ECE/CSE 474

Rania Hussein

Ishan Dane & Anael Aguayo-Chong

October 20, 2022

LAB Report 1

**Section 1: Procedure**

**Task 1:**

For Task 1a we were given code, main.c where it was enabling PF0 and setting it to output, digital port, and setting it to 1 which turned on the LED4. The header file was used for declarations for PF0. Since we wanted to turn all LEDs on and off in a periodic way, we first had to set up PF4, PN0, PN1. Where PF0 and PF4 are LED4 and LED3 respectively, PN0 and PN1 are LED2 and LED1 respectively. Knowing this we had to first configure the GPIOs for Port N:

GPIODEN register to set the pins to the digital port, the GPIODIR register to set the pins to output and the GPIODATA register to set the data for the pins; since this was not previously provided in the header file (configurations provided in Figure 1). After this, we had to correctly configure the pins for the LEDs in the main file. This was done by setting the hexadecimal bits for the LED GPIOs to the correct pins.After being done with setting the GPIOs, we moved on to the main program. We used delays created by FOR loops before assigning the LED to their values so there was a clear pattern from one LED to the next. We then set one LED on one at a time by turning off all the others, this was done through GPIODATA. This allowed us to create a periodic pattern with the LEDs that restarted each time with the while loop. Our periodic system is shown in Figure 2.

In task 1b we were tasked with implementing SW1 and SW2 to their corresponding LEDs 1 and 2. We had previously set up LEDs 1 and 2 with PN0 and PN1 so we used our already configured headers and bit assignments for the port N pins. For the switches we had to use PJ0 and PJ1, thus we first had to change the RCGCGPIO to include port J. We then once again included all the relevant GPIO information required for port J but instead, this time we needed to include GPIOPUR for the pull-up resistors in the header file. After the header file was complete, we had to set the bits for port J’s pins in the main. This was done by setting the GPIOs bits to pins PJ0 and PJ1 except this time, the pin’s GPIODIR was set to input and therefore 0x0 (0 for input, 1 for output). Then using if statements we provided logic to allow if SW1 were to be pressed it would turn on LED1 and if SW2 were pressed it would turn on LED2 else both would be off. Figure 3 shows a bit of the code showing port J and logic used for task 1b.

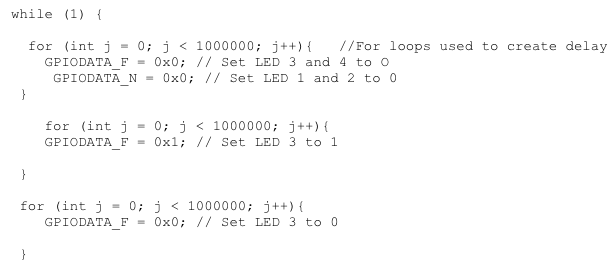
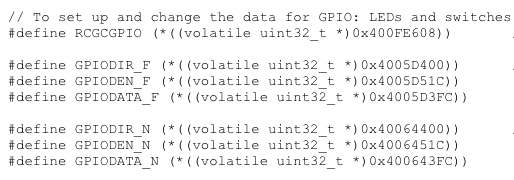


Figure 1: Configuration of ports. Figure 2: part of Task 1a code.

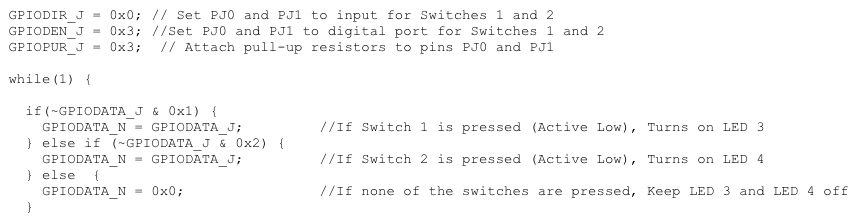


Figure 3: Task 1b code.

