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. 11. 23.

...…….
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

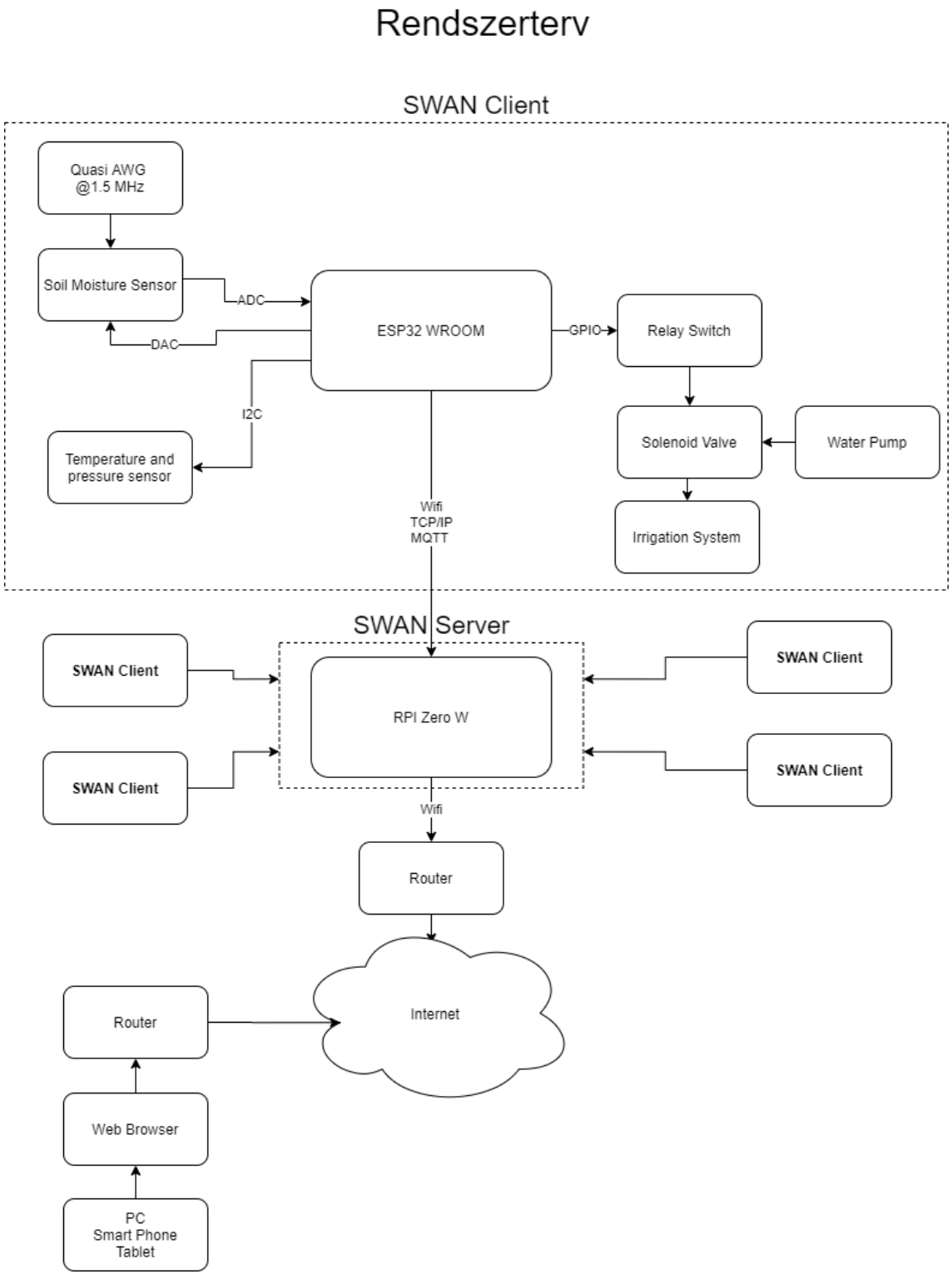
Az öntözőrendszer a Smart Watering Automation System (S.W.A.N.) nevet kapta. A mérő- és beavatkozó egységre SWAN kliensként, a rendszerhez tartozó központi egységre SWAN szerverként hivatkozok a továbbiakban.

Minden kliens a talaj nedvességtartalmát, a levegő hőmérsékletét és páratartalmát méri. A mért adatok, dátum, illetve pontos idő 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 és pontos dátum-idő információt (NTP) szerezhetnek.

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 Mérési elv

2.1 Kapacitív talaj nedvességtartalom mérés

2.1.1 Fizikai jelenség

2.1.2 Kapacitív szenzor mérési elve

2.1.3 Fourier sorfejtés

$$u(t) = \left\{ \begin{array}{l} 1\,, ha\ -\frac{T}{2} \leq t \leq \frac{T}{2} \\ 0\,, \quad egyébként \end{array} \right\}$$

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.

