ECE/CSE 474:
Intro to Embedded Systems

Lab 4: LCD Display, Touch Screen, and RTOS

Introduction

In Task 1 of Lab 4, you'll use everything you have learned so far to interface the TM4C LaunchPad with a touch screen display assembly. With the LCD, you'll create graphics and display messages. Using the touch-screen functionality, you'll create virtual buttons on the display to replace physical buttons. In Task 2, you'll use the touch screen display to create a simple graphical user interface for the traffic light controller and implement it using both bare-metal programming and FreeRTOS.

By the end of this lab, you will:

- Know how to interface with the LCD to display information and graphics.
- Know how to create a user interface using the touch screen.
- Have experience with programming using FreeRTOS.
- Become even better at extracting information from a datasheet

Task 1 - LCD Driver

LCD Hardware

For this lab, we're using the Kentec Display EB-LM4F120-L35 touchscreen LCD. It's a 3.5″ LCD touch screen module designed to be used with Stellaris LaunchPads, with a resolution of 320×240 pixels. To begin, plug the LCD female headers into the male headers of the BoosterPack 2 on your TIVA board, as shown in Figure 1 and Figure 2.

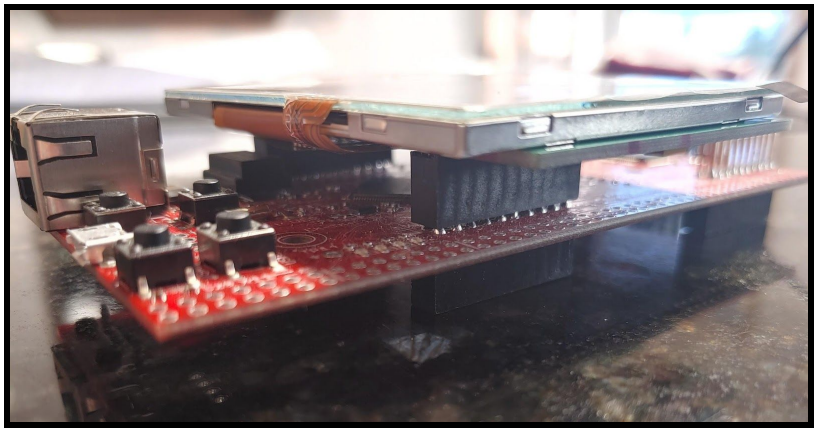

Figure 1: LCD Connected to the TIVA LaunchPad (side view)