**Task 2:**

For Task 2 we were designing an FSM implementing a traffic light system with LEDs. We had Red, Green, and Yellow LEDs. Red was used for our ‘stop’ state, Green for our ‘go’ state and Yellow for our ‘warn’ state. We connected them to the solderless breadboard using figure 4 to assist us in connecting a 220 Ohm resistor in between the voltage source and the LED. We made sure to correctly connect the pins or the legs of the LEDs to prevent a block in the current flow. When the system was turned on by the ON/OFF button shown in figure 5, the traffic light system alternated between the ‘go’ and ‘stop’ states shown by the green and red LEDs alternating. This was done by creating cases: We had a case called ‘idle’ which kept the LEDs off until the on/off button was pressed. On pressing the button, the system would move to the case ‘go’ and the system LED behavior would change based on what button was pressed and when. When the system was turned off with the same ON/OFF button, the LEDs were turned off, we assigned this output to the ‘idle’ state and the traffic light system was off. When there was a pedestrian and the pedestrian button was pressed on the green state, the green LED would switch to the ‘warn’ state yellow LED and then finally to the ‘stop’ state red LED. The system would then resume its default alternation between ‘go’ and ‘stop’. If the pedestrian button was pressed on the ‘stop’ state red LED, this would do nothing and stay on the state as we were already on the ‘stop’ signal.

This pedestrian movement takes action first in the ‘go’ case where we had logic in an if statement that would transition the system to the ‘warn’ state with only the yellow LED lit up if the pedestrian button was pressed on it. The system then traverses from ‘warn’ to ‘stop’ and then back to the default alternation between ‘go’ and ‘stop’, so long as the ON/OFF button was not pressed. If the ON/OFF button is pressed at any point during the transition or alternation, the system goes to the idle state and LEDs are turned off. This is displayed by the Figure 4: FSM chart below.

