



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Sebők Bence

VEZETÉK NÉLKÜLI IDŐJÁRÁS ÁLLOMÁS FELHŐ ALAPÚ SZOLGÁLTATÁSOKKAL

KONZULENS

Naszály Gábor

BUDAPEST, 2018

Tartalomjegyzék

Összefoglaló	5
Abstract.....	6
1 Bevezetés	7
1.1 Dolgok Internete (Internet of Things – IoT)	7
1.1.1 Kommunikáció.....	8
1.1.2 Otthoni időjárás állomások	9
1.1.3 Tudományos célok	10
1.2 Saját időjárás állomás	11
2 Tervezés	13
2.1 Feldolgozó egységek.....	15
2.2 Vezeték nélküli kommunikáció	16
2.3 Szenzorok.....	18
2.3.1 DHT22	20
2.3.2 BMP280	21
2.3.3 BH1750FVI	24
2.4 Felhős szolgáltatások	24
2.5 Felhasználói kezelőfelület.....	26
3 Implementáció	29
3.1 Mérőegység.....	29
3.1.1 Tápellátás	31
3.1.2 Kommunikáció.....	31
3.1.3 Szenzorok.....	35
3.2 Központi egység	40
3.2.1 Tápellátás	41
3.2.2 Felhasználói kezelőfelület.....	41
3.2.3 Kommunikáció.....	41
3.3 Felhős szolgáltatások	43
3.3.1 Microsoft Azure	44
3.3.2 Adatbázis	44
3.3.3 Web szerver	46
3.3.4 Webes kliens alkalmazás	47

4 Tesztelés	50
4.1 Hardver prototípus	51
4.2 Tesztkörnyezet	52
4.3 Mérések.....	52
4.3.1 DHT22 GPIO lábának mérése	52
4.3.2 I2C busz mérése.....	53
4.3.3 Mért adatok	55
4.4 Adatok feldolgozása	56
5 Továbbfejlesztési lehetőségek	58
6 Értékelés	59
7 Hivatkozások	61

HALLGATÓI NYILATKOZAT

Alulírott **Sebők Bence**, 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, 2018. 12. 06.

.....
Sebők Bence

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 Dolgok Internete (Internet of Things – IoT)

A Dolgok Internete (Internet of Things – IoT) nagyon népszerű témakör manapság. Az IoT egy hálózat különféle fizikai eszközöknek – például járművek, háztartási gépek vagy bármilyen beágyazott rendszer, ahol van elektronika, szoftver, szenzorok, beavatkozásszervek és valamilyen kapcsolat, amin keresztül adatokat tudnak gyűjteni és cserélni.

Az IoT a gyakorlatban azt jelenti, hogy internet kapcsolattal látjuk el a hétköznapi eszközöket, amik általában nem rendelkeznek ilyenekkel, ellentétben például a laptopokkal vagy okostelefonokkal. Az internet kapcsolat segítségével távolról elérhetővé és vezérelhetővé válnak ezek az eszközök.

Az IoT rendszerek egyik fő irányvonala a háztartások „felokosítása”. A háztartási eszközök hálózatba kötése az okosotthonok és otthon automatizálás témakörébe tartoznak. Az IoT hálózatnak hála a világítás, fűtés, légkondicionálás, valamint a média és biztonsági rendszerek is automatizálhatóak. **Ennek gazdasági előnyei közé tartozik, hogy az automatikus működés révén csökkenteni lehet a villamos energiafelhasználást és így az áramfelhasználásra költött közüzemi számlák végösszegét.**

Az okosotthonok gyakran egy központi egységen – úgynevezett hubon – alapulnak. Ez a hub vezérli a hálózatba kötött egységeket. Egy elterjedt hub az Apple cég által készített HomeKit¹. Az elektronikai gyártók el tudják úgy készíteni az eszközeiket, hogy a HomeKit segítségével ezek iOS operációs rendszert futtató eszközökkel legyen irányíthatóak – akár egy iPhone-nal vagy egy iWatch-csal. A fejlettebb rendszereket akár hangvezérléssel, például a Siri nevű személyi asszisztenssel is vezérelhetjük. Egy másik hasonló megoldás a Samsung vállalat SmartThings² nevű rendszere. Többféle eszköz is elérhető, ami könnyedén csatlakozhat ehhez az okosotthon

¹ <https://www.apple.com/shop/accessories/all-accessories/homekit>

² <https://www.smarthings.com/>

keretrendszerhez. Szintén van applikáció a népszerű telefonos operációs rendszerekre, mint például az Androidra és az iOS-re.

Az okosotthonok egyik elsődleges feltétele, hogy távolról elérhetőek legyenek az otthoni eszközök és tudjunk bizonyos paramétereket a környezetükről. Például egy termosztát vezérléséhez szükséges ismerünk a beltéri hőmérsékletet, hogy ez alapján döntést tudjon hozni a rendszer, hogy növelni vagy csökkenteni kell-e a fűtés teljesítményén.

A szakdolgozatom témája egy egyszerű otthoni időjárásállomás megtervezése és prototípusának elkészítése, hogy az otthoni időjárási paraméterek mérhetőek legyenek és távolról elérhetőek. A rendelkezésre álló adatok alapján pedig automatizált döntéseket lehet hozni a háztartási eszközök vezérléséhez.

1.1.1 Kommunikáció

Az IoT hálózathoz tartozó eszközök gyakran vezeték nélküli módon kommunikálnak egymással, aminek gyakorlati jelentőségű okai vannak: sokszor fizikailag nem megoldható a vezetékek lefektetése vagy túl költséges lenne a sok hosszú vezeték elhelyezése, karbantartása, javítása.

Többféle vezeték nélküli megoldás is elterjed ilyen jellegű rendszereknél:

- Bluetooth Mesh hálózat: a Bluetooth Low Energy (BLE) specifikációra építő hálózat, ahol nagyszámú ³ node kezelése is lehetséges, valamint 100-1000 méteres elméleti hatótávolsággal rendelkezik
- Near-field communication (NFC): két eszköz közötti kommunikációt tesz lehetővé legfeljebb 4 cm-es távolságon belül
- Wi-Fi: helyi hálózatokon alapuló rendszer, ahol vezeték nélküli hozzáférési pontra van szükség a kommunikációhoz, többnyire szórakoztató elektronikai eszközök használják például okostelefonok, laptopok, tabletek stb.

³ https://en.wikipedia.org/wiki/Bluetooth_mesh_networking#Theoretical_limits

- ZigBee: egy alacsony fogyasztású, alacsony sávszélességű technológia, ami személyi helyi hálózatokat tud létrehozni, az elméleti hatótávolsága körülbelül 10-20 méter.
- Egyéb rádiófrekvenciás technológiák: az ISM ⁴ sávban használható frekvenciatartományokban működő protokollok. Nem igényelnek kormányzati engedélyeztetést. Rövid hatótávú, alacsony fogyasztású kommunikációs rendszereknek ajánlják a használatát. Dedikált frekvenciatartományok ⁵ van levédve ezen rendszereknek, amiket biztosan nem használnak kormányzati és telekommunikációs hálózatok, hogy a rendelkezésre álló frekvenciatartományt minél jobban ki tudják használni az ISM sávban működő civil eszközök.

1.1.2 Otthoni időjárás állomások

Az otthoni időjárás állomások nem tudományos célt szolgálnak, ezért teljesen más paraméterekkel rendelkeznek, mint a nagyobb méretű és precízebb meteorológiai állomások, amik alapján például időjárás-előrejelzéseket készítenek.

Az otthoni időjárás állomások főbb jellemzői:

- Alacsony költségűek
- Kisméretűek
- Alacsony a fogyasztásuk – főleg, ha vezeték nélküliek
- Legfeljebb néhány tizedesjegy pontossággal mérnek
- Grafikus kezelőfelülettel rendelkeznek

Egy könnyedén beszerezhető otthoni időjárás állomás például a Fanju FJ3378 Weather Station⁶.

⁴ Industrial, Scientific, Medical, amik a felhasználási területeket jelentik

⁵ <https://www.itu.int/net/ITU-R/terrestrial/faq/index.html#g013>

⁶ <https://www.aliexpress.com/item/FanJu-FJ3378-Digital-Alarm-Clock-Weather-Station-Wall-Indoor-Outdoor-Temperature-Humidity-Watch-Moon-Phase-Forecast/32850377398.html>



1.1 ábra: Fanju FJ3378 Weather Station

Ez az eszköz közel 38 eurós áron megrendelhető több különböző webáruházból is. Teljesül rá minden fenti kritérium: alacsony áron beszerezhető, kisméretű, alacsony fogyasztással rendelkezik, egy tizedesjegy pontossággal mér, van grafikus felülete. Látható az 1. ábrán, hogy egy kültéri és egy beltéri szenzor méri az adatokat: hőmérsékletet, relatív páratartalmat és légnyomást. Másik népszerű funkció ezeken az egységeken a dátum és idő kijelzése, ami szintén megjelenik ezen a készüléken is.

A háztartásokban használt állomások többnyire a következő időjárási paramétereket mérik:

1. Hőmérséklet
2. Relatív páratartalom
3. Környezeti fényerősség
4. Légköri nyomás

A hőmérsékletet elegendő 1-2 tizedesjegy pontossággal mérni, mert az emberi szervezet az egy tized Celsius fokos különbséget sem érzékeli jelentős változásnak. A relatív páratartalom százalékos mennyiség, ezért azt egész számban szokták ábrázolni. A környezeti fényerősséget is szokás egész számként tárolni, hiszen a fő célja csak azt meghatározni, hogy világos vagy sötét van. A légköri nyomást is egész számként kezeljük, mert ezres nagyságrendben szokott lenni átlagos városi körülmények között és az egy-két Pascal különbséget nem tudja érzékelni a szervezet.

1.1.3 Tudományos célok

Az utóbbi időben kiemelt téma a sajtóban és a tudományos körökben egyaránt a jelenlegi éghajlatváltozás és a globális felmelegedés. Az olcsó, tömegek számára könnyedén elérhető otthoni időjárás állomások nagy segítséget nyújthatnak a

tudományos kutatásoknak is. Rengeteg időjárási adathoz juthatnak az ilyen állomások adatai alapján, amik alapjául szolgálhatnak kutatásoknak, ahol az éghajlati paramétereket, változásokat tudják elemezni ezen adathalmaz alapján.

A tudományos célokra is megfelelő az otthoni időjárás állomások mérési felbontása, hiszen az egy-két tizedesjegynyi változás elegendő az éghajlat jellemzésére. Például a globális felmelegedés leírására is egy tizedesjegy felbontással beszélnek a hőmérséklet növekedésről.

1.2 Saját időjárás állomás

A szakdolgozat keretein belül tervezett időjárás állomás célja egy olyan alacsony költségű, alacsony fogyasztású, kisméretű, megfelelő mérési felbontású összetett rendszer megtervezése és prototípusának elkészítése, ami sokak számára könnyedén elérhető módon alkalmas az otthoni időjárási adatok gyűjtésére, tárolására és megjelenítésére. Fontos szempont, hogy a dolgozatom fő célja egy teljes rendszer megtervezése, a szükséges folyamatok és technológiák megismerése, megértése és kipróbálása. Emiatt gyakran olyan eszközöket választottam, amik már a rendelkezésemre állnak és így a fejlesztési költséget és időt is csökkenteni tudtam.

A saját állomásomnak teljesíteni kell ugyanazokat a követelményeket, amiket a piacon elérhető termékek is, valamint személyre szabható, moduláris felépítéssel egy könnyedén konfigurálható rendszert valósítok meg.

A mérendő időjárási paraméterek:

- Hőmérséklet
- Relatív páratartalom
- Környezeti fényerősség
- Légköri nyomás

A rendszer része egy érintésérzékeny kijelző is, ami két funkciót lát el: grafikusán megjeleníti az adatokat, valamint felhasználói kezelőfelületként különböző menüpontokkal vezérelhető a készülék.

A mérési eredményeket hosszútávon szeretném tárolni, ezért ezeket egy adatbázisban helyezem el. Az eltárolt adatok lehetővé teszik, hogy tudományos kutatáshoz vagy statisztika készítéséhez fel lehessen használni a korábbi méréseket.

A rendszer moduláris felépítése miatt elkülönülnek fizikailag is a különböző feladatokat ellátó részek. A méréseket akkumulátorról üzemelő vezeték nélküli mérőegységet végzik. Egy központi egység irányítja a mérőegységeket, valamint tárolja és továbbítja az adatokat az adatbázishoz. Az adatbázis egy távoli szerveren egy felhős környezetben fut, aminek a fizikai helye nem is ismert.

