StormGazer

Időjárás állomás projekt

Készítette:
Arany Zsolt

Contents

1 Bevezetés

A projektmunka a *Beágyazott rendszerek* című tárgy keretei között lett elkészítve. Az alapkövetelmények szerint egy időjárásállomást kell elkészíteni, amely önálló tápellátással rendelkezik, képes automatikus adatgyűjtésre (hőmérséklet és páratartalom), és ezeket az adatokat vezeték nélkül továbbítani.

Ezen követelmények mellett jobb érdemjegyért valamilyen általunk választott plusz funkciót vagy tulajdonságot helyezhetünk el rendszerünkben. Rendszerem - fantázia nevén **StormGazer** - képes hőmérséklet- és páratartalom-mérése mellett nyomást és a lehullott csapadék mennyiségét mérni. Ezeket egy mikrokontroller begyűjti, majd egy online adatbázisba feltölti, ahol az adatokat elemezni lehet.

Tápellátás céljából egy lítiumion-akkumulátorcellát alkalmaztam, amelyet egy napelemmel egészítettem ki. Ez habár nem nyújt elegendő teljesítményt a rendszer önálló üzemeltetésére, de működési idejét meghosszabbítja. Legvégül a rendszert egy 3D nyomtatott dobozba helyeztem el.

2 Alkatrészválasztás

2.1 Mikrokontroller

Raspberry Pi Pico W: Az egyik legolcsóbb és legmegbízhatóbb mikrokontroller a piacon. Előnyei közé tartozik, hogy képes I2C kommunikációra, valamint több digitális GPIO pinnel rendelkezik, mint versenytársai. A Pico W a Python-alapú *MicroPython* programnyelvet használja, amely hasonló kisebb méretű projektek megvalósítására tökéletesen alkalmas. Emellett már rendelkeztem egy ilyen eszközzel, így ez is elősegítette választásomat. Hátránya viszont, hogy *deepsleep* módban jóval többet fogyaszt, mint például egy ESP32.

2.2 Hőmérséklet- és páratartalom-érzékelő

BME280: A szenzor választásakor figyelembe kell venni a működési határokat. Magyarországon a minimum -35°C és maximum 41°C volt a valaha mért legszélsőségesebb hőmérsékleti érték. A BME280 képes ilyen tartományban is mérni, és az egyre enyhébb időjárás miatt a szenzor legpontosabb tartományában (0–60°C) fog mérni hőmérsékletet és páratartalmat egyaránt. Habár más szenzorok (például a DHT22) is alkalmasak lennének erre a feladatra, és pár száz forinttal olcsóbbak, a BME280 sokkal pontosabb és alacsonyabb fogyasztású. A fogyasztása akár 1–2 μA is lehet, ami nagyságrendekkel kisebb, mint más szenzoroknál.

2.3 Akkumulátor

Samsung INR-18650-35E: Ez a 3.7 V névleges feszültséggel rendelkező akkumulátor bőven elegendő feszültséget és áramerősséget képes biztosítani egy mikrokontroller működtetéséhez. Ennek a típusnak a kapacitása 3500 mAh, ami körülbelül 2 hét működésre elegendő.

2.4 Hall-szenzor

SA-15: Ez a szenzor a lehullott eső mennyiségét méri majd egy speciálisan ehhez 3D nyomtatott doboz segytségével. A szenzor rendelkezik a digitális kimenettel, így a rendszer az impulzusokat tudja számolni.

2.5 Fuel gauge

MAX17048: A líthiumion akkumulátor töltöttségi szintjéről ad információt százalék formájában. Az IC I2C-vel kommunikál így azt a Picoval könnyen ki lehet olvasni.

2.6 Napelem panel

PV110x60-6V/1W: Egy 6V, 150mA peak teljesítményű napelem panel az akkumulátort tölti majd.