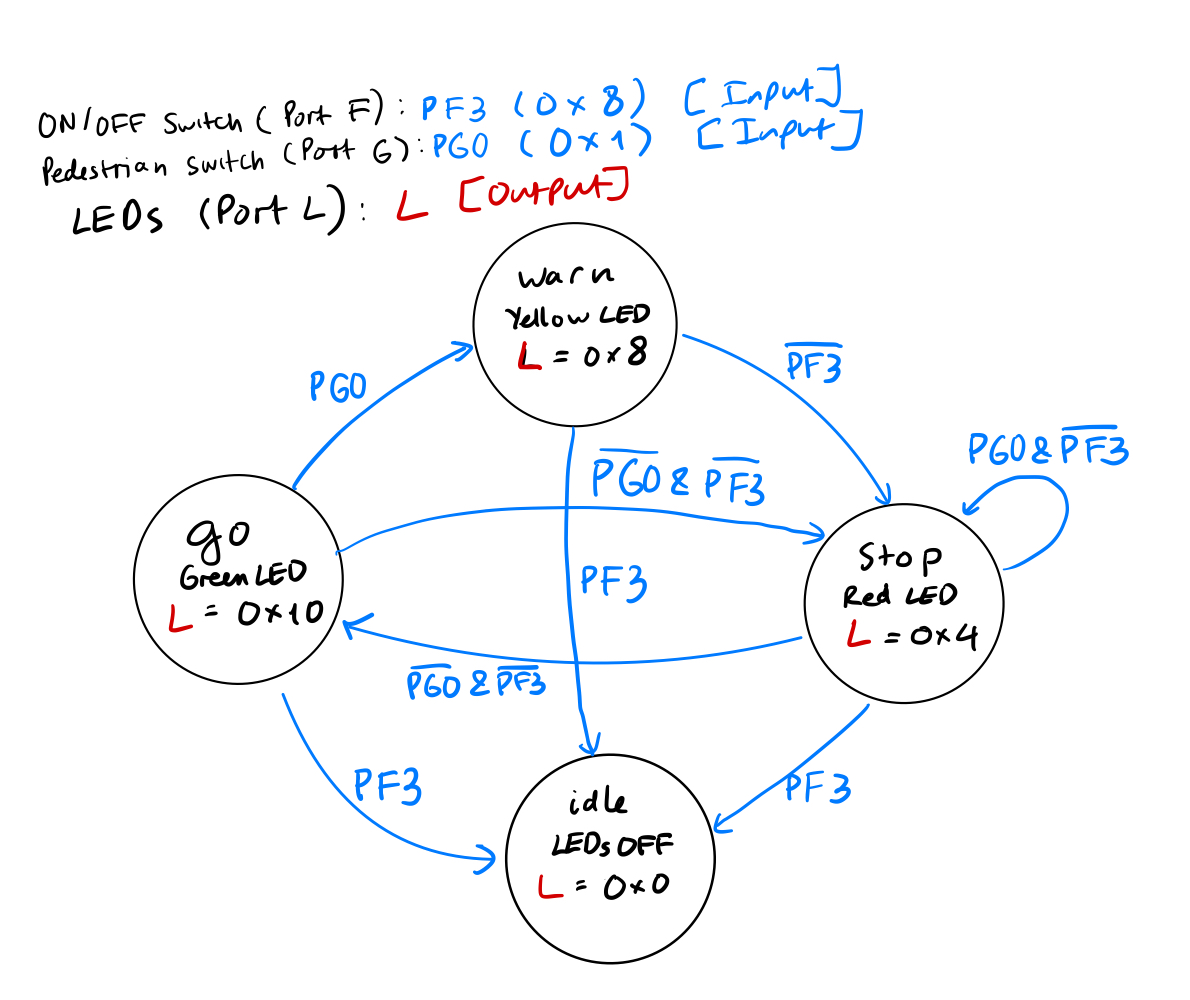


Figure 4: FSM diagram for Traffic System.

To closely mimic a traffic light system, we implemented FOR loops to create delays on each state LED signal. For our FSM states, we used IF statements to traverse between states based on switch conditions. To set our outputs, we assigned values to GPIODATA\_L for the LEDs on each state to correctly light them up. For our case inputs, we read GPIODATA\_F and GPIODATA\_G for switch conditions.

We started off using all port L for the SWs and the LEDs as suggested by the lab but then ran into case logic errors due to confusion and our SWs weren’t working with the LEDs correctly. Therefore to make it easier to design and debug these errors for each case we decided to assign each SW an individual port separate from port L leaving port L only for the LEDs. The on/off button was set to port F through PF3 and the pedestrian SW was set to port G through PG0. This made it much easier to design and manipulate the LEDs in each case according to the switches.

