Thermal Dune Energy Storage

Miguel Baca-Urteaga, Filipe Pestana Frances, Tanner Cyr, and Michael Hernandez

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* – With as much as 50% of electricity use in the United States being consumed for home heating can a plug-and-play thermal energy sand storage device be a valuable alternative for consumers wanting to lower their energy bill by being able to take advantage of time of use electric rates?**

***Index Terms* – Energy Storage, Solar heating, power grids, Smart Grids, Heating Systems, Resistance Heating**

I. INTRODUCTION

With the renewable energy transition, there has always been the question of what to do when the sun doesn't shine and the wind doesn't blow. With the increasing deployment of renewable energy sources, we have seen that the standard response for energy storage has been through the means of Lithium Ion batteries. The demand for energy usage is increasing evermore. With roughly 50% of electricity usage within the United States being for home heating, the Thermal Dune Energy Storage (TDES) system aims to solve the problem of affordable home heating and energy storage. As renewable energy use grows, storing excess energy is key for times when sources like solar power aren't available. This project offers a more affordable alternative than traditional heating solutions like home air conditioning by using sand to store heat, which is then released to warm a room when needed. The system is ideal for homeowners and renters looking to reduce energy costs without the hassle of professional installation. Users can set a desired temperature from a display, and two heating rods rapidly heat the sand. Valves then blow out the heat until the room reaches the set temperature. The advantage of using sand is that it has a high energy density, storing thermal energy that could last for hours. This system works by allowing users to charge during off-peak hours when electricity rates are lower, saving users money. The materials used such as sand do not degrade over time, are widely available, and are environmentally friendly. Possible applications include home and small space heating, offering a sustainable solution for reducing energy bills while promoting carbon neutrality. The system's low cost and ease of use make it accessible to a wide audience. With our Thermal Dune Energy Storage unit, we aim to build a form of energy storage that provides a better cost per kWh storage rate when compared to a lithium-ion battery.

The specifications for this project were to find the necessary materials and components that would enable us to properly build and insulate this project. This goal aligned with the proper design of electrical components that would drive the whole system. With safety being our top priority, a majority of our system runs on DC voltage. This makes it much safer when compared to AC voltage. As an added step, overheating protection was included in the software to minimize the risk of fires/component failures. To ensure adherence to safety standards, we referenced the following from the National Electrical Code (NEC).

NATIONAL ELECTRIC CODE SAFETY REQUIREMENTS

Article 706.2 defines an Energy Storage System (ESS) as "One or more components assembled capable of storing energy for use at a future time. ESS(s) can include but are not limited to batteries, capacitors, and kinetic energy devices (e.r., flywheels and compressed air). These systems can have AC or DC output for utilization and can include inverters and converters to change stored energy into electrical energy."

Although the Thermal Dune Energy Storage unit doesn't exactly match the description of the traditional ESS as detailed in the NEC we decided to use it as a guide for the construction of this project. The reasoning beyond abiding by the NEC was because section 706.2 shows the first considerations of alternative energy storage solutions rather than the typical chemical form of energy storage, detailing kinetic energy devices/storage. Some factors that differentiate the TDES from the NEC definition and scope of ESS are as follows:

- The TDES will not be a permanently installed energy storage system, although in concept there can be a version that does function as a more permanent installation.
- The TDES will not have an AC or DC output, the only output the user will have from this thermal storage system will be hot air that is pumped into the room for heating.
- The TDES will not require an AC-DC converter for the 'charging' of the battery as it's designed to work with AC power input, and will not have an output inverter/converter either.