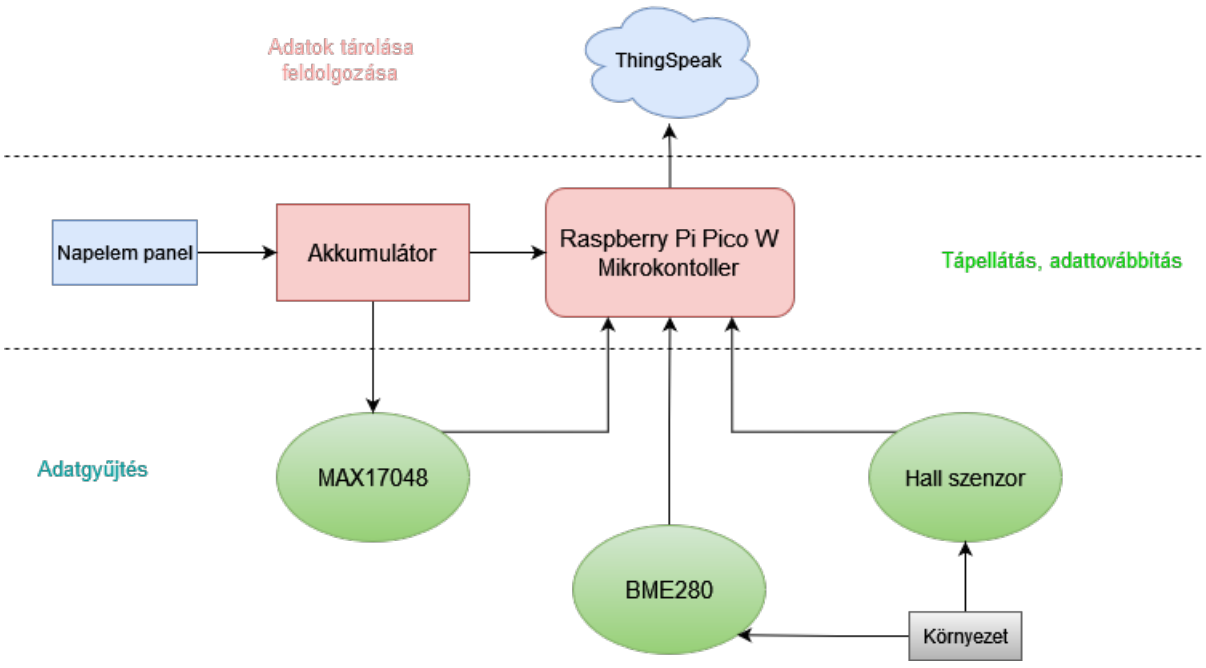

Figure 1: Rendszer blokkdiagramja

3 Kivitelezés

3.1 PCB

A projetem majdnem egésze egy PCB-re lett kivitelezve. Ezen a 90x50 lapon helyezkedik a mikrokontroller, a szenzorok, a töltés vezérlés, tápellátás és az akkumulátor is. A lap a PCBWay-ről lett rendelve, az alkatrészek pedig LCSC Electronics-ról lett rendelve. Minden alkatrészt saját magamnak forrasztottam fel.

3.2 Tápellátás

3.2.1 Reverse Polarity

Fordított polaritás elleni védelem céljából egy n-csatornás alacsony feszültségű *SI2312* mosfet található a rendszerben.

3.2.2 TP4056

A már fentebb említett 18650 líthiumion akkumulátorcella semmilyen védelemmel nem rendelkezik így gondoskodni kell, a túlfeszültség, túlmerülés és a túláram elleni védelemről. A TP4056-ban található DW01A IC ezekenek a védelmeknek tesz eleget, ezzel a rendszer majdnem összes védelmét ellátja. A moduolon továbbá található egy töltésért és a kisülésért felelős 2 n-csatornás mosfet IC (FS8205A). Továbbá található a modulnak nevet adó TP4056 IC töltés vezérlő. Ezen keresztül egy micro USB-vel vagy külső forrásból tölthető az akkumulátor. Jelen esetben külső forrásból, napelem panelből lesz töltve az akkumulátor. (A TP4056 töltési feszültség tartománya: 4-8V, ezért a 6V-os panelt rá lehet kötni a bemeneteire a modulnak.)

Figure 2: TP4056 modul

Megjegyzés: A sematikus ábrán ez a modul nem található meg, a gerber fájlon pedig csak a körvonala található meg.

