Ze and Caitie

Lab 2 Report

Part 1: Procedure

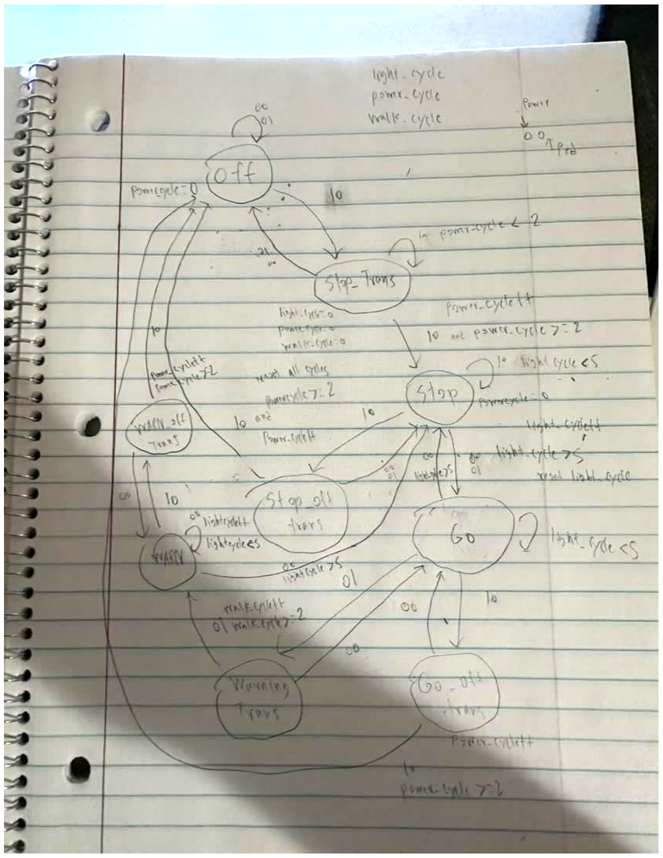
Task 1: General Purpose Timers

1a: Periodic LEDs with Polling

To do this task it was necessary to know the LEDs which would be blinking, the method of turning them on and off for a blinking effect, and the rate at which the timer would need to go. Therefore, from reading the Lab 2 specifications, it was understood that the LEDs would be from Port N for LED 1 and the method would be polling with general purpose timers at a rate of 1 Hz which corresponds to a value of 16,000,000.

The code was divided into logical functions that matched this: initialize the timer, initialize the lights, a polling method, and a main method to run the code. In the header file, the registers were divided up into logical sections so it would be easy to keep track. Then the code was commented with information from the datasheet and spec for easy corrections and adjustments.

1b: Timed Traffic Light with Polling

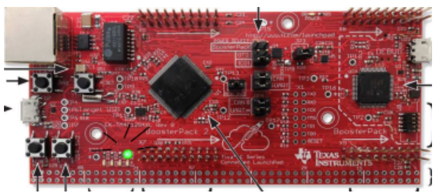


The FSM to the left captures this behavior with the theoretical states of the lights as well as the practical states including transition and off transitional states. This implementation is credited to TA Pedro, for showing me his states and the process to how he implemented his traffic light. It initially began with only 7 states but after finding out that it couldn't store the previous state of an enum two more additional states had to be added to go from a transitioning state to an actual off state. Therefore, it totaled 9 states, each one deciding on which state to go to depending on which button is being pressed and how long the timer has gone.

The timer is tracked by using 3 counter variables: walk_cycle, light_cycle, and power_cycle. For example: from the Go state if the power button was pressed, we would move to the GO_Off transition. Here we check if the power button has been pressed long enough by incrementing power_cycle. If the button is held long enough and it reaches 2 it will turn off, however, if it didn't reach 2 seconds and the button was released it would go back to its previous state, which is GO. If we are holding the button but we haven't reached 2 seconds the power_cycle will just continue to increment. After a transition has happened we want to reset the counter. I did this for each of the states, STOP, WARN, GO. Then the actual states themselves had a 5second check before they transitioned to other states.

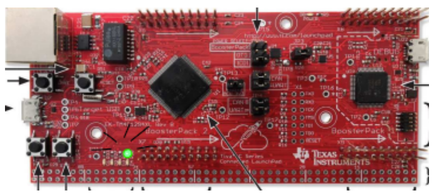
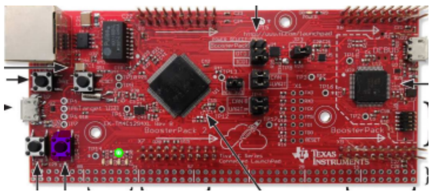
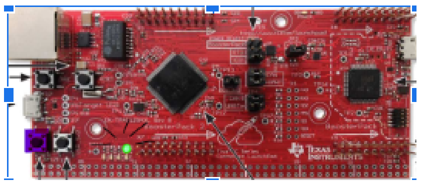
Task 2: Interrupts

2a: Periodic LEDs with Timer Interrupts



For task 2a, the lights were instead to be programmed blinking with interrupts instead of polling. So, from the GPIO I/O controller, the interrupt was set up and then the ISR was able to run the code by itself.. The blinking and registers were the same but instead of polling the framework was designed to handle the ISR. The code was designated into functions with an initialization of the timers, lights, and ISR and then the main code to run the program.

