|  |  |  |  |
| --- | --- | --- | --- |
| UMass-Lowell-logo.png (295×358) | **Lab #1** | **Traffic Light Controller** | |
|  |  | |
| Ronan Dunn | | |
| 01757311 | | |
| EECE.4520.201 – Microprocessors II | | |
| Team Member Names: Patrick Hoey & Peter Lynch | | |
| Date of Completion: September 27, 2021  Demonstration Method: Zoom | |  |

**1. Design**

**1.1 Hardware Design**

The hardware for this lab was documented in a schematic using MultiSim. The Arduino MEGA2560 microcontroller with the components shown in table 1 were used to accomplish this lab.

**Table 1. Components**

|  |  |  |
| --- | --- | --- |
| **Component** | **Quantity** | **Details** |
| LED | 3 | Red, Yellow, Green |
| Resistors | 3 | 1 kΩ |
| Switch | 1 | - |
| Buzzer | 1 | Active |
| Arduino MEGA2560 | 1 | - |

Diagram

Description automatically generated

**Figure 1. Schematic of traffic light controller**

The traffic controller used three LEDs, one corresponding to each color of a traffic light – red, yellow, and green. The resistors used for the LED connections were 1 kΩ. The switch was connected to pin 20 because it allowed for interrupts while the other pin connections to the MEGA2560 were arbitrary. Finally, an active buzzer was connected to pin 5. The choice of 1 kΩ resistors was made for low power consumption to maximize the life of the LEDs.

**1.2** **Software Design**

The traffic light controller is designed to operate a single traffic light for one direction. The red and green lights are to remain on for 20 seconds while the yellow light remains on for only three seconds. Additionally, an active buzzer sounds three seconds before a light change. This state of operation begins once the user click the switch. Until this occurs, the red light blinks every second in an initial state. This process is described by the state diagram as shown in figure 2.

Text

Description automatically generated

Diagram

Description automatically generated

**Figure 2. Traffic light controller state diagram**

The notation used in figure 2 is consistent with that of *Introduction to Embedded Systems: A Cyber-Physical Systems Approach* to remain within the bounds of the course textbook. The only input in this system is when the user clicks the switch to exit the initial state and start the traffic light operation. The two variables used in the software are time and started. Time keeps track of the current time while the red, green, or yellow LED is on. Separate states exist for an LED on, and the buzzer sounding with that LED on to fairly represent the “buzzing” as a state in the diagram. Lastly, the outputs in this system are all pure – either they are present or absent.

Pin assignment was discussed in section 1.1, which states that all pin choices were arbitrary with the exception of pin 20, in which the switch was connected to allow for interrupts.

In short, the system starts with the red LED blinking on and off every second. Once the user clicks the switch, the system enters the Red state in which the red LED is on. After 17 seconds, the system will sound the buzzer to enter the state of Red & Buzzer. Then, after three seconds, the systems signals the green LED and enters the Green state. The same process repeats for the Green and Green & Buzzer states as in the Red and Red & Buzzer states. Finally, the system will enter the Yellow & Buzzer state for three seconds before returning to the Red state.

An important software consideration was how to implement each of these states. The states used in software include INIT, STOP, SLOW, and GO. Referring to the states in figure 2, INIT represents the transition between IDLE and Red before the switch is clicked. STOP includes the Red & Red Buzzer states, SLOW includes the Yellow & Buzzer state, and GO includes the Green and Green & Buzzer states. This generalization of the states allowed for much easier control flow in the program without having to redundantly create extra states. More specifically, the logic used in the Red state was also used in the Red & Buzzer state with additional logic for the buzzer and transition.

The source code1 for the Traffic Light Controller can be located here: <https://github.com/UML-Micro-2/lab_1>

1. This lab was completed in C/C++ and in Arduino to gain a deeper understanding of how the Arduino IDE works and how it can be used with C/C++ compilers. Hence, the GitHub repo contains the C/C++ file with an Arduino header file in the src folder. The Arduino .ino source code written in the Arduino IDE is located in the src-ino folder. The logic and functionality of the two source codes are the same.

**1.3 Results**

This lab produced the results expected and within the requirements. When the circuit was powered on, the red LED blinked until the button was pressed. Once pressed, the red LED remained on for 20 seconds. Three seconds before switching to the green LED on, the buzzer sounded for three seconds. This same process repeated for the green LED and yellow LED, with the yellow LED and buzzer on for only three seconds. After the yellow LED remained on for three seconds, the red LED appeared, and the loop started over. This control flow is represented by the chart in figure 3.

Diagram

Description automatically generated

**Figure 3. Control flow of the traffic light controller**

**2. Problems Encountered and Solved**

Although this lab was relatively simple due to its introductory components, there were still some impediments that were solved through different approaches.

**Interrupts –** First, while working within the Arduino IDE, the switch was connected to pin 6. When this connection was established, interrupts were not working when the switch was pressed. After a quick research of the Arduino MEGA2560, the interrupt hardware only exists on pins 18, 19, 20, and 21. This was solved quickly, and the button click were handled appropriately.

**General Debugging** – While using the Arduino IDE, debugging the program to see the current time and state was difficult at first sight. However, the serial monitor was utilized, and the information needed for debugging was written to the monitor to easily track the necessary variables.

**Setup** – Initially, an interrupt was attempted when the switch was clicked but was not properly set up. Just as pins are setup for input and output, interrupts for specific events need to be attached to a pin and the handling function. This setup was fixed by adding the line of code: attachInterrupt(digitalPinToInterrupt(SWITCH), start, LOW); This effectively setup the interrupt, which began working afterwards.

**3. Personal Contributions to the Lab**

Ronan Dunn

* Designed and created the state machine
* Wrote the loop() control flow function to implement the state machine
* Debugged timer tracking to ensure timer was accurate with interrupts
* Wired up circuit

Patrick Hoey

* Drew the schematic in MultiSim and Altium
* Implemented state machine in C/C++ with <Arduino.h>
* Implemented interrupt for switch press
* Wired up circuit

Peter Lynch

* Determined how to track the time using interrupts
* Implemented the interrupt to control the timer
* Debugged and implemented the initial state switch from INIT to STOP
* Wired up circuit

**4. Lessons Learnt**

There were several lessons learned throughout this lab. First, interrupts in Arduino are dependent on the board’s hardware. In this lab, the Arduino MEGA2560 supported interrupts on pins 18 through 21. Next, in order to debug in the Arduino editor, the serial monitor can be used to output the value of certain variables at specific times throughout the program. It was also discovered that Arduino programs can be written in the Arduino IDE and in C/C++ with the use of the Arduino.h header file. While both methods produce the same results, writing the program in Visual Studio with a .ino extension appears to be easier on the surface in terms of debugging and autocompleting functions.

Further, the setup() function initializes interrupts and establishes the pins for input and output. This is necessary for the pins to carry out the proper functionality during the program. While an interrupt can be attached to an event, such as the switch click in this lab, interrupts can also be handled periodically. As shown in this lab, the timer1 interrupt is created at 1 Hz with multiple registers needing to be initialized such as OCR1A and TCCR1B for interrupt to work.