3.2.3 Napelem panel

A napelem panel nem tud elegendő teljesítményt nyújtani hogy képes legyen az akkumulátort teljesen feltölteni. Líthiumion akkumulátoroknál a figyelni kell, hogy az ilyen kémiával rendelkező akkumulátoroknál 0°C alatt tilos tölteni, mivel ez jelentősen csökkenti a az élettartamát és akár veszélyessé is válhat. A töltés megszakítására a mikrokontrollert nem lehet használni, mert 2 mintavétel között nem aktív az eszköz. A probléma megoldására egy egyszerű SPDT toggle switch-et helyeztem el a PCB-n és a rendszer szoftveresen jelez ha egy bizonyos hőmérsékletet érzékel és figyelmezteti a felhasználót, akinek manuálisan ki kell kapcsolni a töltést.

3.2.4 Pico W védeleme

A Pico W külső tápról való biztonságos üzemeltetéséhez egy, az adatlap szerint, p-csatornás *DMG2305UX* mosfetet helyeztem el.

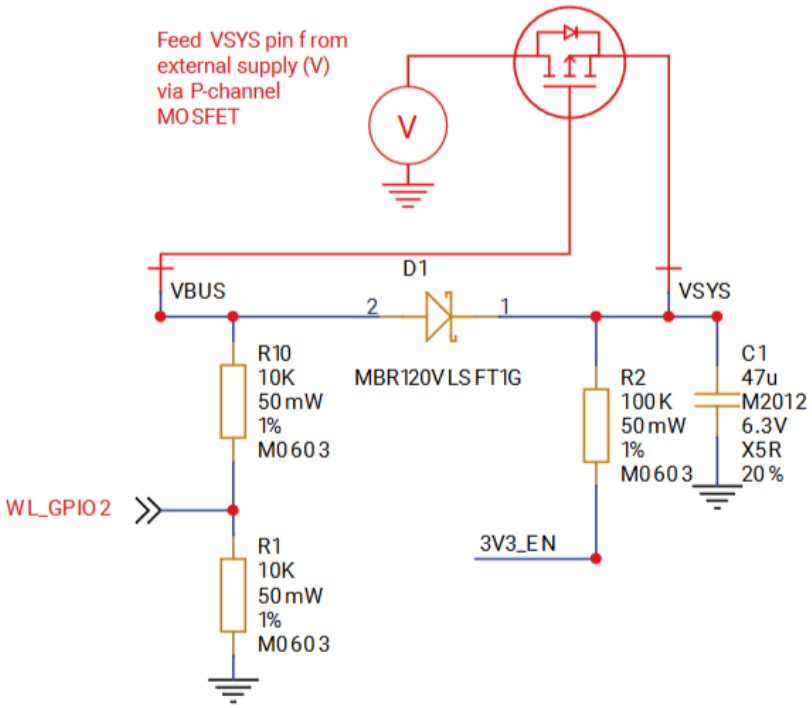

Figure 3: Pico W külsőtápfeszültségről töténő meghajtása

3.3 Hőmérséklet, páratartalom, nyomás mérés

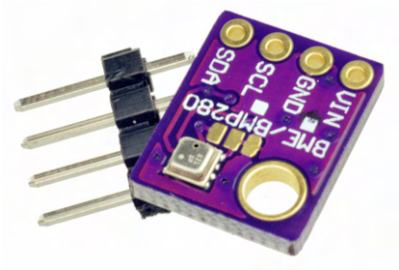

Figure 4: BME280 szenzor modul

Az egyik legfontosabb mérő műszer a rendszerben a BME280 szenzor. Ez a modul képes hőmérséklet, páratartalom és nyomás egyszerre történő kiolvasására viszonylag nagy pontossággal (T: ±1°C, P: ±1 mBar, H: ±3%) képes ezket az adatokat mérni. A modul I2C kommunkációs protokolt használva képes ezeket az adatokat továbbítani a mikrokontrollernek, ami meghívva a megfelelő könyvtárat egy string formában olvassa ki ezeket. Az SDA, SCL pinek mellett a tápellátást szolgáló VCC és GND kivezetések találhatók. A szenzor 3,3V-os feszültséget vár a VCC bemenetére, ez azért is hasznos, mert a Pico W is ezzel a logikai szinttel működik. Ebből kifolyólag szoftveresen megoldható, hogy a szenzor egyedül akkor legyen bekapcsolva amikor a mérni szeretnénk vele. Ezt legegyszerűbben úgy tudjuk megoldani, hogy a VCC-t egy GPIO pinre kötünk és ezt logikai magas állapotba helyezzük amikor mérni szeretnénk és a mérés befejeztével logikai alacsonyba. Ezzel a módszerrel több energiát tudunk megspórolni, habár a szenzor képes hibernációs módba kerülni egy bizonyos idő eltelte után, ahol akár csak néhány μA-t fogyaszt. A modult egy female header konnektorral a PCB-n helyeztem el.

MicroPython kód:

```
from machine import Pin, I2C
from time import sleep
import bme280

# Initialization of pins
led=Pin("LED", Pin.OUT) # Init LED for indication
bme_3v3= Pin(2, Pin.OUT)

# Collect data from BME280 sensor
def BME_read():
    bme_3v3.value(1) #turn on the sensor
    sleep(0.1)
    i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=40000)
    bme= bme280.BME280(i2c=i2c)
    sleep(5) #wait to stabilize
    t= float(bme.values[0])
    p= float(bme.values[1])
    h= float(bme.values[2])
    bme_3v3.value(0) #turn off
    return t, p, h

while True:
    led.toggle()
    t, p, h= BME_read()
    print(t, p, h)
    led.toggle()
    sleep(10)
```

3.4 Lehullott csapdék mérése

Csapadék mérésére egy külön 3D nyomtatott dobozt terveztem, amely egy hall szenzor segítségével képes kapcsolóként üzemelni, ezzel lehetővé teszi a impulzusok megszámolását. A doboz tartalmaz egy tölcsért, amely képes felfogni a vizet nagyobb területen, egy billenő elem amelyben egy mágnes található, ez fogja triggerelni a hall szenzort és egy a szenzort valamint a billenő elemet magába foglaló doboz. Tervezés után tudva a billenő

