Smart Watering Automation Network

Berta Máté

HPD5LB

Konzulens:

Naszály Gábor

Tartalomjegyzék

Feladatkiírás

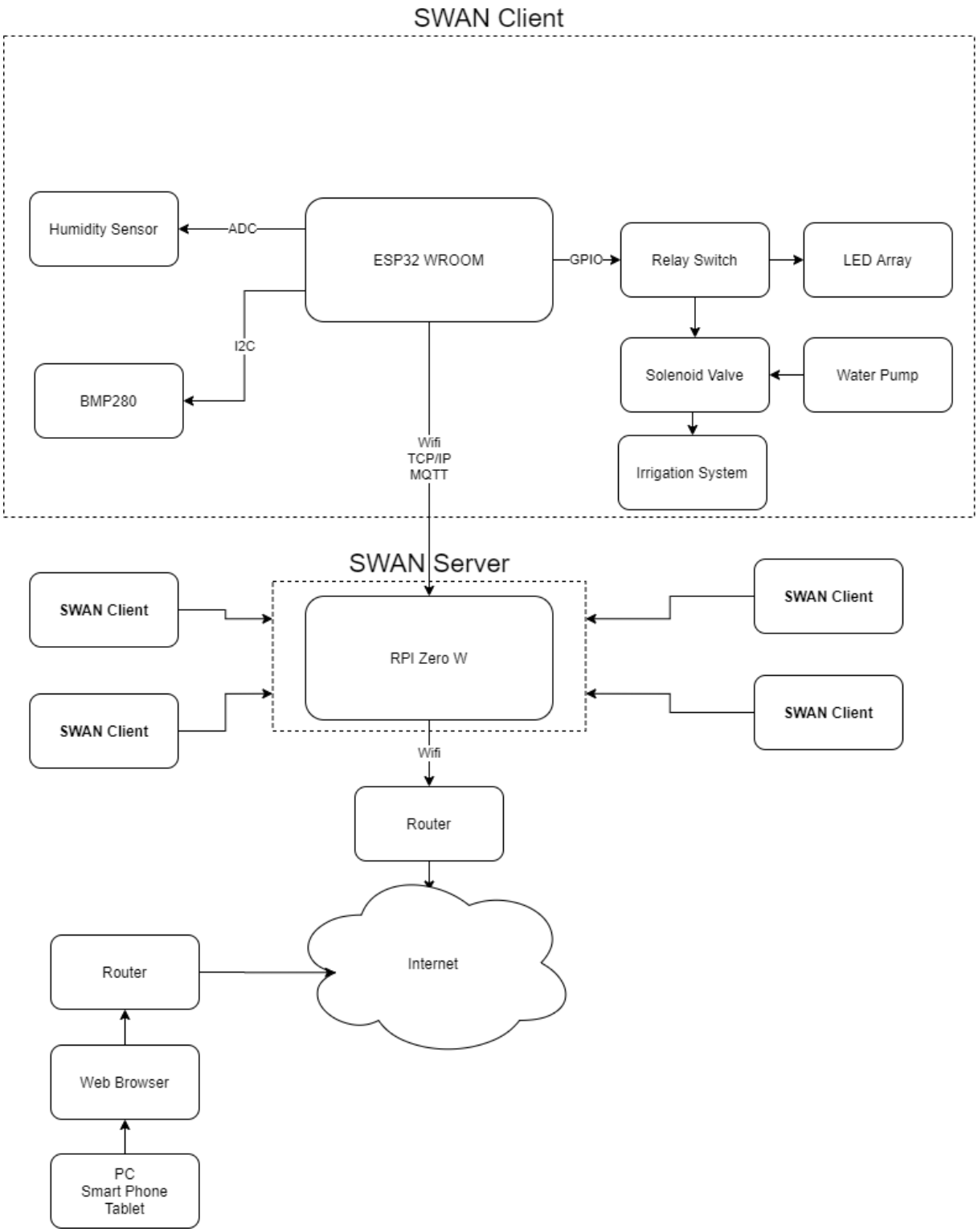
Manapság népszerű kutatási, fejlesztési terület az IoT. Ezen belül is nagy népszerűségnek örvendenek a különféle otthonautomatizálási megoldások. A hallgató feladata is ebbe a témakörbe illeszkedik egy automatizált kerti öntözőrendszer megtervezésével. A rendszer egy szerver egységből és minimum egy kliens egységből áll. További követelmény a rendszerrel szemben, hogy el lehessen érni az Internet felől.

Az első feladat a rendszerterv elkészítése, majd az egyes funkciókat megvalósító egységek számára az alkatrész választás.

A következő feladat a teljes rendszer egy akkora részének megvalósítása, amivel demonstrálható az ötlet működőképessége. A megvalósításnak ki kell terjednie hardveres illesztési feladatokra próbapanelen, valamint mind a szerver, mind a kliens csomópontok beágyazott szoftvereinek kezdeti változatának elkészítésére.

Berta Máté
HPD5LB

Rendszerterv

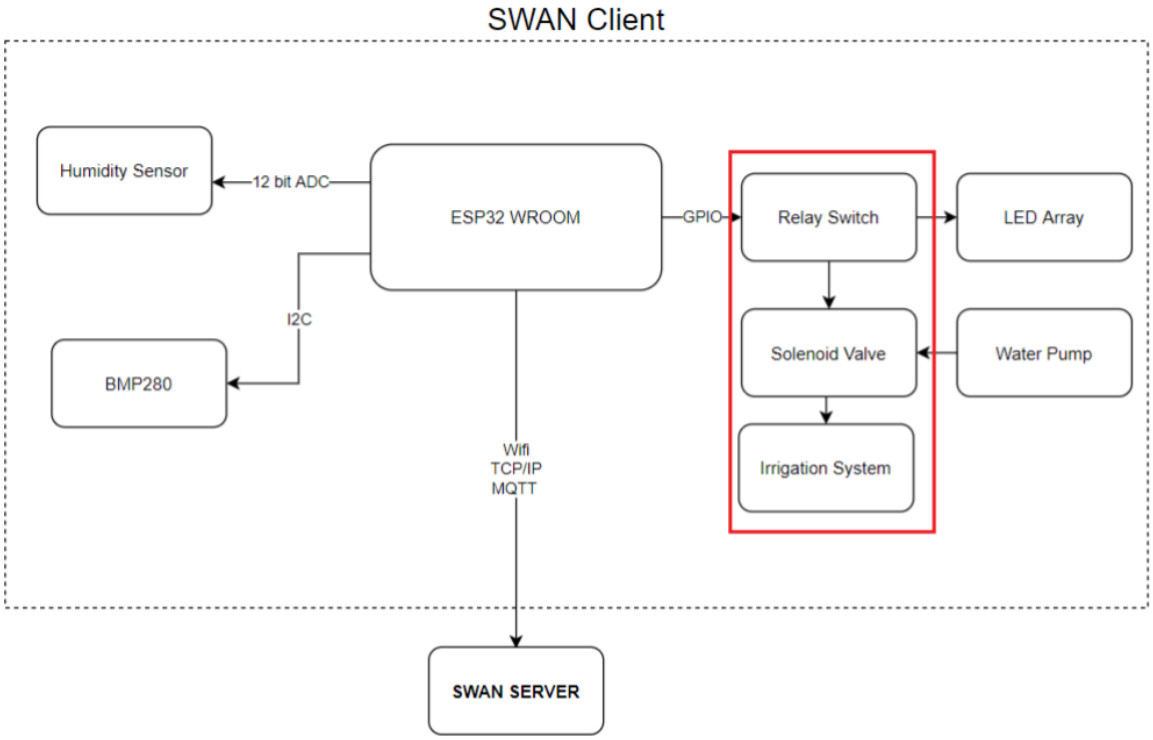


1.ábra

Kliens megvalósítása

A kiválasztott komponens megvalósítása során költségcsökkentésképp a szolenoid szelep és a hozzátartozó elektronika kihagyásra került. A funkciót egy LED jelképezi, amikor a rendszer „öntöz” a LED világít, az öntözést pedig kézileg végzem. A méréseket így nem szükséges „terepen” végezni, a föld, amelyen a viselkedés tanulmányozásra került egy műanyag tárolóban, a szobában (akár laborban) is tartható.

A pirossal jelölt részletek a rendszerterven nem kerülnek megvalósításra:



2.ábra

Platformválasztás

Telekommunikációs megoldás

Szükséges a távoli információszerzés és beavatkozás megvalósításához valamilyen távkommunikációs megoldás beépítése a rendszerbe. A megoldás kiválasztása során figyelembe kell venni a felhasználandó alkatrészek bonyolultságát, villamos energia fogyasztását, költségeit és hatótávolságát.

A kiválasztás során három technológia alkalmazása került szóba: a Wifi, a Bluetooth Low Energy és a LoRaWAN.

