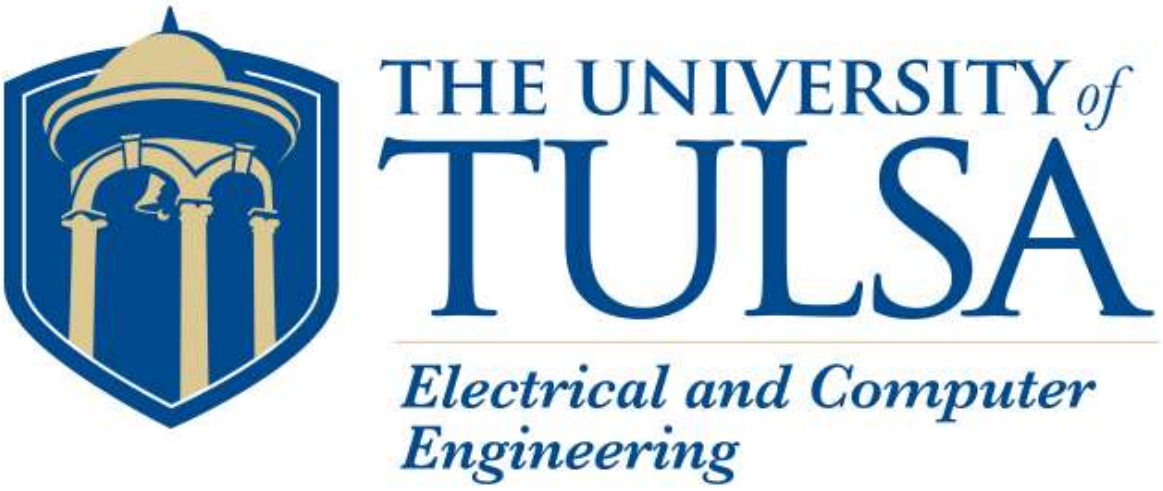

Independent Study (IS) Project: Ambient Light Energy Harvesting

IS Project Report

Ifeoma Ogwu

August 1, 2025

Contents

List Of Figures

Listings

Abstract

In this project, its goal is to explore energy harvesting through, primarily, ambient light harvesting and see how much, if any, this type of harvesting can extend battery life while it powers a load. The load in question is an Arduino pro mini connected to peripheral modules. The device wakes itself periodically, reads temperature, humidity, and pressure from a BME280, reads surrounding light intensity of system using BH1750FVI, logs the time from a DS3231 real-time clock, and measures its own battery voltage and charge percent with a MAX17048 fuel gauge. It also samples its current draw with a MAX471 sensor and saves all data to an SD card. To save power between readings, the microcontroller and sensors enter deep sleep, and the fuel gauge's hibernation features were used.

The results suggest that a custom energy harvester and/or a supercapacitor as a store of energy instead of a lithium-ion battery would better match the system's load. This work demonstrates the importance of matching harvester output to device load and highlights strategies for stretching battery life in low-power applications.

Chapter 1

Introduction

1.1 Introduction

Ambient energy harvesting is the process of capturing small amounts of energy from the surrounding environment and converting it into electrical power. Sources of ambient energy include light (solar), heat including body heat (thermoelectric), motion or vibration (piezoelectric), and radio waves and electromagnetic sources (RF). A harvester uses specialized materials or circuits, such as solar cells more tuned for indoor light levels for light or Peltier devices for heat, to turn these energy forms into usable voltage and current **(Figure 1.1)**.

Ambient harvesters can run indefinitely as long as their energy source remains available. In low-power sensing applications, this technique allows devices to recharge themselves from everyday surroundings, reducing maintenance and enabling truly off-grid operation.

