MŰEGYETEM 1782

SZAKDOLGOZAT FELADAT

Berta Máté (HPD5LB)
Villamosmérnök hallgató részére

Talaj nedvességtartalom mérés IoT eszközzel

Manapság népszerű kutatási, fejlesztési terület az IoT. Ezen belül is nagy népszerűségnek örvendenek a különféle otthon automatizálási megoldások. A hallgató feladata is ebbe a témakörbe illeszkedik egy automatizált kerti öntözőrendszerhez tartozó mérőegység megtervezésével. Az egységgel szemben további elvárás, hogy illeszkedjen a hallgató önálló laboratórium feladatában tervezett rendszerbe, a méréseket adott időközönként és módon végezze, a gyűjtött adatokat a rendszer központi egysége felé továbbítsa.

A feladat első része a mérőegység rendszertervének elkészítése, figyelembe véve a már meglévő rendszerbe történő illesztés támasztotta követelményeket. Ennek része a megfelelő perifériák (például rádiós kommunikációhoz, beavatkozó szervhez) kiválasztása, a mérési adatok megfelelő formátumra hozása, a mérőegység villamosenergia igényének figyelembevétele (becslés akkumulátoros üzemidőre), költségszámítás és az elosztott szenzorhálózatba építés realitásának mérlegelése.

A hallgató feladatának – a fentieken felül – a következőkre kell kiterjednie:

- Szakirodalmi kutatás a talaj nedvességtartalom kapacitív elven történő méréséről
- Kapacitív szenzor kiválasztása
- A választott szenzorhoz különböző vizsgáló jeleket létrehozó áramkörök illesztése
- Kapcsolási rajz készítés
- NYÁK terv készítés
- Az elkészült áramkör bemérése, élesztése
- A mérőegység feldolgozó egységére beágyazott szoftver készítése
- Kalibrációs mérés a szakirodalomban is megtalálható módszerek segítségével
- A központi egység szoftverében módosítások végzése a mért adatok rendszerezett megtekinthetőségéne elérése érdekében

Tanszéki konzulens: Naszály Gábor, mestertanár

Budapest, 2020.10.10.

……..
Dr. Dabóczi Tamás
tanszékvezető
egyetemi tanár, DSc

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Berta Máté

TALAJ NEDVESSÉGTARTALOM MÉRÉS IOT ESZKÖZZEL

KONZULENS

Naszály Gábor

BUDAPEST, 2020

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott Berta Máté, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2020. 12. 05.

...…….
Berta Máté

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

1 Bevezetés

1.1 Otthon automatizálás kertkapcsolatos házakban

Az otthonainkban egyre jobban betörő IoT eszközök megváltoztatják elvárásainkat háztartási eszközeinkkel szemben. Az „smart” jelzővel ellátott használati tárgyaink száma egyre nő, melyek gyakran, egyes kényelmi funkciókon túl, távoli elérést tesznek lehetővé. Olyan információkat gyűjthetünk otthonunkról, mint például napelem celláink teljesítménye, házunk áramfelvétele, háztartási eszközeink fogyasztása vagy akár bejárati ajtónkról élő kameraképet nézhetünk távolról.

Az otthon automatizálási feladatok ellátásával szemben állított jellemző műszaki követelmények közé tartozik az alacsony fogyasztás, integrálhatóság valamilyen IoT rendszerbe, könnyű kezelőfelület és relatív alacsony ár.

Hazánkban lakóövezet szerint a házak 62%-a családi ház, zártkerti vagy külterületi ingatlan. [1] Az ilyen típusú ingatlanok gyakran rendelkeznek kertkapcsolattal, ahol konyhakert vagy gyümölcsös található vagy telepíthető. Az ilyen ingatlanok esetén az otthon automatizálást kiterjeszthető a kerti feladatok automatizálására, melyek közül az egyik legkevesebb infrastrukturális változtatást megkövetelő az öntözőrendszer „okosítása”.

Sok kert rendelkezik kiépített öntözőrendszerrel, amely egy vagy több csap megnyitásával lehetővé teszi az öntözést. Ezek a rendszerek gyakran időzítőkkel vannak ellátva, mely bizonyos szintű automatizált megoldást nyújt, de ezek a rendszerek nem tudják figyelembe venni az időjárási körülményeket, gyakran sok emberi beavatkozást is igényelnek, illetve állapotuk nem ellenőrizhető távolról.

A kiépített öntözőrendszerek viszont lehetőséget adnak a könnyű automatizálásra, hiszen egy mágnesszelepes csap segítségével villamos jelekkel irányítható az öntözés.

1.2 Öntözőrendszer bemutatása

Jelen szakdolgozat témája egy öntözőrendszerbe építhető mérő- és beavatkozó egység. A rendszer követelményeit és a mérőegység alapelvének működőképességét önálló laboratórium feladatom definiálja és támasztja alá, az erről készült dokumentum elérhető nyilvánosan. [2]

1-1. ábra SWAN logó

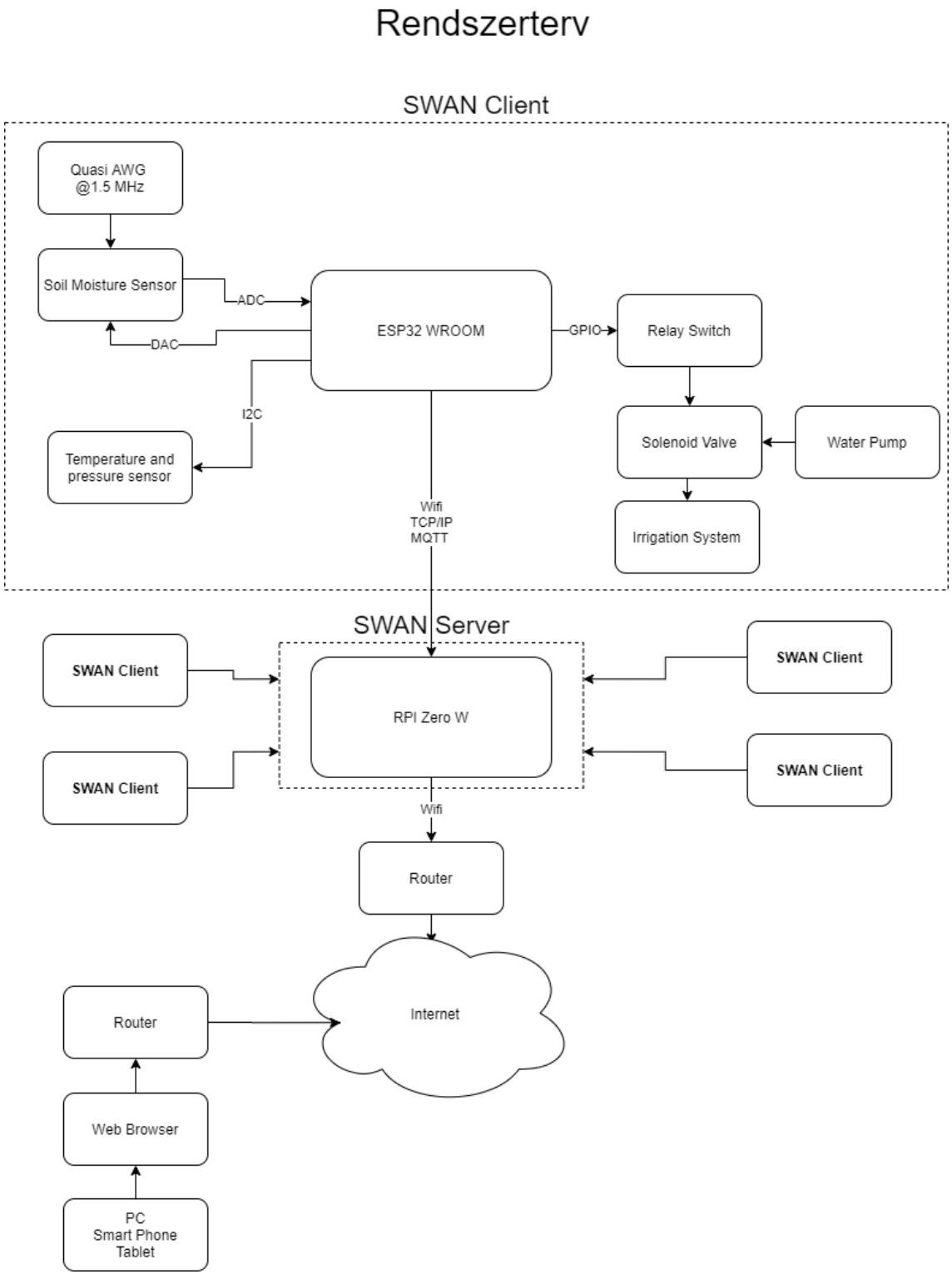
Az öntözőrendszer a Smart Watering Automation System (S.W.A.N.) nevet kapta. A mérő- és beavatkozó egységre SWAN kliensként, a rendszerhez tartozó központi egységre SWAN szerverként hivatkozok a továbbiakban.

Minden kliens a talaj nedvességtartalmát, a levegő hőmérsékletét és nyomását méri. A mért adatok felhasználásával avatkozik be minden egység, amely ellátott az ehhez szükséges hardver elemekkel. Minden mért adatot batchenként közlik a kliensek a szerverrel, mely utasítást is adhat a klienseknek, amikor azok bejelentkeznek hozzá.

A szerver és a kliensek feltételezik, hogy saját rádiós perifériáiknak hatótávolságán belül vezeték nélküli kapcsolaton (WIFI) keresztül hozzáférésük van a helyi hálózathoz, melyen kommunikálnak egymással.

A kliensek akkumulátoros üzemre optimalizáltak, viszont hálózatról (3,5V-6V DC) is meghajthatóak. Az akkumulátor töltése DC tápon keresztül történik, a szükséges töltések közötti idő minél hosszabbra növelése szempontja mind a hardveres, mind a szoftveres tervezésnek.

A SWAN kliens és szerver az alábbi rendszerterv szerint épül fel és csatlakozik egymáshoz.



1-2. ábra S.W.A.N rendszerterv

2 Kapacitív talaj nedvességtartalom mérés

2.1 Fizikai jelenségek

2.1.1 Definíció és validációs módszer

A talaj nedvességtartalma általában százalékban kifejezett mennyiség, amely arra megmutatja, hogy 1 gramm talaj hány g vizet tartalmaz. Mértéke tehát $\left[\frac{g}{g} \cdot 100\right] = [\%]$.

Egyszerű mérési módja, ha ismert tömegű, adott nedvességtartalmú talajt kiszárítunk és tömeg különbségét az elpárolgott víz tömegének vesszük. Ekkor a következőképpen számíthatunk:

$$S_{[\%]} = \frac{m_{H_2O}}{m_{talaj}} = \frac{m_{H_2O}}{m_{minta} - m_{H_2O}} = \frac{m_{minta} - m_{minta}'}{m_{minta}'}$$

Másik mérési módszer lehet, hogy ismert tömegű száraz, zérusnak tekinthető vizet mennyiségű tartalmazó talajhoz ismert tömegű vizet adunk. Ekkor különbség képzés nélkül megkaphatjuk ugyan azt az arányszámot:

$$S_{[\%]} = \frac{m_{H_2O}}{m_{talaj}}$$

Mindkét módszer tömegmérésre visszavezetett, amely jó pontosságú a jelenleg ismert technológiával. Viszont ezek a mérési módszerek nem alkalmasak gyakorlati helyzetekben, hiszen a vizsgálandó talaj általában nem eltávolítható és nem vonható ki belőle minden nedvesség. Emellett a talajba jutó vízmennyiség tömege gyakran nem ismert.

Az előzőekben ismertetett módszerek viszont alkalmasak más mérési módszerek ellenőrzésére, azoknak validációjára. Hiszen más fizikai jelenséget kihasználó mérési módszerrel mért értékekhez, ezekkel a módszerekkel lehetőség nyílik konkrét számértékeket rendelni.

Megjegyzendő, hogy a talaj nincs definiálva jelen dokumentumban, annak ellenére, hogy a talaj nedvességtartalom definícióját nem, de a továbbiakban bemutatott mérési módszert befolyásolja a talaj milyensége. Ennek vizsgálatára nem került sor, az elvégzett mérések kvarchomokban történtek, amely megfelelő tisztaságú és mennyiségű könnyen beszerezhető, illetve jól kiszárítható.

2.1.2 Kapacitív szenzor mérési elve

A talaj nedvességtartalmához valamilyen fizikai mennyiséget kell rendelnünk, annak érdekében, hogy mérni tudjuk a 2.1.1 fejezetben bemutatott módszerek alkalmazása nélkül.

Cél, hogy a mért fizikai mennyiséget feszültségmérésre vezessük vissza, mivel feszültséget kellő pontossággal és mikroprocesszorral tudunk mérni. Figyelembe vehetjük, hogy a talaj elektromos térben való viselkedése jellemezhető a komplex relatív permittivitásával (ε_r^*). HIVATKOZÁS

A következő egyenlet szerint változik ε_r^*:

$$\varepsilon_r^* = \varepsilon_r' - j \cdot \varepsilon_r'' = \varepsilon_r' - j\left(\varepsilon_{relax}'' + \frac{\sigma_{dc}}{2\pi f \varepsilon_0}\right)$$

Ahol σ_{dc} a talaj vezetőképessége, ε_{relax}'' molekulák nyugalmi hozzájárulása, ε_r' egy külső tér által az anyagban tárolt energiáját fejezi ki és j az imaginárius egység. A komplex permittivitás imaginárius része jellemzi, hogy az anyag mennyire „veszteséges az elektromos térre nézve. A veszteség mértéke függ a frekvenciától, a nedvességtartalmától és az anyag só és ion tartalmától. Tehát a közeg nedvesség tartalmával összefüggésben lévő fizikai mennyiség a komplex relatív permittivitás.

Amennyiben G_0 ismert geometriájú szenzor segítségével villamos teret juttatunk a talajba a következő összefüggés szerint definiálhatjuk a kapacitást:

$$C = \varepsilon_r^* \varepsilon_0 G_0$$

Az így bevezetett kapacitás lineárisan függ a komplex relatív permittivitástól, ráadásul villamosan mérhető mennyiség. Kapacitás mérésre több ismert módszer is létezik, melyek többsége feszültség mérésre vezeti vissza a problémát.

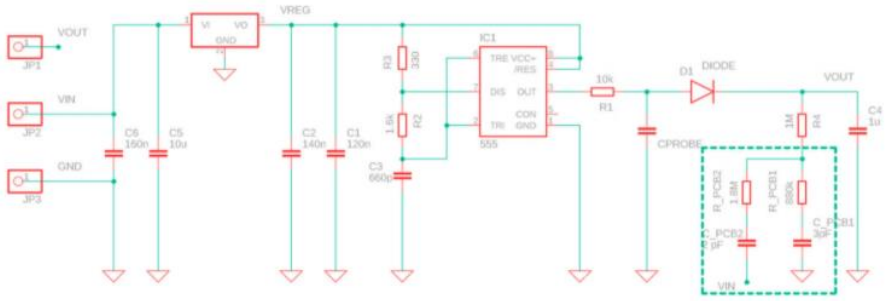
2.2 Szenzor jellemzése és kalibrációs mérés

2.2.1 Választott szenzor és mérési módszere

A választott szenzor egy márkajelzés nélküli különböző internetes áruházakban olcsón kapható és hobbi IoT alkalmazásokban gyakran használt mérőegység. A szenzor gyártója DFROBOT és SKU:SEN0193 néven érhető el. Specifikációja csak a tápfeszültség határait adja meg, működését és kapcsolási rajzát a hivatkozott cikk alapján ismertem meg.