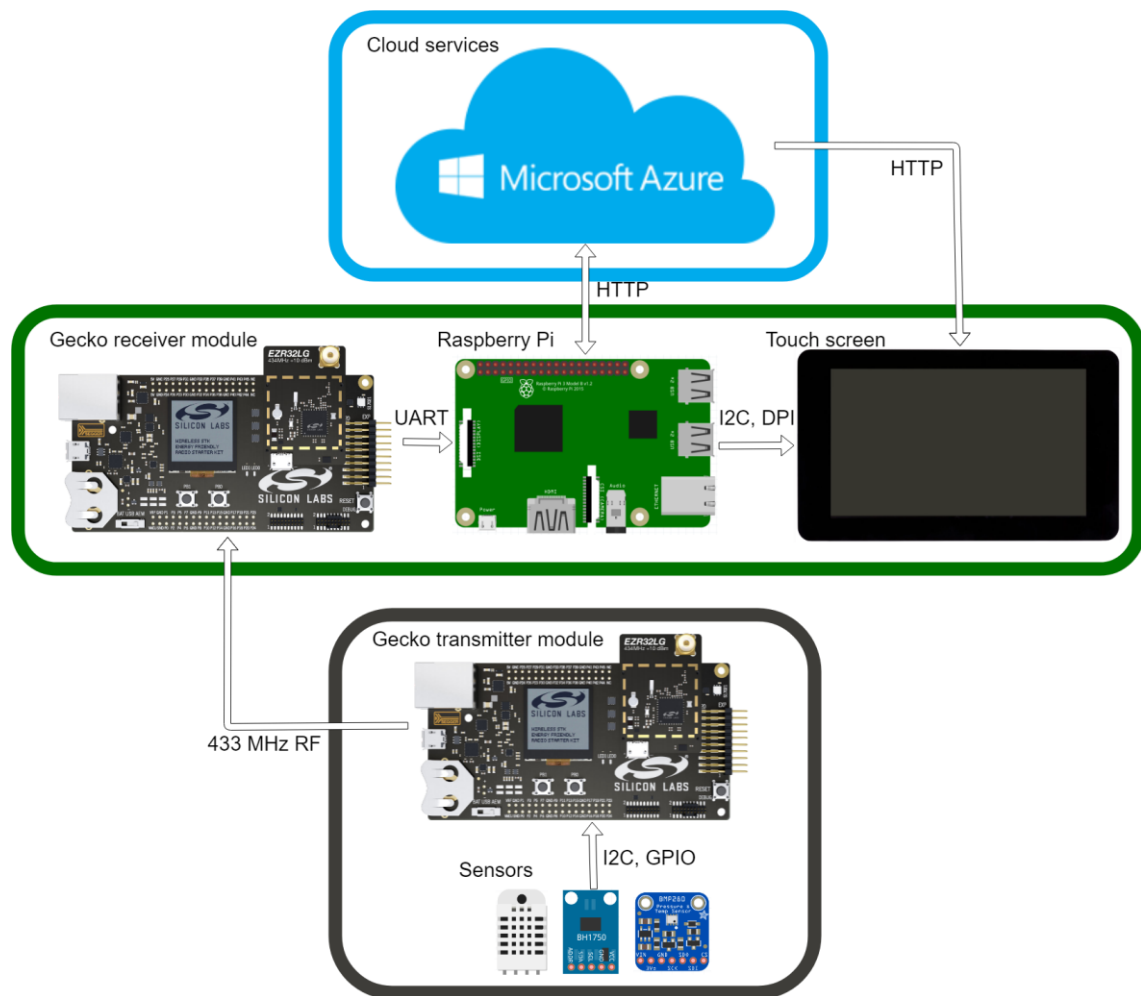
2 Tervezés

A szakdolgozatom időjárás állomását a rendszerterv elkészítésével kezdtem. A rendszer három fő része:

- Felhős szolgáltatások: a webes szerver- és kliensalkalmazás, valamint az adatbázis tartozik ide
- Központi egység: az adatok begyűjtése, feldolgozása és továbbítása a feladata
- Mérőegység: a környezeti paraméterek mérése és ezen adatok tovább küldését végzi

A központi egység és a mérőegység fizikailag közel helyezkednek el egymáshoz. A központi egység minden esetben beltéri helyen található. A mérőegység lehet bel- vagy kültéri helyszínen is. A felhőben futó részek pedig fizikailag teljesen máshol találhatóak egy adatközpontban valahol a Microsoft cég szerverparkjában. A prototípus fejlesztése során a központi és a mérőegység Budapesten volt, a felhős alkalmazás pedig valahol Nyugat-Európában, ennél pontosabban nem is lehetett lekérdezni a részleteket.

A szakdolgozatomként megvalósítandó állomás rendszerterve a következő:



2-1. ábra: Rendszerterv

A felhős szolgáltatások a Microsoft Azure segítségével érhetők el. Itt található egy webszerver, ami fogadja és kiszolgálja a kliens oldali kéréseket, kommunikál az adatbázissal és generálja a webes kliens alkalmazást, amit a felhasználó webböngészőjének küld el.

A központi egység tartalmazza a rádiós adóvevő modult, a Raspberry Pi-t és az érintésérzékeny kijelzőt. Az adóvevő a mérőegységtől érkező adatokat fogadja a 433 MHz-es ISM frekvenciasávot használó rádiófrekvenciás protokollon keresztül. A Raspberry Pi a rádióvevőtől érkező adatokat feldolgozza, ideiglenes eltárolja, majd továbbítja őket a felhőben található adatbázisba. A kijelző valósítja meg a felhasználói felületet, ahol az időjárási adatok megjelennek, valamint néhány menüponton keresztül vezérelhető az állomás.

A mérőegységben található egy mikrokontroller és három szenzor, ami méri az időjárási paramétereket. Két fő feladatot valósít meg ez az egység: a mérést és az adatok küldését a központi egység felé.

A központi egység mivel minden esetben beltéri helyiségben kerül elhelyezésre, ezért a vezetékes internet és hálózati tápellátás megkönnyíti a tervezést. A mérőegységeket beltéri és kültéri üzemre is tervezem, ezért ezeknél fontos a vezeték nélküli kommunikáció és a vezeték nélküli tápellátás is.

2.1 Feldolgozó egységek

A mérőegységek alapját jelentő feldolgozó egységnek mikrokontrollert választottam az alacsony fogyasztás, könnyű programozási lehetőségek és a szenzorokhoz való illesztés egyszerűsége miatt. Erre a célja a Silicon Labs cég által fejlesztett SLWSTK6200A fejlesztőkészletet választottam, mert minden követelményt teljesít és rendelkezem egy ilyen készlettel. Ebben található egy BRD4001 jelű Mainboard-nak nevezett bővítőkártya, ahol számos periféria ki van vezetve a könnyű prototípus készítés miatt. A készlet fő alkotóeleme a BRD4502A EZR32 Leopard Gecko 868 MHz WSTK Radio Board, ami a mikrokontroller, egy integrált rádiós adóvevő modul és egy csatlakozó a Mainboard-on található foglalathoz. A csomagban vannak antennák is a jelerősség növelésére, amik SMA csatlakozóval csatlakoztathatók a Radio Board-okhoz. A gyors prototípus fejlesztéshez nagy előnyt jelent, hogy USB Micro-B kábellel személyi számítógépről programozható a mikrokontroller, valamint a tápfeszültséget is az USB kábelén keresztül kapja.



1.2-2 ábra: WSTK6200 fejlesztőkészlet EZR32 mikrokontrollerrel

A központi egységben kettő feldolgozóegység is található:

- Egy Raspberry Pi 2 B+ miniszámítógép
- Egy a mérőegységben használttal teljesen megegyező WSTK6200 fejlesztőkészlet

Mindkét modul rendelkezésemre áll, ezért rendelési és kiszállítási idő nélkül tudtam elkezdni a fejlesztést. A miniszámítógép a hálózati aljzatból egy adatperen keresztül kapja a tápellátást, valamint Ethernet kábelén keresztül éri el az internetet. A központi egységben található Gecko az adatok fogadását irányítja. Mivel két teljesen azonos Gecko végzi a rádiós kommunikációt, ezért a rádiós fejlesztés nagyon kényelmes és gyors.

2.2 Vezeték nélküli kommunikáció

A fent felsorolt technológiák közül egy rádiófrekvenciás protokollt választottam, amit a Silicon Labs szállított a WSTK készlettel együtt.

Ennek okai:

- A Wi-Fi hálózat gyakran túlságosan leterhelt és felhasználja a rendelkezésre álló sávszélességet a rengeteg szórakoztató elektronikai eszköz miatt, például a mi háztartásunkban van 3 laptop, 4 okostelefon, 2 okos televízió, ami használja a Wi-Fi hálózatunkat.

- A ZigBee modulok túlságosan drágák egy egyszerűbb időjárás állomás költségeihez mérten ⁷.
- Az NFC technológia néhány centiméteres hatótávolsága nem elegendő a háztartáson belüli távolságok lefedésére.
- A Bluetooth Mesh hálózat még nagyon korai fázisban van és nem elég elterjedt a gyors prototípus fejlesztéshez, például kevés fejlesztőmodul és függvénykönyvtár, internetes segédanyag, leírás található hozzá.

A választott rádiófrekvenciás technológia lehetővé teszi a következő lehetőségeket:

- Háztartási távolságok (5-10 méter) lefedése
- Több eszköz közötti kommunikáció
- Alacsony fogyasztású rendszer tervezése
- Engedélyeztetés nélküli frekvenciatartomány használata

A WSTK6200 készletben található integrált Sub-GHz adóvevő egy EZRadioPRO modul ⁸. Ezek nagyteljesítményű, alacsony fogyasztású rádiós adóvevő áramkörök a 119-1050 MHz-es tartományban üzemelve. A rádiós protokollt a Silicon Labs implementálta, a fejlesztést megkönnyítve egy magasabb szintű függvénykönyvtár és grafikus konfigurációs alkalmazás segítségével lehet használni ezeket az egységeket.

A kiválasztott rádiós technológia néhány jellemzője:

- Használható frekvenciatartomány: 119-1050 MHz
- Adóteljesítmény: 13 dBm
- Egyetlen antenna csatlakozó a küldésre és a fogadáshoz egyaránt
- Érzékenység: -133 dBm
- IEEE 802.15.4g szabvány szerint működik

⁷ https://www.hestore.hu/prod_10030987.html

⁸ <https://www.silabs.com/products/wireless/proprietary/ezradiopro-ism-band-transmitters-recievers-transceivers>

- Többféle moduláció közül lehet választani: (G)FSK, (G)MSK, 2(G)FSK, (G)FSK, OOK
- Alacsony fogyasztás:
 - 10-13 mA adatfogadás esetén
 - 18 mA adatküldésnél (+10 dBm mellett)
 - 30-40 nA készletű fogyasztás
- Adatátviteli sebesség: 0.1 kbps – 1 Mbps
- Csomag alapú kommunikáció, ez nagyon részletesen konfigurálható

2.3 Szenzorok

A mikrokontrolleres mérőegységhez választandó szenzorokkal szemben támasztott követelmények:

- Minél könnyebben illeszteni lehessen őket a kiválasztott mikrokontrollerhez
- Alacsony fogyasztásúak legyenek az akkumulátoros működés miatt
- Alacsony költségűek legyenek az egész egység árának alacsonyan tartásához
- Megfelelő periodicitással olvasni lehessen belőlük friss adatokat
- Elég pontos felbontással tudják mérni és ábrázolni a mérendő környezeti paramétereket
- Könnyedén beszerezhetők legyenek megbízható helyekről

Olyan szenzorokat választottam, amik a fenti követelményeknek megfelelnek:




- A mikrokontroller 3,3 vagy 5 V-os kimeneti feszültségei elegendő tápfeszültség nekik
- I2C és GPIO perifériákon keresztül vezérelhetők
- A mérési időn kívül alvó vagy alacsony fogyasztású állapotba állíthatók
- Néhány száz vagy ezer forintos áron beszerezhetők

- Milliszekundum és szekundum nagyságrendben lehet olvasni belőlük adatokat
- Legalább 2-3 tizedesjegy pontossággal mérik és ábrázolják az adatokat
- Magyarországi webáruházakból megrendelhetők és néhány nap alatt kiszállítják őket

A mérőegységhez 3 különböző szenzort választottam:

1. DHT22-t a hőmérséklet és páratartalom mérésére
2. BMP280-t hőmérséklet és légnyomás mérésére
3. BH1750FVI-t környezeti fényerősség mérésére

A szenzorok főbb tulajdonságait összefoglaló táblázat:

Tulajdonságok	DHT22	BMP280	BH1750FVI
Mért paraméterek	Hőmérséklet, páratartalom	Hőmérséklet, légnyomás	Környezeti fényerősség
Kinézet			
Interfész	GPIO	I2C	I2C
Tápfeszültség	5 V	3,3 V	5 V
Fogyasztás mérés alatt	1,5 mA	<7,02 μ A	120 μ A
Fogyasztás alacsony fogyasztású állapotban	0,8 mA	<2,74 μ A	~0,01 μ A

Látható, hogy a DHT22 fogyasztása sokkal magasabb a másik kettő eszközhöz képest, de még mindig elég alacsony, hogy az összfogyasztás ne legyen túl magas és a GPIO interfésze révén a vezérelhetőség egyszerűsége is fontos szempont, hogy kiválasztásra került.

A szenzorok összfogyasztása az adatlapi értékekkel számolva:

- Mérés alatt körülbelül 1,627 mA
- Alacsony fogyasztású állapotban: 802,75 μ A

Látható a fenti adatok alapján, hogy az alvó vagy alacsony fogyasztású állapotban történő várakozással több, mint felére csökkenthető a szenzorok fogyasztása.

2.3.1 DHT22

A DHT22 egy kínai gyártású, egyszerű és alacsony költségű digitális hőmérséklet és páratartalom mérő szenzor. Adatlapja ⁹ és felhasználási útmutatók is elérhetők a népszerű angol elektronikai portálon az Adafruiton ¹⁰.

Egy kapacitív páratartalom mérőt és egy termisztort használ a levegő paramétereinek mérésére. A mért paramétereket egyetlen digitális lábán lehet leolvasni. Nem igényel analóg jelfeldolgozást, valamint egyetlen digitális láb is elég a használatához, amivel könnyebb a kezelése, mint egy magasabb szintű interfészt igénylő szenzor esetén. Az adatok kiolvasásához precíz időzítés szükséges, de ezt a mai mikrokontrollerek segítségével könnyedén meg lehet oldani.

Egyetlen hátránya, hogy legalább 2 másodpercet várni kell két mérés elvégzése között, viszont ez az időjárás állomás mérési gyakoriságánál bőven kevesebb, ezért nem okoz problémát.

A tápellátása egy 5 V-ot feszültséget kiadó láb és egy GND láb által könnyedén megoldható.

