

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Automatika Intézet



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

MODULÁRIS SZENZORHÁLÓZAT TERVEZÉSE ÉS FEJLESZTÉSE

Szerzők: **Kiss-Vámosi Gábor**
villamosmérnök MSc. szak, II. évf.
Pók Péter
villamosmérnök MSc. szak, I. évf.
Törőcsik Márton
villamosmérnök MSc.szak, II. évf.

Konzulens: **Molnár Zsolt**
tanársegéd

Budapest, 2014.

1. Tartalom

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT	1
1. TARTALOM.....	2
2. BEVEZETÉS.....	4
3. SZENZORHÁLÓZATOK	5
3.1. ÁTTEKINTÉS	5
3.2. HÁLÓZATI TOPOLOGIA	5
3.3. KOMMUNIKÁCIÓS ADATSEBESSÉG ÉS HATÓTÁVOLSÁG.....	8
3.4. ENERGIAKEZELÉS, RENDELKEZÉSRE ÁLLÁSI IDŐ.....	9
4. RENDSZERTERV	12
4.1. DIREKTÍVÁK ÉS KÖVETELMÉNYEK	12
4.2. SZENZORHÁLÓZAT FELÉPÍTÉSE.....	12
4.3. RÉTEG STRUKTÚRA	13
4.3.1. <i>Szenzorhálózat vezérlő egység.....</i>	<i>14</i>
4.3.2. <i>Szenzorhálózat átjáró és kontroller.....</i>	<i>15</i>
4.3.3. <i>Szenzorhálózati node</i>	<i>16</i>
5. ALAPLAP	17
5.1. SZEMPONTOK, KÖVETELMÉNYEK	17
5.2. RENDSZERTERV.....	17
5.3. VEZÉRLŐ	19
5.4. TERVEZÉS.....	20
6. ALAPLAPI FIRMWARE	23
6.1. ÁTTEKINTÉS	23
6.1.1. <i>Specifikációs.....</i>	<i>23</i>
6.1.2. <i>Felépítés</i>	<i>23</i>
6.2. BLOKKVÁZLAT	24
6.3. MŰKÖDÉSI MECHANIZMUS	25
6.4. PARANCSON.....	26
6.5. RFID KOMMUNIKÁCIÓ.....	27
7. IQRF VEZÉRLŐ PROGRAM	28
7.1. IQRF ÁTTEKINTÉSE.....	28

7.1.1.	Az IQRF protokoll paraméterei.....	28
7.1.2.	Az IQRF protokoll.....	28
7.2.	IQRF TOVÁBBFEJLESZTÉS.....	30
7.2.1.	Szállítási réteg.....	30
7.2.2.	STD és XLP közti váltás.....	31
7.2.3.	Új API.....	31
7.3.	MEGVALÓSÍTÁS.....	32
8.	VEZÉRLŐ PC-S ALKALMAZÁS.....	34
8.1.	ÁTTEKINTŐ.....	34
8.1.1.	Az alkalmazás futtatásához szükséges követelmények.....	35
8.1.2.	Az alkalmazás futtatása.....	35
8.2.	AZ ALKALMAZÁS FELHASZNÁLÓI ÁTTEKINTÉSE.....	36
8.2.1.	Csatlakozása az IQRF gateway-hez.....	36
8.2.2.	Válaszüzenet konzol értelmezése.....	36
8.2.3.	Hálózati alapadatok lekérdezése.....	37
8.2.4.	Parancsok küldése.....	37
8.2.5.	Válaszüzenet log-ok felépítése.....	37
8.2.6.	Adatok megjelenítése.....	38
8.3.	AZ ALKALMAZÁS FEJLESZTŐI ÁTTEKINTÉSE.....	39
8.3.1.	Az alkalmazás modulok hierarchiája.....	40
9.	TOVÁBBI LEHETŐSÉGEK ÉS CÉLOK.....	42
9.1.	ADAT TÖMÖRÍTÉS.....	42
9.2.	MENTÉS MEMÓRIA KÁRTYÁRA.....	42
9.3.	TELJESEN UNIVERZÁLIS ALAPLAP.....	43
9.4.	DOBOZOLÁS.....	43
9.5.	ÁLTALÁNOSÍTOTT API.....	43
9.6.	KÉT RÉSZRE BONTOTT VEZÉRLŐ ALKALMAZÁS.....	44
10.	ÖSSZEFOGLALÁS.....	45
11.	IRODALOM JEGYZÉK.....	46

2. Bevezetés

A dolgozat részletesen bemutatja egy modern, a mai kor elvárásainak megfelelő Mesh alapú szenzor hálózat tervezésének és fejlesztésének menetét. A szenzorhálózat létrehozása a Bay Zoltán Alkalmazott Kutatási Közhasznú Nonprofit Kft.-nél történt, ahol a szerzők jelenleg junior kutató beosztásban dolgoznak.

