ESSE 2220 – Lab 1 Report

Course: ESSE 2220
Lab Title: Raspberry Pi LED Morse Code
Date Performed: 2025-09-12
Date Submitted: 2025-09-12

---

1. Names & Group Info

Group Number: GROUP 5
Members: Yathharthha Kaushal · Owen Oliver

---

2. Circuit Setup

GPIO Pin Used for LED: `GPIO 26`

We connected the LED (with resistor) to pin 26 and ground so the Raspberry Pi could control it through Python.

Schematic / Wiring Notes:

- Anode (long leg) → Series resistor (212 Ω) → GPIO pin.
- Cathode (short leg) → Ground (GND).
- We used BCM because it is less hardware dependent, and works across different pi's as well (so if this program is ever to be used later on a different device, it will likely work).

---

3. Code (with Comments)

In [ ]:
```python
import RPi.GPIO as GPIO #type: ignore
import time
GPIO.setmode(GPIO.BCM)

# Global Variables / Data
studentNumbers = ["221116678", "221430194"]
unit = 0.2
shortDelay = 1*unit
longDelay = 3*unit
varDelay = 7*unit
LED_PIN = 26  # Define LED pin globally

# Morse code dictionary for alphanumeric characters (uses map and key concept from Java course)
MORSE_CODE = {
    'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.', 'F': '..-.',
    'G': '--.', 'H': '....', 'I': '..', 'J': '.---', 'K': '-.-', 'L': '.-..',
    'M': '--', 'N': '-.', 'O': '---', 'P': '.--.', 'Q': '--.-', 'R': '.-.',
    'S': '...', 'T': '-', 'U': '..-', 'V': '...-', 'W': '.--', 'X': '-..-',
    'Y': '-.--', 'Z': '--..',
    '0': '-----', '1': '.----', '2': '..---', '3': '...--', '4': '....-',
    '5': '.....', '6': '-....', '7': '--...', '8': '---..', '9': '----.',
    ' ': '  '  # Space between words
}

"""Setup function to configure GPIO pins"""
def setup():
    # Configure the LED pin as output
    GPIO.setup(LED_PIN, GPIO.OUT)

"""Convert student numbers to Morse code"""
def convertDataToMorse():
    morseArr = []
    for studentNumber in studentNumbers:
        morse_string = ""
        for char in studentNumber:
            if char.upper() in MORSE_CODE:
                morse_string += MORSE_CODE[char.upper()] + " "  # Add space between characters
        morseArr.append(morse_string.strip())  # Remove trailing space
    return morseArr

"""Functions to control the LED pin"""
def pinOn(duration):
    GPIO.output(LED_PIN, GPIO.HIGH) #type: ignore
    time.sleep(duration)
    GPIO.output(LED_PIN, GPIO.LOW) #type: ignore
```

```
"""Turn off the pin for a specified duration"""
def pinOff(duration):
    GPIO.output(LED_PIN, GPIO.LOW) # type: ignore
    time.sleep(duration)

"""Drive the Morse code to blinks on the LED"""
def driveMorseToBlinks(morseInputArr):
    for i, morseInput in enumerate(morseInputArr):
        print(f"Playing student number {i+1}: {studentNumbers[i]} -> {morseInput}")
        for c in list(morseInput):
            if c == '.':
                pinOn(shortDelay)
                pinOff(shortDelay)  # Short delay after dot
            elif c == '-':
                pinOn(longDelay)
                pinOff(shortDelay)  # Short delay after dash
            elif c == ' ':
                pinOff(longDelay)  # Space between characters (no additional delay needed)

        # Add delay between different student numbers
        if i < len(morseInputArr):
            print("Delay between student numbers...")
            pinOff(varDelay)

def destroy():
    GPIO.cleanup()
```

```
if __name__ == '__main__':    # Program entrance
    setup()
    try:
        while True:
            driveMorseToBlinks(convertDataToMorse())
    except KeyboardInterrupt:   # Handle Ctrl+C gracefully
        print("\nProgram interrupted by user")
    except Exception as e:   # Handle other exceptions
        print(f"An error occurred: {e}")
    finally:
        destroy()
```