A szenzor néhány tulajdonsága:

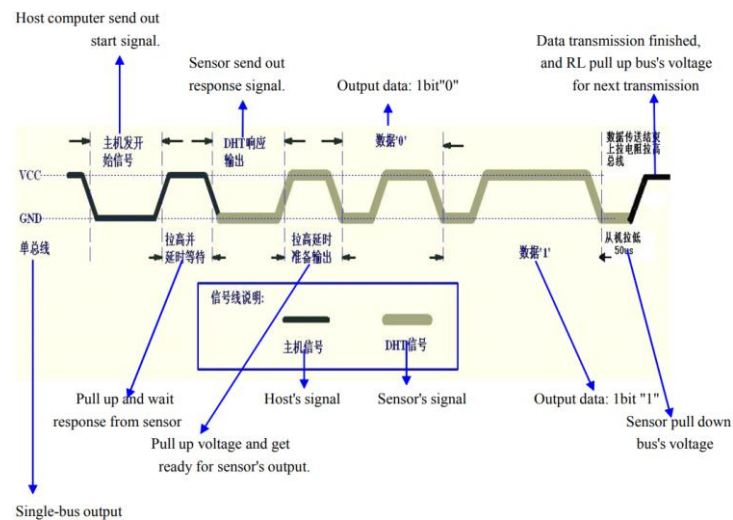
- Páratartalom mérési tartománya: 0-100% relatív páratartalom
- Hőmérséklet mérési tartománya: -40 és 125 °C között
- Pontosság: $\pm 2\%$ relatív páratartalom esetén és $\pm 0,2$ °C a hőmérséklet esetén
- Felbontás: 0,1% relatív páratartalom esetén és 0,1 °C a hőmérséklet esetén

⁹ <http://www.adafruit.com/datasheets/DHT22.pdf>

¹⁰ <https://learn.adafruit.com/dht>

- Hosszútávú stabilitás: $\pm 0,5\%$ relatív páratartalom / év

Az szenzor által mért adatok kiolvasása:



2-3 ábra: DHT22 kiolvasása

A fenti ábra a szenzor részben angol, részben kínai nyelvű adatlapjáról származik ¹¹. Az adatok olvasása:

1. Mikrokontroller mérési utasítást küld a szenzornak egy logikai alacsony jelszint előállításával
2. A szenzor szigorú időzítéssel válaszul adja a mérési adatokat bitenként logikai alacsony és magas jelszintek előállításával
3. A kommunikáció végeztével ismét logikai magas szintbe kerül az adat lába a szenzornak

2.3.2 BMP280

A BMP280 szenzor egy a német Bosch Sensortec vállalat által fejlesztett abszolút légköri nyomást és hőmérsékletet mérő szenzor, amit kifejezetten beágyazott rendszerek számára terveztek. A szenzor tartalmaz egy Piezo-rezisztív ¹² nyomás mérő elemet és egy vegyes jelű ASIC-ot. Az ASIC végzi az analóg jelek digitálissá alakítását, valamint a konverzió végeredményét és a szenzor specifikus kompenzációs paramétereket elérhetővé teszi egy digitális interfészen keresztül.

¹¹ <https://cdn-shop.adafruit.com/datasheets/DHT22.pdf>

¹² https://en.wikipedia.org/wiki/Piezoresistive_effect

Főbb jellemzői az adatlapja alapján ¹³:

- Légköri nyomás mérési tartománya: 300...1100 hPa (-500...+9000 méter tengerszint feletti magasság)
- Relatív pontosság: $\pm 0,12$ hPa (± 1 méter) 950-1050 hPa nyomástartományon 25 °C hőmérsékleten
- Hőmérsékleti együttható eltoltása: 1,5 Pa/Kelvin (12,6 cm/Kelvin)
- Digitális interfészek: I2C, SPI
- Áramfelvétel: átlagosan 2,7 μ A 1 Hz-es mintavételi frekvencia mellett
- Üzemi hőmérséklet tartomány: -40...+85 °C

Háromféle üzemmódja van: alvó mód, normál mód és erőltetett mód. Ezek között a különbség a mérési periodicitás és az áramfelvétel. Van lehetőség túlmintavételezésre, ami nagyban befolyásolja a fogyasztást és a felbontást. A szenzorban van egy integrált IIR szűrő ¹⁴ is, ami a nagyfrekvenciás zajokat próbálja csökkenteni – például ajtó- vagy ablaknyitás vagy csukás. Elérhető többféle előre definiált üzemmód és szűrő beállítás is a fejlesztés támogatására. Az időjárási állomásomhoz leginkább alkalmas üzemmód és beállítás az adatlapban található *Weather monitoring (setting with lowest power consumption)*.

¹³ https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-19.pdf

¹⁴ https://en.wikipedia.org/wiki/Infinite_impulse_response

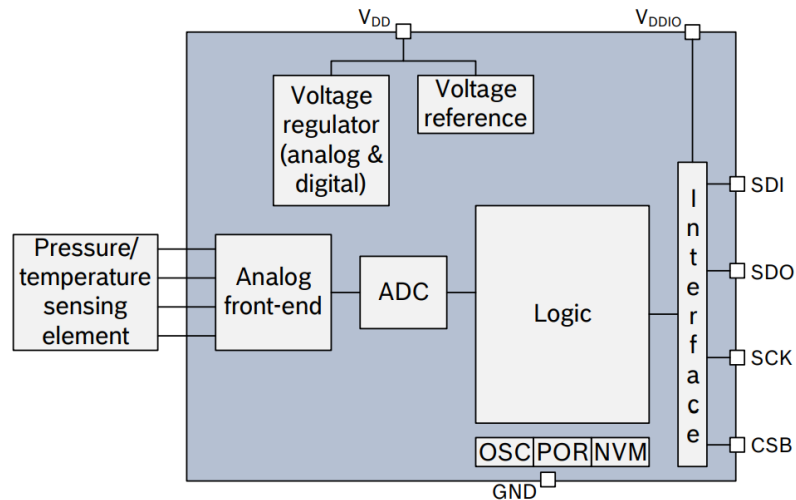


Figure 1: Block diagram of BMP280

2-4 ábra: BMP280 blokkvázlata

A blokkvázlaton látható a nyomás és hőmérséklet mérő elem, a mixed ASIC, ami digitális jellé alakítja a mérési adatokat, a túlmintavételezést és szűrést megvalósító logika, valamint az I2C és SPI kommunikációt lehetővé tevő digitális interfész.

A mérési folyamat a következő:

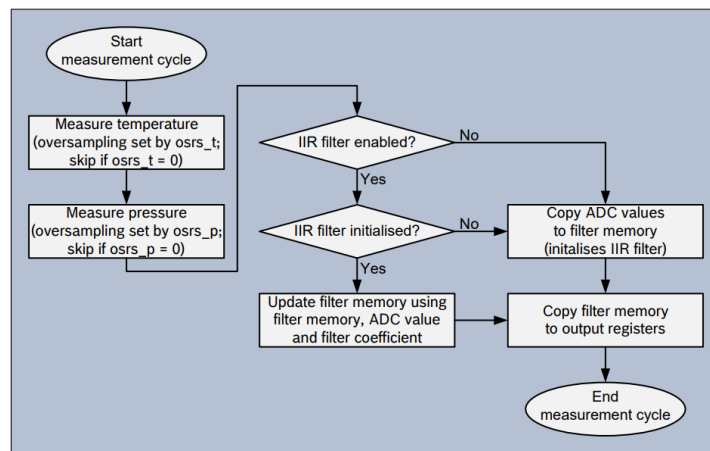


Figure 2: BMP280 measurement cycle

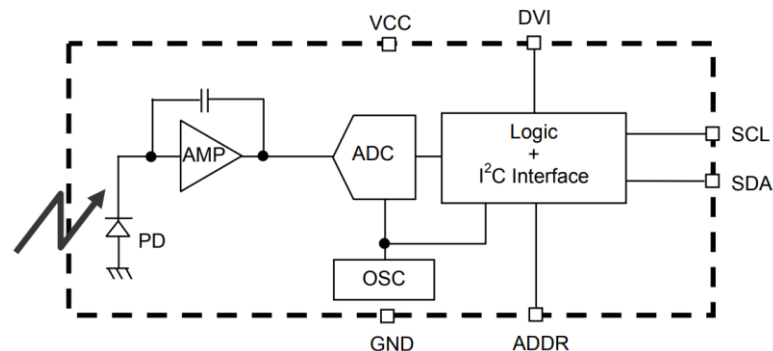
2-5 ábra: BMP280 mérési folyamatára

A mérést indító utasítást követően a mérő elem megméri a hőmérsékletet és a légköri nyomást. Az IIR szűrő engedélyezésétől függően elvégzi a nagyfrekvenciás jelek szűrését, ha be van állítva. A mérés végeztével – szűrés után, ha engedélyezve van – a szenzor regisztereiben elérhetőek a mérési adatok.

Az I2C kommunikációval sokkal több tapasztalatom van, mint SPI-jal, ezért az I2C interfészt használom a szenzornál.

2.3.3 BH1750FVI

A BH1750FVI egy digitális környezeti fénymérő szenzor, ami I2C interfésszel rendelkezik ¹⁵. A mért tartomány: 1...65535 lx. Ez bőven lefedi az emberi szám által érzékelt fénytartományt. Van alacsony fogyasztású üzemmódja. Nem igényel külső alkatrészeket a szenzor. A tápellátást 3,3 V-os egyenfeszültségről a mikrokontroller adja. A mérési idő 16-120 ms között változik a felbontástól függően.



2-6 ábra: BH1750 blokkvázlata

A blokkvázlaton látszik, hogy a fényt mérő elem kimenetét felerősíti egy műveleti erősítő, ami egy ADC segítségével digitális jellé alakul és ezt egy logikai egység I2C interfészen keresztül kiolvashatóvá alakítja. A fényt egy fotódióda méri. Az ADC a dióda áramát feszültséggé alakítja. A logikai egység a feszültségszintből digitális adatot állít elő, amit egy regiszterben tárol el.

Többféle mérési üzemmód is elérhető. Az időjárás állomás paramétereinek alapján én a *One-Time H-Resolution Mode2*-t választottam, mert ritkán mérek, de olyankor legyen nagyfelbontású a mért fényerősség. Ekkora 0,5 lx a felbontás, a mérési idő körülbelül 120 ms és automatikusan energiatakarékos üzemmódba kerül a szenzor a mérés végeztével.

2.4 Felhős szolgáltatások

Négy fő felhős szolgáltatásokat nyújtó technológia vállalat van, akik a számomra szükséges technológiákat kínálnak:

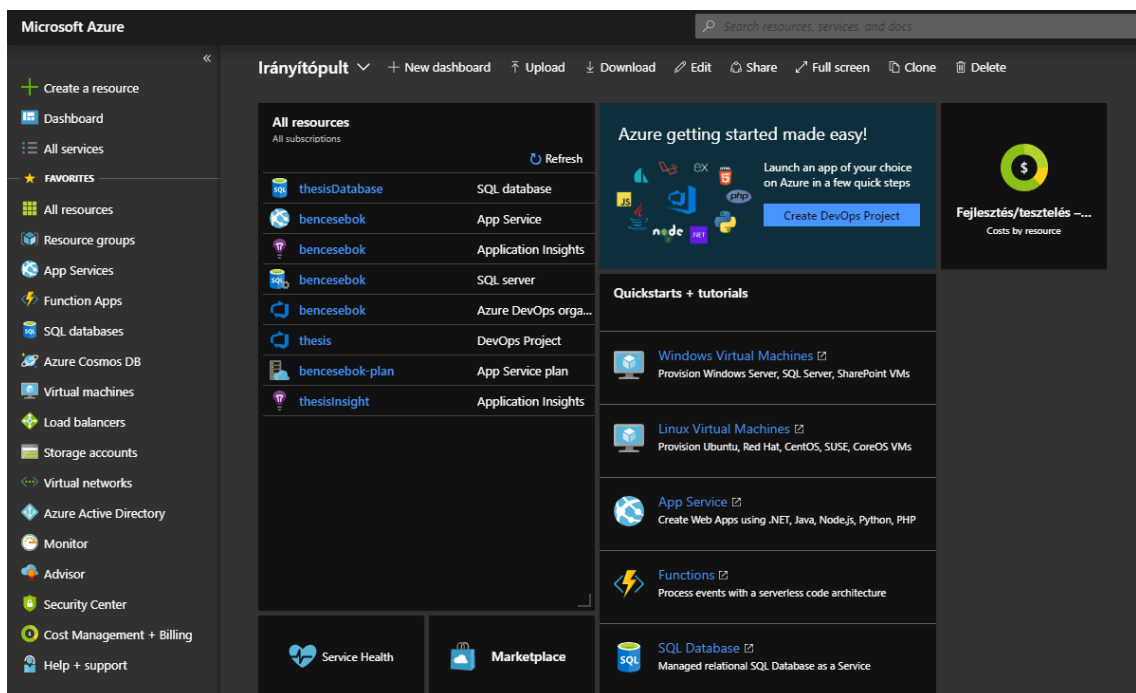
1. Amazon Web Services

¹⁵ <https://www.mouser.com/ds/2/348/bh1750fvi-e-186247.pdf>

2. Microsoft Azure
3. Google Cloud
4. IBM Cloud

Ezek közül a Microsoft Azure-t választottam a következő okok miatt:

- Egyetemi licensszel 100 dollárnyi kreditet használhatok fel ingyenesen
- Grafikus felülettel is rendelkezik, ahol a legtöbb parancssori eszközt egy szép és könnyen kezelhető felhasználói felülettel elérhetek
- Node.js webszerver futtatható rajta – magabiztos tudással és tapasztalattal rendelkezem ezzel kapcsolatban, így gyorsan tudok prototípust fejleszteni vele
- SQL alapú adatbázis készíthető vele, amivel szintén tapasztalatokkal rendelkezem.



