Lucas Black

CPE 301.1001

Final Project

Overview

The purpose of this project was to develop and build an evaporative cooling system, also known as a swamp cooler. This system processes the cooler with a state machine on an Arduino ATMega2560, where the current cooler state is indicated by four individual LEDs. The components used for this project is a L293D IC for driving the DC motor (i.e. fan), a DHT11 temperature/humidity sensor, DS1307 RTC module, 16x2 LCD, stepper motor, stepper motor driver, water sensor, a set of RGBY LEDs, a single push button, two potentiometers, a power supply module, and a handful of resistors. The completed system works by monitoring the water level of a reservoir, turning the fan on only if both the air temperature and water level are above certain thresholds. It can be powered off and on with a button, effectively turning the fan off and on. The current time and date are recorded via Serial when the fan motor turns on and off, as well as when there are any state transitions. The power requirement is 5.0 V for all components except the water sensor which operates at 3.3 V.

There are red, green, blue, and yellow LEDs to indicate the state of the system. The yellow LED shows it's in a "disabled" state and the system is off. The green LED shows it's in an "idle" state and the system is on. The blue LED shows it's in a "running" state and all environmental conditions have been met. The red LED shows it's in an "error" state in order to display there isn't enough water. Yellow, green, blue, and red LEDs are wired to PWM pins 7 through 10, respectively. Then, a method called `write_led()` allows a specific LED to be powered on during a state change.

Controlling the vent is fairly straightforward and can be controlled with a potentiometer. I used the stepper motor kit and a library called `Stepper.h`, connecting the output pins from the stepper driver board to digital pins 34, 35, 38, and 39. Then an analog value is read from a potentiometer on analog pin 8 using `adc_read()` and writes the appropriate step value to a stepper object defined as `mystepper`. The stepper is controlled with the method `read_pot_write_stepper()`. Lastly, the vent can only be controlled during the "idle" and "running" states.

The fan is a single DC motor powered on by a L293D IC when the temperature and water level constraints have been met. The pins connected to the L293D are defined by `fan_enable`, `fan_input1`, and `fan_input2`. Depending on the given state, there are two methods for turning the fan on and off, `turn_fan_on()` and `turn_fan_off()`. Anytime the stop

button is pressed or there is an error with the water level, the fan is turned off. As well as, the motor is connected to a power supply module separate from the Arduino.

The DHT11 temperature/humidity sensor is connected to the Arduino via a single wire to PWM pin 6. I used a library called `SimpleDHT.h` to interface with the DHT11, creating an instance from the library defined as `dht11`. The temperature sensor is used when turning the fan on and off. The data is collected from the sensor using a method called `read_DHT11()` and then displayed on the LCD, allowing the user to monitor the surrounding air temperature and humidity. The temperature threshold for the fan to turn on or off is 72°F.