Section 706.7 talks about disconnects and remote actuation for an ESS. Within our system there will be two main mechanisms that are controlled by the main printed circuit board (PCB), this being the PCB's ability to control the HVAC dampers using solid-state relays (meant to regulate whether air is allowed to flow through the system or not), and the PCB's ability to control the relays that allow power to flow to the internal heating elements responsible for heating the sandbox. When it comes to the safety of 'remote actuation' there are two main safety layers to the system. Solid State Relays are 'Normally Open' meaning that in the situation where the PCB control system suffers from complete failure, the relays revert to their default off states cutting off power to the built-in heating elements, dampeners, and the HVAC fans responsible for pushing the air through the pipes. The second instance of power connection to the system would be the ability to simply disconnect the power plug from the wall. The AC input is wired into a fused AC terminal block alongside a true ground bus bar. The fused terminal block has two main connections, the first output runs through a 5 amp fuse which connects to a 120VAC-24VDC converter, the converter connects to a 24VDC-3VDC power supply which controls our custom PCB, and a 24VDC-12VDC buck converter which powers our HVAC fan to push air through the system.

II. SYSTEM COMPONENTS

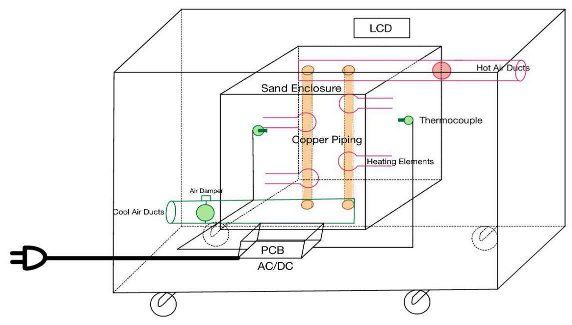

Figure 1: System Diagram

A. MICROCONTROLLER

The ESP32-WROOM-32E microcontroller module serves as the battery management unit for this project. We chose the ESP32 for its robust memory (4MB of flash) and advanced wireless capabilities, including Wi-Fi and Bluetooth Low Energy (BLE), allowing app development and a cloud-based database. Operating at up to 240 MHz across dual cores, this model handles complex computations and smoothly interfaces with our components. Its ultra-low-power coprocessor supports efficient power usage, essential for our low-voltage boards. Development and debugging take place in Arduino IDE, which lets us program in C++ and prototype quickly. With an operating temperature range of -40°C to 85°C, the ESP32-WROOM-32E ensures reliable performance, even in fluctuating heat environments, making it a perfect match for our needs.

B. LCD

The project's display is powered by a Hosyond 3.5-inch IPS TFT touchscreen module, which was chosen for its vibrant 320x480 resolution and wide viewing angles moda possible by IPS technology. This screen, controlled by the ILI9488 display driver and the FT6336U capacitive touch controller, provides precise touch response and smooth visuals, enhancing user interaction. Communication between ESP32 and the display's graphics is managed via SPI, while touch input is handled through I2C for efficient data transfers while keeping wiring simple. The TFT_eSPI library is used for efficient graphics rendering and touch input handling, improving performance and integration into the project's code structure. Our product was designed to be intuitive to the user, allowing for easy control of the settings and set temperatures via the LCD screen.

C. THERMOCOUPLE

In this project, internal temperature is measured using the MAX6675 thermocouple module, which provides precise and reliable readings for monitoring and control. The MAX6675 works with a type-K thermocouple to measure temperatures up to 1024°C with 12-bit resolution, resulting in accurate and responsive thermal data. This product is accurate to 0.25ºC, and it also includes over-temperature protection which alerts the user if the thermocouple reaches a temperature that they are not rated for, which was the perfect match to meet our needs. As thermocouples are commonplace in power plants, manufacturing plants, and HVAC industries, we agreed these thermocouples would be sufficient and safe to measure such high temperatures, despite their low cost. The project employs two MAX6675 thermocouple modules that communicate with the ESP32 via SPI, sending temperature readings in digital format to eliminate analog-to-digital conversion noise. This setup is ideal for applications that require high-temperature monitoring, and it integrates seamlessly into the project's code for real-time data visualization and analysis as the Max6675s are designed to be compatible with the ESP32 chip.