2-7 ábra: A Dashboard az Azure Portalon

Az Azure-ban futó erőforrások, amiket használok:

- SQL database: ez egy általános célú, relációs adatbázis szolgáltatás az Azure-ban, ami támogat többféle struktúrát például relációs adat, JSON, XML stb. Elérhetők hozzá analitikai eszközök és különböző jelentések is

a felhasznált adatmennyiségről például. Ezt a szolgáltatást használom az adatok hosszútávú tárolására.

- App service: egy felhős alkalmazásokat támogató platform, amivel könnyedén lehet programokat fordítani, publikálni, skálázni a terheléssel arányosan. A publikáció GitHub-bal integrált CI/CD ¹⁶ lehetőségekkel van elősegítve. Ezt a szolgáltatást használom a Node.js webszerverem futtatására.
- DevOps project: egy fejlesztői környezetet támogató eszköz, ami segít a népszerű alkalmazás keretrendszerek (az én esetemben ez a Node.js) Azure-ban való létrehozására és fejlesztésére. Automatizált fordítás és publikálás gyorsítja a valódi rendszerek elérhetővé tételét. Analitikai és elemző eszközök mutatják a használat részleteit. Ezzel az eszközzel hoztam létre az Azure-ban található minden felhős szolgáltatásomat az adatbázis kivételével.

2.5 Felhasználói kezelőfelület

A felhasználói kezelőfelület az egyik legfontosabb része egy háztartásokba szánt eszköznek. A technikai oldal általában teljesen el van rejtve az átlagfelhasználók elől, viszont a technológia által elérhető adatok elegáns és átlátható megjelenítése sokat jelent a készülék kiválasztása és használata mellett.

A felhasználói kezelőfelület főbb funkciói:

- Adatok megjelenítése
- Menüpontok közötti választás lehetősége
- Menüpontok alapján különböző funkciók elérése

A kezelőfelület tervezése során fontos szempontok:

- Integrálhatóság a jelenlegi rendszerbe
- Látványos grafikus felület megvalósítása
- Bőséges eszköztár a részletes testreszabásához

¹⁶ <https://en.wikipedia.org/wiki/CI/CD>

- Egyszerű, könnyű kezelhetőség – ne igényeljen tanulást
- Letisztultság, csak a legfontosabb funkciók legyenek elérhetők

A felhasználói kezelőfelület két fő részből áll:

1. Érintésérzékeny kijelző
2. Webes kliens alkalmazás

Érintésérzékeny kijelzőnek egy Raspberry Pi 7” Touch Display-t választottam¹⁷. Manapság rengeteg eszköz – például okostelefonok, tabletek, bolti hirdetőpanelek stb – vezérelhető érintéssel, gesztusokkal, ezért én is egy ilyen megoldást választottam, valamint ez az egység is rendelkezésemre áll, így fejlesztési időt és költséget is spóroltam. A kijelző felbontása 800 x 480 pixel, amivel elég részletes grafikai elemek ábrázolhatók. Kétféle módon kell csatlakoztatni a Raspberry-hez: tápellátást az 5 V-os GPIO pinek egyikéről kap a kijelző, valamint a jelátvitelt megvalósító DSI¹⁸ porton keresztül. A kijelző driver-ei lehetővé teszik a 10 ujjas kezelést, valamint az operációs rendszertől függően képernyő-billentyűzet segítségével egyéb beviteli eszközre nincs is szükség.



2-8 ábra: DSI port használata a kijelző és a miniszámítógép között

¹⁷ <https://www.raspberrypi.org/products/raspberry-pi-touch-display/>

¹⁸ <https://www.raspberrypi.org/blog/the-eagerly-awaited-raspberry-pi-display/>

A webes kliens alkalmazás dinamikusan tud adatokat megjeleníteni az Azure-ban futó Node.js webservert segítségével. A webserveren található egy /hmi végpont, amire HTTP GET kérést küldve a szerver lekérdezi az adatbázisból a legfrissebb időjárási adatokat és ezek alapján generál egy weboldalt, ami tartalmazza a kezelőfelülethez tartozó menüpontokat és a mérési adatokat egyaránt. Az elkészült weboldal visszaküldi a kliens böngészőnek, ami ezt megjeleníti a kijelzőn. A különböző menüpontok további végpontokra irányítanak át, amik szintén a webservice által értelmezésre kerülnek és a menüponthoz tartozó funkciót elvégzi a szerver és visszaadja a megfelelő kliens oldali megjelenítést a kijelzőt kezelő miniszámítógépnek.

3 Implementáció

3.1 Mérőegység

A mérőegységet kültéri és beltéri használatra tervezem, így fontos, hogy egy könnyedén mozdítható kompakt modul legyen. Ennek fontos követelménye, hogy teljes mértékben vezetékek nélkül működjön a lehető legtöbb ideig. Mind a kommunikációt, mind a tápellátást vezeték nélkül kell megoldani: a rádiófrekvenciás adatküldés és egy könnyedén cserélhető akkumulátor megoldást jelent erre a követelményre.

A mérőegység alapját jelentő mikrokontrollert gyártó cég rendelkezik saját fejlesztőkörnyezettel, ami a Silicon Labs Simplicity Studio ¹⁹ néven érhető el.



3-1 ábra: Silicon Labs Simplicity Studio

Ez egy platformfüggetlen integrált fejlesztőkörnyezet, ami többféle eszközzel is segíti a fejlesztést:

- Kódszerkesztés kódkiegészítés funkcióval
- C forráskódok fordítása
- Szükséges függvénykönyvtárak linkelése

¹⁹ <https://www.silabs.com/products/development-tools/software/simplicity-studio>

- Grafikus debug eszköztár
- Szoftver példák
- Dokumentációk és felhasználói útmutatók
- Grafikus konfigurációs alkalmazások – például rádiós protokoll és hardver konfigurátorok

Ebben az IDE-ben készítettem egy projektet a mikrokontrollereken futó alkalmazás fejlesztéséhez. A rádiós protokoll beállításait a grafikus Radio Configurator segédalkalmazás segítségével végeztem el.

3-2 ábra: Radio Configurator használata a Simplicity Studio-ban

A C programozásnál figyelembe vett alapvető tervezési elvek:

- **Modularitás:** minden különálló funkciót külön függvénybe szerveztem, minden szenzorhoz tartozik egy .c forrásfájl és egy .h header fájl
- **Olvashatóság:** részletes kommentekkel magyarázom folyamatosan a kódot, bekezdéseket alkalmazok a kód tagolására
- **Újrahasználhatóság:** olyan hardverfüggetlen rétegben írtam meg egyes API-kat, hogy azt mikrokontroller típustól függetlenül lehessen használni a megfelelő alsóbb rétegek használatával

A fejlesztés folyamatát verziókezeléssel támogattam, hogy korábbi állapotokat vissza lehessen állítani hibás működés esetén, nyomonkövethető legyen a fejlesztés folyamata, az egyes fő funkciók branch-ekben való szétválasztása.

3.1.1 Tápellátás

A tápellátást a modulhoz illesztve kell megoldani, ezért nem választhatok vezetékes megoldást – például a hálózati aljzatot. Könnyű megoldás egy olyan akkumulátor kiválasztása, ami rendelkezik a megfelelő interfésszel a mikrokontrollerrel való összekötéshez, a kimeneti feszültsége a mérőegység bemeneti feszültségével kompatibilis, képes elegendő áramerősség leadására, valamint legyen könnyedén cserélhető vagy tölthető.

A mikrokontrollerrel való használhatóság kapcsán az akkumulátorral szemben támasztott követelmények:

- Kimeneti feszültsége legyen 5 V
- Rendelkezzen USB csatlakozóval
- Legalább néhány száz mA áramot tudjon leadni

Ezekkel a tulajdonságokkal mind rendelkeznek a manapság nagyon elterjedt, elsősorban okostelefonokhoz használt vésztöltők – angol nevükön power bankek. Nekem is van egy ilyen eszközöm, aminek a pontos típusa: Huawei AP08Q 10000mAh vésztöltő ²⁰. Egy USB Micro-B csatlakozón keresztül 5 V-os feszültség mellett 2 A áram leadására is képes, ami abszolút alkalmassá teszi a mérőegység táplálására. Továbbá könnyedén cserélhető és tölthető is, szóval lemerülés esetén gyorsan be lehet tenni másik hasonló vagy ugyanilyen akkumulátor egységet a helyére.

3.1.2 Kommunikáció

A mérőegység kétféle fő magas szintű kommunikációt használ:

1. I2C kommunikáció a mikrokontroller és kétféle szenzor között
2. 433 MHz-es ISM sávban üzemelő rádiófrekvenciás protokoll használata a mérőegység adója és a központi egység vevője között

²⁰ <https://consumer.huawei.com/en/accessories/10000-qc-powerbank>

3.1.2.1 I2C

A Silicon Labs gyártmányú mikrokontrollerekhez elérhető egy `em_i2c`²¹ nevű függvénykönyvtár, ami a legalacsonyabb, bitszintű implementációt valósítja meg. Ez tartalmazza az integrált I2C modul inicializálásához szükséges konstansokat, regiszter címeket, tipikus regiszter értékeket, valamint a használatához szükséges függvényeket.

Erre a rétegre kell implementálni a szenzor specifikus I2C függvényeket. Ezekhez szükségesek az alábbiak:

- I2C-s szenzorok slave címei
- A szenzorok belső regisztereinek kezdőcímei
- A szenzorok tipikus regiszter értékei – például gyakori konfigurációkhoz tartozó konstansok
- A működéshez szükséges írási és olvasási műveletek részletei – például hány bájtot kell olvasni vagy írni az egyes szenzor specifikus műveleteknél

Ezek alapján egy magasabb szintű I2C API sablonja a következő:

- Melyik regiszter kezdőcímtől kezdünk el írni vagy olvasni
- Egy adott művelet az olvasás vagy írás
- Hány bájtot akarunk írni vagy olvasni
- A válaszként kapott bájtokkal mit csináljunk

Egy példa erre:

```
I2C_TransferReturn_TypeDef BMP280_I2C_ReadRegister(uint8_t  
registerAddress, int8_t * bufferRead, uint16_t readSize)
```

Ez a függvény egy I2C olvasási műveletet valósít meg, pontosabban `I2C_FLAG_WRITE_READ` szekvenciát, ami kombinált írás és olvasás. Ennek részletes leírása a következő táblázatban olvasható:

S	ADDR(W)	DATA0	Sr	ADDR(R)	DATA1	P
START bit	Cím, ahova írunk	Adat, amit írunk az előző címre	Ismételt START	Cím, ahonnan olvasunk	Adat, amit olvasunk az előző	STOP bit

²¹ https://siliconlabs.github.io/Gecko_SDK_Doc/ezr32lg/html/group__I2C.html

					címről	
--	--	--	--	--	--------	--

A slave egység címe a függvényen belül van megadva, illetve mivel 7 bites címzést használok, ezért a 8. bitként az olvasás van megjelölve. A registerAddress címtől kezdődően fog olvasni, readSize darabnyi bájtot a bufferRead címtől kezdődő bufferbe. Blokkoló olvasást végzek, vagyis addig nem tér vissza a függvény, amíg be nem fejeződik az olvasás. Ennek oka, hogy általában legfeljebb 6 bájtot olvasunk, ami alig néhány ms alatt lefut. Ehhez hasonlóan implementáltam a többi függvényt ehhez a szenzorhoz és a másik szenzor összes I2C-s függvényét egyaránt.

3.1.2.2 Rádiófrekvencia

A mikrokontrollerhez adott grafikus Radio Configurator alkalmazást használtam a rádiós protokoll paramétereinek beállítására. Ezt a radio-configurator_SLWSTK6200A.isc fájlban tárolja a Simplicity Studio-s projekt és bármikor módosítható, ami egy fordítási idő előtt történő fájl generálással frissíthető.

A legfontosabb paraméterek:

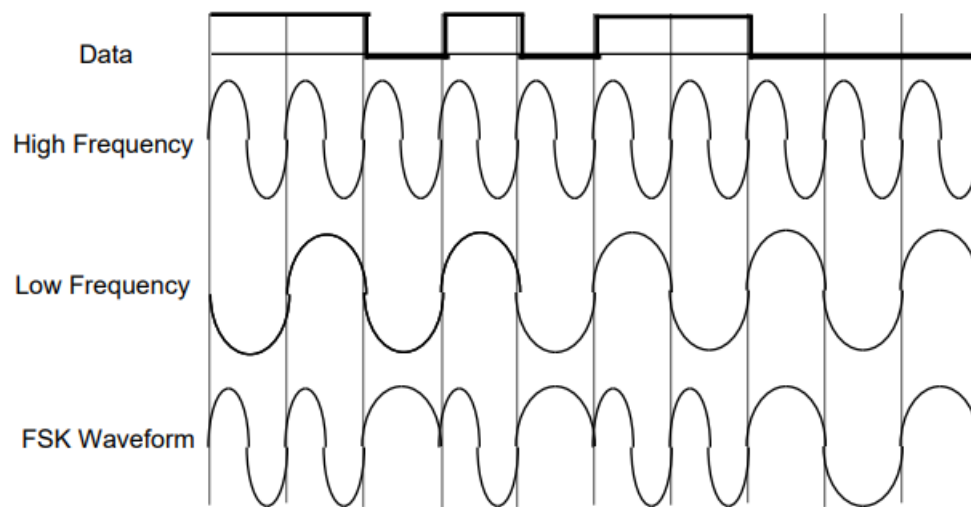
- Base frequency: 433 MHz, mert Európában ez az ISM sáv az, ami nem engedélyköteles (ITU Region 1 ²²)
- Modulation type: 2FSK
- Packet length: 16 byte(s)
- -DRADIO_USE_GENERATED_CONFIGURATION = 1 fordítási csatolóval kiválasztom, hogy az általam generált konfigurációs beállításokat használja az IDE a fordítás során

Az FSK betűszó jelentése: Frequency-shift Keying, vagyis frekvenciaeltolás-billentyűzés. Ez a frekvenciamoduláció egyik módja. A legegyszerűbb FSK moduláció a 2-FSK vagy másik rövidítéssel a BFSK, vagyis bináris FSK, aminek neve arra utal, hogy a bináris 0 és 1 információt 2 különböző diszkrét frekvenciakomponens hordozza. A jel amplitúdója állandó, időrekenként f_1 vagy f_0 frekvenciájú szinuszos csomagokat tartalmaz, amik a 0 és 1 bináris értékeknek ²³ felelnek meg és a jel fázisa is folyamatosan változik. Az amplitúdómodulációhoz képest kevésbé érzékeny a zajokra,

²² https://en.wikipedia.org/wiki/ISM_band#Frequency_allocation

²³ <http://www.sss-mag.com/pdf/1modulation.pdf>

mert azok nagyobb befolyással vannak az amplitúdóra, mint a frekvenciára. A nullátmeneteket jól detektálhatók ennél a moduláció típusnál is.