A szenzor egy vizsgálójelet terhel talaj által reprezentált kapacitással, mely hatására a vizsgálójel csúcsértéke változik. A csúcsérték pedig (csúcsérték detektoron keresztül) követlenül ADC-vel mérhető egyenfeszültség.

A szenzor vizsgáló jelként 1.5 MHz frekvenciájú, egyenkomponensel rendelkező négyszögjelet alkalmaz, mely előállítására bistabil multivibrátort alkalmaz 555 időzítő IC és passzív komponensek alkalmazásával.



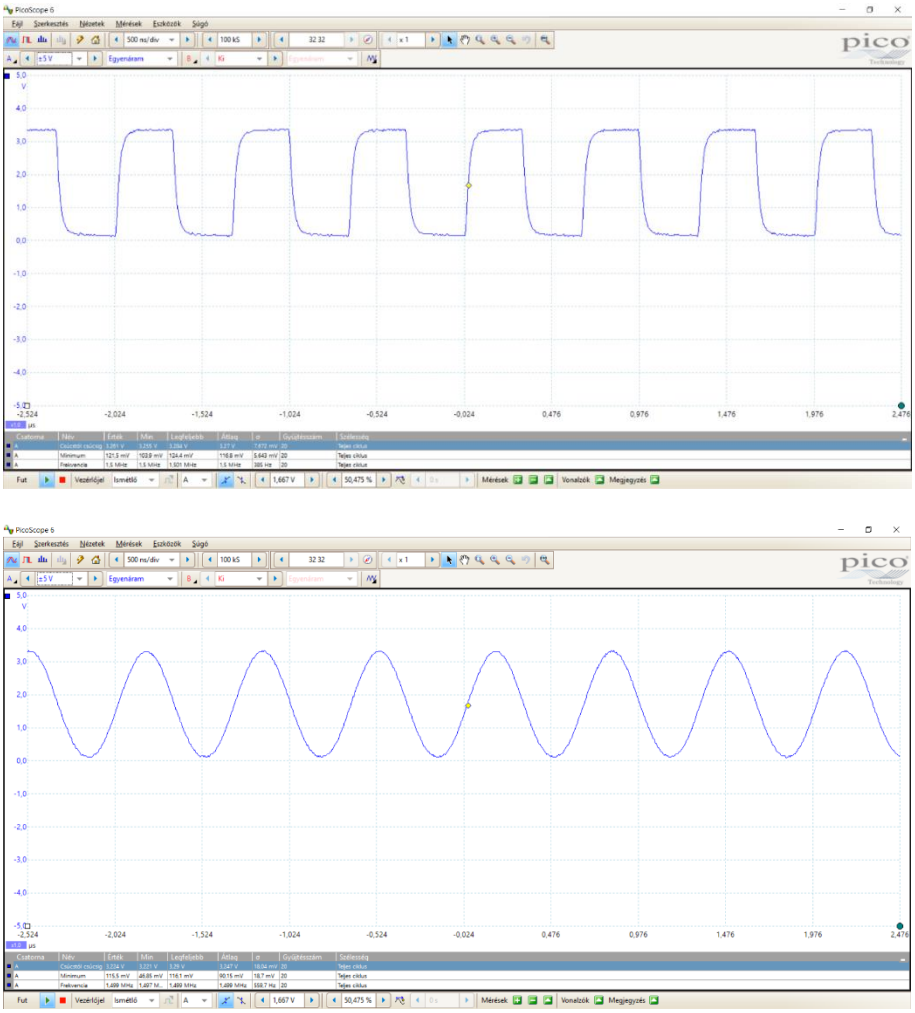
2-1. ábra Capacitive Soil Moisture Sensor v1.2 kapcsolási rajza

2.2.2 Mérőjel alakjának befolyásoló hatásának vizsgálata

A kiválasztott szenzor vizsgáló jele négyszögjel, melyről Fourier-sorfejtés segítségével megállapítható, hogy felharmonikusokat tartalmaz. A kapacitás és a komplex relatív permittivitás (közvetlenül és közvetetten) függ a frekvenciától. Vizsgálatom célja, hogy megfigyeljem milyen hatása van a mérőjel alakjának, ezáltal spektrumának, a mért feszültségre.

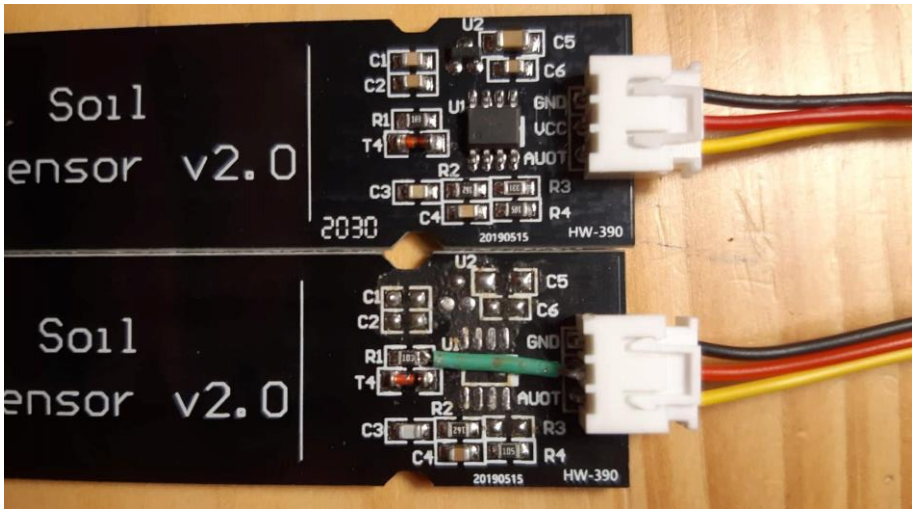
A vizsgálat továbbá lehetőséget nyújt, hogy a mért feszültségek és a talaj nedvességtartalma közötti kapcsolatot karakterizáljam, kalibrációs mérésként felhasználjam, erre a 2.1.1 fejezetben említett második módszert alkalmaztam.

A vizsgálat a szenzor által előállított négyszögjelnél meredekebb élekkel rendelkező négyszögjellel és szinusz jellel történt. Minden jel 0 és 3.3 V közötti, egyenfeszültségű komponenst is tartalmazó és 1.5 MHz frekvenciájú. A használt mérőjelek a 2-2. ábraán láthatóak.



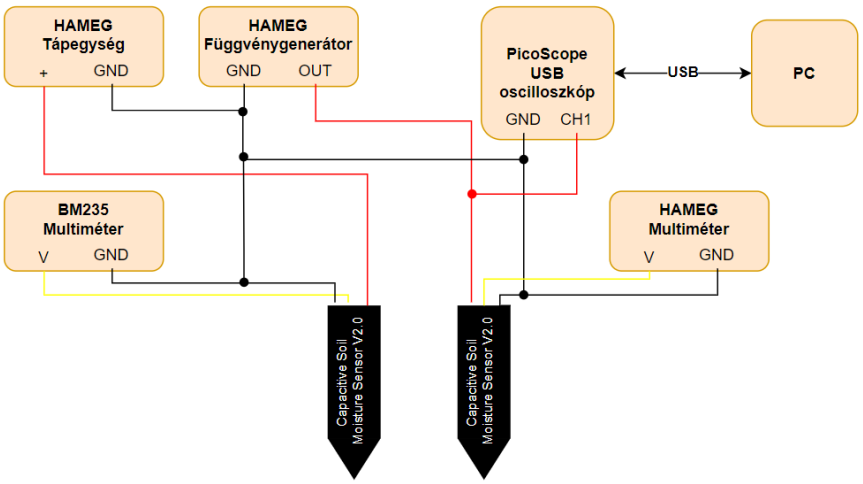
2-2. ábra Függvénygenerátorral előállított mérőjelek

A mérésem során két egy napon gyártott szenzor felhasználásával történt, melyek közül az egyiket módosítottam úgy, hogy külső mérőjel csatlakozhasson hozzá. A módosítás során az 555-ös IC és a hozzá tartozó áramköri elemeket leforrasztottam és a eredeti kapcsolás R1 ellenállását közvetlenül a Vin bemenettel kapcsoltam össze. A két szenzor egymás mellett a 2-3. ábra látható.



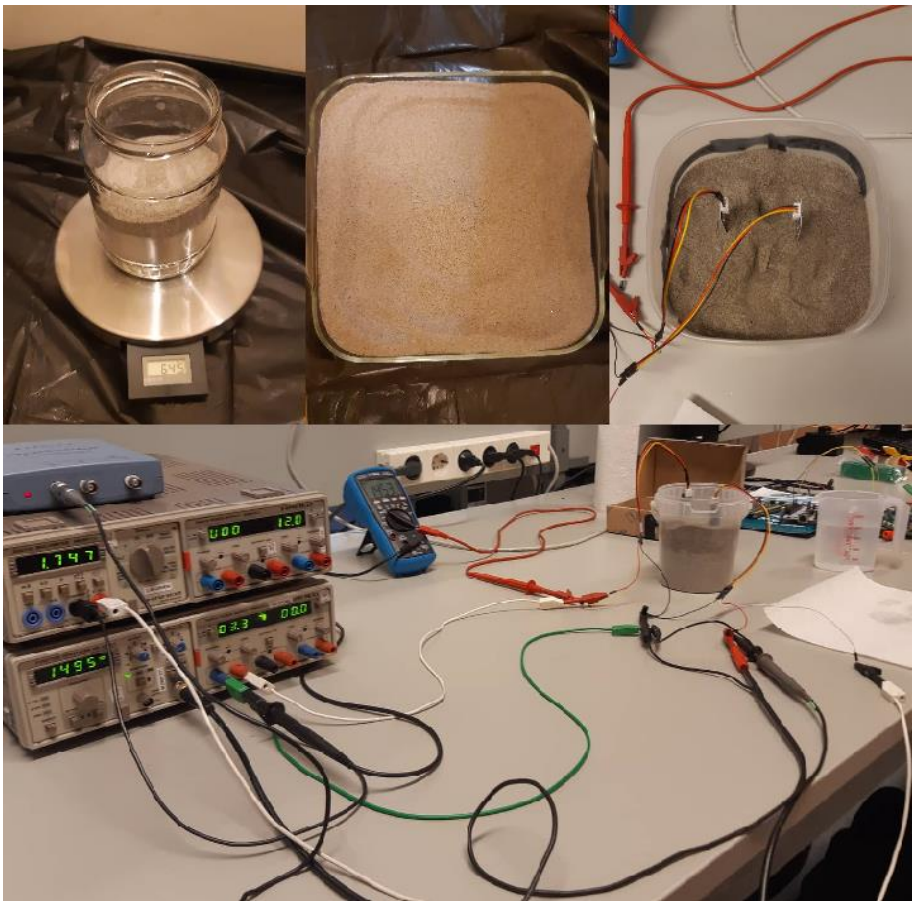
2-3. ábra Eredeti (felső) és módosított (alsó) szenzor

A kalibrációs mérések során a nem módosított és a módosított szenzor feszültségértékeit jegyzem fel és hasonlítom össze táblázatos és grafikus formában. A mérési elrendezés sematikus vázlata a 2-4. ábraán látható.



2-4. ábra Kalibrációs mérés

A mérés előkészületeként 5000g szárított kvarchomokot egy üvegedényben 1 órán keresztül 110 ºC hőmérsékleten tovább szárítottam. Az így kapott homokot tekintem zérus nedvességtartalmú talajnak, számításaim során 0g víztartalmat tulajdonítok neki. A 5000g homokot 1:2:2 arányban felosztottam, egy teszt mérés és két éles mérés számára. A tesztmérés során megfigyeltem, hogy a homok milyen módon veszi fel a vizet, mennyire számít a szenzorok mozgatása és talajban lévő mélysége a mérés során. A 2-5. ábra a mérés előkészítése és a mérés elvégzése közben készített képeket tartalmaz.

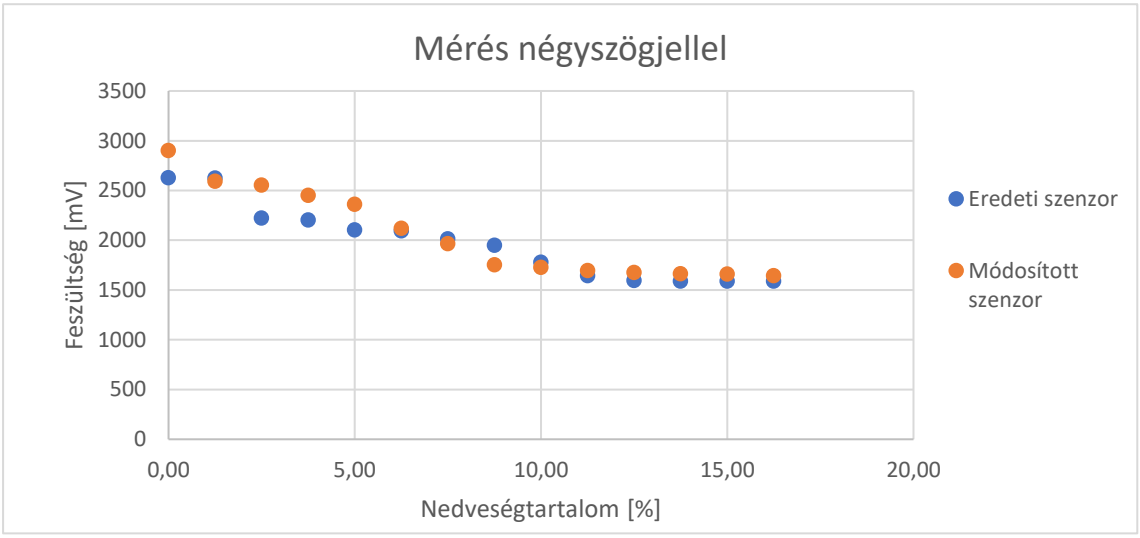


2-5. ábra Kalibrációs mérés fázisai

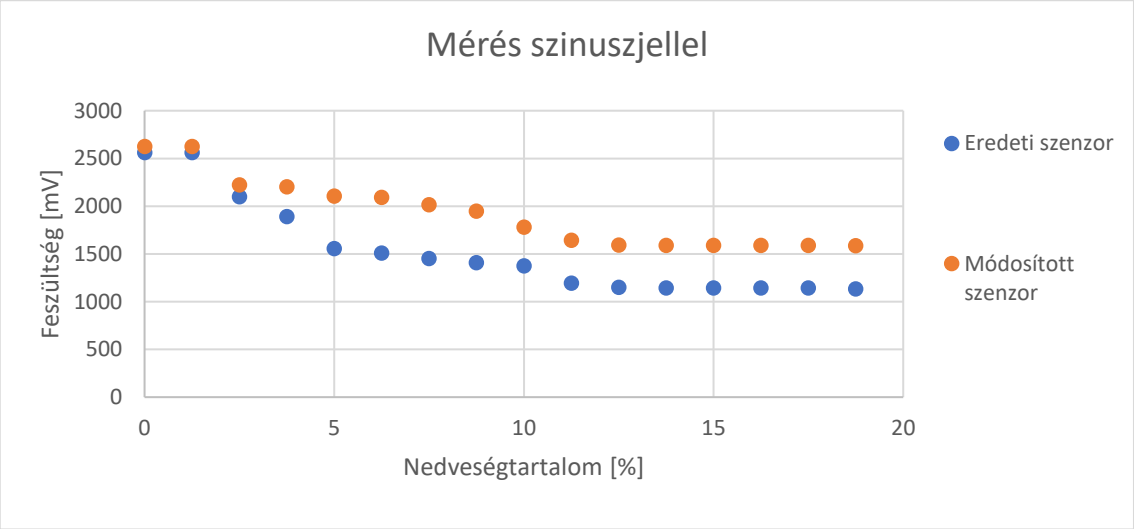
Az éles mérések során ügyeltem, hogy a szenzorok szimmetrikusan helyezkedjenek el a mérőedényben, teljes felületüket talaj fedje és a hozzáadott víz azonos mértékben jusson el hozzájuk. Víz hozzáadása előtt feljegyeztem a szenzorok által mutatott feszültséget, majd 25ml-es lépésenként adtam hozzá szobahőmérsékletű csapvizet. A víz hozzáadása után, megvártam amíg a multiméterek legalább két tizedes jegyig állandó értéket mutatnak és feljegyeztem a feszültséget egy táblázatba. A víz hozzáadását addig folytattam amíg legalább öt azonosnak tekinthető feszültségértéket nem olvastam le. A két éles mérés között a szenzorokat nedvszívó törlőkendővel tisztítottam meg, annak érdekében, hogy a felületen ne maradjon nedves homok.