A munka során a legfontosabb szempontok a moduláris felépítés, univerzális és sokoldalú használhatóság és a hosszú ideig tartó, autonóm működés kivitelezése volt nagy kiterjedésű hálózatok esetén is.

A rendszerben több száz szenzor állomás (node) is lehet, melyek egyenként maximálisan 8 szenzort képesek kezelni. Az állomások az adatokat az úgynevezett IQRF kommunikációs protokollon keresztül továbbítják egy ethernet gateway-nek. A gateway-t egy router-hez csatlakoztatva az adatok távolról, TCP/IP protokollon keresztül egyszerűen megtekinthetők, illetve a szenzor hálózat menedzselése kényelmesen elvégezhető az erre a célra fejlesztett vezérlő alkalmazás segítségével. Ha egy node valamilyen okból kifolyólag kiesik a hálózathoz, akkor a mért adatok egy kézi olvasóval RFID protokollon keresztül is kinyerhetők.

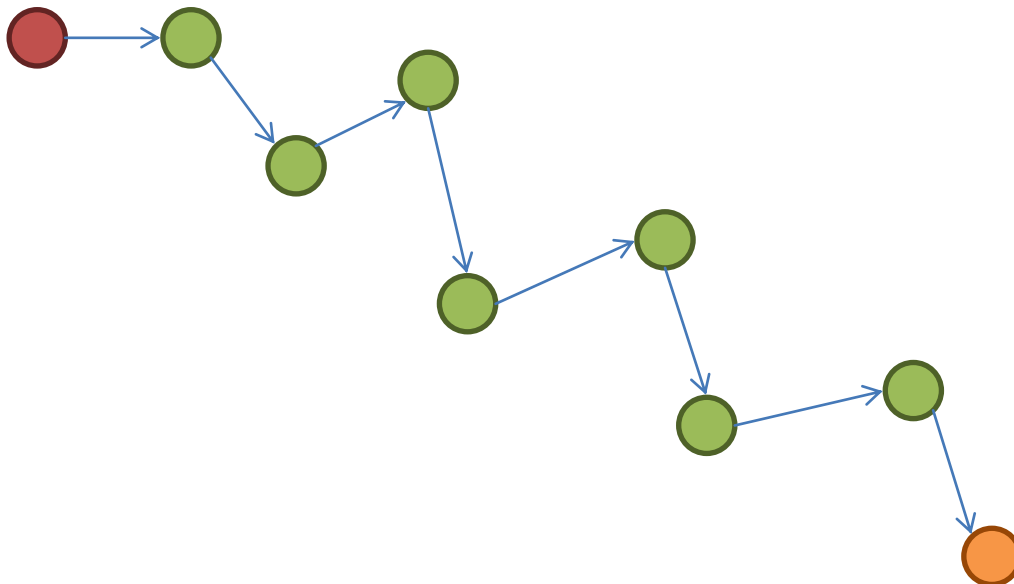
A feladat megvalósítása során elkészült öt szenzor és a node egységek alaplapi áramköre is. Sikeresen maradéktalanul felépíteni az IQRF kommunikációt és megoldani az adatok RFID-n keresztüli kiolvasását is. Emellett a mérési eredmények és egyéb konfigurációs adatok (TCP/IP kapcsolaton keresztül) képesek eljutni egy távoli számítógépre, mely az adatokat vizualizációs eszközök segítségével könnyen értelmezhető formában mutatja meg a felhasználónak, illetve lehetőséget nyújt a beállítások felhasználóbarát módon történő módosítására is.

Munkánk eredményeképpen létrejött egy jól használható, megbízható és univerzális szenzor hálózat rendszer prototípusa. Ahhoz, hogy a rendszer a gyakorlatban is bizonyíthatóan, szükség van még bizonyos mechanikai fejlesztésekre, többek között egy időjárás álló készülék ház létrehozására is.

3. Szenzorhálózatok

3.1. Áttekintés

A szenzorhálózatok általában speciális alkalmazásokhoz elkészített vezeték nélküli kommunikációval megvalósított, fizikai paraméterek nagy kiterjedésű mérését teszik lehetővé, nagyszámú végponttal. A hálózat kommunikációs rétege általában nem klasszikus point-to-point kapcsolatokon alapul, az adatcsomagok továbbításakor a hálózat egésze részt vehet az adatcsatorna létrehozásában, az adatok továbbításában. Ezt szemlélteti a 1. ábra, ahol minden színes kör egy-egy node-ot reprezentál, és a küldő, piros színnel jelölt node-tól az információ a hálózat többi tagján keresztül jut el a címzett, narancssárga színnel jelölt node-ig.



