Input With Buttons Lab Questions

Derricko Swink

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

The LED blinking needs to run in a separate thread because it is a blocking operation (it uses time.sleep() and loops while the LED is on/off). If this was done in the main thread, it would block the program from listening to button presses or responding to other inputs. By moving the LED blinking to a separate thread:

- The main program remains responsive.
- State transitions can still be detected while blinking is in progress.
- It ensures the user interface (buttons/display) doesn't freeze while visual output is happening.

This is essential for embedded systems where real-time input handling is critical.

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

Returning to the STATE_OFF after each completed action allows the system to:

- Reset the state machine and prepare for the next user input.
- Ensure the system is in a known, idle state between actions.
- Prevent accidental re-triggering of the same state if the button is pressed too long or multiple times.

This design creates a clear start and stop point for each state action, simplifying control flow and debugging.

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

You can use serial communication (e.g., UART or USB via /dev/ttyS0) to send new messages to the Raspberry Pi.

A general approach to this is:

- Connect to the Pi using a serial terminal (for example... minicom, PuTTY, or a Python script using pyserial).
- In the program, set up a serial listener that:
    - Waits for input from the user.
    - Parses the input into a valid Morse code message or text.
    - Updates the messages list or a currently selected message.
- Example addition using pyserial:

```
import serial

ser = serial.Serial('/dev/ttyS0', 9600)
if ser.in_waiting:
   incoming = ser.readline().decode().strip()
   messages[0] = incoming  # update first message
```

This would allow users to dynamically change messages without modifying the source code or restarting the program.

4. 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 LCD display can act as a mini debug monitor. You can display information like:

- Current state (for example... "STATE: BLINK")
- Current message being blinked
- Button press detected
- Error messages or invalid inputs
- Thread status ("Thread Running" or "Idle")

Here's a practical example using lcd_display_string():

```
lcd.lcd_display_string(f"State: {state}", 1)
lcd.lcd_display_string("Msg: " + messages[state-1][:16], 2)
```

This allows users who don't have access to the console (e.g., running headless or in a classroom setting) to understand what the program is doing and debug more easily.