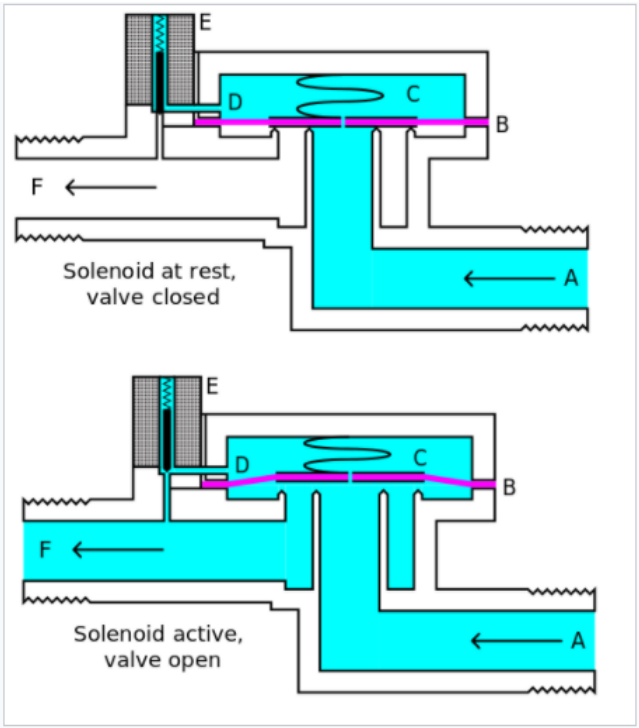
Ez a beavatkozószerv egy mágnesszelep, amely elektromechanikusan működtetett szelep. A mágnesszelepek a folyadékokat szállító rendszerekben leggyakrabban használt vezérlőelemek. Számos alkalmazási területen megtalálhatók. Feladatuk a folyadékok útjának elzárása, felszabadítása, a folyadék adagolása, 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.

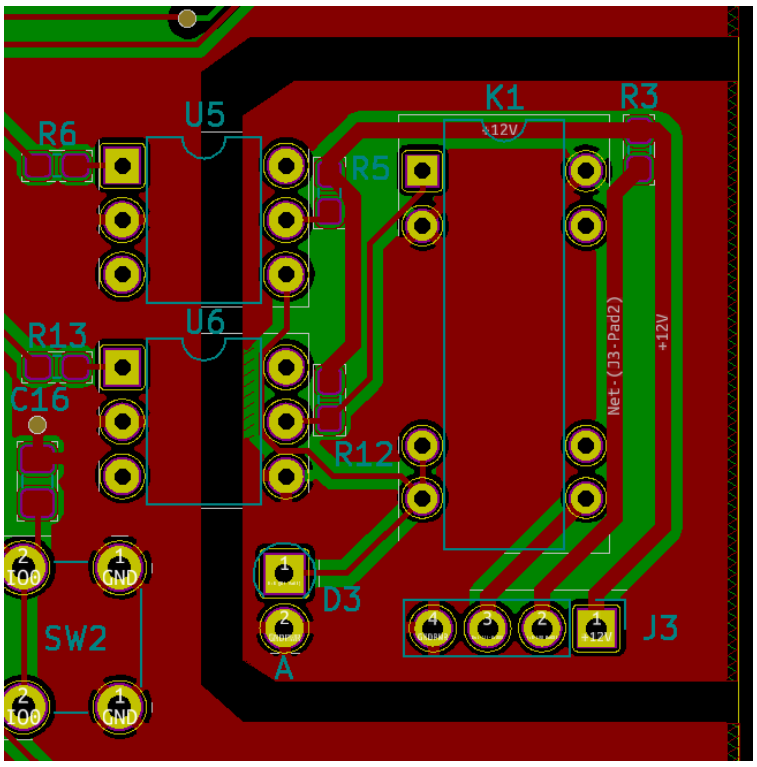
3.1.2 Mágnesszelep vezérlése

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



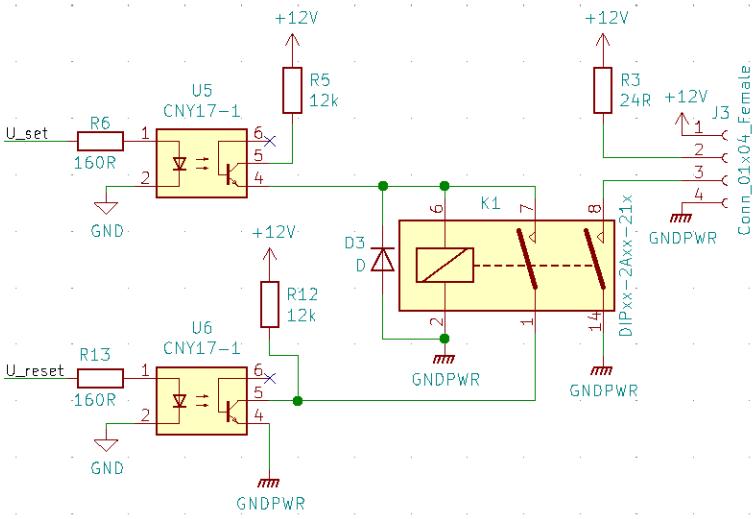
3-2. ábra Galvanikusan leválasztott öntartó relé kapcsolás realizációja