---

4. Logic Explanation (Step-by-Step)

Our program implements Morse code transmission using a Raspberry Pi's GPIO to control an LED, following these steps:

1. We store Morse code patterns in a dictionary for all alphanumeric characters, with timing based on a 0.2s unit (dots = 1 unit, dashes = 3 units).
2. The program hardcodes two student ID numbers in the `studentNumbers` array and initializes the LED pin (GPIO 26) as an output.
3. For each student number, the `convertDataToMorse()` function translates each digit into its corresponding Morse sequence (dots/dashes with spaces).
4. The `driveMorseToBlinks()` function processes each Morse character: blinking the LED briefly for dots (0.2s), longer for dashes (0.6s), with appropriate pauses between elements.
5. The program prints the current student number and its Morse representation to the terminal before blinking, with a longer delay (1.4s) between transmitting different student numbers.
6. The main loop continuously cycles through all student numbers until interrupted, and the `destroy()` function ensures GPIO resources are properly released when the program exits.

---

5. Output (Run Evidence)

5.1 Terminal Output

```
Playing student number 1: 221116678 -> ..--- ..--- .---- .---- .---- -.... -.... --... ---..
Delay between student numbers...
Playing student number 2: 221430194 -> ..--- ..--- .---- ....- ...-- ----- .---- ----. ....-
Delay between student numbers...
Playing student number 1: 221116678 -> ..--- ..--- .---- .---- .---- -.... -.... --... ---..
```