D. TEMPERATURE SENSOR

The project uses a DHT11 temperature and humidity sensor to monitor ambient conditions, providing an additional layer of environmental data to the system. Built for voltages of 3.3-5.5V and temperature range from 0-50ºC, the DHT11 is a practical, low-cost digital sensor that measures temperature and humidity. It outputs data in digital format, reducing noise and simplifying the interface with the ESP32. The project has two DHT11 averaging out for precise data. The communication with the ESP32 is accomplished through a single digital pin, allowing for easy integration and efficient data transfer.

The DHT11 would typically show an error if the ambient temperature went beyond its scope (above 50ºC), so we included a limit in our design that shows "MAX" on the ambient air section of our GUI. This is designed to also shut off the heating functionality of the design. This temperature limit equals 122ºF, well beyond the necessary heating needs in any residential living space. This product's recommended sampling rate was 1s, and given the minimal fluctuations of the PCB's ambient temperature, we decided this slow sampling rate would be more than enough to maintain safe temperatures. It allows for humidity sensing from 20% to 90% relative humidity, however, we decided not to include humidity calculation in the project as the device should not be changing the humidity of the room as it heats or charges. The sensor readings are processed in real-time, contributing to a more comprehensive analysis of the environmental factors in the project code.

E. HEATING ELEMENT

The project includes two MP32YA 8-inch stove burners, which provide a reliable heating solution for applications that require precise and consistent temperature control. Each burner has a power rating of 2100 watts, allowing for fast heating and consistent performance. The ESP32 controls the relays that turn the burners on and off based on real-time temperature readings, ensuring precise temperature management for optimal heating conditions. This setup integrates seamlessly with the project's control system, allowing for precise adjustments and real-time monitoring; this enhances the overall functionality and reliability of the heating elements in this application. Because the heating element's power rating is on the condition that we are running on 240V we needed to maximize the power throughput as we were running the system on a standard 120V outlet, as a result, we ended up wiring the two heating elements in parallel to halve its resistance. With this change we were able to achieve a heating power draw of 860 watts, this ensures fast charging of the energy storage system while ensuring that the overall peak power draw of the system stays well below the standard 15 amp rating of the NEMA 5-15 wall outlet as well as the typical 15 amp circuit breaker.

F.

G. BLOWER/AIR PUMP

The project incorporates the Attwood 1733-4 Turbo 3000 series in-line blower, which is meant to ensure that the heat stored in a designated box is effectively released to warm the ambient space. This device was originally intended for nautical uses as a bilge pump to force water out of a boat's hull to lighten the boat's weight. We were able to repurpose it to act as an air pump to send air through 3in HVAC into 2x 5/8in copper tubes to vent hot air out from our design. This component was simple to install due to its mounting feet where we could screw the blower to the frame of our design. The pump can supply 6.8hp which was more than enough for our system.

We decided to place the blower/pump before the HVAC damper in the air intake part of our design to limit its exposure to hot air when our design remains idle or "charging". This would increase the lifespan of the component and provide reliability to the overall design. The component is also easily replaceable due to its placement in the design, simply taking off a siding panel to our design would provide easy access for the user to troubleshoot, should a component fail. The design's intake vent must be clear from debris or airflow constriction so the blower can reliably push air through the design without burning out its electronics.

By blowing hot air through a copper tube, the blower improves heat transfer, making it an essential component of the heating system. It has an airflow capability of 300 CFM and can provide excellent ventilation when needed. The ESP32 controls the blower, allowing it to work depending on the temperature data, ensuring that the heat is released at the most appropriate time to maintain a comfortable environment. This configuration increases the heating efficiency while maintaining proper air movement.

H. RELAY

The system effectively manages essential operations using three solid-state DC to AC relays (SSR-25DA). One of the relays is dedicated to controlling the charging process, which directly manages the heating components, allowing for precise on-and-off functions as required. The other two relays are used to control the heating process, which focuses on heating the ambient space. These relays enable you to turn on and off the heating, open and close the valve, and activate the blower, ensuring that warm air is efficiently circulated into the environment. These solid-state relays provide reliable, fast switching with minimal heat production and noise, and their integration with the ESP32 allows for automation based on real-time conditions, optimizing the overall heating and airflow processes for improved performance.

