**5-1 Milestone Three: Input With Buttons Lab**

Allan O'Driscoll

Southern New Hampshire University

CS 350: Emerging Systems Architectures & Technologies

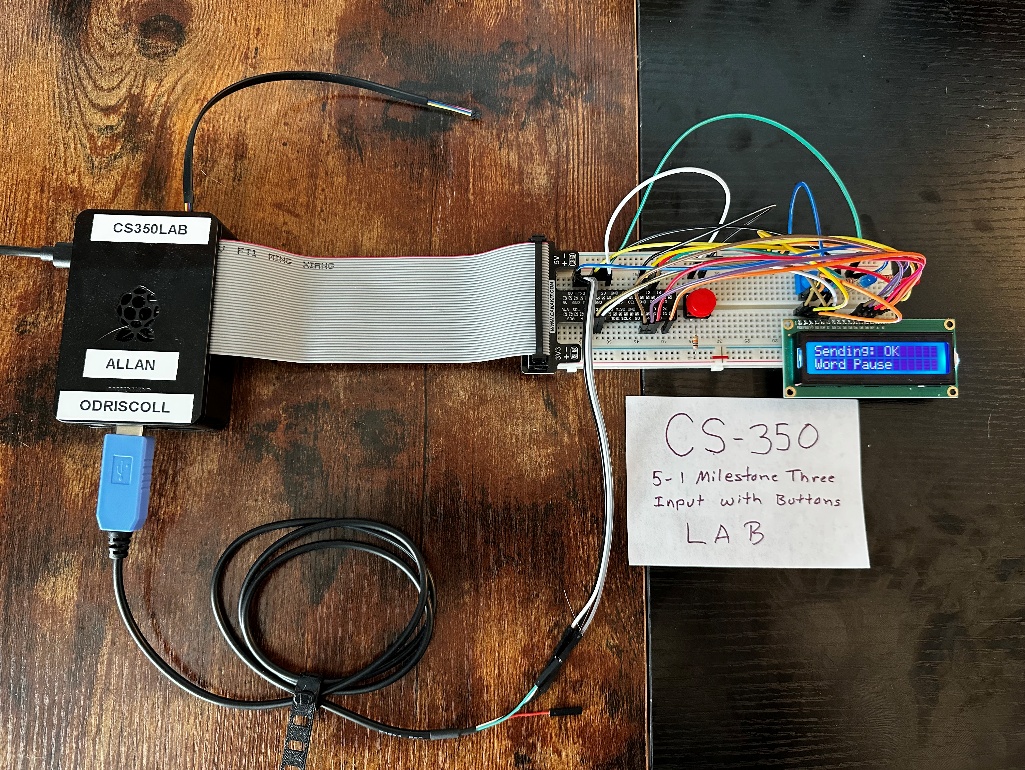
Professor Roland Morales

April 5, 2025

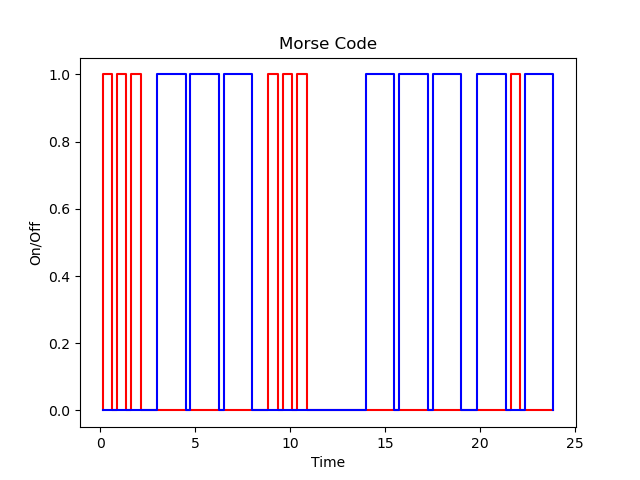
**5-1 Milestone Three: Input With Buttons Lab**

The purpose of this assignment is to develop the logic to transmit a message in Morse code using a red light to represent dots and a blue light to represent dashes. Appropriate pauses are inserted between dots and dashes, letters, and words. Two messages are hard-coded into the program. These are "SOS" and "OK". The message can be toggled by clicking a button that changes the active message but does not interrupt the current transmission. The active message is translated to Morse code in a set of loops. The transmission will be repeated until the message changes (via a button click) or the user cancels execution by clicking on CRTL-C.