Az első mérés négyszögjellel történt és a következő táblázatban rögzített adatokat kaptam, melyeket grafikonon is ábrázoltam.

Nedvességtartalom [%]	Feszültség ("kontroll" szenzor) [mV]	Feszültség (kísérleti szenzor) [mV]
0,00	2214	2901
1,25	1842	2592
2,50	1801	2556
3,75	1645	2451
5,00	1474	2360
6,25	1151	2121
7,50	1064	1965
8,75	979	1755
10,00	961	1728
11,25	945	1697
12,50	946	1678
13,75	947	1662
15,00	945	1659
16,25	939	1644



Nedvességtartalom [%]	Feszültség ("kontroll" szenzor) [mV]	Feszültség (kísérleti szenzor) [mV]
0	2563	2627
1,25	2562	2626
2,5	2100	2223
3,75	1893	2205
5	1555	2105
6,25	1510	2093
7,5	1452	2015
8,75	1410	1950
10	1375	1780
11,25	1195	1643
12,5	1152	1595
13,75	1144	1590
15	1145	1590
16,25	1144	1590
17,5	1143	1589
18,75	1134	1587



2.2.3 Következtetés

3 Hardver tervezés

3.1 Elvi kapcsolási rajz egységek

3.1.1 Mágnesszelep működése elve

A kliensek kimenete egy az öntözőrendszerben meglévő beavatkozó szervhez kapcsolódik, mely működési elvének megértése szükséges a megfelelő vezérlőjel előállításához.

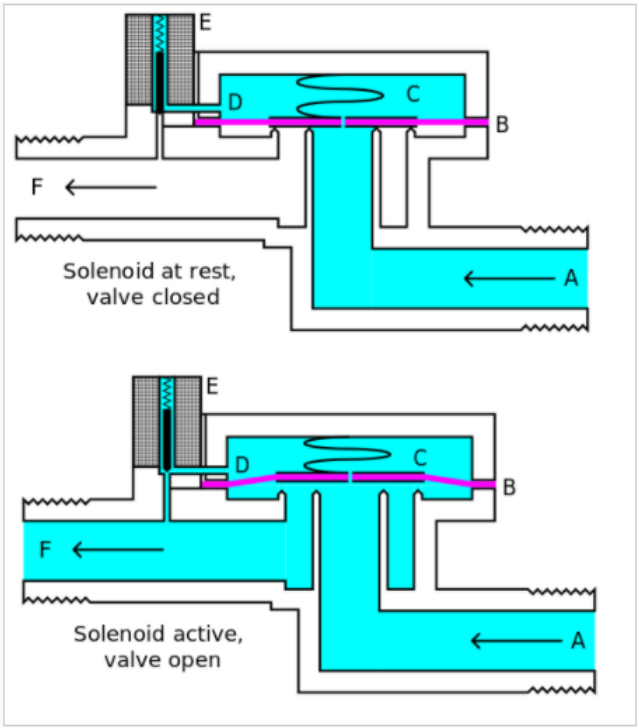
Ez a beavatkozószerv egy mágnesszelep, amely elektromechanikusan működtetett szelep. A mágnesszelepek a folyadékokat szállító rendszerekben leggyakrabban használt vezérlőelemek. Számos alkalmazási területen megtalálhatók. Feladatuk a folyadékok útjának elzárása, felszabadítása, a folyadék adagolása, elosztása vagy keverése. A mágnesszelepek gyorsak és biztonságosak, nagy megbízhatóságúak, hosszú élettartammal rendelkeznek.

A 3-1. ábra egy általános szelep kialakítását mutatja, amely a víz áramlását szabályozza ebben a példában. A legfelső ábrán a szelep zárt állapotban van. A nyomás alatt levő víz A pontba kerül. B egy rugalmas membrán, fölötte pedig egy gyenge rugó található, amely lenyomja a membránt. A membrán középpontjában lyuk található, amely lehetővé teszi, hogy nagyon kis mennyiségű víz áramoljon rajta keresztül. Ez a víz úgy tölti ki a membrán másik oldalán lévő C üreget, hogy a nyomás a membrán mindkét oldalán egyenlő legyen, azonban az összenyomott rugó összességében lefelé irányuló erőt szolgáltat. A rugó gyenge, és csak azért képes bezárni a beömlőnyílást, mert a membrán mindkét oldalán kiegyenlítődik a víznyomás.

Amint a membrán bezárja a szelepet, az alja kimeneti oldalán lévő nyomás csökken, és a fenti nagyobb nyomás még szilárdabban zárja. Így a rugónak nincs jelentősége a szelep zárva tartása szempontjából.

Mindez azért működik, mert a kis D lefolyójáratot egy csap rögzíti, amely az E mágnesszelep armatúrája, amelyet egy rugó nyom le. Ha az áram áthalad a mágnesszelepen, a csapot mágneses erővel visszahúzza, ekkor a C kamrában lévő víz gyorsabban üríti ki a D járatot, mint amennyit a lyuk képes feltölteni. A C kamrában a nyomás csökken, a beérkező nyomás megemeli a membránt, ezáltal kinyitva a fő szelepet. A víz a mágnesszelep inaktiválódásáig közvetlenül A-ból F-be áramlik.

Amikor a mágnesszelep inaktiválódik, a D járat ismét záródik, ekkor a rugónak nagyon kevés erőre van szüksége a membrán újbóli lenyomásához, a főszelep bezárul. A gyakorlatban gyakran nincs külön rugó; a membrán úgy van kialakítva, hogy rugóként is funkcionáljon. [3]



A- Input side
B- Diaphragm
C- Pressure chamber
D- Pressure relief passage
E- Electro Mechanical Solenoid
F- Output side

3-1. ábra Mágnesszelep működése

Ez a kialakítás normál állapotában zárt (NC), amely ebben az alkalmazásban előnyös, hiszen energiát, csak nyitott állapot közben fogyaszt. Egy öntözőrendszer egy nap során legtöbbet zárt állapotban van, mivel nem öntözünk folyamatosan.

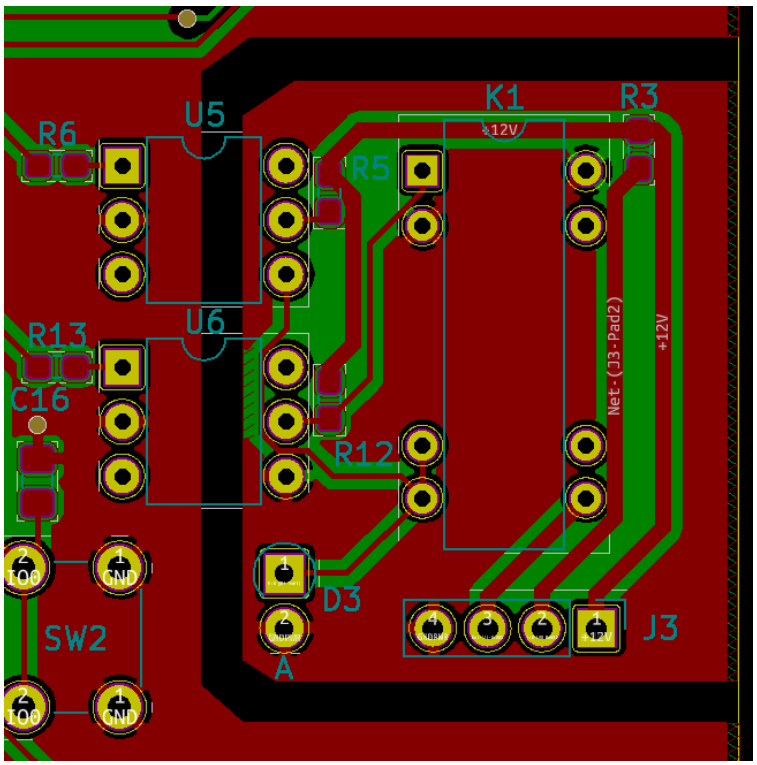
3.1.2 Mágnesszelep vezérlése

A mágnesszelep működési elvéből kitalálhatjuk, hogy villamos vezérlés szempontjából egy tekercset (vagy gondolkodhatunk úgy is, hogy egy elektromágnest) kell energizált állapotba hozni, majd kikapcsolni. A rendszer olyan szelepeket tud kezelni, melyeket elegendő DC feszültséggel ellátni, így a tekercs energizálásához nem szükséges szinuszos vezérlő jelet előállítani. A méretezés ebben az esetben 12 V-ról működő és maximum 6 W-ot fogyasztó szelep alapján történt.

A kliensek alacsony fogyasztású mérőeszközök, nem rendelkeznek 12 V-os tápfeszültséggel, viszont 12 V-os táp csatlakoztatásával vezérelhető lesz a mágnesszelep.

A 12 V-os táp galvanikusan leválasztott a kliensek akkumulátorról előállított feszültségeitől. A leválasztást optikai csatoló áramkörrel és külön PWRGND rézréteg segítségével realizáltam, az IPC2221 szabvány által definiált minimumtávolság többszörös betartásával. (3-2. ábra)

(HIVATKOZÁS)https://www.smpspowersupply.com/ipc2221pcbclearance.html



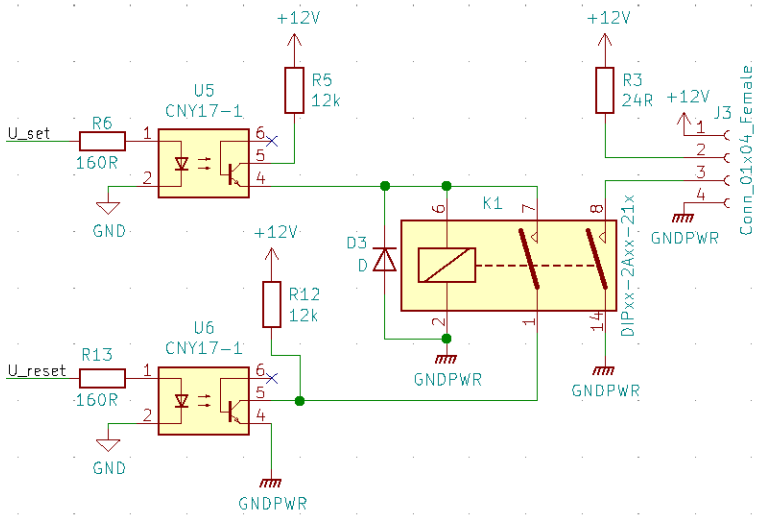
3-2. ábra Galvanikusan leválasztott öntartó relé kapcsolás realizációja

A 3-3. ábra tartalmazza az elvi kapcsolási rajzot. Az áramkörnek két bemenete van U_{set} és U_{reset}, melyek a be- és kikapcsolásért felelnek. A be- és kikapcsoló vezérlőjelek azonosak, legalább 6 µs széles 3.3 V feszültségű impulzusok. A R_6, R_{13} soros ellenállások áramkorlátozó szereppel rendelkeznek, 2 mA a maximális áram.

A bekapcsoló impulzus hatására U5 kapcsolón keresztül 1 mA-es áram folyik K1 relé tekercsén, mely hatására K1 NC kontaktusai bezárnak és J3 csatlakozón keresztül maximum 500 mA áram mellett a mágnesszelep tápot kap.

A bekapcsoló impulzus után az áramkör öntartásban marad K1 relé 1-7 kontaktusán keresztül, mindaddig amíg a relé tekercsének potenciálját a kikapcsoló impulzus hatására U6 kapcsoló 0 V-ra nem húzza.

A K1 relé tekercsének kikapcsolásakor figyelembe kell venni, hogy a tekercs mágneses terében tárolt energia kontrollált módon disszipálódjon el. D3 teljesítménydióda ezt a feladatot látja el, kikapcsolás esetén a tekercs a diódán keresztül disszipálja el energiáját.



3-3. ábra Galvanikusan leválasztott öntartó relé kapcsolás

A 12V-os külső táp J3 csatlakozón keresztül kerül a NYÁK-ra, így szükség esetén más DC feszültséget is kapcsoltathatunk a kliens eszközökkel, például 24 V-ot. Ekkor az ellenállás értékek változtatása szükséges lehet. A huzalok szélessége és egymástól való távolsága megengedi 3.3 V-tól 24 V-ig terjedő külső feszültség használatát.

3.1.3 Kliens tápellátása

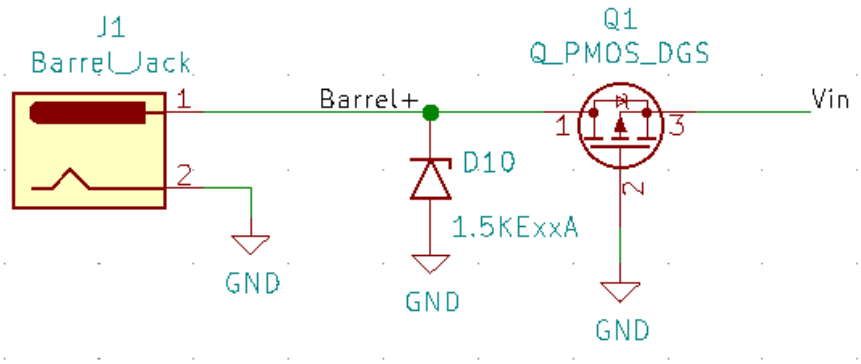
A kliensek mérési helyszíne nem teszi lehetővé a kényelmes csatlakozást a villamos hálózathoz. Mindenképpen telepes vagy akkumulátoros tápellátásra van szükség.