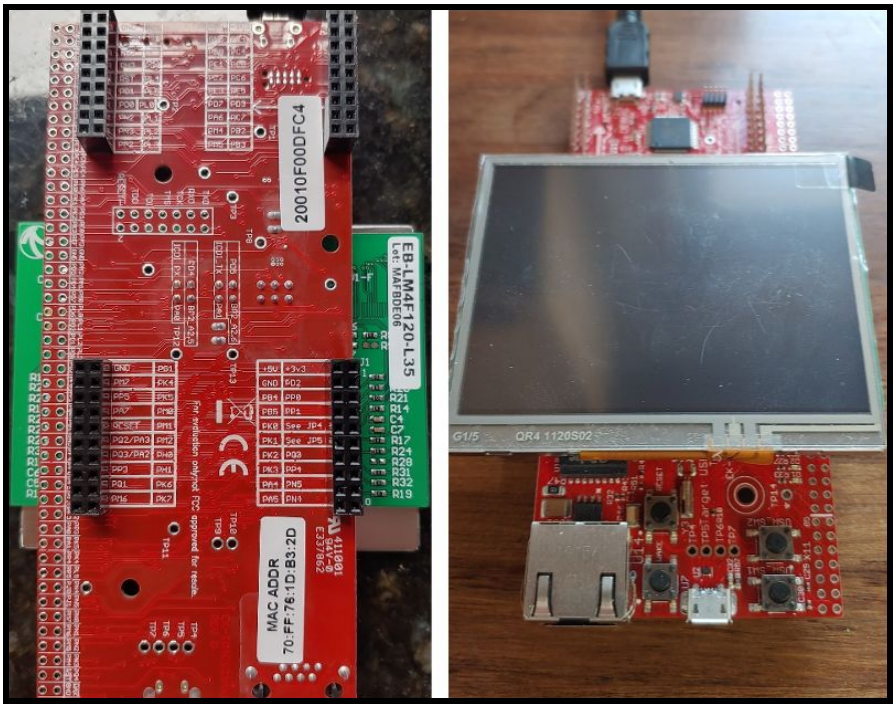

Figure 2: LCD connected to the TIVA LaunchPad (front view left, back view right)

LCD Software

On the course's Canvas webpage, you will find files to drive the LCD display. There's a lot of code there – some of it you'll need to fill in, some of it you'll use, some of it you won't use, and some of it doesn't work. How exciting! The format of the driver files provides an excellent example of coding style when leveraging header files. This is the kind of professional driver you might come to expect from a real-world application.

For the first part of the lab, you'll be working with and adding to the device driver code for the LCD. A device driver contains function definitions and appropriate I/O port mappings to ensure the proper functionality of the device. The goal with any driver is to create an easy-to-use interface between the device and other systems. You will find the given header files for the LCD driver extremely helpful. **In particular, SSD2119_Display.h enumerates the functions available for you to use.**

Task 1 Assignment

a. **Complete LCD Driver**

Your first task is to get the LCD driver and display working. To begin, download the SSD2119.zip from the course Canvas webpage. If you are curious, the "datasheet" of the

LCD is available on the Canvas page. You can ignore the column labeled "Launch Pad Pins" of the datasheet; this column references a different TM4C board.

In the SSD2119_Display.c file, fill in the sections marked "TODO" based on the comments found there. There are two functions in particular that you should look into:

- LCD_GPIOInit()
- LCD_PrintFloat()

Note 1: The `LCD_GPIOInit()` *is called by the* `LCD_Init()` *function. This function is very long, and it is sourced from Kentec Display. You don't need to know any of the details of what happens in LCD_Init().*

Note 2: The "TODO" code in `LCD_GPIOInit()` *will be similar to port initializations you've done in previous labs. Just follow the comments that have been provided. Create the corresponding code.*

Note 3: The `LCD_Prinf()` *function cannot print out correct floating point numbers even with the correct* `LCD_PrintFloat()` *function. There is more work to do in order to make the* `LCD_Prinf()` *fully functional, but you are not required to fix it.*

Once you have completed the `LCD_GPIOInit()` function, you should test to see if everything is working. In `main()`, call `LCD_Init()`, and then call `LCD_ColorFill()` with a color of your choice. The "color" input of LCD_ColorFill( ) maps to the unsigned short array "Color4[16]" declared at the top of SSD2119_Display.c. Running the code, you should see the screen filled with color. If it doesn't, try power cycling (resetting) the board.

b. Temperature Readings

In Lab 3, you displayed the temperature of your TM4C over UART to a terminal in PuTTY. Now, display the temperature on the LCD display.

Print the temperature readings in both Celsius and Fahrenheit using the following format:

```
The current temperature is {temp_C} C, {temp_F} F.

The current clock frequency is {freq} MHz.
```

The {temp_C} and {temp_F} are **displayed as floating-point numbers** with at least two decimal places, and the {freq} is displayed as an integer. The LCD should print this message only at the top of the display, and it shouldn't print any extra characters.

Additionally, you need to keep all the functionalities described in Lab3 Task 1b. I.e., you should still be able to change the system clock frequency by pressing the onboard buttons.