Alacsony fogyasztás és nagy hatótávolság miatt a LoRa technológia egyértelmű választás lenne, viszont magas prototipizálási költséggel rendelkezik (1:10-hez aránylik a LoRa:Wifi/BLE development boardok árához). Emellett a távoli beavatkozás megvalósítása nagyban növelte volna a villamos energia fogyasztást (Class A működés esetén túl sokszor kellene felébreszteni az eszközt, Class C működés esetén pedig nem tekinthető alacsony fogyasztású eszköznek), amely az eszköz legnagyobb előnyét mérsékelte volna. [1]

A Bluetooth Low Energy alacsony belépési költséggel rendelkezik, viszont a kb. 10 méteres hatótávolsága (ESP32 SoC-vel) miatt nem megfelelő egy átlagos kert - ház távolságának áthidalására, mesh hálózat kiépítése pedig feleslegesen növelné az alkatrészek számát és költségét. [2]

Végül a Wifi technológiára esett a választás, az alacsony ára miatt, emellett mérsékelhető fogyasztással és megfelelő hatótávval rendelkezik egy kerti öntöző megvalósításához (extrém esetben eltúlzott nagyságú vevőantennával akár 10 km). Ráadásul az alkalmazandó szolenoid tekercses szelepek miatt, a modul hálózati táplálással kell, hogy rendelkezzen, emiatt a fogyasztás csökkentése csak a fenntartási költség mérséklése miatt fontos. Tehát nem szükségesek az ultra low power megoldások, mint ahogy például egy telepről működő IoT adatgyűjtő esetén lenne. [3]

Wifi Chipset és mikrokontroller választás

Manapság könnyen beszerezhetők olyan Wifi kommunikációra alkalmas SoC-k, melyek sok általános mikrokontrolleres feladatot ellátnak, tartalmazzák a megfelelő hardverelemeket (nem vezeték nélküli kommunikációs perifériák, AD/DA átalakítók, PWM generátorok stb.). Célszerű egy ilyen kontrollert választani, így csökkentve az alkatrészek számát.

Két könnyen hozzáférhető mikrokontroller jött szóba a projekttel kapcsolatban: ESP8266 és ESP32. Előbbi kisebb teljesítményű, viszont kisebb beszerzési költségekkel jár és a feladat elvégzésére is alkalmasnak tűnik.

Mindenképpen mérlegelni kellett a skálázhatóságot, hiszen egy I2C szenzor kiolvasása és egy ADC csatornán való feszültségmérése még bőven megoldható ESP8266 segítségével (bitbanged I2C-vel és egy ADC-vel rendelkezik) azonban, ha több mérési pontot szeretnénk a rendszerbe vinni, akkor problémákat okozhat a hardveres korlátoltság.

Emellett a számítási kapacitásra is gondolni kell, hiszen a hálózati kommunikáció gyakran sokáig foglalhatja az erőforrásokat. Azáltal pedig a szabályozó ritkább mintavételekkel dolgozhat, amely a rendszer időállandóját növeli, ami akár hibás szabályozást is eredményezhet. [4]

A választás az ESP32 SoC-ra esett, mivel hardveres I2C perifériával rendelkezik 18 csatornás 12 bites SAR ADC-vel (ADC1 – 8 csatorna ADC2 – 10 csatorna), illetve két processzor maggal, amelyek segítségével teljesen elkerülhető a szabályozást végző task kiéheztetése. [5]

Másik fontos érv az ESP32 mellett, az volt, hogy a gyártó támogatja a FreeRTOS operációs rendszert a chipen, némi kiegészítéssel a két magos processzor miatt. [6]

Fejlesztői környezet

ESP32 platformra rengeteg programozási segédlet, fejlesztő eszköz és szoftver könyvtár létezik, mivel kedvelt a hobbi IoT „kütyük” készítői között. Ezen eszközök közül legismertebb az Arduino Open Hardware/Software platform és a hozzá tartozó Arduino IDE. Gyors és könnyű fejlesztést tesz lehetővé, viszont a hardver egyes elemeihez nehezíti vagy ellehetetleníti a hozzáférést (pl.: energia csökkentő módok ki/be kapcsolása). [7]

A választott megoldás a gyártó által kiadott szoftver könyvtár, az Espressif IoT Development Framework (ESP-IDF) használata lett. A fejlesztés könnyítésére nem csak egyszerű text editor és a gyártó által kiadott programozói szoftvert (esptool) használtam fel, hanem az Eclipse IDE és ESP-IDF plugin felhasználásával egy felületen írhattam a kódot, programozhattam az eszközt és monitorozhattam a debug portot (UART). [8]

A fejlesztői környezet telepítését egy virtuális gépen végeztem el, így könnyen „mozgatható” munkaállomást kaptam végeredményül. A virtuális gép Ubuntu 19.10 operációs rendszerrel rendelkezik, a fejlesztőkörnyezet működtetéséhez szükséges szoftver komponensek a következők: [9]

- Eclipse CDT (C/C++ plugin, GCC compiler)
- Python 3.7
- Java 8
- Git
- ESP-IDF v4.0

A gyártó által kiadott telepítő állomány folyamatosan változik, az újabb és újabb verziók telepítése során gyakori, hogy hibaüzenetekkel találkozik a felhasználó. A telepítés idejében a legfrissebb a virtualenv python package legújabb verziójával nem tudott települni, ezért vissza kellett azt állítani egy régebbi verzióra.

Hőmérséklet és nyomás mérés

I2C

Gyakori problémát jelent több szenzor alkalmazása során az AD átalakítók korlátozott száma. Annak érdekében, hogy több azonos vagy más fizikai mennyiséget mérő szenzorral tudjunk mikrokontrolleres környezetben mérni, valamilyen integrált áramkörök közötti busz kommunikációt szükséges használni.

A projektben ritka (pl. percenkénti) adatgyűjtésre van szükség, nem időkritikus egy mért adat kiolvasása, így az IoT alkalmazásokban egyik leggyakoribban alkalmazott busz rendszert az I2C-t (Inter-Integrated Circuit) lehetett használni.

A busz két vezetékes (SDA, SCL), 100 kHz és 5 MHz közötti sebességű kommunikációra képes. Szükség esetén lassú slave eszközök lassíthatnak a kommunikáción, az SCL vonal alacsony szinten tartásával.

Minden eszköz egyedi, 7 bites címmel rendelkezik, melyeket részben a gyártók adnak meg. Annak érdekében, hogy azonos mérőeszközök is a buszon lehessenek, a gyártók a címek csak egy részét specifikálják, így a szabadon hagyott bitek hardveres beállításával külön cím adható kettő vagy több azonos típusú chipnek.

Előnye ennek a protokollnak, hogy kis erőforrásokat igényel, illetve akár hardveres támogatás nélkül is használható a mikrokontroller oldaláról. [11]

BMP280

A Bosch által gyártott BMP280 chip hőmérséklet és nyomás mérésére alkalmas, I2C és SPI buszon keresztül vezérelhető szenzor, melynek alacsony a fogyasztása és ezen felül még fogyasztáscsökkentő alvó üzemmódja is van. [12]

A kerti öntöző számára értékes információt jelent a levegő hőmérséklete, melyből egyaránt közvetlen és közvetett következtetéseket vonhatunk le és dönthetünk a beavatkozás szükségességéről.

Közvetlenül tudjuk, ha fagy miatt nem lehet öntözni, illetve túl nagy hőség esetén dönthetünk úgy, hogy hűtés céljából öntözünk.

Közvetett információt nyerhetünk a napszakról, évszakról, több mérési pont alkalmazásával akár a kert felületét érintő árnyékról is.

Nyomás mérésével közvetetten információt gyűjthetünk a páratartalomról - adott hőmérséklet mellett - ezzel akár különböző növényfajták ideális környezetére is szabályozhatunk.

A szenzor használatához a gyártó által biztosított C nyelvű drivereket használtam fel [13], melyeket csak a konkrét mikrokontrollerhez tartozó I2C/SPI kezelő kóddal kellett kiegészíteni.

Berta Máté
HPD5LB

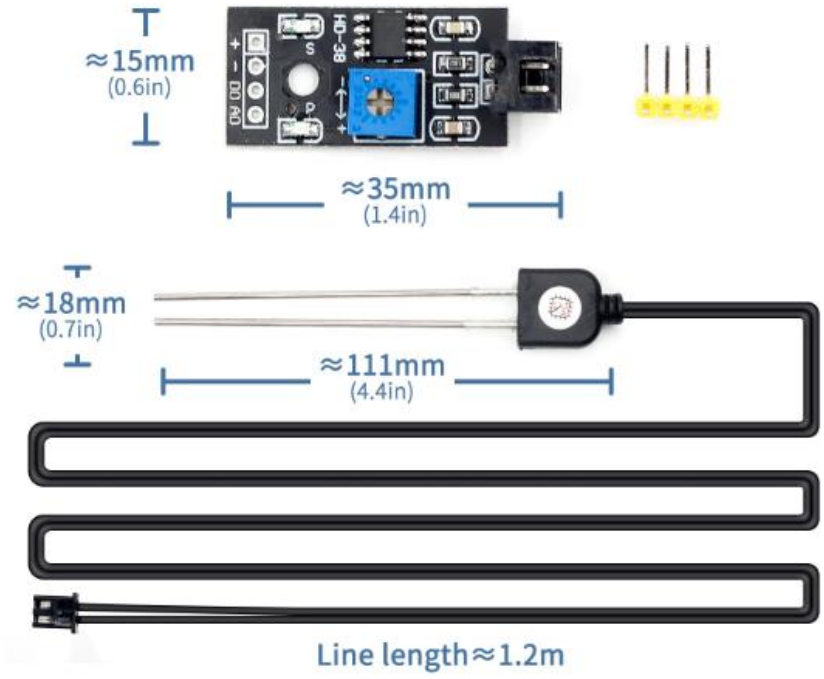
Talaj nedvességtartalom mérés

Kétféle mérési elv összehasonlítása céljából két a piacon kapható szenzor került megrendelésre, viszont a 2020-as koronavírus helyzet miatt a kiszállítások bizonytalanná váltak. A két megrendelt szenzor közül egy érkezett meg a félév során, így csak ennek az egy szenzornak a kipróbálására került sor a mérések során.