Alacsony fogyasztású alkalmazásoknál a telepes tápellátás nagy előnyökkel bír, hiszen technológiai előnye az akkumulátorokkal szemben, hogy elhanyagolható az önkisülési jelenség. Viszont egy tipikus CR2032 telep 100 mA-es csúcsnál nagyobb áram leadására nem képes, emiatt nagy méretű hidegítő kondenzátorokra (akár szuper kondenzátorokra) lenne szükség, hogy az ESP32 vezeték nélküli kommunikációja közben elegendő áramot tudjon szolgáltatni egy ilyen cella. Mindemellett 200-300 mAh-s tárolókapacitása, sem lenne elég a kliens ellátására.

Az viszonylag magas áramleadási szükséglet és nagy 2000-3000 mAh-s tárolókapacitás érdekében Li-ion akkumulátor alkalmazása mellett döntöttem.

A választás egy 18650-es és 3.7 V nominális feszültséggel rendelkező akkumulátorra esett, melynek a kapacitása 2600 mAh és maximális áramterhelhetősége 10A. A 18650 elnevezés az akkumulátor felépítésének dimenzióit adja meg, emiatt legalább 6,5 cm (praktikusan 8 cm) helyre van szükség a NYÁK-on az akkumulátornak. A helyigény mellett az akkumulátor beszerzési ára a legnagyobb hátránya ennek a megoldásnak, hiszen az körülbelül az alkatrészek árának ¼ része. (A hardveres költségek összegzését külön alfejezet tárgyalja.)

3.1.4 Bemeneti feszültség védelmei



3-4. ábra ESD és fordított polaritás védelem

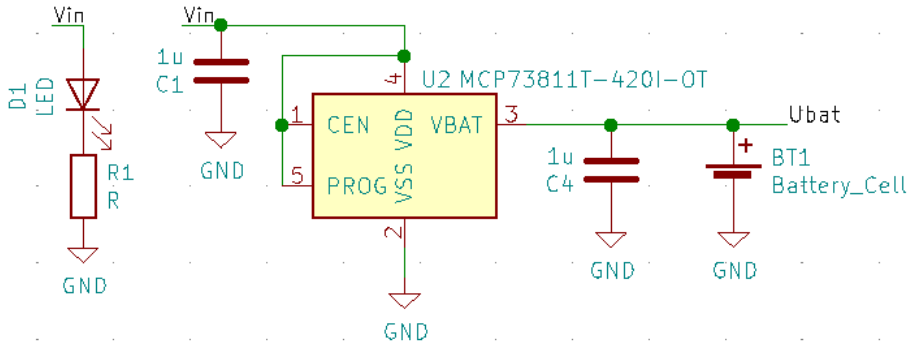
A 3-4. ábra a bemeneti feszültségre alkalmazott ESD és fordított polaritás védelem áramkörének kapcsolási rajzát tartalmazza. Az ESD védelmet TVS dióda látja

el. Amennyiben J1 csatlakozón fordított polaritású feszültség jelenik meg Q1 P csatornás MOS-FET szakadásként viselkedik, ezáltal védve az áramkört.

Ennek bizonyítására végezzünk gondolatkísérletet. Tegyük fel, hogy Q1 kapcsoló zárt állapotban van, ekkor V_{in} GND-hez képest negatív feszültség. Egy P csatornás FET csak negatív gate-source feszültség (U_{GS}) mellett van zárt állapotban. Viszont V_{in} GND-hez viszonyítva pontosan -$U_{GS,}$ tehát negatív V_{in} feszültség esetén U_{GS} pozitív, melyből következik, hogy a tranzisztor nyitott állapotban kell legyen.

Összegezve Q1 tranzisztor csak helyes bemeneti feszültség mellett lehet nyitott állapotban, U_{DS} maximális feszültségig védi az áramkört a fordított polaritás ellen.

3.1.5 Akkumulátor töltő áramkör



3-5. ábra ábra Li-ion akkumulátor töltő áramkör

Az akkumulátor töltését egy single-cell (egy cellás) Li-ion akkumulátor töltő integrált áramkör látja el. Az akkumulátort konstans áramú töltés mellett 4.2 V feszültségig tölti az áramkör. Az IC PROG lábának logikai magas feszültségre kötésével 500 mA-es töltőáram lett beállítva. Így az akkumulátor $\frac{2600\ mAh}{500\ mA} = 5.2h \sim 5h$ alatt tölthető fel. Azáltal, hogy töltésre ritkán van szükség és az akkumulátor kevesebb, mint egy éjszaka alatt feltölthető, így elfogadható a töltési idő.

C_1 és C_4 kondenzátorok a be- és kimeneti feszültségek hullámosságának csökkentésére és impulzusszerű áramfelvételek töltés szükségletének ellátására szolgálnak. Más szempontból nézve passzív RC szűrőként funkcionálnak a tápvonalakon.

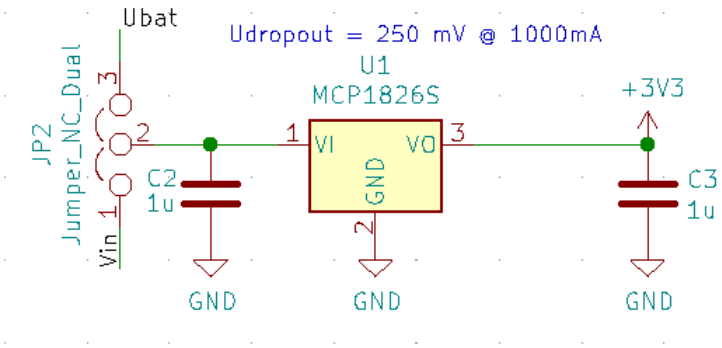
3.1.6 Szabályozott tápfeszültség előállítása

Az ESP32 és egyéb a kliens részét képző IC-k számára a tápfeszültséget egy 1000 mA-t leadni képes Low Dropout Voltage lineáris ún. áteresztő tranzisztoros szabályozó látja el. LDO-ra szükség az akkumulátor feszültség nominális értékének és 3.3 V-os előállítandó feszültségnek közelsége és a felesleges hőfejlesztés elkerülése miatt volt szükség.

C_2 és C_3 kondenzátorok szerepe megegyezik a 3.1.5 pontban szereplő C_1 és C_4 kondenzátorokkal.

A feszültségszabályzó bemenetére jumper segítségével külső tápfeszültséget is lehet csatlakoztatni, de ennek szerepe szigorúan fejlesztési célú.

A lineáris feszültségszabályozókat nagyfrekvenciás zajok ellen védeni kell, ennek gyakori megoldása egy a szabályzóval sorba kötött L-C szűrő. A szűrésre, azért van szükség, mivel a szabályozóban lévő tranzisztor nem képes végtelen sávszélességgel kapcsolni és a többletfeszültséget eldisszipálni. Azonban a Li-ion akkumulátorok feszültsége kevés zajt tartalmaznak, eleve jól szűrt tápforrásnak tekinthetők. Emellett még passzív LC szűrő alkalmazásához is szükség van egy viszonylag drága áramköri elemre, a tekercsre. A felsorolt érvek miatt (kis zajtartalmú forrás, költségminimalizálás) nem került szűrő az LDO bemenetére.



3-6. ábra LDO fix 3.3 V kimeneti feszültséggel

A kimeneten található 1µF-os kerámia kondenzátor mellett minden IC táplábaihoz közel 100 nF-os kondenzátor van párhuzamosan kötve C_3 kapacitással. Ennek segítségével növelve a kondenzátorok effektív sávszélességét.

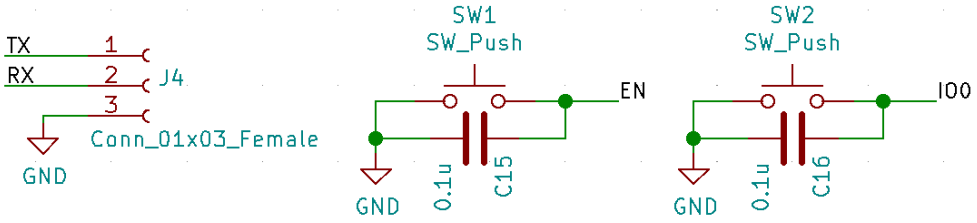
3.1.7 Programozó interface

Az ESP32 programozása UART periférián keresztül történik. Ez az UART periféria nem csak a programozhatóságot teszi lehetővé, hanem debug portként is használható fejlesztés közben, vagy javítások elvégzésénél. (Hibaüzenetek küldhetők rajta.)

Az ESP32 modul programozható állapotba hozásához szükséges a processzor IO0 lábát alacsony logikai szinten tartani, erre nyomógomb segítségével van lehetőség. Programozás után hardveres reset szükséges, melyet EN jel logikai alacsony szintre húzásával érhetünk el, nyomógomb segítségével.

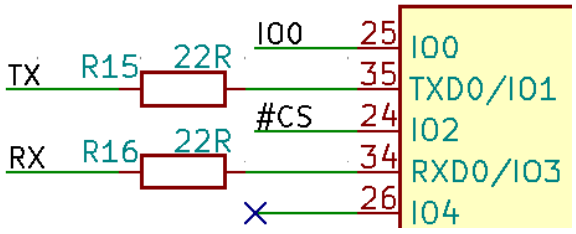
A nyomógombok pergésmentesítésére az ESP32 Dev Kit C adatlapja által javasolt megoldást alkalmaztam. (HIVATKOZÁS)

A programozó interface és a pergésmentesített nyomógombok kapcsolási rajzát a 3-7. ábra tartalmazza.



3-7. ábra Programozó interface

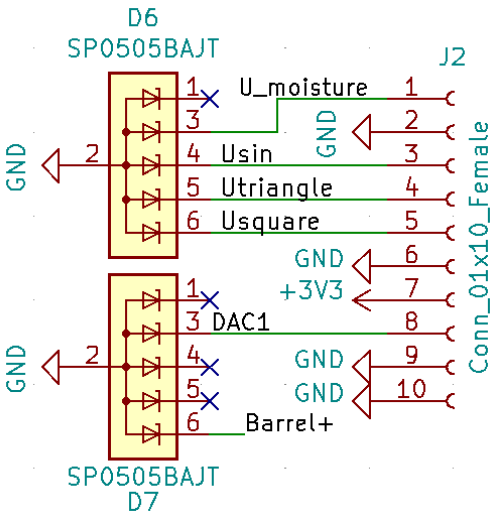
Kihasználva a NYÁK-on megjelenő parazitakapacitásokat, kis értékű 22 Ω értékű ellenállások felhasználásával TX és RX jelek meredek éleit szűröm, ezzel csökkentve az általuk a NYÁK-on keltett zajt.



3-8. ábra Digitális jel szűrőáramköre

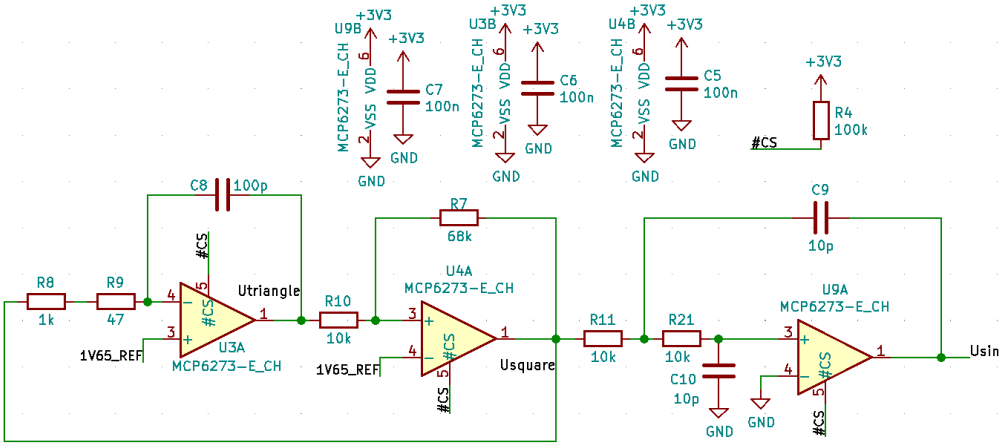
3.1.8 Méréshez szükséges csatlakozó

A méréshez szükséges gerjesztőjelek és a mért jel, 3.3 V és GND jelekkel kiegészítve tüskesor csatlakozón találhatók a NYÁK-on. A mérővezetékek hossza, illetve a külvilággal való rendeltetésszerű kapcsolata miatt ESD védelemmel láttam el a tüskesor csatlakozót, TVS dióda tömb alkalmazásával.



3-9. ábra Tüskesor csatlakozó ESD védelme

3.1.9 Mérőjel előállító áramkör



3-10. ábra Mérőjel előállító áramkör

A 3-10. ábra tartalmazza a méréshez szükséges 1.5 MHz-es jeleket előállító áramkört. A méréshez DC komponenst is tartalmazó, háromszögjelet, négyszögjelet és szinuszjelet állítunk elő.

Az áramkör három alapkapcsolásból, egy invertáló integrátorból, egy Schmitt-triggerből és egy Sallen-Key szűrőből áll, melyek együttesen hozzák létre a kívánt jeleket.

Ha egy invertáló integrátor áramkör bemenetére konstans egyenfeszültséget kapcsolunk, akkor annak kimenetén lineárisan csökkenő feszültséget kapunk. Amennyiben ezt a lineárisan csökkenő feszültséget egy Schmitt-trigger bemenetére kapcsoljuk, majd annak kimenetét visszacsatoljuk az invertáló integrátorra, folyamatosan változó előjelű lineáris feszültségváltozást mérhetünk a kimeneten, azaz háromszögjelet állítottunk elő. Emellett a Schmitt-trigger kimenetén Vcc és GND között váltakozó egyenfeszültségeket mérhetünk, azaz négyszögjelet állítottunk elő.

A szinuszjel előállítására analóg aktív szűrőt alkalmazhatunk. Ismeretes, hogy periodikus jel leírható szinuszjelek végtelen sorának összegeként. Amennyiben, a jel periódus ideje által meghatározott alap frekvencia és az első felharmonikus frekvenciája közé helyezett törési frekvenciájú aluláteresztő szűrőt alkalmazunk a jelen, a szűrő kimenetén jó alakhűséggel rendelkező szinusz jelet kapunk.

A 3-10. ábra által ábrázolt kapcsolásban a létrehozott négyszögjel kerül szűrésre, egy Sallen-Key kapcsolás másodfokú aluláteresztő kapcsolás alkalmazásával.

