ECE/CSE 474

Rania Hussein

Ishan Dane & Anael Aguayo-Chong

11/29/2022

LAB 4 Report

**Section 1: Procedure**

**Task1:**

For task 1a we were tasked with displaying a solid color on the entire LCD screen. Utilizing the tm4c1294ncpdt header file and the SSD2119\_Display file we enabled the indicated pins for ports D, K, M, N, P, and Q. We then created a main file where we included the appropriate header files SSD2119\_Display, SSD2110\_Touch, and the tm4c1294ncpdt. In our main function, we initialized the LCD screen and called the LCD\_ColorFill() method. We chose a color from the color fill array by including it as Color4[(color array number)], seen in Fig.1. This resulted in the LCD screen turning whichever color we chose.

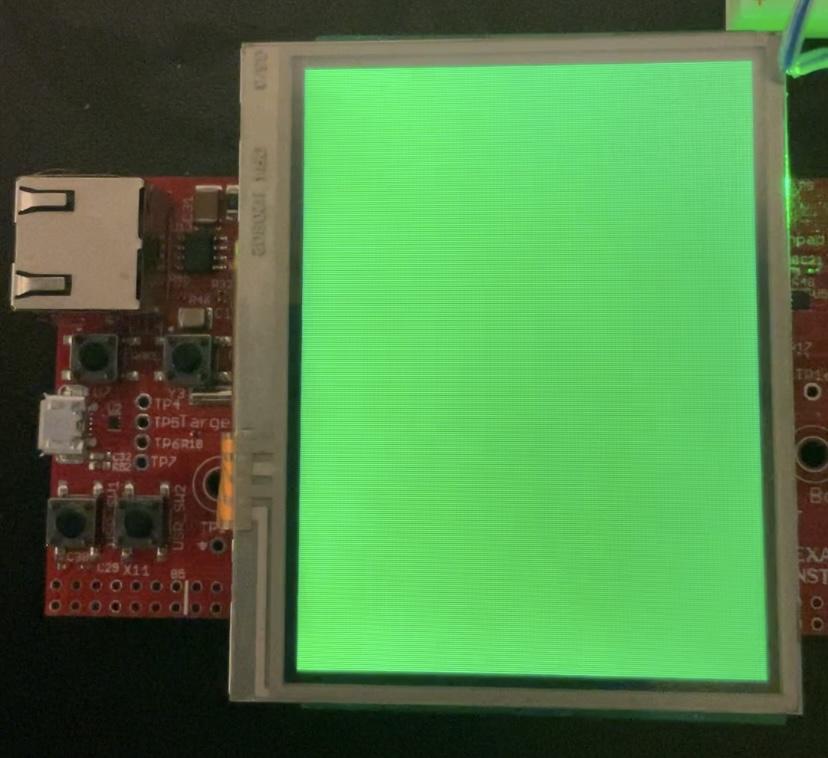
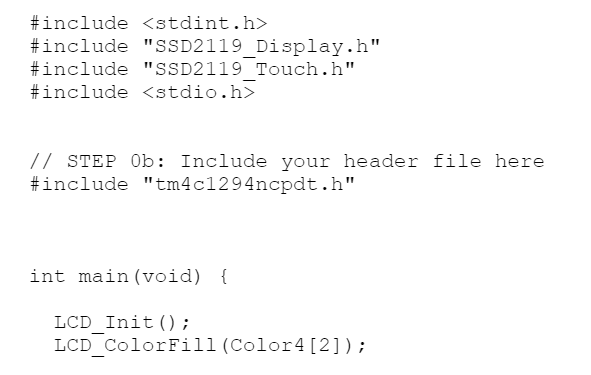


Figure 1: main file for LCD screen. Figure 2: LCD Screen showing Color4[2] (green).

