ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

Scrubbing CO2 Operational Prototype

Team #32

SEUNGHWAN HONG
(sh34@illinois.edu)
ALAN CARDIEL
(acard6@illinois.edu)
KINJAL DEY (kinjald2@illinois.edu)

Prof: Arne Fliflet
TA: David Null

Dec 1, 2023

Abstract

In order to fight water insecurity around the world and global warming, Professor Jont Allen of the ECE Department has proposed a system to desalinate seawater with minimal power.

In order to test his hypothesis, he has built a prototype and enlisted out help to make his prototype more user-friendly, and to display any data gathered in a manner that is easy to understand.

Our project can be split into three major parts: the hardware, firmware and the software. With some fixes to our PCB design and implementing some changes to help the firmware and software communicate via serial USB, our project should be able to completely control his prototype system.

The hardware design issues prevented us from being able to collect temperature data from all but two of the nine temperature sensors. However, it is able to accurately collect humidity and level sensor data. It is also able to automatically control the safety features of the prototype when needed.

The software design allows the system to graph all data in real-time, however due to communication issues with the ESP32 microcontroller, it is unable to receive or send signals through serial USB.

Contents

1 Introduction

1.1 Problem Overview

The Sahara desert is the only region in the world where water insecurity amongst the local communities is steadily increasing. The region has also grown 10% in size since the 1920s and on wards, due to global warming [1]. Access to clean drinking water is a basic human right, and this is a problem that must address as soon as possible.

Professor Jont Allen from UIUC proposes a solar-powered desalination system, pumping ocean saltwater into a desert tank. Solar heat evaporates water, separating salt, and the vapor condenses in a colder tank for clean water [2]. The prototype, with over 50% yield, awaits modifications. However, separate components and a lack of user-friendly controls pose efficiency risks in future adjustments to the system.

Professor Allen faces challenges with his prototype: separated components cause setup issues, and some require manual adjustments, impacting experiment reliability. Streamlining and storing experiment data lack an efficient method. Lastly, improving system efficiency is hindered by a lack of easy variable control, making enhancements more challenging.

1.2 Solution

We will work along side Professor Jont Allen in his research to develop and improve upon current issues in his prototype. Helping him be able to run experiments more easily and concise along with data collection will allow him to further drive his research for potential upscale. This includes creating a control system to access and control the lab setup more easily for use in testing and experimenting. We will also be measuring the status of the system with the help of sensors to be able to understand, predict, and act on that status to ensure safety and efficiency in the system.

To address the current issues in his current prototype of organization we plan to consolidate and house all the components into a single place. This will allow for a more centralized way to interact with each component, from sensors to heater, and radiators. As for data collection and testing we plan to set up an algorithm that will take time and a flow rate as inputs to be able to autonomously run the lab setup with redundancies for faults, failure, safety, and termination so that each individual component does not need to be set or initialized by the user. As for the visualization we will process and collect the data and make it readable, presentable, and usable for the user to digest and understand.

1.3 Project Overview

Our solution can be broken down into three high level requirements that allow us to quantify the success of our project. Those are to consolidate and house all the components into one cooperative system, automate the system from read data to control power devices, and lastly store and present the data graphically for user readability. The three high level

requirements split off into three main subsystems which entails of reading data from sensors, controlling the power of the system, and the user interface for starting experiments and collecting and presenting data.

Reading data from the sensors is our sensing subsystem which reads the temperature, humidity and, level sensors. The sensing subsystem allows us to observe the experiment while its running and detects any safety concerns for us. The sensing subsystem then interacts with the power and control subsystems, where the power subsystem regulates power to the different components to ensure power operation of 3.3v, 5v, 12v, or 120v and ensures stability in power flow. As for the control subsystem it controls the voltage to the higher power devices so to not harm other subsystems for operational or safety reasons. The control and power subsystems allow us to control when power supply turns on and off to ensure that the system operates safely and efficiently. The three subsystems then interact with our ESP32 microcontroller which interacts with the UI via USB to allows users to input arguments and automate, read, and control the system. This then gives the user a graph of the data that is understandable.1

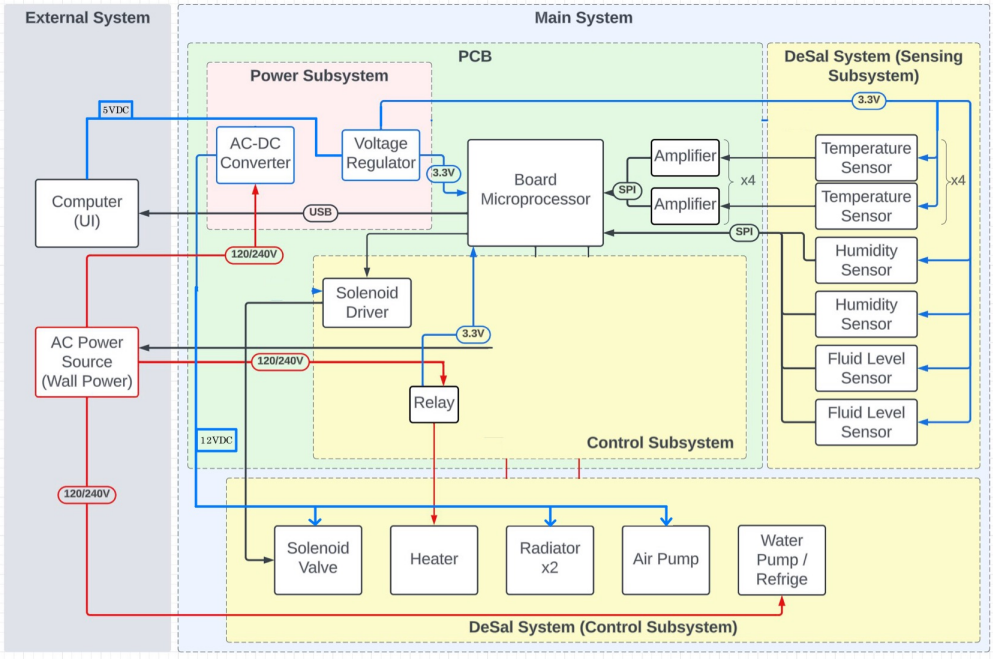

Figure 1: Block Diagram

2 Design

Table 1 outlines the values associated with all components in our control and sensing subsystem. The configuration of Sensing Board for collecting temperature and humidity data is given in Figure2 and the configuration of Control Board for controlling various components is given in Figure3.