1. ábra A szenzorhálózat kommunikációs rétegének kialakítása

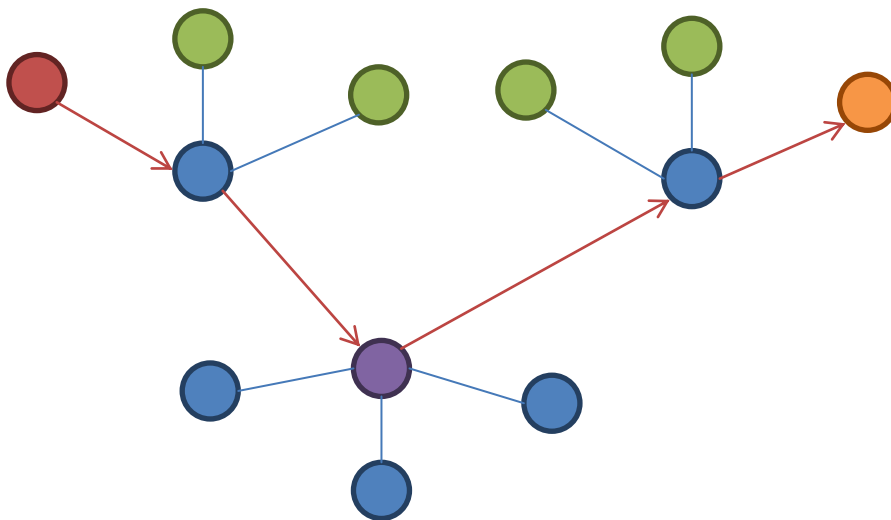
Szenzorhálózatok tervezésekor vannak kritikus jelentőségű szempontok, melyek a létrehozott hálózat működését, hatékonyságát nagymértékben befolyásolják, ezeket a következő alfejezetekben taglaljuk.

3.2. Hálózati topológia

A hálózat kapcsolati működése szempontjából lehet statikus vagy dinamikus felépítésű, az előbbi azt jelenti, hogy a hálózat kommunikációs felépítése a hálózat kiépítésekor kerül meghatározásra, és a node csak meghatározott node-okon keresztül képesek csak kapcsolódni a hálózatba. Dinamikus működés esetén a node-ok közötti kommunikáció

nem ilyen kötött szabályok alapján működik, a node-ok közötti adatcsatorna a node-ok tetszőleges (vagy a node-ok heterogén felépítését figyelembe vevő) kombinációja alapján létrejöhet.

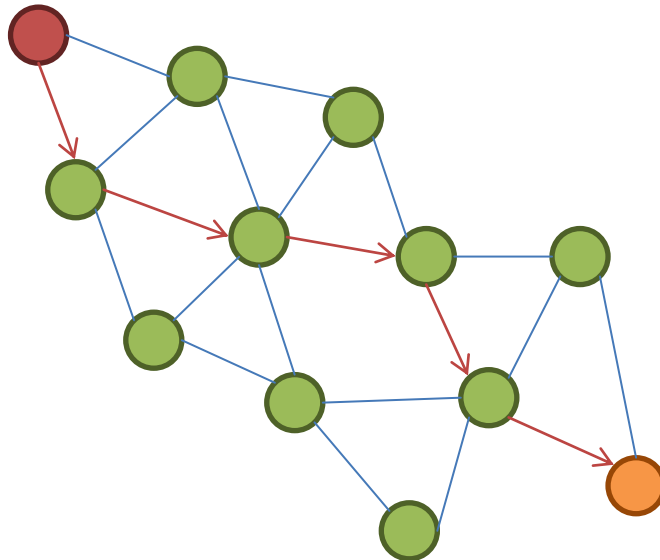
A hálózati topológia létrehozása általában kiterjesztett csillaghálózat vagy mesh-hálózat elrendezés alapján történik, gyűrű topológia nem volna praktikus egy nagy hálózatban, busz topológiáról pedig vezeték nélküli hálózatban nem érdemes beszélni. Csillaghálózatban szükség van kinevezett csomóponti node-okra, melyeken keresztül a hozzájuk csatlakoztatott node-ok kommunikálnak, valamint egy központi hálózat vezérlő egységre, melyen keresztül a kiterjesztett csillaghálózatok közötti üzenetek áthaladnak. Ezt a felépítést mutatja a 2. ábra, ahol a piros színnel jelzett végponti node a narancssárga színnel jelzett node-nak küld üzenetet, a kék színnel jelzett node-ok pedig a csomópontokat reprezentálják.