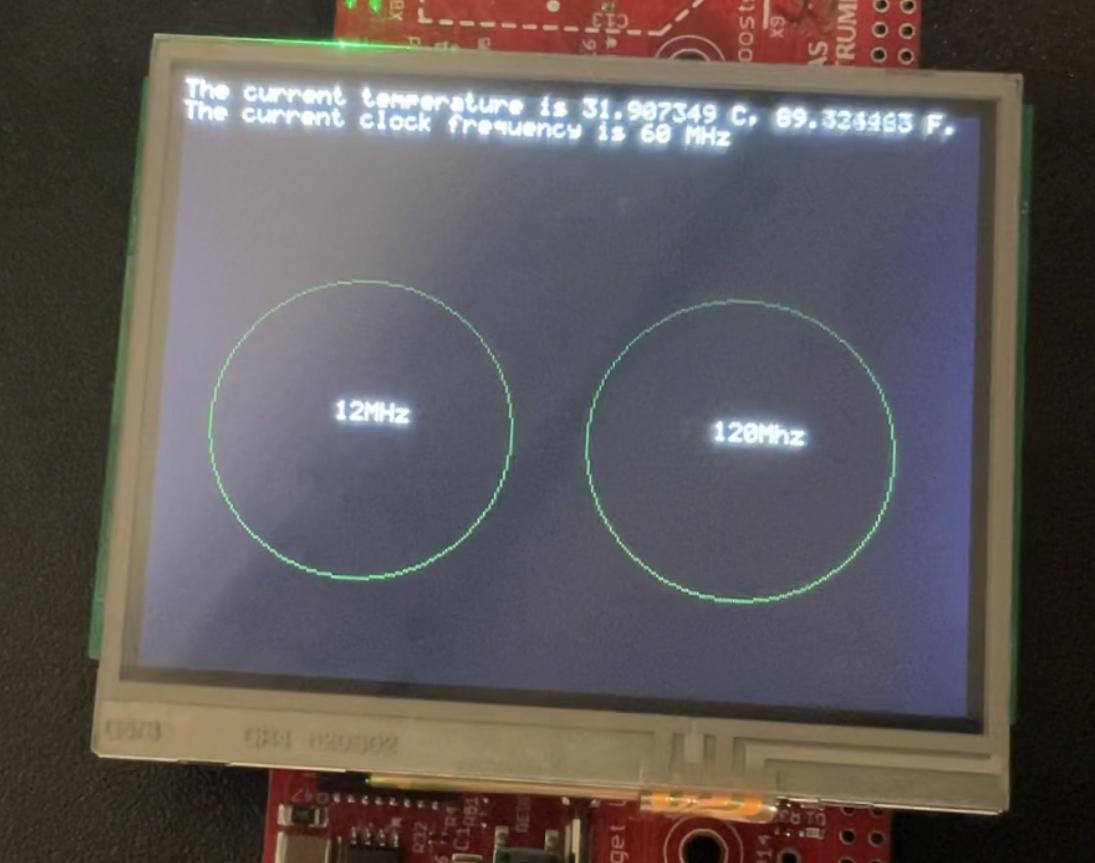
For task 1b we wanted to print the microcontroller's temperature on the LCD Screen in both celsius and Fahrenheit. We also had to print our current clock frequency below the temperature. Both statements were to be printed at the top of the LCD screen and to be updated by our ADC. Using our code from lab 3 UART temperature measurement, we set up the PLL and ADC0 interrupts necessary for temperature measurement. We also had to include the new formula to represent the temperature in Fahrenheit. To then print these values onto the LCD screen, we first had to figure out how to set up the LCD\_PrintFloat() function in the LCD display module in order to correctly print a temperature float value to at least 2 decimal places. To do this, we made a string character array of length 30 and used sprintf to convert the float into characters and store them in the string array. We then used a loop to extract and print out each character in the string array using LCD\_PrintChar(). To then print out the temperature and clock frequency messages on our LCD screen, we used LCD\_Printf() and LCD\_PrintFloat() to print the message containing the values, seen in Fig.3. We also used LCD\_SetCursor(), setting it to ‘0,0’ to ensure that the messages were being printed at only the top of the LCD screen. We then used if statements in our main module to set the microcontroller clock frequency to 12MHz or 120MHz based on which on-board switch was pressed. The values for the temperature provided by the ADC0 SS3 interrupt and clock frequency provided by the switches are then printed and continuously updated on the LCD screen, similar to our Lab3 UART communication. 