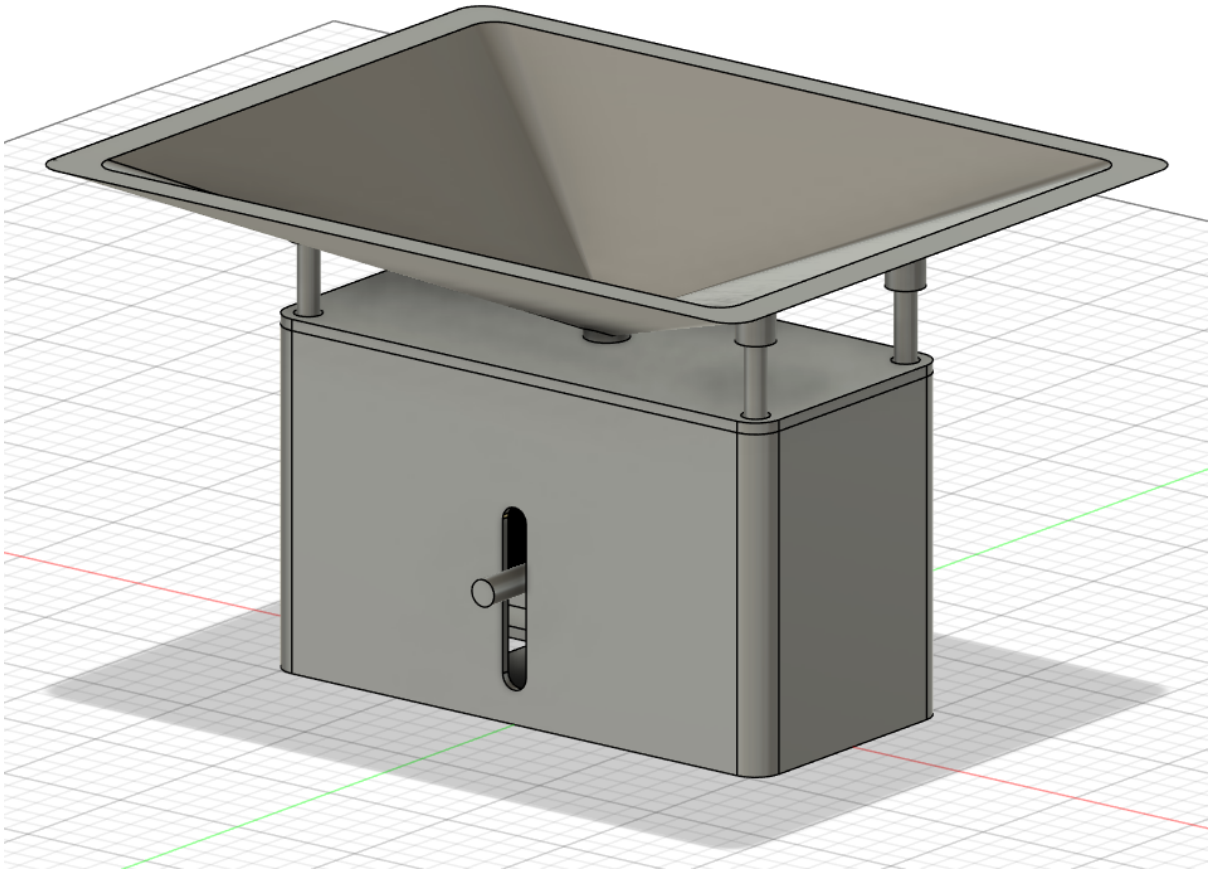

Figure 5: 3D design a csapadék mérőnek Fusionben

elem űrtartalmát és a tölcsér befogó területét vissza lehet számolni, hogy egy impulzus hány milliméter eső eset le. Azonban a mikrokontroller deepsleep állapotban nem tudja számlálni az impulzusokat a PCB-re egy 4 bites falling edge ripple counter IC-t helyeztem el. Ezzel a megoldással, amikor inaktív a Pico a ripple counter számolja és tárolja az adatokat. A mikrokontroller kikerülve a deepsleep üzemmódból párhuzamosan kiolvassa és átváltja a bináris adatot decimálisra és beszorzza a impulzusonkénti milliméter értékkel. Végül reseteli a ripple counter értékeit.

Megjegyzés: A szenzor a ripple counter tulajdonságai miatt minden második impulzust számol, ezért általába egy iterációnyi pontatlansággal jelzi ki az adatokat.

Tesztelés során a hall szenzor valóságot nem tükröző értékeket mutatott. A problémát azt okozta, hogy mivel a billenő elem rugalmasan ütközik a dobozzal, ezért a hall szenzor digitális kimenete peregni fog. Ezt a PCB tervezésekor ezzel a problémával nem számoltam így azt egy breadboardon utólag valósítottam meg a pergésmentesítést.

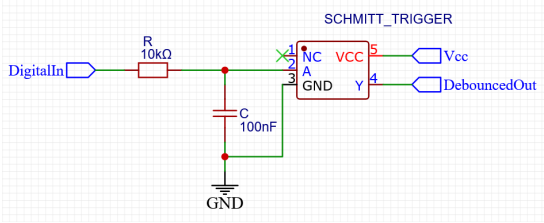

Figure 6: pergésmentesítés áramköri rajza

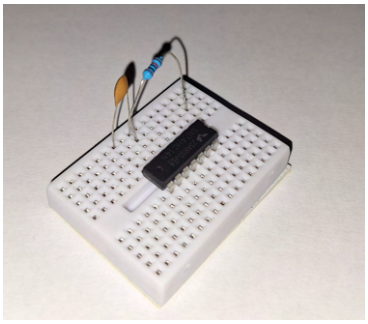

Figure 7: Schmitt-trigger megvalósítása

Az áramkör egy 6 csatornás Schmitt-trigger IC-ből, egy 10 kΩ ellenállásból és egy 100 nF kondenzátorrból áll a *Figure 5* kapcsolásirajz alapján.