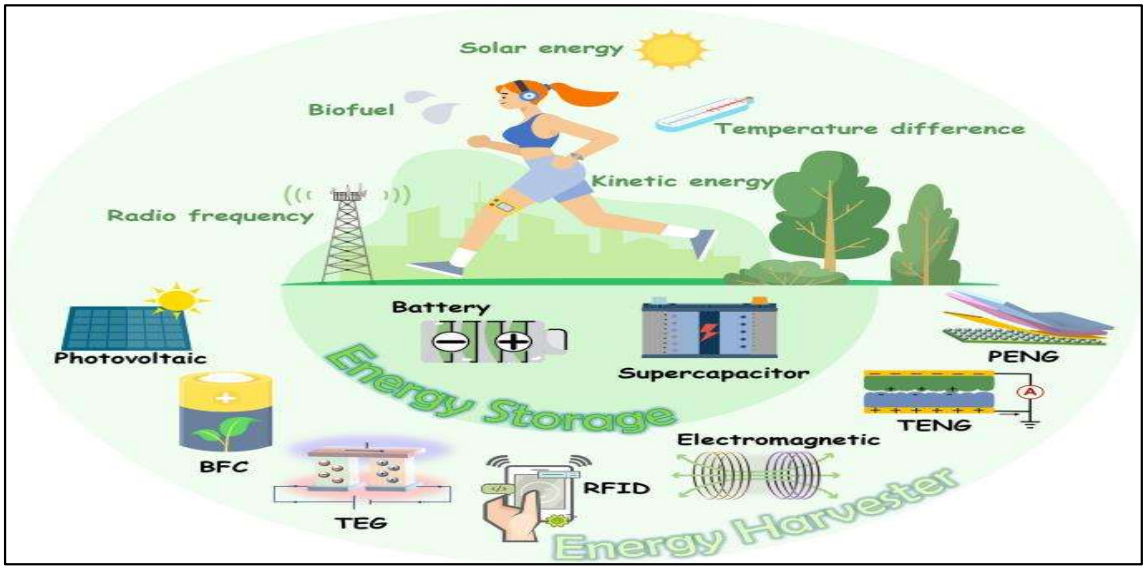

Figure 1.1: Ambient Energy Harvesting Sources

Chapter 2

Hardware Design

2.1 Mechanical Design

The mechanical design of this project is centered around the Arduino Pro Mini 3.3V / 8MHz board and an energy harvester for the solar cells used. The Arduino is also connected to BME280 sensor, DS3231 RTC, Adafruit MAX17048 fuel-gauge, MAX471 sensor, BH1750FVI sensor using I2C and a SD card module using SPI. The Arduino with its peripherals were powered using both USB and an 18650 3.7V lithium-ion battery during the experiment. The solar cells used were the LL200-2.4-75 solar cells from PowerFilm.

2.2 Electronic System Design

The electronic system design for this project centers around the Arduino Pro Mini microcontroller, which handles all processing tasks. The board is programmed through a USB connection to a computer, and powered through the same USB during the first phase of the project and powered through battery during the second phase of the project. The additional external components and sensors involved have been mentioned above.

2.3 Why Did I Power The System Through USB At All?

You may wonder why I powered the Arduino through USB at all considering the main scope of the project was focused on battery life and power. The reason was, earlier in testing, I had issues with code and the SD card not being written to when powered by battery. Due to this,

I thought it best to test the Arduino separately over a week and use those results to set up tests for the battery and solar cells. However, after the USB test, I adjusted the code more and ran a few more tests and was able to have it run properly on battery so I also tested with battery. The USB test also helps give a picture of the voltage draw of the system.

The image **(Figure 2.1)** below shows the system setup when run on battery (excluding the BH1750FVI for the computer setup).

