Kiana Peterson and Nattapon Oonlmaom
CSE 474: Intro to Embedded Systems
November 2nd, 2023
Lab 2: General-Purpose Timers and Interrupts

Procedure

The purpose of this lab is to delay CPU operation to the made Lab 1, implementing timers and interrupts. The goal is to understand general purpose timers and interrupts and how they will delay the system operation effectively, reducing unnecessary CPU execution.

Task 1a: In this task, we must repeat the task from Lab 1, where we were asked to turn off and on the onboard LEDs in a periodic pattern. In addition to doing so, the LEDs should turn off and on at a rate of 1Hz by polling a timer. Once we understood how timers work and how to access them, the implementation became simple. We watched the lecture by the professor regarding the timer, looked at the datasheet, and used the information to make adjustments to Lab 1 Task 1a. After initializing all the ports and timer according to the datasheet, we made a function to turn on/off the LEDs in a sequence with delays that included the timer. In our lab 1, we used a delay that was made up of a for loop with a set number. For this lab, we had a while loop that polled for the flag to be raised that the timer reaches 0.

Task 1b: In this task, we used the circuit from Lab 1 Task 2 to make a timed traffic light system. We made the LED respond accordingly to the button pressed with a timer delay. For the system to respond, the button must be held by the user at least for two seconds for all button presses. However, the system system should be off when reset. To approach this, we modified the FSM from the Lab 1 Task 2 to have delays within the states transition. We added counters to track how many seconds the on/off and pedestrian button are held. We also modified the timer to take in a parameter, “desired second” to set specific times for the delays. We then integrated the timer and counters into a for loop to approximate the seconds. Five seconds for the lights to be on/off, and 2 seconds for the button response. During the developing stage, we ran into inputs timing and state transitions issues, we kept debugging using the step into functions and rearranged the code accordingly until it worked. We found that the system was operating too quickly to take in the delayed inputs. To solve the problem, we ended up using a for loop to delay the states transition, so the system takes in the inputs correctly.

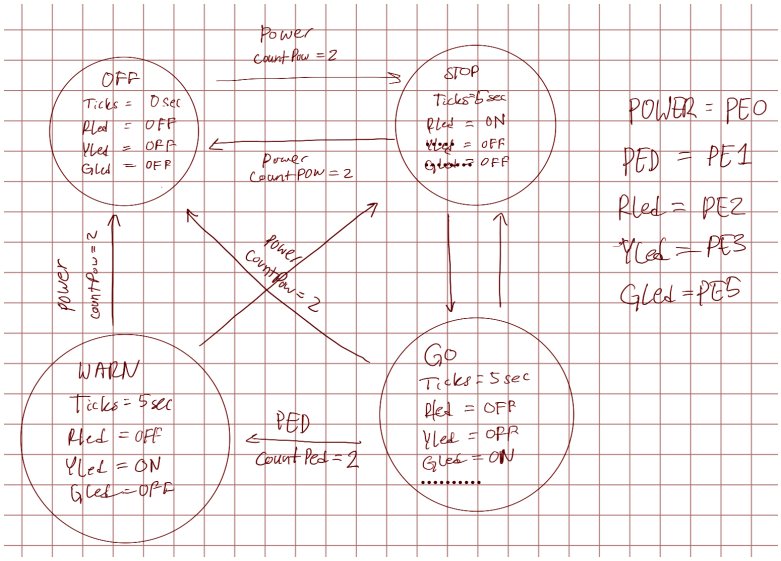

Figure 1: State Diagram of Traffic Light Controller

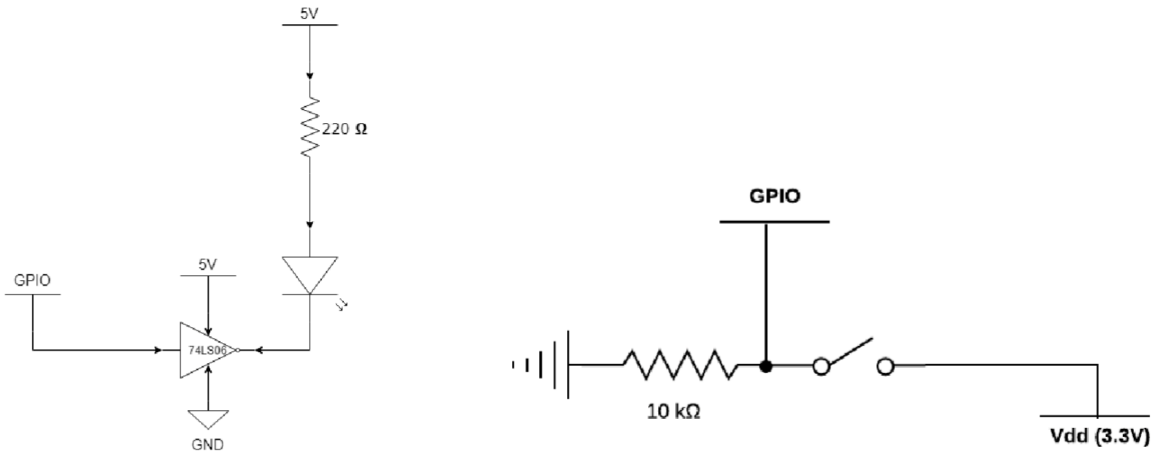

Figure 2:Circuit Schematics for Lab 1 Task 2

Task 2a: We repeat Task 1a, but instead of polling the timers, we use interrupts. To implement the interrupts, we followed the steps outlined in the lab document and referenced the datasheet. In addition to port/interrupt settings, we had to create a handler for the timer and update the cstartup_M.c file by adding the function declarations and updating the vector table as shown in Figure 2.

```
PendSV_Handler,
SysTick_Handler,
0,
0,
0,
0,
0,  // 20
0,
0,
0,
0,
0,  // 25
0,
0,
0,
0,
0,  // 30
0,
0,
0,
0,
Timer0A_Handler, // 35

};
```

Figure 3: Snippet of cstartup_M.c Vector Table

The handler was defined in the main_task1a.c file and uses a global variable to track the five configurations the LEDs could be in. Once the 1 second timer runs out, the interrupt is raised and the handler defines what to happen at each trigger.