Table 1: Desalination System: Component Values

Component	Voltage Input	Rated Current	Connection
Temperature Sensors [3]	3.3VDC	0.9mA	SPI
Humidity Sensors [4]	3.3VDC	1.5mA	MSB
Microprocessor [5]	3.3VDC	500mA	USB-B
Heater	120VAC	4A	AC wall power + Relay
Solenoid Driver	12VDC	540mA	AC/DC Adapter
Cooling Fan	12VDC	300mA	AC/DC Adapter
Air Pump	12VDC	300mA	AC/DC Adapter

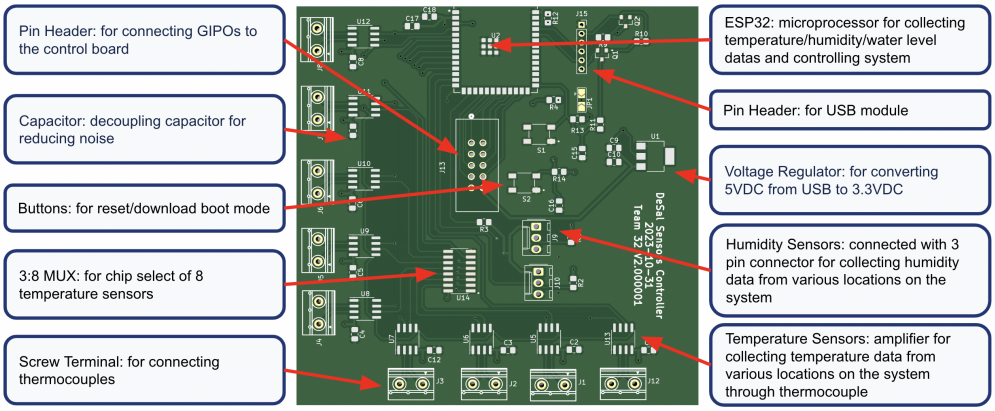

Figure 2: Low Power/Sensing Board

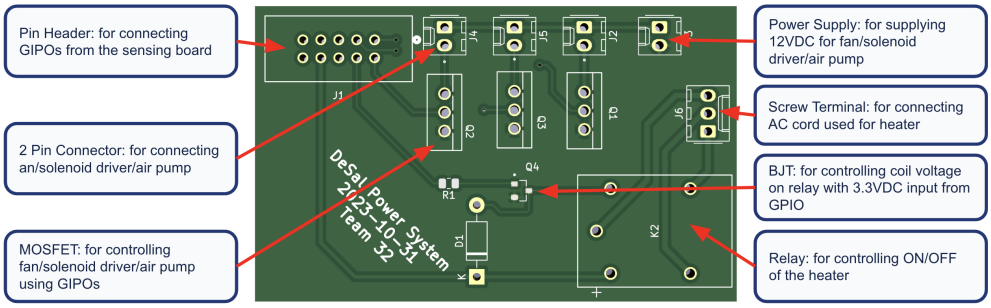

Figure 3: High Power/Control Board

2.1 Power Subsystem

2.1.1 Voltage Regulator

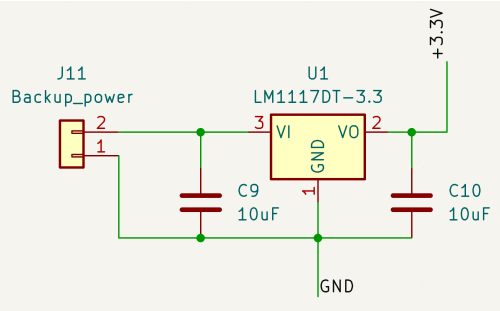

Figure 4: LM1117DT-3.3 Voltage Regulator

LM1117IMPX-3.3/NOPBTR-ND voltage regulator in Figure4 was chosen because it consistently provides a fixed output of 3.3VDC when the input voltage is 5VDC, which perfectly aligns with the power supply sourced from the USB port's VBUS. The 3.3VDC output from this regulator is ideal for providing power to 9 temperature sensors, 2 humidity sensors, and the microprocessor. Additionally, a crucial consideration was its capability to deliver a current output of 800mA[6], which perfectly suited the needs of the temperature sensors drawing 900uA, the humidity sensors drawing 1.5mA, and the microprocessor drawing 500mA.

2.2 Sensing Subsystem

2.2.1 Temperature Sensors

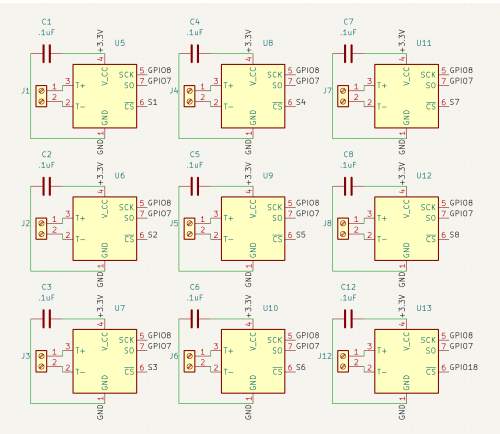

Figure 5: MAX31855JASA+T

MAX31855KASA+T temperature sensor in Figure5 was selected for several reasons. First, it offered a significantly more cost-effective solution compared to the MAX6659 used

by Professor Allen. Secondly, its voltage supply range falls within 3.0VDC to 3.6VDC, perfectly matching the output voltage of our voltage regulator and other components, eliminating the need for an additional voltage regulator to accommodate varying voltage supplies. Moreover, its temperature measurement capabilities span a wide range from +1800°C to -270°C, which more than adequately covers the requirements of our experiment. Additionally, the sensor provides temperature data in a signed 14-bit format, which is compatible with SPI and read-only. This format can be easily interpreted by our microprocessor [3].