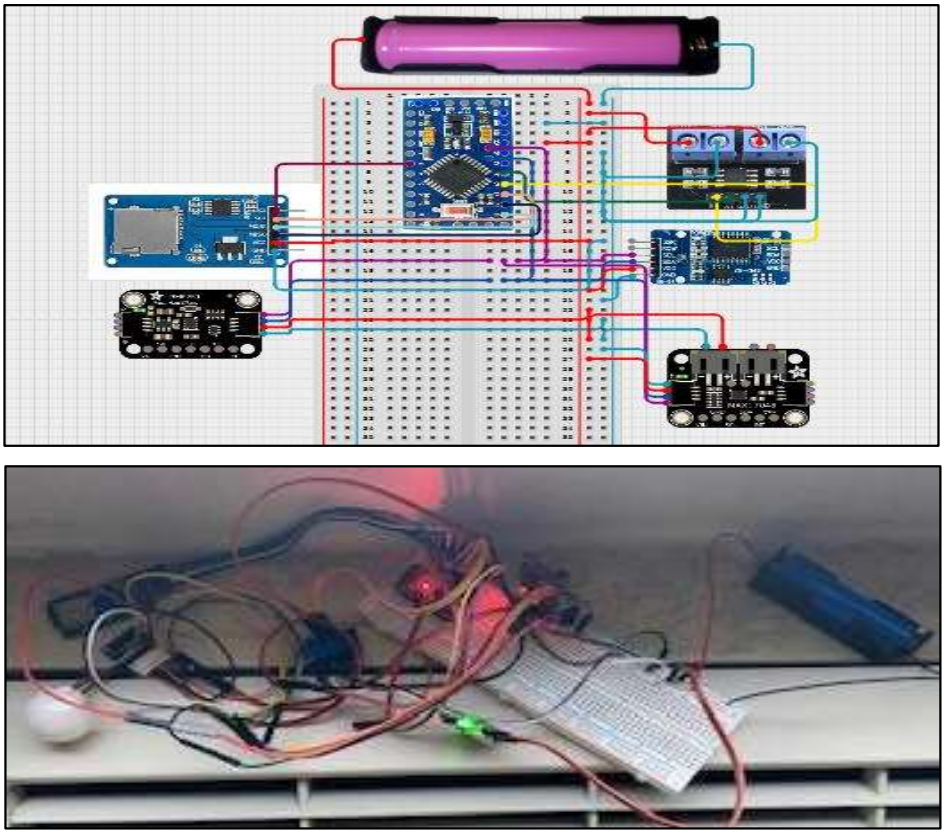

Figure 2.1: Setup Of Project When Run On Battery

The connections for each device are summarized in the table below **(Listing 2.1)**.

Peripheral Pin	Arduino Pin
BME280:	
VCC	VCC
GND	GND
SDA	SDA (A4)
SCL	SCL (A5)
BH1750FVI:	
VCC	VCC
GND	GND
DAT	SDA (A4)
SCL	SCL (A5)
DS3231:	
VCC	VCC
GND	GND
SDA	SDA (A4)
SCL	SCL (A5)
MAX471:	
VIN	VCC
VOUT	VCC
GND	GND
AT	A1
VT	A0 (when connected to USB)
MAX17048:	
VIN	VCC
GND	GND
SDA	SDA (A4)
SCL	SCL (A5)
BATT[+] (Also JST[+])	VCC
BATT[-] (Also JST[-])	GND
SD Card Reader:	
VCC	VCC
GND	GND
MISO	D12
MOSI	D11
SCK	D13
CS	D3
18650 Battery:	
BATT[+]	VCC
BATT[-]	GND

Chapter 3

Software Design

3.1 Code Description

The sketch begins in ”setup()” by powering down the ADC to save roughly 200 µA, starting I²C, and bringing up each sensor in turn. First, the BME280 environmental sensor is initialized to measure temperature, pressure, and humidity. Next, the DS3231 real-time clock is brought online and set to the compile date/time if it has lost power. The MAX17048 fuel gauge is then started to report precise battery voltage, state-of-charge, and charge/discharge rate. Finally, the BH1750 light sensor is initialized to read ambient illumination in lux. Once all the peripherals respond, the SD card interface is activated; if no log.csv file exists, a new CSV with the proper header line is created.

In the “loop()”, the microcontroller enters an 8-second deep sleep repeatedly until the real-time clock reports that at least half an hour has elapsed. Upon waking for a read cycle, the code first samples the BME280, converting pressure into millimeters of mercury. It then queries the DS3231 for the exact date and 24-hour time, formatting both into fixed-width strings. Next, the two analog pins attached to the MAX471 module are read: one through the onboard 5× voltage divider to measure voltage draw (when powered with USB), and one through the amplifier output to calculate load current. After that, the MAX17048 is polled for its battery voltage, state-of-charge, and percent-per-hour charge rate, and the BH1750 returns ambient lux. Between each analog sample, the ADC is disabled again to minimize drain. Finally, the SPI bus and SD card are re-initialized, and all collected data, date, time, environmental readings, battery metrics, current draw, and light level, is appended as one line to log.csv. If the battery voltage falls below 3.2 V or the charge percentage dips under 10 %, the loop returns early to avoid further writes and protect the battery.