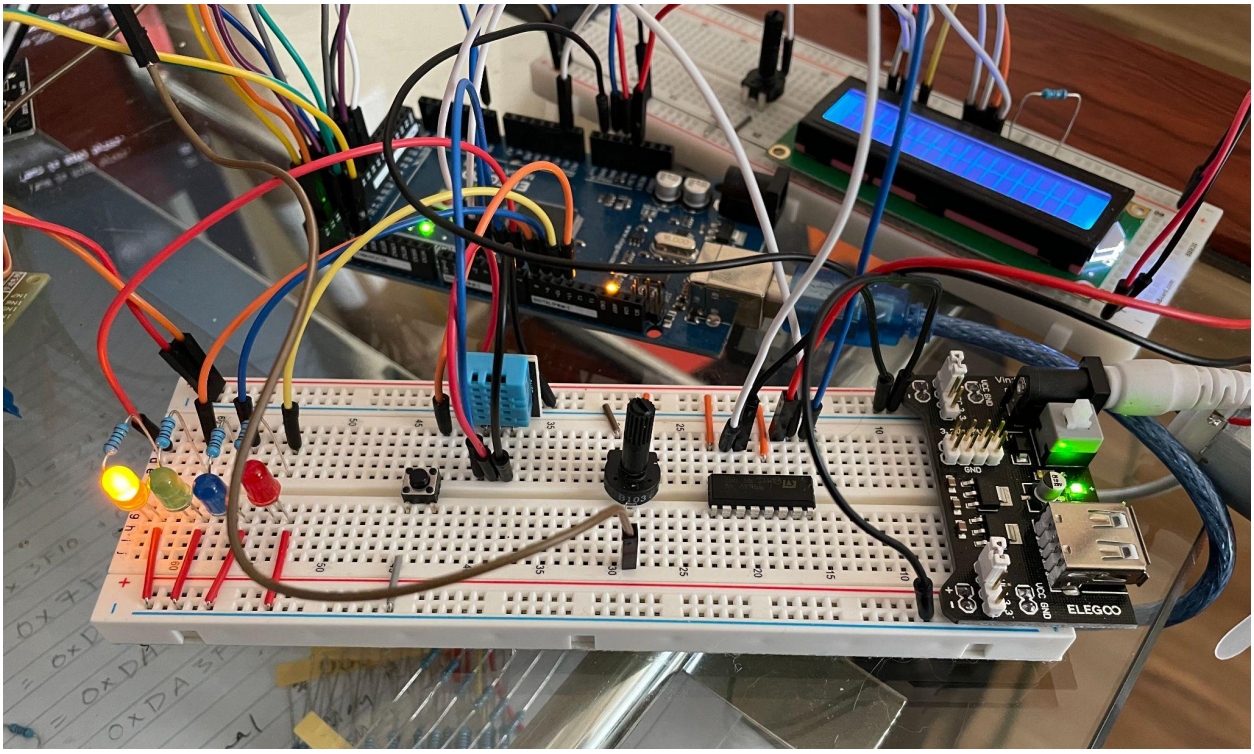
The DS1307 RTC module is used to keep track of the date and time for event reporting. The module is connected to the Arduino via I2C, and the date and time information is written via Serial. The module allows for the time and date to be recorded whenever the fan motor turns on and off, as well as whenever there are any state transitions. I used a library called `RTClib.h` to interface with the DS1307 RTC module and defined an instance as `rtc`. Then the method `serialPrintDateTime()` prints a timestamp using UART and the `rtc` instance.

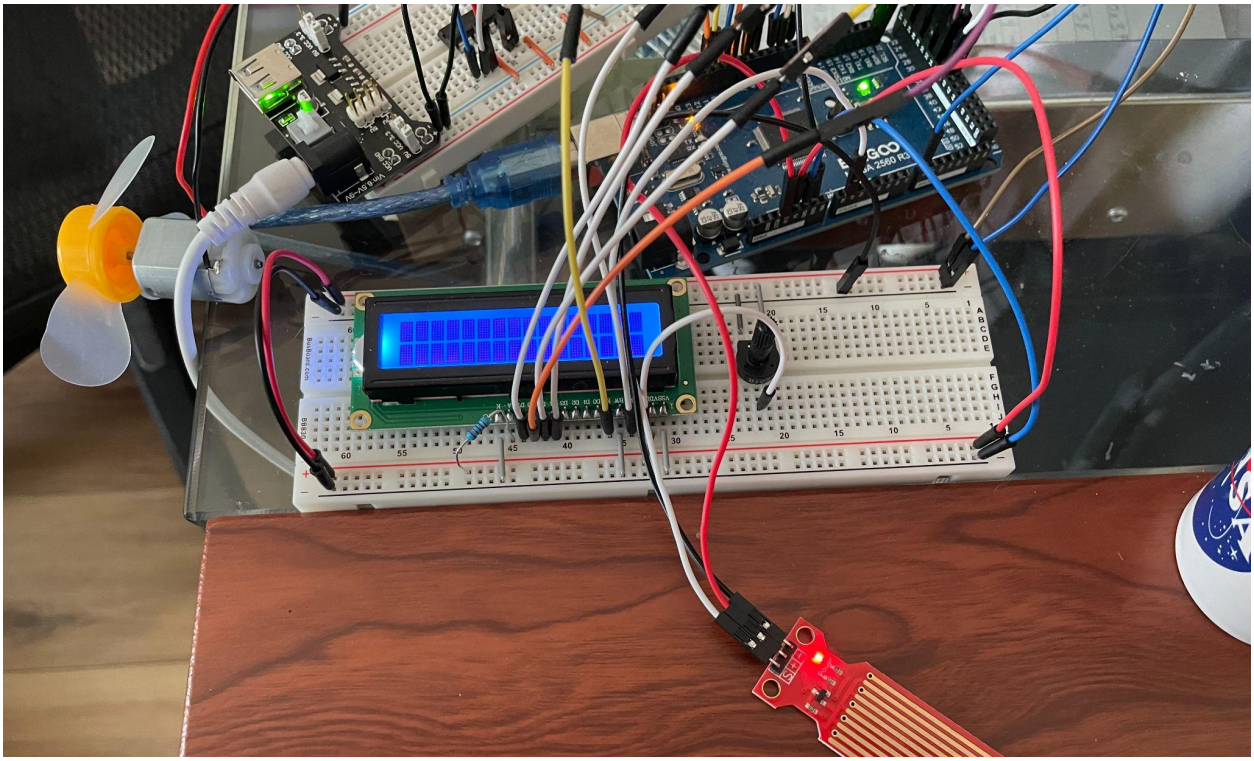
The LCD is used to display relevant information to the user, such as the current temperature and humidity levels, or any errors that occur such as low water level. I used the library `LiquidCrystal.h` to interface with the LCD and defined an instance as `lcd`. When an error occurs, the message, “Water level is too low!” is displayed to the screen. When the system is in a disabled state, the LCD display is cleared of any information. Then, the temperature is displayed with the temperature and humidity levels using a method called `write_data_to_lcd()`.