2. ábra Kommunikáció csillaghálózatban

Mesh hálózati topológiában a kommunikációs csatorna kialakítása szempontjából nincsenek kiemelt szerepű egységek, az információ a node-ok között számos útvonalon terjedhet. Ez persze az adatcsatorna felépülésének bonyolításával jár, hiszen a küldő és a címzett közötti útvonal számos módon létrejöhet. A probléma feloldására alapvetően két módszer van, az egyik az, ha a hálózat egységei ismerik a teljes, vagy legalább a környező hálózat felépítését, akkor ki tudják választani egy csomag továbbítására a legmegfelelőbb szomszédos egységet. A másik módszer, ha az adatsomagokat a hálózat egységei minden szomszédos egységnek továbbítják, „elárasztva” a hálózatot az adatsomaggal. Ebben az esetben a hálózati interferenciák, zavarok elkerülésére szükség van az adattovábbítással

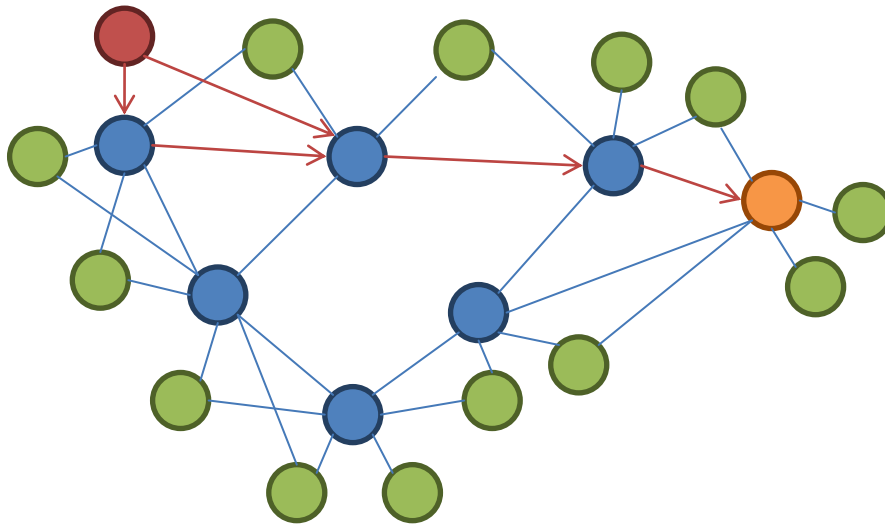
kapcsolatos egyébe megkötéseket alkalmazni (pl. időosztás alapú csomagtovábbítás). Egy ilyen mesh-topológiát mutat a 3. ábra.



3. ábra Mesh-hálózati topológia

A fentebb vázolt mesh-topológiát bizonyos szempontok szerint hatékonyabbá tehetjük oly módon, hogy a hálózat alapjául egy mesh-topológiába rendezett hálózat szolgál, melyek tagjait nevezzük router-eknek. Ezt egészíti ki egy csillaghálózat-szerű végponti node hálózat, melyek kizárólag router-ekhez képesek kapcsolódni, egymáshoz nem, viszont egy node akár több router-hez is kapcsolódhat. Ebben a hálózatban a végponti node-ok redundáns módon képesek kapcsolódni a hálózathoz, és ugyanez igaz a router-ek alkotta alaphálózatra is. Azonban egy ésszerűen összeállított végponti node és routerhálózat felépítéssel hatékonyabb az adattovábbítás, mint egy ugyanennyi egységből álló „tisztán” mesh-hálózat esetén, hiszen a végponti node-oknak nem kell route-olási feladatokat végezniük, csupán a hozzárendelt router-ek felé kell üzenetet küldeni, és a rendelkezésre állási ideje is lehet kevesebb, mint a központi hálózatot alkotó router-eknek. Így a végponti egységek és a router egységek felépítésben is különbözhetnek (pl. a router-ek erősebb tápellátással vannak felszerelve), de már a hálózati szerepek megosztása miatt is nevezhetjük heterogénnek a hálózatot. Ezt a felépítést mutatja a 4. ábra, ahol a piros színnel jelölt végponti node a narancssárga színnel jelölt router node-nak küld üzenetet. Az üzenet először a hozzárendelt kék színnel jelölt router-ekhez érkezik, melyek a route-olási algoritmus szerint kiválasztják az üzenet számára

optimális útvonalat. Ebben a hálózatban már ügyelni kell a több útvonalon is közlekedő csomagokból adódó ütközések elkerülésére, az ilyen üzeneteket a router-eknek ki kell szűrnie.