To summarize how the system works, it is shown in the flowchart below:

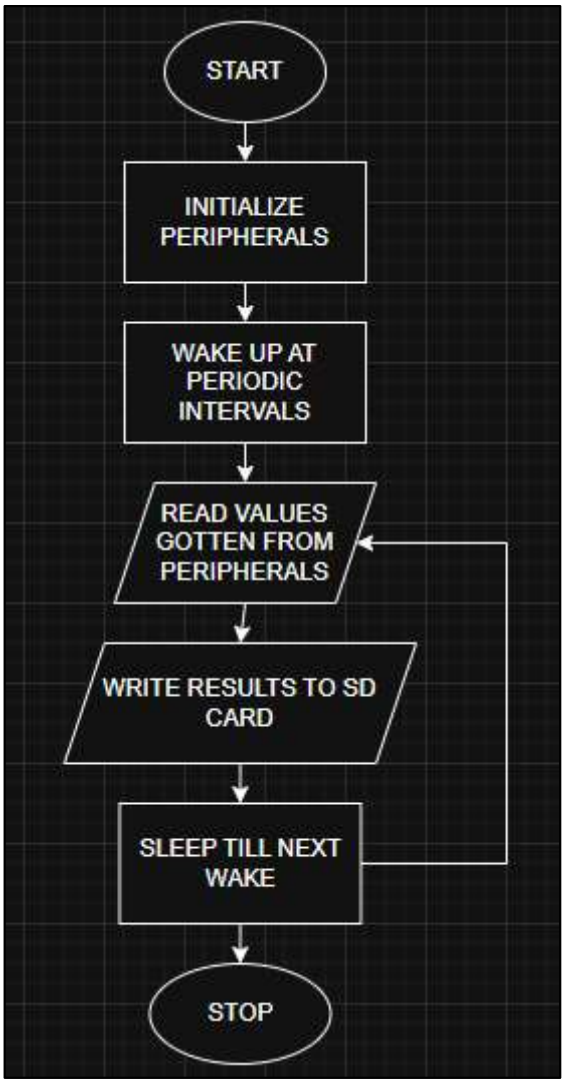

Figure 3.1: Code Flowchart

Chapter 4

Testing Requirements

The tests run for this project were:

- The Arduino and peripherals run through USB. This helps give a picture of voltage draw.
- The Arduino and peripherals run on battery. This helps give more on current draw and how long the battery would last on its own without energy harvesting.
- The solar cells open circuit voltage and short circuit current.
- The solar cells voltage and current when trying to connect through the energy harvester, AEM10941.

Chapter 5

Users Guide

5.1 How It Works

When powered, the data logger automatically goes to sleep to conserve energy. Every thirty minutes (when powered on battery) or hour (when powered on USB) it wakes itself, reads environmental data from the BME280 (temperature, pressure, humidity), checks the current time on the DS3231 clock, measures its own battery voltage and state-of-charge via the MAX17048, samples its load current through the MAX471, and takes a light reading from the BH1750 (when powered on battery). It then writes all of these values as one line in log.csv on the inserted SD card before returning to deep sleep. No buttons or displays are needed, the user simply powers the unit, lets it run, and later removes the SD card to view the CSV in a spreadsheet.

Chapter 6

Results

For reference, all measurements were taken with a multimeter.

For testing the solar cells on their own:

- The open circuit voltage ranged from 1.5V – 2.5V for one cell, depending on lighting conditions.
- The short circuit current was 142µA for one cell at a time of measurement.
- For four cells in series, the open circuit voltage was 7.33V, short circuit current was 133µA.

For testing the four solar cells in series when connected with the AEM10941 energy harvester, the battery and the Arduino, the measurements gotten when the HVOUT was connected to RAW on Arduino were:

- The SRC voltage (solar cells voltage) was 1.82V.
- The SRC current was 176.6µA.
- HVOUT was 489mV.
- Current from HVOUT was 106.7µA, with a max of 406.8µA which happens when the Arduino would attempt to start up.