Rezisztív alapon működő mérési elv

Egyszerű és kézenfekvő ötlet, hogy a talaj nedvességtartalmának változása befolyásolja a vezetőképességét, ebből kiindulva közvetetten mérhetjük a nedvességtartalom változását. Magas nedvességtartalom mellett, a vízben oldódik a talaj ásványianyag-tartalma, amely jó vezetőképességű közeget alkot. Alacsony nedvességtartalom mellett pedig a kiszáradt talaj nem vezet jól, így könnyű megkülönböztetni a két állapotot. A kérdés, hogy a két állapot közötti szakaszon, milyen karakterisztikát kapunk, annak alakját és paramétereit mennyire befolyásolja a talaj minősége vagy a mérőszonda állapota (öregedés, korrózió).

A felhasznált szenzor márkajelzés nélküli, interneten vásárolható eszköz, amelyről nem sok információt lehet szerezni. Működése könnyen visszakövethető a PCB vizuális vizsgálata alapján. A mérőszonda egy ellenállásosztó egyik tagjaként modellezhető, amely a PCB-ről kap tápellátást. Ennek az osztónak a kimeneti feszültségét (A_0) közvetlenül mérhetjük mikrokontrollerrel. Emellett egy potenciométer segítségével állíthatunk be egy olyan feszültségértéket, amellyel komparáljuk a szondán eső feszültséget és a digitális kimeneten (D_0) jelez az eszköz.



3.ábra

Az implementációban az analóg kimenetet használtam fel, a digitális kimenet funkcióját (vészjelzés) szoftveresen oldottam meg.

Tapasztalható volt a szenzor használata során a mérést „meghamisító” önmelegedés, illetve általánosan a rossz érzékenység. Felmerül a hosszú 1.2 méteres mérőkábel okozta rendszeres hiba, hiszen kétvezetéses mérés történik. Ennek ellenére felhasználható ezen a területen, hiszen a kerti növényeknek szükséges talaj nedvességtartalom nem definiálható pontosan, szubjektív megítélés és tapasztalat alapján kell dönteni, hogy milyen értéken akarjuk azt tartani.

Amennyiben lehetséges, érdemes a kapacitív elven működő szenzort használni, a rezisztív módszerrel nem tapasztaltam megfelelő műszakilag megbízható működést. [14]

Kapacitív alapon működő mérési elv

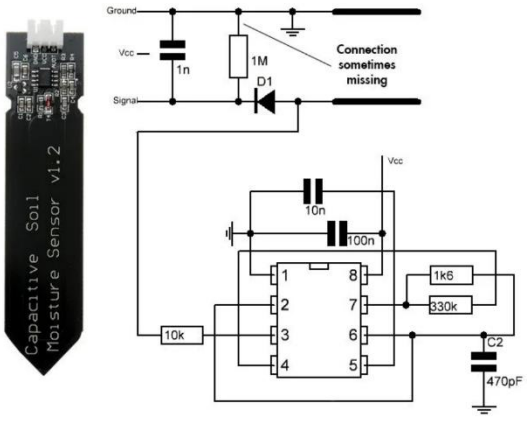
A kapacitív elven működő szenzorok impedanciamérésre vezetik vissza a talaj komplex permittivitás változását, ennek segítségével következtethetünk a talaj nedvességtartalmára.

Ez az elv a talajjal való érintkezés miatti korróziót kizárja, mivel nincs szükség fémes kapcsolatra a szonda és a talaj között. A mérőjel változtatásával (frekvencia változtatás, mérőspektrum változtatása) lehetőség adódna a mérési módszer talajtípushoz való hangolására is.

A kimérhető karakterisztika alakja továbbra is erősen függ a talaj típusától, lehet nemlineáris is. Emellett a gazdaságosan beszerezhető kész szenzor megoldások a mérőjel előállítására nem használnak pontos megoldásokat, ráadásul ~10 kHz-es tartományban mérnek, amelynél nagyobb frekvenciát ajánlanak az ide vonatkozó szakirodalmak. [15]

Internetről könnyen beszerezhető szenzor, a mérőjel előállításához klasszikus 555 timer IC-t használ a stabil multivibrátor kapcsolásban, amely nem képes nagyfrekvenciás jelet előállítására és spektruma sem egy frekvencián emel ki, hiszen négyszögjelet állít elő. A frekvencia bizonytalanságát nem csak az IC, hanem a felhasznált diszkrét elemek is meghatározzák, ezek viszont gyakran nagy szórású alkatrészek.

Ezt a fajta mérési elvet érdemes még tanulmányozni és gyakorlatban kipróbálni. Jövőbeli terveim között szerepel a különböző mérőjelek használatának összehasonlítása is.



4.ábra

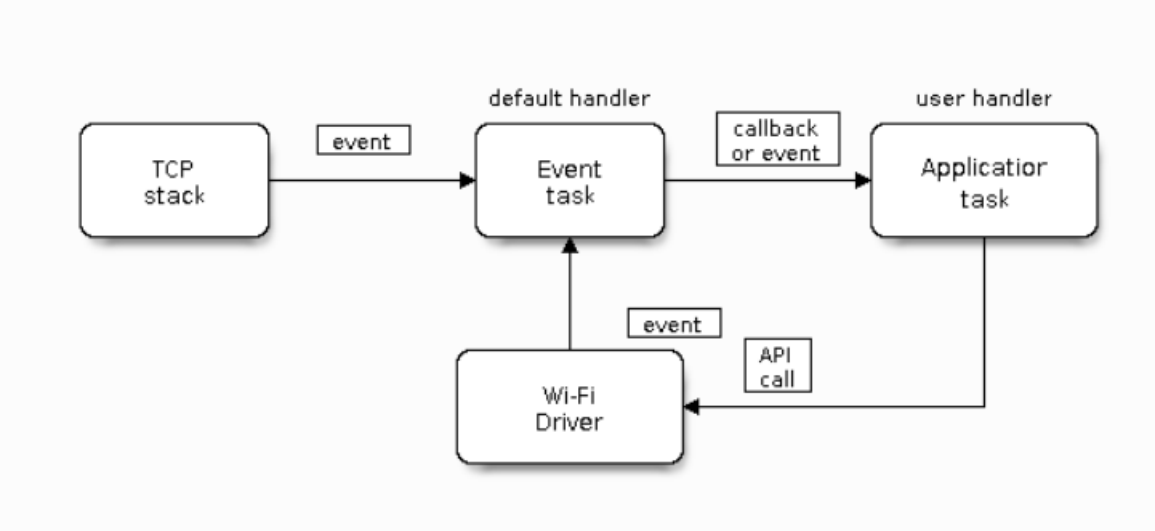
Programszervezés az ESP32 szoftverében

Felhasznált driverek

Az ESP IDF segítségével bizonyos funkciók kezelése driverekkel történik, amelyekhez API segítésével férhetünk hozzá. Ennek előnye, hogy a szoftver írása közben egy olyan projektnél, ahol pl. a TCP/IP stack-re erőforrásként van szükség, nem szükséges a tényleges stack kezelésének lépéseit felügyelni.

A projekt megvalósítása során a TCP/IP stack (lwIP), illetve a Wifi kezelése driverek segítségével készült.

Példa egy driver működésére a következő ábra: [16]



5.ábra

Megvalósított FreeRTOS taskok

A SWAN kliens funkcióit a beágyazott operációs rendszerben taskok valósítják meg, melyeket az ESP32 mikrovezérlőhöz kiegészített FreeRTOS ütemező vezérel. A taskok létrehozásakor konkrét processzor maghoz rendelhető egy-egy task. Amennyiben nincs ilyen hozzárendelés az ütemező bármely processzor magon futtathatja a taskokat.

A jelenlegi implementáció nem tartalmaz olyan taskot, amely futása dedikált processzor magon történne.

Main app Task

A main app task inicializálási feladatokat lát el. Itt kerül létrehozásra a többi task is, illetve ehhez a taskhoz tartozó memóriaterületen vannak regisztrálva a Wifi és az MQTT event handler-hez tartozó callback függvények.

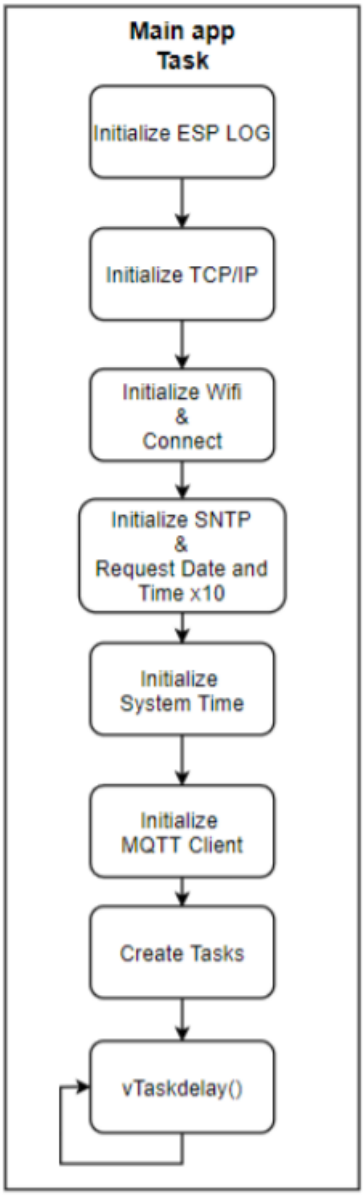
Az ESP LOG struktúra teszi lehetővé az IDF könyvtári függvényei számára az egységes és dokumentált hibakezelést. UART-on keresztül hibaüzenetek küldésére alkalmassá válik a rendszer, ez segíti a fejlesztést és a hibakeresést is.