2.2.2 3:8 MUX

SN74HC138DR in Figure6was selected as the logic chip for selecting temperature sensors because it offers a broad operating voltage range, spanning from 2V to 6V. This flexibility allows us to use the same voltage supply provided by our voltage regulator [7]. A decoder MUX was also used to allow us to use fewer pins on our board to access the many temperature sensors.

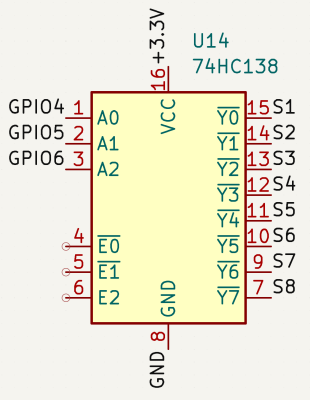

Figure 6: SN74HC138DR

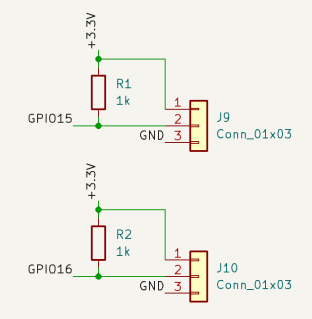

Figure 7: Connection for DHT22

2.2.3 Humidity Sensors

It was not necessary to pick humidity sensor on our own because Professor Allen already provided one. Therefore, we chose to connect it using an available 01x03 connector in the ECE 445 lab. The circuit diagram for humidity sensor connection is in Figure7The value for resistor was from the provided diagram on the data-sheet [4].

2.3 Control Subsystem

2.3.1 Microprocessor

Initially, our decision for selecting the ESP32 in Figure8 was driven by its Wi-Fi capabilities. However, as we progressed, we continued to use it primarily because it operated on the same 3.3VDC voltage supply [5]. Regarding the strapping pin, we devised a method

to connect them to a voltage source through a resistor. This allowed us to toggle between high and low states by attaching and detaching the resistor.

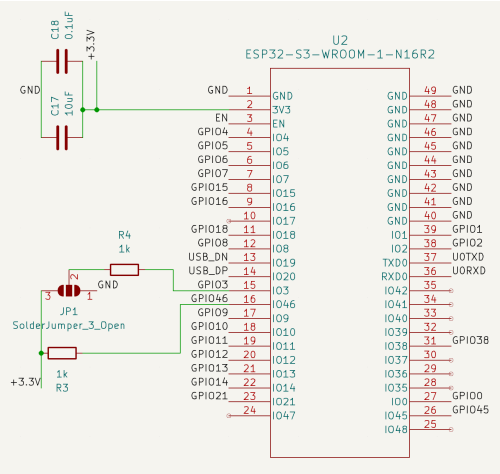

Figure 8: ESP32-S3-WROOM-1-N16

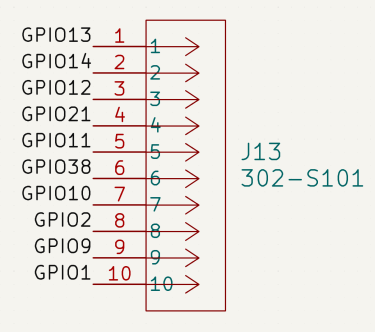

Figure 9: Pin header for GPIOs

2.3.2 Pin header

We selected a 10-pin header in Figure9 to ensure there were enough GPIOs available for communication with the high power board responsible for the control subsystem.

2.3.3 USB Port

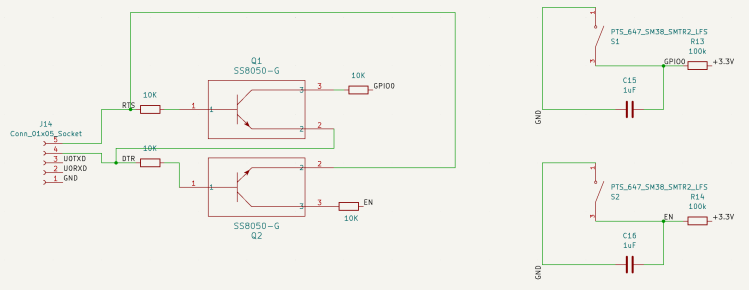

Figure 10: Connection for USB bridge/port

The circuit for USB Port in Figure10 was already provided at ECE 445 website. For the BJT, we selected the SS8050-G, and for the button, we selected the PTS 647 SM38 SMTR2 LFS. Our choices were primarily based on their appropriate size and the fact that they had readily available KiCad symbols and footprints on SnapEDA.

2.3.4 Relay/BJT

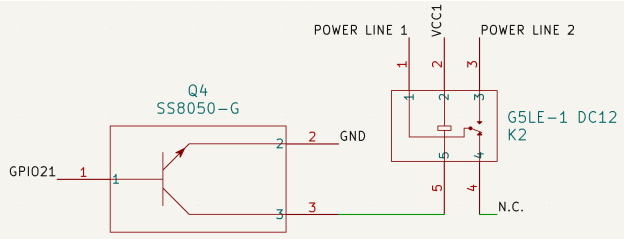

Figure 11: Relay and BJT

We selected the G5LE relay in Figure11 due to its rated coil voltage of 5VDC [8]. This implies that the must-turn-on voltage, which is 75% of the rated coil voltage, is 3.75VDC. This voltage is sufficient for control using a GPIO that outputs 3.3VDC, facilitated by a BJT and a 5VDC external voltage supply from the USB. Specifically, we chose the SS8050-G BJT to govern the relay with a 3.3VDC GPIO. In this setup, the emitter is grounded, the base is linked to the GPIO, and the collector is connected to the relay, which is powered by a 5VDC source. When the GPIO at the base is in a high state at 3.3VDC, current will flow between the collector and emitter, delivering 5VDC to the relay, meeting the must-turn-on voltage requirement of 3.75VDC.