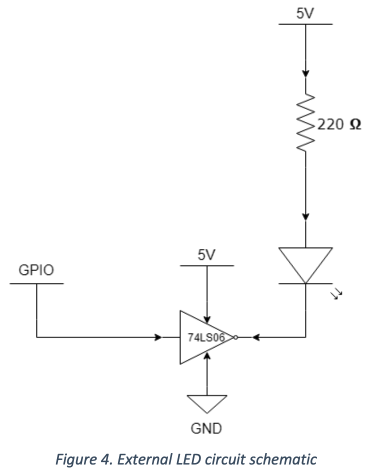
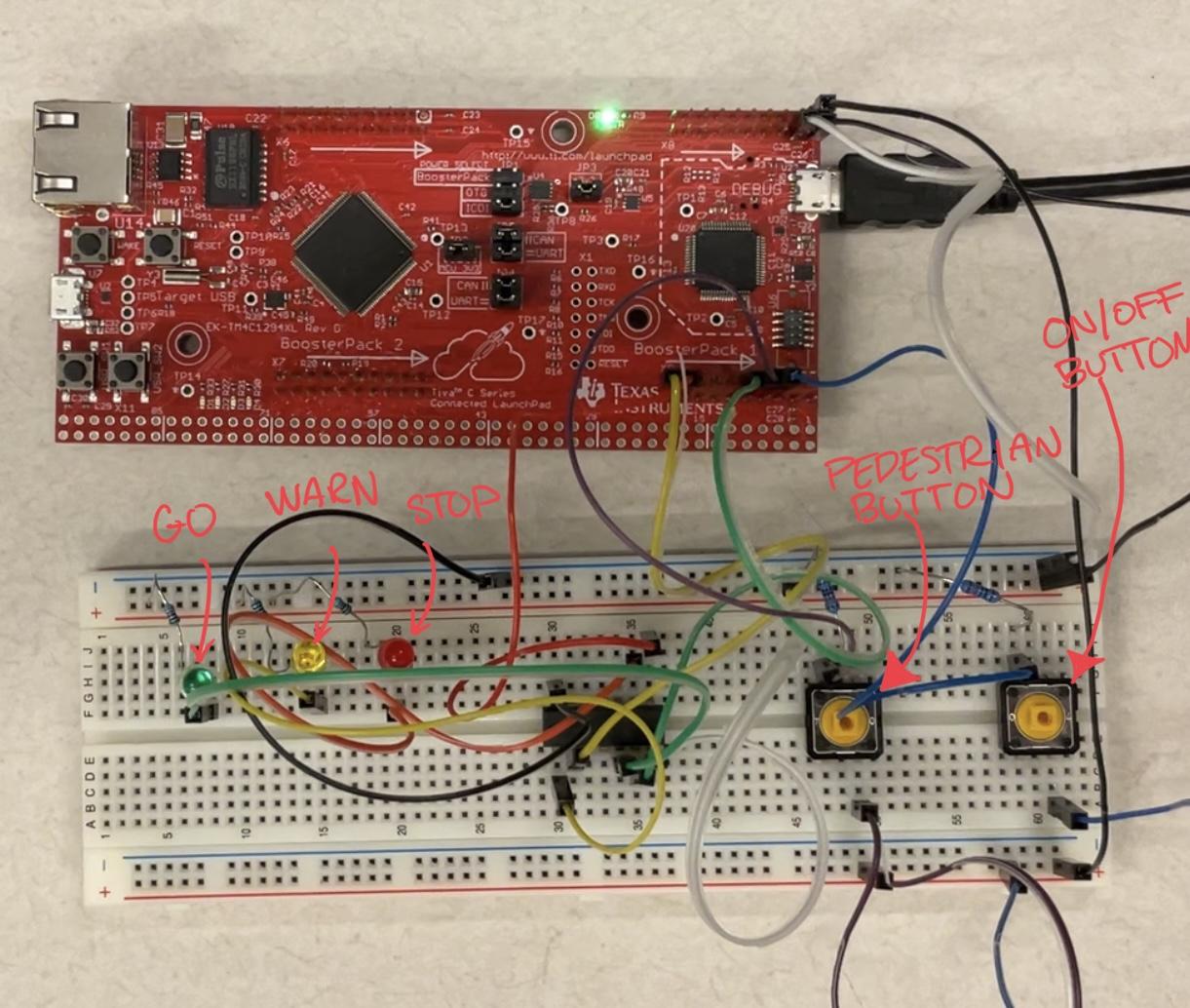


Figure 5: demonstrating our circuit.

**Section 2: Results**

**Task 1:**

For Task 1a we were asked to alternate the 4 LEDs in a periodic pattern, so we incorporated FOR loops to achieve the delays we were needing to make the pattern slower and observable. After writing the code we connected the board but were having issues with the LEDs they were not turning on. After debugging our circuitry and code we realized we hadn’t configured the project settings. So, once we took a look at the IAR New Project tutorial we tried again by setting the debugging tools to TI STELLARIS and it finally worked. We were able to turn the LEDs on.

For Task 1b we were also able to connect the SWs to the LEDs, making it so that in every click of either SW1 or SW2 it would turn on LED1 or LED2, shown in Figure 6. The biggest problem we had was that it took us a while to realize we had skipped step 3 of attaching the pull-up resistors in the handout and therefore our switches were not working. It took us a while to figure this out and we spent a lot of time trying to debug the code, specifically our GPIOs bit setting in the main and values assigned to them in the header. We first tried to set up and debug one switch at a time to fix this issue. However, once we went back and reread the task, we noticed our mistake and included the GPIOPUR register for port J and assigned it to pins PJ0 and PJ1 and the switches worked. Besides the pull-up resistors, we did not run into any other problems setting up our LEDs on our board. This was because we had already set up the LEDs and ensured they worked in task 1a.

