code to complete the diagnostic mode for every hardware we used.

Luis Poroj: I worked on the transmitter code but due to a communication error me and Philip ended up writing a code for the transmitter side, with this, we decided that whichever code was easiest to work with was the one the team was going to move forward with and we ended up using Philips' code. Then I worked on the structure of the diagnostic code getting most of the important features needed and with the help of Aziz we were able to complete the diagnostic code.

"All of the students listed at the top of this report have read it and agree with its content."

References

Arduino Project Hub. 2022. *Arduino projects*.
[online] Available at: <https://create.arduino.cc/projecthub>
[Accessed 23 July 2022]

H. Geoffrey. “Linux Serial Ports Using C/C++.” embedded.ninja.
https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp
(accessed 21 July 2022)

P. Padala. “NCURSES Programming HOWTO.” The Linux Documentation Project.
https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/index.html
(accessed 26 July 2022)

Z. Ben-Halim *et al. ncurses(3X)*. Accessed: July 26, 2022. [online]. Available:
https://linux.die.net/man/3/ncurses

Appendix

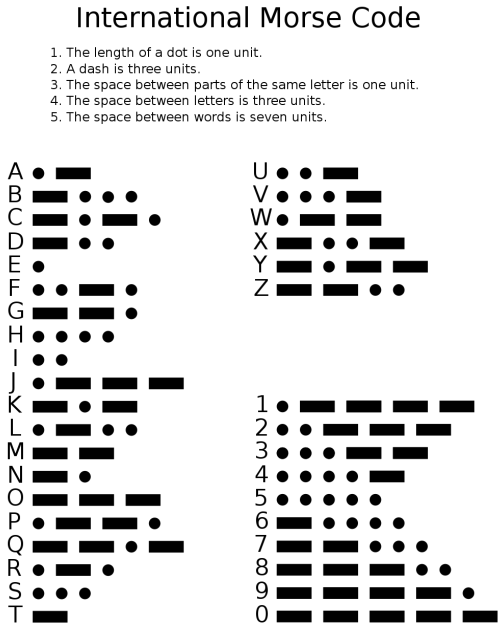


Figure 1: Morse Code Translation Table

Figure 2a: Hardware Design, First Iteration (Transmitter Side and Stand)

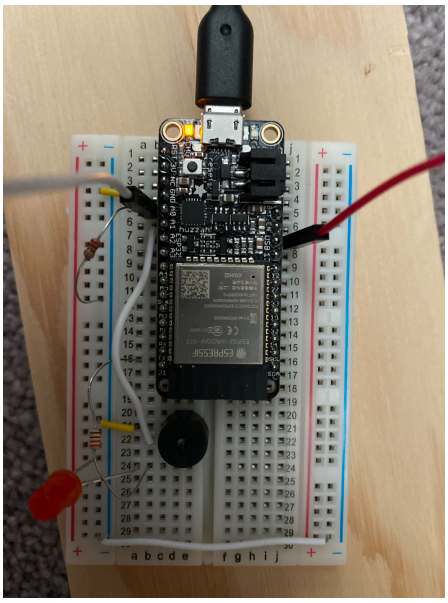


Figure 2b: Hardware Design, First Iteration (Receiver Side)