2.3.5 MOSFETs/Power Supply

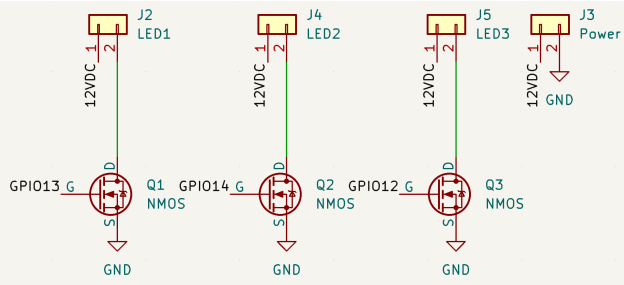

Figure 12: MOSFETs and 12VDC Power Supply

AC/DC converter from the power subsystem has been substituted with a 12VDC adapter. This change was necessary to convert the 120VAC wall power into 12VDC, specifically for the fan, solenoid driver, and air pump. This modification was made for ease of implementation and to eliminate unnecessary components on the PCB. We selected the IRF540 MOSFETs in Figure12 for controlling the ON/OFF functions of our fan, solenoid driver, and air pump. The decision was based on the fact that these MOSFETs have a minimum Gate-source threshold voltage of 2.0VDC [9], which is sufficient for allowing current to flow between the drain and source when the GPIO is set high at 3.3VDC. Furthermore, despite the fact that our fan, solenoid driver, and air pump operate at a common voltage of 12VDC, it was essential to fulfill their distinct current specifications. The fan requires a

current of 300mA, the solenoid driver requires 540mA, and the air pump requires 300mA. Ensuring a minimum drain current of 540mA was critical if we intended to control all of them using a single MOSFET. According to a typical characteristic provided in data-sheet at Figure13,

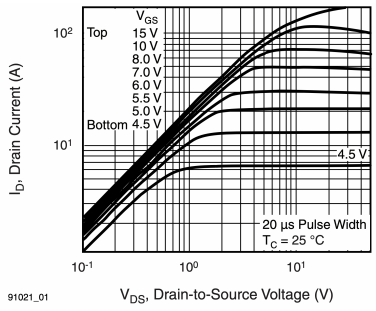

Figure 13: IRF540 Typical Characteristics

the IRF540 is expected to exhibit a drain current exceeding 1A when the gate-source voltage is 3.3VDC, little below 4.5VDC and the drain-source voltage is 12VDC, which meets the requirements for the all components when connected to a 12VDC power supply from the adapter through 2 pin connector. Finally, it was essential to evaluate whether a heat sink was required for our MOSFET by measuring the increase in temperature when power was applied to the MOSFET. The maximum power transistor can dissipate compare to actual power transistor can dissipate can be modeled by the following equation :

$$P_L = R_{DS,on} \times I_L^2 \leq \ P_D = \frac{T_{max} - T_{amb}}{R_{th,JA}}$$

[10] where:

P_L = Actual power transistor dissipate [W]

$R_{DS,on}$ = Drain-source on-state resistance [Ω]

I_L = Whatever our load draws [A]

P_D = Maximum power transistor can dissipate [W]

T_{max} = Maximum junction temperature [°C]

T_{amb} = Ambient temperature [°C]

$R_{th,JA}$ = Maximum junction-to-ambient [°C/W]

For the MOSFET, T_{max} is 175°C, T_{amb} is 25°C, and $R_{th,JA}$ is 62°C/W [9].

$$P_D = \frac{175 - 25}{62}$$

Resulting P_D of 2.4W. For our system, the maximum $R_{DS,on}$ is 0.077Ω and the maximum I_L is 540mA from the solenoid driver which makes P_L clearly below P_D indicating that the system is safe to operate without a heat sink.

2.4 External System

2.4.1 Software

We decided to use a Flask app as it is straight-forward and easy to implement. Python also has a lot of libraries to select from and the code would be easier to write than in other lower-level languages.

3 Design Verification

3.1 Power Subsystem

3.1.1 Voltage Regulator

The requirement and verification for voltage regulator is given in Table 4. For the voltage regulator, it was essential to accurately generate a 3.3VDC output from the 5VDC input provided by the USB, as it supplies voltage to all components on the low-power/sensing board. When the voltage was examined using a voltmeter, the reading was 3.42VDC, falling within the acceptable range of 3.3VDC with a tolerance of $\pm$5%. Although we couldn't identify a temporary load equivalent to the circuit's load for current testing, connecting all sensors and the microprocessor to the voltage regulator demonstrated that they operated without any issues, indicating sufficient current output.

3.2 Sensing Subsystem

3.2.1 Temperature Sensor

The requirement and verification for temperature sensor is given in Table 5. In our system we had to deal with two versions of the temperature sensors, those which had their chip select pin (CS) connected to the MUX and the single sensor connected directly to our microcontroller. After connecting the sensors to our board and ensuring that they were properly set and powered we began test reading with the lone directly connected sensor. Adafruit provides a library for being able to read their sensors which deals with SPI communication between the sensor and microcontroller allowing for us to simply call on that function by using the cs pin, sck pin, and data pin with the main focus being control of the cs pin. Since the ESP32 has direct access to the cs pin on the single sensor reading from this sensor caused no issue.