The following is an image of my setup:



The image on the following page is a graphical representation that shows the transmission of the two messages, first “SOS” and then after a pause “OK”:

****

This graph was created by temporarily instrumenting the code as follows:

import pandas as pd

import matplotlib.pyplot as plt

…

records = []

…

class CWMachine(StateMachine):

…

def on\_enter\_dot(self):

if(DEBUG):

now = time.time()

records.append({"time": (now-start\_time), "red": 0, "blue": 0})

records.append({"time": (now-start\_time), "red": 1, "blue": 0})

def on\_exit\_dot(self):

if(DEBUG):

now = time.time()

records.append({"time": (now-start\_time), "red": 1, "blue": 0})

records.append({"time": (now-start\_time), "red": 0, "blue": 0})

…

def on\_enter\_dash(self):

if(DEBUG):

now = time.time()

records.append({"time": (now-start\_time), "red": 0, "blue": 0})

records.append({"time": (now-start\_time), "red": 0, "blue": 1})

…

def on\_exit\_dash(self):

if(DEBUG):

now = time.time()

records.append({"time": (now-start\_time), "red": 0, "blue": 1})

records.append({"time": (now-start\_time), "red": 0, "blue": 0})

…

df = pd.DataFrame(records)

# Create a plot

plt.plot(df['time'], df['red'], color='red')

plt.plot(df['time'], df['blue'], color='blue')

plt.xlabel("Time")

plt.ylabel("On/Off")

plt.title("Morse Code")

# Save the plot as a PNG image

plt.savefig("morse.png")

**Why does the loop that processes the LED blinking need to run in a separate thread?**

The transmit method is designed as a loop that cycles through the message one character at a time, translating the words into Morse code. The code interacts with the state machine by sending messages. These internal processes need to be on separate threads to allow each to execute their activities independently from the others. The timing needs to be fairly precise, and some methods block while performing their function. Without a proper threading model, this behavior would interfere with other activities. In addition, doing this work on the main thread while trying to do anything else would complicate the code. There are several other threads that perform different functions, such as blinking the LED and monitoring the state of the button. While the program is running, the following threads can be seen executing:

[

<\_MainThread(MainThread, started 281473712650016)>,

<\_callback\_thread(Thread-6, started daemon 281473349382528)>,

<Thread(Thread-7 (transmit), started 281473317925248)>,

<HoldThread(Thread-8 (held), started daemon 281473162736000)>,

<GPIOThread(Thread-22 (\_blink\_device), started daemon 281473152250240)>

]

**What is the purpose of returning to the off state after each completed state action?**

The system needs to be able to transition between a set of pre-defined states. If a transition from the current state is not defined for a given action, then a TransitionNotAllowed exception will be thrown. Consequently, each action needs to leave the state machine in a state that is valid for the next set of possible actions. In this case, each action begins by assuming that the machine is in the off state. Thus, each action needs to return to this state upon completion. This simplifies the code as well because each state action has a valid starting point. Otherwise, each action would have to know how to transition from every other state. This would get complicated very quickly.

**How could you integrate serial communications to facilitate changing the messages available to the program?**

It would be simple to integrate serial communications into the state machine to facilitate message changes. The program would simply need to read the message from the serial interface, as we did in the lab from module four, and then change the active message. A client application would send messages through the serial port, and a server application would read them. The server would also manage the transmission of the message in Morse code via the GPIO. The project is currently set up to toggle between two messages, but with some minor changes, it could be modified to transmit arbitrary messages. The program would only need to adjust the active message with the message received so that it would be queued for the next transmission. Reading from the serial port would need to be handled on a different thread so that it wouldn't interfere with the transmission.

**How could you use the 16x2 display to provide debugging information to the user when they don’t have access to the application console?**

The 16x2 display is used to display short messages to the user. During normal operation, you would likely only display functional messages. However, if the application is running in DEBUG mode, then you can display any additional information that is helpful for understanding the program's operation. For example, you might display a message when the button is clicked to let the user know what the next message will be. You can also show informational messages about what is happening at any given moment, such as the character or word that is being transmitted. This extra information is useful for troubleshooting problems but can be easily disabled by switching the program out of DEBUG mode.