Fig.3: Message containing Temp values Fig.4: Message with Temp and clock buttons

For Task 1c, we had to adapt our previous LCD temperature print task to use 2 virtual buttons to change the system clock frequency instead of the 2 on-board switches. To create our virtual buttons on the screen, we used functions LCD\_DrawCircle, LCD\_SetCursor, and LCD\_Printf to draw the buttons on the screen and label them with either 12MHz or 120MHz. Our next step was

to check the regions occupied by the buttons in order to implement our button press design. To do this, we used a stylus and functions Touch\_ReadX and Touch\_ReadY with LCD\_Printf to coordinate and set up our boundaries for the virtual buttons. Once we tested and knew the boundaries to set in our if statements to replicate a button press, we were able to set our virtual buttons to change the clock microcontroller to 12MHz or 120MHz using PLL\_Init(), based on which button was pressed. This can be seen visually in Fig.4.

**Task 2:**

For Task 2a, we were tasked with implementing our traffic light controller onto our LCD display with virtual buttons instead, which can be seen in Fig.5. Like our previous controller, our traffic light system should turn on if the on/off button is pressed and held for 2 seconds, should switch to the pedestrian yellow light if the pedestrian button is pressed and held for 2 seconds, and should alternate between red and green, staying on each light for 5 seconds before switching to the next. Once again, in our code, we used the same Timer 0 and Timer 1 interrupts but instead changed how they interacted with the system in their timer handlers. We used a LED\_COLOR flag to check which traffic light state we were currently at in both timer handlers. We also used an int pedestrian flag and on/off flag which would increment as their button was being held down and would set itself to 0 if their button was let go of. Our timer handlers would then check these flags for a 2-second button press (ped or on/off flags > 0) or simply to change between lights and fill in the correct circles for the system, resetting the timers as well.

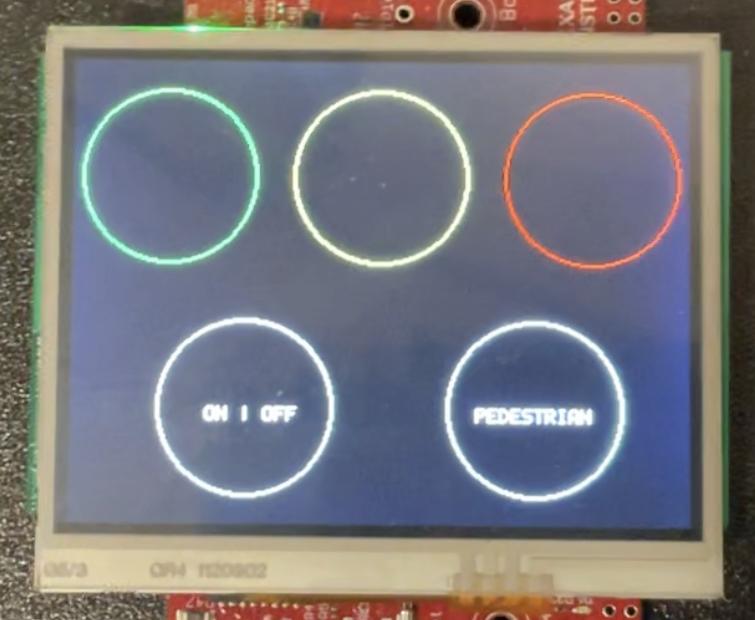


Fig.5: Traffic light Controller w/virtual buttons

For task 2b, we needed to implement the same traffic light from the previous task, but this time using the RTOS. We started by downloading the .exe file from canvas and following the directions on the spec which told us to add a different main file with instructions on how to implement the RTOS. In main, we first created the button and traffic light displays, which we just copied from the previous task. We then initialized the LCD screen, touch, and PLL to run the system clock at 60Hz. We then changed the priority of the functions StartStop(), Control(), and Pedestrian(), making StartStop() the highest priority and Control() the lowest priority. We wanted the buttons to have a higher priority than the control since the buttons should always override the control states. The StartStop() button was the most important because the program needed to turn on/off, regardless of the state and whether or not the pedestrian button was pressed. After we set the priority, we implemented the three functions Control(), StartStop(), and Pedestrian() using a similar structure in each one based on the instructions. For the StartStop() and Pedestrian() functions, when their virtual button was clicked, we had to check that the tick time of the button press was greater than or equal to 2000, which was 2 seconds before changing the state accordingly. If it was greater than 2 seconds, the flag that tracked the pedestrian button (in the pedestrian function provided), or the start/stop button (in the start/stop function provided) would be set to 1 and the timer is cleared by setting the previous tick count to the current tick count. If the virtual button was let go of early, the flags were set back to zero. The Control() function was slightly different; it checked if any of the flags were set to one and if the tick time was greater than or equal to 5000 which indicated 5 seconds. If it was, the control function would then call on the FSM() function to change states according to which light the system was on and which button was pressed.