TCP/IP-hez lwIP használatát ajánlja a gyártó, melyhez kész drivert is biztosít. A Wifi driver konfigurálása után a kapcsolat létrehozása is megtörténik.

NTP szervertől többszöri probálkozással unix time lekérése történik, ennek segítségével időbélyegek készíthetők az adatokhoz, illetve a hosszú várakozási idők is nyomon követhetőek.

MQTT kliens bejegyzése után a hozzátartozó Task végez ciklikusan műveleteket, de a különböző eseményekhez tartozó struktúra is itt kerül definiálásra.

Taskok létrehozása után már a hardveres funkciók is működésbe lépnek.



6.ábra

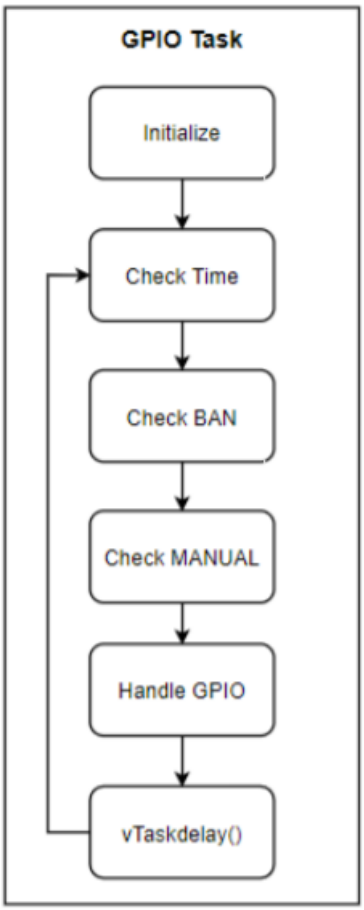
GPIO Task

A hardveres funkció demonstrációjaként egy LED vezérlése került megvalósításra, viszont amennyiben egy kliens több „szelepet” vezérelne, itt lehetne inicializálni ehhez a GPIO-kat.

A task ciklikus működés közben időbélyeget kér, majd elkezdi az ellenőrzést a tiltással vagy a manuális öntözéssel kapcsolatban. Manuális öntözés indításakor MQTT üzenetet is publikál, ezzel jelezve a szerver felé, hogy reagált a kérésre. A manuális öntözés 15 perc után automatikusan leáll.

Amennyiben se tiltás, se manuális öntözés parancs nem érkezett, az ADC által mért talaj nedvességtartalom alapján dönt a rendszer, hogy bekapcsolja-e az öntözést. Jelenleg 50 % -os érték alatt kapcsol.

A bekapcsolt állapot 5 percig tart, majd ezt 5 perces „öntiltás” követi. Így van ideje a növényeknek beszívni a kilocsolt vizet, illetve a szivattyú motorja sincs túl gyorsan ki-be kapcsolgatva a beállás körül.



7.ábra

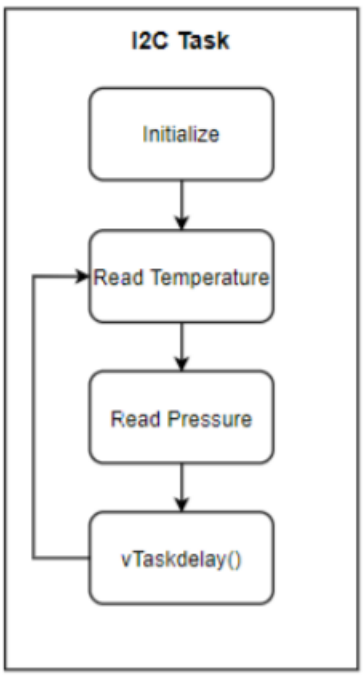
I2C Task

Az inicializálás során sok adatstruktúra feltöltése szükséges, hiszen nem csak az ESP32 I2C buszának használatára kell felkészíteni a taskot, hanem a Bosch által kiadott driverhez szükséges adatszerkezeteket is elő kell állítani.

A buszt szoftverből állítható felhúzó ellenállásokkal 100 kHz-es órajellel használom.

Amennyiben a szenzor megtalálható a buszon, hibaüzenet nélkül várakozásba kezd a szenzor, mérés nélkül. Ennek oka, hogy a szenzor bekapcsolás után nem olvasható ki egyből, viszont a minimum hozzáférési idő kiszámítható a beállításokból.

A task ciklikus része kiolvassa a szenzor hőmérséklet és nyomás regisztereit és különböző számábrázolási módokkal elmenti azokat.



8.ábra

ADC Task

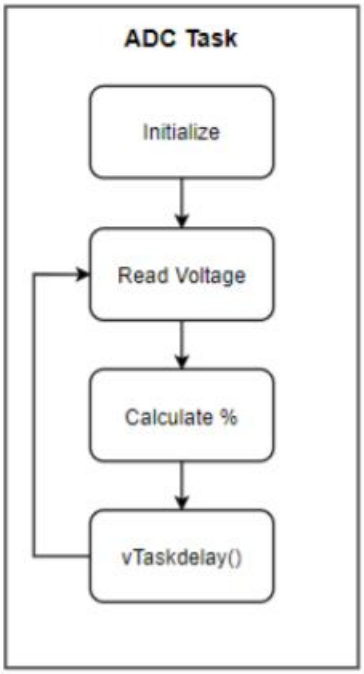
Jelenleg a funkciók bemutatására egy szenzor van használatban, így az ADC1 egyetlen egy csatornájának kiolvasása történik. Az ADC 1 V-os referencia feszültséggel rendelkezik, ezért a szenzor 0-3.3V közötti értékét 11 dB-es csillapítás mellett lehet mérni, erre van az ADC-ben szűrő, melyet szoftveresen kell beállítani.

Emellett az ADC-t kalibrálni is van lehetőség szoftveresen, erre szintén az inicializálás során kerül sor.

A task ciklikusan olvassa ki a szenzor feszültségét, majd maradékos osztás segítségével százalékos értéket számít.

$$Soil\ moisture\ [\%] = 100 - \frac{U_{meas}\ [V]}{3.3\ [V]} \cdot 100$$

A kivonás azért szükséges, mivel a szenzor maximum feszültséget ad ki magából, ha minimális nedvességet mér.



9.ábra

MQTT Task

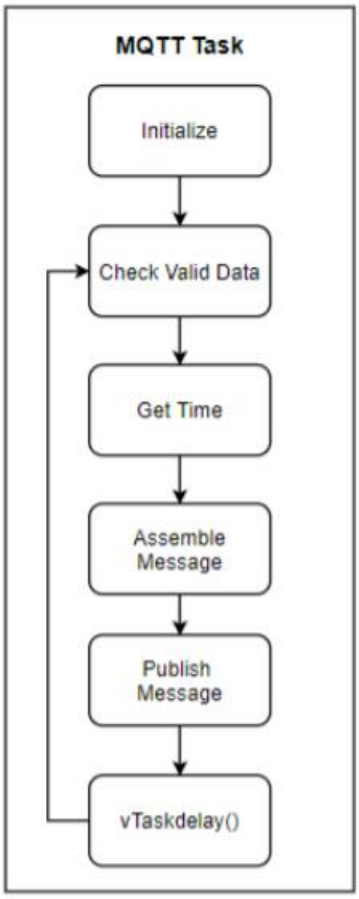
Az MQTT egy könnyű, alacsony erőforrás és energia igényű hálózati kommunikációt támogató protokoll. Elsődlegesen TCP/IP-re épül, de más hálózaton is használható. Publish-subscribe architektúra alapján továbbít üzeneteket eszközök között. Alapvetően egy bróker (szerver) és kliensek közötti a kommunikáció. Egy üzenet egy témába kerül továbbításra, és akik feliratkoztak egy témára megkapják azt.

Lehetséges a csatlakozáshoz/lecsatlakozáshoz üzeneteket rendelni, sőt akár nem üzemszerű lecsatlakozás esetén is üzenetet definiálni (LWT). Így rossz hálózati kapcsolati körülmények között is használható.

Az üzenetek QoS kategóriákba sorolhatóak aszerint, hogy hányszor próbálkoznak meg a résztvevők a kézbesítéssel. [17]

Az ESP32 kliensként csatlakozik a SWAN szerveren futó mosquitto mqtt brókerhez.

Sikeres csatlakozás /topic/qos0 témába iratkozik fel a kliens, üzeneteket pedig /topic/qos1 témába küld. A témák elnevezése utal az alkalmazott QoS szintre is.



10.ábra

Az alkalmazásom során MQTT-n kétféle módon is kommunikálok: a task ciklikusan küld időbélyeggel ellátott üzeneteket, emellett MQTT eseményekre is reagál a rendszer, változókat ír át érkezett üzenetek szerint, UART-on log üzeneteket küld bármilyen esemény történésének hatására.