Mindhárom alapkapcsoláshoz lineáris műveleti erősítőre van szükség, mellyel szemben a következő feltételeket határoztam meg:

- legalább 1.5 MHz működési sávszélesség
- aszimmetrikus táp lehetősége
- rail-to-rail működés
- alacsony fogyasztás

A választás a MCP6273 gyártói jelű integrált áramkörre esett, mely egy 170 µA fogyasztási móddal és aszimmetrikus táppal rendelkező 2 MHz sávszélességű rail-to-rail műveleti erősítőt tartalmaz. (HIVATKOZÁS)

Az erősítő engedélyezéséhez egy aktív alacsony jelet kell a mikrovezérlőnek előállítani. A $\overline{CS}$ lába az IC-nek 100 kΩ ellenálláson keresztül 3.3 V feszültségre van kötve.

Az mérőjelet előállító áramkör működéséhez szükség van az erősítéseket, frekvenciát és hiszterézist meghatározó passzív R, C elemekre.

A méretezés során figyelembe vettem, hogy 0603 és 0508 méretekkel rendelkező SMD alkatrészekkel történik a megvalósítás, illetve az E24-es szabványsor szerinti értékek érhetők el.

A mérőjel frekvenciáját meghatározó RC-tag a 3-10. ábra C8, R8 és R9 elemeiből tevődik össze.

A frekvencia a következő egyenlet szerint számítható:

$$f = \frac{1}{2\pi} \cdot \frac{1}{(R_8 + R_9) \cdot C_8} \cong 1520104.52\ \text{Hz} \approx 1.5\ \text{MHz}$$

A komparátor (Schmitt-trigger) billenési feszültségét a következő módon származtathatjuk:

$$IN_+ = U_{ref}$$

$$IN_+ = U_{trigangle} \cdot \frac{R_7}{R_{10} + R_7} + U_{square} \cdot \frac{R_{10}}{R_{10} + R_7}$$

$$U_{billenési} = (1 + \frac{R_{10}}{R_7}) \cdot U_{ref} - \frac{R_{10}}{R_7} \cdot U_{square}$$

,ahol $U_{ref} = 1.65V$, $R_7 = 68\ k\Omega$ *és* $R_{10} = 10\ k\Omega$

Ezek alapján a billenési feszültségek:

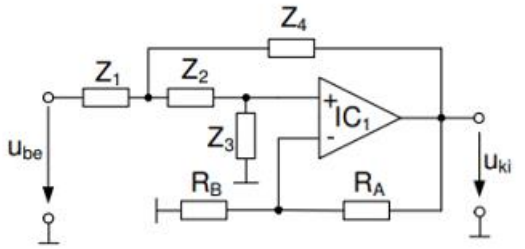
$$U_1 \cong 1.41V$$

$$U_2 \cong 1.89\ V$$

(HIVATKOZÁS)

A bevezetett hiszterézis ugyan a háromszögjel szimmetriáján ront, viszont biztosítja, hogy nem történhet oszcillálás a komparátor kimenetén.

A szinuszos mérőjel ellőállításához Sallen-Key alaptagot alkalmaztam, mellyel másodfokú aluláteresztő szűrőt valósítok meg. (HIVATKOZÁS)



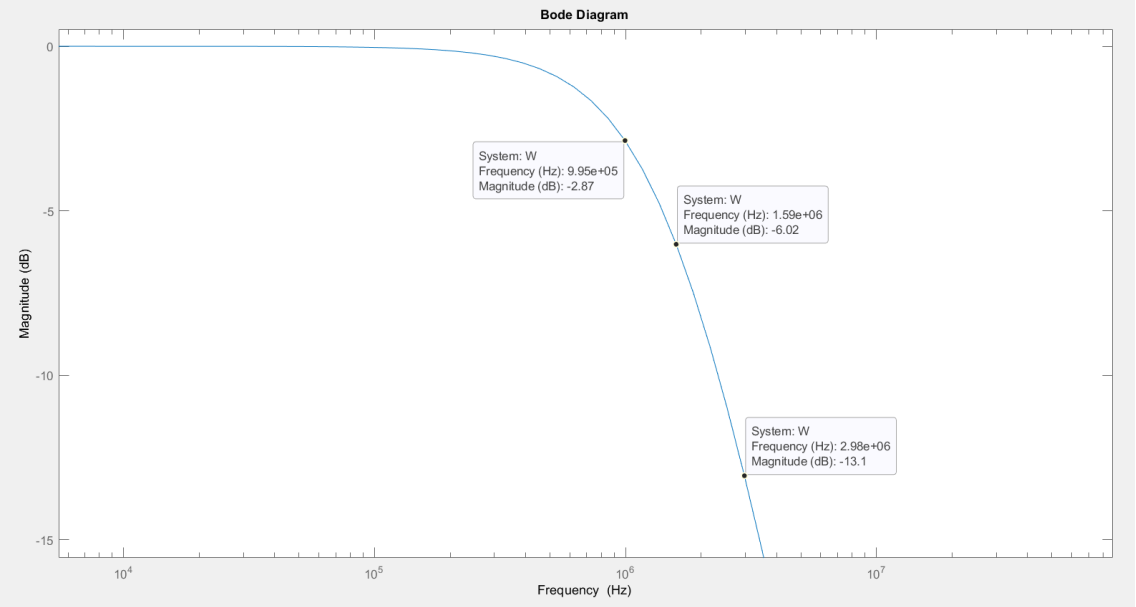
Z1	Z2	Z3	Z4	MEGVALÓSÍTOTT SZŰRŐ
R1	R2	C3	C4	Másodfokú aluláteresztő szűrő

3-11. ábra Sallen-Key alaptag

A 3-10. ábra szerinti kapcsolásban R_1 szerepét R_{11}, R_2-t R_{21}, C_3-t C_{10} és C_4-t C_9 veszi fel.

A kapcsolás átvitele a következő:

$$\frac{U_{ki}\,(s)}{U_{be}\,(s)} = \frac{\frac{1}{R_{11}C_{10}R_{21}C_9}}{s^2 + s\left(\frac{1}{R_{21}C_9} + \frac{1}{R_{10}C_9}\right) + \frac{1}{R_{11}C_{10}R_{21}C_9}}$$



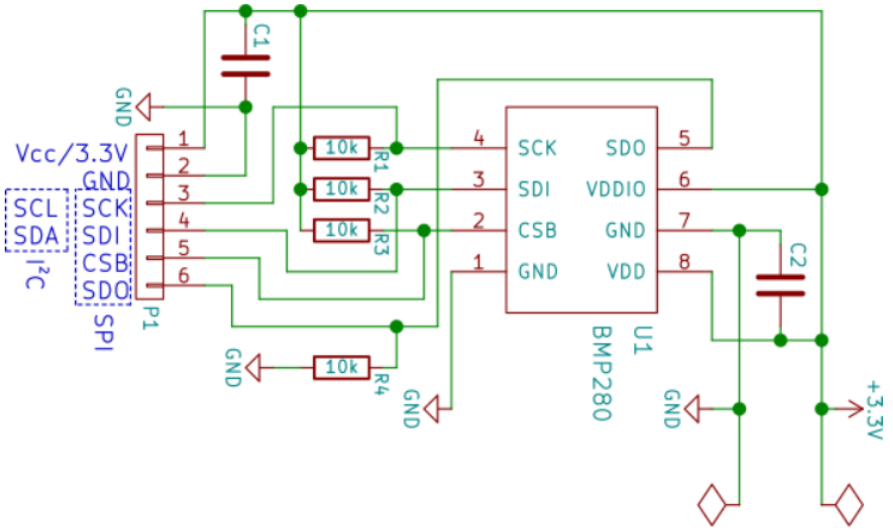
3.1.10 Hőmérséklet- és nyomásmérő áramkör

Az öntözőrendszer számára fontos, hogy a környezeti hőmérsékletet (elsősorban talajközeli levegőréteg hőmérsékletét) legalább 1 ºC pontossággal mérje, annak érdekében, hogy fagy vagy fagyközeli levegőhőmérséklet esetén, ne kapcsolja be az öntözést, még a talaj nedvességtartalmának csökkenése esetén sem.

A levegő nyomásának mérésével következtethetünk az időjárás változására például a nyári záporok előtt nyomásváltozás történik. (HIVATKOZÁS)

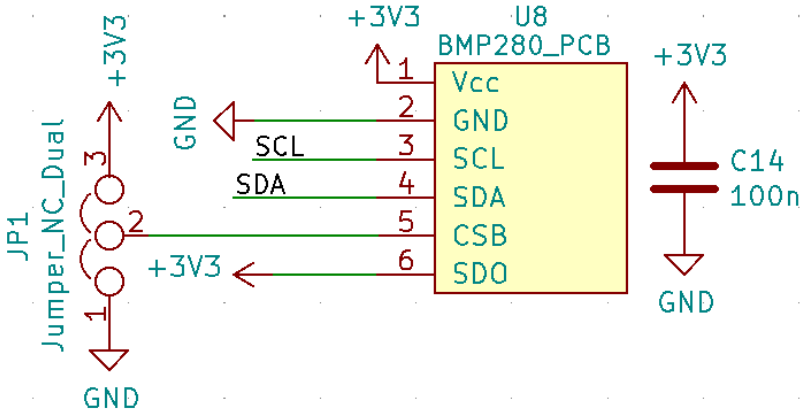
Erre a feladatra a BMP280 elnevezésű szenzort használtam. Amellett, hogy megfelelő pontossággal mér hőmérsékletet és nyomást, kapható ún. breakout boardon is, így külön kis NYÁK-ra került a szenzor és a nagyáramot is szállító NYÁK hőmérséklete kevésbé befolyásolja a hőmérséklet mérést.

A szenzort I2C (opcionálisan SPI) buszon lehet elérni 0x76 és 0x77 címen, melyen hardveresen lehet állítani, egy jumper segítségével. A kapcsolás belső felépítését a 3-12. ábra, a kliens NYÁK-hoz csatlakozását a 3-13. ábra mutatja.



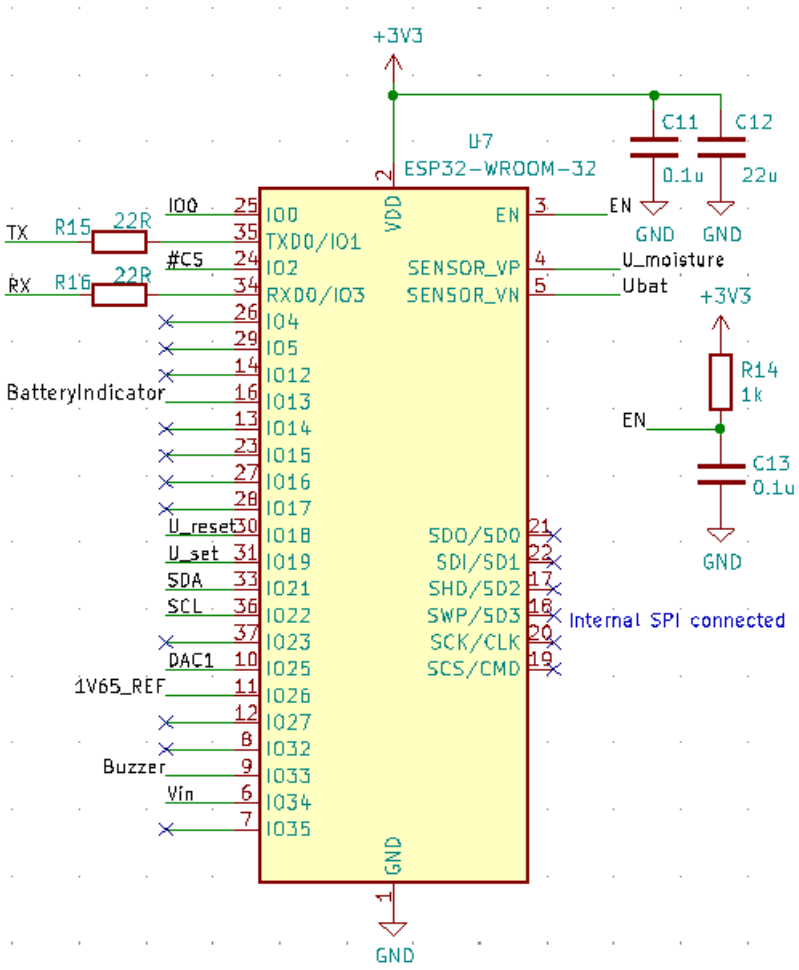
3-12. ábra BMP280 PCB belső felépítése

https://startingelectronics.org/pinout/GY-BMP280-pressure-sensor-module/ (HIVATKOZÁS)



3-13. ábra BMP280 PCB külső áramköre

3.1.11 ESP32 WROOM-32 periféria áramkörei



3-14. ábra ESP32 WROOM-32

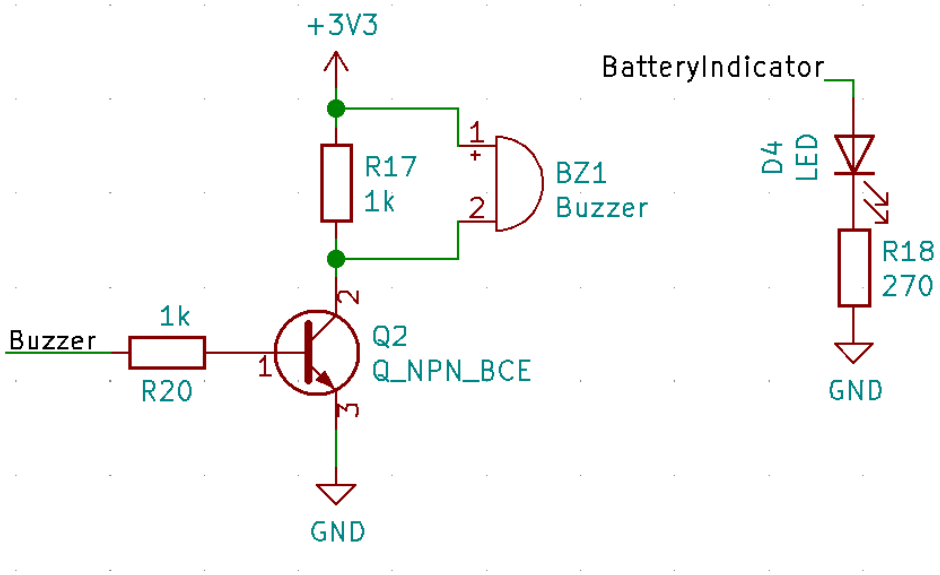
Az ESP32 modul ezen alkalmazáshoz szükséges periféria áramkörök nagy részét tartalmazza, így nem volt szükség sok külső alkatrész csatlakoztatásához.

A 3-14. ábra C_{11} és C_{12} hidegítő kondenzátorok a gyártó által javasolt méretekkel rendelkeznek.

R_{14} és C_{13} az engedélyező jel (EN) hullámosságának csökkentésére szolgál.

Az általam használt WROOM-32 modul tartalmaz 4 MB flash memóriát, amely SPI buszon keresztül összeköttetésben áll a processzorral. Emiatt a modul 17-22 lábait nem szabad semmilyen külső egységhez kötni.

3.1.12 Egyéb funkciókat ellátó áramkörök



3-15. ábra Buzzer és LED meghajtó áramkörök

A 3-15. ábra tartalmazza a keresési funkcióhoz szükséges buzzer meghajtó áramkört, illetve az akkumulátor indikátor áramkört.