The measurements gotten when the HVOUT was connected to VCC on Arduino were:

- The SRC voltage (solar cells voltage) was 2.02V.
- The SRC current was 188µA.
- HVOUT was 534.1mV.
- Current from HVOUT was 142µA, with a max of 484µA which happens when the Arduino would attempt to start up.

Before connecting the Arduino, the measurement gotten were:

- The SRC voltage (solar cells voltage) was 7.67V.
- The SRC current was 133µA.
- HVOUT was 3.32V.

All of the above measurements were made in an environment with an average of 145 lux.

When connecting the Arduino to battery alone, there was a max current draw of 12.44mA at a point in time.

The measurements suggest that the solar cells cannot sustain the current draw needed by the Arduino and thus there is voltage collapse. These results also suggest that the AEM10941 evaluation board has a default configuration whereby the load is mainly powered by the cells when there is sufficient light. This is not the configuration desired for this project as we want the load always mainly powered by the battery and the battery charged by the solar cells in turn.

For reference, this is how the parts were connected for the AEM10941 evaluation board:

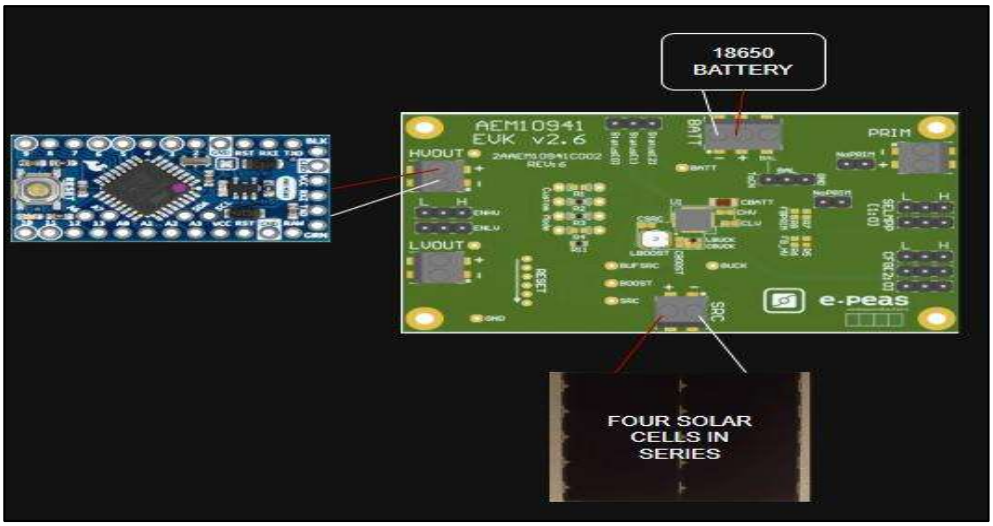

Figure 6.1: AEM10941 Evaluation Board With Connections

The results gotten from the csv file when the Arduino was connected to USB, over the course of a week, are summarized below:

- The average temperature recorded was 23.18 °C with a minimum of 21.43 °C and maximum of 26.05 °C.

- The average pressure recorded was 743.71 mmHg with a minimum of 741.03 mmHg and maximum of 746.29 mmHg.
- The average humidity recorded was 67.53% with a minimum of 52.11% and maximum of 85.14%.
- The average voltage draw recorded was 3.38V with a minimum of 3.32V and maximum of 3.79V.
- The average current draw recorded was 9.73 mA with a minimum of 8.92 mA and maximum of 9.85 mA.

The results gotten from the csv file when the Arduino was connected to battery are summarized below:

- The average temperature recorded was 25.08 °C with a minimum of 22.18 °C and maximum of 31.13 °C.
- The average pressure recorded was 742.51 mmHg with a minimum of 740.69 mmHg and maximum of 743.78 mmHg.
- The average humidity recorded was 58.87% with a minimum of 46.95% and maximum of 73.41%.
- The average discharge rate per hour of the battery recorded was 0.51% with a minimum of 5.62% and maximum of 0.42%.
- The average current draw recorded was 10.04 mA with a minimum of 9.92 mA and maximum of 12.05 mA.
- The average lamination intensity of the environment recorded was 65.02 lux with a minimum of 0.83 lux and maximum of 417.5 lux.

