CPE 301 Final Project

Richard Gunkel

Table of Contents

Project Overview

The goal of this project was to provide a basic prototype for an Arduino-controlled water cooler. This involved designing input circuitry and programming such that appropriate environmental variables could be monitored and certain external components such as the angle of a vent and fan speed could be controlled. Additionally, user interaction was added to the system through the implementation of push buttons and visual displays.

Components

Equipment/Components	**Parameters/Electrical Characteristics**
Arduino MEGA 2560 Controller Board and USB Connection Cable	256kB Flash, 4kB EEPROM, 8kB RAM (86) General Purpose I/O Pins (12) 16 bit PWM Channels (4) Serial UARTS (16) ADC Channels
(2) Full Size Solderless Breadboards	-
Breadboard Connector Wires	Jumper Wires, Female-to-Male Dupont
Elegoo Starter Kit: Breadboard Power Supply Module	3.3V/5V Output Voltage 700mA Max Output Current
Computer/Laptop Workstation with USB Port Connection	Arduino IDE Software Cirkit Designer 1.1.0
ELEGOO Starter Kit: DHT11 Temperature and Humidity Module	16 bit, 3.5V~5.5V DC Input 0.3mA Measurement, 60µA Standby Sampling Period > 2 sec Range at 25°C: ±5% RH, ±2°C
Elegoo Starter Kit: Water Level Detection Sensor Module	5V DC Input, <20mA Working Current 40mm x 16mm Detection Area 10°C to 30°C, 10% to 90% Humidity Output Voltage Range: 0V~4.2V
DS1307 Real Time Clock Module V03	64x8, Serial, 4.5V~5.5V DC Input 1.5mA Active, 0.2mA Standby Operating Temperature: 0°C to +70°C Output Pin Voltage: -0.5V to +7.0V
(5) Push Buttons	-
(4) LEDs - Yellow, Red, Green, Blue	5mm diameter
Resistors	(9) 220Ω, (1) 110Ω
ELEGOO Starter Kit: DC Motor + Fan Blade	3V~6V DC Input
L293D Push-Pull Driver	4.5V~36V DC Supply Voltage

	24mA Max Supply Current 7V Max Input/Enable Voltage 6mA Max Input Current ESD Protection, Thermal Shutdown Internal Clamp Diodes 0.6A~1.2A Output Current
ROHS 28BYJ-48 Step Motor	5V DC Input, 100Hz Frequency
ULN2003 Stepper Motor Driver Module	5V~12V DC Supply Voltage Separated Power Pins Internal Clamp Diodes 500mA Max Current Per Output
LCD1602 Module (With Pin Header)	16x2, Alphanumeric, LED Backlight 4.5V to 5.5V Operating Voltage 0.6mA Max Operating Current 25mA Max Current LED Backlight
Rotary Potentiometer	10kΩ

Circuit Design

In order to implement all the necessary parts for a functioning water cooler, various input sensors, button controls, motors, and output displays were connected to the Arduino Mega 2560 and an external 5V Power Supply Module. These can be seen in the schematic diagrams and system picture below.

Sensors/RTC

The first sensor implemented was a DHT11 Temperature and Humidity Sensor, connecting using the Arduino Kit Module. The sensor input was connected with Digital Pin D12 on the Arduino, and the sensor was connected to the 5V power supply and ground via the breadboard. The humidity sensor and the water level sensor discussed next were both connected to the Arduino Power Supply Module, The module is not pictured in the circuitry diagram below, however it would have been connected to the +/- connections on the leftmost breadboard (the actual setup is shown in the System Picture below).

The next sensor implemented was the Arduino Starter Kit's Water Level Detection Sensor Module. The water sensor was also connected to the breadboard for power and ground, and was connected to the Arduino via Analog Pin A0.

Finally, the DS1307 Real Time Clock Module was connected to the power supply and ground. The SDA/SCL pins were connected to the corresponding SDA/SCL pins (D20 and D21) on the Arduino controller

Button Controls

Overall, 5 button controls were implemented for various functions on the water cooler. All of the buttons were connected to the Arduino breadboard using a pullup configuration with 220Ω resistors, as demonstrated on the breadboard schematic below.

The stop, start, and reset buttons' GPIO was connected with the Arduino Controllers' Digital Pins D53, D10, and D11 respectively. The two stepper motor control buttons were connected to Digital Pins D8 and D9.

Output Motors

To provide basic functionality for the water cooler, two basic motors were implemented. The first was a simple 3-6V DC motor connected with a fan blade (not pictured in the breadboard schematic) in order to provide cooling. To manage power requirements from the DC motor and in order to potentially supply power to more powerful fans in more advanced designs, the fan motor was connected to the Arduino controller through a L293D push-pull driver and both types of motors were powered via the external power supply module.