Note: In part c, you will call the function Touch_Init() of the LCD driver, which initializes the touch-detection capabilities of the LCD driver. Currently, this function uses the ADC1 module for touch detection. Avoid configuring ADC1 to read the temperature sensor. If you want to use ADC0 for touch detection instead, uncomment line 11 and comment line 12 of SSD2119_Touch.h, which is shown in Figure 3.

```
/*****************************************************************
 * SSD2119_Touch.h
 * The header file of the SSD2119 touchpad driver.
 *****************************************************************/

#ifndef __SSD2119_TOUCH_H__
#define __SSD2119_TOUCH_H__

// Uncomment the corresponding line below to choose the ADC
// module for the touchpad.
//#define TOUCH_USE_ADC0
#define TOUCH_USE_ADC1
```

Figure 3: SSD2119_Touch.h ADC configuration

c. Resistive Touch Screen

Using the touch-screen functionality of the LCD display module, create two virtual buttons on the display to replace the on-board physical buttons. Then, repeat Task 1b using the virtual buttons you created. Label the buttons appropriately, and make sure that they work the same way as the physical buttons. The virtual buttons and the readings should be shown on the display at all times. You will want to use a stylus-like object to operate the resistive touch screen. SSD2119_Touch.h contains the touch-detection functions that you need for this subtask. Note that the value returned by those functions are not the actual coordinates on the display.

Task 2 - Bare-Metal Programming vs Real-Time Operating System

For Task 2, you will implement the Traffic Light FSM on the LCD instead of your breadboard. Once you have successfully implemented the FSM on the LCD, you'll explore how to implement the same FSM using RTOS by following a tutorial.

FreeRTOS

In the previous labs, you used "bare-metal" programming to implement all the functionalities that were required by each application. All the tasks were done through either polling or interrupts. In task 2b, you'll use the Real-Time Operating System to implement the traffic light system. The RTOS provides a predictable execution pattern for time constraint tasks. The execution pattern is typically determined by the RTOS scheduler based on the assigned task priority.

FreeRTOS is a Real-Time Operating System that is designed small enough to run on a microcontroller. It provides core functions such as real-time scheduling, shared resource protection, software timer, inter-task communication. To satisfy different application requirements, FreeRTOS kernel can be easily configured through a configuration file. The "FreeRTOSConfig.h" allows you to customize things such as scheduling preemption, RTOS tick rate, mutex, etc. You will get experience with FreeRTOS by completing the given starter code, implementing your favorite Traffic Light FSM.

Task 2 Assignment

a. Implement a virtual traffic light controller using the display module

Recall the timed traffic light from lab 2. Repeat this task, but instead of using external LEDs and buttons, create virtual traffic lights and virtual buttons on the LCD display. It should have the same functionality as described in task 1 of lab 2. Note that the traffic lights could be as simple as three circles with the corresponding colors. Interrupts are recommended, but not required. Moreover, your system should be running at 60MHz for this task at all times.

b. Implement the Traffic Light Controller using RTOS

Part 1: Preparing running RTOS on the TIVA board

1. Download and install SW-TM4C-2.1.4.178.exe, which can be found under Canvas -> Files -> Labs -> Lab4 -> RTOS. Accept the default path as requested by the installer. You can find this executable on Canvas.
2. Go to the installation directory, then go to examples -> boards -> ek-tm4c123gxl. **(The name of the folder suggests a different board, but that is okay.)** Here you'll find a folder called freertos_demo. This is the folder you'll be working with in this Lab. Figure 4 below shows this directory.