Event	Reaction
Connected	Log to UART
Disconnected	Log to UART
Subscribed	Log to UART
Unsubscribed	Log to UART
Published	Log to UART
Data Recieved	Log to Uart & set variable
Error	Log to UART
Default	Log to UART

11.ábra

SWAN szerver

Platform választás

Ebben a félévben a fókusz a kliens kipróbálásán volt, viszont fontosnak tartottam, hogy a teljes rendszer ötletét demonstrálni lehessen valamilyen formában.

A szerver alapvető funkcióinak megvalósításához, akár egy személyi számítógép is megfelel, viszont már a korai tesztekhez is szükség volt 1-2 napos futások megfigyelésére, ezért inkább egyből dedikált megoldást választottam.

Raspberry Pi Zero W single board computert választottam központi számítógépnek. Ennek oka, hogy sokféle Linux disztribúció kerülhet rá, illetve 40 GPIO szabadon van hagyva és ki van vezetve PCBre, így akár a központi egység is elláthat beágyazott feladatokat is.

Operációs rendszernek a Dietpi-t választottam [18], ami egy terminál felületet biztosító lightweight linuxon alapuló szoftver. Az SBC erőforrásait így nem foglalják le a grafikai felülethez tartozó különböző szoftveres feladatok. A számítógéphez SSH-n keresztül lehet hozzáférni.

Telepített szoftverek

Mosquitto

MQTT bróker, amelyet C nyelven implementáltak. Nyílt forráskódú, elterjedt IoT alkalmazásokban. [19]

SQLite

SQL adatbázis implementáció, C nyelven írták. Nyílt forráskódú és az egyik legelterjedtebb a világon. [20]

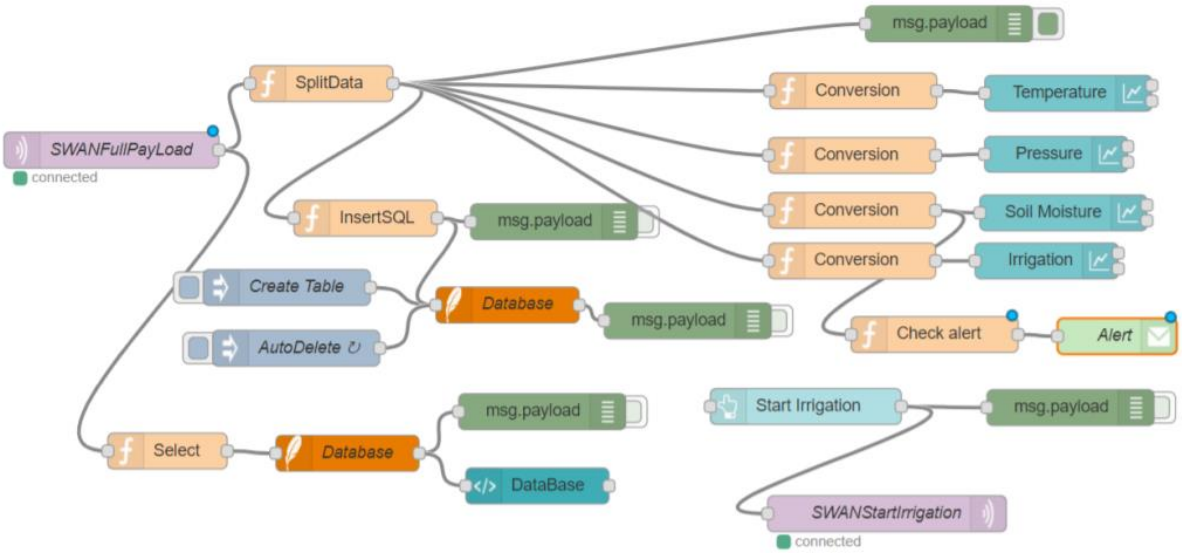
NodeRED

IoT alkalmazásokhoz készült programozási eszköz, amely hardver elemek és webes API-k összekötésében segít. Tartalmaz egy alapvető funkciókat megvalósító frontend web interface-t az elkészített alkalmazások megjelenítésére is. [21]

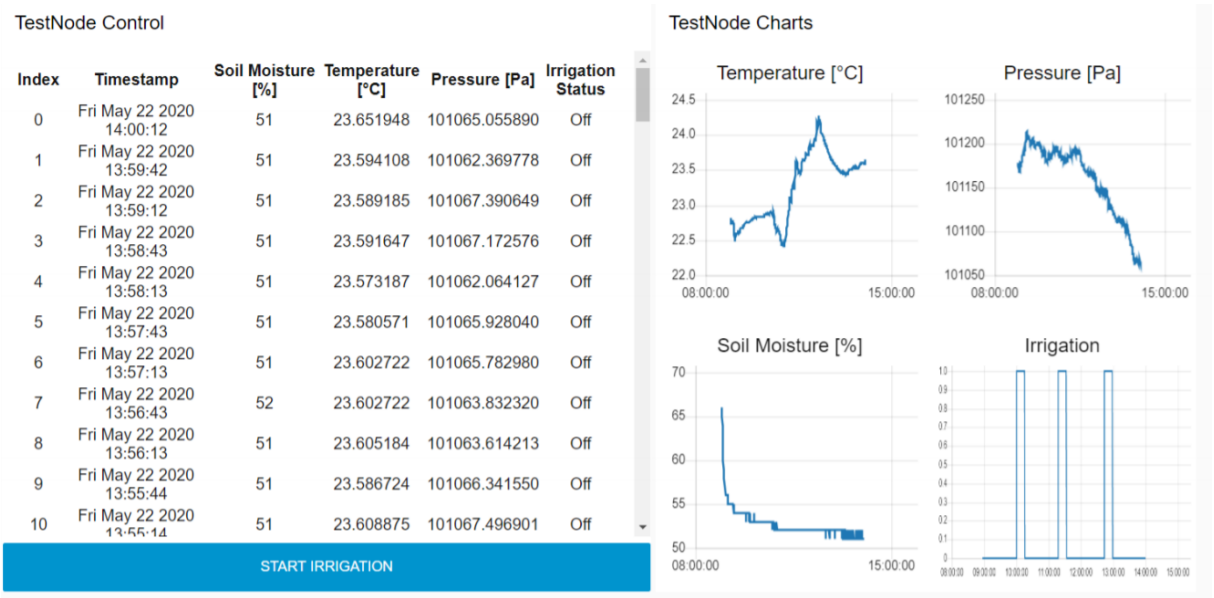
Megvalósított NodeRED szoftver

A telepített szoftver elemek indítása után, a feladat arra egyszerűsödött, hogy a NodeRED fejlesztő környezetében funkcióblokkokból összerakjam a demonstráláshoz szükséges programot, majd azt NodeRED kontrollpanel webes felületén megjelenítsem a lokális hálózaton.

A mellékelt ábrán látható ez a kapcsolás, amely az MQTT-n érkező adatokat szétválogatja, adatbázisba menti és megjeleníti azokat szöveges és grafikus formában. Az adatbázis karbantartása (méret korlátozás automatikus törléssel) is innen történik. Plusz funkcióként, ha a talaj nedvességtartalma 90% fölé emelkedik, akkor figyelmeztető emailt küld a rendszer.



12.ábra



13. ábra

Záró gondolatok

Nem egyszerű egy ilyen méretű műszaki probléma önálló feldolgozása és megoldása. Az alapötlet nem hangzik bonyolultnak, hosszadalmasnak vagy realizálhatatlannak, ennek ellenére rengeteg tématerületről kell ismereteket szerezni a megoldáshoz.

Azt gondolom, hogy a téma megfelelő a tanuláshoz, nem csak az érintett tématerületekről szerezhető tudás miatt, hanem azok koordinálása és felhasználása miatt is. A műszaki tervezés során szükség van a megszerzett tudás praktikus felhasználására, és úgy éreztem ez a projekt lehetőséget adott ennek gyakorlására is.

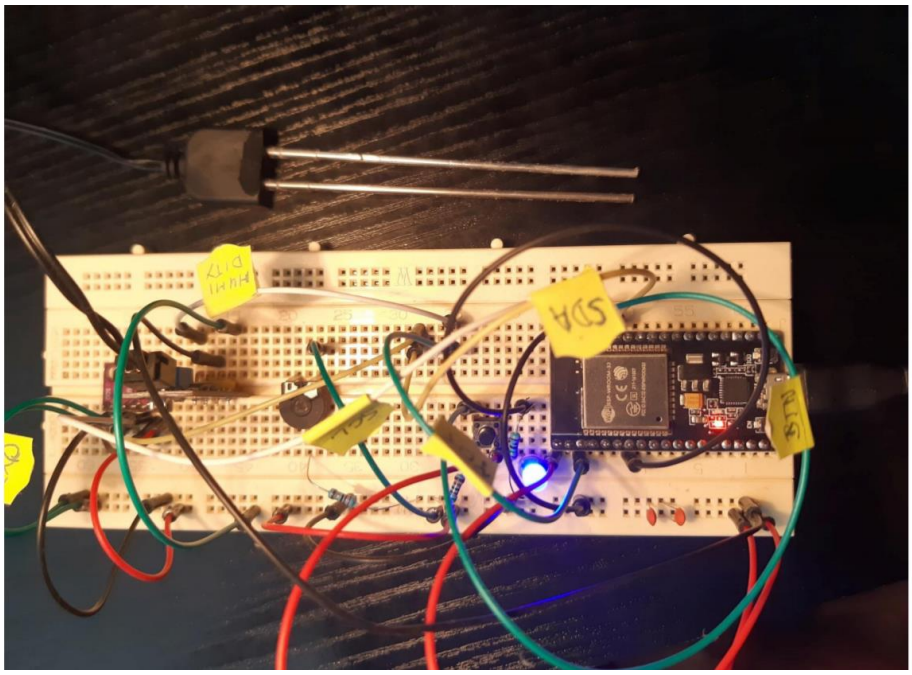
Az elkészült prototípus igazolja, hogy az elképzelés megvalósítható, illetve, hogy érdemes is elkészíteni. Mindenképpen továbbfejleszthető, fejlesztendő és erre megfelelő alapot biztosít a most kialakított szoftveres/hardveres környezet.