I. ELECTRONIC DAMPER

The electric air damper is an essential component of the project's airflow system, designed to manage air passage with precision and durability. This valve is suitable for high-temperature applications. It has an electronic mechanism that opens and closes automatically based on control signals from the ESP32, which adjusts the airflow in real-time for maximum efficiency. The damper takes an average of 10s to fully close or open (when initially fully opened/closed), which is a response time adequate for our design's requirements.

When the heating process is active, the valve opens, allowing air to pass through from the blower, which then circulates warm air into the ambient. When the valve is closed, it uses its silicone sealing ring to cut off airflow around the edges. To switch states from on-off or off-on, the device must be powered electronically with 24VDC. The damper is rated for 760ºC, well surpassing our design's specifications/temperature maximum. Upon testing internal temperatures of the sand enclosure close to 160ºC, the metal of the HVAC damper only read to be 25ºC which guaranteed the part would be within its safe environmental temperatures. Its automated operation not only increases system efficiency but also integrates smoothly with the project's overall control system to maintain stable and comfortable temperature levels in the ambient.

J. HVAC/COPPER TUBING

Our design included 3-inch HVAC piping, the blower and electronic damper listed above, and 2 ⅝ in copper tubes to carry the air through the sandbox to absorb the heat from the sand. The two copper tubes run parallel through the core of the sand enclosure, and they run through the center of their respective heating element. The proximity of the copper tubing and heating elements allows for heated air to pass through the system if it is turned on from a cold state. As sand radiates heat slowly, we needed the ability to heat the sand closest to the ventilation quickly so that the user could turn on the device and expect heat within 15 minutes. When the device has time to "charge" the sand with heat before venting the heat into the room, the user can expect heat instantaneously. As the diameter of the tubing from 3 inches to 1 inch (total) decreases when entering the enclosure, we were able to create a system with decreasing pressure. This meant that the air pressure in the 3-in diameter was greater than that of the 1-in (2 x 5 /8in), which helped keep the heated air from inside the enclosure from flowing backward into the air damper and fan.

Through Bernoulli's Principle, we were able to calculate that our pressure lowered when the pipe diameter lowered. We were also able to conclude that the velocity of the air flowing through the system tripled as the cross-sectional area of the pipe shrunk by two-thirds.

Bernoulli's Principle:
In Bernoulli's Principle equations seen below. A1, V1, and P1 refer to the 3-inch HVAC pipe entering our system, and A2, V2, and P2 refer to the copper tubing running through the enclosure to the exhaust vent.

A1*V1 = A2*V2

$$P_1 + \frac{1}{2}\rho v_1^2 + \rho g h_1 = P_2 + \frac{1}{2}\rho v_2^2 + \rho g h_2$$

ρ	= fluid density
g	= acceleration due to gravity
P_1	= pressure at elevation 1
v_1	= velocity at elevation 1
h_1	= height of elevation 1
P_2	= pressure at elevation 2
v_2	= velocity at elevation 2
h_2	= height at elevation 2

In other words, as our pipe diameters shrink, the air velocity inside them increases, but the pressure decreases. This concept is often considered counterintuitive, but we can explain naturally occurring phenomena with this principle, such as arterial blood flow and aerosol cans where the gas/liquid is pushed into a smaller blood vessel or tube due to the higher pressure from the wider diameter into the smaller circumference.

To maximize the heat applied to the copper tubes, we were able to coil more copper on top of the sand enclosure underneath the insulation. This allowed for the steel frame of the enclosure to radially heat the copper as the copper tubing traveled to the exhaust vent. This allowed the air/copper to stay in contact with heat for an extended amount of time and reduced the amount of insulation needed to wrap the exposed copper in insulation, allowing the product to be safer for the user and to cut down on material cost. We found that using 90º copper joints allowed the air velocity to stay congruent with the incoming air. This was a necessity as we were limited to constant airflow due to our blower's lack of speed variability.