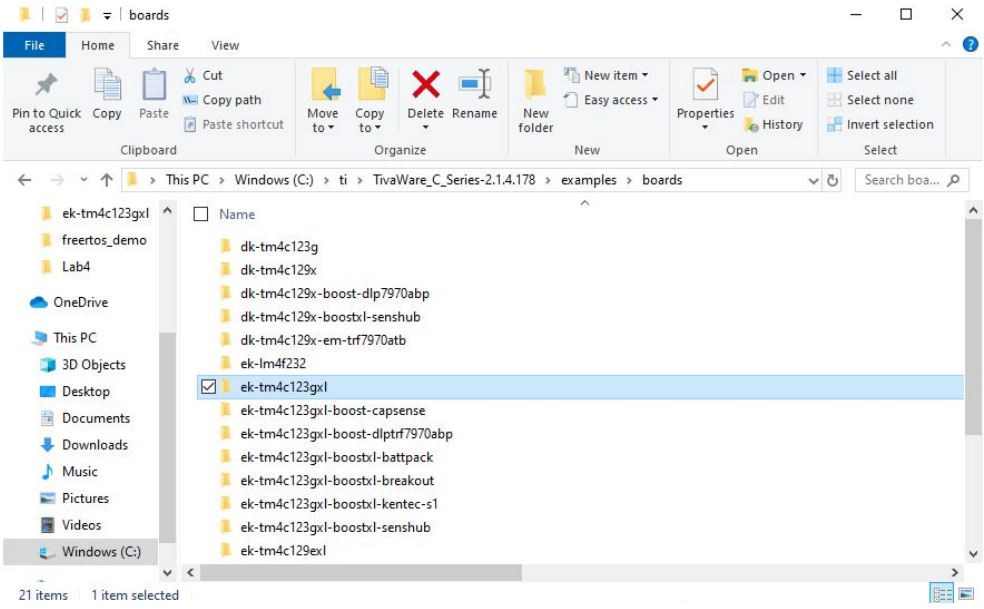

Figure 4: The ek-tm4c123gxl directory

3. Open the IAR Workspace file in the ek-tm4c123gxl directory to open the IAR Project. This file is shown in Figure 5.

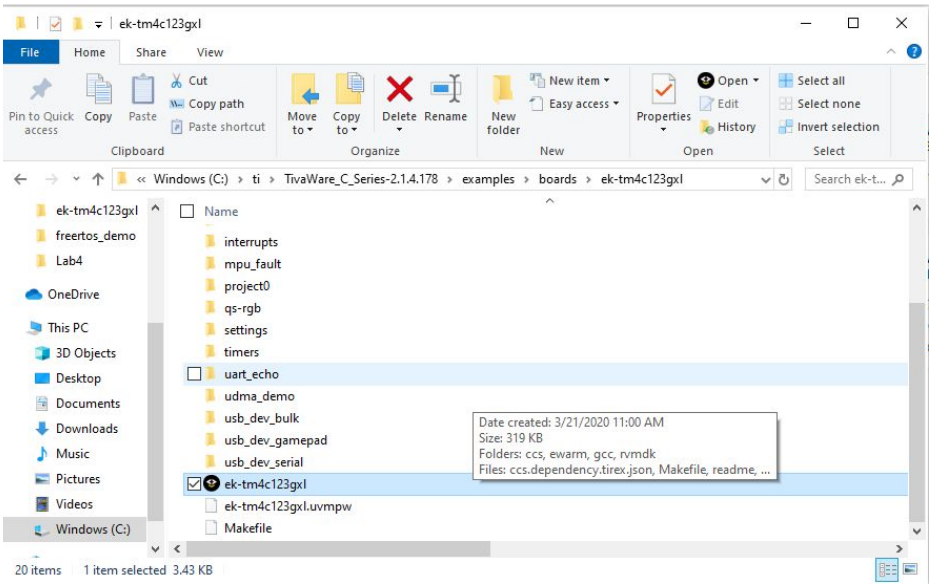

Figure 5: IAR Workbench file inside ek-tm4c123gxl

4. Once you have opened IAR, you'll find a list of subdirectories in the workspace panel. We only need the freertos_demo directory for this task. Click on the freertos_demo at the bottom of the workspace panel to enter the directory, as emphasized by the red arrow in Figure 6.