For the DC fan motor, the push-pull driver was connected to the power supply via the Vs and Ground pins. As shown in the breadboard schematic, the DC fan motor power supply was connected to Output 1 (pin 3) of the driver and the breadboard ground supply. Input 1 (pin 2) of the driver was connected to the external power supply, while the Enable 1 (pin 1) was connected to the Arduino controller's digital pin D38.

Next, the Arduino Starter Kit Stepper Motor was implemented in order to control the direction of a vent (as demonstrated in the video). In the actual implementation of the design (see System Picture below), the stepper motor was connected to the Arduino Controller through the kit's ULN2003 Stepper Motor Driver Module, however that component was unavailable in the diagramming software used, so the main chip (ULN2003) was used in its place on the diagram. Then the four inputs (IN1 - IN4)on the Driver Module were connected to the odd numbered Arduino controller digital pins D33 - D39 according to the pin diagram shown below.

Output Displays

Two visual output displays were implemented for the cooler. The first consisted of four distinct colored LEDs (red, blue, yellow, and green) which visually represent the different program states, as discussed in the system operation section below. These LEDs were connected in series with 220Ω resistors and Digital Output pins D53, D51, D49, D47 on the Arduino controller according to the wiring and pin diagrams below.

The other visual output display (and final components) setup for the water cooler were the alphanumeric 16x2 LCD display module, with a 110Ω Resistor and a 10KΩ Potentiometer . First the LCD module's VCC and VSS pins were connected to a 5V power supply and ground, respectively. The R/W pin was also connected with ground, as the only functionality necessary for the water cooler is writing to the display. The RS and Enable pins were connected to the

Arduino controller's digital pins D7 and D6 as seen in the diagrams below. The LCD display module is converted from celsius to fahrenheit to make it more understandable.

Then the 110Ω resistor was used to connect the anode pin (pin A) of the LED Backlight to the power supply and the backlight cathode pin (pin K) to the breadboard's ground. The potentiometer was connected to a 5V power supply, ground, and to the input voltage (V0 pin) on the LCD module. Finally, the LCD module output pins D4 - D7 were connected to the Arduino's digital pins D5 - D2 respectively.

Program

The final program for the water cooler combined numerous different inputs, sensor monitoring, interrupts, motor controls, and visual displays. The various system controls were combined to provide four operational states: Disabled, Idle, Running, and Error. The RTC is also initiated and programmed to provide the Serial port with real-time status reports when state changes and changes in vent position occur.

Cooler States

Disabled

The disabled state sets the Water Cooler into a 'standby' functionality, such that there is still power supplied to the system but no sensor monitoring or controls are active. The only active components within the disabled state are the yellow LED—which is driven HIGH or 'on' to provide visual confirmation to the user of the change in state—and an ISR monitoring the GPIO input from the 'start' button. The water cooler's Disabled state can be triggered from any of the other states, simply through the user pressing the 'stop' button. The system will then stay in its disabled state until the start button ISR is triggered, which sends the system to the second possible state, Idle.

Idle

Idle is one of the two states in which the water cooler is 'functioning' (the other being the Running state). When the cooler switches into Idle state, the yellow-LED is turned off and the green-LED is turned on instead. Meanwhile, the program actively monitors the temperature and water level for conditions that may trigger state transitions. For example, if the temperature sensor reads that the environment has gotten warmer and passed the temperature threshold (in the program, this was set at 76°F), then the system would transition into its Running state. Alternatively, if the water level drops below a predefined threshold then the system would transition into Error mode.

Running

The Running state operates similarly to Idle, with only a few exceptions. When switching into Running state, the active LED is switched to be the blue LED to visually demonstrate the change. Similarly to Idle state, the temperature and water level are actively monitored for scenarios which might trigger a state change, and the Error state can be triggered at any point by the water level dropping too low. The most notable difference between the two states is that in the Running state, the fan motor is activated in order to cool off the external environment and lower the temperature back below the threshold.

Error

Error mode, which can be activated from either the Running or Idle states, provides a fail-safe which ensures the system has the resources (namely water/humidity/evaporation) to run efficiently and within the system operating constraints. When the system enters Error mode, the red-LED is turned on (any other LEDs are turned off when entering this state), and an error message “Water lvl low” is displayed on the LCD display. The system remains in the Error state until the water level is raised above the threshold. Unlike the Idle and Running states where water level is continuously monitored, in the Error state the Reset button is monitored and only once pressed does the system check if the water level has been fixed. The system will remain in Error state until the water level once again satisfies the system’s operating requirements.

Additional Features

In addition to implementing the four above mentioned states, some system-wide functionality was implemented for all states except Disabled. Namely, a timer interrupt was designed such that every 60 seconds, the current temperature and humidity readings are displayed on the screen. Additionally, the two vent control buttons were programmed to continuously monitor for button presses, which trigger vent position changes (using the stepper motor) throughout every state except Disabled.