As for the sensors connected to the MUX, one sensors was initially tested and worked by simply setting the cs pin in the function to an unconnected pin and manually enabling the MUX and setting the desired pin to output low for the CS. This initial testing allowed us to read one sensor that was connected to the MUX, however after attempting to establish a connection with the remaining sensors cause issues. These issues relate to the timing differences between the MUX and the sensor as well as clobbering in the data bus once the other sensors were connected. This then impacted every temperature sensor and prevented us from properly reading values. In order to fix this issue would require fixing the

MUX since it is the root of the issue.

3.2.2 3:8 MUX

The requirement and verification for 3:8 MUX is given in Table 6. In our initial PCB design we had mistakenly forgotten to connect the enable pins of the MUX to our board, this would prevent us to properly read the temperature sensors since they could never be set low. This mistake was fixed in our second PCB design and allowed us to have proper control. Once the MUX was connected onto our PCB we had first powered our board and connected to an oscilloscope to ensure that the MUX was properly connected and powered. After the chip was connected and powered we simply tested it by setting its enable pins to allow us to read from the select pins and then use an oscilloscope again and read if the pins were outputting the correct voltage. For this the enable pins had to be set to either 0V or 3.3V and the select pins would be 0V or 3.3V depending on the desired output at which point the selected output would read 0V while the remaining outputs read 3.3V.

Once the MUX was set up we further tested it by trying to read from one of the sensors connected to it, which was reading correctly to our tolerance for the temperature sensor. However after connecting the remaining temperature sensors we ran into the issue of not being able to read any of the temperature sensors and the data line having a zero value. We had discovered that this was a timing issue between the output of the MUX's selected pin once enabled and the expected timing for the MAX31855 chip after the CS is low an when the clocking begins. This was known since the data bus for the sensors would only read a value after the lone sensors directly connected to the ESP32 was read and data was still on the line. This issues all stems from the fact that the temperature sensors function requires direct access to the cs pin which causes our timing issue. We believe that a fix to this issue would be rather than have the CS pins on the MUX, connected them directly to the ESP32 and use a 8:1 MUX for the data bus to avoid the clobbering issue.

3.2.3 Humidity Sensor

The requirement and verification for humidity sensor is given in Table 7. Before testing on any physical system started, testing began using an online emulator of our ESP32 board and a DHT22 humidity sensor[11]. This emulator allowed us to test how to read the humidity sensor and the necessary set up so that once we moved to a physical prototype the written code would translate without issue. The initial physical testing was accomplished via an ESP32 dev board where humidity sensor was tested by moving out of a hot room and into the cold of the night. The readings of this testing lied within our required tolerance and thus allowed us to move to testing on the final product. Once the sensor was connected to our board it had worked as intended like previous test and present the humidity data fro us to read and process.

3.2.4 Level Sensor

The requirement and verification for level sensor is given in Table 8. By connecting the level sensors through a pin on our board we can read signals being passed. The level sensor is a simple magnetic switch so by continuously feeding it a value we know if the switch has been turned off if no signal is read.

3.2.5 Microprocessor

The requirement and verification for microprocessor is given in Table 9. Before testing our ESP32 microcontroller we had test code that worked on a developer board that set the values of pins and reads humidity sensor. After ensuring that the pre-established code worked we translated it over to our board. After setting the strapping pins to be floating or hard set and enable having a button to ground we could upload code to our microcontroller via USB from a host computer. Once a connection was established and we could flash programs to our board we started with the aforementioned reading of the humidity sensor, thus we were able to read data from the sensor. This later expanded to at least be able to read 2 temperature sensors as well, one connected to the MUX and the other directly connected to the ESP32. After this point our software was set up to be able to automatically control the power flow in our daughter board containing the 12V and 120V power components. Lastly we weren't able to connect our board to our UI and read its inputs and send data for visualization which is expanded on in the external system section. As for the automation aspect since we did not have enough time to connect our board with the prototype we could not test out entirely if the project could truly run autonomously and independent.

3.3 Control Subsystem

3.3.1 MOSFET

The requirement and verification for MOSFET is given in Table 10. The footprint chosen for the MOSFET in our custom PCB order had a different pin configuration compared to the actual MOSFET received. The footprint had the Drain on the first pin and the Gate on the second pin, whereas the MOSFET we obtained had the Gate on the first pin and the Drain on the second pin. To address this mismatch, it was necessary to manually bend the pin using pliers to align with the correct configuration, ensuring that the Drain and Gate pins did not touch each other.

Following the pin adjustment, three MOSFETs and two-pin connectors for the fan, solenoid driver, and air pump were soldered onto the control board, and a verification test was conducted. For the solenoid driver and air pump, which couldn't be connected to the actual components, two fans provided by Prof. Allen were used as replacements. Since all components operate at a common voltage of 12VDC, as long as the current flow meets the minimum requirement of 540mA, they should function similarly when connected to the real components.

During the test, the three fans alternated between ON and OFF states every 5 seconds.

The assessment for the minimum current requirement was omitted as it was evident from the data-sheet that our MOSFET has the capacity to output a current exceeding 1A. This verification process guarantees the consistent operation of our components when the gate is connected to our microprocessor that outputs 3.3VDC.

3.3.2 Relay

The requirement and verification for relay is given in Table 11. The relay was required to have a must-turn-on voltage below 3.3VDC (a voltage that would definitely turn on the relay) in order for our microprocessor that outputs 3.3VDC to be able to control ON/OFF of the relay. When the test was conducted, the relay successfully toggled ON/OFF every 1 second. Additional requirements for the relay included its high-voltage section being suitable for 120VAC and capable of delivering 5A. Ultimately, our goal was to have the microprocessor manage the activation and deactivation of the heater by toggling the relay ON and OFF through GPIO input. This requirement was for our heater that operates at 120VAC and required minimum current of 4A. In the testing setup, when the relay is switched on, there should be a current flowing through pin 1 and pin 3, supplying a voltage of 120VAC. To test this configuration, we used the other end of our AC cord, a monitor power cable. This end was connected to our existing 12VDC adapter. The adapter, a 12V 5A 60W Power Supply Adapter designed for charging computers, demonstrated its capability to use the 120VAC from the relay to output 12VDC and a current of 5A when successfully charging the computer.