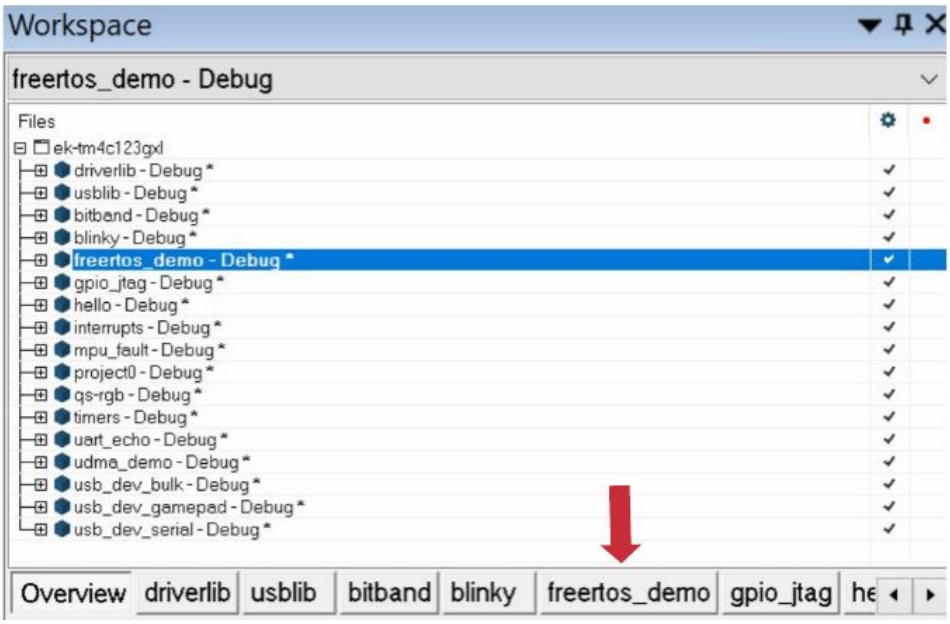

Figure 6: The ek-tm4c123gxl IAR Workspace Panel (with emphasis on the freertos_demo project)

5. Your window should now look similar to the screenshot in Figure 7.

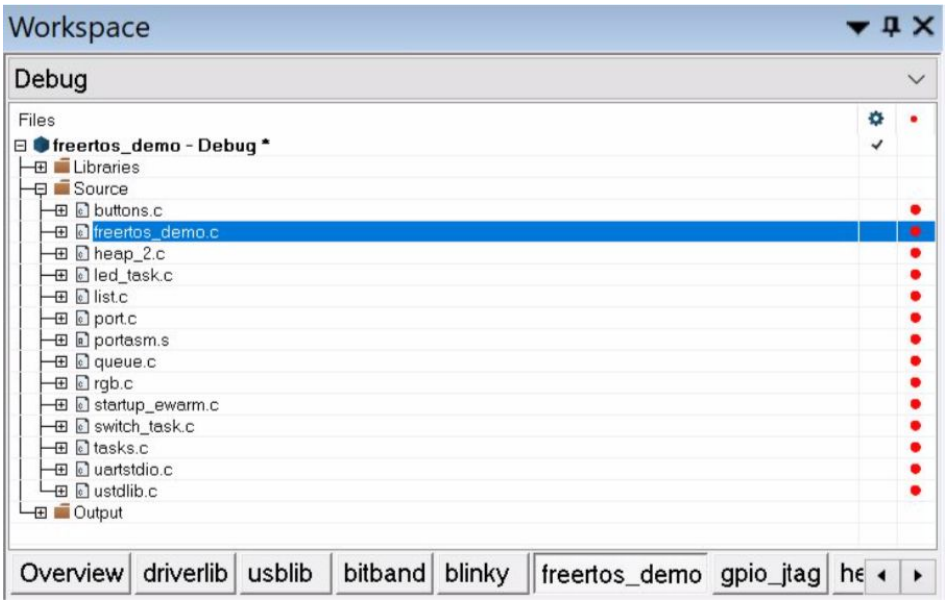

Figure 7: The ek-tm4c123gxl IAR Workspace with the freertos_demo project open

6. Replace the `freertos_demo.c` with the `main.c` we provided, which can be found under Canvas -> Files -> Labs -> Lab4 -> RTOS. We will only be using files #include by the starter code for this assignment; all other files you see in this directory is out of the scope of this assignment. Your project tab should now look similar to Figure 8.