Task 2b: For this task, we used timers to program a blinking LED1 at a rate of 1Hz. Then, we use SW1 and SW2 buttons to interrupt the program. When SW1 is pressed, the timer will stop counting down, LED1 will shut off, and LED2 will turn on. It will not return to the original pattern of LED1 blinking until SW2 is pressed. We implemented this task by going step-by-step. First, we used the cstartup_M.c file, timer.c file, and main_task1a.c implementation to build off of with some modifications. For example, the startup file needed to add a GPIO port J handler added to the declarations and vector table. To initialize the SW interrupts, we drew from lecture notes, the datasheet, and youtube videos on interrupts. The hardest part was figuring out how to

properly initiate the SW interrupts and thinking about how to switch modes during a button press. In the end, we were able to target the needed register settings.

Task 2c: In this task, we did a repeat of Task 1b, but we used interrupts instead of polling. To implement this, we combined Lab 1 Task 2 with the previous parts of Lab 2 Task 2. We added another timer, Timer0B, to keep track of the length a button was pressed and used Timer0A to track the seconds spent in each state without a button press. The priority settings were a very important part of this lab as the button presses had to have priority over Timer0A, but Timer0B had to have priority over the button presses. The second important factor was tracking a change in button presses. If a button was released, TimerA needed to start back up and TimerB needed to be reset and turned off. If a new button was pressed, TimerA needed to stay off, but TimerB needed to be reset and stay on. Each timer handler makes a call to the FSM function to determine what state to go to next.

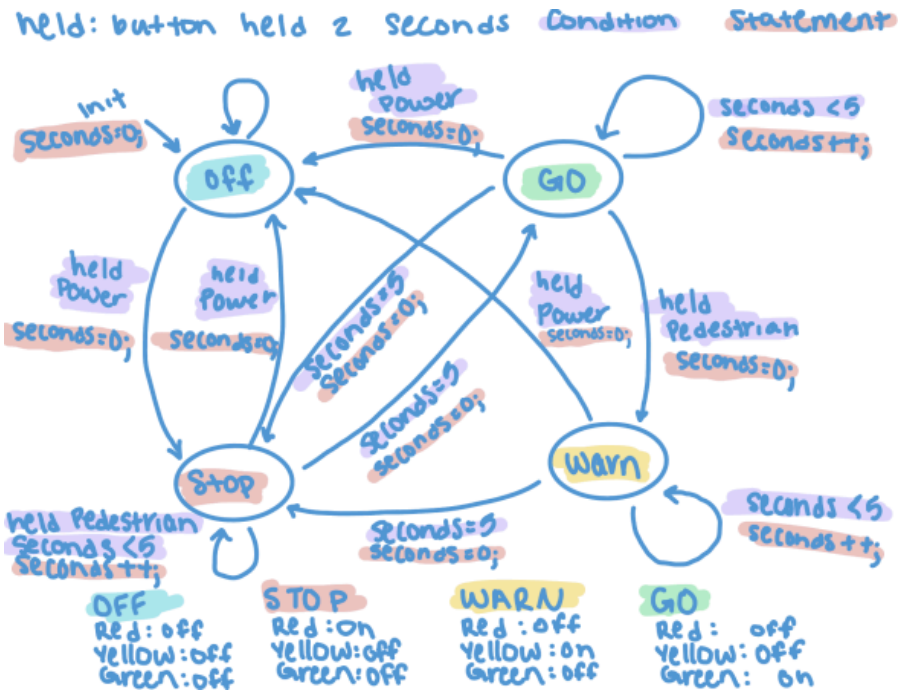

Figure 4: State Diagram of Traffic Light Controller

Results:

Task 1a: The onboard LEDs successfully cycle through two at a time from LED4 to LED1 with a one second delay in between.

Task 1b: As a result, when the on/off button is pressed and held for at least 2 seconds approximately, the system begins at the stop state, where only a red LED should turn on. After 5 seconds, the system will move to the go state, where only a green LED turns on. This happens indefinitely when no button is pressed. Then, when on/off is pressed and held for at least 2 seconds, then the system turns off. However, when the pedestrian button is pressed (also must be held for 2 seconds) while in go state, the system immediately goes to warn state and remains for 5 seconds before moving to stop and repeating the pattern.

Task 2a: The onboard LEDs successfully repeat Task 1a using interrupts, instead of polling.

Task 2b: As a result, when SW1 is pressed, the timer stops counting down and LED2 turns on. This consequently turns off LED1. On the other hand, when SW2 is pressed, the timer counts down again, then returns LED1 to its original behaviors while LED2 turns off.

Task 2c: The timed traffic light also works with interrupts. When the on/off button is pressed and held for approximately 2 seconds, the system turns on only the red LED (the stop state) and transitions to only the green LED (the go state) after approximately 5 seconds of no button presses (happening indefinitely when no button is pressed). At any point while the system is on, if on/off is pressed and held for at least 2 seconds, then all LEDs turn off. When the pedestrian button is pressed and held for 2 seconds while in go state, the yellow LED turns on and remains on for 5 seconds before moving to stop and repeating the pattern. If the pedestrian button is pressed while in the stop state, the traffic light stays in the stop state. If a button is pressed, but not for 2 seconds, the traffic light controller stays where it is and the timer counts down again to move on to the next state.