Lab 3 Report

Kerollos Lowandy, Pablo Ruiz, Mohammed Khassim
Completion Date: 04/20/2021
Microprocessor II and Embedded System Design
Lab 3: Controlling a Fan

1.INTRODUCTION:

In this lab, our goal is to power and control a DC motor indirectly through a L293D driver and display the current status of the motor and the current time on an LCD screen. Additionally, two methods of user inputs are required, a push button to toggle its rotational direction between clockwise (CW) and counterclockwise (CCW) and an IR receiver and remote set to allow the user to adjust the speed of the motor as well as change direction. The corresponding buttons for these functions on the IR remote are volume +, volume -, and play/pause respectively. Furthermore, a real time clock (RTC) was used to maintain an accurate time even when the system is powered off. Communication with the RTC was achieved through an I2C digital interface. Finally, a liquid crystal display (LCD) screen was used to display the direction and percent of maximum speed as well as the current time. Several hardware and software challenges were encountered during this lab and are discussed and addressed in the following report.

2. DESIGN:

2.1. HARDWARE:

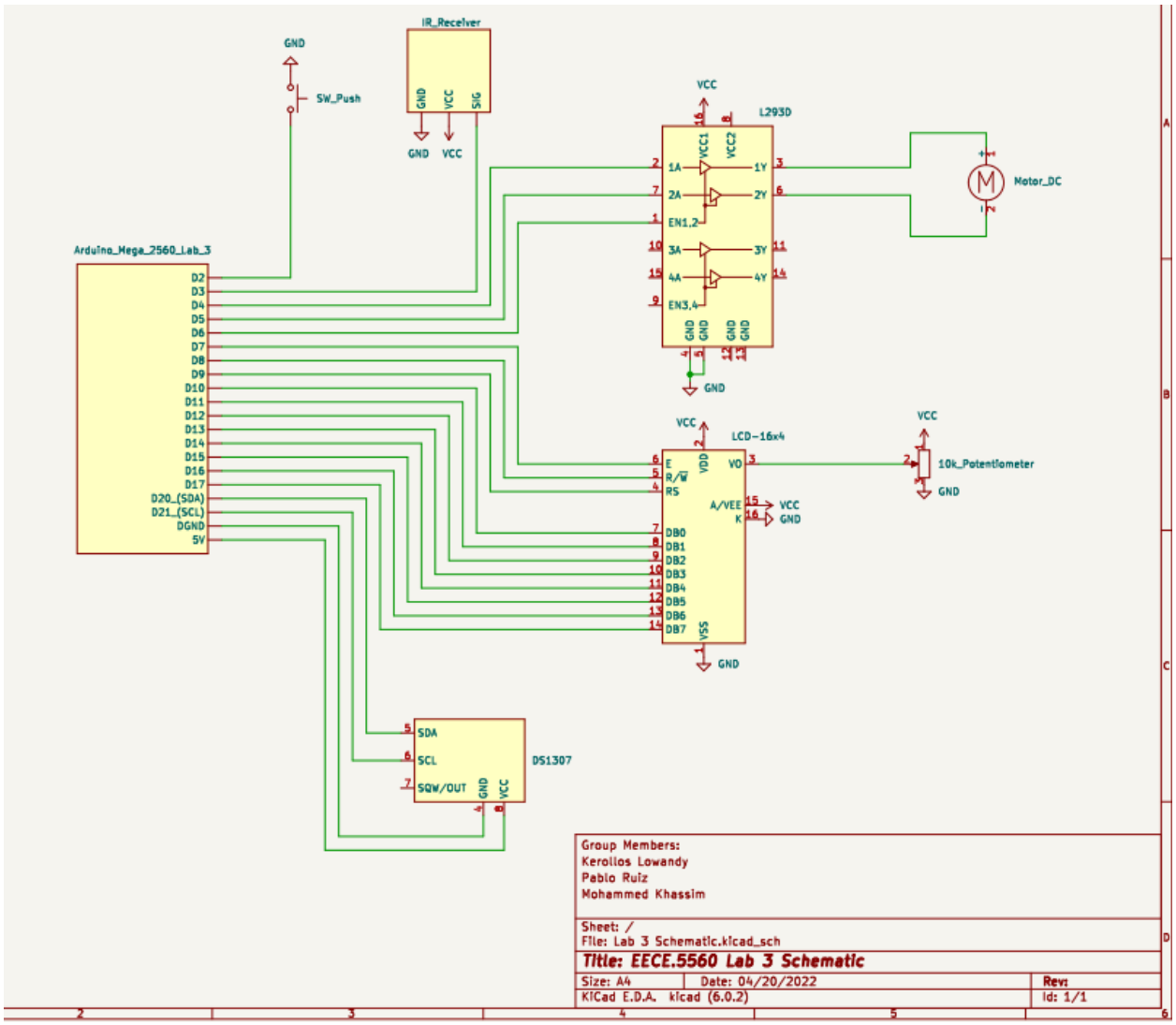

Figure 1. Lab 3 electrical schematic.