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

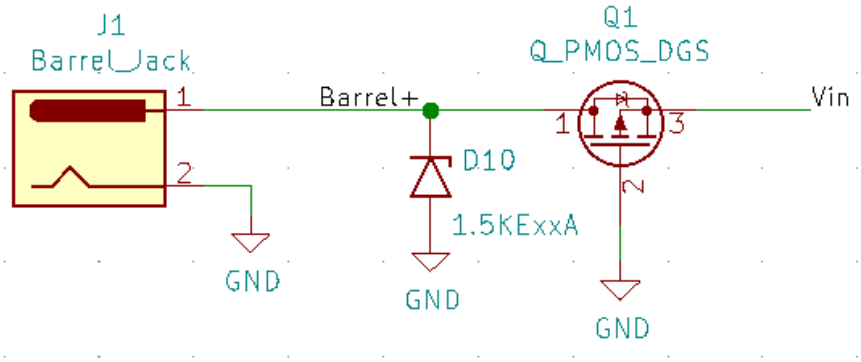
A kliensek mérési helyszíne nem teszi lehetővé a kényelmes csatlakozást a villamos hálózathoz. Mindenképpen telepes vagy akkumulátoros tápellátásra van szükség.

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

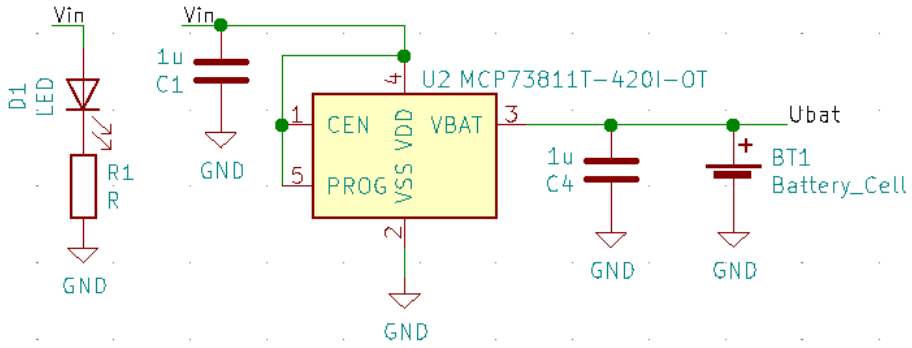
A 3-4. ábra a bemeneti feszültségre alkalmazott ESD és fordított polaritás védelem áramkörének kapcsolási rajzát tartalmazza. Az ESD védelmet TVS dióda látja

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.

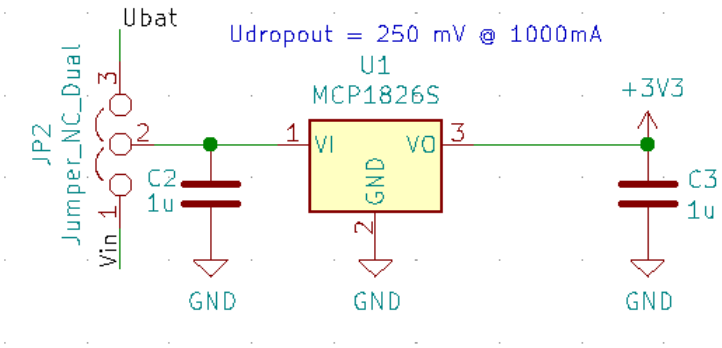
3.1.6 Szabályozott tápfeszültség előállítása

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.

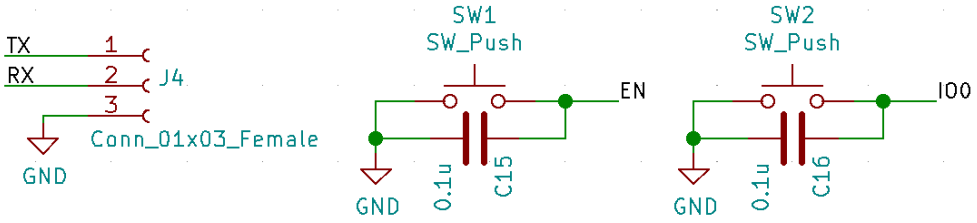
3.1.7 Programozó interface

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.