5.2 Photo of Physical Circuit

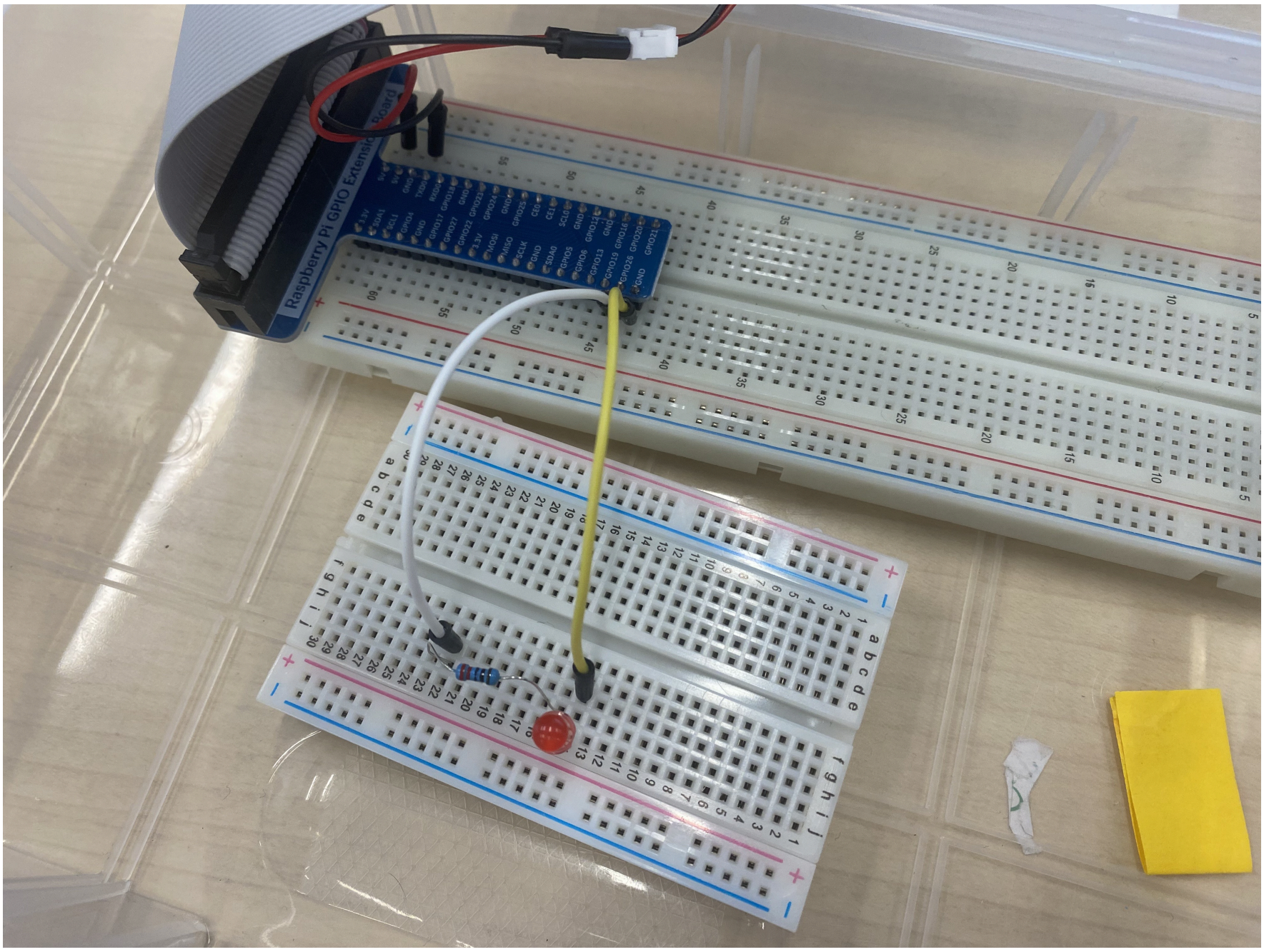

Figure 1 - Lab 1 circuit

---

6. Reflection

- What did you learn about Python (loops, functions, timing, GPIO usage)?
  I learned how to use counter variables in python loops, and how arrays work in python, and that python has no switch statements :(, and (accidentally) learned about using objects in python through our morse library that we used earlier (but it's not present in the final version, since we couldn't get internet on the pi to run `pip install morse3`)
- What was easiest? Coming up with the logic/pseudocode for this lab, as well as the actual connections for the rpi and the breadboard.
- What was hardest or most confusing?
  It was a little bit harder moving from java to python - many core concepts worked in the same way logically like arrays, and loop iteration counters, however, python had a much different syntax than the conventional java/c++ syntax that I had to learn to implement the same thing.
- One improvement you would add with more time (e.g., user input for speed, repeating message, second LED, buzzer, error handling).
  Not sleeping the thread for the delay, since that prevents the program from performing other actions simultaneously, if we wanted the program to simultaneously be able to do other things as well, we would use multithreading and be able to run the program without having to put the main thread to sleep whenever setting a delay on the light - this would also make it better for use as a mcu, as the typical mcu usage requires multiple operations being done simulataneously by the core program (for example recieving and transmitting values from multiple sensors, controlling other hardware devices (for example magnetorquers, propellants, servos, etc.))
- Any issue you troubleshot (and how you fixed it).
  There were multiple issues that I had to troubleshoot:
  - We couldn't import morse3
    - We tried installing that library with `pip install morse3`, however there was no internet connectivity on the pi
    - We tried connecting the pi to wifi, it still had network as unreachable for some reason
    - We ended up just a HashMap (We remembered that from java course) based approach and defined key/vaule pairs for morse/alphanumeric and then translated the student number string into morse in a method.
  - I forgot to set LED_PIN's, and the delay variables' scope to global
    This was easily fixed by removing them from the setup method, and putting them directly below the imports.
  - There was a syntax error, there was a > typed instead of a .
    This was a relatively trivial fix - once there was a compilation error, I found where the > was, and fixed the type

---

7. In-Class Demonstration Checklist

- [✔] Python script runs without errors.
- [✔] LED blinks full student ID in Morse code correctly.
- [✔] Code commented and readable.
- [✔] GPIO cleaned up (no warning messages).

End of Report.