A fejlesztés következő lépése egyértelműen a szolenoid szelepet üzemeltető elektronika megtervezése, illetve a kapacitív mérési elven működő szenzor alkalmazása. Emellett fel lehetne készíteni a rendszert, több kliens fogadására a szerver oldalon. Amennyiben folytatásra kerül sor, érdemes esetleg az ESP32 mikrokontroller Bluetooth perifériájának kihasználása a kliens – kliens közötti kommunikációra.

A rendszert a kinti környezetre fel kell készíteni, az ezzel járó időjárási körülmények figyelembevételével kell elkészíteni a már nem prototípusként működő hardvert.

Érdekes fejlesztési lehetőség lehet még, az akkumulátoros vagy telepes táplálás a kliensek oldalán.

Összességében, az ebben a félévben feldolgozott anyag és előkészített hardver/szoftver alkalmas sok IoT-hoz tartozó területre való belépéshez.



14.ábra

Irodalomjegyzék

[1] LoRa technológia összefoglaló dokumentáció
https://buildmedia.readthedocs.org/media/pdf/lora/latest/lora.pdf

[2] Bluetooth hatótávolság az ESP32 SoC-vel
https://www.esp32.com/viewtopic.php?t=6959

[3] ESP32 extrém nagy hatótávolságú wifi kommunikáció
https://www.espressif.com/en/media_overview/news/esp32%E2%80%99s-wi-fi-range-extended-10-km-directional-antenna

[4] ESP8266 dokumentáció
https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[5] ESP32 technical reference manual
https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[6] ESP32 FreeRTOS kiegészítése
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html

[7] Arduino hivatalos dokumentáció és források
https://www.arduino.cc/

[8] ESP32 IDF programming tools
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html

[9] ESP IDF plugin for Eclipse
https://github.com/espressif/idf-eclipse-plugin#Prerequisites

[10] ESP IDF CMake Project szerkezete
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html

[11] I2C dokumentáció
https://i2c.info/

[12] BMP280 dokumentáció
https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf

[13] BMP280 driver github elérése
https://github.com/BoschSensortec/BMP280_driver

[14] Rezisztív mérés (absztrakt alapján)
https://iopscience.iop.org/article/10.1088/0508-3443/2/4/301

[15] Kapacitív mérés (absztrakt alapján)
https://www.sciencedirect.com/science/article/pii/0022169495030034

[16] Wifi driver működése
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html

[17] MQTT protokoll
https://en.wikipedia.org/wiki/MQTT

[18] Dietpi
https://dietpi.com/phpbb/viewtopic.php?t=5

[19] Mosquitto MQTT broker
https://mosquitto.org/

[20] SQLite
https://www.sqlite.org/index.html

[21] NodeRED
https://nodered.org/

Ábrajegyzék

Mellékletek

A teljes projekt megtalálható a következő linken:

https://github.com/Zaion-BM/SWAN

app_main.c

```
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "tcpip_adapter.h"
#include "protocol_examples_common.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"
#include "freertos/event_groups.h"

#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"

#include "esp_sntp.h"
#include "esp_system.h"
#include "esp_log.h"
#include "esp_attr.h"
#include "esp_sleep.h"

#include "protocol_examples_common.h"

#include "mqtt_client.h"

#include "driver/gpio.h"
#include "sdkconfig.h"

#include "esp_pm.h"

//Custom headers
#include "components/ADC/ADC.h"
#include "components/I2C/I2C.h"

//String for ESP debug port
static const char *TAG = "SWAN_CLIENT";

//Defines
#define LED 2
#define valveLED 32
```

```
49 #define BUTTON 23
50
51 //Global variables
52 int humidityPercent = 0;
53 double temperature = 0;
54 double pressure = 0;
55 int valveBAN = 0;
56 int manualIrrigation = 0;
57 int valveState = 0;
58
59 //Task handlers for custom scheduling
60 TaskHandle_t ADCTaskHandler;
61 TaskHandle_t I2CTaskHandler;
62 TaskHandle_t MQTTTaskHandler;
63
64 //Time keeping variables
65 char strftime_buf[64]; //date
66 struct tm timeinfo = { 0 };
67 time_t now = 0; //global time
68
69
70
71 static esp_err_t mqtt_event_handler_cb(esp_mqtt_event_handle_t event)
72 {
73     esp_mqtt_client_handle_t client = event->client;
74     int msg_id;
75     // your_context_t *context = event->context;
76     switch (event->event_id) {
77         case MQTT_EVENT_CONNECTED:
78             msg_id = esp_mqtt_client_subscribe(client, "/topic/qos0", 0);
79             ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);
80             break;
81         case MQTT_EVENT_DISCONNECTED:
82             ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
83             break;
84         case MQTT_EVENT_SUBSCRIBED:
85             break;
86         case MQTT_EVENT_UNSUBSCRIBED:
87             ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
88             break;
89         case MQTT_EVENT_PUBLISHED:
90             ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
91             break;
92         case MQTT_EVENT_DATA:
93             ESP_LOGI(TAG, "MQTT_EVENT_DATA");
94             printf("TOPIC=%.*s\r\n", event->topic_len, event->topic);
95             printf("DATA=%.*s\r\n", event->data_len, event->data);
96             if(strcmp(event->data,"1")){
97                    printf("Manual irrigation on\r\n");
98                    manualIrrigation = 1;
99             }
100            break;
101        case MQTT_EVENT_ERROR:
102            ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
103            break;
104        default:
```

```
105             ESP_LOGI(TAG, "Other event id:%d", event->event_id);
106             break;
107     }
108     return ESP_OK;
109 }
110 
111 static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id,
    void *event_data) {
112     ESP_LOGD(TAG, "Event dispatched from event loop base=%s, event_id=%d", base, event_id);
113     mqtt_event_handler_cb(event_data);
114 }
115 
116  void mqtt_app_start(void *clientptr )
117 {
118           char humidityData[32]=" ";
119           char tempData[32]= " ";
120           char pressureData[32]= " ";
121           char msg[128] =" ";
122 
123           esp_mqtt_client_handle_t client = *(esp_mqtt_client_handle_t*) clientptr;
124 
125     while(1){
126           while( (pressure==0)) {vTaskDelay(1000/portTICK_PERIOD_MS);}
127 
128           sprintf(humidityData, "%d", humidityPercent);
129           sprintf(tempData, "%f", temperature);
130           sprintf(pressureData, "%f", pressure);
131 
132           time(&now);
133           localtime_r(&now, &timeinfo);
134           strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
135           strcpy(msg,strftime_buf);
136 
137           strcat(msg,",");
138           strcat(msg,humidityData);
139           strcat(msg,",");
140           strcat(msg,tempData);
141           strcat(msg,",");
142           strcat(msg,pressureData);
143           strcat(msg,",");
144           if(valveState == 1){strcat(msg,"On");}
145           if(valveState == 0){strcat(msg,"Off");}
146 
147           esp_mqtt_client_publish(client, "/topic/qos1",msg, 0, 1, 0);
148           strcpy(msg," ");
149 
150           vTaskDelay(30000/portTICK_PERIOD_MS);
151     }
152 }
153 
154 #ifdef CONFIG_SNTP_TIME_SYNC_METHOD_CUSTOM
155 void sntp_sync_time(struct timeval *tv)
156 {
157    settimeofday(tv, NULL);
158    ESP_LOGI(TAG, "Time is synchronized from custom code");
159    sntp_set_sync_status(SNTP_SYNC_STATUS_COMPLETED);
```

```
160 }
161 #endif
162
163 void time_sync_notification_cb(struct timeval *tv)
164 {
165     ESP_LOGI(TAG, "Notification of a time synchronization event");
166 }
167
168  static void initialize_sntp(void)
169  {
170      ESP_LOGI(TAG, "Initializing SNTP");
171      sntp_setoperatingmode(SNTP_OPMODE_POLL);
172      sntp_setservername(0, "pool.ntp.org");
173      sntp_set_time_sync_notification_cb(time_sync_notification_cb);
174  #ifdef CONFIG_SNTP_TIME_SYNC_METHOD_SMOOTH
175      sntp_set_sync_mode(SNTP_SYNC_MODE_SMOOTH);
176  #endif
177      sntp_init();
178  }
179
180 void GPIO_Task(void *clientptr){
181     gpio_pad_select_gpio(valveLED);
182     gpio_set_direction(valveLED, GPIO_MODE_OUTPUT);
183     gpio_set_level(valveLED, 0);
184     valveState = 0;
185
186     char humidityData[32]=" ";
187          char tempData[32]= " ";
188          char pressureData[32]= " ";
189          char msg[128] =" ";
190          struct tm *tmp;
191
192     esp_mqtt_client_handle_t client = *(esp_mqtt_client_handle_t*) clientptr;
193
194          while(1){
195                   time(&now);
196                   localtime_r(&now, &timeinfo);
197                   strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
198                   tmp = gmtime(&now);
199                   printf("%d",tmp->tm_hour);
200                   if((tmp->tm_hour<6) || (tmp->tm_hour>23))  {valveBAN =1;}
201                   if((tmp->tm_hour>10) && (tmp->tm_hour<18)) {valveBAN =1;}
202
203                   if((manualIrrigation==1) && (valveBAN==0) ){
204
205                            manualIrrigation = 0;
206                            gpio_set_level(valveLED, 1);
207                            valveState=1;
208
209                            sprintf(humidityData, "%d", humidityPercent);
210                            sprintf(tempData, "%f", temperature);
211                            sprintf(pressureData, "%f", pressure);
212
213                   strcpy(msg,strftime_buf);
214                            strcat(msg,",");
215                            strcat(msg,humidityData);
```

```
216                             strcat(msg,",");
217                     strcat(msg,tempData);
218                     strcat(msg,",");
219                     strcat(msg,pressureData);
220                             strcat(msg,",On");
221
222                             esp_mqtt_client_publish(client, "/topic/qos1",msg, 0, 1, 0);
223                             strcpy(msg," ");
224
225                             vTaskDelay(900000/portTICK_PERIOD_MS);
226                     }
227                     else{
228                             if((humidityPercent<50) && (valveBAN == 0)){
229                                     valveBAN = 1;
230                                     gpio_set_level(valveLED, 1);
231                                     valveState = 1;
232                                     vTaskDelay(300000/portTICK_PERIOD_MS);
233                             }
234                             else{
235                                     gpio_set_level(valveLED, 0);
236                                     valveState = 0;
237                                     vTaskDelay(60000/portTICK_PERIOD_MS);
238                                     valveBAN = 0;
239                             }
240                     }
241             vTaskDelay(1000/portTICK_PERIOD_MS);
242             }
243 }
244
245 void app_main()
246 {
247     ESP_LOGI(TAG, "[APP] Startup..");
248     ESP_LOGI(TAG, "[APP] Free memory: %d bytes", esp_get_free_heap_size());
249     ESP_LOGI(TAG, "[APP] IDF version: %s", esp_get_idf_version());
250
251     esp_log_level_set("*", ESP_LOG_INFO);
252     esp_log_level_set("MQTT_CLIENT", ESP_LOG_VERBOSE);
253     esp_log_level_set("MQTT_EXAMPLE", ESP_LOG_VERBOSE);
254     esp_log_level_set("TRANSPORT_TCP", ESP_LOG_VERBOSE);
255     esp_log_level_set("TRANSPORT_SSL", ESP_LOG_VERBOSE);
256     esp_log_level_set("TRANSPORT", ESP_LOG_VERBOSE);
257     esp_log_level_set("OUTBOX", ESP_LOG_VERBOSE);
258
259     ESP_ERROR_CHECK(nvs_flash_init());
260     tcpip_adapter_init();
261     ESP_ERROR_CHECK(esp_event_loop_create_default());
262
263     /* This helper function configures Wi-Fi or Ethernet, as selected in menuconfig.
264      * Read "Establishing Wi-Fi or Ethernet Connection" section in
265      * examples/protocols/README.md for more information about this function.
266      */
267     ESP_ERROR_CHECK(example_connect());
268
269     initialize_sntp();
270
271     // wait for time to be set
```

```
272     int retry = 0;
273     const int retry_count = 10;
274     while (sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET && ++retry < retry_count) {
275             ESP_LOGI(TAG, "Waiting for system time to be set... (%d/%d)", retry,
    retry_count);
276             vTaskDelay(2000 / portTICK_PERIOD_MS);
277         }
278 
279     time(&now);
280 
281     // Set timezone to CET and print local time
282     setenv("TZ", "CET-02:00:00CEST-03:00:00,M10.1.0,M3.3.0", 1);
283     tzset();
284     localtime_r(&now, &timeinfo);
285     strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
286     ESP_LOGI(TAG, "The current date/time in Budapest is: %s", strftime_buf);
287 
288     //Setting up mqtt
289     esp_mqtt_client_config_t mqtt_cfg = {.uri = CONFIG_BROKER_URL,};
290     esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
291     esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, client);
292     esp_mqtt_client_start(client);
293     const esp_mqtt_client_handle_t* clientptr = &client;
294 
295     xTaskCreate(ADC_Task,"ADC_read", 3*1024, NULL, 5, NULL);
296     xTaskCreate(I2C_Task,"I2C_Task", 3*1024, NULL, 5, NULL);
297     xTaskCreate(mqtt_app_start,"MQTT Task",3*1024,(void *) clientptr,5,NULL);
298     xTaskCreate(GPIO_Task,"GPIO_Task",3*1024,(void *) clientptr,4,NULL);
299 
300     while(1){
301           vTaskDelay(1000/portTICK_PERIOD_MS);
302     }
303 
304 
305 }
```