3-3 ábra: FSK moduláció hullámformái

A 16 bájtos csomaghossz jelentése, hogy egyszerre 16 bájtot képes átküldeni az adó a vevőnek. Ez az én alkalmazásomhoz ideális, mert a 16 bájtbba könnyedén elférnek a mérési adatok. A 16 az a legkisebb kettő hatvány, ami elegendő az általam küldendő adatok mennyiségéhez.

Az általam használt rádiós protokoll Proprietary minősítésű, vagyis egy cég levédette az implementációt és így csak az elérhetővé tett API-t használhatom, a forráskódok és a részletek nem nyilvánosak. Az általam használt mikrokontrollerre integrálva van a rádiós adóvevő, emiatt ez a gyártó – Silicon Labs – által implementált Proprietary rádiós protokollt használja.

A protokoll implementáció használata:

- Az adóvevő áramkör hardveres megszakítással jelzi a processzor felé, ha értelmezhető csomag érkezett
- A küldendő adatok feldolgozása (bájtokká alakítása) után egy szoftveres flaggel jelzem, hogy kezdje meg a küldést a rádiós modul
- A küldendő és a fogadott adatokat egy-egy RAM memóriában található bufferben tárolom. Ezeket a megadott csomagmérethez mérten kell létrehozni és kezelni

3.1.3 Szenzorok

A korábban már említett szenzorok bővebb leírását és a konkrét implementációs részleteket ebben a fejezetben írom le.

3.1.3.1 DHT22

A DHT22 szenzor olvasásához nincs szükség magasabb szintű kommunikációs interfészre, ezt egyetlen GPIO láb használatával vezérelni lehet. A mikrokontrollerhez elérhető `em_gpio` függvénykönyvtár segítségével minden szükséges funkciót meg tudok valósítani ezen szenzor használatához. Szükséges GPIO függvények:

- GPIO láb üzemmódjának beállítása: `GPIO_PinModeSet`
- GPIO láb értékének beállítása: `GPIO_PinOutSet` és `GPIO_PinOutClear`
- GPIO láb értékének beolvasása: `GPIO_PinInGet`

A szenzor kezelése szoftveresen:

1. Logikai magas állapotba kell húzni a GPIO lábat a mérés indításához legalább 250 ms-ig, majd logikai alacsony szintre legalább 20 ms-ig
2. A kritikus időzítési paraméterek miatt minden jellegű megszakítást letilt ideiglenesen a rendszer
3. A mérési folyamat indítását jelző jelsorozat utolsó része, hogy legalább 40 ms-ig magas szintbe állítom a GPIO lábat
4. Pull-up üzemmódba állítom a lábat, hogy a szenzor tudja vezérelni a mérési adatoknak megfelelően (alacsony és magas szintekkel)
5. A szenzor válaszul egy közel 80 ms ideig tartó magas, majd alacsony jelszintet állít elő
6. Ezután a szenzor bitenként elküld 5 bájtot (összesen 40 bit olvasása)
 - a. Minden egyes bit egy 50 ms-os alacsony szinttel kezdődik és ezt követi egy változó hosszúságú logikai magas szintű rész
 - b. Ha a változó hosszúságú rész körülbelül 28 ms hosszú, akkor ez egy 0-ás bit; ha körülbelül 70 ms hosszú, akkor ez egy 1-es bit
7. A 40 bit beolvasása után egy mérés befejeződött és kezdődhet a nyers adat feldolgozása

A nyers mérési adatok értelmezése: a beérkező 5 bájt jelentése az alábbi táblázatban olvasható:

Páratartalom	Páratartalom	Hőmérséklet	Hőmérséklet	Checksum
MSB	LSB	MSB	LSB	

A páratartalom két bájtjából adódik a páratartalom %-os értéke. A hőmérséklet két bájtja kettes komplementben van ábrázolva, ezért az előjelet még bele kell venni a számításba. A hőmérséklet Celsius fokban értendő, ezért a lehetséges nemzetközi felhasználás miatt ajánlott megírni a Celsius fok → Fahrenheit fok átváltást. A checksum a kommunikáció ellenőrzésére van: a felső négy bájt összegének az alsó 8 bitjének egyeznie kell a sikeres kommunikáció esetén. Ha nem egyezik a checksum, akkor újra próbálkozik a rendszer a szenzor kiolvasásával.

$$c \text{ } ^\circ\text{Celsius to } f \text{ } ^\circ\text{Fahrenheit} : (c \text{ } ^\circ\text{C} \times \frac{9^\circ\text{F}}{5^\circ\text{C}}) + 32 \text{ } ^\circ\text{F} = (c \times 1.8) \text{ } ^\circ\text{F} + 32 \text{ } ^\circ\text{F} = f \text{ } ^\circ\text{F}$$

3-4.ábra: A Celsius fokból Fahrenheit fokba átváltás képlete

A szenzor használatához a DHT22_ReadSensor függvény meghívása szükséges, ami egy globális struktúrába írja bele a beolvasott és átalakított páratartalom és hőmérséklet adatokat.

3.1.3.2 BMP280

A BMP280 szenzor az egyik I2C buszon csatlakozik a mikrokontrollerhez, ezért a szenzor használatához az em_i2c függvénykönyvtárat használom. Ez a függvénykönyvtár csak a nagyon alacsony szintű I2C réteget valósítja meg, a különböző regiszterek írása és olvasása, a szükséges bájtok mennyisége már egy magasabb rétegű API megírását igényli a programozótól:

```
BMP280_I2C_ReadRegister(uint8_t registerAddress, int8_t * bufferRead,
uint16_t readSize)
BMP280_I2C_WriteRegister(uint8_t registerAddress, int8_t * bufferWrite,
uint16_t sizeWrite)
```

A fenti két API feladata: egy adott regiszter címétől kezdődően a buffer-ben szereplő size darabnyi bájt írása vagy olvasása a szenzor regisztereibe. Az olvasáshoz WRITE_READ, az íráshoz WRITE_WRITE I2C szekvenciát használok.

Ezekén kívül az I2C busz általános inicializálást is el kell végezni, de ez nem szenzor specifikus művelet.

A szoftvernek a mérési adatok megfelelő értelmezéséhez még kalibrációs adatokra is szüksége van, amiket a szenzor regisztereiből lehet kiolvasni. Ezek a kalibrációs paraméterek gyártás során a szenzor nemfelejtő memóriájába kerülnek beégetésre, ami minden egyes szenzor példánynál egyediek lehetnek.

Table 18: Memory map

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state	
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00	
temp_lsb	0xFB	temp_lsb<7:0>								0x00	
temp_msb	0xFA	temp_msb<7:0>								0x80	
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00	
press_lsb	0xF8	press_lsb<7:0>								0x00	
press_msb	0xF7	press_msb<7:0>								0x80	
config	0xF5	t_sb[2:0]			filter[2:0]			spi3w_en[0]		0x00	
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00	
status	0xF3	measuring[0]					lim_update[0]				0x00
reset	0xE0	reset[7:0]								0x00	
id	0xD0	chip_id[7:0]								0x58	
calib25...calib00	0xA1...0x88	calibration data								individual	

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Revision	Reset
Type:	do not write	read only	read / write	read only	read only	read only	write only

3-5 ábra: BMP280 memóriatérképe a regiszterek rövid leírásával

A szenzor használata:

1. A szenzorban tárolt CHIP ID lekérdezése és összehasonlítása az adatlapban tárolt számmal, hogy az eszköz helyes működését ellenőrizsem
2. A 3-5 ábra: BMP280 memóriatérképe a regiszterek rövid leírásával ábránál látható, hogy a kalibrációs paraméterek kiolvasása úgy történik, hogy a 0x88 címtől kezdően 26 bájtot kiolvasok a szenzor regisztereiből és eltárolom ezeket egy globális struktúrában
3. A tényleges mérési adatok kiolvasásához szükséges, hogy a 0xF7 címtől (press_msb) kezdődően 6 egymást követő bájtot olvassak ki a regiszterekből. Ez a 6 bájt a hőmérséklet és a nyomás 3-3 bájton elférő bitjeit jelenti.
4. A hőmérséklet és nyomás 3-3 bájtját ezután a kalibrációs paraméterekkel kompenzálom és további számításokkal megkapom a hőmérsékletet Celsius fokban és a nyomás Pa mértékegységben.

var1 =	128793,1787		var1 = (((double)adc_T)/16384.0 - ((double)dig_T)/1024.0) * ((double)dig_T2);
var2 =	-370,8917052		var2 = (((double)adc_T)/131072.0 - ((double)dig_T)/8192.0) * (((double)adc_T)/131072.0 - ((double)dig_T)/8192.0) * ((double)dig_T3);
tfine =	128422		t_fine = (BMP280_S32_t)(var1 + var2);
T =	25,08	Temperature [°C]	T = (var1 + var2) / 5120.0;
integer result (**):	2508	Temperature [1/100 °C]	
var1 =	211,1435029		var1 = ((double)t_fine/2.0) - 64000.0;
var2 =	-9,523652701		var2 = var1 * var1 * ((double)dig_P6) / 32768.0;
var2 =	59110,65716		var2 = var2 + var1 * ((double)dig_P5) * 2.0;
var2 =	187120057,7		var2 = (var2/4.0) + ((double)dig_P4) * 65536.0;
var1 =	-4,302618389		var1 = (((double)dig_P3) * var1 / 524288.0 + ((double)dig_P2) * var1) / 524288.0;
var1 =	36472,21037		var1 = (1.0 + var1 / 32768.0) * ((double)dig_P1);
p =	633428		p = 1048576.0 - (double)adc_P;
p =	100717,8456		p = (p - (var2 / 4096.0)) * 6250.0 / var1;
var1 =	28342,24444		var1 = ((double)dig_P3) * p * p / 2147483648.0;
var2 =	-44875,50492		var2 = p * ((double)dig_P8) / 32768.0;
p =	100653,27	Pressure [Pa]	p = p + (var1 + var2 + ((double)dig_P7)) / 16.0;
int32 result (**):	100653	Pressure [Pa]	
int64 result (**):	25767236	Pressure [1/256 Pa]	

(**) The actual result of the integer calculation may deviate slightly from the values shown here due to integer calculation rounding errors

3-6 ábra: Nyers adatokból értelmezhető mennyiségek számítása a BMP280 szenzornál

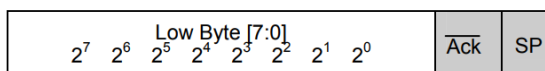
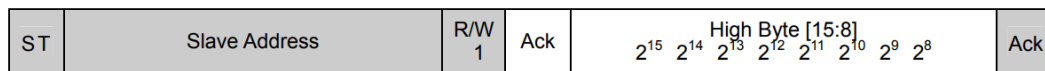
A BMP280 szenzor szoftveres kezelése összefoglalva:

1. Kalibrációs adatok kiolvasása
2. Nyers mérési adatok kiolvasása
3. Kompenzálás
4. Hőmérséklet és légnyomás felhasználása

3.1.3.3 BH1750FVI

A BH1750FVI szenzor a BMP280-hoz hasonlóan I2C buszon kapcsolódik a mikrokontrollerhez. A nagyon alacsony szintű I2C rétegre kell építeni egy magasabb szintű réteget, ami a konkrét szenzor specifikus módon használja az I2C írás és olvasás műveleteket. Ennél a mérő egységnél különböző utasításokat lehet kiadni a modulnak és erre ő válaszolni képes, ha olyan instrukciót adunk meg. Például a reset utasításra nem várunk választ, de egy *One-Time H-Resolution Mode2* utasításra (BH1750FVI fejezetben részletezem ennek az üzemmódnak a működését) válaszként a mérési adatokat kapjuk vissza a szenzortól.