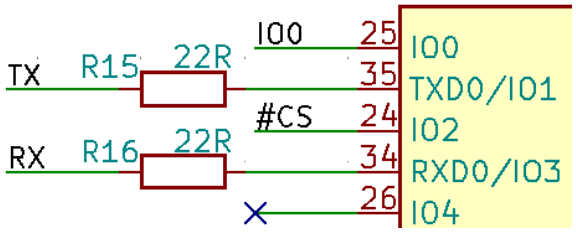
A nyomógombok pergésmentesítésére az ESP32 Dev Kit C adatlapja által javasolt megoldást alkalmaztam. (HIVATKOZÁS)

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



3-7. ábra Programozó interface

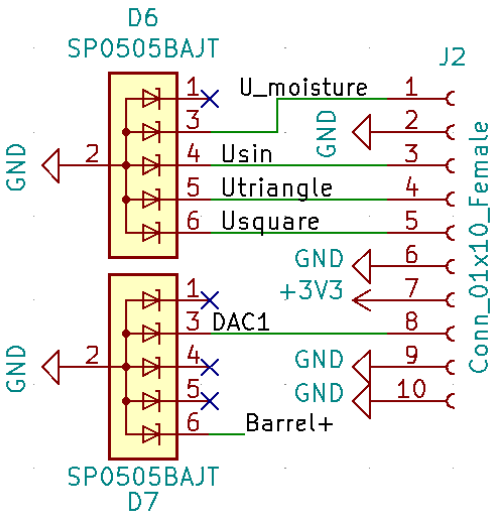
Kihasználva a NYÁK-on megjelenő parazitakapacitásokat, kis értékű 22 Ω értékű ellenállások felhasználásával TX és RX jelek meredek éleit szűröm, ezzel csökkentve az általuk a NYÁK-on keltett zajt.



3-8. ábra Digitális jel szűrőáramköre

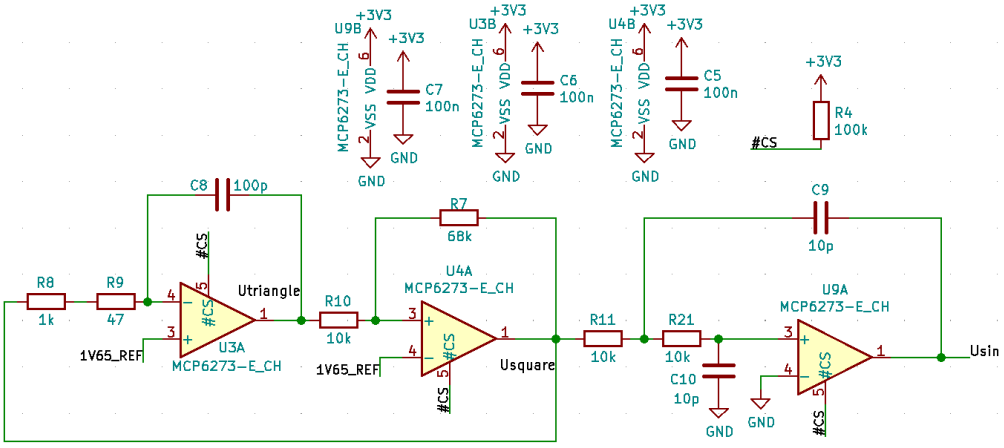
3.1.8 Méréshez szükséges csatlakozó

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:

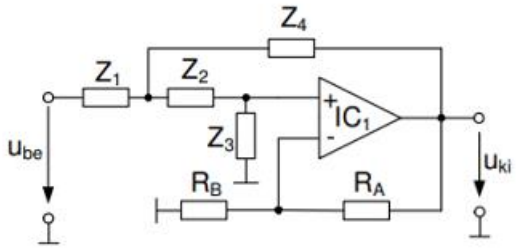
$$U_1 \cong 1.41V$$

$$U_2 \cong 1.89\ V$$

(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)



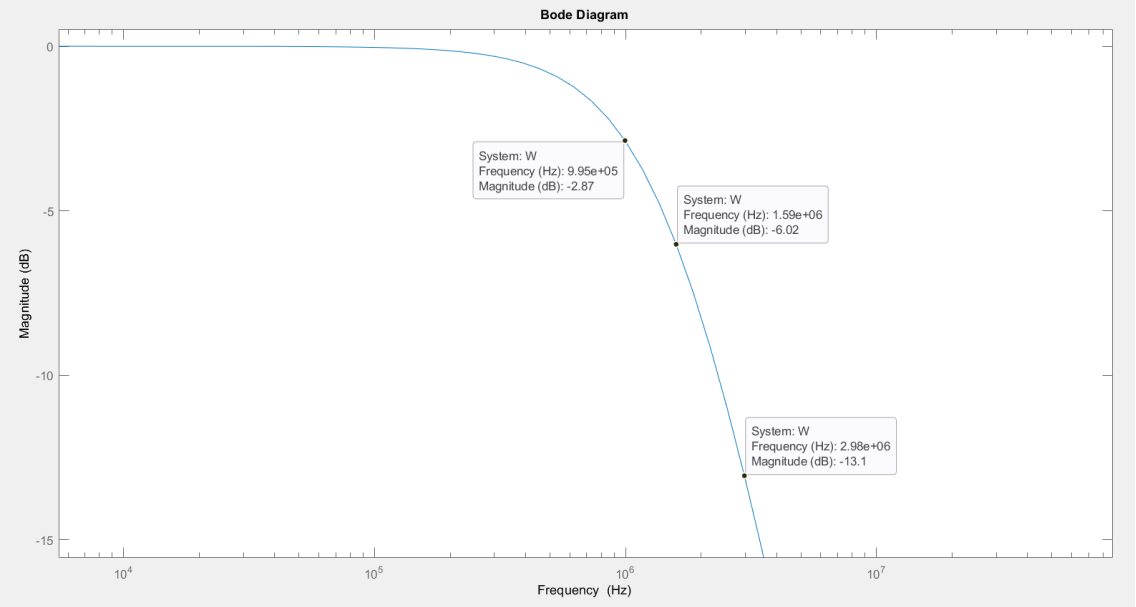
Z1	Z2	Z3	Z4	MEGVALÓSÍTOTT SZŰRŐ
R1	R2	C3	C4	Másodfokú aluláteresztő szűrő