This lab required 7 different components and 15 digital I/O pins on the arduino with at least one of them being capable of outputting PWM signals as well as the I2C SDA and SCL lines. The first element, the pushbutton, connected to pin D2 on one end and was connected to ground on the other. The internal pullup resistor within the arduino was enabled which meant the pin would normally be high and only go low when the button is pressed. Next, the L293D dual half-H driver was used to drive the motor since the Arduino board itself can't supply

enough current through its I/O pins. Furthermore, the L293D allows us to reverse the polarity of the motor and thus its rotating direction simply by flipping its 1A and 2A from high to low or vice versa. Additionally, the voltage output to the motor through the L293D can be modulated by a PWM signal on pin EN1,2, this results in the ability to control the motor speed. Power for all the components except for the DS1307 RTC was provided through a power supply which is not pictured in the schematic. The RTC was powered through the Arduino due to its low power requirements and in order to prevent any noise in the I2C bus affecting the time readings. The IR receiver receives 38kHz pulses of IR light from the IR remote and outputs a binary code which is unique to the button pressed. For this reason it only requires a single signal line to the Arduino. Ultimately, the 16x2 LCD panel required the highest number of pins to control and an additional 10k potentiometer connected to its V0 pin to adjust the contrast of the display characters so that they would be clearly visible.