A LED 5 mA-es árammal és 2 V nyitófeszültséggel lett méretezve.

A buzzer áramkört 20-20 kHz közötti négyszögjellel kell meghajtani.

3.2 NYÁK terv

Net Classes

Name	Clearance	Track Width	Via Size	Via Drill	µVia Size	µVia Drill	dPair Width	dPair Gap
Default	0.25 mm	0.25 mm	0.9 mm	0.7 mm	0.3 mm	0.1 mm	0.25 mm	0.25 mm
Power	0.25 mm	0.3048 mm	1.5 mm	1 mm	0.3 mm	0.1 mm	0.25 mm	0.25 mm

4 Szoftver tervezés

4.1 Kliens beágyazott szoftvere

4.2 Központi egység szoftvere

4.2.1 SWAN szerver szoftveres környezete

A SWAN szerver egy Raspberry Pie Zero W single board computeren fut. A RPI hivatalos operációs rendszerét DietPi szoftverre cseréltem, annak érdekében, hogy minél kevesebb erőforrást használjon el az operációs rendszer és több maradjon az MQTT broker futtatására és a webes felület kiszolgálására.

https://www.raspberrypi.org/products/raspberry-pi-zero-w/?resellerType=home

https://dietpi.com/dietpi-software.html

A teljes rendszerhez való hozzáférés SSH-n keresztül vagy monitor és USB billentyűzet csatlakoztatásával lehetséges. Ezen felül az eszköznek kiosztott IP címen keresztül különböző telepített szoftverek kezelőfelületei is elérhetőek lokális hálózatról.

A teljes szoftveres felépítéshez egy publikusan elérhető SD kártya image-t használtam fel, amely több használt alkalmazás szoftverét előre telepített és „élesztett" állapotban tartalmazta.

https://www.sensorsiot.org/diet-pi-supporting-material-videos-126-and-128/

4.2.2 MQTT broker

Az MQTT protokoll ideális IoT rendszerek adatainak közvetítésére, mivel kis erőforrás és kód igényű, megbízható és kétirányú kommunikációt valósít meg.

Az MQTT Publish/Subscribe architektúrát leíró protokoll, amelyben alap esetben egy broker és több kliens közötti kommunikációt tesz lehetővé.

Minden üzenet küldés akár egy kliens, akár a bróker küldi egy adott témába (topic) kerül publikálásra. A kliensek feliratkoznak egyes témákra, amelyet a bróker jegyez és abban az esetben, ha arra üzenet érkezik minden feliratkozott kliensnek közvetíti. A brókeren keresztül megy minden üzenet, nincs lehetőség kliens-kliens közvetlen kapcsolatra.

A választásom a brókert megvalósító szoftverre, a mosquitto mqtt broker, amely egy nyílt forráskódú lightweight megoldás.

https://mosquitto.org/

4.2.3 SQLite

A kliensek felől érkezett adatokat tárolni kell a szerveren, hogy azokat feldolgozhassuk, illetve megjelenítsük a felhasználó számára. Nagy mennyiségű adat rendszerezett tárolására az egyik legelterjedtebb módszer a relációs adatbázisok alkalmazása. Az adatbázisokhoz való hozzáférés szabványosan Structured Query Language (SQL) nyelv segítségével tehetjük meg.

A SWAN szerver SQLite adatbázis motort alkalmazza, mely nyílt forráskódú lightweight megoldást nyújt. Lekérdezések mellett böngészőben megnyitott felületen is van lehetőségünk elérnünk az adatbázis kezelő motort. Ilyen módon került létrehozásra az a két tábla (SWAN,serialnumbers), amelybe adatokat szúr be a szerver működése során.

CREATE TABLE 'SERIALNUMBERS' ('key' INTEGER PRIMARY KEY NOT NULL, 'timestamp' DATETIME DEFAULT CURRENT_TIMESTAMP)

CREATE TABLE 'SWAN' ('primkey' INTEGER PRIMARY KEY NOT NULL, 'id' INTEGER, 'moisture' REAL, 'temperature' REAL, 'pressure' REAL, 'status' INTEGER, 'battery' REAL, 'timestamp' DATETIME DEFAULT CURRENT_TIMESTAMP)

https://www.sqlite.org/index.html

https://vik.wiki/images/f/f0/Info2_Adatb%C3%A1zisok_jegyzet_%28r%C3%A9gi%29.pdf

4.2.4 Node-RED

A SWAN szerver három fő feladatot lát el, felhasználói felületet biztosít a mért adatok megtekintéséhez és a kliensekhez való hozzáférésnek, MQTT brókert futtat és adatbázisban tárolja a mért adatokat.

A három fő funkció összefésülésére egy IoT rendszerekben gyakran használt programozási eszközt használtam, a Node-RED alkalmazást.

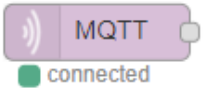
https://nodered.org/

Az alkalmazás lehetővé teszi, hogy úgynevezett node-k összekötésével egyszerű és átlátható módon készítsünk backend-et és frontend-et összekötő szoftvert. Lehetővé teszi, hogy adatáramlást figyelve programozzunk, magas absztrakciós „nyelven”. Node.js alapon működik, amely egy esemény vezérelt modellű lightweight szoftverrendszer.

https://nodejs.org/en/

Az alkalmazást, úgynevezett flow-t, böngészőben elérhető grafikus felületen érhetjük el. A következőben ismertetem a SWAN szerver elkészült szoftverében szereplő node-okat általánosan, majd külön alfejezetben a konkrét programot.

MQTT input node:



Feladata a MQTT brókeren keresztül érkező üzenet fogadása egy megadott témában. Tekinthetjük úgy, hogy a node-red flow így iratkozik fel egy témára. Legfontosabb kimenete, msg.payload amely az üzenetet tartalmazó karakterlánc.

MQTT output node:

Kimenetként az MQTT bróker felé továbbít egy üzenetet egy megadott témával. Tekinthetjük úgy, hogy a node-red flow így publikál egy adott témában. Legfontosabb kimenetei; msg.payload és msg.topic. A node beállításakor a Quality of Service és Retain flag beállítására is ügyelnünk kell.

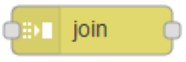
JSON node:

JSON objektumot karakterláncá vagy karakterláncot JSON objektummá alakít. msg.payload attribútumon kívül mást nem módosít.

Function node:

Bármilyen javascript programot tartalmazhat ez a node. Kimenetét return kulcsszóval vagy node.send() függvénnyel küldheti tovább a következő node-nak. A kimeneti üzenet a node-red konvenciónak megfelel, amennyiben a függvény blokk nem állítja a kapott msg attribútumait azokat tovább adja a következő node számára.

Join node:

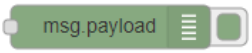


Több node által küldött üzeneteket egyesíthetünk ezzel a node-dal. Beállítása szerint az üzenetek száma, időkorlát vagy msg.complete attribútum bebillentése okozza az addigi üzenetek összekapcsolását. Szintén beállítható az egyesítés módja (tömb, kulcs-érték objektum vagy objektum összefonás).

SQLite node:

Az adatbázissal kapcsolatot teremtő node. A bemenetére érkezett üzenet témáját (msg.topic) kérdezi le az adatbázistól. Az adatbázis válaszát a kimenetre üzenetként továbbítja (msg.payload).

Debug node:

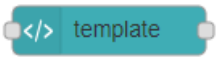


Fejlesztéshez elengedhetetlen, hogy a debug fülön nyomonkövethető legyen az adatáramlás. Ez a node a bemenetét a debug fülre továbbítja, illetve időbélyeggel is ellátja azt. A jobb oldalán található zöld gombbal ki/be kapcsolható hatása, a könnyebb fejlesztés érdekében.

Inject node:

Beállításai szerint induláskor vagy periodikusan vagy a fejlesztő felületen megnyomásakor az aktuális időbélyeget küldi ki üzenetként. Debugolásra vagy timer megvalósításra alkalmas node.

UI template node:



A felhasználói felületet UI node-k reprezentálják. A template node bármilyen HTML állomány fogadására képes. A HTML állományban leírt objektum a node-hoz rendelt dashboard elemen jelenik meg.

Chart node:

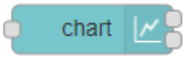
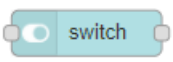


Chart node segítségével egyszerűen hozhatunk létre dinamikusan formálódó kétdimenziós ábrát. A bemenetén kapott Y értékeket automatikusan árbrázolja az aktuális időbélyeg szerint egy kétdimenziós ábrán. A bejövő üzenet témája (msg.topic) külön ábrázolja az adatokat. Állapotát nem őrzi meg, minden újraindítás után üresen indul az ábra. Lehetőség van a Chart.js könyvtár dokumentációja szerinti paraméterezésre, amellyel egyszerre több X-Y pár megadása is lehetséges (pl. régebbi adatok betöltése érdekében). Ezt a lehetőséget a SWAN szerveren futó node-red verzióval viszont nem sikerült kihasználni. https://www.chartjs.org/

Dashboard Switch node:

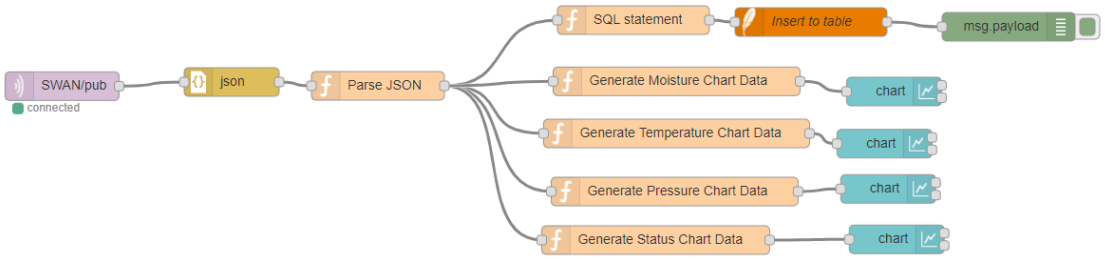


A felhasználói felületen egy kapcsoló jelenik a beállított dashboard elemen. A kapcsolóra való kattintás során a node előre beállított üzenetet továbbít.

4.2.5 SWAN SERVER flow

A szervert megvalósító node-red flow négy különálló részre bontható. Az adatokat fogadó és ábrázoló részre, új kliensek számára sorszámot szolgáltató részre, a felhasználói felületet kezelő részre és az adatbázist karbantartó/lekérdező részre.

A 4-1. ábra az adatokat fogadó és ábrázoló részt megvalósító node-k közötti kapcsolatok tartalmazza.



4-1. ábra Mérési adatok mentése és ábrázolása

MQTT

Feladat:	SWAN/pub témájú üzenetek fogadása.
Beállítás:	server: localhost:1883, QoS2 annak érdekében, hogy legfeljebb egyszer fogadjunk minden üzenetet.

Parse JSON

Feladat:	A hálózaton érkező üzenet JSON objektumot karakterlánc formátumra hozta az előző JSON node. A Parse JSON függvény feladata külön változókba való adatmentés, illetve a bizonyos adatok átlagolása és két számjegyre kerekítése.
Kód:	1 var avgtemp; 2 var avgmoist; 3 var avgpres; 4 var status= msg.payload.status; 5 var id=msg.payload.Client;

```
var battery =
msg.payload.Battery_Voltage;

avgtemp = (msg.payload.temp[0]+
msg.payload.temp[1]+
msg.payload.temp[2]+
msg.payload.temp[3]+
msg.payload.temp[4])/5;
avgpres = (msg.payload.pressure[0]+
msg.payload.pressure[1]+
msg.payload.pressure[2]+
msg.payload.pressure[3]+
msg.payload.pressure[4])/5;
avgmoist = (msg.payload.U_moisture[0]+
msg.payload.U_moisture[1]+
msg.payload.U_moisture[2]+
msg.payload.U_moisture[3]+
msg.payload.U_moisture[4])/5;



temp = avgtemp.toFixed(2);
pres = avgpres.toFixed(2);
moist = avgmoist.toFixed(2);

var data = {payload:
      [id,moist,temp,pres,status,battery]};

return data;
```

Function

SQL statement (insert payload)

Feladat:	A beérkező adatokból SQL állítás összeállítása.

Kód:

```
var topic = " INSERT INTO \"SWAN\"
(\"ID\"\,\"moisture\"\,\"temperature
\"\,\"pressure\"\,\"status\"\,\"battery\")
VALUES ('"+msg.payload[0]+"\'\,\'"+msg.payload[1]
+"\'\,\'"+msg.payload[2]+"\'\,\'"+msg.payload[3]+
"\'\,\'"+msg.payload[4]+"\'\,\'"+msg.payload[5]+"\')";
var newmsg={}
newmsg.topic = topic;
return newmsg;
```

Function

Generate Moisture Chart Data

Feladat:	Chart node számára szükséges formátum előállítása.

Kód:

```
var msg1 = { payload:msg.payload[1],
topic:msg.payload[0] };
```

	2 3 return msg1;

Generate Temperature Chart Data

Feladat:	Chart node számára szükséges formátum előállítása.
Kód:	1 var msg1 = { payload:msg.payload[2], topic:msg.payload[0] }; 2 3 return msg1;

Generate Pressure Chart Data

Feladat:	Chart node számára szükséges formátum előállítása.
Kód:	1 var msg1 = { payload:msg.payload[3], topic:msg.payload[0] }; 2 3 return msg1;

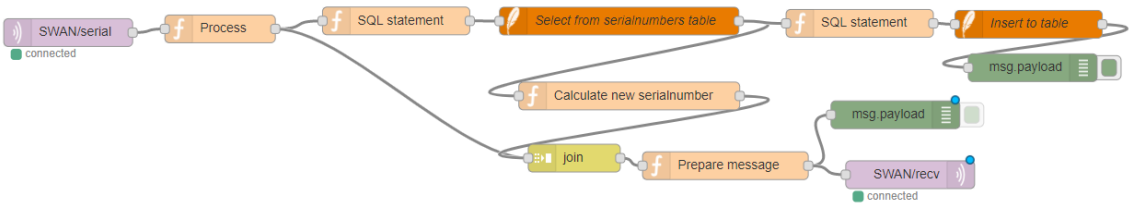
Generate Status Chart Data

Feladat:	Chart node számára szükséges formátum előállítása.
Kód:	1 var msg1 = { payload:msg.payload[4], topic:msg.payload[0] }; 2 3 return msg1;

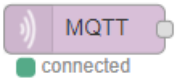
Chart

Feladat:	A felhasználói felületen kétdimenziós ábra megjelenítése.
Beállítás:	X-tengely HH:mm:ss formátummal definiált, az elmúlt egy napban érkezett adatokat, de legfeljebb 1000 pontot ábrázolunk.

A 4-2. ábra a kliensek sorszámát generáló részt megvalósító node-k közötti kapcsolatok tartalmazza.



4-2. ábra Kliensek sorszám generálása



MQTT

Feladat:	SWAN/serial témájú üzenetek fogadása.
Beállítás:	server: localhost:1883, QoS2 annak érdekében, hogy legfeljebb egyszer fogadjunk minden üzenetet.