3.3.3 BJT

The requirement and verification for BJT is given in Table 12. Initially, when designing the circuit for the G5LE-1A4 Relay, we believed the rated coil voltage to be 5VDC based on the specifications in the data-sheet [8]. This implied a must-turn-on voltage of 3.75VDC (75% of the rated coil voltage), which is above the 3.3VDC output capability of our microprocessor, the ESP32. Consequently, we decided to incorporate a BJT for controlling an external 5VDC power supply with the 3.3VDC output from the ESP32. This was possible by connecting the base to the ESP32's GPIO, the emitter to ground, and the collector to pin 5 of the relay along with connecting pin 2 to the 5VDC power supply.

In this common emitter configuration, BJT will be at saturation mode when the base is set to a high state of 3.3VDC, a forward biased, allowing the collector current to flow through the relay supplying 5VDC. However, during soldering, a need arose to switch the emitter and collector sides due to a mismatch between our footprint and the actual component. Since our BJT had a surface-mount design instead of a through-hole design, a simple solution like bending the pin was not possible. Consequently, we attempted to address this by using insulation tape to cover the emitter and collector sides on the PCB. We then rewired it to the correct configuration by soldering wires directly onto the BJT.

Following all the adjustments, the test outcomes indicated that our BJT was unable to effectively control the relay using the 3.3VDC output from the ESP32. This failure could

be attributed to the insulation tape melting during the soldering of the wire, leading to a short circuit where unintended electrical connections between components.

Later on, we found out that the relay we obtained was actually a G5LE-1A4 DC3 with a rated coil voltage of 3VDC. This allowed us to control the relay directly using the 3.3VDC output from the with 3.3VDC output from the ESP32. A circuit design enhancement could be done by eliminating the BJT and direcltly connecting the GPIO to pin 5 of the relay and pin 2 to ground. This modification would eliminate unnecessary components on the PCB, making it simpler and easier to operate.

3.4 External System

3.4.1 Software

The requirement and verification for software is given in Table 13. The frontend was designed using a Flask with jinja2 templating. To connect to the the micro-processor through USB we used pyserial and to implement live-graphing, we used the FuncAnimation function from the Animation class in the Matplotlib library.

We successfully managed to implement the real-time plotting using FuncAnimation. Initially we ran into a threading issue, however using FuncAnimation relieved that issue. Unfortunately, the frontend failed to communicate successfully with the micro-processor. We believe this was because the software and the firmware were improperly handling the reading and writing of the necessary signals.

4 Cost

4.1 Parts

[1]

[1]Resistors and capacitors not mentioned as they were given to us for free by the ECE department

Table 2: Parts list

Description	Manufacturer	Quantity	Price	Link
ESP32-S3-WROOM-1-N16	Espressif Systems	1	$3.48	link
amplifier Adafruit MAX31855	adafruit	10	$7.579	link
LM1117IMPX-3.3/NOPB	Texas Instrument	1	$1.14	link
K-type thermocouple	Smartsails	1	$9.99	link
PTS 647 SM38 SMTR2 LFS	C&K	2	$0.20	link
SN74HC138DR	Texas Instrument	1	$0.42	link
5100H1FL	VCC	3	$7.02	link
SS80580-G	Comchip Tech.	3	$0.29	link
G5LE-1A4 DC3	Omron Electronics	1	$1.64	link
PRT-12796	SparkFun Electronics	2	$2.10	link
302-S101	On Shore Tech. inc.	2	$0.33	link
PPTC051LFBN-RC	Sullins Con. Sol.	1	$0.48	link
Waterproof box	Diivoo	1	$26.99	link
IRF540NPBF	Vishay Siliconix	4	$0.91	link

4.2 Labor

The average ECE graduate from UIUC makes about $51 per hour. working at 40hrs/week for 3 team members working for 9 weeks, $\$51/hr \times 40hrs/week \times 3 \times 9weeks = \$55,080$ in labor costs

4.3 Schedule

Table 3: Schedule for Project Progression

Week	Task	Person
Oct. 2nd - Oct. 9th	Identify the electronic components required for our project	Everyone
	Build circuit schematic for the sensing PCB	Everyone
	Design the sensing PCB	Everyone
Oct. 9th - Oct. 16th	Revise circuit schematic	Everyone
	Revise PCB design	Everyone
Oct. 16th - Oct. 23rd	Order the sensing PCB	Alan
	Order electronic components for the sensing PCB	Seunghwan
	Build circuit schematic for the control PCB	Everyone
	Design control PCB	Everyone
	Revise circuit schematic for the sensing PCB	Seunghwan & Alan
	Revise the sensing PCB design	Seunghwan & Alan
Oct. 23rd - Oct. 30th	Order the control PCB and the revised sensing PCB	Alan
	Order electronic components for the control PCB	Seunghwan
	Requirement & verification for the sensing PCB on breadboard	Seunghwan
	Program Microprocessor with Dev Board	Alan
Oct. 30th - Nov. 6th	Solder components on the sensing PCB	Alan
	Solder components on the control PCB	Seunghwan
	Requirement & verification on the sensing PCB	Alan
	Requirement & verification on the control PCB	Seunghwan
	Develop Software	Kinjal
Nov. 6th - Nov. 13rd	Continue requirement & verification on the sensing PCB	Alan
	Continue requirement & verification on the control PCB	Seunghwan
	Continue developing Software	Kinjal
	Prepare for Mock Demo	Everyone
Nov. 13rd - Nov. 20th	Mock Demo	Everyone
	Bug fixing	Everyone
Nov. 27th - Dec. 4th	Final Demo	Everyone
	Mock Presentation	Everyone
Dec. 4th - Dec. 7th	Final Presentation	Everyone
	Final Paper	Everyone

5 Conclusion

5.1 Accomplishments

Our project succeeded in reading the humidity data, the binary level sensor, and two of the nine temperature sensors. It also succeeded in controlling the fan, and the solenoid driver.