Operating Constraints

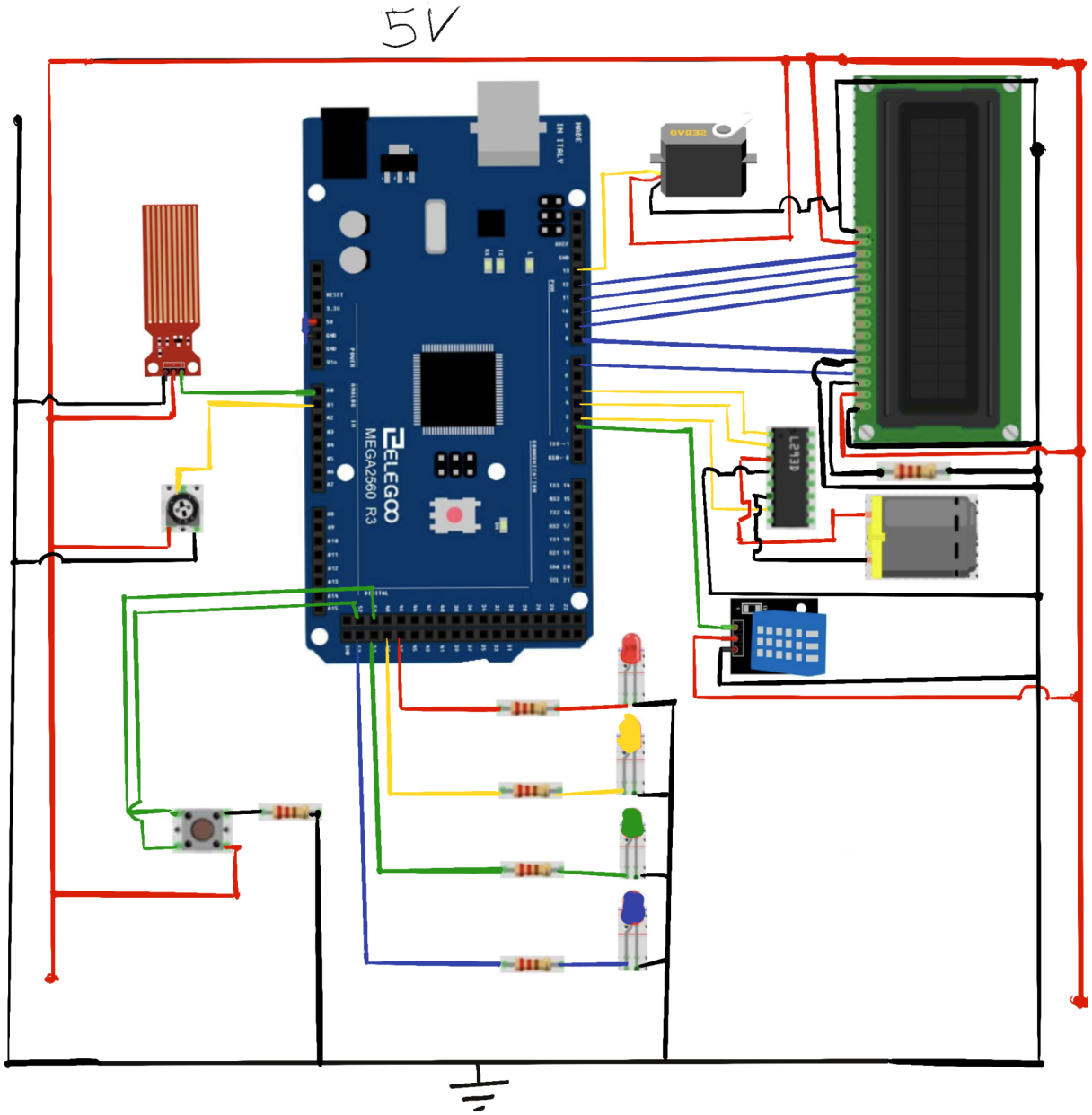
As only two motors are needed for the water cooler functionality, the breadboard power supply module and the Arduino power supply output provide sufficient voltage and current ratings so as to not exceed the sources' power ratings and limit the operation of the motors. However, there are other constraints which must be taken into consideration.

Water coolers involve an environment which can be temperamental for electronics through variances in temperature and humidity. For instance, the specifications of the DHT11 Temperature and Humidity Sensor dictate that the sensor collects its most accurate measurements within the temperature range 25°C ±2°C and the relative humidity range of ±5% RH. This could limit the operation of the water cooler, as it may have worse performance if the external environment has a large temperature range.

Additionally, the design of the water cooler increases the relevance of the water sensor temperature constraints limiting its working temperature range to 10°C to 30°C. As the water sensor remains in the water, if the temperature of the water changes drastically there could be negative impacts on the performance of the water sensor.