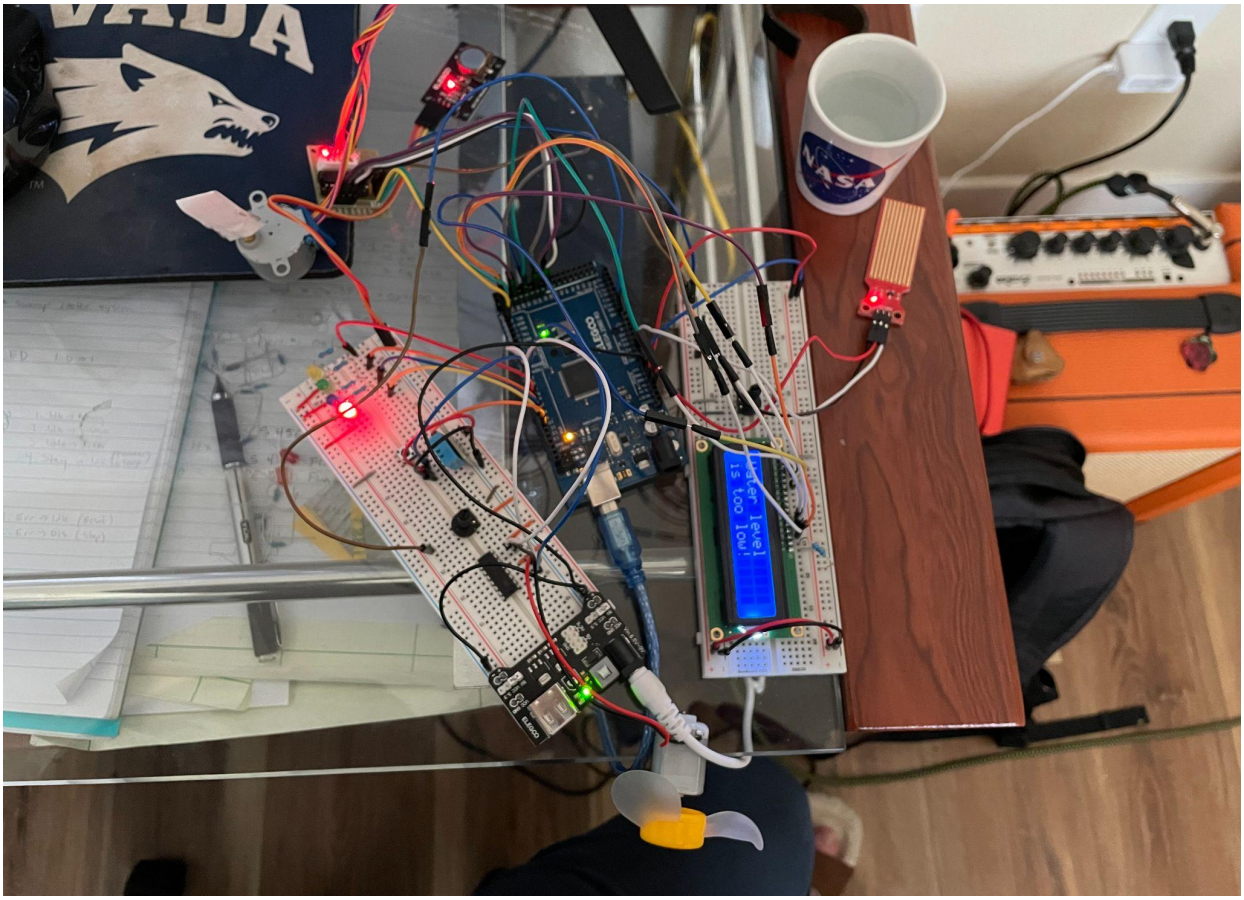
The water sensor is used to detect the level of water in the reservoir, and is connected to the Arduino via analog pin 1. When the detected water level is lower than the water threshold, it sends the system to the error state. If water level is above the threshold, then it allows the cooler system to transition to “idle” and “running” states. The water threshold is defined as the variable `water_threshold` and the water level is defined as the variable `water_level`. Water levels are continuously monitored except during the “disabled” state. I defined a method called `read_water_level()` which simply returns an integer value using `adc_read()` with a prescaler.