The micro-controller successfully parses the data from the sensors and decodes it in way

that is easy to interpret. It is also able to send the appropriate signals to the fan and the solenoid driver to turn them on and off.

Within the software component, real-time graphing was successfully implemented. The graph is able to update automatically with new values every five seconds.

5.2 Uncertainties

Unfortunately, we were unable to accomplish all the goals we set out initially. On the hardware side, the BJT and the MUX was does not work as intended. On the software end, the frontend is incapable of communicating with the micro-processor.

Our PCB footprint for the BJT and the actual BJT we used are not compatible. Since the BJT was a surface-mount design, there was no simple solution to this issue, and we attempted to address this by using insulation tape. Unfortunately, this short-circuited the BJT module and we were unable to fix the design in the given time.

While we were designing our PCB, we failed to connect the EN pin for the MUX. This resulted in the micro-processor not having any access to the data the thermocouplers are reading. This means that eight of the nine thermocouplers could not be read.

Another issue with the MUX was the timing issue between the MUX'x selected output time and the temperature sensors read timing after its CS pin were set low. This issue caused clobbering in our data bus for the temperature sensors as well as not being able to read seven of the nine the sensors. We were only able to read one sensor connected to the mux by hard setting the MUX's values and the lone sensor directly connected to the ESP32

The frontend software was unable to communicate with the micro-controller as it was not able to read any data sent through serial USB. In addition the micro-controller was unable to read the data the software was attempting to write to it. We think the issue with this lies in how the software connects to the micro-controller.

5.3 Ethical considerations

Our project involves electronics in close proximity with water, so there are a few measures to take to ensure that no component that should not get wet comes in contact with water. The testing will take place in a room in the ECEB that Professor Allen has stored his prototype. In order to ensure that no water comes into contact any of the electronics, we will only connect the PCB to the prototype when the lid of the compartments are closed. In addition, the PCB will also be stored in a closed compartment away from the water enclosures to prevent any possibility of contact if there is spillage.

The heater used to raise the temperature of incoming water to 40 °C within the evaporation chamber has the potential to result in system damage by overheating if not properly controlled. To address this concern, we are implementing a control system where the heater will automatically switch off without any manual interference once the lower water level is triggered.

On the other hand, the full scale solution can potentially have a few ethics and safety issues. One such potential issue would be siphoning too much salt water from one specific place, thus disrupting the environmental balance of that ecosystem. This would be in direct violation of ACM 1.2 [12]. Any real-life instance of this project must consider rotating between multiple sources and saltwater ecosystems in order to keep disruption as minimal as possible. Another potential concern could be securing the perimeter of the aquifer and all other water deposits to minimize injury to any on-site engineers and any others.

5.4 Future work

In the future, we can redesign the PCB using the correct footprints and removing the BJT. We also inadvertently forgot to connect the EN pin for the MUX, so the next design can fix that as well. We would also select a different relay with the appropriate specifications for our needs.

We would also work on helping the software communicate with the micro-controller through serial USB. This would involve changing how the software and firmware sends and receives signals from the micro-controller. After fixing the existing issues with the project, we can implement it into Professor Allen's prototype to control the system and enforce safety features at the touch of a button.

In regards to ways to improve our system, one possible improvement would be to add bluetooth capability. Due to the nature of Professor Allen's vision, our system would need to be implemented near water; making it wireless would make the experiment safer to run. From a user's perspective, it would make it much more convenient.

References

[1] R. A. Dargham. "Water doesn't come from a tap." (), [Online]. Available: https://unicef.org/mena/water-doesnt-come-tap#:~:text=The%20Middle%20East%20and%20North%20Africa%20is%20the%20world's%20most,world's%20most%20water-scarce%20countries..

[2] J. Allen. "Scrubbing co2." (), [Online]. Available: http://auditorymodels.org/index.php?n=Site.IROSE (visited on 08/22/2022).

[3] "Max31855." (), [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/max31855.pdf.

[4] "Digital+humidity+and+temperature+sensor+am2302." (), [Online]. Available: https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf.

[5] "Esp32-s3-wroom-1$_w room - 1u_d atasheet_e n$." (), [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf.

[6] "Lm1117 800-ma, low-dropout linear regulator datasheet (rev. q)." (), [Online]. Available: https://www.ti.com/lit/ds/symlink/lm1117.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1697693812014.

[7] "Snx4hc138 3-line to 8-line decoders/demultiplexers datasheet (rev. g)." (), [Online]. Available: https://www.ti.com/lit/ds/scls107g/scls107g.pdf.

[8] "J107f1cs1212vdc.36." (), [Online]. Available: https://www.digikey.com/en/products/detail/cit-relay-and-switch/J107F1CS1212VDC-36/12502789.

[9] "Irf540." (), [Online]. Available: https://www.vishay.com/docs/91021/irf540.pdf.

[10] "Mosfets and how to use them — addohms 11." (), [Online]. Available: https://www.youtube.com/watch?v=GrvvkYTW_0k.

[11] "Esp32 emulator." (), [Online]. Available: https://wokwi.com/.

[12] IEEE. ""IEEE Code of Ethics"." (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 02/08/2020).