Process

Feladat:	A beérkezett karakterlánc egy egész számot tartalmaz. Ez a függvény a karakterlánc és egész szám közötti konverziót végzi.

Kód:	`msg.payload = parseInt(msg.payload);`
	
	`return msg;`

SQL statement (Select maxkey)

Feladat:	A függvény feladata, hogy SQL node számára előállítsa a SQL lekérést. newmsg.topic tartalmazza az érvényes SQL állítást. A serialnumbers táblából SELECT utasítással a key oszlop értékeit kérjük le. MAX() függvénnyel módosítjuk a lekérést, ezáltal csak az oszlop legnagyobb értékét kapjuk vissza. A létrejövő MAX(key) javascript objektumot hibás szintaxisú, ezért AS kulcsszóval új nevet adunk neki, maxkey, amely már javascript kompatibilis.

Kód:	1 var topic = "SELECT MAX(key) AS maxkey FROM serialnumbers"; 2 3 var newmsg={} 4 newmsg.topic = topic; 5 return newmsg;

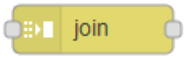
SQL statement (inject newkey)

Feladat:	A beérkező adatokból SQL állítás összeállítása.
Kód:	1 var tempnum = msg.payload[0].maxkey+1; 2 serialnumber = tempnum.toString(); 3 4 5 var topic = " INSERT INTO \"SERIALNUMBERS\" (\"key\") VALUES ('"+serialnumber+"\')"; 6 var newmsg={} 7 newmsg.topic = topic; 8 return newmsg;

Calculate new serialnumber

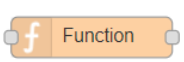
Feladat:	A lekérdezés a jelenlegi legnagyobb sorszámot adja vissza. Az új sorszám eggyel nagyobb lesz.

Kód:	1 if (msg.payload[0].maxkey === null){
	2 msg.payload = 1;}
	3 else{
	4 msg.payload = msg.payload[0].maxkey + 1
	5 };
	6 return msg;



Join

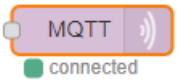
Feladat:	Process és Calculate new serialnumber függvények üzeneteit tömbbe tömöríti.
Beállítás:	Két üzenet beérkezése után üzenet küldése. Üzenet formátuma tömb.



Prepare message

Feladat:	A beérkező tömb két elemét , karakterrel elválasztva karakterláncként tovább küldeni .

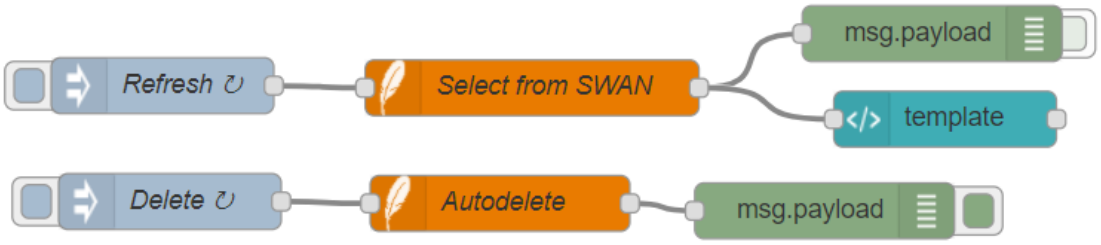
Kód:	`msg.payload = msg.payload[0].toString() + "\," + msg.payload[1].toString();`
	
	`return msg;`



MQTT

Feladat:	SWAN/recv témájó üzenet publikálása. A SWAN SERVER flow-ban több helyen is.
Kód:	server: localhost:1883, QoS1 annak érdekében, hogy legalább egyszer elküldjünk minden üzenetet. Retain:false, mivel fontos elkerülni, hogy egy kliens az energiacsökkentett állapotából visszatérve téves üzenetet kapjon, mert a bróker új feliratkozóként regisztrálja.

A 4-3. ábra az adatbázist karbantartó részt megvalósító node-k közötti kapcsolatok tartalmazza.



4-3. ábra Adatbázis karbantartása

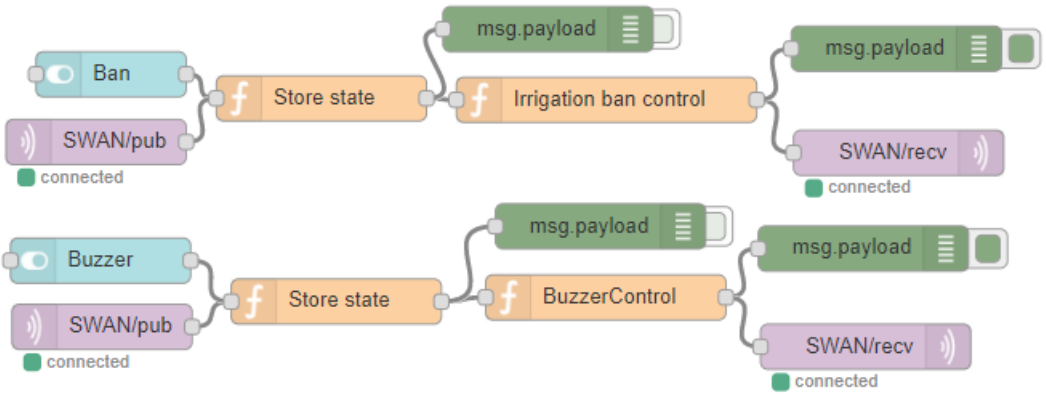
Refresh

Feladat:	Friss adatok lekérdezése percenként.
Beállítás:	Inject at start once? és perces ciklusidő beállítva. Topic: *SELECT * FROM SWAN ORDER BY primkey DESC*

Delete

Feladat:	Óránként a 30 napnál régebbi adatok törlése az adatbázisból.
Kód:	Egy órás ciklusidő beállítva. Topic: *DELETE FROM SWAN WHERE TIMESTAMP <= strftime('%s','now', '-30 days')*1000*

A 4-4. ábra az felhasználó felületet kezelő részt megvalósító node-k közötti kapcsolatok tartalmazza. Az azonos nevű node-k azonos funkciót látnak el, ezért a már korábban bemutatott node-okat nem kerülnek újra bemutatásra.



4-4. ábra Felhasználói felület kezelése

Store state

Feladat:	A switch node állapota nem lekérdezhető, ezért valamilyen javascript nyelvi eszközzel minden állapot változást követni kell. Erre a kontextus a megfelelő eszköz. Ez a függvény elmenti a switch állapotát, majd minden futása során a kimeneten továbbítja azt.

Kód:	`var state = context.get('state') || false;`
	
	`if (msg.topic == 'switch') {`
	`  state = msg.payload;`
	`  context.set('state', state);`
	`}`
	
	`return {payload: state};`

Irrigation ban control

Feladat:	A felhasználói felületen található kapcsoló állapotáról az éppen a hálózatra jelentkező kliensnek tudnia kell. Ez a függvény a kapcsoló állapota szerinti karakterláncot továbbít az MQTT node-nak.

Kód:	`if (msg.payload === true){`
	`  msg.payload = "banned";`
	`}`
	`else {`
	`  msg.payload = "enabled";`
	`}`
	
	`return msg;`

Buzzer Control

Feladat:	A felhasználói felületen található kapcsoló állapotáról az éppen a hálózatra jelentkező kliensnek tudnia kell. Ez a függvény a kapcsoló állapota szerinti karakterláncot továbbít az MQTT node-nak.

Kód:	`if (msg.payload === true){`
	`  msg.payload = "lost";`
	`}`
	`else {`
	`  msg.payload = "found";`
	`}`
	`return msg;`

Table

ID		Moisture Voltage [V]	Temperature [°C]	Pressure [hPa]	Status	Timestamp
1	1		23.27	1010.05	0	2020-11-29 21:07:38
1	1		22.6	1010	0	2020-11-29 20:51:53
1	1		21.85	1010.11	0	2020-11-29 20:36:10
1	1		22.56	1010.09	0	2020-11-29 20:20:27
1	1		23.18	1010	0	2020-11-29 20:04:45
1	1		23.73	1009.92	0	2020-11-29 19:49:03
1	1		24.13	1009.78	0	2020-11-29 19:33:21
1	1		24.09	1009.67	0	2020-11-29 19:17:37
1	1		24.04	1009.58	0	2020-11-29 19:01:53
1	1		23.98	1009.63	0	2020-11-29 18:46:09
1	1		23.89	1009.59	0	2020-11-29 18:30:25
1	1		23.92	1009.43	0	2020-11-29 18:14:42
1	1		24.02	1009.46	0	2020-11-29 17:58:58
1	1		24.17	1009.41	0	2020-11-29 17:43:15
1	1		24.15	1009.38	0	2020-11-29 17:27:31
1	1		23.81	1009.26	0	2020-11-29 17:11:47
1	1		23.47	1009.27	0	2020-11-29 16:56:05
1	1		23.43	1009.19	0	2020-11-29 16:40:26
1	1		23.43	1009.17	0	2020-11-29 16:38:36
1	1		23.42	1009.13	0	2020-11-29 16:36:20
1	1		23.37	1009.06	0	2020-11-29 16:30:34
1	1		23.36	1009.05	0	2020-11-29 16:29:15
1	1		23.36	1009.04	0	2020-11-29 16:28:26
1	1		23.3	1008.94	0	2020-11-29 16:17:43
1	1		23.29	1008.91	0	2020-11-29 16:15:25
1	1		23.28	1008.94	0	2020-11-29 16:13:07
1	1		23.27	1008.88	0	2020-11-29 16:10:49

5 Validációs mérés

5.1 NYÁK bemérése

5.2 Szenzor bemérése

6 Irodalomjegyzék

[1] KSH, A lakott lakások lakóövezeti jelleg szerint (2001)
http://www.ksh.hu/nepszamlalas/docs/tablak/lakasviszonyok/12_02_28.xls

[2] Önálló laboratórium projekt
https://github.com/Zaion-BM/SWAN

[3] Mágnesszelep működése
https://en.wikipedia.org/wiki/Solenoid_valve (2020.10.15)

[4]

[5]

[6]

[7]

[8]

[9]

[10]

[11]

[12]

[13]

[14]

[15]

7 Függelék

Sallen_Key_filter.m

```
%% Sallen-Key 2nd order Low Pass Filter transfer function

    %Parameters
    R10 = 1.E4;
    R21 = 1.E4;
    C10 = 1.E-11;
    C9 = 1.E-11;

    %Transfer function
    W = tf((1/(R10*C10*R21*C9)),[1, 1/(R21*C9)+1/(R10*C9), ...
        1/(R10*C10*R21*C9) ]);

    %Bode diagram solely with amplitude
    h = bodeplot(W);
    setoptions(h,'FreqUnits','Hz','PhaseVisible','off');
```

buzzer.h

```
#ifndef BUZZER_H
#define BUZZER_H

void buzzer_TASK(void *pvParameters);

#endif /*BUZZER_H*/
```

buzzer.ino

```
#include "Buzzer.h"

void buzzer_TASK(void *pvParameters){
    const byte buzzerInputGPIO = 33; //GPIO connected to buzzer
    pinMode(buzzerInputGPIO, OUTPUT); //Select pinmode
    vTaskDelay(1000); //yield

    Serial.println("Buzzer task starts.");
    while(lost){ //If lost, than generate squarewave (~400 Hz)
         digitalWrite(buzzerInputGPIO, HIGH);
         vTaskDelay(1);
         digitalWrite(buzzerInputGPIO, LOW);
         vTaskDelay(1);
      }
    while(1){
      vTaskResume(sleep_TaskHandle); //Resume sleepTask
      Serial.println("Buzzer task ends.");
      vTaskDelay(delayTime*2);
     }
}
```

Sleep.h

```
1 #ifndef SLEEP_H
2 #define SLEEP_H
3
4 void sleep_TASK(void *pvParameters);
5
6 #endif /*SLEEP_H*/
```

Sleep.ino

```
 1 #include "Sleep.h"
 2
 3 void sleep_TASK(void *pvParameters){
 4   while(1){
 5     vTaskDelay(100);//yield
 6     Serial.println();
 7     Serial.println(F("SWAN Client enters deep sleep ..."));
 8     //Enable and configure wakeup cause
 9     esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP *
   uS_TO_S_FACTOR);
10     //Go to deep sleep on both cores
11     esp_deep_sleep_start();
12   }
13 }
```

MQTT.h

```
1 #ifndef MQTT_H
2 #define MQTT_H
3
4 void MQTT_TASK(void *pvParameters);
5
6 #endif /*MQTT_H*/
```

MQTT.ino

```
    1     #include "MQTT.h"
    2     #include <WiFi.h>
    3     #include <PubSubClient.h>
    4     #include <ArduinoJson.h>
    5
    6     /*Credential for wifi and mqtt broker IP*/
    7     const char* ssid = "****";
    8     const char* password = "****";
    9     const char* mqtt_server = "192.168.0.202";
   10
   11     // Callback function called when MQTT message is received
   12     void callback(char* topic, byte* message,
          unsigned int length) {
```

```
13          Serial.print("Message arrived on topic: ");
14          Serial.print(topic);
15          Serial.print(". Message: ");
16          String messageTemp;
17          //Process  and log message
18          for (int i = 0; i < length; i++) {
19            Serial.print((char)message[i]);
20            messageTemp += (char)message[i];
21          }
22          Serial.println();
23
24          //Set and log associated variables
25          if( messageTemp == "banned") {
26            Serial.print("I'm banned");
27            Serial.println();
28            banned = true;
29            }
30          if( messageTemp == "enabled") {
31            Serial.print("I'm enabled");
32            Serial.println();
33            banned = false;
34            }
35          if( messageTemp == "lost") {
36            Serial.print("I'm lost");
37            Serial.println();
38            lost = true;
39            }
40          if( messageTemp == "found") {
41            Serial.print("I'm found");
42            Serial.println();
43            lost = false;
44            }
45        }
46
47      void MQTT_TASK(void *pvParameters){
48        vTaskDelay(10); //yield
49        Serial.println();
50        Serial.print("Connecting to ");
51        Serial.println(ssid);
52
53        //Start wifi connection
54        WiFi.begin(ssid, password);
55
56        while (WiFi.status() != WL_CONNECTED) {
57            //If not connected retry every 0.5s
58            vTaskDelay(500);
59            Serial.print(".");
60         }
61
62        /*Log wifi information*/
63        Serial.println("");
64        Serial.println("WiFi connected");
65        Serial.println("IP address: ");
66        Serial.println(WiFi.localIP());
67
68        /*Set up connection with MQTT broker*/
```

```
69        client.setServer(mqtt_server, 1883);
70        client.setCallback(callback);
71        client.connect("SWAN1");
72        client.subscribe("SWAN/recv");
73        Serial.println("MQTT connected");
74
75        char payload[512]; //Buffer array
76
77        //Send messages only after mutex is taken
78        if(xSemaphoreTake( measurementMutex, ( TickType_t ) 10 )
          == pdTRUE){
79          Serial.println("Publishing...");
80
81          //"Creating JSON"
82          StaticJsonDocument<512> doc;
83
84          doc["Client"] = serialNumber;
85          doc["Battery_Voltage"] = Ubat;
86          doc["status"] = irrigatonStatus;
87
88          JsonArray temperature = doc.createNestedArray("temp");
89          for(int i=0;i<5;i++){
90            temperature.add(temp[i]);
91          }
92
93          JsonArray pres = doc.createNestedArray("pressure");
94          for(int i=0;i<5;i++){
95            pres.add(pressure[i]);
96          }
97          JsonArray U_mois = doc.createNestedArray("U_moisture");
98          for(int i=0;i<5;i++){
99            U_mois.add(U_Moisture[i]);
100         }
101
102         //Generate string from JSON
103         serializeJson(doc, payload);
104         //Publish payload to /pub topic
105         client.publish("SWAN/pub", payload);
106
107         //UART logging
108         Serial.print(payload);
109         Serial.println();
110
111         //Give mutex
112         xSemaphoreGive(measurementMutex);
113
114         //yield
115         vTaskDelay(delayTime/2);
116
117         //Resume buzzerTask
118         vTaskResume(buzzer_TaskHandle);
119       }
120       while(1){vTaskDelay(delayTime);}//yield
121     }
```