We advise the end-user to keep the intake and exhaust vents of our design clear of debris or objects that would obstruct the airflow of the system. The exhaust temperatures should regularly output air at safe temperatures, however, we do advise the user to be mindful/cautious of the exhaust air. Extended exposure to heat can cause damage to household objects and can be a fire hazard. We suggest keeping the design at least 2-3 feet away from combustible objects, curtains, and flammable paint. We suggest only using extension cords for the device capable of handling 120V at 15A to operate the machine.

Using cables rated for smaller currents can increase the risk of electrical fires and damage to said cables.

The HVAC-copper connection was mechanically made using tools designed for HVAC technicians. We were able to bevel out the end of the copper tube connected to the HVAC so that it would rest against the inside of the HVAC tube. We lowered the diameter of the 3-inch HVAC tubes by making lateral incisions to the end of the tube and collapsing the "flaps" inward towards the beveled copper. We were able to utilize high-temperature steel paste to help the HVAC connection maintain its shape around the copper. Through this design, we were able to allow for maximum airflow into the copper tubing and minimized air backflow. Our Design reliably takes 75-80ºF air and outputs air at temperatures from 120º-140ºF. This is typical of most space heaters at their exhaust point/area, which can be harmful to human extremities for long exposures. We have not tested the device in climates colder/dryer than 70ºF at 20% humidity, but we expect similar heating results from our device.

K. POWER REGULATION

Our design can run on any 120V 15A household circuit, and as the design contains "raw" 120V, we took precautions to ensure the safety of the design team and the end user. Our heating elements required 120VAC connections, but all other components required DC voltages. To connect the heating elements with AC, we sent our power cable to a bus bar with a fuse built-in and connected a solid state relay (SSR) before the heating elements in parallel. This relay allowed for a low voltage to act as a switch for turning on/off the power to the heating elements. We used similar SSRs to control the power to the air pump and the air damper valve.

One way we managed this setup was by using an off the shelf (OTS) AC/DC converter to change 120VAC to 24VDC to power the rest of the electronics. With 6A available at 24VDC, we powered all electronic components besides the heating elements. We also have an OTS 24V-12VDC converter to provide power to the air pump which requires 12VDC. Being able to switch the power on and off for each component excluding the GUI was essential for our design to operate as specified.

III. SOFTWARE

The ESP32-WROOM-32 was programmed using the Arduino IDE. The ESP32 natively supports C++ programming, this is beneficial as it allows for greater control of overall memory management and allows for object-oriented development. Although C++ allows for greater system control the main drawback would be that implementing higher-level features comes with some extra difficulty, although some features we implemented would've been easier with Python, we still succeeded in ensuring that every feature we wanted to implement could be done in C++. The programming and software of our project extend beyond just the ESP32 as we have the ESP32 connecting to a cloud-hosted webserver, a cloud-hosted database, and finally, we have an android-based web application that can interact with the ESP32.

The Android Application (named 'TDES') was made using the Android Studio IDE. The Arduino IDE allowed for streamlined creation of the mobile app user interface, especially since the user interface isn't required to be a very dynamic system. The TDES app has three pages each with differing functionality. The main landing page after you log into your account allows you to get live readings from the ESP32, being able to see the currently set desired ambient temperature, see the current room ambient temperature, see what the currently set heating and charging schedule is, and have the button functionality to be able to toggle the current heating/charging status of the ESP32. The second page denoted as 'add battery' allows the user to associate a certain TDES unit with their account, in this page they're able to give the TDES a nickname as well as having a field for the actual battery identification number. When the user is on the 'add battery' page the 'add battery' button changes in functionality to then send a post request to the webserver to search the MongoDB for the desired TDES system, searching if the TDES exists and if so it will be associated with the user's account.