When run on USB, the system was placed on a desk, not near a window. When run on battery, it was placed under a window sill, on an air conditioning unit.

The fully charged battery lasted about 3.5 days before dying, even though it read about 80% and based on our average current draw of 10.04mA and the battery's rating of 1800mAh, the system should be able to go about 7 days before the battery dies. I suspect there is an unexpected event that occurs that draws a lot of current, draining the battery as this also occurred with a battery dying at 55%. This unexpected event could be coming from the battery or connections, as when the Arduino is powered through USB, there are no records of large current draws.

Chapter 7

Observations & Lessons Learned

Having noticed that the AEM10941 evaluation board was attempting to mainly feed the load from the solar cells instead of the battery, I brainstormed ways one could correct this.

I believe using a supercapacitor with a high current rating would be a possible counter solution and, in that setup, the Arduino would read all sensors and write to SD card when capacitor is charged enough to power it. However, this solution will likely still be held back by the default configuration of the AEM10941. Due to this, I believe a custom solution for an energy harvesting and charge management board might be best.

I designed a potential PCB that would help with the project. It is based on the BQ25570 nano power boost charger and buck converter that has a cold start voltage of $\geq$ 600 mV, has continuous energy harvesting from input voltage as low as 100 mV, input voltage regulation preventing collapsing high impedance input sources and full operating quiescent current of 488 nA.

Main features of the PCB:

- A relatively steady output voltage of 3.6V
- A set overcharge of 4.12V for a lithium-ion battery
- Ability to choose Maximum Power Point Tracking for solar cells through jumpers

The below images show the proposed schematic and 3D view of the PCB.