4. ábra Heterogén node-okból álló mesh-hálózat

3.3. Kommunikációs adatsebesség és hatótávolság

A szenzorhálózat által kezelhető adatmennyiség szempontjából kritikus kérdés, hogy a hálózat milyen adatsebességgel képest információt továbbítani. Egy környezeti paramétereket folyamatosan monitorozó szenzorhálózat esetén a hálózat üzemeltetője nyilván szeretne minél frissebb és pontosabb adatokat kapni a monitorozott mennyiségekről. Ha a node-ok által szolgáltatott adatok azonos fontosságúak, akkor a teljes hálózatra vett adatfrissítési idő az egyes node-ok adatelérési idejeinek az összege, ami a hálózattól következően tág határok között is változhat, főleg, ha a hálózatban terjedő információ továbbítása időosztás alapú, azaz egy node leghamarabb akkor továbbít egy üzenetet, ha hozzá egy csomag már teljes mértékben megérkezett.

Optimális esetben egy üzenetnek a feladó és a címzett közötti legkevesebb node-on kell áthaladni (R_{node_i}), ami pedig a node-ok hálózati elhelyezkedésétől függ. A hálózati paraméterektől függően kell, hogy definiáljunk egy maximális csomag hosszt (n_{packet}), amely meghatározza, hogy egy node mekkora csomagegységet tud kezelni. Ez nyilván adódhat a node-ok üzenet-puffereléséből, vagy egyszerűen a kapcsolati protokoll időzítéseiből. Fontos továbbá a node-ok közötti kommunikációs réteg adatsebessége (f_{baud}), amely alapján megkapjuk a node-ok közötti üzenet terjedés idejének első közelítését:

$$T_{node_message} \approx \frac{n_{message}}{f_{baud}}.$$

Fontos tényező az is, hogy a hálózatban a közös adatátviteli közeg használatából adódó összeütközések elkerülése végett a node-ok kommunikációját össze kell hangolni, ezt általában egy koordinátornak kinevezett node végzi. Alapesetben kommunikációt csak ez a kiemelt egység kezdeményezhet, így oldható meg legegyszerűbb módon az ütközésselkerülés, mivel válaszolni csak a megszólított node fog. Persze a több úton történő üzenetterjedés még további megoldandó problémákat vet fel, ezeket a későbbiekben tárgyaljuk. Lényeg az, hogy azzal, hogy kommunikációt csak koordinátor kezdeményezhet, az üzenetek teljes átviteli ideje a duplájára nő (feltéve, ha minden node üzenetének címzettje a koordinátor).

3.4. Energiakezelés, rendelkezésre állási idő

Vezeték nélküli szenzorhálózatokban a node-ok energiafogyasztásának minimalizálása és az energiatároló (és adott esetben az energia harveszter) kapacitásának maximalizálása fontos kritérium, mivel nagyszámú node esetén az egységek elemcseréje egy nehéz feladat lenne. Környezeti paramétereket monitorozó hálózatban általában megoldható valamilyen harvesztálási módszerrel (pl. napelem) a node-ok folyamatos (vagy újratölthető) energiaellátása, de vannak olyan rendszerek is, ahol erre nincs lehetőség, és ultra-alacsony fogyasztásra kell törekedni a hosszabb élettartam elérése érdekében.

Egy paraméter monitorozó szenzor hálózat egységei alapvetően két energiaigényes műveletet hajtanak végre, az egyik a paraméterek mérése, a másik pedig a kommunikációs műveletek végrehajtása, tehát ezen műveletek optimalizálásával jelentősen csökkenthető az eszköz fogyasztása.

A szenzoros mérési műveleteknél első megközelítésben úgy optimalizálhatunk, ha egy megfelelő felbontást és mintavételi időt választunk a paraméterek mérésére, és a mérések között minimális-energiafogyasztási állapotba helyezzük a vezérlőt.