BH1750FVI is not able to accept plural command without stop condition. Please insert SP every 1 Opcode.



from Master to Slave



from Slave to Master

3-7 ábra: Utasítás küldése a BH1750FVI szenzornak

A One-Time H-Resolution Mode2 üzemmód használatához szükséges lépések az I2C kommunikáció részeként:

1. Master küld egy START bitet az I2C buszon
2. Master kiküldi a slave egység címét (0x23) és az írás jelölő 0 bitet
3. A slave egység ACK-al jelzi, hogy figyel és fogadni fogja az érkező adatokat
4. A master elküldi az üzemmód kódját (0x21)
5. A slave ACK-val jelzi, hogy fogadta
6. Master küld egy STOP bitet
7. Várakozás a mérés befejezésére, ennél az üzemmódnál ez legfeljebb 180 ms
8. Master küld egy START bitet az I2C buszon
9. Master kiküldi a slave egység címét (0x23) és az olvasást jelölő 1 bitet
10. A slave egység ACK-al jelzi, hogy figyel
11. A slave egység elküldi a mért adat felső bájtját
12. A master ACK-val jelzi, hogy fogadta
13. A slave egység elküldi a mért adat alsó bájtját
14. Master küld egy NACK és egy STOP bitet a kommunikáció lezárására

A szenzor használatának összefoglalása:

1. Mérési üzemmód küldése a szenzornak
2. Várakozás a mérés befejezésére
3. I2C olvasás indítása
4. Mérési adatok fogadása 2 bájtton (először MSB)
5. Nyers bájtokból a lx számítása

A nyers bájtokból való lx számítása: az MSB és LSB bájtokból a 2 bájtos adat előállítás, ezt nevezzük `raw_data`-nak. Mivel H-resolution üzemmódot használunk, ezért a `raw_data`-t el kell osztani kettővel, mert a legalsó bit kettőnek a -1-edik hatványát jelöli. Ezután a `raw_data / 2` művelet eredményét el kell osztani a Measurement accuracy-vel, aminek tipikus értéke 1,2. Tehát a `feny_lx`-jel jelölt fényesség lx mértékegységben a következő: $\text{feny_lx} = (\text{raw_data} / 2) / 1,2$.

3.2 Központi egység

A központi egység több különböző funkciót lát el:

1. Adatok fogadása a rádiós protokollon keresztül
2. Adatok feldolgozása (például nyers adatból előjeles hőmérséklet kiszámítása)
3. Adatok továbbítása az adatbázishoz

A központi egység két fő részből áll: egy rádió adóvevőt tartalmazó mikrokontrolleres kártyából és egy single-board computerből (SBC). A mikrokontrolleres fejlesztőkártya a mérőegységben használt eszközzel teljesen azonos. Ennek oka, hogy egy Silicon Labs által gyártott rádiós kezdőkészlet van, amiben két megegyező rádió adóvevős modul található. A mérőegység modulja csak küldi az adatokat, a központi egység modulja kizárólag fogadja az érkező adatokat. A single-board computer egy Raspberry Pi 2 B+ miniszámítógép. A frissebb kiadásokon már megtalálható az integrál WiFi és a Bluetooth vezérlő, de az én 2 B+ egységemen még egyik sincs az előző két áramkör közül. Erre megoldást jelent, hogy a Gecko-s kártya rádiós vevőként tudja fogadni az adatokat, valamint Ethernet technológia segítségével vezetékes kapcsolaton keresztül kommunikál az interneten. A központi egység egyik fontos eleme a miniszámítógéphez csatlakozó érintésérzékeny képernyő. A kijelző egy a

felhőben futó szerverem által generált oldalt jelenít meg, ahol mindig a legutóbbi mérési adatok láthatók, valamint a jelenlegi dátum és idő.

3.2.1 Tápellátás

A központi egység nagy kijelzője és folyamatos vezetékes internetkapcsolata miatt a tápfeszültséget is a hálózati aljzathból kapja, ezért nem szükséges akkumulátor vagy egyéb energiát tároló egység. A Raspberry Pi-hez szükséges feszültséget és áramot egy Raspberry Pi Universal Power Supply ²⁴ szolgáltatja. Ez az egység a hálózati feszültségből előállítja a miniszámítógéphez szükséges 5,1 V-ot, illetve maximálisan 2,5 A áramot tud leadni. A Raspberry a micro USB csatlakozóján keresztül kapja a szükséges feszültséget és áramot a Power Supply-ből.

3.2.2 Felhasználói kezelőfelület

A felhasználói kezelőfelület a Raspberry Pi-hoz tartozó érintésérzékeny kijelzőn található. A felhasználónak lehetősége van a legutóbbi mérési adatok megtekintésére, az aktuális dátum és idő kiolvasására, valamint néhány egyszerűbb funkció közötti választásra. A webszerver rendelkezik egy /hmi endpoint-tal, ami rendereli és visszaküldi a kliensnek a megjelenítendő kezelőfelületet. Minden egyes felhasználói beavatkozás újabb http kérést indít a szerver felé, ami lehetővé teszi a legfrissebb adatok folyamatos megjelenítését. A dátum és idő pontos megjelenítése miatt az oldal percenként újratöltődik.

3.2.3 Kommunikáció

A központi egység többféle kommunikációt is használ:

1. Először rádiófrekvenciás protokollon keresztül adatokat fogad a mikrokontrolleres modul a mérőegységtől.
2. Ezután a mikrokontroller UART-on keresztül elküldi a rádión érkezett adatokat a Raspberry-nek.
3. A Raspberry a frissen kapott mérési adatokat HTTP GET kéréssel elküldi a felhőben futó webszervernek.

²⁴ <https://www.raspberrypi.org/products/raspberry-pi-universal-power-supply/>

3.2.3.1 Rádiófrekvenciás protokoll

A központi egység Gecko-s vevő egysége ugyanazt a Silicon Labs által levédetett, Proprietary rádiós protokollt használja, mint az adó egység. A rádiós áramkör egy hardveres megszakítással jelzi a processzor számára, ha a protokollnak megfelelő csomag érkezett, ami készen áll a feldolgozásra. Ekkor lefut a megszakításhoz tartozó IT handler függvény, amiben feldolgozom a bájtokként érkezett mérési adatokat. Ez főként csak a megfelelő helyiértékek előállítását jelenti, hiszen a bájtokból elő kell állítanom a 2 illetve 4 bájtos adatokat. Például a fényerősség 2 bájtos előjel nélküli szám, mert 0...65535 között ábrázolja a fényességet. Tehát az MSB és LSB beérkezése után egyszerű 8 bites bináris eltolással előáll a fényerősség egy uint16_t típusú változóban.

3.2.3.2 UART

A Gecko-s vevő áramkör a rádión beérkező adatokat UART-on továbbítja a Raspberry Pi miniszámítógép felé egy TX vezetéken keresztül. Az érkezett adatokból egy sztringet állítok elő, amit karakterenként küldök tovább a központi számítógép felé minden egyes rádiós csomag érkezésekor.

A Gecko egy hardveres megszakítást kap, amikor érvényes rádiós csomag érkezik a mérőegységtől a 433 MHz-es frekvencián. A csomag tartalma a nyers mérési eredmények, amikből egy sztringet készítek. A sztring célja, hogy egy keretbe foglalja az adatokat és így tudni lehet a miniszámítógépnél, hogy megérkezett-e az egész üzenet, vagy valahol hiba történt és lemaradt a keret eleje vagy a vége. Ha az egész keret átért, akkor kijelenthetjük, hogy a kommunikáció sikeres volt és lehet tovább dolgozni az adatokkal a Raspberry-n.

A miniszámítógépen egy Python szkript figyeli a soros porton érkező adatokat. Ha az általam meghatározott keret eleje és vége is megérkezik, akkor a keret határai közötti adatokat elkezdi feldolgozni és továbbítani a szerver felé.

```
bencesebok@bencesebok-vmubuntu:~$ python serialReader.py  
[*temp=26.12,hum=46.21,pres=100023,light=95,measured=2018.11.09.17:15:58*]
```

3-8 ábra: Soros porti adatfogadás a miniszámítógépen

3.2.3.3 HTTP

A Raspberry Pi egy Python nyelven írt egyszerű szkript segítségével az UART-on érkezett adatokból elkészíti a GET paramétereit, majd elküldi a /new végpontra.

A végpont a domain/thesis/new útvonalon érhető el és GET kéréssel kell küldeni az adatokat a kérés paramétereiként. POST HTTP kérés helyett elegendő a GET is, mert nem küldök érzékeny, biztonságkritikus adatokat, mint például jelszót.

Példa egy mérési adat feltöltésére:

<http://bencesebok.azurewebsites.net/thesis/new?node=indoor1&temperature=21&humidity=40&pressure=100000&light=1000>

Ebben a GET kérésben könnyedén leolvasható a szükséges formátum:

- node: melyik mérőegység mérte az adatokat
- temperature: a mért hőmérséklet Celsius fokban
- humidity: a páratartalom %-ban
- pressure: légnyomás Pa-ban
- light: fényerősség lx-ban

Ezekből az adatokból egy SQL lekérdezést generálok:

```
const query = "INSERT INTO [Measurements] ( [node], [temperature],  
[humidity], [light], [pressure], [measured] ) VALUES ('" + node + "', " +  
temperature + ", " + humidity + ", " + light + ", " + pressure + ",  
GETDATE() );";
```

Itt a kapott paraméterek alapján előállítok egy sztringet, amit lefuttatok ezután az adatbázisban, hogy eltárolja hosszútávon a mért adatokat.

3.3 Felhős szolgáltatások

A felhős szolgáltatások implementációjához olyan technológiákat választottam, amikben már van tapasztalatom és így a tanulás helyett tudok a prototípus fejlesztésre koncentrálni.

A webes technológiák terén a HTML, CSS alapvető fontosságú, hiszek ezek írják le a weboldal struktúráját és stílusát. Ezek a leíró nyelvek csak statikus tartalom leírására szolgálnak, ezért dinamikus elemekhez szükség van egyéb technológiákra is.



Manapság az egyik legnépszerűbb és legmodernebb nyelv, ami képes dinamikus tartalmak kezelésére a JavaScript. Ez annyira sokoldalú, hogy képes mind a kliens- és mind a szerveroldali kódok elkészítésére.

3.3.1 Microsoft Azure

A Microsoft Azure-ban futtatható Node.js alapú webservert, ami JavaScript nyelven van implementálva. A kliensoldali részt a PUG.js keretrendszerrel valósítottam meg. Látható, hogy JavaScript nyelven implementáltam a szerver- és kliens kódokat egyaránt, így egyetlen szkript nyelv segítségével tudtam fejleszteni a felhős szolgáltatásokat.








3.3.2 Adatbázis

Az Azure-ban egy Azure SQL Database nevű adatbázist használok, ami egy felhő alapú relációs adatbázis. Ehhez egy SQL server és egy SQL Database erőforrásra van szükség az Azure-ban.

	thesisDatabase (bencesebok/thesisDatabase)	SQL database	VstsRG-bencesebok-a4ad
	bencesebok	SQL server	VstsRG-bencesebok-a4ad

3-9 ábra: Adatbázis erőforrások az Azure-ban

Az Azure-ban többféleképpen is lehet kezelni az adatbázist. A legtöbb funkciót az Azure SQL lekérdező nyelv segítségével lehet elérni. Vannak nagyon gyakran használt eszközök, ezeket egy grafikus felületen is el lehet érni.

▼	Tables
▼	dbo.Measurements
	id (PK, int, not null)
	node (varchar, not null)
	temperature (real, not null)
	humidity (real, not null)
	light (int, not null)
	pressure (int, not null)
	measured (datetime, not null)

3-10 ábra: Adatbázis struktúra grafikusán

Az adatbázisban egyetlen tábla van: Measurements. Ennek struktúrája:

- id: egyedi azonosító szám, ami minden egyes rekordra egyedi
- node: melyik mérőegység mérte az adatokat
- temperature: a hőmérséklet valós számként Celsius fokban
- humidity: a páratartalom valós számként %-ban
- light: a fényerősség egész számként, mert csak 0...65535 értékeket vehet fel
- pressure: légnyomás egész számként Pa mértékegységben
- measured: datetime típusú időbélyeg, ami a mérés időpontját jelenti

3.3.2.1 Azure SQL

Az adatbázishoz tartozó legtöbb funkciót inkább az Azure SQL nyelvvel érdemes használni, mert itt az igényekhez megfelelően szabadon lehet paraméterezni minden lekérdezést.

```
select *  
from INFORMATION_SCHEMA.COLUMNS  
where TABLE_NAME='Measurements';
```

Az előző kódrészlet például lekérdezi az adatbázisból a Measurements tábla adatait, ami szövegesen jeleníti meg a 3-10 ábra: Adatbázis struktúra grafikusán képen látható struktúrát.

Az adatok lekérdezésére szolgáló SQL kérés:

```
select top 3 * from Measurements order by ID desc;
```

Ezzel a legutóbbi 3 mérési eredményt lehet kiolvasni az adatbázisból.

Az új adatok beszúrásához a következő JavaScript részletet használom:

```
const query = "INSERT INTO [Measurements] ( [node], [temperature],  
[humidity], [light], [pressure], [measured] ) \  
VALUES ('" + node + "', " + temperature + ", " + humidity + ", " +  
light + ", " + pressure + ", GETDATE() );";  
console.log(query);
```

Ez a kódrészlet a kientől kapott mérési eredmények alapján készít egy SQL lekérdezést, amit az adatbázis már értelmezni tud és így elmenti a Measurements táblába a legfrissebb adatokat.

3.3.3 Web szerver

Az Azure-ban egy Node.js alapú webszervert készítettem, ami kiszolgálja a kliens böngészőből érkező kéréseket és ezek alapján elvégzi a szükséges műveletet.

A webszer főbb műveletei:

- Adatok lekérdezése az adatbázisból
- Adatok feltöltése az adatbázisba
- Kliens oldali felületek generálása az adatbázisban szereplő adatok alapján
- Statikus kliens oldali felületek generálása az adatbázistól függetlenül

3.3.3.1 Node.js

A Node.js egy JavaScript nyelven implementált webszerver, ami manapság nagyon elterjedt. Ennek nagy előnye, hogy nagyon sok segédanyag és példa kód érhető el. A hivatalos dokumentáció is nagyon részletes és nagy segítséget jelent a fejlesztés során.

