PHEBCS: Passive House Exterior Blinds Control System

Analog Electronics Laboratory
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Julia Arnold
jul@mit.edu

Nancy Hidalgo
nancyhid@mit.edu

***Abstract*—Passive houses primarily rely on their architecture and the environment to maintain a comfortable temperature for their residents. Specifically, they have large windows on the south-side with large eaves that block the high sun in the summer and allow for more sunlight in the winter when the sun is low. This energy harvesting can be facilitated with exterior blinds on the south side with a sensor-controlled system, PHEBCS. PHEBCS uses readings from commercial temperature and wind speed sensors to determine when to raise and lower the blinds to keep the temperature inside the house within the desired range. These sensors will use an RF communication scheme to send information to an Raspberry Pi inside the house that will then communicate with the blinds to raise and lower them. In the event of power outages or other extenuating circumstances, the system must continue functioning; therefore, another aspect of the project was to design a power supply system.**

I. Problem definition and background

Passive houses take advantage of and largely rely on natural phenomena to heat and cool the interior. Harvesting energy by leveraging environmental sources reduces the cost of energy and the use of non-renewable energy sources. For example, the windows are designed so that in the summer, the high summer sun doesn't directly enter the house, but the lower and less intense winter sun does. See Figure 1 for a diagram of this effect.

To provide more control over the indoor temperature, there are also exterior blinds on all of the windows that are manually controlled with a remote. The blinds can be manually lowered when the indoor temperature gets too hot, or raised when the wind speed gets too high, since high winds could damage the blinds. The goal of this project was to design a system that automatically raises and lowers the blinds, based on the indoor temperature and the outdoor wind speed in order to increase convenience and optimize energy collection. Additionally, a power supply for this system is included so that in the case of a power outage, the system would continue to function.

The overall goal was to design and build a system that would automatically raise and lower the blinds based on current wind and temperature conditions. The system is totally autonomous in determining whether to raise or lower the blinds based on interior temperature with built in protection from wind damage. If the residents still want to manually control the blinds (i.e. to make a room darker) the remotes are still functional.

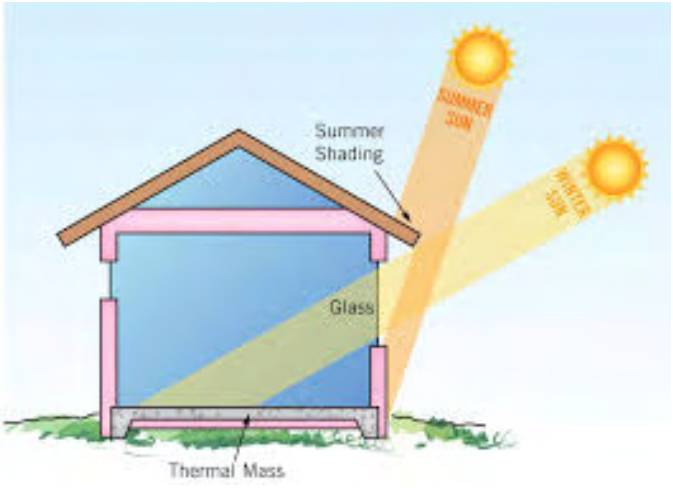


Fig. 1. Diagram of Passive House

The system was designed with several constraints in mind. First, the remote for the blinds operates at a frequency of 433 MHz. Next, the placement of both the wind and temperature sensors relative to each other posed a challenge. The wind sensor had to be placed outside, and far enough away from the house to get an accurate measurement of the wind speed. Meanwhile, the temperature sensor had to be placed inside with the Raspberry Pi. The power supply had to function both from a typical American wall outlet, and from a 9V battery. It also needed to be as efficient as possible to maximize the time it could run off of a 9V battery. The Raspberry Pi could handle currents up to 825 mA, but has been tested to have optimal (minimal) power consumption when drawing 20 mA of current.

PHEBCS was designed with the author's house specifically in mind. An illustration is included in Figure 2 for a demonstration of the large windows typical of a passive house. Additionally, the RF controlled blinds can be seen where they

Fig. 2. The author's house

have been installed over the windows.

II. Circuit Design

The overall block diagram is laid out in Figure 3. The wind sensor and temperature sensor flow into the Raspberry Pi responsible for instructing to raise or lower the blinds based on received information. Blocks surrounded in red will only be simulated; the remaining modules will also be built physically. We tried to make sure that the projected could be completed remotely and asynchronously, which proved to be no small feat.

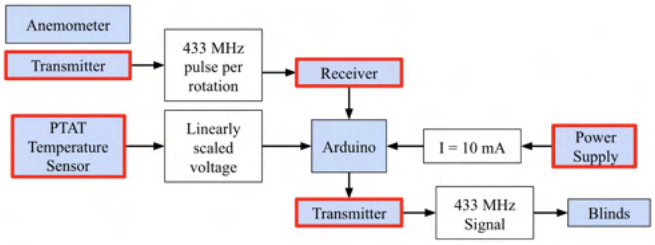


Fig. 3. Block diagram of PHEBCS

A. Power Supply-Nancy

The power supply had two main modes of functionality: a normal, wall-outlet mode and a power-outage, 9V battery mode. In the Normal mode, the power supply needed to take the 120V AC from the wall and turn it into 3.3-5V DC. Additionally, the Raspberry Pi could only handle currents up to 825 mA. In power-outage mode the system had to take 9V DC and turn it into 3.3-5V DC. I learned that in order to minimize the power consumption of the Raspberry Pi, one should run the clock at 20 MHz and with an input current of about 20 mA, hence the decision to set the output current at 20 mA.

For normal operation, the voltage is stepped down, then rectified and smoothed, then stepped down further. This was accomplished with a step-down transformer, a Full-Wave Bridge Rectifier with a smoothing capacitor, and a switching voltage regulator, also known as a Buck Converter.

For the Transformer and Bridge Rectifier, the VPP24–1250 transformer was used to step down the wall outlet voltage from 120 V AC to 12 volts AC. This was the voltage decided so that once it went through the recitifier and selection circuit it would be high enough to power the IC and it would provide a high enough voltage for the selection circuit to function properly. Then, for the Bridge Rectifier Schottky diodes were used to minimize power consumption. The CMDSH2-3 was chosen because of its low forward voltage (.3 Volts) and suitable reverse breakdown voltage of 30 Volts. This was more than enough for this application. For the smoothing capacitor, we used a 100 μF capacitor to minimize the voltage ripple on the output voltage. This resulted in a 0.4 V ripple on the rectified signal.

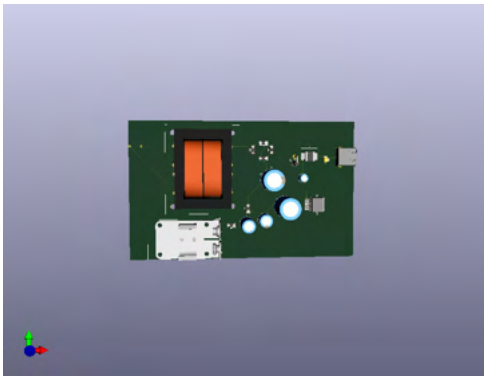


Fig. 4. Power Supply Layout

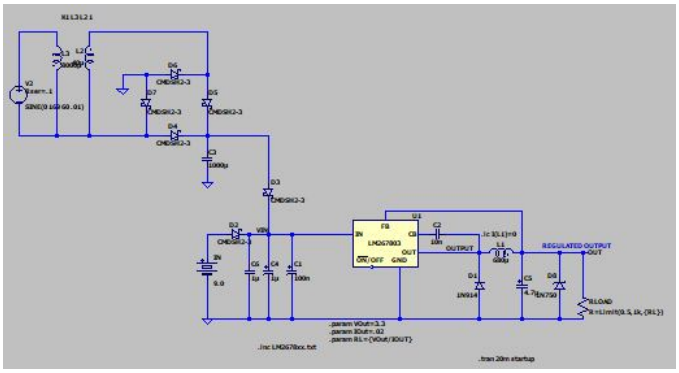


Fig. 5. Power Supply Schematic

For operation during a power-outage, the voltage just needed to be stepped down. To accomplish this, the 9V battery was connected to the Buck-converter.

