Kiana Peterson and Nattapon Oonlamom
CSE 474: Intro to Embedded Systems
November 30th, 2023
Lab 4: LCD Display, Touch Screen, and RTOS

Procedure

The purpose of this lab is to interface the TM4C LaunchPad to display graphics on LCD and create virtual buttons with the touch screen, then using both bare-metal programming and FreeRTOS.

Task 1a: This task was to get the LCD to work. We connected the LCD to the board as shown in Figure 1. We downloaded the SSD2119.zip and followed the TODO sections as instructed. We followed the comments provided under LCD_GPIOInit() to initialize the ports. After all the initializations, we tested the LCD by calling the LCD_ColorFill() function, assigning and declaring the color in SD2119_Display.c.

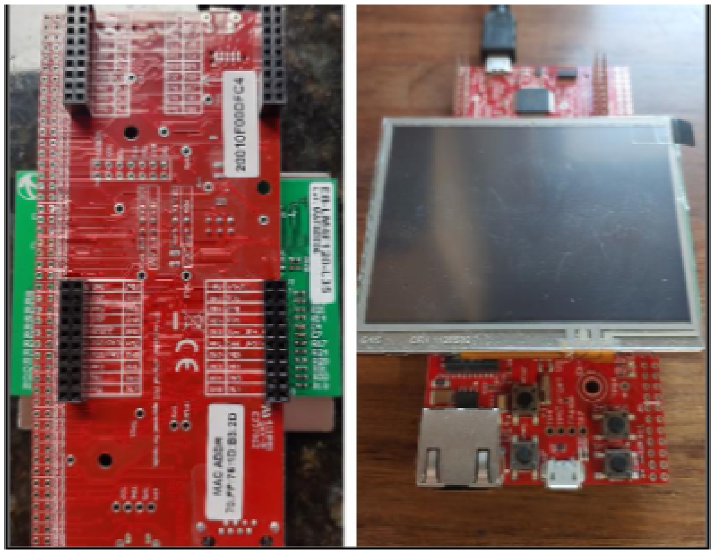


Figure 1: LCD connected to TIVA Board

Task 1b: For this task, we were to use the code from Lab3 Task 1b to read the temperature from on-board sensors, then display the temperature reading at the top of the LCD display. As instructed, we kept the functionalities the same. To approach the new requirements, we began by initializing all the temperature reading modules from lab 3 and defined the frequency for the PLL. Debugging through trials and errors, we took the code from Lab 3 Task 1b and modified the printing functions to use the LCD_Print() and LCD_PrintFLoat() functions to monitor the output temperature values on the LCD screen rather than the Terminal I/O.

Task 1c: The task was to create two virtual buttons to replace the on-board buttons using the LCD touch screen functions. We were to repeat Task 1b with virtual button controls, with a requirement that the buttons and the temperature reading must always stay displayed. To approach this, we initialized the virtual buttons using Buttons_Init(), then used an if/else to define the frequency speed for each. (SW1 = 12MHz and SW2 = 120MHz). Debugging through trials and errors, we then redesigned the main 1c conditions to take in the virtual button instead of the external buttons. Lastly, we kept the output mechanism from 1b the same.

Task 2a: For this task, we implemented a virtual traffic light controller. We used the virtual buttons from Task 1c and instead of using external LEDs, we created virtual traffic lights on the LCD display. As instructed, we kept the functionality the same as described in Lab 2 Task 1. We also made sure that our system was running at 60MHz for this task. Our approach was to use the FSM from Lab 2 Task 1b, but reassign the button inputs to the virtual button and the LEDs output to the virtual traffic lights on the LCD. The former state diagram can be found in Figure 2.1 and the reorganized one in Figure 2.2. To begin, we used the virtual buttons created in 1c without any assigned frequency speed and PLL. For the virtual lights, we initialized the Lights_Init() module, where we used LCD_DrawFilledCircle() to draw the lights and position them accordingly. We then initialized Light_On() and Light_Off() to assign colors to the virtual lights, giving a parameter color that can be used with LCD_DrawFilledCircle(). We then replaced these helper functions of virtual lights and buttons as new inputs and outputs for the FSM. Additionally, we found faults in the Lab 2 Task 1b FSM timing system, so we modified the FSM using a new pointer next and added the initialized state to store initial values of the pointers and button. The button makes sure all inputs are checked. The boolean next returns true if the FSM is ready to move to the next state. Lastly, we put all of these components together, debugging using the given features until we are satisfied with the results.