A webszerver különböző végpontjait az Express.js keretrendszer segítségével kezelem (Using middleware, 2018). Ez egy Node.js package, ami leegyszerűsíti a különféle útvonalakra érkező http kéréseket.

A webszer végpontjai, amiken a HTTP kéréseket fogadja:

- /measurements: a mérési eredmények között lehet böngészni
- /hmi: a felhasználói kezelőfelületet generálja le az adatbázisban szereplő adatok alapján
- /new: új mérési eredményt lehet feltölteni az adatbázisba
- /list/last: a legutolsó mérési eredményt adja vissza, ez a /hmi végponthoz is szükséges
- /db: az adatbázis tesztelésére szolgál

Egy példa kérés:

<http://bencesebok.azurewebsites.net/thesis/new?node=indoor1&temperature=21&humidity=40&pressure=100000&light=1000>

Ez a GET kérés a /hmi végpontra érkezik, ami a feltöltendő adatokat is tartalmazza. Ennek a feldolgozása az Express.js-ben a következő:

```
/* Insertion of new measurement endpoint. */
router.get('/new', function (req, res) {
  insertData(req.query);
  res.send(req.query);
});
```

A kérésben szereplő paraméterek értelmezése már az insertData függvényben található:

```
const node = data.node
const temperature = data.temperature;
const humidity = data.humidity;
const pressure = data.pressure;
const light = data.light;
const query = "INSERT INTO [Measurements] ( [node], [temperature],
[humidity], [light], [pressure], [measured] ) \
VALUES ('" + node + "', " + temperature + ", " + humidity + ", " +
light + ", " + pressure + ", GETDATE() );";
```

Ez a kódrészlet logikai egységekre bontja a kérés paramétereit, ezekből készít egy SQL lekérdezést az adatok beszúrásához, amiket tovább küld ezután az adatbázisnak.

3.3.4 Webes kliens alkalmazás

A webes kliens alkalmazásnak két fő része van:

1. Böngészőben futó weboldal
2. A beágyazott kijelzőn megjelenő felhasználói interfész

Mindkét részt a webservert generálja le. A kliens oldali tartalmakat a PUG.js keretrendszer segítségével implementáltam.

3.3.4.1 PUG keretrendszer

A PUG.js egy olyan npm package, ami a Node.js-sel és az Express.js keretrendszerrel is könnyedén integrálható (PUG Documentation, 2018).

Ezzel a JavaScript keretrendszerrel könnyedén leírhatók a weboldalak struktúrája, ami a HTML és CSS kódokat generálja a leírásunk alapján. A webes kliens minden oldalának a közös menüje a következő kódrészlettel van leírva:

```
.navbar.navbar-inverse.navbar-fixed-top
  .container
    .navbar-header
      button.navbar-toggle(type='button',      data-toggle='collapse',
data-target='.navbar-collapse')
      span.icon-bar
      a.navbar-brand(href='/thesis') Thesis
    .navbar-collapse.collapse
      ul.nav.navbar-nav
        li
          a(href='/thesis') Home
        li
          a(href='/thesis/measurements') Measurements
        li
          a(href='/thesis/statistics') Statistics
        li
          a(href='/thesis/help') Help
        li
          a(href='/thesis/about') About
```

Látható a fenti kódrészleten, hogy ez se nem HTML, se nem CSS. Viszont ezen leírás alapján a PUG keretrendszer elkészíti a szükséges HTML és CSS fájlokat.

Vannak újrahasznosítható, moduláris részei is, amik függvény-szerűek működnek és *mixin* a nevük:

```
mixin displayWeather(data)
  .article
    .article-wrapper
      p Temperature: #{data.temperature} °C
      p Atmospheric pressure: #{data.pressure} Pa
      p Relative humidity: #{data.humidity}%
      p Light level: #{data.light} lux
      p Timestamp: #{data.measured}
```

Ez a mixin egy data névvel jelölt mérési eredmény struktúrát jelenít meg egy konkrét elrendezéssel és stílussal.

A beágyazott kijelzőn futó felhasználói kezelőfelületet a /hmi végpontra érkező kérés alapján generálja le a webszerver. Ez is egy PUG-ban leírt weboldal struktúra, ami a kis felbontású kijelzőre van tervezve és optimalizálva.

Simple Home Weather Station



2018. 12. 03. 12:20:19

Today's weather:

Temperature: 21 °C

Atmospheric pressure: 100000 Pa

Relative humidity: 40%

Light level: 1000 lux

Timestamp: 2018. 12. 03. 12:17:01

Bence Sebok, Fall 2018

Budapest University of Technology and Economics

Department of Measurement and Information Systems

3-11 ábra: Beágyazott felhasználói kezelőfelület

4 Tesztelés

Az elkészült prototípus szinte az összes szükséges funkciót megvalósítja. Ezek helyes működést ellenőrizni, tesztelni kell. A prototípus részei:

- A mérőegység részeként egy mikrokontroller és a három szenzor. A kontroller végzi a szenzorok vezérlését – mérések indítása és adatok kiolvasása – valamint a mérési adatok továbbítását a központi egység felé.
- A központi egység alapját jelentő miniszámítógép fogadja a mérési adatokat a rádiós vevőmodulként működő Gecko mikrokontrollertől, valamint kezeli a felhasználói kezelőfelületet megvalósító érintésérzékeny kijelzőt, és továbbítja az érkező adatokat a felhős webszerver felé.
- A felhős szolgáltatások az Azure-ban futó webszerver segítségével érhetők el. A különböző végpontok által lekérdezhetők és feltölthetők a mérési adatok, valamint többféle kliens oldali megjelenítést is le tud generálni.

A tesztelés folyamata:

1. Várakozás mérésre (5 perces mérési periódusidő)
2. Mérés automatikus elvégzése a mikrokontroller által
3. Mérés után a mérési adatok ellenőrzése a mikrokontroller debuggolásával
4. Várakozás a mérési adatok fogadására a központ egységnél
5. Az adatok beérkezése után a fogadott adatok összehasonlítása a mikrokontroller debuggolása során leolvasott adatokkal
6. Várakozás a webszerver felé való továbbításra
7. A webszerver új mérési adatokat fogadó végpontjára érkező HTTP kérés tartalmának ellenőrzése a webszerver debug konzolján

8. A webservertől az adatbázisba frissen elmentett adatok összehasonlítása a mikrokontroller által mért és a központi egység által fogadott adatokkal
9. Várakozás a felhasználói kezelőfelület frissítésére (1 perces frissítési periódusidő)
10. A felhasználói kezelőfelület frissítése után a kijelzett adatok összehasonlítása a mikrokontroller által mért és a központi egység által fogadott adatokkal, valamint az adatbázisban szereplő eltárolt adatokkal
11. Mérés kiértékelése

A kiértékelés során 5 különböző helyen kell ellenőrizni a mérési adatokat:

- Mikrokontroller belső RAM memóriája a debug interfészen keresztül
- Miniszámítógép által fogadott adatok egy Linux-os parancssorban
- Webservertől érkező adatok a szerver debug konzolján (Linux-os parancssor az Azure-ban futó virtuális gépen)
- Az adatbázisban eltárolt adatok a *Measurements* táblában
- A felhasználói kezelőfelületen megjelenő grafikus adatok

Ha ezek bármelyikén eltérés vagy hiba jelenik meg, akkor az adott helynél egy lépéssel korábbi egység és a vizsgált hely közötti kommunikáció debuggolása során ki kell deríteni az eltérés vagy hiba okát.

4.1 Hardver prototípus

A szakdolgozatomban részeként még nem tervezem saját nyomtatott áramkört. A WSTK6200-hoz tartozó fejlesztőkártyához készítettem próbanyákon egy kiegészítő áramkört, ahol összeforrasztottam a szükséges szenzorokat, hogy modulárisan kezelhető legyen. A fejlesztőkártya aljára lehet illeszteni a próbanyákon szereplő szenzorokat egy hüvelysor segítségével. Ezáltal tetszőleges fejlesztőkártyához lehet illeszteni a demonstrációs célú szenzor rendszert, ahol megtalálhatók a szükséges perifériák – például az I2C busz.

TODO: fotó a próbanyákról

4.2 Tesztkörnyezet

A megépített prototípust beltéri környezetben teszteltem. A tesztelés paramétereit:

- A központi egység a hálózati aljzatból kapja a tápfeszültséget
- A központi egység összes része – Raspberry Pi, Gecko-s vevő egység, beágyazott kijelző – egy asztalon helyezkedik el a környű kezelhetőség miatt
- A mérőegység körülbelül 5 méternyire található az ablakpárkányban
- A mérőegység tápellátását egy feltöltött powerbank végzi

TODO: fotó a mérésről

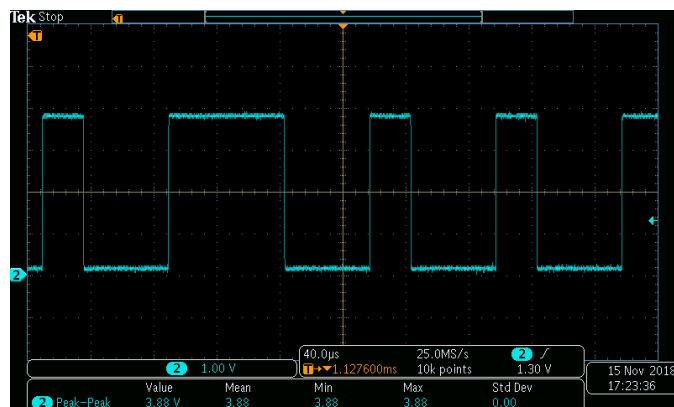
4.3 Mérések

Az egyes szenzorok kimeneteit, méréseit mérőműszerek segítségével is megvizsgáltam. A mikrokontrollerhez illesztett szenzorok GPIO és I2C jeleit megmértem és ábrázoltam oszcilloszkóp segítségével.

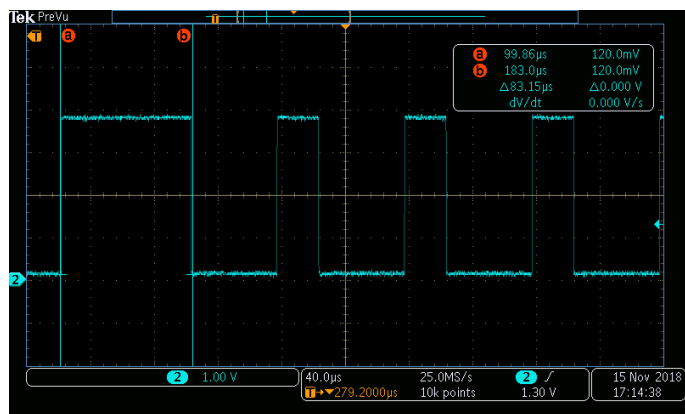
4.3.1 DHT22 GPIO lábának mérése

A DHT22 szenzor által mért hőmérséklet és páratartalom adatok egyetlen GPIO lábon mért feszültség szint időbeli változásával megkaphatók.

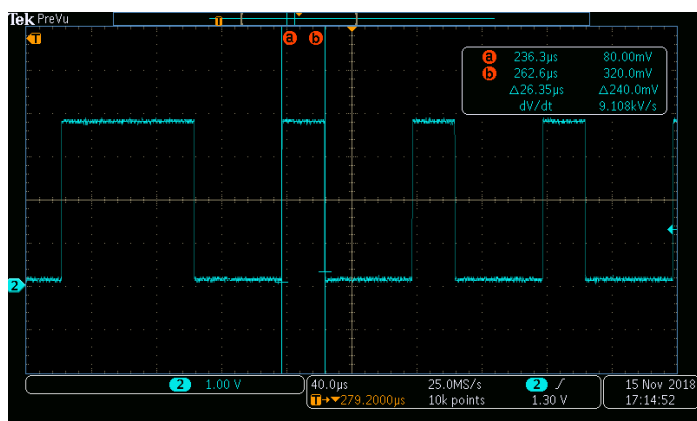
A 4-1 ábra: DHT22 kimeneti jelsorozata képen látható a GPIO lábon mérhető jelsorozat. A különböző hosszúságú impulzusok megfelelnek a bináris 0-1 értékekhez tartozó logikai alacsony és magas szintekhez.



4-1 ábra: DHT22 kimeneti jelsorozata



4-2 ábra: Logikai magas szint pulzusszélessége

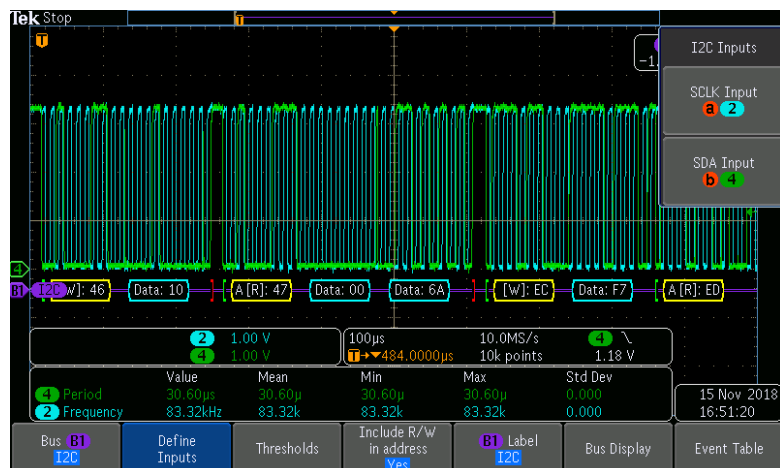


4-3 ábra: Logikai alacsony szint pulzusszélessége

A 4-2 ábra: Logikai magas szint pulzusszélessége és az azt követő 4-3 ábra alapján a logikai magas szinthez egy körülbelül 83 μ s hosszúságú impulzus tartozik, míg az alacsony szinthez egy nagyjából 26 μ s-os impulzus.