To switch between the two modes, a simple selection circuit was made with two diodes and a capacitor. This way, when the capacitor was charged up from the wall, the potential difference and the diode would block current flow.

To step-down the voltage we chose to use a switching buck converter topology rather than a linear regulator. We made this choice because this would conserve energy and be much more efficient. For the Buck converter, a typical buck converter topology was used, with a IC to control the duty-cycle of the pulse-width modulation of the mosfet that controls the on/off cycle of the Buck converter.

The Buck converter works steps down the input voltage and steps up the input current by switching between an on state in which the current through the inductor increases and an off state. During the on state, the mosfet allows current to flow resulting in a voltage drop across the inductor equal to the difference between V_{out} and V_{in}. In the off state, the stored energy in the inductor enables it to behave as a current source and the current through the inductor decreases with a slope equal to -V_{out}/L.

The IC chosen was the LM2678. This one was chosen because of its high-efficiency PWM control. While other IC's were found that would've likely been more efficient at these values of output current and output voltage, unfortunately there were no LTspice equivalents for them (although there were encrypted pSpice simulations available from TI as well as 'Tina-TI' models). This IC measures the output of the Buck Converter, and from the amplified error adjusts the duty cycle of the mosfet that drives the On-Off cycle of the Buck converter. The mosfet is an N-channel power MOSFET, which means it has a low on resistance, but a higher V_{gs}. To meet this higher V_{gs}, the IC has a pin for a bootstrap capacitor. This bootstrap capacitor works as a step-up charge pump to raise the V_{gs} of the mos within the IC. This demonstrates the need for an external input voltage for the IC. The IC has an internal oscillator crystal that has a switching frequency of 260 kHz. This higher switching frequency generally means increased efficiency. In normal mode, the Buck converter had a duty cycle of about 30 percent; in power-outage mode the duty cycle was about 36 percent.

Using this IC had certain challenges. The primary one was setting the output current. It was tricky to drive such a low current with this IC. When trying to drive down the output current, there was a little trouble with the mode of conduction of the Buck converter. There were some iterations that would operate in discontinuous conduction mode, meaning the current in the inductor falls below zero before the end of a single cycle. This resulted in strange behavior including increased losses and noise in the output voltage. Other losses were likely due to the inductor chosen (MSS1210) . This specific 680 microhenry inductor was chosen because of its ability to handle currents of up to 1.3 Amps and the amount of current it could drive, although it does have a series resistance of about 500 milliOhms. This inductance value was determined from the following equation:

$$L = Vout \times (1 - D) \div (f \times DeltaI)$$

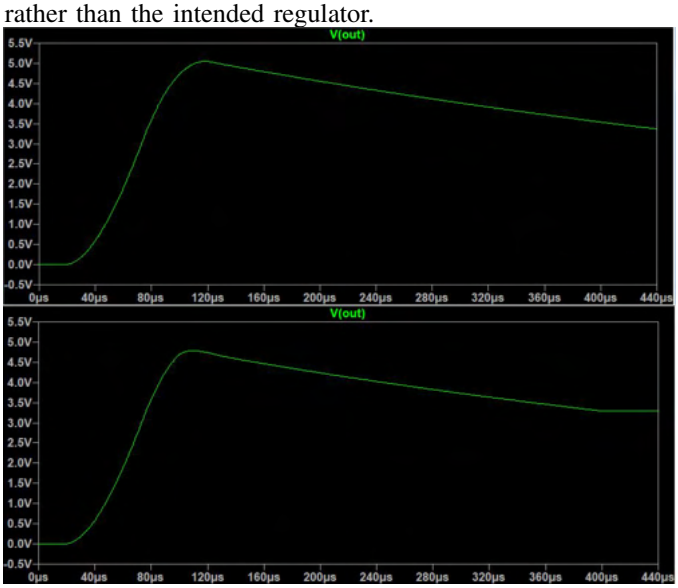
where

$$D = V_{\text{out}} \div V_{\text{in}}$$

and ΔI is the change in current from the input current to the output current.

The final design is operating in continuous conduction mode, meaning that the current through the inductor doesn't fall below zero. However, throughout the design process as the current was driven lower and lower, the conduction mode of the Buck converter teetered on the border between continuous and discontinuous, with the inductor current dropped to just 3 mA at its lowest point in a single cycle. This mode of operation introduced a lot of loss into the system, and is likely why I ended up with the efficiency that I did. The lower power constraint of the system proved to be more challenging to overcome than expected.

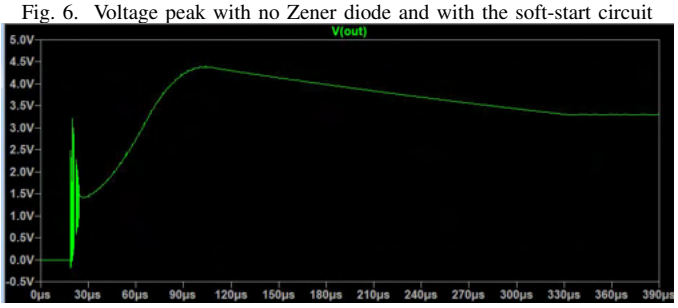
Furthermore, before achieving steady state, the circuit reaches a voltage of about 5 volts. Since the Raspberry Pi can only handle up to 5 V, I tried to mitigate this voltage peak with a Zener diode. However, when the V_{out} exceeds the breakdown voltage, the current through the diode is so high (approximately -90mA) that in real life this would only work once before frying the Zener diode. It would act as a fuse rather than the intended regulator.



Mark suggested using the soft start pin of the IC to reduce the initial output voltage spike. The LTspice model of this IC doesn't have a pin for this, however there is a way to implement the same functionality without the pin. You can use the feedback and output pins to 'trick' the feedback pin so that it seems as though the output voltage is higher than it is. This then reduces the PWM duty cycle, decreasing the output voltage.

If the circumstances were different,(i.e sans COVID-19) I likely would've went without using the IC to control the PWM of the Buck Converter. Instead, I would've and driven an N-channel power mosfet with a square-wave of a varying duty cycle based on the error of the output voltage. Nonetheless, this was a valuable experience with simulation tools and circuit design.

Fig. 6. Voltage peak with no Zener diode and with the soft-start circuit



B. Temperature Sensor - Julia

To design a temperature sensor for inside the house, a proportional to absolute temperature (PTAT) circuit was chosen to leverage the temperature variations of BJTs to create a current output that is linearly dependent on temperature, which can then be used to measure the indoor temperature.