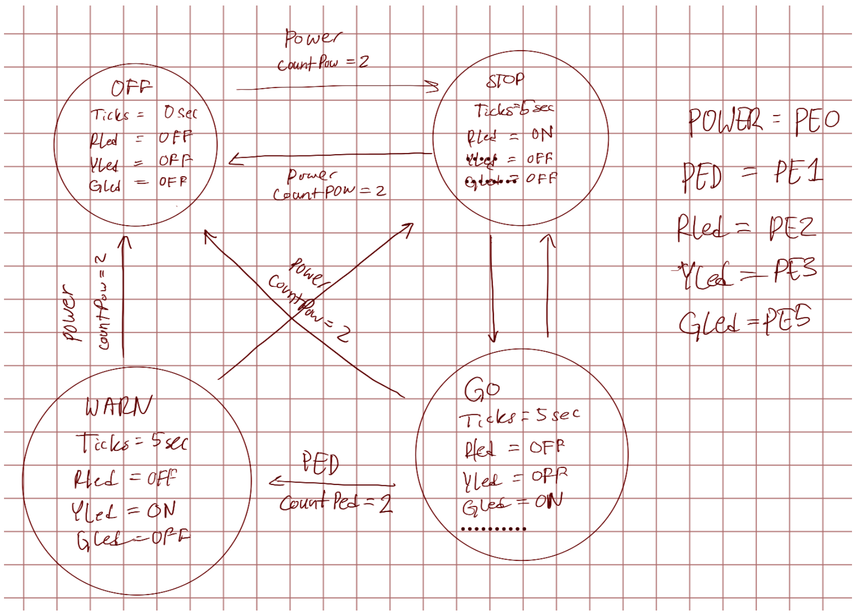


Figure 2.1: Old State Diagram of Traffic Light Controller (Lab 2 Task 1b)

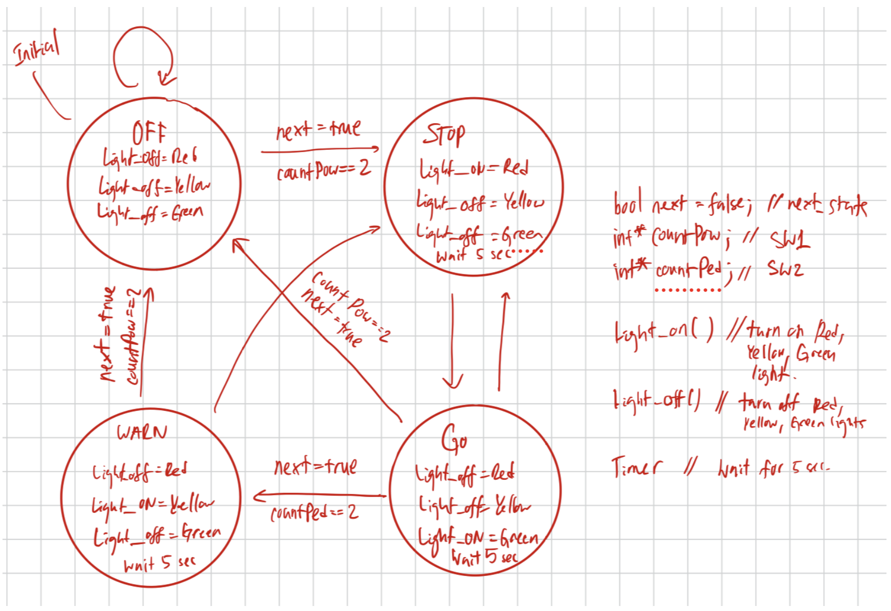


Figure 2.2: Updated State Diagram of Traffic Light Controller (Lab 4 Task 2a)

Task 2b: We implemented the traffic light controller from Task 2a with a real-time operating system (RTOS). We downloaded and installed SW-TM4C-2.1.4.178.exe as illustrated in the lab document. Following the installation directions from part 1, we replaced the freertos_demo.c with the main.c provided and changed configCPU_CLOCK_HZ to 60000000 in FreeRTOSConfig.h. After finishing the installation process as directed, we read carefully through the main.c file to understand its structure. Throughout the process, we did not use interrupts or timers and ensure that the system was running at 60 MHz at all times as suggested. With all the requirements in mind, we carried on to part 2 of the task to complete the skeleton code. We approached it by following all the steps given in the provided main.c file. We used all the same initialization from 2a and called for the same header files. We then look at the FreeRTOSConfig.h to help with assigning priorities to the tasks.After having done so, we carried on the finish the if/else conditions and flagging system for the on/off_status and pedestrian_status. After, we initialized the states using enum before putting in the FSM from 2b. We took out the timer mechanism and reassigned the if/else conditions with the global flags on/off_status, pedestrian_status, and current/previous timing pointers accordingly. We ended up encountering a hardware faulty problem, so we debugged using the given IAR features, trials and errors, and even provided lecture notes.

Result

Task 1a: The LCD screen is filled with color as it should display.