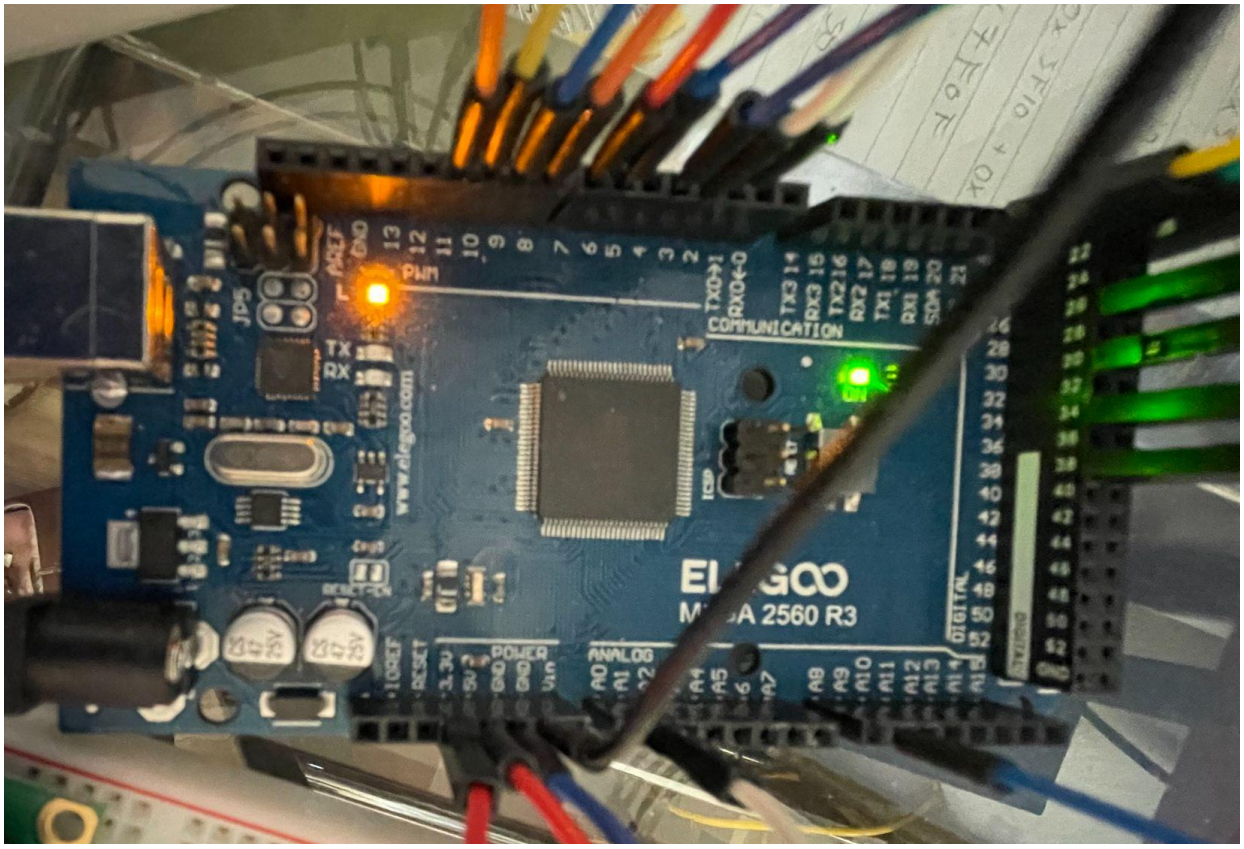
Finally, a push button is used to power or reset the system using an ISR. I enabled a pullup resistor and PCINT0 interrupts on PB5 (PWM pin 11). Then the system debounces the button whenever a PCINT0 interrupt is triggered on PB5 using the set of variables `button_timer` and `previous_button_timer` to do so. A variable called `SYSTEM_TOGGLE` indicates the button was pushed on debounce, allowing any state changes that require button presses to occur.