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:

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



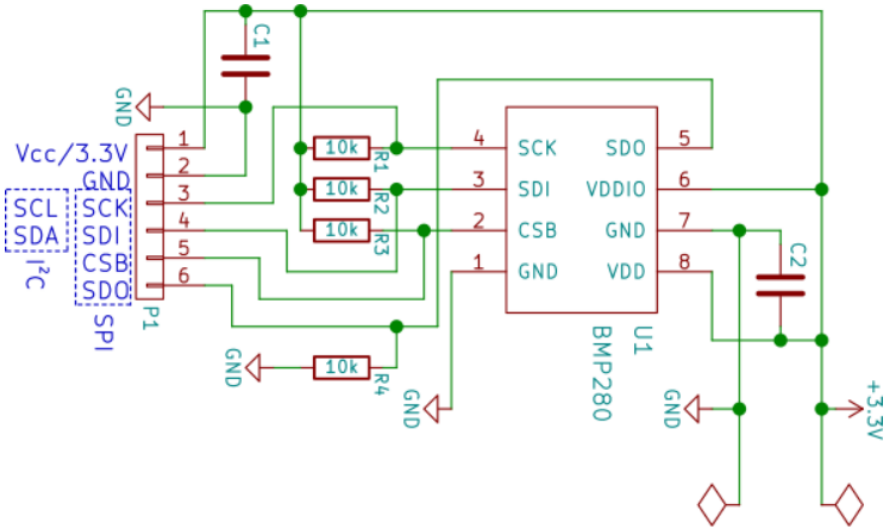
3.1.10 Hőmérséklet- és nyomásmérő áramkör

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)

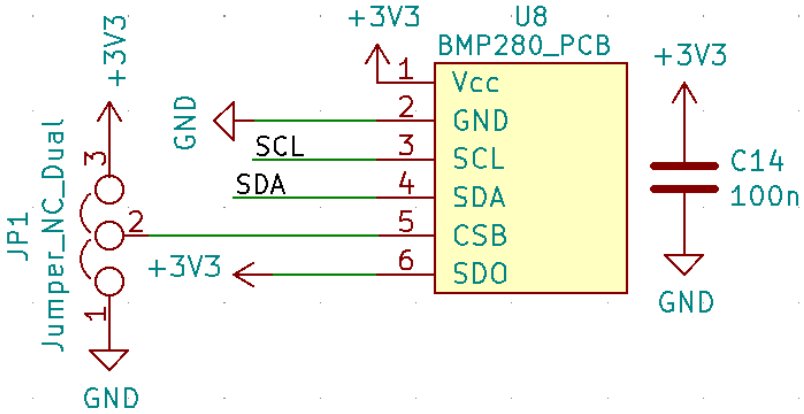
Erre a feladatra a BMP280 elnevezésű szenzort használtam. Amellett, hogy megfelelő pontossággal mér hőmérsékletet és nyomást, kapható ún. breakout boardon is, így külön kis NYÁK-ra került a szenzor és a nagyáramot is szállító NYÁK hőmérséklete kevésbé befolyásolja a hőmérséklet mérést.

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.