The approach to extending the functionality of the TDES beyond just the ESP32 was done to mimic a MERN stack application to the greatest extent. With the MERN stack implementation style in mind, we set up the webserver to run using Expressjs in a Nodejs environment. Our web server is hosted on the digital ocean platform, the 'droplet' is running an Ubuntu server with '1 vCPU', 512 MB of RAM, and 10 GB of space. The web server is hosting API endpoints for both the actual TDES unit as well as the mobile app, the webserver endpoints allow for the ESP32 to check for important flags like the 'heatingToggle', the 'chargingToggle', and 'scheduleFlag' flags. If the user toggles the charging status, and heating status, or saves a new schedule to the webserver these flags will be set to true for the TDES object that's saved in the 'sandBattery' collection of the MongoDB. When the user saves a new schedule on the mobile app a post request is sent with the desired start and end times for charging and heating, the webserver then assigns a true value to the 'schedulingFlag' variable so the ESP32 will be able to determine that a new schedule is going to be followed.

The MongoDB is a pretty simple implementation, splitting its data amongst two collections, the first collection being 'users' which stores the user JSONs, the attributes for data in this collection are simply the user's email, encrypted password, and the batteryID associated with their account. The second collection is titled ‘sandBatteries’ which simply stores all the TDESs’ and all their associated attributes.

The ESP32-WROOM-32 is set up using FreeRTOS to have our program run as a real-time multithreaded application. The ESP32 needs to interface with a touchscreen, two SPI-connected thermocouples, two DHT11s, and three relays, and maintain an internet connection to a cloud-based database. With the amount of components and processing the ESP32 needs to do, having the ESP32 run in a multithreaded fashion allows for the limited resources to be shared by the 6 tasks and their corresponding functions. When the ESP32 is powered on often global variables like whether the room is being actively heated or not will turn off, the reason for this is to have the ESP32 revert to a sort of resting state upon initial start-up. The threads and their functionality would be as follows:

- showTime - responsible for ensuring that the time is displayed on the main screen
- wifiRask - handles the DNS requests from the captive portal which is displayed after connecting to the local access point to allow the user to connect the ESP32 to the internet
- sendDataTask - checks if there is currently an internet connection, if not the ESP32 runs the ‘connectToWiFiTask()’ which is responsible for creating the local access point and captive portal. Upon connection the thread starts running the functions responsible for checking the toggle flags in the database as well as sending periodic TDES status updates every 15 seconds.
- internalTemp - reads the temperatures from the DHT11’s and k-type thermocouples, aside from constantly updating the global variables every 500 ms, the temperature is displayed on the screen so long as the LCD is currently displaying on the main screen.
- heater - the heater thread is largely responsible for the scheduling functionality of the overall system. The heater thread checks the global variables that are storing the starting hour and minute for room heating, start hour and minute for battery charging, end hour and minute for heating, and end hour and minute for charging. The heater thread is also responsible for the built-in safety functionality of the overall TDES system. As an example, the user can toggle/override the room's current charging and heating state. The heater thread along with the ‘internalTemp’ thread is responsible for ensuring that the ambient temperature and the sand temperature are within the safety parameters we have set, for example, we want to ensure that the sand does not surpass a temperature of 500C, so with this, we make sure that the global average internal temperature variable does not exceed our 500ºC number.
- touchInterface - this thread is amongst the most vital in the whole application. The touch interface constantly polls the FT6336U touch controller checking its current touch status flag. When the touch interface is interacted with the touch status displays a true value. After the screen is touched we proceed to get the precise x and y coordinates of the touch location, within this section of the code we check the current state of the global ‘screenStatus’ variable, the reason we do this is that with a value of 0, we would know that we are on the main screen but a value of 1 will indicate that we are on the settings page. In the instance where we have a screen status of 1, we have the main screen divided into different touch sections. As an example we have the charging and heating toggle buttons, in case the charging button is pressed we would toggle the global ‘chargingState’ variable and then call the ‘chargeFunction()’ to then be able to properly toggle the charging state of the TDES. In the instance where the user presses the heating toggle button on the main screen, we would check the status of the global ‘heatingRoom’ variable and then call the corresponding ‘turnOffHeat’ or ‘turnOnHeat’ function. In the case where the user taps on the internal temperature section of the main screen then the display would change from having the Celsius reading to showing the relative percentage charge of the battery. When the user presses the settings button from the main screen we would then change the global ‘screenStatus’ to 1 to indicate that we are now on the settings screen and then we would call the ‘printSettings()’ function. If the user is already in the settings button we various selection regions, we are highlighting the different areas of the screen that are selected. In the settings page, the user will have the ability to highlight and select the start and end heating/charging times. Upon selection, the user would be able to increment and decrement the selected time by 15-minute intervals. When the 15-minute interval reaches the hour the minutes section resets and the hour is incremented by 1. The user also has the option of selecting and changing the desired ambient temperature, we have an upper limit of 37C for the overall ambient temperature as we want a big