ADC.h

```
1  /*
2   * ADC.h
3   *
4   *  Created on: Mar 19, 2020
5   *      Author: mate.berta
6   */
7 
8 
9  #ifndef ADC_H
10 #define ADC_H
11 
12 #include "driver/adc.h"
13 #include "esp_adc_cal.h"
14 #include "freertos/FreeRTOS.h"
15 #include "freertos/task.h"
16 
```

```
17 extern int humidityPercent;
18 extern TaskHandle_t ADCTaskHandler;
19 extern TaskHandle_t I2CTaskHandler;
20 void ADC_Task();
21
22 #endif /*ADC_H*/
```

ADC.c

```
 1    #include "ADC.h"
 2
 3
 4    void ADC_Task(){
 5
 6              adc1_config_width(ADC_WIDTH_BIT_12);
 7              adc1_config_channel_atten(ADC_CHANNEL_0, ADC_ATTEN_DB_11);
 8
 9              //Characterize ADC at particular attenuation
                esp_adc_cal_characteristics_t *adc_chars = calloc(1,
10   sizeof(esp_adc_cal_characteristics_t));
11       esp_adc_cal_value_t val_type = esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_11,
     ADC_WIDTH_BIT_12, 1100, adc_chars);
12
13       //Check type of calibration value used to characterize ADC
14       if (val_type == ESP_ADC_CAL_VAL_EFUSE_VREF) {
15           printf("eFuse Vref\n");
16       }
17       else if (val_type == ESP_ADC_CAL_VAL_EFUSE_TP) {
18           printf("Two Point\n");
19       }
20       else {
21           printf("Default\n");
22       }
23       while(1){
24            uint32_t reading =  adc1_get_raw(ADC1_CHANNEL_0);
25            uint32_t voltage = esp_adc_cal_raw_to_voltage(reading, adc_chars);
26            humidityPercent = 100-(reading*100 / 4095);
27            printf("ADC1_CH0 value:%d, voltage:%d mV\n",reading,voltage);
28            printf("Soil humidity: %d%% \n",humidityPercent);
29            vTaskDelay(14999/portTICK_PERIOD_MS);
30       }
31   }
```

I2C.h

```
1 /*
2  * I2C.h
3  *
4  *  Created on: Mar 21, 2020
5  *      Author: mate.berta
6  */
7
```

```
 8 #ifndef MAIN_I2C_H_
 9 #define MAIN_I2C_H_
10
11 #include "freertos/FreeRTOS.h"
12 #include "freertos/task.h"
13 #include "driver/i2c.h"
14 #include "driver/gpio.h"
15 #include <stdio.h>
16 #include "bmp280.h"
17 #include "bmp280_defs.h"
18 #include "esp_err.h"
19
20 #define SDA_GPIO 21
21 #define SCL_GPIO 22
22
23 extern double temperature;
24 extern double pressure;
25
26 extern TaskHandle_t I2CTaskHandler;
27 extern TaskHandle_t MQTTTaskHandler;
28
29 void delay_ms(uint32_t period_ms);
30
31 int8_t i2c_reg_write(uint8_t i2c_addr, uint8_t reg_addr, uint8_t *reg_data, uint16_t length);
32
33 int8_t i2c_reg_read(uint8_t i2c_addr, uint8_t reg_addr, uint8_t *reg_data, uint16_t length);
34
35 void print_rslt(const char api_name[], int8_t rslt);
36
37 void I2C_Task();
38
39 #endif /* MAIN_I2C_H_ */
```