**Section 2: Result**

**Task1:**

For task 1a we wanted to display a solid color on the entire LCD screen. Following the spec, we enabled all of the indicated pins for the ports in the LCD Display file. We then created a main file and followed the directions given in the spec to initialize and display the color. Once we found the LCD\_ColorFill(unsigned short color) method in the SSD2119\_Display.c file we did not know how to call it in our main. From the color array, we tried calling in the number 2 for the color green as LCD\_ColorFill(2) but it did not take 2 as a parameter. After carefully reading through the provided LCD Display module and understanding it better, we tried including the color array as LCD\_ColorFill(Color4[2]). Fortunately, that was all it took to light up the screen seen in Fig.2.

For task 1b we were tasked with printing a string and the temperature of the microcontroller on the LCD screen. To do this we reused our code from lab 3 and converted all of the pin names to the names used in the tm4c. Once we did this we wanted to print the temperature data, so we utilized printf to print the statement (“The current temp…”) with the temperature. We noticed that the statement would print but without the temperature value. We went back to the spec and saw that we needed to set up the LCD\_PrintFloat to change the temperature from a float to a string. We converted and stored the float value in a char array and character by character turned it into a string and print it onto the screen. We then utilized LCD\_Printf to print the statement and the temperature in both celsius and Fahrenheit. It was finally able to print the temperature change while keeping the text in place with LCD\_SetCuroses(), and not keep printing down the page, as seen in Fig.3.

For task 1c we did the same as in part b but now instead of using onboard buttons, we changed to LCD touch buttons. We were able to draw the buttons on the LCD, but when we tried to fill in the circle for the buttons and insert the label on top of the filled circle it would freak out and the label would show up and then disappear. We then opted to just create the circle outline with the label in the middle which didn't give us any problem. The virtual buttons’ labels were then printed on the LCD screen, but would not activate the 12MHz clock or the 120MHz. After going to OH we found that the issue was because we didn't include the PLL\_Init() method, so once we added it and configured the virtual buttons the system worked properly. This can be seen visually in Fig.5.

**Task 2:**

For task 2a we were tasked with implementing a virtual traffic light controller using the LCD display. We started by including code from lab 2 and changed it so that when the on/off button was pressed it would fill the red circle and then alternate to filling the green circle, and when the pedestrian button was pressed on the green state the system would switch to filling the yellow circle with every circle filling in and holding for 5 seconds before switching states. Here we needed help with the time it would stay in one circle. It would be holding it for 3 seconds before changing. We went over the code and realised we turned the 5 second timer on before the circles were filled, and therefore since the system would take 1 or 2 seconds to fill in the circle, this was causing the timer to run out after 3 seconds of the filled circle. So we tried filling in the circles first and starting the timer after and that made all the difference, fixing our issue. Another confusion we had was we thought the LCD had its own interrupt method with the ADC. This made it very difficult for us to start as we did not know where to find the LCD interrupts in the datasheet. After consulting OH we saw that we had to actually set flags for button presses instead which would be evaluated at the timer handlers. After resolving this issue the system worked, as seen in Fig.5.

For task 2b we are now implementing the traffic light system using RTOS. We followed the freeRTOS download instructions given to us and continued to use the RTOS. The first issue we ran into was clearing the timer in RTOS because we were clearing the timer ICR but this wasn’t working. After consulting OH we found out that RTOS works on the tick counts and by setting the previous tick to the current tick we would be able to clear the timer. We had this done and our code looked fine but was still getting an error, seen in Fig.\_. We were back in OH trying to figure out what was giving this error, and could not find a solution. After spending some time we decided to download freeRTOS again to see if maybe we deleted something on accident. It turned out we did, and when we tried it on the LCD screen the system worked great with all of the correct pauses.

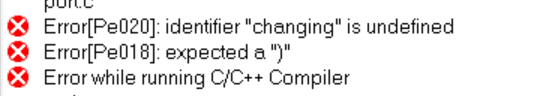


Fig.6: Error message due to freeRTOS not downloaded correctly.