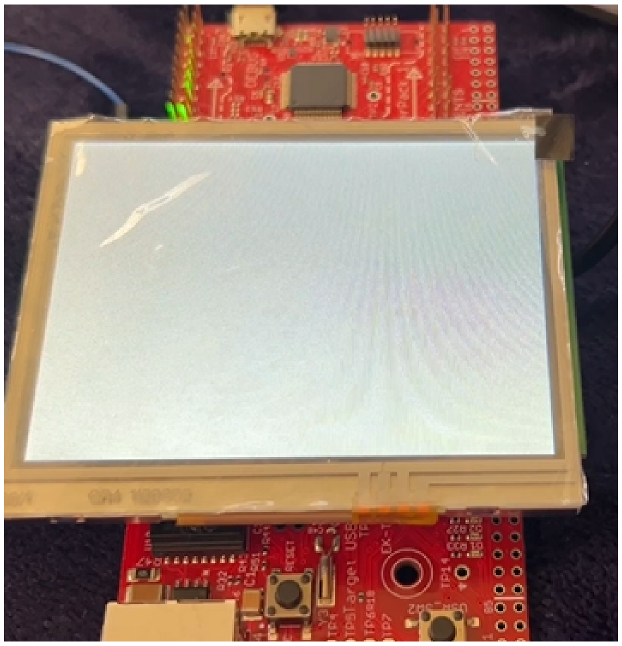
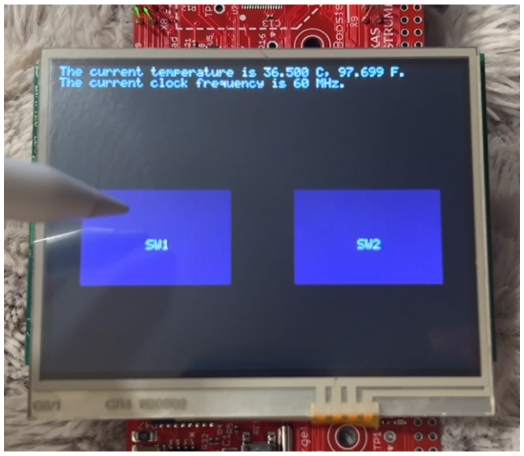


Figure 3: Working LCD Display

<u>Task 1b:</u> The temperature readings from the board's internal sensor in both Celsius and Fahrenheit are displayed on top of the LCD screen. The values update as the buttons are pressed. The same functionalities from Lab3 Task 1b remained. When the120MHz SW is pressed, the temperature output on the LCD increases as the system heats up while when the 12MHz SW is pressed, the printed temperature output decreases as the system cools down.

<u>Task 1c:</u> The LCD display displayed virtual buttons (SW1 and SW2) that function as expected when pressed by the stylus. Every time the virtual switch assigned with 120MHz (faster frequency) is pressed, the printed temperature output on the LCD increases as the system heats up while when the virtual switch assigned with 12MHz (slower frequency) is pressed, the printed temperature output on the LCD decreases as the system cools down.



```
The current temperature is 36.500 C, 97.699 F.
The current clock frequency is 60 MHz.
```

Figure 3.1: Working LCD Display with Functional Virtual Buttons

<u>Task 2a:</u> As a result, the LCD display shows the virtual buttons and the 3 traffic lights (red, yellow, and green) that function as required. SW1 is the On/Off button and SW2 is the Pedestrian button. When SW1 is pressed

and held for approximately 2 seconds, the system begins at the stop state, where only a red virtual traffic light lights up on the LCD screen. After 5 seconds, the system moves to the go state, where only a green virtual traffic light turns on. This happens indefinitely when no button is pressed. However, when on/off (SW1) is pressed and held for at least 2 seconds, then the system turns off where none of the virtual traffic lights are on. However, when SW2 is pressed for approximately 2 seconds while in go state, the system immediately goes to warn state and remains for 5 seconds, where only a yellow virtual traffic light turns on, before moving to stop and repeating the pattern.

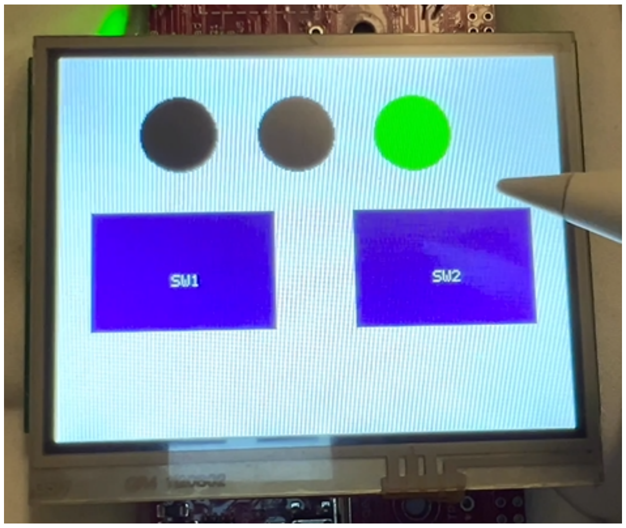


Figure 3.2: Working Virtual Traffic Light Controllers

<u>*Task 2b:*</u> Although this task used RTOS, it has the *same result as Task 2a as expected* and checked all the requirements! The virtual traffic lights system functioned as required as described under Task 2a results.