Photos:

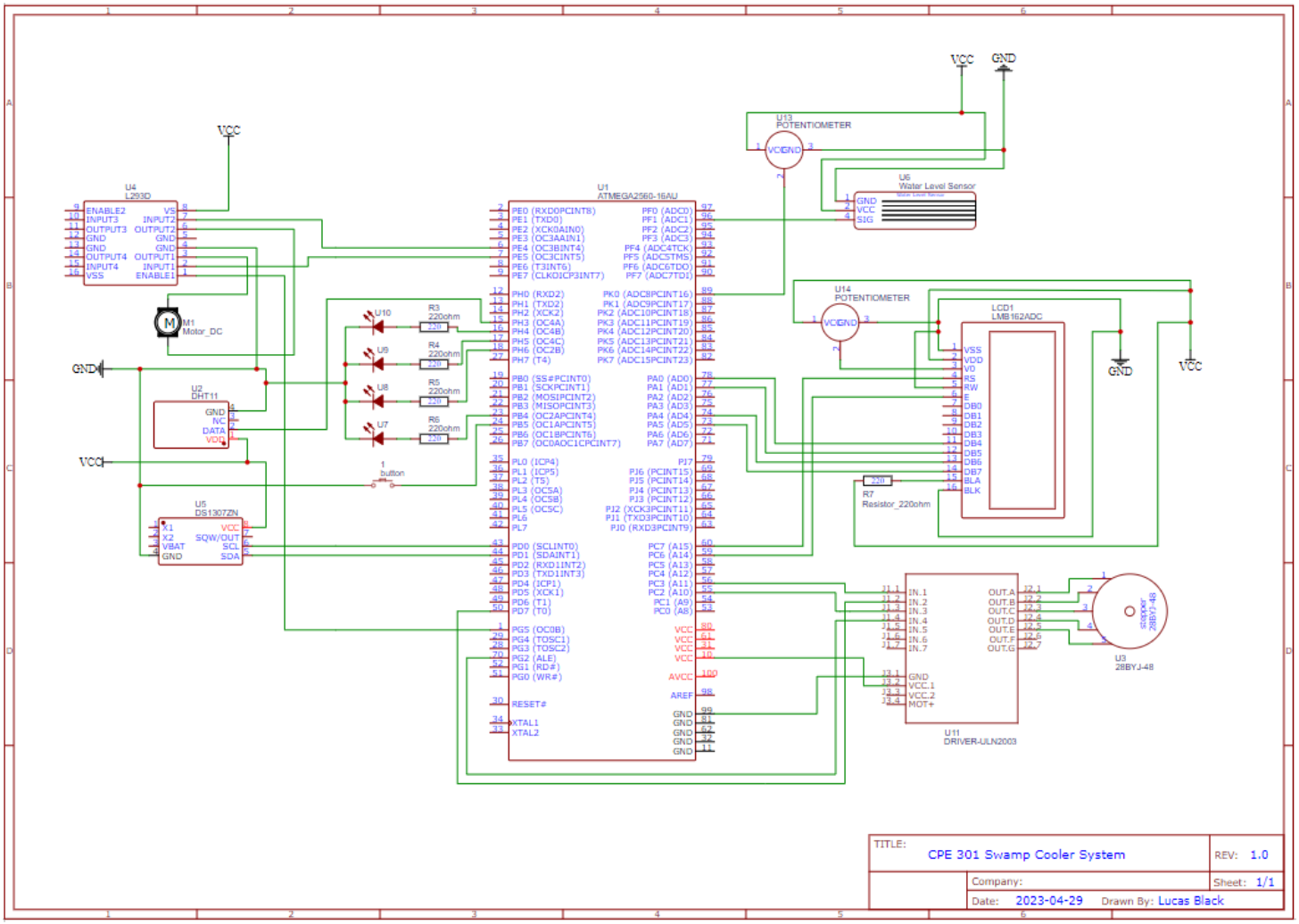





Water level
is too low!


PWM
COMMUNICATION
POWER
ANALOG
DIGITAL

Schematic:



Github Repository: https://github.com/lbblack/CPE301-Final

Demonstration Video (YouTube Link): https://youtu.be/whaf6ygajNw

Specification Sheets:

Arduino ATMega2560 - https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

DHT11 - https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

LCD Display - https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf

L293D - https://www.ti.com/lit/ds/symlink/l293d.pdf?ts=1682925752729&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FL293D

DS1307 - https://www.analog.com/media/en/technical-documentation/data-sheets/DS1307.pdf

ULN2003 - https://www.ti.com/lit/ds/symlink/uln2003a.pdf?ts=1682964128633&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FULN2003A