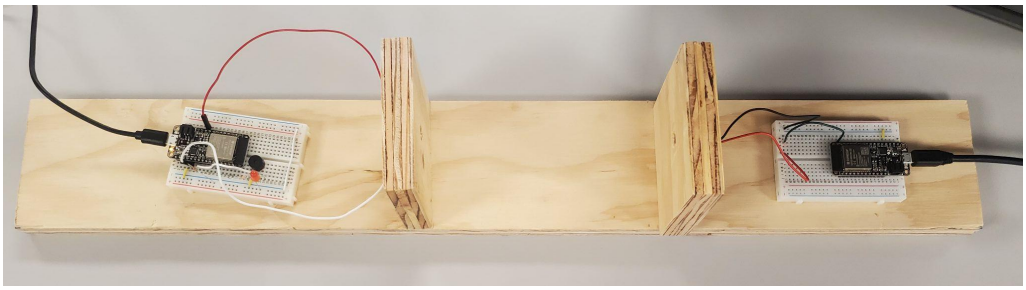


Figure 3: Hardware Design, Final Version

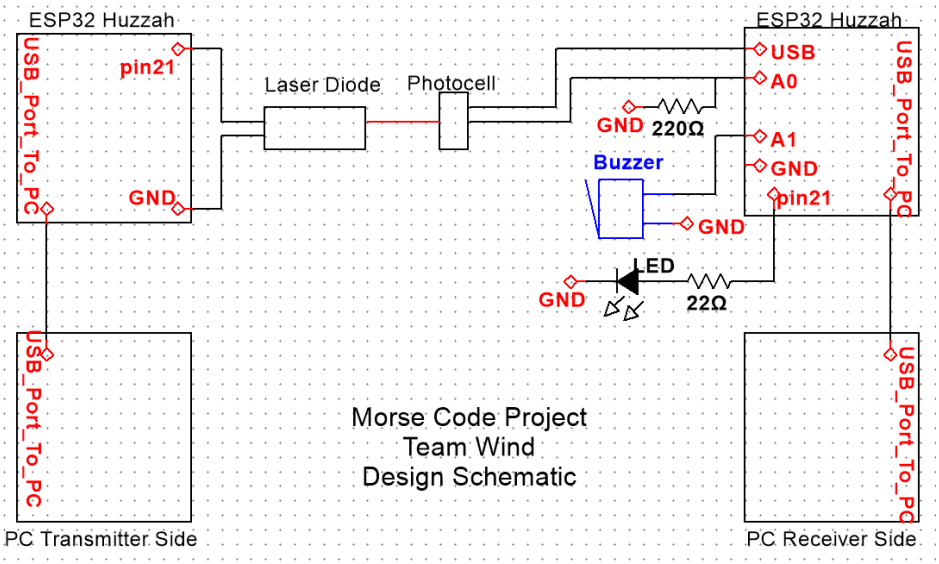
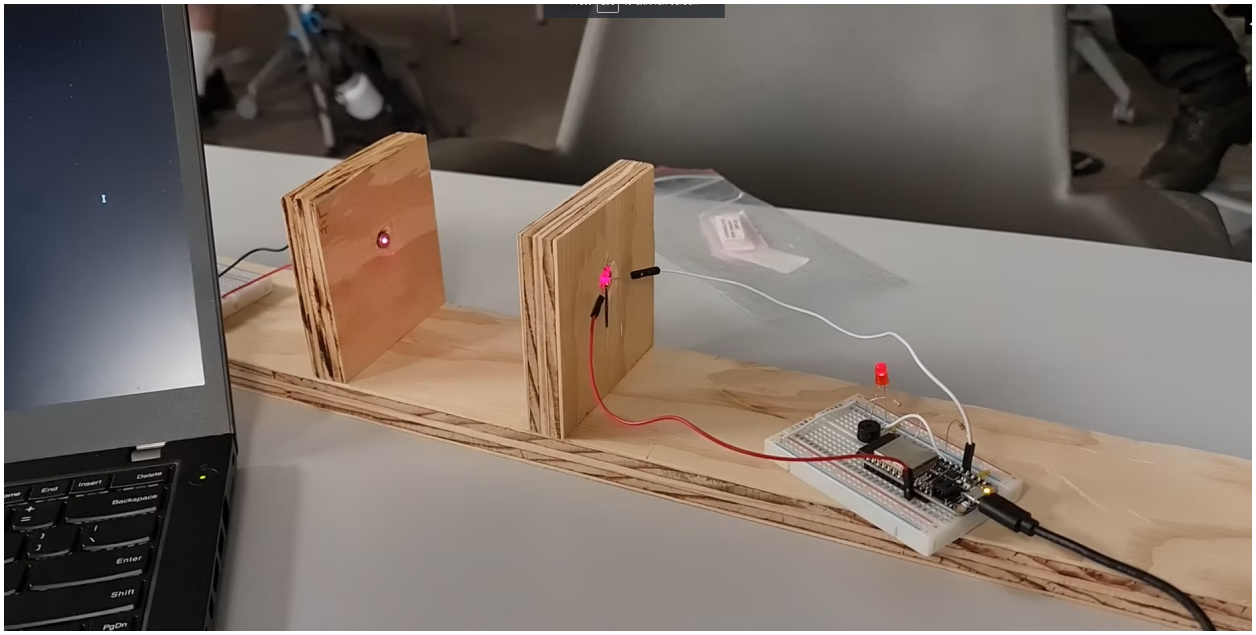


Figure 4: Design Schematic



Video of the Working Project: https://vimeo.com/manage/videos/739033125