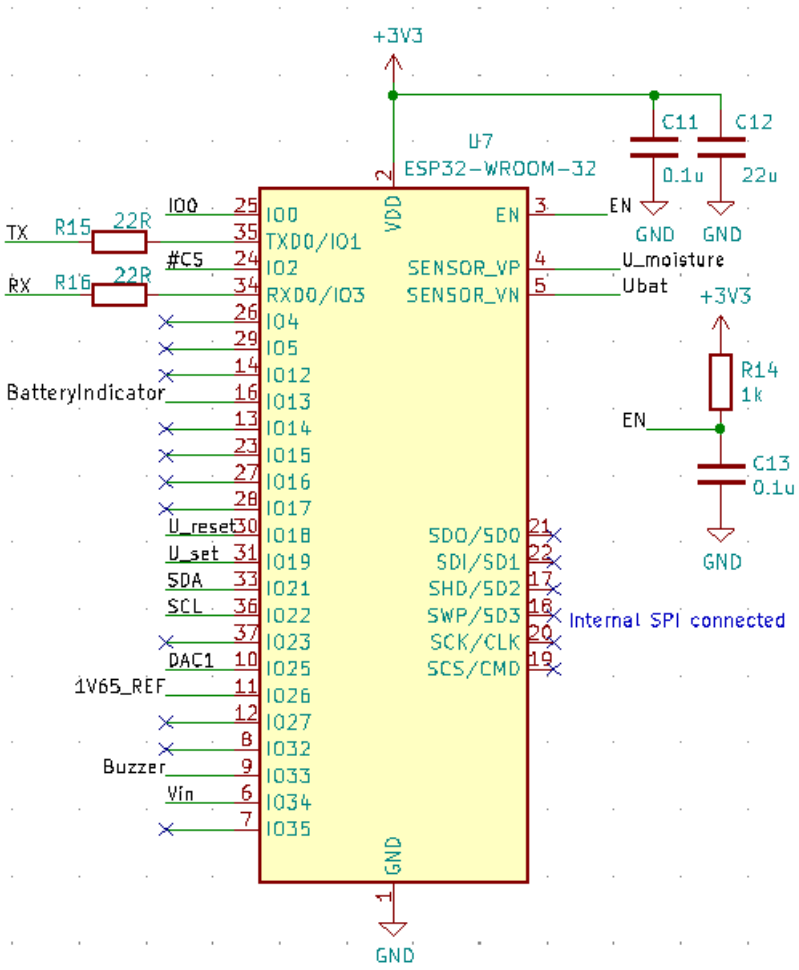
3-12. ábra BMP280 PCB belső felépítése

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

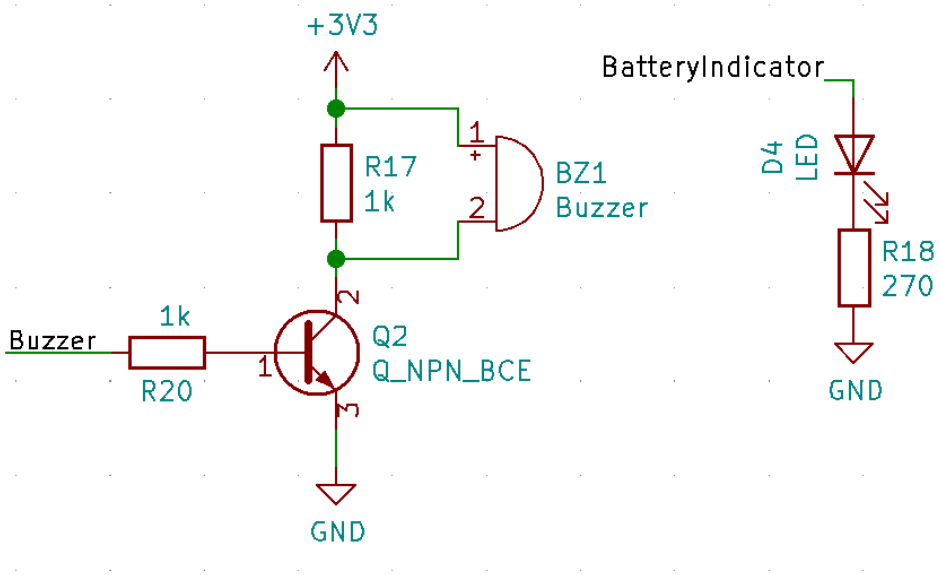
Az ESP32 modul ezen alkalmazáshoz szükséges periféria áramkörök nagy részét tartalmazza, így nem volt szükség sok külső alkatrész csatlakoztatásához.

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

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;
    pinMode(buzzerInputGPIO, OUTPUT);

    while(lost){ //If lost activate buzzer sound (~400 Hz)
         digitalWrite(buzzerInputGPIO, HIGH);
         vTaskDelay(1);
         digitalWrite(buzzerInputGPIO, LOW);
         vTaskDelay(1);
      }
    while(1){
      vTaskResume(sleep_TaskHandle);
      vTaskDelay(delayTime*2);
     }
}
```

Sleep.h

```
#ifndef SLEEP_H
#define SLEEP_H

void sleep_TASK(void *pvParameters);

#endif /*SLEEP_H*/
```

Sleep.ino

```
#include "Sleep.h"

void sleep_TASK(void *pvParameters){
  while(1){
    vTaskDelay(delayTime/10);
    Serial.println();
    Serial.println(F("SWAN Client enters deep sleep ..."));
    esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP/10 * uS_TO_S_FACTOR);
    esp_deep_sleep_start();
  }
}
```

MQTT.h

```
#ifndef MQTT_H
#define MQTT_H

void MQTT_TASK(void *pvParameters);

#endif /*MQTT_H*/
```

MQTT.ino

```
 #include "MQTT.h"
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>


const char* ssid = "****";
const char* password = "****";
const char* mqtt_server = "192.168.0.202";

void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
      Serial.print((char)message[i]);
      messageTemp += (char)message[i];
```