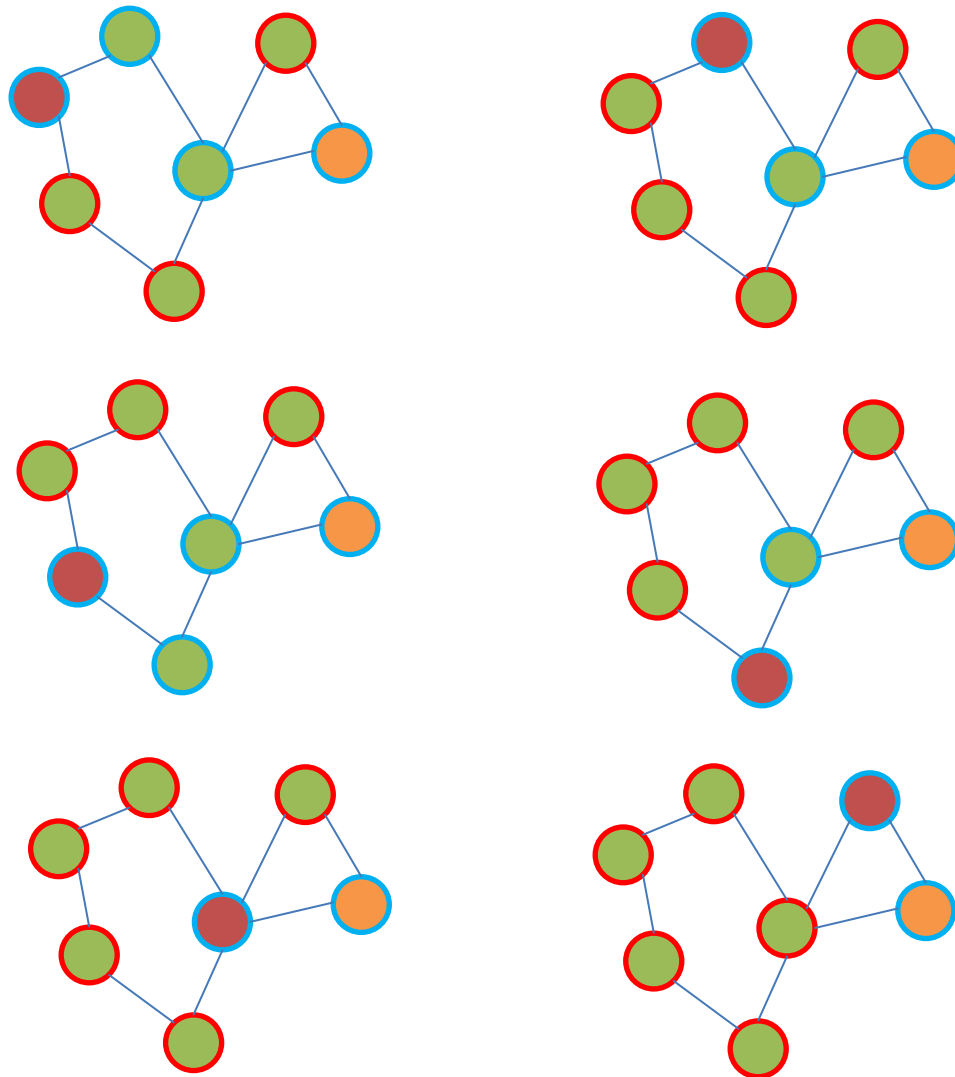
A kommunikációs műveletekből adódó energiafogyasztás minimalizálása már nehezebb feladat, hiszen a hálózat létrehozásakor általában a cél a minél nagyobb területi lefedettség (rádiós hatótávolság növelésével) és az egységek rendelkezésre állási idejének maximalizálása az energiafogyasztás alacsony értéke mellett. Megközelíthetjük úgy a problémát, hogy az egységeket minimális fogyasztású állapotban tartjuk egészen addig, amíg az egység lekérdezése meg nem történik. Két lekérdezés között a node viszont csak annyira legyen tétlen, hogy a méréseket, és a mért értékek mentését el tudja végezni, lekérdezéskor a mentett értékeket küldje ki. De mivel a kommunikáció a hálózat többi egységén keresztül történik, az adatcsatorna kialakításához több egységnek is aktív állapotban kell lennie, hogy

továbbítani tudják a hálózat üzeneteit. Ha a minimális energiafogyasztás a vezeték nélküli hálózati vezérlők kikapcsolásával is jár, a hálózat koordinátora nem tudja aktív állapotba helyezni az adatcsatorna létrehozásához szükséges node-okat. Ezt a problémát megoldhatjuk úgy, hogy a node-ok valamilyen ütemezett időzítés szerint aktív állapotba kerülnek, amíg a hálózat aktív részének a lekérdezése megtörténik. Ehhez a lekérdezési ütemezést előre meg kell határozni, és a node-oknak pontos időzítéssel tudnia kell, hogy mikor kerüljenek aktív állapotba.