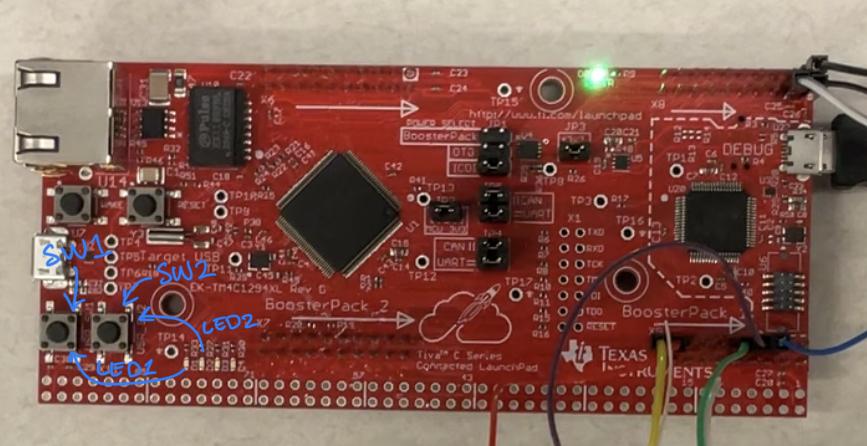


Figure 6: Board showing SW1, SW2, and LED1 and LED2

**Task 2:**

For Task 2, our problem occurred when we tested our code. When we compiled and ran our code, we noticed that the LEDs were not turning on regardless of which button was pressed and when. We had previously tested the buttons, and the LEDs individually without the FSM, and everything checked out. Therefore it could not have been the circuitry but our code instead.

We were trying to debug the code for a while and could not figure it out as our FSM code syntax and implementation was correct as compared to the ZyBooks equivalent. Since we couldn't find the solution, our next step was to ask the TA during the LAB Section. Consequently, we were told that we need to assign our FSM function Traff\_Latch in the header file to tell the compiler to use the function in the main. With the function in the header file, it was still giving us an error but now related to the GPIODATA parameters’ type we were passing through our function. To fix this error, we first tried to use the int type for the GPIO parameters but still received the same error. We then tried uint\_32 and matched the function call that was in the header file to the call out in the main, and it was able to compile the code correctly without error.

The next problem that occurred was that the ON/OFF button was not communicating with the LEDs. This was due to an issue we saw in our IF statements. We had done: IF (GPIODDATA\_G & 0x1 == 0x1) , so our order of operations was off. Once we added the extra parentheses: IF ((GPIODDATA\_G & 0x1) == 0x1) as advised by the TA, shown in Fig.7, the if statements started to activate the LEDs.

The problem we had next was to do with our on/off button not working consistently. We thought that the button itself might be causing the issue because when we wanted to turn the system on or off it would alternate between doing it on the first click or requiring two or more tries for the system to actually turn on or off. We asked the TA and he mentioned that it could be a button issue, so our next step was switching it out for a new one. However, this did not fix the problem. The TA then said the next possibility was an issue with our logic, where we were not allowing the system to recognize the button click, or we had not included the other button in the logic. Per his advice, we went back to our code and started debugging our logic using breakpoints to see when the FSM would enter the faulty ON/OFF IF statement to turn the system on or off. We found that the problem lied in the logic behind our IF statement. We had used ((GPIODDATA\_F & 0x8) == 0x8) to turn on and off the system with the switch. However, once we changed the code to just (GPIODDATA\_F & 0x8), our on/off button started to work correctly. An example of this change in the ‘go’ state is provided in Figure 7.

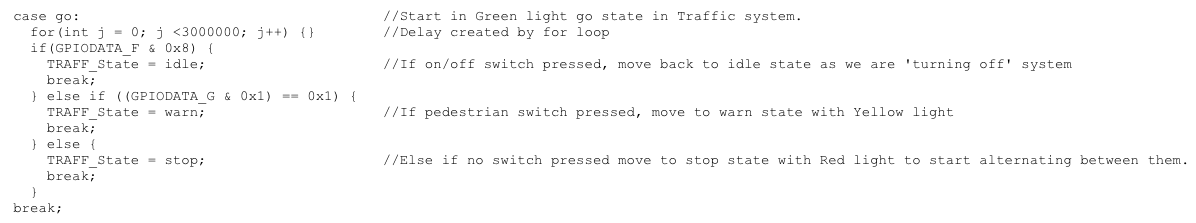


Figure 7: Case ‘go’ showing the final result after troubleshooting.

After we fixed our on/off button, we did not encounter any more problems. Everything in our circuitry system worked in accordance to the code and our Traffic Light system was finally complete.