2.2. SOFTWARE:

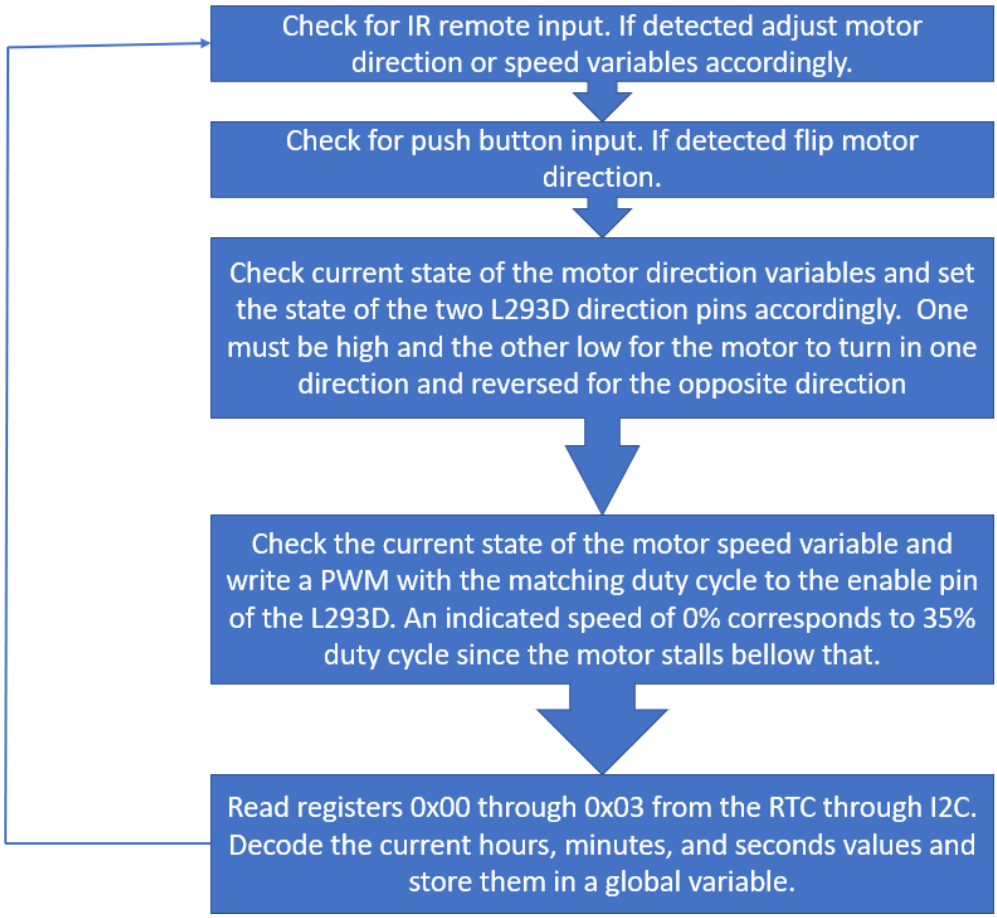

Figure 2. Main loop logic flow chart.

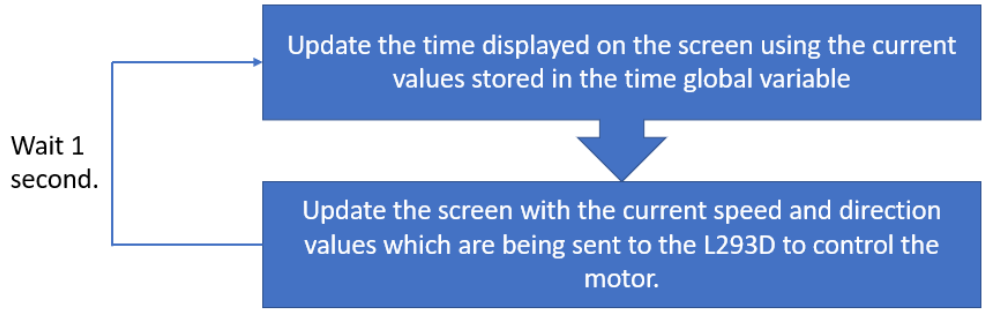

Figure 3. Screen update timer interrupt logic flow chart.

The software for this lab was divided into two main components, the main loop which repeats endlessly and the timer interrupt service routine (ISR) which only runs once every second. In order to set up the timer interrupt we manipulated the ATMega 2560 registers directly to select Timer 1 which has a resolution of 16-bits. Due to the onboard oscillator running at 16 MHz which is much too fast for 1 second intervals when the timer is only 16-bits we used a 1024 prescaler which decreases the effective oscillator frequency to a more manageable 15625 Hz. Finally, we enabled clear timer on compare (CTC) mode which resets the timer counter every time that it reaches the value of the compare register, in this case 15625 since we are aiming for 1 second intervals. The sole purpose of the timer interrupt is to update the screen at strict 1 seconds intervals therefore providing an equivalent refresh rate of 1 FPS. In order to have the ISR adhere to the timing constraints and return control to the main loop as fast as possible, all calculations and/or sensor polling/communication were placed in the main loop and the ISR's only responsibilities are to update the screen with the latest stored values.