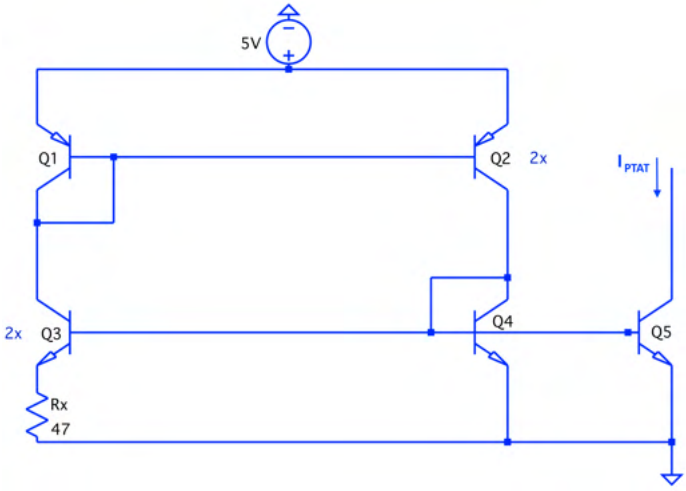


Fig. 7. A schematic of the PTAT current source

The first step of the design process was to design a current source that is dependent on temperature. The schematic for the current source may be found in Figure 7. Due to the 1:2 ratio between Q1 and Q2, and the 2:1 ratio between Q3 and Q4, we may determine that

$$I_{C2} = 2I_{C1}$$

$$I_{C4} = 2I_{C3}$$

$$I_{S3} = 2I_{S4}$$

We also can define the voltage over the resistor R_x as

$$V_x = V_{BE4} - V_{BE3}$$

$$V_x = V_{th} ln\left(\frac{I_{C4}}{I_{S4}} - \frac{I_{C3}}{I_{S3}}\right)$$

$$V_x = V_{th} ln\left(\frac{I_{C4} I_{S3}}{I_{C3} I_{S4}}\right)$$

$$V_x = V_{th} ln(4)$$

where $V_{th} = \frac{kT}{q}$. Finally, to find I_{PTAT}, we know V_x is related by the following

$$V_x = I_{C3} R_x$$

$$I_{C3} = \frac{1}{2} I_{C4}$$

$$I_{PTAT} = I_{C4} = \frac{2V_{th} ln(4)}{R_x}$$

While we cannot know the exact I_S dependence of each of the transistors, we can use the ratios that we know from matching transistors fabricated next to each other on a die. The addition of parallel BJTs at Q2 and Q3 is crucial to creating a large V_{BE} difference between Q3 and Q4. [3]

We may now turn out attention to the transimpedance amplifier. The inverting input of the transimpedance amplifier is connected to I_{PTAT} of the current source. R_f connects V_{out} with the inverting input of the op-amp to create a feedback loop. The non-inverting input is biased at 1V to maintain the same voltage at the inverting input. The feedback resistor creates the relationship

$$V_{out} = I_f R_f$$

The resistor R_p is in place to subtract a constant current from I_{PTAT} to uniformly decrease V_{out} and avoid railing the op-amp. The value for R_p was calculated to consume 1 mA of current by

$$R_p = \frac{5V - 1V}{1mA} = 4k\Omega$$

The current in the feedback loop then becomes .5 mA to .9 mA so R_f then becomes

$$R_f = \frac{2.5V - 1V}{.5mA} = 3k\Omega$$

since the ideal voltage range is from about 2.5 V to 3.5 V. In reality, the measured current was a bit higher than in simulation, so R_f was decreased to 2kΩ to proportionally decrease V_{out}.

The result of these design considerations can be seen in Figure 8. The analog voltage at V_{out} will be connected as an input to the Raspberry Pi to assist the logic in determining the blinds state based on the interior temperature.

C. Wind Sensor - Julia

The wind sensor was designed to be placed outside to measure the speed of the wind. For the wind sensor, we used a Hall sensor and an anemometer (Figure 9) with a magnet attached to one of the cups to measure the relative wind speed. Because the wind sensor has to be placed far from the temperature sensor and the Raspberry Pi, RF transmitters and receivers were used to send the generated signal.

The general schematic for this circuit may be seen in Figure 10. The behavior of the Hall sensor is shown in Figure 11. Essentially, the Hall sensor outputs a high voltage when there

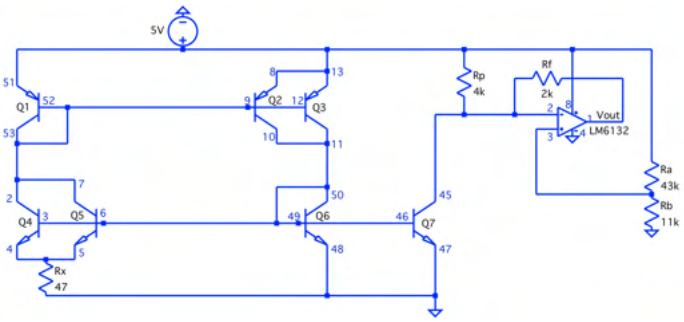


Fig. 8. Schematic for PTAT on-chip thermometer circuit

Fig. 9. Anemometer used for project

is no magnetic force nearby, but when an object like a magnet creates magnetic flux, the output goes to zero. The design in Figure 10 uses an NPN to invert this output so that the antenna then transmits only when there is a magnet nearby.

This system was designed with a low-power constraint in mind. Since the module was intended to be battery powered, it should be able to last as long as possible before switching the battery. By leveraging the inverse output of the BJT, the power consumption of this circuit was significantly lowered.

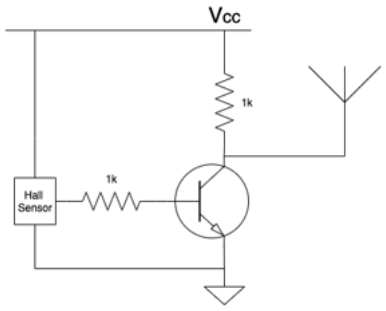


Fig. 10. Schematic for wind sensor

The greatest challenges in designing this module were the range of transmission and communication protocol. The range of the purchased transmitter was directly related to the voltage applied to V_{cc}, which made a 9V battery preferable to a smaller voltage. Additionally, it was discovered that the blinds send a constant pulse which created a lot of noise at the same frequency. In order to improve this design, a stronger communication protocol should be added to ensure that the

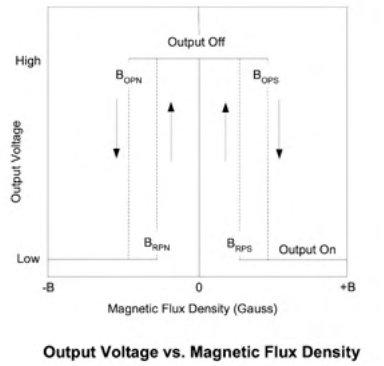


Fig. 11. Behavior of Hall sensor

Raspberry Pi successfully receives the signal from the wind sensor.

D. RF Sniffer

In order to be able to send a signal the blinds would recognize, the signal sent by the remotes had to be recorded and then replicated. This began with inspiration from Ray's Hobby [5]. The first attempt involved wiring a receiver with an audio jack at the digital output and then recording the output in the sound mixing software Audacity. While this method was effective, the next solution was better once the project moved on board the Raspberry Pi.

The next method for RF sniffing involved the GPIO functions included with Raspberry Pi. Please reference the appendix for the code ReceiveRF.py that was used to capture and graph each signal so that it could be broken down into 0s and 1s for re-transmission [2].

E. Software - Nancy and Julia

In order to control the blinds, the Raspberry Pi was programmed to take the information from the wind sensor and temperature sensor, and then decode it and determine whether the blinds should be raised or lowered. Then it had to send commands to connected RF transmitter to control the blinds. The script "PHEBCS.py" is available for reference in the appendix.

The first step was to decode the wireless signal sent by the existing blinds remotes as discussed in the previous section. The programming then uses this information to transmit to the blinds upon command.