MicroPython kód:

```
from machine import Pin
from time import sleep

# Init the pins
led=Pin("LED", Pin.OUT)
Q0= Pin(12, Pin.IN)
Q1= Pin(14, Pin.IN)
Q2= Pin(15, Pin.IN)
Q3= Pin(13, Pin.IN)
MR= Pin(11, Pin.OUT)

# Define a function for read the ripple counter and reset it
def ripple_read():
    q0= Q0.value()
    q1= Q1.value()
    q2= Q2.value()
    q3= Q3.value()
    temp= (q0*2**0+q1*2**1+q2*2**2+q3*2**3)*2 # convert the bin values
   to dec
    print(q3, q2, q1, q0)
    MR.value(1)
    sleep(0.5)
    MR.value(0)
    return temp

while True:
    led.toggle()
    print(ripple_read())
    sleep(5)
```

3.5 MAX17048

A líthiumion akkumlátor töltési állapotát megfigyelésére egy MAX17048 IC-t használok. Ez az IC szintén I2C protokolt használ. Ehhez az alkatrészhez, úgy mint a BME280-hoz, is létezik könyvtár amit a GitHubon találtam. Bár egy régebbi IC-hez lett ez a library írva kis változtatásokkal általam használt IC-vel is kompatibilis lett, így néhány sor kóddal megmérhető az akkumulátor töltöttségi szintje.

