# Victor Udeh Cs350 Module 5 Milestone 3

13th Sept, 2024

**Timer and Interrupts**

1. **How the Timer and Interrupts Were Used**

The **timer** in my system is responsible for managing time-based transitions between Morse code elements (dots, dashes, and breaks). It is configured to fire every 500 milliseconds (ms), triggering a callback function (timerCallback()) that controls the LED states, ensuring the Morse code is accurately displayed. The timer period is set to 500,000 microseconds (500ms) to match the duration of each Morse code element. For example, dots last for 500ms, and dashes last for 1500ms.

The **interrupts** are used for detecting button presses. The button is configured to trigger an interrupt when pressed, calling the gpioButtonCallback() function. This allows the user to switch between displaying the "SOS" and "OK" Morse code messages. The button press does not immediately interrupt the current message. Instead, the system waits for the current message to finish before switching to the next one.

**Code Reference**:

* **Timer Setup**: The timer is initialized in the initTimer() function with a 500ms period.
* **Button Interrupt**: The gpioButtonCallback() function detects button presses and toggles the message between "SOS" and "OK".

**Purpose and Structure of the State Machine**

The **purpose** of the state machine is to control the blinking of the LEDs to display Morse code messages. It ensures that each element of the Morse code (dots, dashes, and breaks) is displayed for the correct duration, and that transitions between Morse code elements are handled smoothly. The state machine also manages the transition between the two messages ("SOS" and "OK"), allowing for message switching without interrupting the current message.

The **structure** of the state machine consists of:

* **LED ON (Red)** for dots.
* **LED ON (Green)** for dashes.
* **LED OFF** for breaks between dots, dashes, characters, and words.
* **SOS Mode** for the "SOS" message sequence.
* **OK Mode** for the "OK" message sequence.

The state machine uses arrays (sosMessage[] and okMessage[]) to define the Morse code patterns for each message. Transitions between states are triggered by the timer, which drives the LED blinking, while button presses toggle between messages.

**Code Reference**:

* The timerCallback() function handles the state transitions for the Morse code elements.
* The gpioButtonCallback() function toggles between the "SOS" and "OK" messages.

**Challenges Faced and How I Overcame Them**

* **Challenge 1: Timing Accuracy**  
  **Problem**: Ensuring that the LEDs blink for the correct durations to represent dots, dashes, and breaks accurately.  
  **Solution**: I adjusted the timer to fire every 500ms, which allowed me to control the duration of the dots (500ms) and dashes (1500ms). By carefully managing the timer and testing different intervals, I ensured that the LED blinking matched the correct Morse code timing.
* **Challenge 2: Handling Button Presses Without Interrupting Messages**  
  **Problem**: The button press needed to switch messages without interrupting the current Morse code message.  
  **Solution**: I used the button interrupt (gpioButtonCallback()) to detect the button press, but ensured that the message only switched after the current Morse code message finished. This was achieved by updating the BUTTON\_STATE only after the current message completed.
* **Challenge 3: Implementing the State Machine**  
  **Problem**: Ensuring that the state machine properly controlled the LEDs and handled transitions between dots, dashes, and breaks.  
  **Solution**: I implemented arrays (sosMessage[] and okMessage[]) to store the Morse code patterns. The state machine processed these arrays, ensuring that each LED blinked correctly according to the Morse code rules.

**Code Reference**:

* The messageCounter variable tracks the current position in the Morse code sequence, and the state transitions are handled by timerCallback().