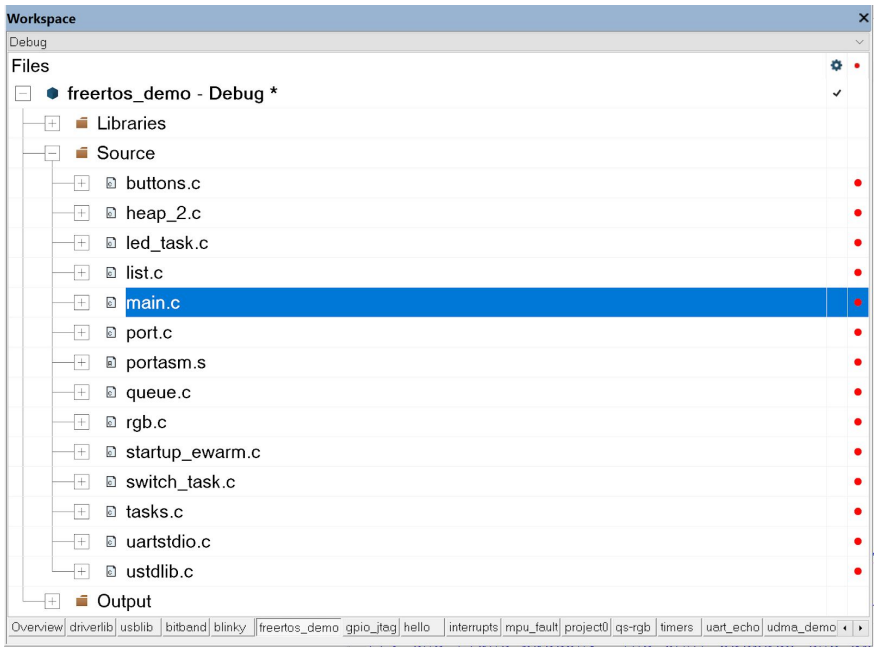

Figure 8: The freertos_demo project with the added starter code (main.c)

7. Open FreeRTOSConfig.h in the freertos_demo directory and change configCPU_CLOCK_HZ to 60000000.

Part 2: Complete the main.c file

1. Read carefully through the *entire* main.c file to get a basic understanding of your task and the structure of the file.
2. You should not use interrupts and timers for this assignment. The FreeRTOS application will use context switching instead.
3. Complete the skeleton code (you can ignore the warning: Inconsistent wchar_t size).
4. Your system must be running at 60 MHz at all times for this task. Otherwise, it will not function properly.

Lab Demonstration and Submission Requirements

- Submit demo videos of Task 1 and Task 2 on their respective due dates. The videos should show the expected results described in Task 1 and Task 2 on the TIVA Board. Feel free to add narration or text in the video. Each video must be less than 90-seconds long. To receive full credits, your videos must thoroughly demonstrate all the required functionalities. Before submitting, refer to the rubric shown on the "Demo Video" Canvas Assignment.
- Write a very brief Lab Report, as framed by the "Lab Report Guide" on Canvas -> Files -> Labs -> Lab Report and Code. Make sure to include a drawing of your FSM design for this lab.
- Revise the style of your Lab Code, as framed by the "Programming Style Guide" on Canvas -> Files -> Labs -> Lab Report and Code.
- Submit your Lab Report as a pdf, and submit your Lab Code (.c and .h files). Please do not submit compressed ZIP folders or other files such as the workspace files (.eww) and project files (.ewp). Before submitting, refer to the "Lab Report and Code Rubric" on Canvas -> Files -> Labs -> Lab Report and Code.
- On Padlet, write about a problem you had in the lab and the fix to it, and share a tip or trick you learned while working on the lab. You can also share an aha moment that you discovered while working on the lab. Avoid duplicating comments made by your classmates. NO videos for this Padlet task, please use textual comments. The link to the Padlet can be found on Canvas -> Assignments -> Lab4: Tips and Tricks. Please note that the Tips and Tricks assignment is part of the Community Participation, and thus the *Grace Period* does not apply.