Appendix A Requirement and Verification Tables

Table 4: Voltage Regulator - Requirement & Verification

Requirements	Verification
• Use the voltage input of 5VDC and set it to a constant level of 3.3VDC to be utilized by temperature sensors, humidity sensors, MUX, and microprocessor.	• Connect input of the regulator to 5VDC voltage supply from the USB. • Check that the input voltage and output voltage using an oscilloscope to ensure voltage was properly converted from 5VDC to 3.3VDC $\pm$5%.
• The output current from the voltage regulator must meet the minimum current requirement of 800mA for microprocessor.	• Connect input of the regulator to 5VDC and connect the output of the converter to a temporary load that equivalent to the circuit's load. • Check the output current using ammeter to ensure output current is 800mA $\pm$5%.

Table 5: Temperature Sensor - Requirement & Verification

Requirements	Verification
• The temperature sensor must operate at a minimum of 3VDC	• Connect the VCC of the IC to a 3.3VDC used to power our board along with other components • Using a multi-meter read the voltage of the IC by connecting to the VCC and GND pins. Reading a value between 3-5V ensures proper powering.
• Accurately read the 14-bit data within a tolerance of $\pm$2°C of the actual value	• Ensure that power is connected to input of IC to be able to read data. • Test with a known ambient temp, set CS to low and call read function. Compare reading to known expected value.

Table 6: 3:8 MUX - Requirement & Verification

Requirements	Verification
• MUX operates at a 2-6V	• Connect VCC and GND pins of the IC to the 3.3VDC power supply on our board • Using a multi-meter ensure the voltage read between GND and VCC is 3.3V
• Correctly set one of the 8 output pins from a given 3 bit input	• Correctly set the enable pins to output a low signal to corresponding output from the input • Reading the set value with a multi-meter should read 0V while the rest read read 3.3V

Table 7: Humidity Sensor - Requirement & Verification

Requirements	Verification
• The humidity sensor must operate with a minimum voltage of 3.3VDC	• Connect the VCC pin to a 3.3VDC power supply used to power our board with other components • Using a multi-meter read the voltage of the IC by connecting the leads to the VCC and GND pins. a reading around 3.3V ensures proper powering.
• Read the 16-bit humidity data from the humidity sensor properly	• Ensure that power is connected to the input of the sensor. • Using a known humidity for the space the sensor is in, use DHT read function. Compare the humidity data to the known humidity. reading must be within a tolerance of $\pm 2\%$.

Table 8: Level Sensor - Requirement & Verification

Requirements	Verification
• Able to read a signal that was passed in	• Send out a high signal through a pin and read through another pin that high value

Table 9: Microprocessor - Requirement & Verification

Requirements	Verification
• Decode data from the sensors to use in our algorithm for self running experiments.	• Setting the values of the sensors inputs to set value and ensuring that the processor is properly reading the data correctly from the sensors.
• Utilize our software and algorithm using the data read from the sensors and UI to be able to run experiments autonomously by powering relays.	• Continuously feeding the processor data on possible different sensor reading connect an oscilloscope to the output pins connected to the relays to see if the change when necessary.
• Process and send data it gathers about the sensors to the computer to be used for visualization for user interface.	• Save the data gathered by the microprocessor in a csv file format by hard setting the sensors to known values to ensure proper data transmission.

Table 10: MOSFET - Requirement & Verification

Requirements	Verification
• The minimum drain current of 540mA should flow between drain and source when drain to source voltage is set to 12VDC and gate to source voltage is set to 3.3VDC.	• Connect drain to fan/solenoid driver/air pump that requires current of 540mA series with 12 VDC voltage supply. • Connect gate to red probe of oscilloscope and source to black probe of oscilloscope. • Generate a square wave with a peak-to-peak voltage of 3.3VDC, DC offset of 1.6VDC, and a frequency of 0.1Hz. • Check whether fan/solenoid driver/air pump turns ON/OFF every 5 seconds.

Table 11: Relay - Requirement & Verification

Requirements	Verification
• The relay should switch ON with a 3.3VDC voltage input.	• Connect one pin 5 of relay to red probe of oscilloscope and pin 2 of relay to black probe of oscilloscope. • Generate a square wave with a peak-to-peak voltage of 3.3VDC, DC offset of 1.6VDC, and a frequency of 0.5Hz. • Check whether relay switches ON/OFF every 1 second.
• The high-voltage section of the relay should be suitable for 120VAC.	• Cut the AC cord that supplies 120VDC into two segments. • Reconnect neutral and ground wires together. • Connect one end of power wire to pin 1 and other end of power wire to pin 3 using screw terminal. • Connect the AC cord to the wall outlet and verify if it can produce 120VAC.
• The relay needs to be capable of delivering a minimum current of 5A.	• Cut the AC cord that supplies 120VDC into two segments. • Reconnect neutral and ground wires together. • Connect one end of power wire to pin 1 and other end of power wire to pin 3 using screw terminal. • Connect the AC cord to the wall outlet and verify if it can deliever 5A.

Table 12: BJT - Requirement & Verification

Requirements	Verification
• Collector current should flow when base-to-emitter voltage is set to 3.3VDC.	• Connect the base to a GPIO pin on the ESP32. • Connect the emitter to ground. • Connect the collector to pin 5 of relay. • Connect pin 2 of relay to 5VDC voltage supply. • Generate a 3.3VDC output using the GPIO on the ESP32. • Verify whether relay turns on, indicating that collector current is flowing through relay, supplying 5VDC.

Table 13: Frontend Software - Requirements & Verification

Requirements	Verification
• The software must be able to connect to the micro-processor through serial USB.	• Set the properties for the COM port in use. • Check if signals can be sent and if the micro-processor responds appropriately. • Check if data sent by the micro-processor can be interpreted accurately.
• The software must be able to graph the data in real-time for a specified amount of time.	• Print the data received and the data point graphed and check if it makes sense. • Check if the graph updates automatically when another data point is received. • Check if the plotting stops after the specified time has elapsed.