Irrigation.h

```h
#ifndef IRRIGATION_H
#define IRRIGATION_H

void irrigation_TASK(void *pvParameters);

#endif /*IRRIGATION_H*/
```

Irrigation.ino

```ino
#include "Irrigation.h"

void irrigation_TASK(void *pvParameters){
    const byte resetPulse = 18; //GPIO connected to optocoupler
    const byte setPulse = 19; //GPIO connected to optocoupler
    pinMode(resetPulse, OUTPUT); //Set pinmode
    pinMode(setPulse, OUTPUT);  //Set pinmode
    float limit = 1.5;

    Serial.println("Irrigation task starts.");

    while(1){
        if(frozen || banned) {//Turn off valve
          digitalWrite(setPulse, LOW);
          digitalWrite(resetPulse, HIGH);
          irrigatonStatus = 0;
          Serial.println("Irrigation task end(OFF).");
          vTaskDelay(delayTime);
         }
        else{
          //Turn on valve if average sensor voltage is under limit
          if(((U_Moisture[0]+U_Moisture[1]+U_Moisture[2]+U_Moisture[3]+U_Moisture[4])/5) > limit ){
            digitalWrite(resetPulse, LOW);
            digitalWrite(setPulse, HIGH);
            irrigatonStatus = 1;
            Serial.println("Irrigation task end(ON).");
            vTaskDelay(delayTime);
          }
          else{//Turn off valve if average sensor voltage is above limit
              digitalWrite(setPulse, LOW);
              digitalWrite(resetPulse, HIGH);
              irrigatonStatus = 0;
              Serial.println("Irrigation task end(OFF).");
              vTaskDelay(delayTime);
            }
          }
      }
 }
```

BMP280_task.h

```
#ifndef BMP280_TASK_H
#define BMP280_TASK_H

void BMP280_TASK(void *pvParameters);
void printValues(int);
void BMP280_Sleep(int device_address);

#endif /*BMP280_TASK_H*/
```

BMP280_task.ino

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>

#include "BMP280_task.h"

Adafruit_BMP280 bmp; //variable for sensor
int device_address = 0x76; //I2C address

RTC_DATA_ATTR float temp[5]; //Temperature values [°C]
RTC_DATA_ATTR float pressure[5]; //Pressure values [hPa]


void BMP280_TASK(void *pvParameters){
    unsigned long retryTime = 300000;   //5*6*10*1000*1ms = 5min
    bool status; //I2C device status
    float avgTemp; //Average temperature [°C]

    status = bmp.begin(device_address);  //Start I2C
communication

    if (!status) {
        Serial.println("Could not find sensor!");
        while (1){//Periodically attempt to find sensor
          vTaskDelay(retryTime);
          };
    }

    //Sleep to save power
    BMP280_Sleep(0x76);

    while(1) {
    Serial.println();
    Serial.println("BMP280_TASK begins");

    bmp.begin(device_address);
    Serial.println("BMP280 to Normal mode...");

    if(measurementCounter <5){//Read measurements in normal mode
       temp[measurementCounter] = bmp.readTemperature();
```

```
39        pressure[measurementCounter] = bmp.readPressure() /
40 100.0F;
41       }
42
43     for(int i=0;i<5;i++){
44       avgTemp += temp[i]; //Calculate average temperature
45      }
46     avgTemp = avgTemp / 5;
47
       //If temperature is under 5°C, ambient temperature can be
48 below 0°C
49     if( (avgTemp - 5.0)< 0.01 ){frozen = true;}
50
51     printValues();
52
53     //Sleep to save power
54     BMP280_Sleep(device_address);
55     Serial.println("BME280 to Sleep mode...");
56
57     Serial.println("BMP280_TASK ends");
58     vTaskDelay(delayTime);
59   }
60 }
61
62 void printValues() {//UART logging
63     Serial.print("Temperature:  ");
64     for(int i=0; i<5;i++){
65       Serial.print(temp[i]);
66       Serial.print(" *C ");
67     }
68     Serial.print(", ");
69
70     Serial.print("Pressure:  ");
71     for(int i=0; i<5;i++){
72       Serial.print(pressure[i]);
73       Serial.print(" hPa ");
74     }
75     Serial.println();
76 }
77
78 void BMP280_Sleep(int device_address) {
79   // Mode[1:0]  Mode
80   //    00      'Sleep'  mode
81   //  01 / 10   'Forced' mode, use either '01' or '10'
82   //    11      'Normal' mode
83
84   Wire.beginTransmission(device_address);
85   Wire.write((uint8_t)0xF4);
86   Wire.write((uint8_t)0b00000000);
87   Wire.endTransmission();
   }
```

ADC.h

```
1 #ifndef ADC_H
```

```
2 #define ADC_H
3
4 void ADC_TASK(void *pvParameters);
5
6 #endif /*ADC_H*/
```

ADC.ino

```
1   #include "ADC.h"
2
3    float Vin= 0.0; //Power supply voltage [V]
4    RTC_DATA_ATTR float U_Moisture[5]; //Moisture sensor
    voltage [V]
5    float Ubat = 0.0; //battery voltage  [V]
6
7    int ADC1_CH0 = 36; //GPIO36, U_Moisture
8    int ADC1_CH3 = 39; //GPIO39, Ubat
9    int ADC1_CH6 = 34; //GPIO34, Vin
10
11  void ADC_TASK(void *pvParameters){
12
13    //Take mutex
14    xSemaphoreTake(measurementMutex, portMAX_DELAY);
15
16    while(1){
17      if(measurementCounter==5){//Cyclic index
18        measurementCounter=0;
19        }
20
21      dacWrite(26, 127);//Set 1.65V as reference
22      digitalWrite(2,LOW); //Enable opamps
23      vTaskDelay(500); //Wait for transiants
24
25      Serial.println();
26      Serial.println("ADC_TASK begins");
27
28      //Measure mmoisture sensor voltage
29      U_Moisture[measurementCounter] = (float)( analogRead(
    ADC1_CH0) / 4095);
30
31      //Measure battery voltage
32      Ubat = (float)( analogRead(ADC1_CH3) / 4095);
33
34      //Measure power supply voltage
35      Vin = (float)( analogRead(ADC1_CH6) / 4095);
36
37      //Disable opamps
38      digitalWrite(2,HIGH);
39
40      /*Serial logging*/
41      Serial.print("Voltage of soil moisture sensor: ");
```

```
42          for(int i=0; i<5;i++){
43            Serial.print(U_Moisture[i]);
44            Serial.print(" ");
45          }
46          Serial.print(" V, ");
47
48          Serial.print("Battery voltage: ");
49          Serial.print(Ubat);
50          Serial.print(" V, ");
51
52          Serial.print("PSU voltage: ");
53          Serial.print(Vin);
54          Serial.println(" V");
55
56          //Increment measureindex
57          measurementCounter++;
58
59          /*When 5 measurements are taken,
60          resume irrigation, MQTT task and give mutex*/
61          if(measurementCounter == 5 ){
62            xSemaphoreGive(measurementMutex);
63            vTaskResume(irrigation_TaskHandle);
64            vTaskResume(MQTT_TaskHandle);
65            };
66
67          Serial.println("ADC_TASK ends");
68          vTaskDelay(delayTime);
69        }
70      }
```

SWAN_CLIENT.ino

```
1  #if CONFIG_FREERTOS_UNICORE
2  #define ARDUINO_RUNNING_CORE 0
3  #else
4  #define ARDUINO_RUNNING_CORE 1
5  #endif
6
7  #define uS_TO_S_FACTOR 1000000 //us to s conversion
8  #define TIME_TO_SLEEP  900 //sleep time in s
9
10
11 /*Includes*/
12 #include <WiFi.h>
13 #include <PubSubClient.h>
14 #include <EEPROM.h>
15 #include "getSerialNumber.h"
16
17 #define EEPROM_SIZE 1
18
```

```
19 RTC_DATA_ATTR int measurementCounter = 0; //index for
20 measurements
21 unsigned long delayTime = 10000;    //10*1000*1ms = 10sec
22 unsigned int serialNumber; //serial number of client
23 bool banned = false;  // if true client won't actuate
24 bool frozen = false;  // if true client won't actuate
25 bool lost = false; // if true client runs buzzer
26 int irrigatonStatus; //
27
28 /*Wifi and MQTT client instances*/
29 WiFiClient espClient;
30 PubSubClient client(espClient);
31
32 /*FreeRTOS task handles*/
33 TaskHandle_t bmp280_TaskHandle = NULL;
34 TaskHandle_t adc_TaskHandle = NULL;
35 TaskHandle_t sleep_TaskHandle = NULL;
36 TaskHandle_t buzzer_TaskHandle = NULL;
37 TaskHandle_t irrigation_TaskHandle = NULL;
38 TaskHandle_t MQTT_TaskHandle = NULL;
39
40 /*Mutex*/
41 SemaphoreHandle_t measurementMutex;
42
43 void setup() { //Run once after every reset
44   setCpuFrequencyMhz(80); //Slows CPU to save power
45
46   Serial.begin(115200); //UART for debugging
47
48   //Reading non-volatile serialnumber from EEPROM
49   EEPROM.begin(EEPROM_SIZE);
50   serialNumber = EEPROM.read(0);
51
52   //Acquire serialnumber if it is not set
53   if(serialNumber == 255){
54      getSerialNumber();
55      serialNumber = EEPROM.read(0);
56      Serial.print(" SerialNumber: ");
57      Serial.println(serialNumber);
58      delay(10);
59      ESP.restart();
60    }

61    //Create mutex to block MQTT task while measurements are
   made
62    measurementMutex = xSemaphoreCreateMutex();
63
64   //Create task
65    xTaskCreatePinnedToCore(
66       BMP280_TASK
67       ,  "BMP280Task"
68       ,  2048
69       ,  NULL
70       ,  3
71       ,  &bmp280_TaskHandle
72       ,  ARDUINO_RUNNING_CORE);
```

```
 73
 74  //Create task
 75   xTaskCreatePinnedToCore(
 76      ADC_TASK
 77      ,  "ADC_TASK"
 78      ,  1024
 79      ,  NULL
 80      ,  3
 81      ,  &adc_TaskHandle
 82      ,  ARDUINO_RUNNING_CORE);
 83
 84  //Create task
 85   xTaskCreatePinnedToCore(
 86      sleep_TASK
 87      ,  "SleepTASK"
 88      ,  1024
 89      ,  NULL
 90      ,  2
 91      ,  &sleep_TaskHandle
 92      ,  ARDUINO_RUNNING_CORE);
 93
 94  //Suspend task immediately
 95   vTaskSuspend(sleep_TaskHandle);
 96
 97  //Create task
 98   xTaskCreatePinnedToCore(
 99      buzzer_TASK
100      ,  "BuzzerTask"
101      ,  1024
102      ,  NULL
103      ,  2
104      ,  &buzzer_TaskHandle
105      ,  ARDUINO_RUNNING_CORE);
106
107  //Suspend task immediately
108   vTaskSuspend(buzzer_TaskHandle);
109
110  //Create task
111   xTaskCreatePinnedToCore(
112      irrigation_TASK
113      ,  "IrrigationTask"
114      ,  1024
115      ,  NULL
116      ,  1
117      ,  &irrigation_TaskHandle
118      ,  ARDUINO_RUNNING_CORE);
119
120  //Suspend task immediately
121   vTaskSuspend(irrigation_TaskHandle);
122
123  //Create task
124   xTaskCreatePinnedToCore(
125      MQTT_TASK
126      ,  "MQTTTask"
127      ,  5120
128      ,  NULL
```

```
129       ,  2
130       ,  &MQTT_TaskHandle
131       ,  ARDUINO_RUNNING_CORE);
132
133   //Suspend task immediately
134    vTaskSuspend(MQTT_TaskHandle);
135 }
136
137 void loop(){//Mapped to idle task
138   client.loop();
139 }
```

getSerialNumber.h

```
 1 #ifndef GETSERIALNUMBER_H
 2 #define GETSERIALNUMBER_H
 3
 4 #include <WiFi.h>
 5 #include <PubSubClient.h>
 6 #include <EEPROM.h>
 7
 8 void getSerialNumbert(void);
 9
10 #endif /*GETSERIALNUMBER*/
```

getSerialNumber.ino

```
 1 #include "getSerialNumber.h"
 2
 3 //Wait for broker sending serialnumber back
 4 bool wait = true;
 5
 6 //Pseudorandom number to identify with broker without serialnumber
 7 int rndNum;
 8
 9 // Callback function called when MQTT message is received
10 void serialcallback(char* topic, byte* message, unsigned int length) {
11     Serial.print("Message arrived on topic: ");
12     Serial.print(topic);
13     Serial.print(". Message: ");
14     String messageTemp;
15
16     //Process  and log message
17     for (int i = 0; i < length; i++) {
18       Serial.print((char)message[i]);
19       messageTemp += (char)message[i];
20     }
21
22     //Process message string into random number and serialnumber
23     int numberTemp = messageTemp.toInt();
24     if(rndNum == numberTemp){//If sent and received random number is
   identical get serialnumber
25       int separatorIndex = messageTemp.indexOf(",");
```

```
      numberTemp = messageTemp.substring(separatorIndex+1).toInt();
      if(numberTemp != 0){
        EEPROM.write(0, numberTemp);
        EEPROM.commit();
        }
      wait = false;
      Serial.println();
    }
    Serial.println();
}

void getSerialNumber(){
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  //Start wifi connection
  WiFi.begin(ssid, password);

  //If not connected wait
  while (WiFi.status() != WL_CONNECTED);

  /*Set up connection with MQTT broker*/
  client.setServer(mqtt_server, 1883);
  client.setCallback(serialcallback);
  client.connect("SWAN");
  client.subscribe("SWAN/recv");
  Serial.println("MQTT connected");

  char payload[512];
  rndNum = esp_random();//generate random number
  sprintf(payload,  "%d", rndNum); //convert int to string

  //Publish random number
  client.publish("SWAN/serial", payload);

  while(wait){client.loop();}
  return;
}
```