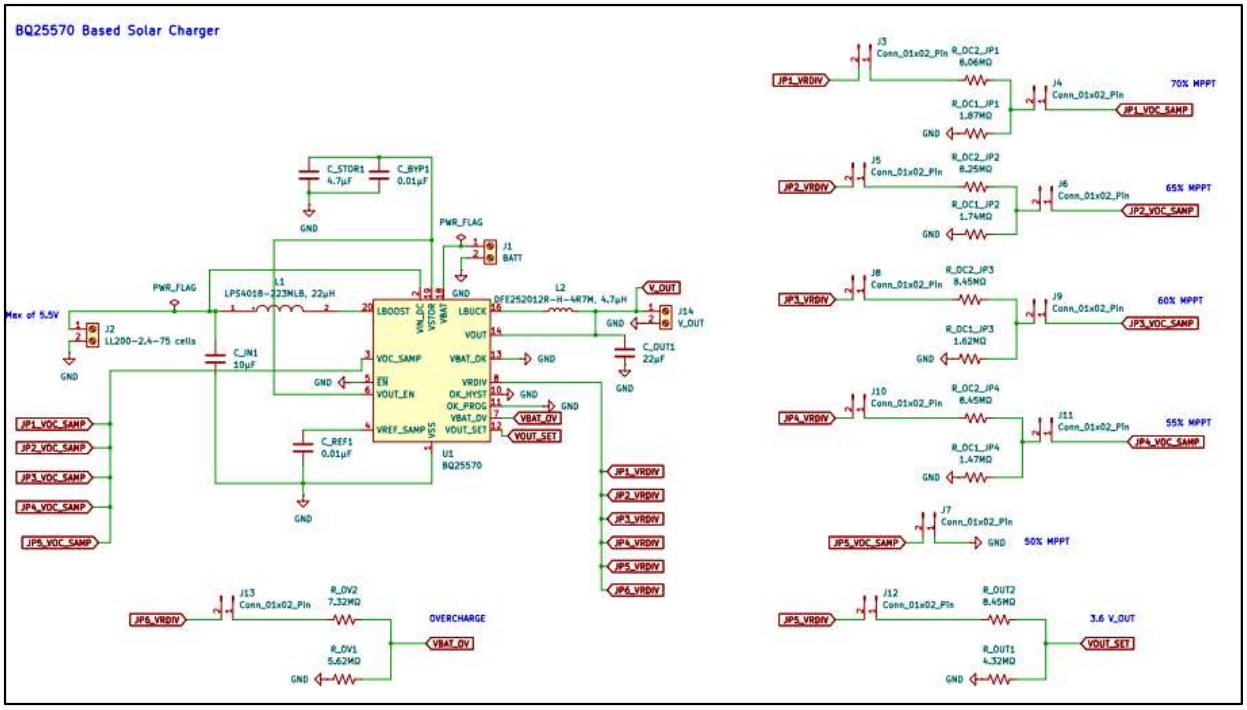

Figure 7.1: KiCAD Schematic Of Proposed PCB

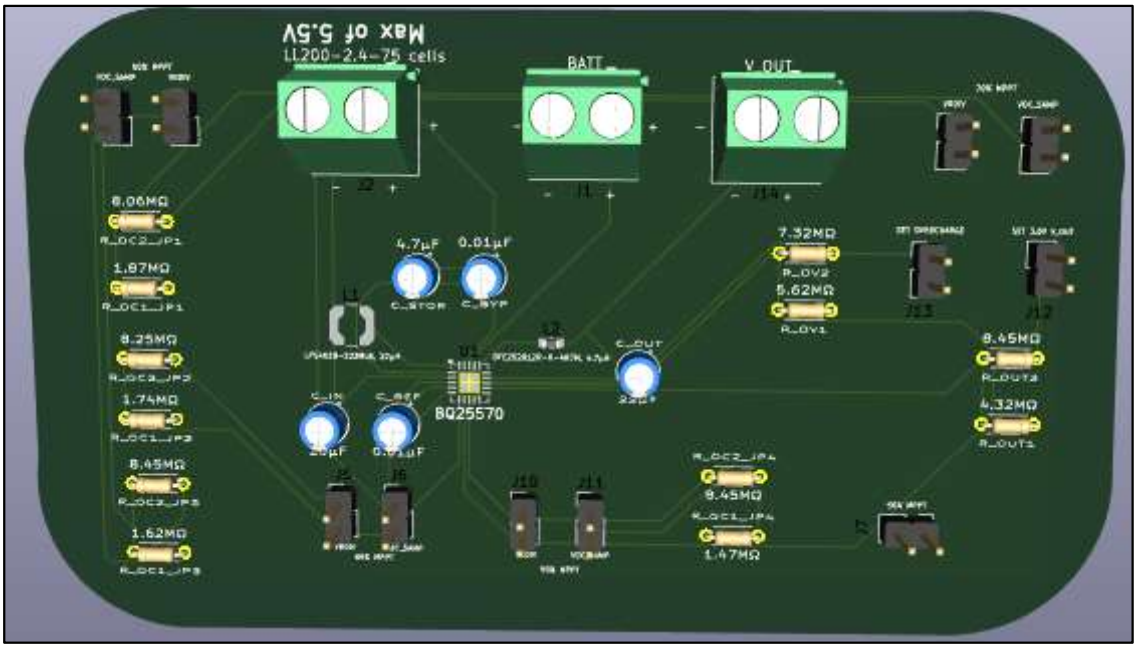

Figure 7.2: KiCAD 3D View Of Proposed PCB

Through this project, I have learnt a lot on energy harvesting and the challenges the technology faces in progressing. I hope to continue my research on personal time, even past this project, to learn more about the field and perhaps even contribute to it.

Appendix A

Source Code

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <Adafruit_BME280.h>
#include <Adafruit_MAX1704X.h>
#include <BH1750.h>
#include <RTClib.h>
#include <LowPower.h>

// I2C addresses
#define BME280_ADDR 0x77

// SD card chip-select
#define SD_CS_PIN   3

// MAX471 analog inputs
#define VT_PIN     A0    // module's VT output (5× divider)
#define IT_PIN     A1    // module's AT output (shunt amplifier)

// ADC & shunt parameters
const float VREF          = 3.3f;  // Pro Mini Vcc
const float VDIV_FACTOR   = 5.0f;  // MAX471 on-board divider
const float SENSE_V_PER_A = 0.1f;  // 100 mV per amp

// BME280, RTC, fuel-gauge and BH1750FVI instances
Adafruit_BME280    bme;
RTC_DS3231         rtc;
Adafruit_MAX17048  maxlipo;
BH1750 lightMeter;