The script also receives the signal from the wind sensor and counts the instances where there is a clear zero because that is when the air became clear based on observation. If the threshold number of zeros was reached, the blinds would open to protect them from wind damage. This algorithm is not as robust as it could be and would benefit from a better communication protocol with the wind sensor.

The script also needed the temperature in order to execute its logic. For demonstration purposes, the Raspberry Pi script fetched the temperature from a website using an API. This was the ultimate reason to switch to Raspberry Pi from Arduino

since the Teensy 3.2 does not have WiFi capabilities. A more complex code to fetch the indoor temperature of the house was attempted but a lag in driver software ultimately created a problem outside the scope of this project. The script "java_test.py" in the appendix documents this attempt for when GeckoDriver becomes available for the correct version of Firefox.

F. Receiver - Julia

In order to quickly implement this project, a finished receiver 12 was ordered for the physical build. Simultaneously, a receiver module was designed and laid out for future use.

Fig. 12. Receiver used for physical build

The schematic of the receiver module may be seen in Figure 13. A larger view may be seen in the appendix.

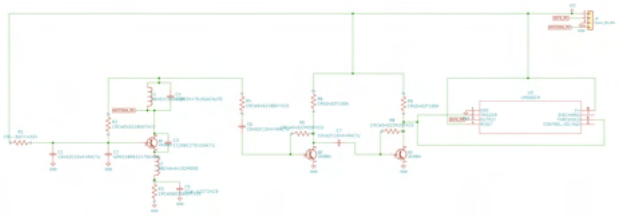


Fig. 13. Schematic of receiver module

In the first stage of the layout, the front-end regenerative receiver uses positive feedback to increase the gain of the incoming signal. L1 and C4 form the LC tank that is tuned to 433 MHz. The two stages in the middle are designed to differentiate between a signal at the desired frequency and noise. Lastly, the 555 timer compares the input voltage and will output high or low to indicate the presence of incoming data.

G. Transmitter - Julia

The transmitter pictured in Figure 14 was purchased with the receiver in order to speed the project timeline.

Fig. 14. Transmitter used for physical build

The schematic of the receiver module may be seen in Figure 15. A larger view may be seen in the appendix.

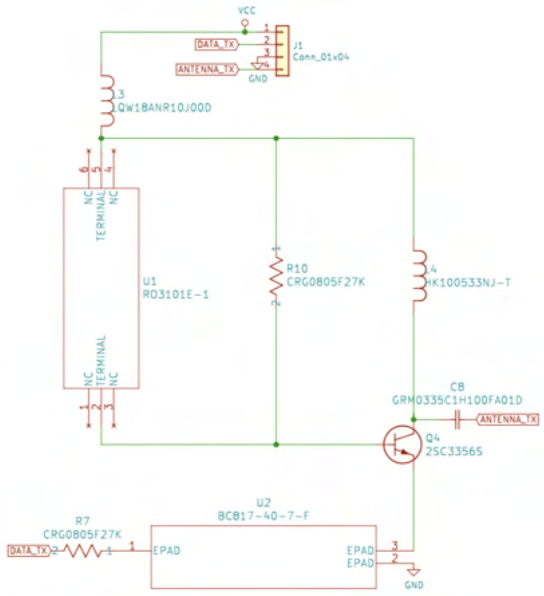


Fig. 15. Schematic of transmitter module

The data to be transmitted enters as a pulse and turns U2 (an NPN) on and off. The SAW resonator provides a 433MHz wave at Q4. Current flows when both are on and scales to create a wave at the antenna output over C8.

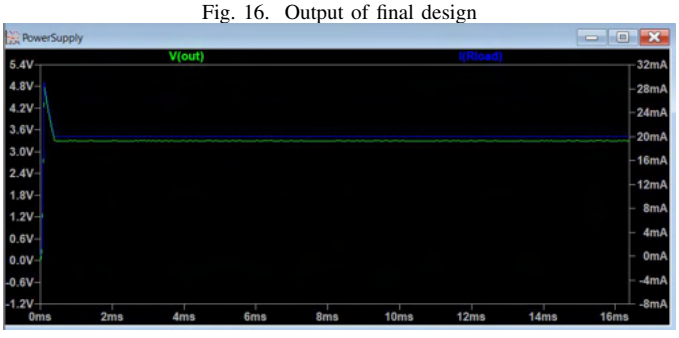
Adding an antenna to both transmitter and receiver significantly help with communication success and increasing range. A quarter-wave dipole at 433MHz should be about 17cm long [1].

III. Results

A. Power Supply

The power supply ended up working mostly as expected. The output voltage and current did meet the constraints previously mentioned, with the output current at 20 mA and the output voltage at 3.3 V with a ripple of +/- 4 millivolts.

Fig. 16. Output of final design



There were two unexpected behaviors: the startup behavior and the overall efficiency. While some noise can be expected from the series resistance of the capacitor, I didn't expect there to be such a large spike in the first millisecond of

operation. This is likely because the 9V battery is so close to the minimum input voltage of the IC (8 V). This could be corrected with a soft-start circuit or with the selection of a different control IC (like the TPS62175 which is specifically designed for light loads like this one).

The second of the unexpected behavior was the low efficiency of the system. When being powered by the battery the system was only about 60 percent efficient, which is rather low for this topology. As mentioned previously in the circuit design section, this is likely due to the low power load. Nonetheless, it only dissipates about 55 milliwatts of power. Since the typical 9V battery has about 5 watt-hours when discharging 10mA, a typical 9V battery could theoretically power the Buck Converter for more than the proposed two day-long power outage. The final design draws about 8mA of current from the 9V battery, so this could work.

However, as batteries discharge their output voltage drops over time, which would be another factor to take into account. Batteries are considered to be 'dead' when they drop below 50 percent of their original output voltage. As currently designed, our system is constantly checking the current wind and temperature conditions. The IC needs 8 Volts to function, and with similar applications, the 9V battery would likely drop below this at around 10 hours. The soft-start circuit could help with this, as it lowers the current the IC initially draws from the 9V battery. This would prolong the life of the battery when compared to the original circuit.

Generally the circuit functions as intended, although there are a few ways in which the circuit could be modified for a longer lasting, more efficient system.

B. Power Supply Schematic and PCB Layout

Doing the power supply schematic was a tedious but useful exercise. Tracking down the different footprints and 3D models for the different parts was frustrating at times but will be invaluable experience in industry.

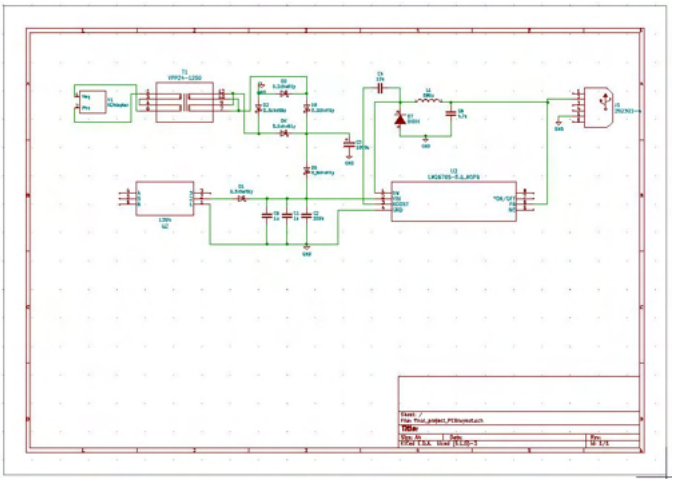


Fig. 17. Schematic of the Power Supply from KiCad

Note that in the schematic a custom footprint was used for the AC Power Entry Connector (6160.0141), but for demonstrative purposes a generic DC Barrel Jack was used in the 3D rendering.