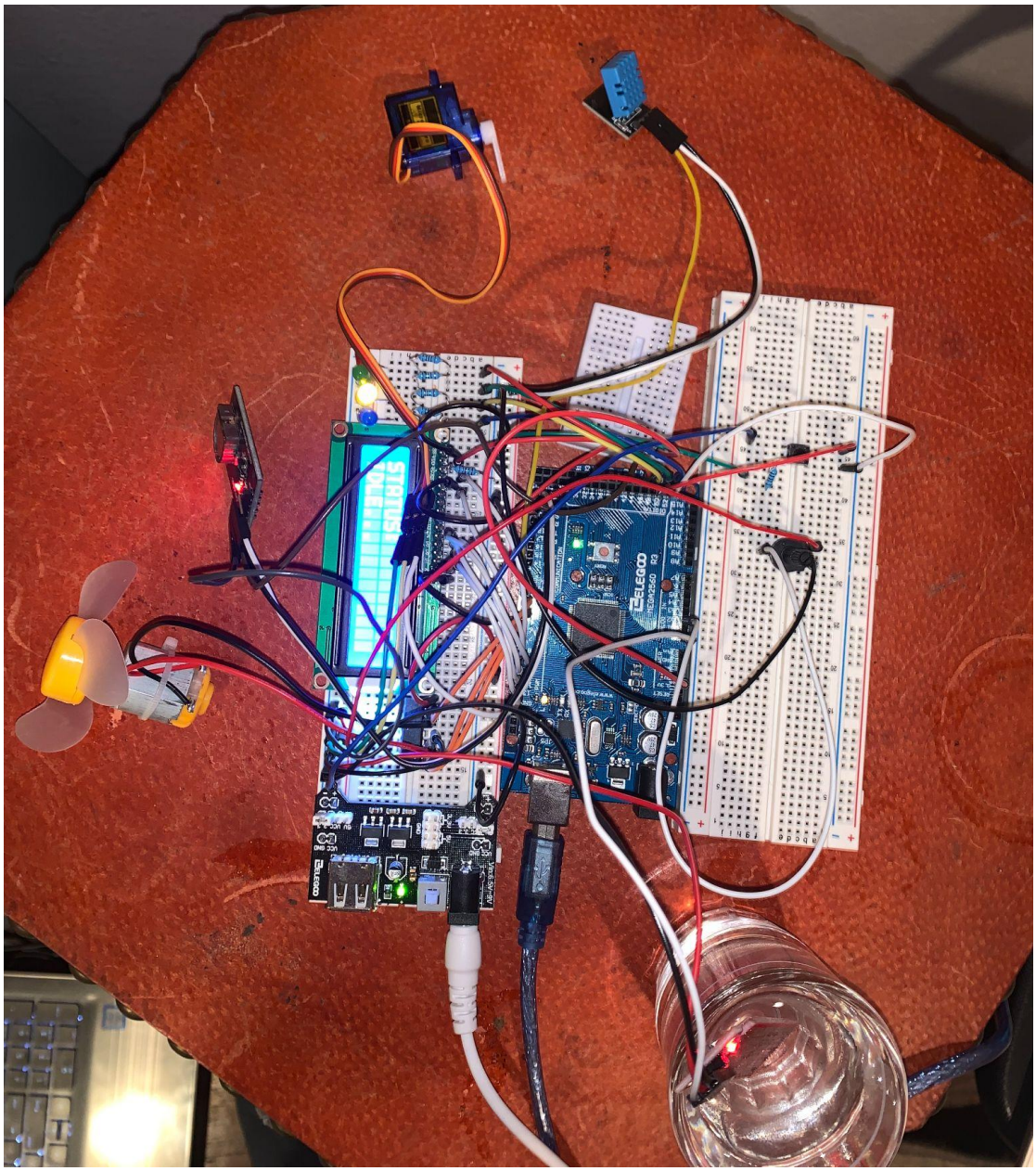
One other operating constraint that was observed during the process of designing and building the water cooler was the longevity of the water sensor. It was observed that when the water sensor remained in the water throughout the system operation, it started rusting fairly quickly. That in turn interfered with the water sensor's ability to accurately measure the water level. This issue could potentially be solved through differing design configurations for the water sensor to be able to detect a fall in water level.

Wiring and Pin Diagram



Final System

System Picture



Video of Operation

https://youtu.be/zdQoIhD-Fg0

Github Repository

https://github.com/Rickyrich2579/CPE-301-Final-Project.git

Relevant Specification Sheets

Arduino MEGA 2560
DS1307 RTC Module
ELEGOO MEGA 2560 Starter Kit
LCD 1602 Module
L293D Push-Pull Driver
ULN2003 Stepper Motor Driver