enough buffer to ensure safe room temperatures for our DHT11s as well as any users, lastly, the user can change the temperature scale of the overall system from celsius to Fahrenheit, this would change the temperature displays on the main screen to Farenheit and changes the temperature scale for the desired ambient temperature section to Fahrenheit as well.

IV. ELECTRONIC HARDWARE

Our hardware was built to meet the power and communication requirements of the TDES. The main electronic systems used for our project involve the following:

A. MCU Board (Nightsky)

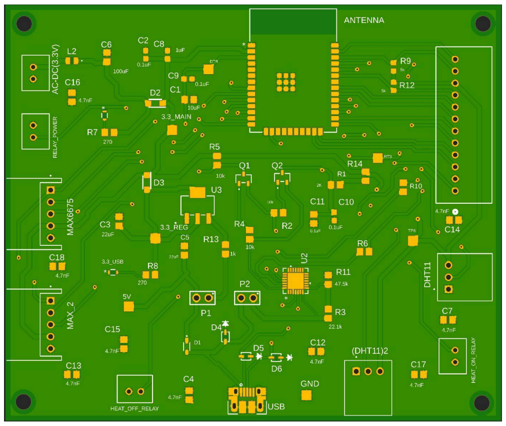

Figure 2: Nightsky PCB

The Nightsky PCB serves as the main processing unit of this project, housing a 2-layer board with an ESP32. Each peripheral securely connects to the board via JST connectors, receiving both power and data. Power to the ESP32 and peripherals can be supplied in two ways: via a micro USB, which provides a 5V input regulated to 3.3V using an AMS1117 LDO, or through a custom power regulation circuit. This circuit, located on the top left, includes a 2-pin JST connector that connects to a custom buck regulator. The input is filtered by a ferrite bead and bypass capacitor to ensure clean power flow.

To prevent one power source from leaking into the other, we use two Schottky diodes with a low forward voltage drop of 0.2V. The 5V from the USB also powers the programming interface, using the CP2102 USB-to-UART bridge to interface with the ESP32 and open the COM port for the Arduino IDE. A voltage divider is employed on the bridge's VBUS input to maintain correct voltage levels, calculated using the standard voltage divider formula.

$$V_{OUT} = V_{IN} \times \frac{R2}{R1 + R2}$$

The bridge needed a way to pull down GPIO 0 and the ENABLE line of the ESP32 to enter bootloader mode efficiently. To achieve this, we placed two NPN transistors on the RTS (Request to Send) and DTR (Data Terminal Ready) lines, creating a push-pull configuration. Combined with a 10-12K ohm pull-up resistor, this setup enabled code uploads and allowed the ESP chip to resume normal operations afterward.

Electromagnetic compatibility (EMC) was a priority in the board layout design. Since analog and digital signals share the same plane, interference can occur, leading to erratic behavior. To prevent this, we implemented PCB zones, ensuring circuits stayed close to each other while remaining distant from high-speed digital lines like SPI and sensitive analog lines. As shown in Figure 1, the PCB antenna is positioned at the board's edge, away from other components, minimizing RF interference. A keep-out zone around the antenna prevents grounding beneath it, avoiding interference feedback into the reference plane. For optimal current capability, power traces were made with 36 mil traces, while signal lines used 20 mil traces, ensuring compact board layout and minimized resistance.