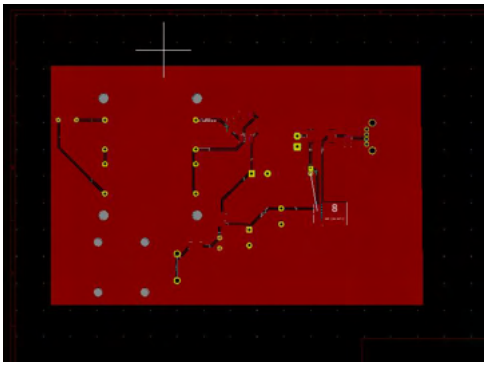


Fig. 18. PCB Layout

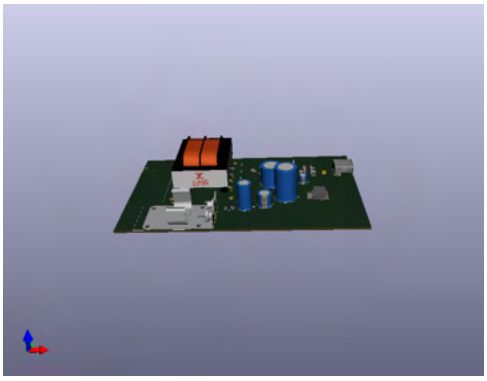


Fig. 19. 3D Rendering

C. Temperature Sensor

First, the model of the circuit can be evaluated by hand and in LTSpice. Figure 20 shows the output voltage over temperature by both LTSpice simulation and hand calculation. The LTSpice model is about 0.7V lower, although both predicted about the same change in voltage per degree Celsius.

In order to measure the temperature of the die, the device breakout board included a flat chip temperature sensor connected between two pins, as seen in Figure 21. The resistance of the chip was related to the temperature by the equation

$$R = 100(1 + AT + BT^2)$$

The Peltier module was heated until the temperature sensor reached the specific resistances corresponding to each temperature. Then the output voltage was measured and recorded. The plot in Figure 22 compares the measured results to the LTSpice simulation. The linear regression of the measured data has an R^2 value of .9985, meaning the output voltage of the temperature sensor is almost perfectly linear. The slope of trend line is 1.7 V per 100 °C.

However, there is a clear, curving trend in the variation between the linear trend and the data points as can be seen in Figure 23. This may be due to imperfections in the matching of I_S of the transistors in parallel.

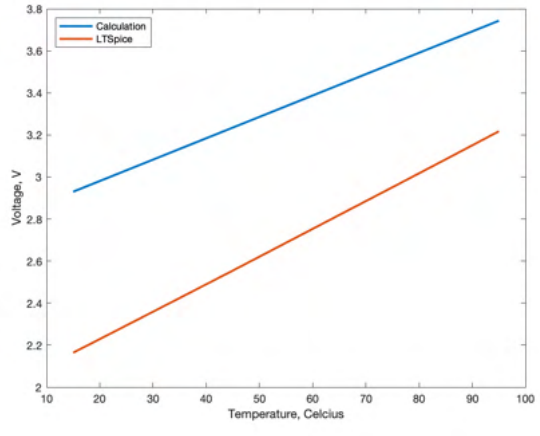


Fig. 20. Output voltage over temperature by hand calculation and LTSpice simulation

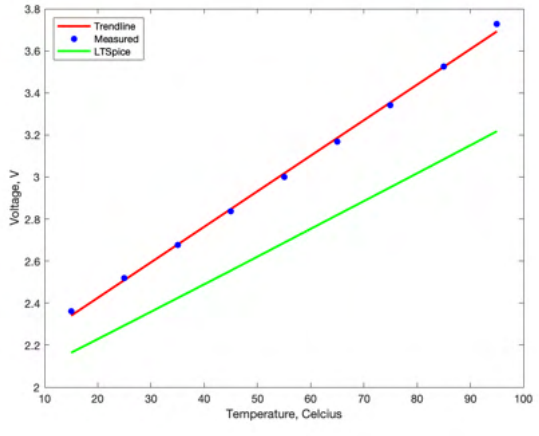


Fig. 22. Measured output voltage by temperature, as compared to LTSpice results

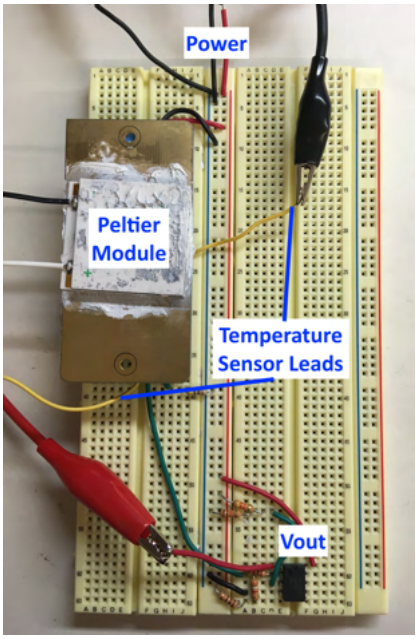


Fig. 21. Experimental setup, including the temperature sensor connections

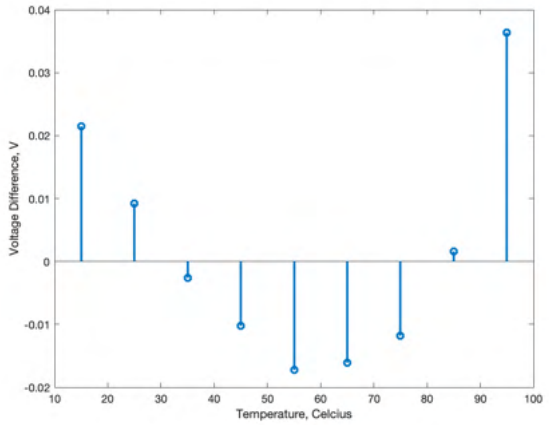


Fig. 23. Variation in measured output voltage by temperature

There were also noticeable differences between the calculated, simulated, and measured output voltages. This could be because of imperfections in the breadboard. It was difficult to maintain proper ground and power due to the larger amount of current flowing through the rails of the board, which then acted as resistors. Due to this phenomenon, measured ground was tens to hundreds of microvolts above 0V and 5V was a few millivolts below the theoretical 5V as well.

One design change that helped improve the performance of the circuit on the bench was to upgrade from the LM741 to the LM6132. The LM741 had non-ideal behavior when it railed at 1.3V and 4.2V and maintained a voltage of 1.8V at the inverting input, regardless of the bias voltage at the non-inverting input. The switch to the LM6132 widened the rails and allowed the voltage at the inverting input to be set to 1V.

Despite these challenges, the PTAT is a useful choice for the purpose of this project. A PTAT temperature sensor is fairly predictable because the output depends on the ratios of the currents of the transistors together rather than their individual exact values. However, the thermal response of the PTAT circuit needs to be calibrated to achieve very high accuracy each time a new chip is made due to variations in production processes [4]. After including a more robust op-amp, this circuit is dependable and perfectly suited for the application. The designed sensor achieves a 1.7V voltage swing per 100°C temperature change and operates from 15°C to 95°C with excellent linearity and precision.