// wake-tick
static uint16_t tick = 0;
```

```
static uint32_t lastLogTime = 0;

// hPa → mmHg
float hPa_to_mmHg(float h) { return h * 0.750062f; }

void setup() {
  Wire.begin();

  Serial.begin(57600);
  delay(10);
  Serial.println(F("Starting logger…"));

  if (!bme.begin(BME280_ADDR)) while (1);
  if (!rtc.begin())          while (1);
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  delay(200);

  // Init MAX17048 fuel gauge
  if (!maxlipo.begin()) while (1);

  // Init BH1750FVI
  lightMeter.begin();

  // SD init + header
  if (SD.begin(SD_CS_PIN)) {
    if (!SD.exists("log.csv")) {
      File f = SD.open("log.csv", FILE_WRITE);
      if (f) {
        f.println(F("Date,Time,Temp_C,Pressure_mmHg,Humidity_%,Batt_V,Batt_%,Char
geRate_%/hr,Load_mA,Lux_lx"));
        f.close();
      }
    }
  }

}

void loop() {
  // —— Sleep ~8 seconds at a time ——
  LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);

  DateTime now = rtc.now();
  if ((now.unixtime() - lastLogTime) < 1800) return; // Only log once 30 minutes
(1800 seconds)
  lastLogTime = now.unixtime();
```

```
// Read BME280
float tempC     = bme.readTemperature();
float pressMmHg = hPa_to_mmHg(bme.readPressure() / 100.0F);
float humidity  = bme.readHumidity();

// Read DS3231 RTC
char dateStr[11], timeStr[9];
snprintf(dateStr, sizeof(dateStr), "%02d-%02d-%04d",
         now.day(), now.month(), now.year());
snprintf(timeStr, sizeof(timeStr), "%02d:%02d:%02d",
         now.hour(), now.minute(), now.second());

// Read battery voltage
uint16_t rawV = analogRead(VT_PIN);
float vAtA0   = rawV * (VREF / 1023.0f);
float loadV   = vAtA0 * VDIV_FACTOR;
ADCSRA &= ~_BV(ADEN); // —— Disable ADC to save µA ——

// Read load current
uint16_t rawI = analogRead(IT_PIN);
float vAtA1   = rawI * (VREF / 1023.0f);
float vDiff   = vAtA1 - (VREF / 2.0f);
float loadmA_neg   = vDiff / SENSE_V_PER_A;
float loadmA  = loadmA_neg * -1.00f;
ADCSRA &= ~_BV(ADEN); // —— Disable ADC to save µA ——

// read voltage & % from MAX17048
float battV = maxlipo.cellVoltage();
float battP = maxlipo.cellPercent();
float chargeRate = maxlipo.chargeRate();
ADCSRA &= ~_BV(ADEN); // —— Disable ADC to save µA ——

// Read BH1750FVI lux
float lux = lightMeter.readLightLevel();
ADCSRA &= ~_BV(ADEN); // —— Disable ADC to save µA ——

// Re-init SD/SPI
SPI.begin();
if (!SD.begin(SD_CS_PIN)) return;

// Append to SD
File log = SD.open("log.csv", FILE_WRITE);
if (log) {
```

```
    log.print(dateStr);      log.print(',');
    log.print(timeStr);      log.print(',');
    log.print(tempC, 2);     log.print(',');
    log.print(pressMmHg, 2); log.print(',');
    log.print(humidity, 2);  log.print(',');
    log.print(battV, 3);     log.print(',');
    log.print(battP, 2);     log.print(',');
    log.print(chargeRate, 2);     log.print(',');
    log.print(loadmA, 2);     log.print(',');
    log.println(lux, 2);
    log.close();
  }

  if (battV < 3.2 || battP < 10.0) {
  // Save power and skip SD write
  return;
      }
}
```

Appendix B

Bill Of Materials

- Arduino Pro Mini 3.3V / 8MHz
- FTDI USB to TTL breakout
- Waveshare BME280
- Adafruit MAX17048
- BH1750FVI
- Micro SD card reader
- Micro SD card
- AEM10941 evaluation board
- MAX471
- LL200-2.4-75 solar cells

Most bullet points above are clickable.