B. 12/24V to 3.3V Buck Regulator

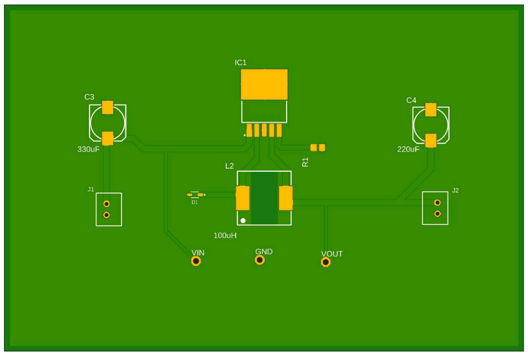

Figure 3:Buck Regulator

Having our off-the-shelf AC-DC converter handle the conversion for 24V DC, we were required to have another board that could handle the conversion for our low-voltage devices. This was the requirement that caused the creation of our custom power supply. The board in Figure 3 utilizes an LM2596 switching regulator that down steps our 24V to 3.3V, the optimal range for our Nightsky board. Using two JST connectors, the LM256 can handle a voltage input of up to 40V with a maximum current load of 3A. The layout for this board resembles the typical application that is used for continuous operation. Two large electrolytic aluminum capacitors were used to filter large voltage transients from occurring at the input and to help with voltage regulation at the output. When choosing the capacitors, it was imperative to find models that have a low ESR value at 100kHz based on the graph below

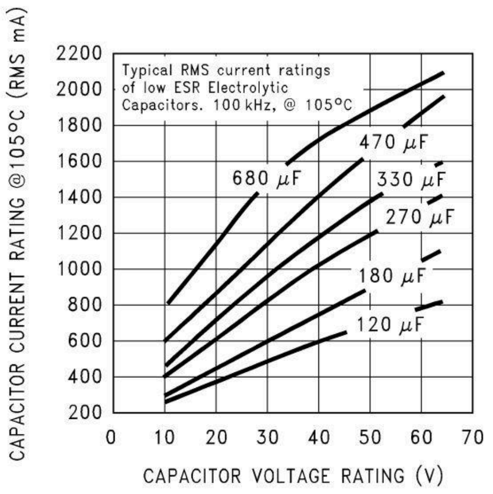

Figure 4: Capacitor Ratings

The inductor here below is used for filtering out ripple voltage that may be caused from switching of the regulator. While the diode is required as a catch diode that catches current from the inductor when the regulator turns off.

IV. CONCLUSION

Combining the mentioned systems above, we were able to create a fully functioning system with overlapping engineering fields/standards. The design is capable of storing energy as heat and is optimized for temperature regulation in a home, with a simple and intuitive user interface.

V. BIOGRAPHY

Tanner Cyr will receive his Bachelor of Science in Electrical Engineering in December of 2024. He currently works as an engineering intern for Lockheed Martin Corporation and as an Electrical Engineer for Prairie House Coffee Roasting. Before Lockheed Martin, he was employed by ZRadio (88.3FM) as an Audio Engineering Intern. His primary interests lie in PCB design, FPGA integration, and audio hardware design and development.

Michael Hernandez will receive his Bachelor of Science in Electrical Engineering in December of 2024. He currently works as an electrical engineer intern for Hoverfly Technologies Inc. Before Hoverfly, he served in the United States Marine Corps as a Radar Technician. His primary interests are in circuit design, power systems, and embedded/IoT applications.

Miguel Baca-Urteaga will receive his Bachelor of Science in Computer Engineering in December of 2024. He currently has a systems engineering internship at ESA Solar. His primary interests are in embedded systems and multi-thread applications.

Filipe Pestana Frances will receive his Bachelor of Science in Computer Engineering in December of 2024. He is currently working on a personal project and looking for a job as a software engineer. His primary interests are in design verification engineering, front-end development and embedded systems..