CS 350 – Milestone Three Lab Questions

Mohamed Aziz Zaghdoudi

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

The loop that processes the LED blinking needs to run in a separate thread to allow the main program to remain responsive to user input and other system events, such as button presses. By running the blinking logic in its own thread, the program can handle LED timing and state transitions independently, without blocking the rest of the application. This ensures that actions like detecting a button press to change the Morse code message can happen immediately, rather than waiting for the blinking loop to complete. Multithreading is especially important in embedded systems where both real-time hardware control and user interaction are required simultaneously.

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

Returning to the off state after each completed state action (dot, dash, or pause) ensures that the LEDs are only illuminated for the exact duration needed for each Morse symbol, and that the circuit is idle between blinks. This makes the Morse code output clear and unambiguous, matching the required timing standards. The off state acts as a reset point, making it easier to coordinate transitions between different actions and maintaining a consistent rhythm. It also prevents LEDs from staying on longer than intended, which could make the Morse code difficult to interpret.

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

Serial communications can be integrated by setting up a serial interface (such as UART, USB-to-serial, or even over a network using sockets) that allows an external device (like a computer or another microcontroller) to send new messages to the program. The program can listen for incoming serial data, parse it, and update the list of available Morse code messages in real time. For example, a user could type a new message into a serial terminal on their computer, and the Raspberry Pi could receive and store this message to be blinked out using the LEDs. This would make the system much more flexible and interactive, as messages could be changed without modifying the code or pressing a physical button.

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 display can be programmed to show important debugging information directly to the user, such as the current state of the state machine, which message is being transmitted, or whether a button has been pressed. For example, the display could show the current Morse code symbol being sent, indicate when a message change has been detected, or display error messages if an invalid input occurs. This feedback helps users monitor the system’s operation in real time, even if they can’t see the application’s output on a console or computer screen. It’s especially helpful in embedded or remote deployments where console access is not practical.