D. Wind Sensor

After building the wind sensor, several observations can be gathered. First, the wind sensor was working successfully in that the change in voltage could be observed as expected whenever the magnet passed by the Hall sensor. This signal was also capable of reaching the receiver and could be observed

there. Several improvements could be made. First, it must next be made weather proof by placing the in a Tupperware or other housing so that it may work well outside, which is an important thing to consider when designing anything for physical implementation. Second, it was discovered the Hall sensors may be broken by a voltage that is too high or by static electricity (thus the use of an ESD bag). Caution should be used when handling sensitive devices. Finally, further design would be helpful to improve the communication protocol of this device. The operating environment turned out to be fairly noisy which made it easy for the un-tagged pulses to be lost.

E. Receiver

The final board can be seen in Figure 24. The receiver was useful both for recording the signal from the original remote and for receiving the signal for the wind sensor.

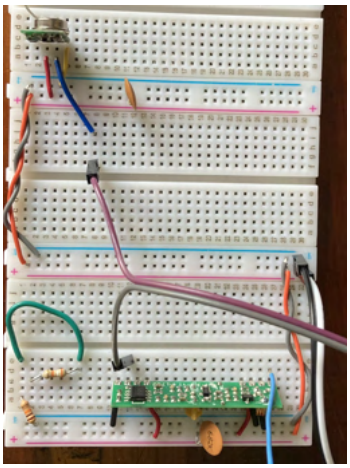


Fig. 24. Implementation of receiver with Raspberry Pi

In Figure 25, one of the eight pulses issued by a remote is shown. The signal was converted into zeros (short high value, long low value) and ones (long high value, short low value). The total signal consists of four of one pattern and then four of another pattern for a total of 8 signals like the figure.

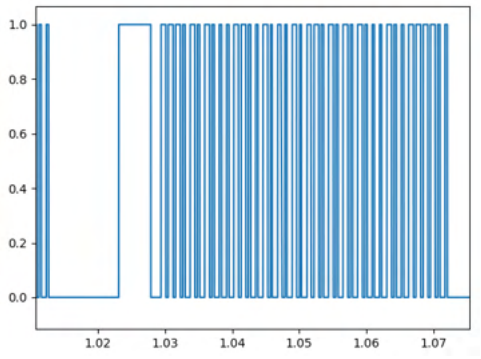


Fig. 25. Measured signal on receiver from remote

This method of decoding the original signal was very useful, but as previously discussed the communication protocol with the wind sensor could be improved for future work. One idea is to make the protocol similar to that of the remotes.

F. Transmitter

The final board can be seen in Figure 26. The transmitter was useful for sending a signal from the wind sensor and to the blinds on command.

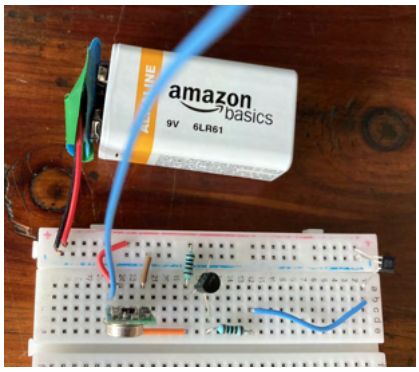


Fig. 26. Implementation of transmitter with wind sensor

In Figure 27, the signal sent from the transmitter to the blinds is shown. The Raspberry Pi was well suited for the frequency needed and was very reliable sans coding errors despite general noise from other sources.

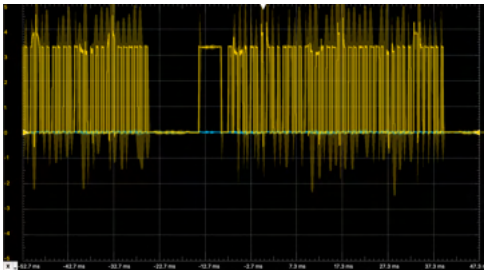


Fig. 27. Physical transmitter output signal

For future work, more study on the exact voltage to range relationship could be looked at in order ton understand how much power is required to overcome noise.

As for the simulation of the transmitter, the results turned out well. It was very interesting to find a working method for the SAW resonator. The output of the transmitter can be found in Figure 28 of a high data input followed by a low input.

G. PCB Development - Julia

This section demonstrates the development process for the transmitter and receiver. The layout is shown in Figure 29.

Figure 30 shows the the 3D render of the board design. In some cases it may be useful for the board pins to be on the bottom, but they have been designed to connect with jumper wires to a Raspberry Pi in this case.

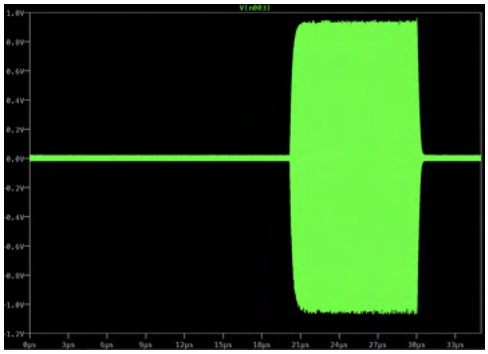


Fig. 28. Simulated transmitter output signal

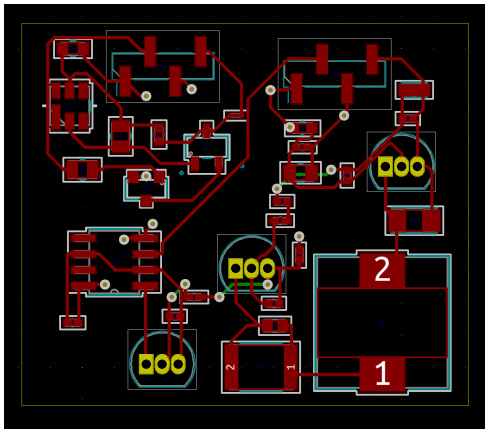


Fig. 29. PCB transmitter and receiver board layout

IV. Conclusion

Overall, this system is a promising start at a robust system to automate passive houses and improve environmental sustainability. The greatest improvements to be made for future work are a more robust communication protocol and the addition of more sensors. The transmitter for the wind sensor did not have a particular method of identifying itself to the receiver which made it tricky to discern the signal from other noise. Building the temperature sensor and adding a module to measure direct sunlight would be excellent improvements to the accuracy of the system's logic and decision making.

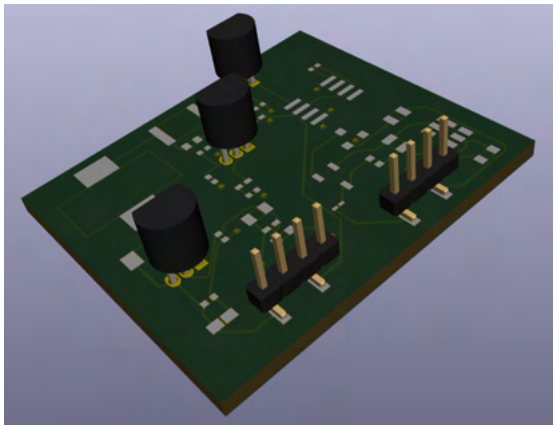


Fig. 30. 3D render of transmitter and receiver PCB

References

[1] Electro-tech online, *Them pesky cheapo 433Mhz transmitters!*, https://www.electro-tech-online.com/threads/them-pesky-cheapo-433mhz-transmitters.143245/, 2014.

[2] Instructables, *Super Simple Raspberry Pi 433MHz Home Automation*, "https://www.instructables.com/id/Super-Simple-Raspberry-Pi-433MHz-Home-Automation/", n.d.

[3] Kent H. Lundberg, *Become One with the Transistor*, Unpublished Preprint, 2005.

