**Mode 1**

Functions:

1. simple_tlc
2. tlc_timer_isr
3. lcd_set_mode
4. Call_Timer
5. ChangeLED
6. handle_mode_button

For Mode 1, a traffic light controller where each light lasted a different amount of time was required. By using a single timer, this required different values to be set as the timeout value each time. We did this through matching up the states with its corresponding timeout value in an array. For example in execution: when going from NS Red light (state 0) to green light (state 1), a timeout of 6 seconds was passed through as six is stored in the [1] position of the timeout array.

A flag was created to indicate that the timer needed to be called the first time since we were only updating the state as the timer time out occurred. We also created a helper function CallTimer() because calling the traffic lights timer was common to all modes.

**Mode 2**

Functions:

1. init_buttons_pio
2. pedestrian_tlc
3. NSEW_ped_isr
4. PedestrianReset
5. Call_Timer (resued)
6. ChangeLED (resued)

For Mode 2, pedestrian states were to be added to the traffic light controller. We added a button interrupt to process and record the keypress(es) while the traffic lights are running. The ISR would set the pedestrian flag/s for the NS or EW lights depending on the button. When the current state of the controller is updated, the relevant pedestrian flag is checked at NS Red, and EW Red and the LEDs change accordingly. If the pedestrian state is on, it is reset when the light changes to Yellow state.

As changing the LEDs on the board was common to both mode one and mode two, we made another helper function ChangeLED() that changed the LEDs based on the current state.

**Mode 3**

Functions:

1. configurable_tlc
2. timeout_data_handler
3. pedestrian_tlc (resued)
4. ChangeLED (reused)

For Mode 3, timeout values were to be changed through user input through UART transmission. We had several issues in receiving the user input correctly from PuTTY.

1. The main issue was the blocking fgetc() line of code.

When the enter key was pressed at the end of the timeout values string, the program kept waiting for another enter key input before our “packet received” print statement showed up. However, the entire string received would then end up being incorrect.

We attempted to solve this by adding another flag when the enter key is pressed so that the program would not execute the fgetc() function again and move on. But we were bewildered when the program execution crashed after adding this. It seemed that the placement of where the flag was set and reset was the cause of the crashes. Through a lot of trials and error testing, putting the flag inside the timeout_data_handler() and outside in the configurable_tlc(), we were able to update the timeout values successfully.

Since we need to process the user input from UART, the string recorded would be processed and split up using comma characters as the separator. We used the atoi() function to change the characters into integers and then updated the time out values.

Incorrect input strings were accounted for by recording the number of commas (5) present. Non-integer inputs set off the wrong input flag immediately, and negative numbers or numbers greater than four digits were also not allowed. If the correct format was not received, the invalid statement would be shown.

2. The second issue was in printing to PuTTY through UART.

Whilst testing our code for the UART print statements, they would not display or be significantly delayed. We fixed the problem by adding the ‘fflush’ function after each 'fprintf' statement to output the buffer and move it to the buffered data to the console.

**Mode 4**

1. camera_tlc
2. handle_vehicle_button
3. camera_timer_isr
4. timer_isr_function
5. configurable_tlc (reused)

For Mode 4, a red light camera was to be added that records the time a vehicle was in the intersection simulated by a button press. A snapshot of the car was to be taken immediately or after 2 seconds under certain circumstances. To implement this, we added two separate timers to account for both the vehicle time and the 2 second time limit simultaneously. The interrupt button of key two was included in the existing button ISR in mode 2. When testing our code, we ran into a problem.

1. The program would crash when we pressed the button to simulate a vehicle entry/exit.

Originally we thought the car enter and exit flags being set and reset in the button ISR was a factor in this. However, after testing, it turned out that the timers were the problem. The timers were being started repetitively every loop when a car entry was registered, as long as the state was correct. To fix this, we added an additional flag to indicate that the timers had already been started for the specific vehicle. These timers were stopped prior to mode changes, and all were flags reset to avoid this issue too.

**Design Approach**

We approached the project by completing each mode separately in a separate file. This caused duplicate functions and resulted in lengthy code.

After having all the modes completed, we combined all the functions into one file and made the code more concise by scanning through commonalities and joining them together.

We wanted our project to be easily understandable, so we used well-defined variable names and informative comments. Additionally, the functions for each mode were separated from each other so other people can clearly see which functions contribute to which modes. We used enumeration in our code to better define and recognise the state of the traffic light controller (i.e. state 0 referred to as RR_NS and state one as GR_NS). Our project is also highly reusable where the settings (e.g. button inputs, LED outputs, common constants) can be altered through global variables.

**Future Work**

In the future, we would like to make our project even more readable and concise so that other people can reference our code in their projects more efficiently. For example, some functions could have been slimmed down, even more, thereby reducing the unnecessary complexity in the code. In some rare circumstances, the UART statements were not printed clearly on PuTTy so we would like to make our project error-free in the future.

Our project could also utilise more features of the board, such as making use of the 8-bit segment display to show the timing between key two activations. Two red LED lights could also be configured to illuminate to signal a pedestrian button (i.e. key0/key1) has been pressed.

| | Time | Reasoning |
|---|---|---|
| Mode 1 | 5 hours | Thinking of a design approach and watching lecture videos. |
| Mode 2 | 2 hours | Implementations for key0/key1 for pedestrians. Small additions to mode 1. |
| Mode 3 | 1 day | Unable to get PuTTy working. Problems with fgetc() and fprintf functions. |
| Mode 4 | 10 hrs | Problems with the buttons and timing issues. |
| Combining all modes | 1 hr | Make code more concise and eliminate unnecessary code. |
| Testing | 3 hours | Testing all the modes and making sure they work properly. |
| Report | 8 hours | Reflection on the assignment and editing. |