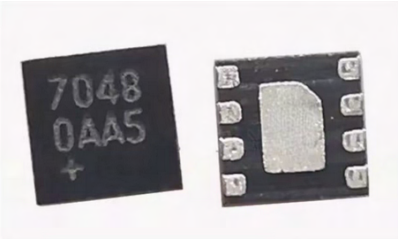

Figure 8: MAX17048 fuel gauge IC

3.6 PCB és 3D modell tervek

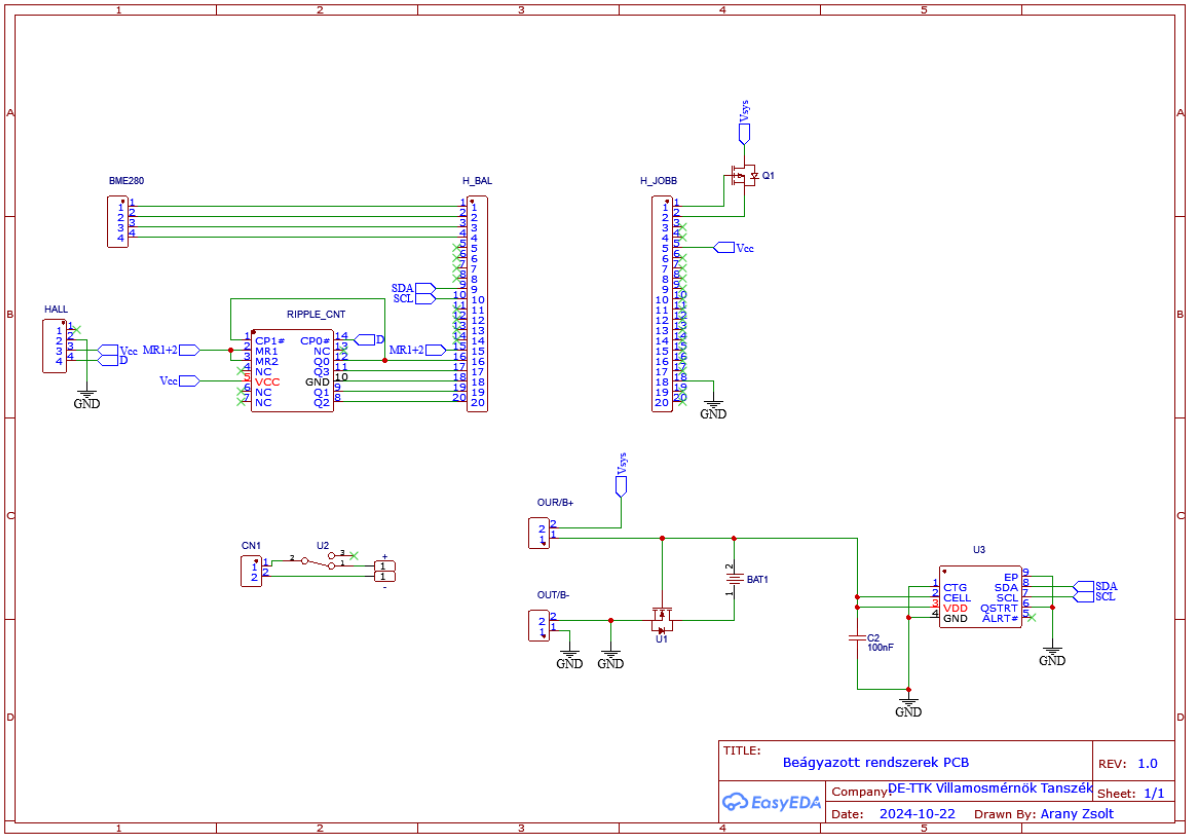

Figure 9: Sematikus ábra

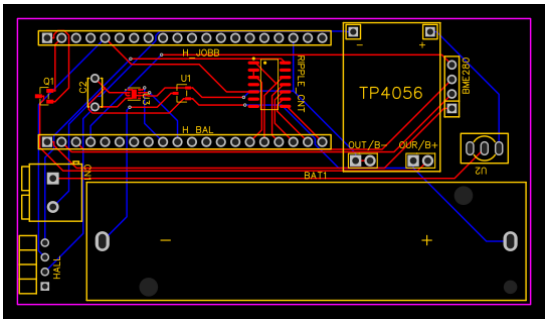

Figure 10: Gerber fájl

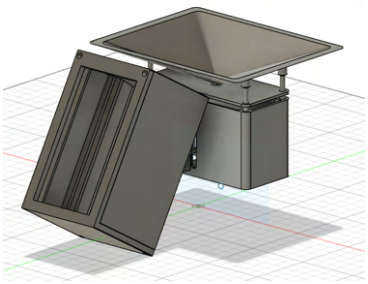

Figure 11: Teljes 3D modell

3.7 Mevalósított PCB, 3D doboz, webszerver és kód

3.7.1 PCB, 3D doboz

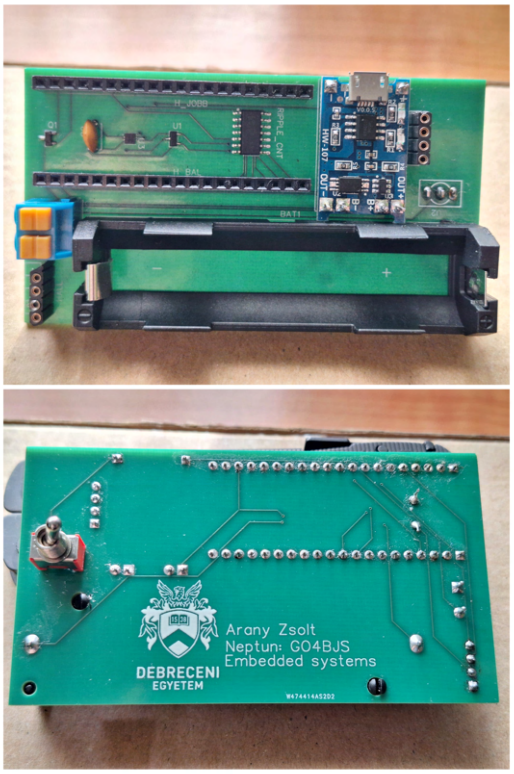

Figure 12: Megforraszott PCB

Figure 13: Kinyomtatott 3D doboz

3.7.2 Thingspeak

A mért adatokat a MathWorks által üzemeltett ThingSpeak felhő adatbázisba tárolom. Itt mindegyik mérésre külön grafikont lehet létrehozni és ezeket MatLab scriptekkel, valamint további beépített modulokkal lehet testreszabni. Egyszerre akár 8 mező adatát egy url kérelemmel lehet továbbítani az adatbázisnak. Projektemben én csak 5 mezőt használok (hőmérséklet, nyomás, páratartalom, csapadék mennyiség, akkumulátor töltöttség). Ezen adatok megjelenítes mellett ezen az oldalon jelez a rendszer a felhasználónak, hogy a töltést meg kell szüntetni.