```
20      }
21      Serial.println();
22      if( messageTemp == "banned") {
23        Serial.print("I'm banned");
24        Serial.println();
25        banned = true;
26        }
27      if( messageTemp == "buzzer") {
28        Serial.print("I'm lost");
29        Serial.println();
30        lost = true;
31        }
32    }
33
34
35
36
37  void MQTT_TASK(void *pvParameters){
38    Serial.println();
39    Serial.print("Connecting to ");
40    Serial.println(ssid);
41
42    WiFi.begin(ssid, password);
43
44    while (WiFi.status() != WL_CONNECTED) {
45        vTaskDelay(500);
46        Serial.print(".");
47     }
48
49    Serial.println("");
50    Serial.println("WiFi connected");
51    Serial.println("IP address: ");
52    Serial.println(WiFi.localIP());
53
54    client.setServer(mqtt_server, 1883);
55    client.setCallback(callback);
56    client.connect("SWAN1");
57    client.subscribe("SWAN/recv");
58    Serial.println("MQTT connected");
59
60    char payload[256];
61
62    while(1){
63      Serial.println("Publishing...");
64
65      StaticJsonDocument<256> doc;
66
67      doc["Client"] = "01";
68      doc["Battery_Voltage"] = Ubat;
69
70      JsonArray temperature = doc.createNestedArray("temp");
71      for(int i=0;i<5;i++){
72        temperature.add(temp[i]);
73      }
74
75      JsonArray pres = doc.createNestedArray("pressure");
```

```
76      for(int i=0;i<5;i++){
77        pres.add(pressure[i]);
78      }
79      JsonArray U_mois = doc.createNestedArray("U_moisture");
80      for(int i=0;i<5;i++){
81        U_mois.add(U_Moisture[i]);
82      }
83
84      client.publish("SWAN/pub", payload);
85
86      serializeJson(doc, payload);
87      Serial.print(payload);
88      Serial.println();
89
90      vTaskDelay(delayTime/2);
91
92      vTaskResume(buzzer_TaskHandle);
93      vTaskResume(irrigation_TaskHandle);
94    }
95 }
```

Irrigation.h

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

Irrigation.ino

```
1  #include "Irrigation.h"
 2
 3 void irrigation_TASK(void *pvParameters){
 4     const byte resetPulse = 18;
 5     const byte setPulse = 19;
 6     pinMode(resetPulse, OUTPUT);
 7     pinMode(setPulse, OUTPUT);
 8
 9     while(1){
10         if(frozen || banned) {
11           digitalWrite(setPulse, LOW);
12           digitalWrite(resetPulse, HIGH);
13           vTaskDelay(delayTime);
14          }
15         else{
16           digitalWrite(resetPulse, LOW);
17           digitalWrite(setPulse, HIGH);
18           vTaskDelay(delayTime);
19           }
20       }
21   }
```

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;
int device_address = 0x76;

RTC_DATA_ATTR float temp[5];
RTC_DATA_ATTR float pressure[5];


void BMP280_TASK(void *pvParameters){
    unsigned long retryTime = 300000;   //5*6*10*1000*1ms = 5min
    bool status;
    float avgTemp = 0.0;

    status = bmp.begin(device_address);  //I2C address can be
  0x77 or 0x76

    if (!status) {
        Serial.println("Could not find sensor, check PCB
connection!");
        while (1){
          vTaskDelay(retryTime);
          };
    }

    BMP280_Sleep(0x76);

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

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

    temp[measurementCounter] = bmp.readTemperature();
    pressure[measurementCounter] = bmp.readPressure() / 100.0F;

    for(int i=0;i<5;i++){
      avgTemp += temp[i];
```

```
42      }
43     avgTemp = avgTemp / 5;
44     if( (avgTemp - 5.0)< 0.01 ){frozen = true;}
45
46     printValues();
47
48     BMP280_Sleep(device_address);
49     Serial.println("BME280 to Sleep mode...");
50
51     Serial.println("BMP280_TASK ends");
52     vTaskDelay(delayTime);
53   }
54 }
55
56 void printValues() {
57     Serial.print("Temperature:  ");
58     for(int i=0; i<5;i++){
59       Serial.print(temp[i]);
60       Serial.print(" *C ");
61     }
62     Serial.print(", ");
63
64     Serial.print("Pressure:  ");
65     for(int i=0; i<5;i++){
66       Serial.print(pressure[i]);
67       Serial.print(" hPa ");
68     }
69     Serial.println();
70 }
71
72 void BMP280_Sleep(int device_address) {
73   // Mode[1:0]  Mode
74   //    00      'Sleep'  mode
75   //  01 / 10   'Forced' mode, use either '01' or '10'
76   //    11      'Normal' mode
77
78   Wire.beginTransmission(device_address);
79   Wire.write((uint8_t)0xF4);
80   Wire.write((uint8_t)0b00000000);
81   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

