Group Name: As You Can See
Names: Timothy Ang, Randall Cheng, Darren Ly, Brian Wu
Due Date: 05/10/2024

Swamp Cooler Overview

Description:
Equipment List (and associated datasheets):

- Arduino ATMega2560 (link 1, link 2)
- Solderless breadboard, jumper cables
- 4 colored LEDs (yellow, green, red, blue), 330Ω resistors
- LCD1602 Display Module (link)
- DHT11 Temperature/Humidity Sensor (link)
- DS1307 RTC Module (link)
- Stepper Motor + ULN2003 Driver Module (link 1, link 2, link3)
- 3–6V motor + fan blade
- Water Level Detection Sensor Module (link)

Design Overview, Constraints:

This project's end product is an evaporation cooling system (AKA 'swamp cooler') that has four states (DISABLED, IDLE, ERROR, RUNNING) that are activated through user input or environmental conditions (temperature, humidity, and water level). Each state illuminates a corresponding colored LED (yellow, green, red, and blue, respectively) and displays a message on the LCD1602 display. Temperature/humidity readings are shown in IDLE and RUNNING, while ERROR also displays a "water level too low" error message.

To collect the relevant data to operate the fan motor, the DHT11 temperature/humidity sensor and the water level detection sensor module were used. As these peripherals output an analog data signal, analog-digital conversion is necessary for the Arduino to use those signals. Using these peripherals comes with certain limitations. The DHT11 sensor's operational temperature range is 0–50°C ± 2°C. The water level detection sensor's sensor area is only 40x16mm (640 mm^2) and operates within -30°C–50°C.

User input involves 4 DIP buttons: one for starting/stopping program execution, another for resetting it, and the last two adjust the stepper motor 'vent' left or right. When the 'vent''s position is adjusted, a timestamp is recorded to the serial monitor using the DS1307 RTC module.

These components work together to control the fan, and ultimately the cooler. Instead of using a separate power supply board to power the cooler, a simpler implementation comprising a MOSFET, resistor, and diode to safeguard the Arduino from the kit's motor was used.

final.ino Walkthrough:

As defined by the handout, we included the allowed **Stepper**, **LiquidCrystal**, **RTClib**, and **DHT** libraries to create the associated objects with specified digital GPIO pin #s in **setup()**. Named pointers to the registers of other peripherals (e.g. UART0, Timer1, ADC, relevant pin ports) were initialized to facilitate low–level operations (e.g., **U0init(int)**, **setup_timer_regs()**).

Within **setup()**, we establish serial communication at 9600 baud, configure the timer and ADC, assign values to DDR pointers of digital pin ports, and define parameters for peripheral objects (e.g., **lcd.begin(<dimensions>)**, **rtc.adjust(<DateTime>)**, **stepper.setSpeed(<int>)**).

In **loop()**, we would have the water level, temperature, and humidity be monitored for fluidity between states. **turnState()** will display the time the stepper changed position, then take 60 steps in the specified direction (left/right). When in DISABLED, **turnState()** can be executed when the left/right button is pressed. The yellow LED will be turned on and the fan will be off. It will go through a system of checks to display times or to change to RUNNING or DISABLED.

IDLE will allow for **turnState()**, and the green LED will be turned on. The LCD will show the temperature/humidity. It will check to see if it will go to ERROR, which happens if **waterLevel** is less than the threshold. The system state will become DISABLED if the on/off button is pressed. The system will enter the RUNNING state if the temperature is above 19°C. If it does not go through the previous checks, it will stay IDLE.

Upon entering the ERROR state, the fan is deactivated, time is displayed, and the LCD indicates "Water level is too low" with the red LED on. Checks are conducted for reset button input and water level above the threshold. Transition to DISABLED state occurs upon pressing the on/off button.

In the RUNNING state, **turnState()** is enabled, time is displayed, and the blue LED is lit, showing temperature and humidity on the LCD. The fan is activated if the temperature surpasses the threshold. Transition to the ERROR state occurs if the water level is below the threshold. Pressing the on/off button transitions to the DISABLED state.