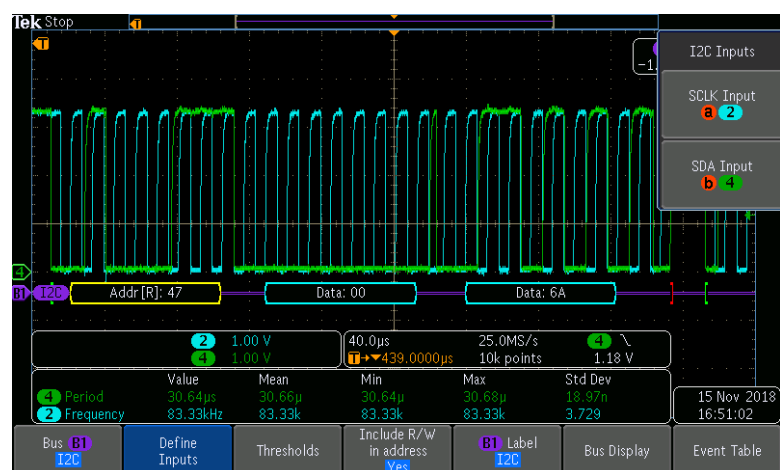
4.3.2 I2C busz mérése

Az I2C buszhoz tartozó SDA és SCL jelek oszcilloszkópon való mérésből a következő ábrák készültek:



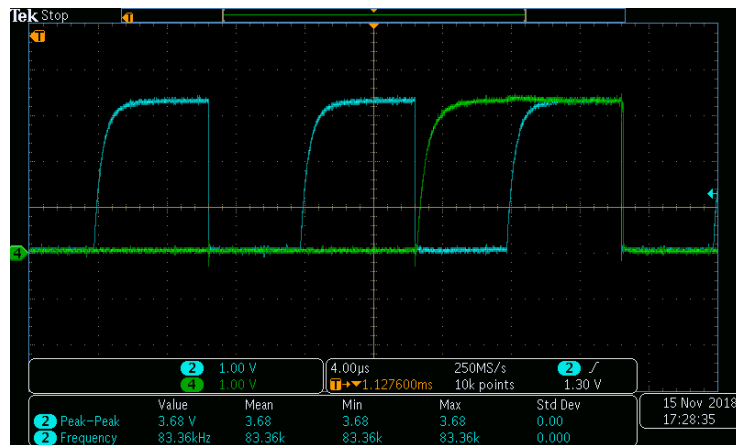
4-4 ábra: I2C busz mérése

A 4-4 ábra: I2C busz mérése ábrán látható az I2C buszon mérhető digitális jelek együttes képe. Egy ennél részletesebb ábrán a pontosabb jelalak és a hozzá tartozó adat is leolvasható:



4-5 ábra: I2C adatok mérése

A 4-5 ábra: I2C adatok mérése képen látható a négyszögjelekre hasonlító órajel és adat vonal jelalakja. A használt oszcilloszkópon van I2C busz analízátor üzemmód, ami megmutatja, hogy a 0x47 című slave egységről olvasok adatot ennél a mérésnél, ahol a kiolvasott két bájt a 0x00 és a 0x6A.



4-6 ábra: I2C jelforma

A 4-6 ábra: I2C jelforma ábrán látszik, hogy nem tökéletes négyszögjelek az I2C buszon mérhető digitális jelek. Ennek okai a nem ideális alkatrészek és a parazita hatások.

4.3.3 Mért adatok

Több mérést is végeztem, amik közül egy esti alkalmat szeretnék kiemelni. Egy olyan mérési adathalmazt mutatnék be, ahol jól észrevehetők a triviális tendenciák.

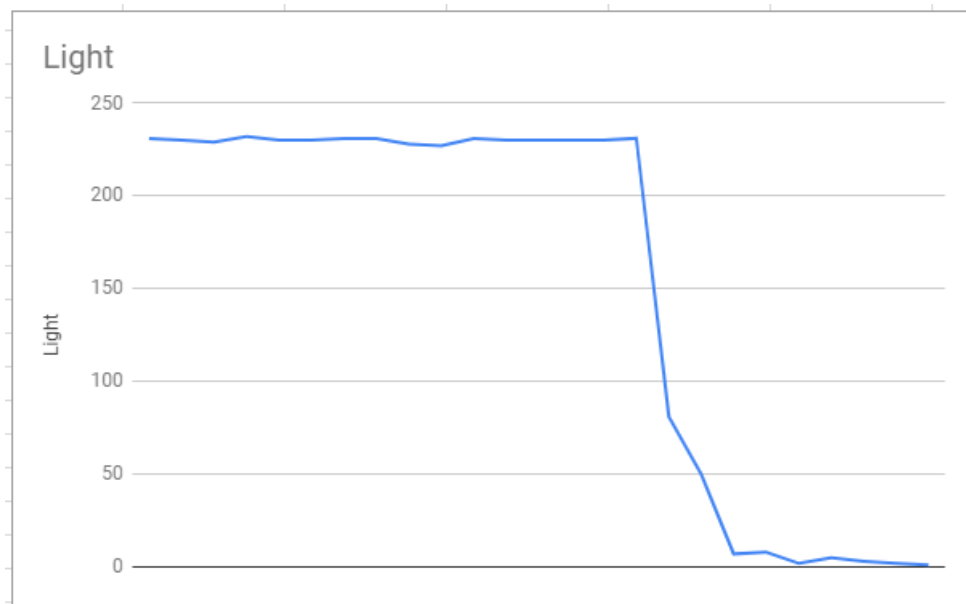
Temperature	Pressure	Light	Humidity
26.15	100023	231	43.45
26.14	100022	230	43.44
26.17	100024	229	43.44
26.11	100029	232	43.44
26.13	100021	230	43.43
26.13	100022	230	43.44
26.09	100023	231	43.44
26.05	100024	231	43.45
26.06	100025	228	43.44
26.02	100024	227	43.43
26.03	100023	231	43.44
26.01	100029	230	43.43
26	100030	230	43.43
26	100032	230	43.44
25.98	100029	230	43.43
25.97	100028	231	43.44
25.91	100026	81	43.43
25.95	100027	50	43.43
25.93	100025	7	43.43
25.92	100026	8	43.43
25.9	100026	2	43.44
25.87	100026	5	43.45
25.88	100025	3	43.46
25.87	100023	2	43.45
25.88	100024	1	43.45

4-7 ábra: Esti mérési adatok

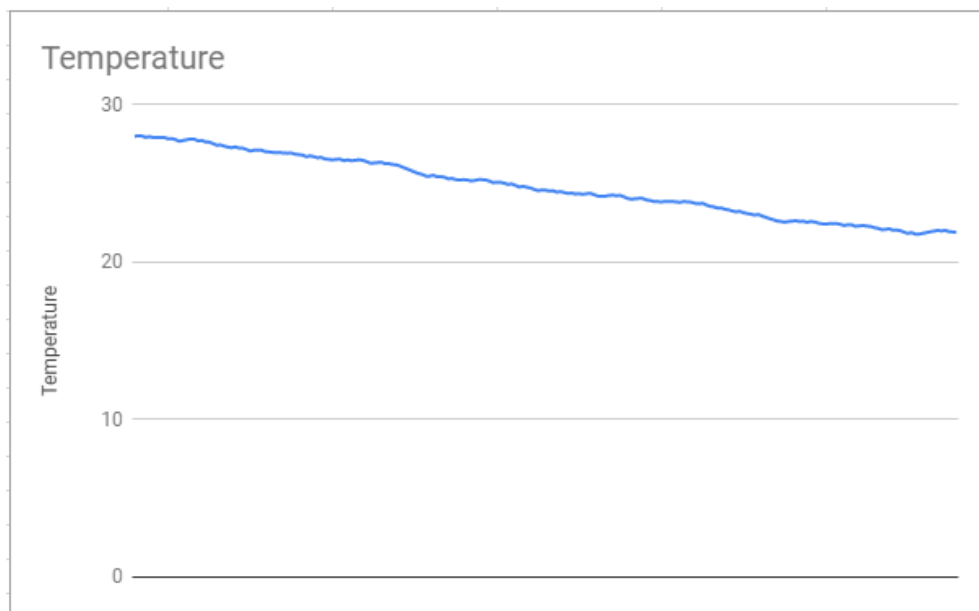
Az esti mérési alkalomból adódik, hogy a hőmérséklet folyamatosan csökken, ahogy a kinti hideg fokozatosan hűti le a szoba hőmérsékletét. A légnyomás egy közel állandó szinten marad, mert a hőmérséklet nem változik túl nagy mértékben és a tengerszint feletti magassága állandó a mérőegységnek. A fényesség mértékénél egyértelműen látszik a szobában található lámpák fokozatos lekapcsolása, a mérés végére már az összes lámpánk ki van kapcsolva. A páratartalom sem ingadozik túlságosan, hiszen csapadék nem került a mérőegység közelébe, a szobában tartózkodók lehelete vagy légmozgás pedig nem befolyásolja nagy mértékben a páratartalmat ebben az esti időszakban.

4.4 Adatok feldolgozása

A mérési adatokat számszerűen tárolom az adatbázisban. Mivel egy rövidebb (körülbelül 30-45 perces) mérést mutatok be, ezért jelentős változás nem látható a mért adatokon. Egyedül a fényerősség mérésénél vehető észre a szobában található lámpák fokozatos lekapcsolása.



4-8 ábra: Fényerősség változása a lámpák lekapcsolásának hatására



4-9 ábra: Hőmérséklet monoton csökkenése az esti órákban

5 Továbbfejlesztési lehetőségek

Többféle különböző módon is tovább lehet fejleszteni a prototípusomat:

1. Egy sokkal kompaktabb hardvert lehetne tervezni, ahol a szenzor modulok helyett az integrált áramköröket és a passzív alkatrészeket egy sokkal kisebb és optimálisabb NYÁK-ra lehetne elhelyezni
2. A webszerverhez valamilyen npm package-ek segítségével sokkal vizuálisabb ábrázolási módokat lehetne implementálni, ahol átlátható és elegáns grafikonokat lehetne generálni a szerver oldalon
3. A beágyazott szoftveren lehetne tovább optimalizálni a fogyasztást az egyes modulokkal kapcsolatos áramfelvételi mérések után. Az egyes szenzoroktól akár a tápfeszültséget is el lehetne venni az alvó vagy energiatakarékos üzemmód alatt
4. A mérőegységhez és a központi egységhez egyaránt lehetne készíteni például egy 3D nyomtatott dobozt, amiben sokkal életszerűbb, valódi fogyasztóiipai termékhez közeli állapotban lehetne tárolni az egységeket
5. A mérőegységhez lehetne választani állandó, tölthető akkumulátort a hozzá tartozó töltőáramkör megtervezésével együtt, ezáltal nem kellene akkumulátort tölteni és cserélni a lemerülés esetén

6 Értékelés

A szakdolgozatomként egy féléves projektként elkészítettem egy otthoni időjárás állomás részletes specifikációját, funkcionális tervét és ezek alapján csináltam egy működő prototípust.

Egy háztartásokba szánt okosotthon-kiegészítőként szolgáló rendszer egészét megvizsgáltam és áttanulmányoztam. Egy ilyen IoT eszközhöz tartozó minden területtel megismerkedtem:

- Célhardverek kiválasztása és alkalmazása
- Egyszerűbb prototípus megépítése
- Költség- és energiahatékonyság alapján való tervezés
- Energiatakarékosságra optimalizált beágyazott szoftver fejlesztése
- Kiegészítő segédfunkciókhoz tartozó nyelvek és fejlesztői környezetek – például Python, Matlab az adatok feldolgozására, továbbítására, ábrázolására
- Szenzorok kimeneteinek mérése oszcilloszkópon, a fogyasztás méréssel való nyomonkövetése

Sokat tanultam a szenzorok illesztéséről és a hozzájuk tartozó szoftver komponensek külön-külön való megírásáról és az ezután következő integrációjukról. Megtapasztaltam, hogy a külön-külön tökéletesen működő hardveres és szoftveres elemek egyberakva már nem feltétlenül működnek együttesen.

Például olyan anomáliát tapasztaltam, hogy a Gecko-s fejlesztőkártya SPI-on vezérelt integrált LCD kijelzője olyan GPIO lábakat is inicializált, amik az I2C busszal átfedésben voltak és például az SCL lábat lehúzták a földre és így tönkretették az I2C kommunikációt. Ennek egyszerű megoldása volt, hogy az LCD kijelzőt nem használtam ezután sem mérési adatok, sem debug információk megjelenítésére.

Megtanultam a fejlesztés során, hogy a beágyazott rendszereknél a különböző bithosszúságú típusok különösen fontosak. Egy furcsa jelenség volt, hogy a Gecko-s mikrokontrolleren a float típusú változók érték szerinti paraméterátadása esetén a belső

RAM-ban nem tudta ábrázolni a szenzorból kiolvasott értéket. Ugyanezt a float változót cím szerinti paraméterátadással viszont tökéletesen kezelni és ábrázolta. Másik hasonló eset, amikor az uint8_t, uint16_t vagy uint32_t típusok között végeztem implicit vagy explicit konverziót. Nagyon oda kellett figyelni, hogy nehogy előforduljon valamilyen nagyságrendbeli adatvesztés.

7 Hivatkozások

Azure SQL Database Documentation. (2018.. 10. 9.). Forrás: docs.microsoft.com:
<https://docs.microsoft.com/en-us/azure/sql-database/>

Node.js Documentation. (2018. 10 1). Forrás: nodejs.org/en/docs:
<https://nodejs.org/en/docs/>

Using middleware. (2018. 10 4). Forrás: Express Guide:
<https://expressjs.com/en/guide/using-middleware.html>