[4] Dexin Kong and Fengui Yu, *An auto-calibration technique for BJT-based CMOS temperature sensors*, IEICE Electronics Express, 2017.

[5] Ray's Hobby Net, *Reverse Engineer Wireless Temperature / Humidity / Rain Sensors*, "https://rayshobby.net/reverse-engineer-wireless-temperature-humidity-rain-sensors-part-1/", 2014.

[6] Everett Rogers, *Understanding Buck Power Stages in Switchmode Power Supplies*, "http://www.ti.com/lit/an/slva057/slva057.pdf?&ts1589333654796", 1999.

[7] Jens Ejury *Buck Converter Design* "https://www.mouser.de/pdfdocs/BuckConverterDesignNote.pdf", 2013.

APPENDIX

A. ReceiveRF.py

```
from datetime import datetime
import matplotlib.pyplot as pyplot
import RPi.GPIO as GPIO

RECEIVED_SIGNAL = [[], []]  #[[time of reading], [signal reading]]
MAX_DURATION = 5
RECEIVE_PIN = 23

if __name__ == '__main__':
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(RECEIVE_PIN, GPIO.IN)
    cumulative_time = 0
    beginning_time = datetime.now()
    print('**Started recording**')
    while cumulative_time < MAX_DURATION:
        time_delta = datetime.now() - beginning_time
        RECEIVED_SIGNAL[0].append(time_delta)
        RECEIVED_SIGNAL[1].append(GPIO.input(RECEIVE_PIN))
        cumulative_time = time_delta.seconds
    print('**Ended recording**')
    print(len(RECEIVED_SIGNAL[0]), 'samples recorded')
    GPIO.cleanup()

    print('**Processing results**')
    for i in range(len(RECEIVED_SIGNAL[0])):
        RECEIVED_SIGNAL[0][i] = RECEIVED_SIGNAL[0][i].seconds + RECEIVED_SIGNAL[0][i].microseconds/1000000.0

    print('**Plotting results**')
    pyplot.plot(RECEIVED_SIGNAL[0], RECEIVED_SIGNAL[1])
    pyplot.axis([0, MAX_DURATION, -1, 2])
    pyplot.show()
```

B. PHEBCS.py

```
import time
import sys
from datetime import datetime
import matplotlib.pyplot as pyplot
import RPi.GPIO as GPIO
#from dot3k import lcd
import requests

RECEIVE_PIN = 23
TRANSMIT_PIN = 24

blind_state = 1 #0 = closed, 1 = open
zero_count = 0
NUM_ATTEMPTS = 4

down1 = '11101010001100101010110101010000100110011'
down2 = '11101010001100101010110101010000100111100'
up1 =   '11101010001100101010110101010000100010001'
up2 =   '11101010001100101010110101010000100011110'
dad_down1 = '11101001001100101011001010110000000110011'
dad_down2 = '11101001001100101011001010110000000111100'
dad_up1 = '11101001001100101011001010110000000010001'
dad_up2 = '11101001001100101011001010110000000011110'
stop =  '11101010001100101010110101010000101010101'
short_delay = 0.00036
long_delay = 0.00070
extended_zero = 0.01043
extended_one = 0.00478
gap_zero = 0.00154

GPIO.setmode(GPIO.BCM)
GPIO.setup(TRANSMIT_PIN, GPIO.OUT)
GPIO.setup(RECEIVE_PIN, GPIO.IN)

###RECEIVE CODE
def wind():
    global zero_count
    incoming = GPIO.input(RECEIVE_PIN)
    if incoming == 0:
        zero_count = zero_count + 1
    else:
        zero_count = 0
    if zero_count > 10:
        change = datetime.now()-last_time
        if change < 1:
            blinds(1) #open the blinds due to high wind
            #lcd.write('High windspeeds')
        last_time = datetime.now()
    return

###GET TEMPERATURE
def temp():

    r = requests.get('http://api.openweathermap.org/data/2.5/weather?zip=45385,us&appid=a818dce96a521dcfddbd80799d426148')
    r = r.json()
    temp = r["main"]["temp"]
    temp = 1.8*(temp-273.15)+32 #in farenheit

    if temp > 80:
        blinds(0) #close blinds
    if temp < 65:
        blinds(1) #open blinds
    return temp

def transmit(code1, code2):
    '''Transmit a chosen code string using the GPIO transmitter'''
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(TRANSMIT_PIN, GPIO.OUT)
    for t in range(NUM_ATTEMPTS):
        for i in code1:
            if i == '0':
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(short_delay)
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(long_delay)
            elif i == '1':
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(long_delay)
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(short_delay)
            else:
                continue
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(extended_zero)
        GPIO.output(TRANSMIT_PIN, 1)
        time.sleep(extended_one)
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(gap_zero)
    for t in range(NUM_ATTEMPTS):
        for i in code2:
            if i == '0':
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(short_delay)
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(long_delay)
            elif i == '1':
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(long_delay)
```

```
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(short_delay)
            else:
                continue
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(extended_zero)
        GPIO.output(TRANSMIT_PIN, 1)
        time.sleep(extended_one)
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(gap_zero)
    GPIO.cleanup

def blinds(command):
    global blind_state
    if command != blind_state:
        if command == 1:
            lcd.write('Opening blinds...')
            transmit(dad_up1, dad_up2)
        else:
            lcd.write('Closing blinds...')
            transmit(dad_down1,dad_down2)
        blind_state = command
    return

#infinite loop
while True:
    wind()
    temp = temp()
    print(temp)
    #lcd.write("Inside temperature is" + t_in)
    #lcd.write("Outisde temperature is" + t_out)
    #blind status
    #if blind_state == 1:
    #    lcd.write('Blinds are open')
    #else:
    #    lcd.write('Blinds are closed')
    GPIO.output(TRANSMIT_PIN, 0)
    time.sleep(120)
```

C. *java_test.py*

```
import time
import sys
from datetime import datetime
import matplotlib.pyplot as pyplot
import RPi.GPIO as GPIO
from selenium import webdriver
import selenium as se
from webdriver_manager.chrome import
    ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import
    WebDriverWait
from selenium.webdriver.support import
    expected_conditions as EC

###GET TEMPERATURE
def temp():

    test = """
    function wait(ms)
    {
        var d = new Date();
        var d2 = null;
        do { d2 = new Date(); }
        while(d2-d < ms);
    }

    document.querySelector("#login_email").value = "
    greg.arnold32@gmail.com"
    document.querySelector("#login_password").value
    = "yynn343C"
    document.querySelector("#loginBtn").click()
    """

    test2 = """
    return document.querySelectorAll(".
    sbListIconDetail")[0].children[1].innerText
    """

    test3 = """
    return document.querySelectorAll(".
    sbListIconDetail")[1].children[1].innerText
    """

    myscript = test
    options = se.webdriver.ChromeOptions()
    options.add_argument('headless')

    driver = webdriver.Chrome(ChromeDriverManager().
    install())
    driver.get("https://www.buildequinox.com/cervice
    /")
    result = driver.execute_script(myscript)

    wait = WebDriverWait(driver, 5)
    element = wait.until(EC.element_to_be_clickable
    ((By.CLASS_NAME, 'sbListIconDetail')))
    t_in = driver.execute_script(test2)
    t_out = driver.execute_script(test3)
    driver.quit()

    if t_in > 73:
        blinds(0) #close blinds
    if t_in < 68:
        blinds(1) #open blinds
    return t_in, t_out
```