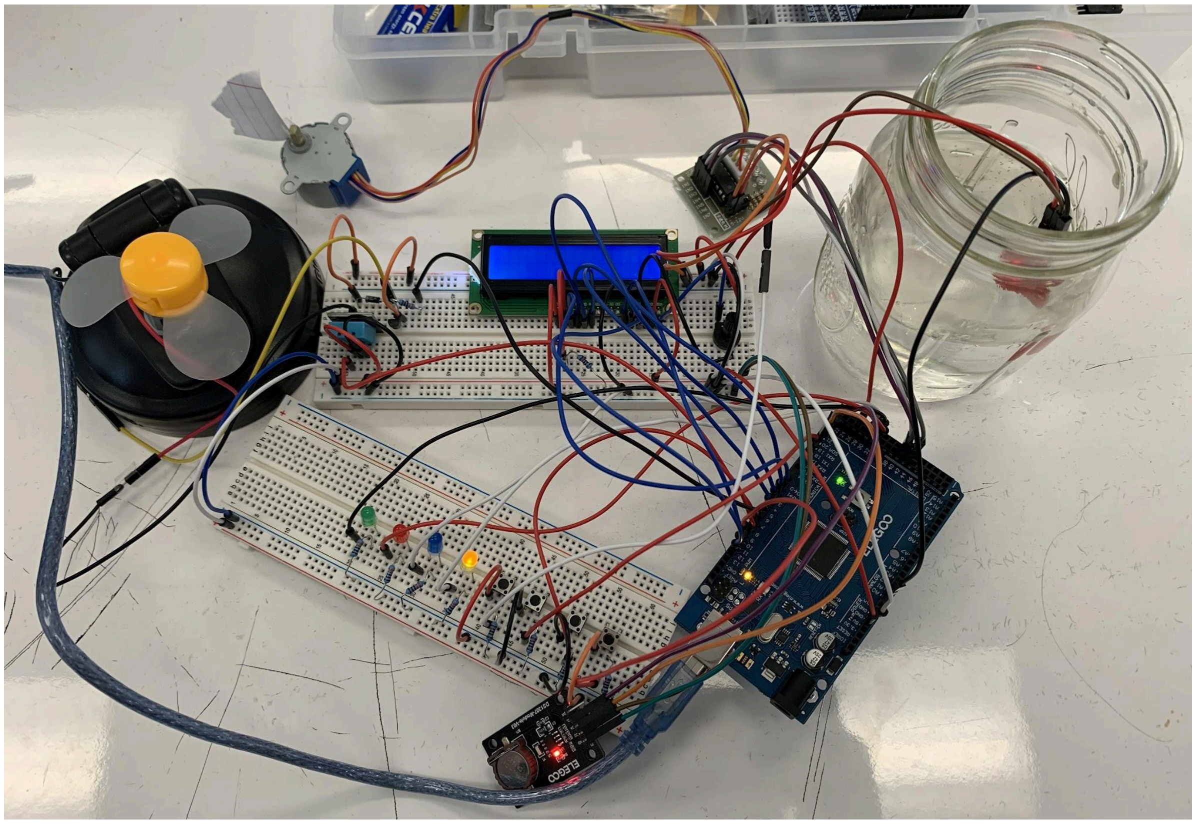
As we are not allowed to utilize the **Serial** class' methods, alternate functions were written to replace **begin(int)**, **available()**, **read()**, and **write(char)**; they are **U0init**, **kbhit**, **getChar**, and **U0putchar**. Similarly, built-in analog-digital conversion functions are not permitted; **adc_init** and **adc_read** were written to serve this purpose. **ToggleSystem** occurs during **ISR**. **DisplayTime** shows the date and the format of the date.

As **delay(int)** is unavailable, **millis()** is employed instead. Whereas **delay(int)** halts execution and idles the ATmega2560 for a specified duration, our **millis()** equivalent utilizes a global **time_now** variable to maintain activity during the designated delay period.

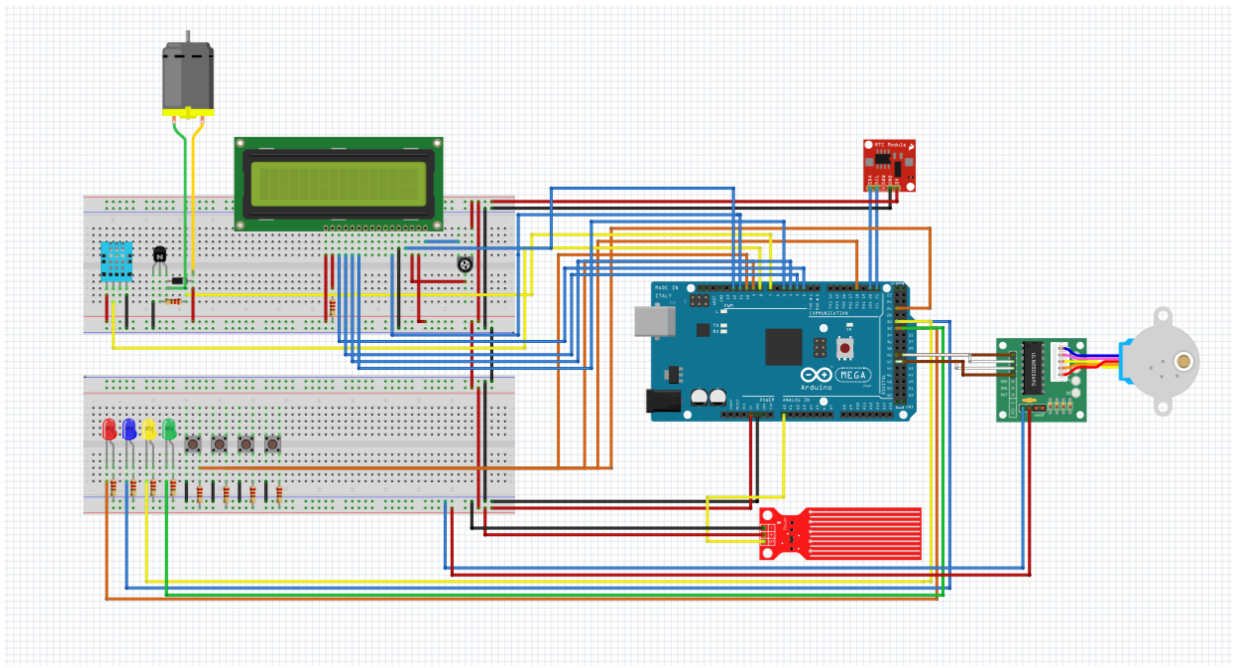
GitHub Project Repository:
https://github.com/planetbrian-unr/CPE301Final.git

Media:

Circuit Image:



Fritzing Schematic:



Videos:

Google Drive Folder, video descriptions in GitHub repository's *media* subdirectory