Figure 14: ThingSpeak felület

3.7.3 Kód

A kóde egy GitHub repositoryban is megtalálható.

main.py

```
"""
2024. november 28.
Debreceni Egyetem - TTK Villamosmernok Tanszek - Embedded systems
-----------------------------------------------------------------
The following code uses of a Pico W microcontroller, a BME280, a hall sensor and lastly a MAX17043 fuel-gauge IC.
When executed it measures and uploads the collected data to a ThingSpeak server.

Made by: Zsolt Arany
"""

from machine import Pin, I2C, deepsleep
import bme280
import network
from time import sleep
import urequests
from max17043 import max17043

# Wi-Fi and ThingSpeak configuration
ssid= "---"
password= "---"
api_key= "---"
server= "---"
field_temp= 1
field_pres= 2
field_hum= 3
field_rain= 4
field_battery= 5

# Pin initialization
led=Pin("LED", Pin.OUT)
bme_3v3= Pin(2, Pin.OUT)
Q0= Pin(12, Pin.IN)
Q1= Pin(14, Pin.IN)
Q2= Pin(15, Pin.IN)
Q3= Pin(13, Pin.IN)
MR= Pin(11, Pin.OUT)

# Connect to Wi-Fi
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
    if not wlan.isconnected():
        print("Waiting for connection...")
    while not wlan.isconnected():
        sleep(0.1)
    ip= wlan.ifconfig()[0]
    print(f"Connected on {ip}")

# Send data to Thingspeak
```

```
51 def send_TS(temp, pres, hum, rain, battery):
52     url= f"{server}/update?api_key={api_key}&field{field_temp}={temp}&
       field{field_pres}={pres}&field{field_hum}={hum}&field{field_rain}={
       rain}&field{field_battery}={battery}"
53     request= urequests.post(url)
54     request.close()
55 
56 # Collect data from BME280 sensor
57 def BME_read():
58     bme_3v3.value(1)
59     sleep(0.1)
60     i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=40000)
61     bme= bme280.BME280(i2c=i2c)
62     sleep(5)
63     t= float(bme.values[0])
64     p= float(bme.values[1])
65     h= float(bme.values[2])
66     bme_3v3.value(0)
67     return t, p, h
68 
69 # Read and convert values from ripple counter and then reset it
70 def ripple_read():
71     q0= Q0.value()
72     q1= Q1.value()
73     q2= Q2.value()
74     q3= Q3.value()
75     temp= (q0*2**0+q1*2**1+q2*2**2+q3*2**3)*2
76     MR.value(1)
77     sleep(0.5)
78     MR.value(0)
79     return temp
80 
81 # Define functions to read and store the counter's value in a txt file
82 def cnt_read():
83     with open("data.txt", "r") as file:
84         content= file.read().strip()
85         cnt_value, time= content.split(',')
86         return int(cnt_value), int(time)
87 
88 def cnt_write(cnt_value, time):
89     with open("data.txt", "w") as file:
90         file.write(f"{cnt_value},{time}")
91 
92 while True:
93     led.toggle()
94 
95     # Rain value from file and ripple counter and then store it in data
       .txt file
96     cnt, time = cnt_read()
97     cnt+= ripple_read()
98     if (time==5):
99         rain=cnt
100        cnt=0
101        time=0
102    else:
103        rain=0
104        time+=1
105
```

```
    cnt_write(cnt, time)


    # Read values from BME280
    t, p, h= BME_read()

    # Read SoC of the lithium-ion battery
    m= max17043()
    battery= m.getSoc()

    print(t, p, h, rain, battery)

    # Connect to WiFi and send the data to ThingSpeak
    led.toggle()
    sleep(0.5)
    led.toggle()
    connect_wifi()
    led.toggle()
    sleep(0.5)
    led.toggle()
    send_TS(t, p, h, rain, battery)

    led.toggle()
    deepsleep(300000)
```

4 Köszönet nyílvánítás

Köszönet szeretnék mondanani Kiss Rebekának, aki kinyomtatta a 3D dobozom nagyrészét és Heim Péternek, hogy segített a PCB összeforrasztásában és 3D nyomtatásban.