The main loop is where the user inputs are polled and processed as well as all the state variables and output variables are updated according to those inputs. This allows the program to feel responsive even if the screen is only updating at 1 FPS. First the IR receiver is checked for input and if detected the resulting hexadecimal code is compared to the codes of our three valid inputs volume +, volume -, and play/pause whose side effects are motor speed +5%, motor speed -5%, and direction switch (CW to CCW and vice versa) respectively. Next, the push-button is polled with an included debounce routine which uses the change of state and a minimum duration for each press to prevent unwanted inputs. If a button press is detected the motor's direction is switched as outlined previously. Subsequently, the digital outputs corresponding to the 1A and 2A pins on the L293D driver, as seen in *figure 1*, are set according to the current state of the direction variable. These outputs determine in which direction the motor will turn and must be opposite to each other, ie. if 1A is high, 2A must be low and vice versa. Additonally, in the following commands the current value of the velocity percentage (0 to 100) is multiplied by 165 and then divided by 100 and 90 is added to the result. The resulting integer will have a value between 90 and 255 which represents a PWM signal with a duty cycle of 35% to 100% respectively. This PWM signal is written to the digital I/O pin connected to the EN1,2 pin on the L293D which then adjusts the voltage going to the DC motor accordingly thus controlling its rotational speed. Finally, the stored time is updated by communicating with the real time clock (RTC) through the I2C interface. The values of the one byte registers starting

from address 0x00 and ending in address 0x03 are read and then decoded to extract the seconds, minutes, and hours values respectively. The 4 least significant bits of each byte represent the ones position of each value while the 4 highest significant bits represent the tens position of each value, ie. 0100 0001 would be decoded as 41. Following this action the program would loop and restart the main loop from the beginning until it is interrupted by the screen update timer ISR discussed above.

Github Link:

https://github.com/tame-potato/Micro-Lab-3

3. RESULTS:

Demo Video Link:

https://studentuml-my.sharepoint.com/:f:/g/personal/pablo_ruiz_student_uml_edu/Enh5hxOqTN5BuT_xnOpl6OEBBq-3DssvCkpAt9GvzGVg5Q?e=c4icZ1

4. DISCUSSION:

This lab did have a few difficulties that we had to troubleshoot through. At one point our speed control would only activate the fan when it was set to 100% and would turn off at anything less. We figured out that this came from the way we set up the math to determine the analogWrite() value. We had 255 be the equivalent of 100% duty cycle as this is the maximum output of analogWrite() which is about 3.3 V. We then wrote that analogWrite should take in the output of the equation (percent value)/100*255. The reason this did not work the first time is because we had declared these variables as integers and thus, integer math was done. Since all percent values that would be inputted are less than 100, when integer math is done, analogWrite() would always take in 0*255 until the percent value reached 100- where it would thus be 1*255. Once we rearranged the order of the operands to be (percent value)*255/100 we obtained the desired result and were able to modulate the speed of the fans through the L294D driver.

Another problem that we faced came in the communication between the I2C and the RTC. Our RTC was originally grounded at the power supply which resulted in some noise, however grounding it at the arduino instead solved the issue. This shows the importance of grounds in more complex circuits like this one which utilize digital communication buses, such as I2C, in ensuring that all parts of the circuit function properly. Ground is our reference point and without a common reference point, voltage values essentially have no meaning.

Finally, whenever a button was pressed on the remote, the IR receiver would return a unique hexadecimal code depending on what button was pressed. We needed to find out which hex code corresponded to which button so that we could use those codes in our

conditions for when a button was pressed. We tested this by writing the output of the IR receiver to the serial monitor. Doing this allowed us to read the hex code produced by each unique button press. We then took this information and were able to link each button's desired side effect to its associated hex code.

On the software side, we debated including the RTC I2C time update request inside our timer interrupt, but ultimately kept it inside the main loop and only wrote the latest stored values to the LCD during the interrupt. This was done to simplify the code within the interrupt routine as much as possible to speed up its execution making it more likely to abide by its hard 1 FPS timing requirements and also return control to the main program logic as soon as possible to minimize the time during which user inputs were not being processed.

Another observation that we made was the minimum required PWM duty cycle sent to the L293D for the motor to produce enough torque for the fan to turn. Through testing this with the complete design, we were able to find that at 35% duty cycle was when the fan would start spinning. Therefore we decided to map the minimum velocity level that the user can select (VEL%:0) in the LCD to a 35% PWM duty cycle so that the fan never stalled while still receiving voltage in order to prevent a possible malfunction of the DC motor.