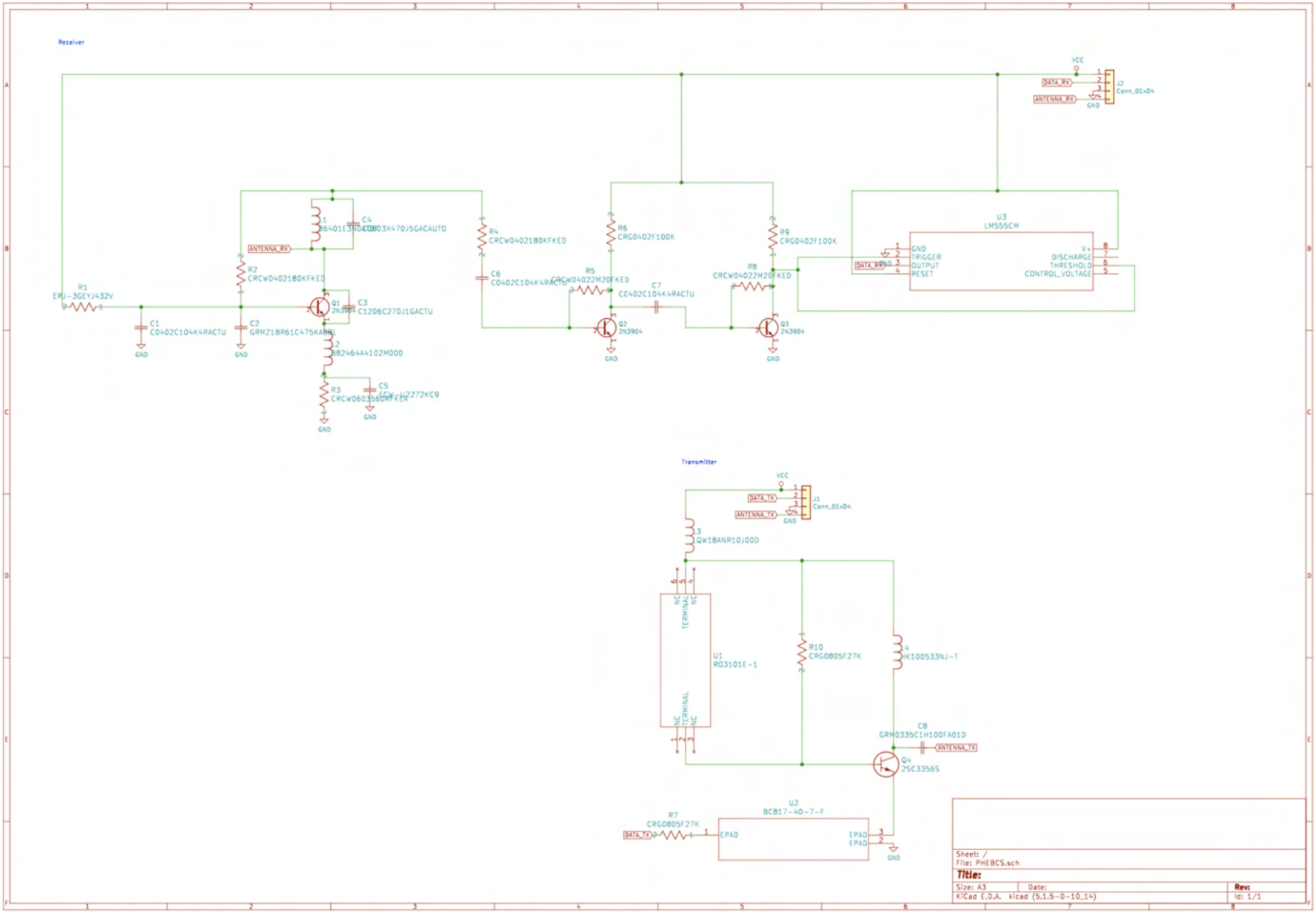
D. *TransmitRF.py*

```
import time
import sys
import RPi.GPIO as GPIO

down1 = '1110101000110010101011010101000100110011'
down2 = '1110101000110010101011010101000100111100'
up1 =   '1110101000110010101011010101000100010001'
up2 =   '1110101000110010101011010101000100011110'
dad_down1 = '
    1110100100110010101100101011000000110011'
dad_down2 = '
    1110100100110010101100101011000000111100'
dad_up1 = '1110100100110010101100101011000000010001'
dad_up2 = '1110100100110010101100101011000000011110'

stop =  '1110101000110010101011010101000101010101'
short_delay = 0.00036
long_delay = 0.00070
extended_zero = 0.01043
extended_one = 0.00478
gap_zero = 0.00154

NUM_ATTEMPTS = 4
TRANSMIT_PIN = 24

def transmit_code(code1, code2):
    '''Transmit a chosen code string using the GPIO
    transmitter'''
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(TRANSMIT_PIN, GPIO.OUT)
    for t in range(NUM_ATTEMPTS):
        for i in code1:
            if i == '0':
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(short_delay)
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(long_delay)
            elif i == '1':
```

```
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(long_delay)
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(short_delay)
            else:
                continue
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(extended_zero)
        GPIO.output(TRANSMIT_PIN, 1)
        time.sleep(extended_one)
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(gap_zero)
    for t in range(NUM_ATTEMPTS):
        for i in code2:
            if i == '0':
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(short_delay)
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(long_delay)
            elif i == '1':
                GPIO.output(TRANSMIT_PIN, 1)
                time.sleep(long_delay)
                GPIO.output(TRANSMIT_PIN, 0)
                time.sleep(short_delay)
            else:
                continue
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(extended_zero)
        GPIO.output(TRANSMIT_PIN, 1)
        time.sleep(extended_one)
        GPIO.output(TRANSMIT_PIN, 0)
        time.sleep(gap_zero)
    GPIO.cleanup()

transmit_code(dad_down1, dad_down2)
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRANSMIT_PIN, GPIO.OUT)
GPIO.output(TRANSMIT_PIN, 0)
time.sleep(20)
GPIO.cleanup()
transmit_code(dad_up1, dad_up2)
GPIO.setup(TRANSMIT_PIN, GPIO.OUT)
GPIO.output(TRANSMIT_PIN, 0)
time.sleep(10)
print("clean up")
GPIO.cleanup() # cleanup all GPIO
```

E. Receiver and Transmitter Schematic

The schematic is available on the following page.


Receiver
VCC
DATA_RX
ANTENNA_RX
GND
J2
Conn_01x04
U3
LM555CM
GND
TRIGGER
OUTPUT
RESET
V+
DISCHARGE
THRESHOLD
CONTROL_VOLTAGE
R1
ERJ-3GEYJ432V
R2
CRCW0402180KFKED
L1
C4
C1
C0402C104K4RACTU
C2
GRM21BR61C475KA88L
Q1
2N3904
C3
C1206C270J1GACTU
BB2464A4102M000
R3
C5
R4
CRCW0402180KFKED
C6
C0402C104K4RACTU
R5
CRCW04022M20FKED
R6
CRG0402F100K
Q2
2N3904
C7
C0402C104K4RACTU
R8
CRCW04022M20FKED
R9
CRG0402F100K
Q3
2N3904
Transmitter
VCC
DATA_TX
ANTENNA_TX
GND
J1
Conn_01x04
L3
QW1BANR10J00D
U1
RO3101E-1
NC
TERMINAL
R10
CRG0805F27K
L4
HK100533NJ-T
C8
GRM0335C1H100FA01D
Q4
2SC3356S
U2
BC817-40-7-F
EPAD
R7
CRG0805F27K
Sheet: /
File: PHEBCS.sch
Title:
Size: A3
Date:
Rev:
KiCad E.D.A. kicad (5.1.5-0-10_14)
Id: 1/1