```
#include "ADC.h"

 float Vin= 0.0;
 RTC_DATA_ATTR float U_Moisture[5];
 float Ubat = 0.0;

 int ADC1_CH0 = 36; //GPIO36, U_Moisture
 int ADC1_CH3 = 39; //GPIO39, Ubat
 int ADC1_CH6 = 34; //GPIO34, Vin

void ADC_TASK(void *pvParameters){

  while(1){
    if(measurementCounter==5){measurementCounter=0;}

    dacWrite(26, 127);
    digitalWrite(2,LOW);
    vTaskDelay(500);

    Serial.println();
    Serial.println("ADC_TASK begins");

    U_Moisture[measurementCounter] = (float)(
analogRead(ADC1_CH0) / 4095);

    Ubat = (float)( analogRead(ADC1_CH3) / 4095);
    Vin = (float)( analogRead(ADC1_CH6) / 4095);

    if(Vin>0.0 && Ubat > 3.299){digitalWrite(13,HIGH);}

    digitalWrite(2,HIGH);

    Serial.print("Voltage of soil moisture sensor: ");
    for(int i=0; i<5;i++){
      Serial.print(U_Moisture[i]);
      Serial.print(" ");
    }
    Serial.print(" V, ");

    Serial.print("Battery voltage: ");
    Serial.print(Ubat);
    Serial.print(" V, ");

    Serial.print("PSU voltage: ");
    Serial.print(Vin);
    Serial.println(" V");

    measurementCounter++;
    if(measurementCounter == 5 ){vTaskResume(MQTT_TaskHandle);};

    Serial.println("ADC_TASK ends");
    vTaskDelay(delayTime);
  }
}
```

SWAN_CLIENT.ino

```
#if CONFIG_FREERTOS_UNICORE
#define ARDUINO_RUNNING_CORE 0
#else
#define ARDUINO_RUNNING_CORE 1
#endif

#define uS_TO_S_FACTOR 1000000
#define TIME_TO_SLEEP  30

#include <WiFi.h>
#include <PubSubClient.h>

RTC_DATA_ATTR int bootCount = 0;
RTC_DATA_ATTR int measurementCounter = 0;
unsigned long delayTime = 10000;    //6*10*1000*1ms = 1min
unsigned int serialNumber;
bool banned = false;
bool frozen = false;
bool lost = false;

WiFiClient espClient;
PubSubClient client(espClient);

TaskHandle_t bmp280_TaskHandle = NULL;
TaskHandle_t adc_TaskHandle = NULL;
TaskHandle_t sleep_TaskHandle = NULL;
TaskHandle_t buzzer_TaskHandle = NULL;
TaskHandle_t irrigation_TaskHandle = NULL;
TaskHandle_t MQTT_TaskHandle = NULL;

void setup() {
 setCpuFrequencyMhz(80);

 if(bootCount == 0 || bootCount == 10){
    bootCount = 1;

    Serial.begin(115200);

    xTaskCreatePinnedToCore(
      BMP280_TASK
      ,  "BMP280Task"
      ,  2048
      ,  NULL
      ,  3
      ,  &bmp280_TaskHandle
      ,  ARDUINO_RUNNING_CORE);

    xTaskCreatePinnedToCore(
      ADC_TASK
      ,  "ADC_TASK"
      ,  1024
      ,  NULL
      ,  3
```

```
54        ,  &adc_TaskHandle
55        ,  ARDUINO_RUNNING_CORE);
56
57      xTaskCreatePinnedToCore(
58        sleep_TASK
59        ,  "SleepTASK"
60        ,  1024
61        ,  NULL
62        ,  2
63        ,  &sleep_TaskHandle
64        ,  ARDUINO_RUNNING_CORE);
65      vTaskSuspend(sleep_TaskHandle);
66
67      xTaskCreatePinnedToCore(
68        buzzer_TASK
69        ,  "BuzzerTask"
70        ,  1024
71        ,  NULL
72        ,  1
73        ,  &buzzer_TaskHandle
74        ,  ARDUINO_RUNNING_CORE);
75      vTaskSuspend(buzzer_TaskHandle);
76
77      xTaskCreatePinnedToCore(
78        irrigation_TASK
79        ,  "IrrigationTask"
80        ,  1024
81        ,  NULL
82        ,  1
83        ,  &irrigation_TaskHandle
84        ,  ARDUINO_RUNNING_CORE);
85      vTaskSuspend(irrigation_TaskHandle);
86
87      xTaskCreatePinnedToCore(
88        MQTT_TASK
89        ,  "MQTTTask"
90        ,  5120
91        ,  NULL
92        ,  2
93        ,  &MQTT_TaskHandle
94        ,  ARDUINO_RUNNING_CORE);
95      vTaskSuspend(MQTT_TaskHandle);
96    }
97
98    else{
99      esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP/10 *
    uS_TO_S_FACTOR);
100     bootCount++;
101     esp_deep_sleep_start();
102     }
103 }
104
105 void loop(){
106   client.loop();
107 }
```