I2C.c

```
 1 /*
 2  * I2C.c
 3  *
 4  *  Created on: Mar 21, 2020
 5  *      Author: pep
 6  */
 7
 8 #include "I2C.h"
 9 /*!
10  *
11  *
12  *  @param[in] period_ms  : the required wait time in milliseconds.
13  *  @return void.
14  *
15  */
16 void delay_ms(uint32_t period_ms)
17 {
18     /* Implement the delay routine according to the target machine */
19           ets_delay_us(period_ms * 1000);
```

```
20 }
21
22 void i2c_master_init() {
23     i2c_config_t i2c_config = {
24             .mode = I2C_MODE_MASTER,
25             .sda_io_num = SDA_GPIO,
26             .scl_io_num = SCL_GPIO,
27             .sda_pullup_en = GPIO_PULLUP_ENABLE,
28             .scl_pullup_en = GPIO_PULLUP_ENABLE,
29             .master.clk_speed = 100000
30     };
31     i2c_param_config(I2C_NUM_0, &i2c_config);
32     printf("Driver install... ");
33     printf("%s\n",esp_err_to_name(i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER, 0, 0, 0)));
34 }
35
36 /*!
37  *  @brief Function for writing the sensor's registers through I2C bus.
38  *
39  *  @param[in] i2c_addr : sensor I2C address.
40  *  @param[in] reg_addr : Register address.
41  *  @param[in] reg_data : Pointer to the data buffer whose value is to be written.
42  *  @param[in] length   : No of bytes to write.
43  *
44  *  @return Status of execution
45  *  @retval 0 -> Success
46  *  @retval >0 -> Failure Info
47  *
48  */
49 int8_t i2c_reg_write(uint8_t i2c_addr, uint8_t reg_addr, uint8_t *reg_data, uint16_t length)
50 {
51
52           /* Implement the I2C write routine according to the target machine. */
53               int8_t iError;
54               esp_err_t esp_err;
55               i2c_cmd_handle_t cmd_handle = i2c_cmd_link_create();
56               i2c_master_start(cmd_handle);
57               i2c_master_write_byte(cmd_handle, (i2c_addr << 1) | I2C_MASTER_WRITE, true);
58               i2c_master_write_byte(cmd_handle, reg_addr, true);
59               i2c_master_write(cmd_handle, reg_data, length,true);
60               i2c_master_stop(cmd_handle);
61               esp_err = i2c_master_cmd_begin(I2C_NUM_0, cmd_handle, 1000 /
   portTICK_PERIOD_MS);
62               if (esp_err == ESP_OK) {
63                   iError = 0;
64               } else {
65                   iError = -1;
66               }
67               i2c_cmd_link_delete(cmd_handle);
68               return iError;
69 }
70
71 /*!
72  *  @brief Function for reading the sensor's registers through I2C bus.
73  *
74  *  @param[in] i2c_addr : Sensor I2C address.
```

```
 75  *  @param[in] reg_addr : Register address.
 76  *  @param[out] reg_data    : Pointer to the data buffer to store the read data.
 77  *  @param[in] length   : No of bytes to read.
 78  *
 79  *  @return Status of execution
 80  *  @retval 0 -> Success
 81  *  @retval >0 -> Failure Info
 82  *
 83  */
 84 int8_t i2c_reg_read(uint8_t i2c_addr, uint8_t reg_addr, uint8_t *reg_data, uint16_t length)
 85 {
 86          /* Implement the I2C read routine according to the target machine. */
 87                  int8_t iError;
 88              esp_err_t esp_err;
 89              i2c_cmd_handle_t cmd_handle = i2c_cmd_link_create();
 90
 91              i2c_master_start(cmd_handle);
 92              i2c_master_write_byte(cmd_handle, (i2c_addr << 1) | I2C_MASTER_WRITE, true);
 93              i2c_master_write_byte(cmd_handle, reg_addr, true);
 94
 95              i2c_master_start(cmd_handle);
 96              i2c_master_write_byte(cmd_handle, (i2c_addr << 1) | I2C_MASTER_READ, true);
 97
 98              if (length > 1) {
 99                  i2c_master_read(cmd_handle, reg_data, length - 1, I2C_MASTER_ACK);
100              }
101              i2c_master_read_byte(cmd_handle, reg_data + length - 1, I2C_MASTER_NACK);
102              i2c_master_stop(cmd_handle);
103
104              esp_err = i2c_master_cmd_begin(I2C_NUM_0, cmd_handle, 1000 /
    portTICK_PERIOD_MS);
105
106              if (esp_err == ESP_OK) {
107                  iError = 0;
108              } else {
109                  iError = -1;
110              }
111
112              i2c_cmd_link_delete(cmd_handle);
113
114
115              return iError;
116 }
117
118
119 /*!
120  *  @brief Prints the execution status of the APIs.
121  *
122  *  @param[in] api_name : name of the API whose execution status has to be printed.
123  *  @param[in] rslt     : error code returned by the API whose execution status has to be
    printed.
124  *
125  *  @return void.
126  */
127 void print_rslt(const char api_name[], int8_t rslt)
128 {
```

```
129     if (rslt != BMP280_OK)
130     {
131         printf("%s\t", api_name);
132         if (rslt == BMP280_E_NULL_PTR)
133         {
134             printf("Error [%d] : Null pointer error\r\n", rslt);
135         }
136         else if (rslt == BMP280_E_COMM_FAIL)
137         {
138             printf("Error [%d] : Bus communication failed\r\n", rslt);
139         }
140         else if (rslt == BMP280_E_IMPLAUS_TEMP)
141         {
142             printf("Error [%d] : Invalid Temperature\r\n", rslt);
143         }
144         else if (rslt == BMP280_E_DEV_NOT_FOUND)
145         {
146             printf("Error [%d] : Device not found\r\n", rslt);
147         }
148         else
149         {
150             /* For more error codes refer "*_defs.h" */
151             printf("Error [%d] : Unknown error code\r\n", rslt);
152         }
153     }
154 }
155
156
157 /*!
158  *  @brief Example shows basic application to configure and read the temperature.
159  */
160
161 void I2C_Task(){
162           printf("I2C_Task start\n");
163           i2c_master_init();
164           printf("i2c_master_init finished\n");
165
166           int8_t rslt;
167           struct bmp280_dev bmp;
168           struct bmp280_config conf;
169           struct bmp280_uncomp_data ucomp_data;
170           int32_t temp32;
171           double temp;
172           uint32_t pres32;
173           uint32_t pres64;
174           double pres;
175
176           /* Map the delay function pointer with the function responsible for implementing
    the delay */
177           bmp.delay_ms = delay_ms;
178
179           /* Assign device I2C address based on the status of SDO pin (GND for PRIMARY(0x76)
    & VDD for SECONDARY(0x77)) */
180           bmp.dev_id = BMP280_I2C_ADDR_PRIM;
181
182           /* Select the interface mode as I2C */
```

```
183           bmp.intf = BMP280_I2C_INTF;
184
              /* Map the I2C read & write function pointer with the functions responsible for
185 I2C bus transfer */
              bmp.read = i2c_reg_read;
186           bmp.write = i2c_reg_write;
187
188           rslt = bmp280_init(&bmp);
189           print_rslt(" bmp280_init status", rslt);
190
191           /* Always read the current settings before writing, especially when
192            * all the configuration is not modified
193            */
194           rslt = bmp280_get_config(&conf, &bmp);
195           print_rslt(" bmp280_get_config status", rslt);
196
197           /* configuring the temperature oversampling, filter coefficient and output data
198 rate */
              /* Overwrite the desired settings */
199           conf.filter = BMP280_FILTER_COEFF_2;
200
201           /* Temperature oversampling set at 4x */
202           conf.os_temp = BMP280_OS_4X;
203
204           /* Pressure over sampling none (disabling pressure measurement) */
205           conf.os_pres = BMP280_OS_4X;
206
207           /* Setting the output data rate as 1HZ(1000ms) */
208       conf.odr = BMP280_ODR_1000_MS;
209       rslt = bmp280_set_config(&conf, &bmp);
210       print_rslt(" bmp280_set_config status", rslt);
211
212       /* Always set the power mode after setting the configuration */
213       rslt = bmp280_set_power_mode(BMP280_NORMAL_MODE, &bmp);
214       print_rslt(" bmp280_set_power_mode status", rslt);
215
216       /*Waits for measurment values in the registers are valid*/
217       vTaskDelay(1000 / portTICK_PERIOD_MS);
218
219
220
221       while (1){
222
223           /* Reading the raw data from sensor */
224           rslt = bmp280_get_uncomp_data(&ucomp_data, &bmp);
225
226           /* Getting the 32 bit compensated temperature */
227           rslt = bmp280_get_comp_temp_32bit(&temp32, ucomp_data.uncomp_temp, &bmp);
228
229           /* Getting the compensated temperature as floating point value */
230           rslt = bmp280_get_comp_temp_double(&temp, ucomp_data.uncomp_temp, &bmp);
231
232           /* Getting the compensated pressure using 32 bit precision */
233           rslt = bmp280_get_comp_pres_32bit(&pres32, ucomp_data.uncomp_press, &bmp);
234
235           /* Getting the compensated pressure using 64 bit precision */
```

```
236         rslt = bmp280_get_comp_pres_64bit(&pres64, ucomp_data.uncomp_press, &bmp);
237
238         /* Getting the compensated pressure as floating point value */
239         rslt = bmp280_get_comp_pres_double(&pres, ucomp_data.uncomp_press, &bmp);
240
241         /*Print Pressure Data*/
242         printf("UP: %d, P32: %d, P64: %d, P64N: %d, P: %lf\r\n",
243                                                         ucomp_data.uncomp_press,
244                             pres32,
245                             pres64,
246                             pres64 / 256,
247                             pres);
248
249         /*Print Temperature Data*/
250         printf("UT: %d, T32: %d, T: %f \r\n", ucomp_data.uncomp_temp, temp32, temp);
251
252         temperature = (double) temp;
253         pressure = (double) pres;
254
255         /* Sleep time between measurements = BMP280_ODR_10000_MS */
256         vTaskDelay(14999 / portTICK_PERIOD_MS);
257     }
258
259 }
260
```