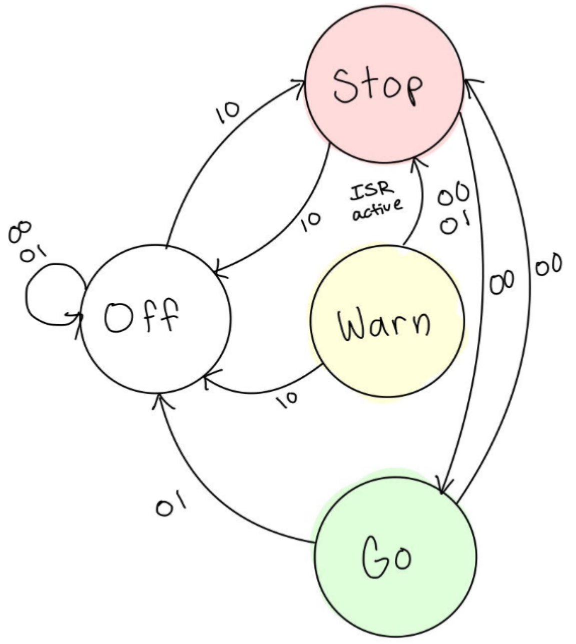
2b: Controlling Timer from Switch

Task 2b elaborated on Task 2a in the sense that the light still blinked the same way as before but if the user interacted with the SW1, then a second light turned on instead until SW2 was interacted with in which case the blinking pattern resumed as it was before being interrupts.

As the program was very much the same as Task 2a, the code was still divided into the function to initialize the timer, initialize the I/O framework, and initialize the lights and port handler. The extra functionality of the switches was given its own function for the sake of keeping the code clean and organized.

2c: Timed Traffic Light with Interrupts



This task required the traffic light be created with interrupts as opposed to polling and various timers. This meant rather than a cascade of timers for the pedestrian button being clicked and triggering timers for the warning and stop lights, the pedestrian button was treated as an interrupt in the system and triggered its own behavior. The code was organized with the interrupts in its own section and the other methods as was. A rough drawing of the altered FSM is shown. Without the need for timers to count all the different transitions between states, the FSM would have been far simpler due to the states changing upon the interrupt routine as opposed to manually creating each change of state.

Part 2: Results

Task 1: General Purpose Timers

1a: Periodic LEDs with Polling

Task 1a did not reach full functionality due to an improper counting down the TACDIR bit when XOR was being as well as = instead of &=, to capture the other possibilities of the bit before the count down. Therefore, the program was running infinitely and was not polling correctly. Alternatively, the code could have been improved by the proper polling. Furthermore, initially, the project was not using the proper project settings with TI Stellaris which caused a delay in the coding.

Had the system worked properly, initially, the user would have seen the LED1, at the bottom of the board, blink at a rate of 1Hz which is equivalent to roughly 1 second. The light would have blinked indefinitely with no interrupts or changes on the system at any point.

1b: Timed Traffic Light with Polling

The timed traffic light *did* reach full functionality. The light was programmed to have a total of 9 states as it was also taking into account the transitions to the stop state, as well. The system showcased a red light for the STOP state, a yellow light for the WARN state, and a green light for the GO state. The system properly waited 5 seconds in between the GO and STOP states, unless the pedestrian button was pressed and then the WARN state was triggered for 5 seconds which transitioned to the OFF state. The system also properly waited at least 2 seconds for button inputs.

Task 2: Interrupts

2a: Periodic LEDs with Timer Interrupts

The lights with the timer featured the identical behavior as from part 1a with no exceptions. The light blinked at the same rate as well as well there being exceptions present to the light blinking continuously.

2b: Controlling Timer from Switch

The behavior from the switch takes the situation from 2b and adds extra functionality. Therefore, when the user goes to use the system, they notice the LED1 blinking as it was. However, if the user decides to press SW1, the notice that LED1 turns off (regardless if it was in the state of On or Off) and LED2 turns on instead. This LED2 does not blink and instead is solidly on. This behavior continues until the SW2 is triggered in which case the original blinking behavior resumes as if nothing had interrupted it to begin with. This behavior basically means that SW1 acts as a pause for LED1 as it interrupts with LED2 until the user decides to return with SW2.

2c: Timed Traffic Light with Timer Interrupts

While 2c did not reach full functionality, the lacking components were transitioning to all of the states properly. As the program was handling the finite state machine with an interrupt system as opposed to timers, we were faced with confusion when trying to find when to clear the flags or how to handle the port handler. We understood that we needed two timers for the state transitions but were unable to fully implement them.