A logo of a company

Description automatically generated

California Polytechnic State University Pomona

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

# Microcontroller Laboratory

ECE 3301L-01

LAB #8 & #9

Traffic Light Controller with use of System Timer

Prepared by

Rayan Alruainy & Kaung Nyi Naing

Presented to

Felix Pinai

November 4, 2024

**Objectives:**

To implement the traffic light system using a color LCD screen, to adopt a multi-file project layout for larger programs, and to use external interrupt pins in PORTB to implement the pedestrian crossing requests and flashing/maintenance mode.

**Summary:**

* Since most components were already wired in Lab 7, the wiring for Lab 8 and Lab 9 was simple. The main difference between lab 7 and lab 8/9 was the use of the LCD screen instead of a dual 7-segment display. Besides these wiring changes, there was also the matter of the logic analyzer: the logic analyzer had many more connections in lab 8/9 than in lab 7. The differences in wiring between lab 8/9 were also extremely minimal. In lab 9, we simply added a few push buttons to the circuit for the pedestrian crossing functions and reset mode.

Lab 8

In lab 8, we replaced the dual 7-segment display with an LCD screen. However, this required extra code to implement which explains the addition of Main\_Screen header and c files, ST7735\_TFT header and c files, and utils header and c files. We also had a header file which cleared our main c file of many lines of definitions and other value declarations. Many of our hard coded variables such as the traffic signal wait times, color values, direction values and pinout definitions are also handled in the main header file instead of the c file. Additionally, most non-experiment specific functions such as buzzer, timing and waiting functions are moved to the utils c file so that they are easily transferred between projects.

In the main c file, we had to rewrite functions such as the PED\_Control and Wait\_N\_Seconds. We made sure that both functions have loops that count down instead of ones that count up. The PED\_Control was generalized to handle both crossing directions and port specific outputs were removed. We added the required LCD update commands to all main c file functions and most waiting functions. Although the ST7735\_TFT files were not modified, we added extra conditional statements in the Main\_Screen.c file. In the update\_LCD\_color file, if statements to draw the red, yellow, and green traffic signals were added. Additionally, in the update\_LCD\_PED\_Count and update\_LCD\_count functions, more case statements were added to display all six different count downs on the LCD. One more LCD function that we modified was update\_LCD\_misc, which handled updating the operating mode, light sensor reading, and input signals such as the four switches. More if statements were added to this function in Lab 9 to handle the request and status signals.

Since there are many files in this lab, it took a lot longer to debug variable name and function name problems which are usually caused by typing them in the wrong case. But after thoroughly debugging the code, the program ran properly and communicated with the LCD to display the appropriate traffic signal and count downs.

Lab 9

In lab 9, we expanded the code in Lab 8 by including two extra files interrupt.c and interrupt.h. These files are used to handle the external interrupt signals taken from the buttons connected to pins RB0, RB1 and RB2 from PORTB. The reason for using these pins is that they are the only three pins that are multiplexed with the three interrupt inputs INT0, INT1 and INT2. External and internal interrupts are useful because they allow the CPU to perform other tasks instead of it having to constantly watch for changes which is resource heavy and time consuming.

In order to receive an interrupt signal, we must configure several registers related to each interrupt signal, mainly INTCON, INTCON2 and INTCON3. Each flag must have their enable flag set to 1, the interrupt detection flag cleared before use, and an edge flag set to 0 or 1 to determine if the interrupt occur at the rising or falling edge of a signal. Most importantly, the global interrupt enable flag in INTCON register must be set to 1 or no interrupt events will be detected. We set our edge flags to 0 for falling edge detection since our high inputs are pulled to low when we press the buttons.

In part A of Lab 9, we changed the main code to always wait for an interrupt event to occur and output a line in terminal when an interrupt event happens. This was used to test our wire connections and to double-check that our interrupt codes are working as intended.

In part B of Lab 9, we implemented the three interrupt signals into our traffic control system. Two of the interrupts will replace the NS and EW pedestrian crossing control switches and third one will be used to trigger a new mode called flashing mode. For the pedestrian control buttons, we added extra char variables to replace the hard coded pedestrian switch inputs and changed the interrupt service routines to set these char variables instead of the interrupt flag variables. Some code that relies on the hard coded switch inputs had to be modified and we made sure to clear the pedestrian control variables after the count down is finished or if the operating mode is changed to night mode. This is important in our experiment, since no pedestrians are allowed to cross during nighttime and the interrupt service routines also have conditional statements to prevent pedestrian count activation during night time.

In part C, we implemented a traffic maintenance mode which is simply called flashing mode in this experiment. As the name suggests, we will turn all traffic signals red for one second then turn all traffic signals off for one second. When the button is pressed, the interrupt event sets a variable called Flashing\_Request which initializes the flashing mode. This event immediately clears the Flashing\_Request so that a new interrupt event with INT2 can take place and sets the Flashing\_Status variable to 1. As long as the Flashing\_Status is 1, the flashing mode will flash the traffic signals, thus, to exit the loop, the Flashing\_Request must be set again. In that way, one press of the third button will start the Flashing\_mode and another will quit it.

During our lab demo, our circuit froze. However, we could not find what causes this type of problem. When we tried to reproduce the same scenario again, we were not successful.

**Conclusion:**

In conclusion, Labs 8 and 9 built upon the traffic light control system we developed in Lab 7 by integrating a color LCD screen for better visual feedback and implementing external interrupts of the PIC18F for better pedestrian crossing and flashing requests implementations. The transition to a multi-file project structure allowed us to separate and organize code for different functionalities, making it easier to manage and expand the project for future labs. By using interrupt signals on PORTB, we reduced the CPU load though interrupted processing by avoiding the constant checking method. Thus, code that uses interrupts are generally more resource-efficient and suitable for real-world applications. Although debugging variable naming and case sensitivity across multiple files took time, the structured code facilitated cleaner debugging and expandability. Overall, these labs provided practical insights into the important features of the PIC18F such as LCD serial connections, external interrupts, and timers equipping us with essential skills for developing more complex embedded systems.