Egy ilyen lehetséges lekérdezési folyamatot mutat be az 5. ábra, ahol a narancssárga színnel jelzett node a koordinátor, a piros pedig a lekérdezett node. A node-ok körül látható külső kör kék színe jelzi az aktív node-ot, a piros pedig az inaktív, „alvó” node-ot.

Az ábrán látható, hogy a hálózat lekérdezése során a node-ok aktivitásából következő terhelése nem egyenletes, a hálózatnak van olyan csomópontja, melynek a lekérdezések nagy részében aktív állapotban kell lennie. Viszont az ütemezésből nagyjából meghatározható az ilyen egységekre jutó többletterhelés, és a node-ok tervezésekor fel tudunk készülni a többlet energiaigény kezelésére, például nagyobb kapacitású tápegységgel.

Azonban ennek a lekérdezési módnak nagy hátránya, hogy az ütemezést csak az elkészült, teljes hálózat ismeretében tudjuk meghatározni, és a hálózat bővítése esetén új ütemezést kell az egységeknek ismerniük. Tehát nagy szinkronításra van szükség, mind az ütemezési rend, mind az óra-szinkronítás terén.



5. ábra Szinkronizált lekérdezési folyamat

Az fentebb ismertetett módszerrel lehet javítani például úgy, hogy az aktív/inaktív állapotokat intelligens módon kezeljük. Például az alacsony fogyasztású kommunikációs módban a hálózati interfészt ki- és bekapcsoljuk úgy, hogy bekapcsoláskor a vezérlő belehallgat a hálózatba, és ellenőrzi, hogy várakozik-e szomszédos node felőle üzenet a számára. Ha igen, akkor továbbítja az üzenetet aktív állapotban, ha nem, visszamegy inaktív állapotba. Tehát az inaktív állapotokat bizonyos időközönként megszakítjuk egy teljesen aktív állapottal, a szomszédos node pedig legalább ennyi ideig kell, hogy sugározza az a várakozó adatot (ami adatvesztéshez és kaotikus üzenetterjedéshez vezethet), vagy egy felébresztési jelet („preamble time”, ez a megvalósíthatóbb).

Így az inaktív és az aktív állapot között lesz egy alacsony fogyasztású „félíg éber” állapot, melynek következményeként az adatterjedés nyilván lassabb lesz, viszont a hálózat felépítése nem kötött, és nincs szükség szinkronításra sem.

4. Rendszerterv

4.1. Direktívák és követelmények

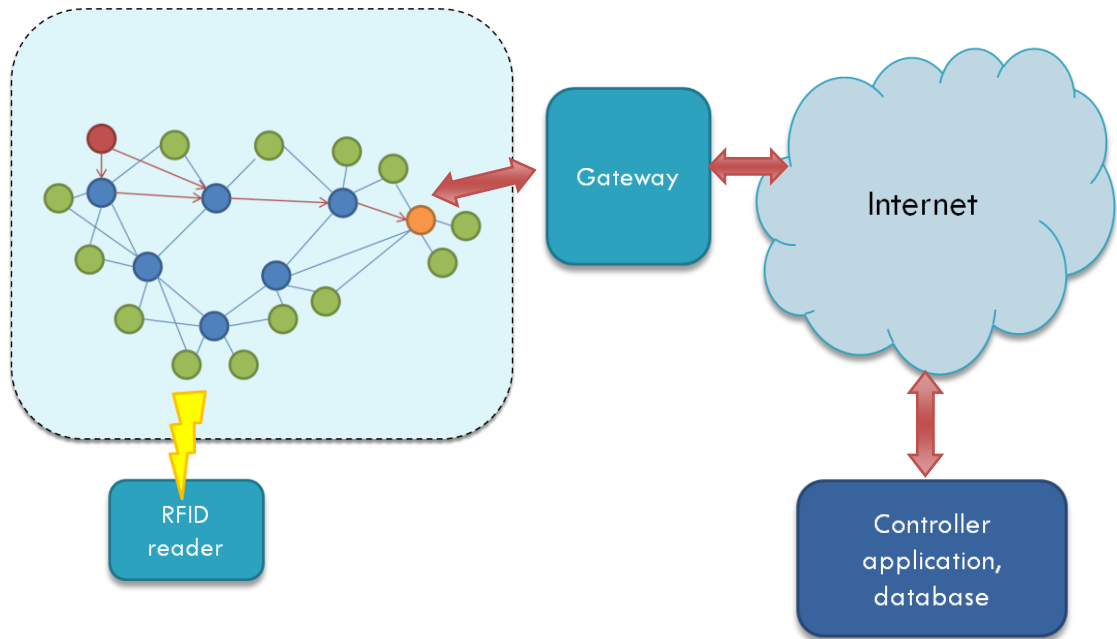
A kutatási projektben a szenzorhálózat működésének direktíváinak és követelményeinek meghatározásakor szem előtt tartottuk a hálózat tervezett alkalmazási helyét (botanikus kert) valamint a későbbi bővíthetőség lehetőségét (például robotokból álló hálózat vezérlőegysége). Ezeket tartalmazza az alábbi felsorolás:

- környezeti paraméterek monitorozása (légnyomás, hőmérséklet, UV és infra fény, páratartalom, talajnedvesség)
- ultra-alacsony fogyasztás, újratölthető energiatároló egységgel, energia-harvesztálási lehetőséggel
- nem real-time rendszer, elég a node-okban tárolt log-ok periodikus lekérdezése
- másodperc körüli mérési időalap
- néhány perces adatmentési időalap
- 16 bites érték-mentési tartomány
- 8 szenzor kezelése
- moduláris felépítés
- szenzorok bővítésének lehetősége
- két irányban biztosított kapcsolati réteg, a hálózati adatvesztés elkerülésére

4.2. Szenzorhálózat felépítése

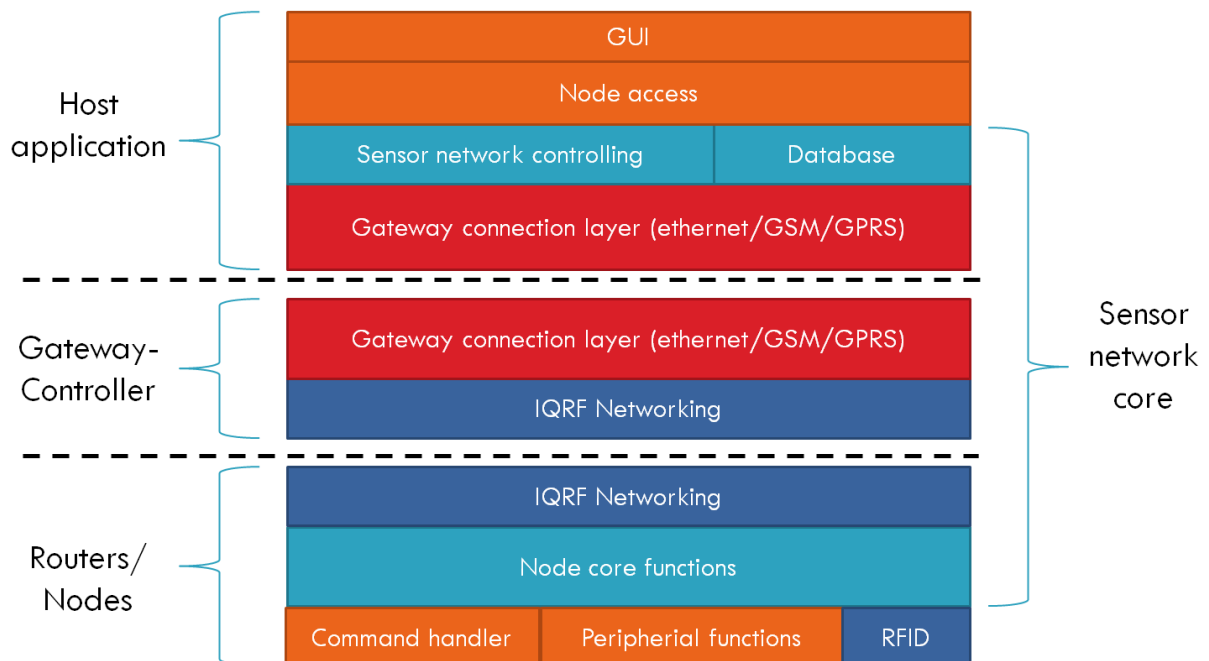
A szenzorhálózat tervezésekor három fő rendszer-elemet határoztunk meg: egy teljes hálózatot vezérlő, a hálózat felett álló vezérlő egységet, praktikusán egy PC-s alkalmazást, egy átjárót, mely a vezérlő egység és a hálózat között áll és egyben ellátja a hálózat koordinátori feladatokat valamint magát a node-okból álló szenzorhálózatot.

Így áll össze a 6. ábrán felvázolt rendszer, ahol a vezérlő és az átjáró közötti kapcsolati protokollt még nem definiáltuk, az átjáró és a hálózat között a kapcsolati protokoll nyilván adott.



6. ábra A teljes rendszer vázlata

4.3. Réteg struktúra



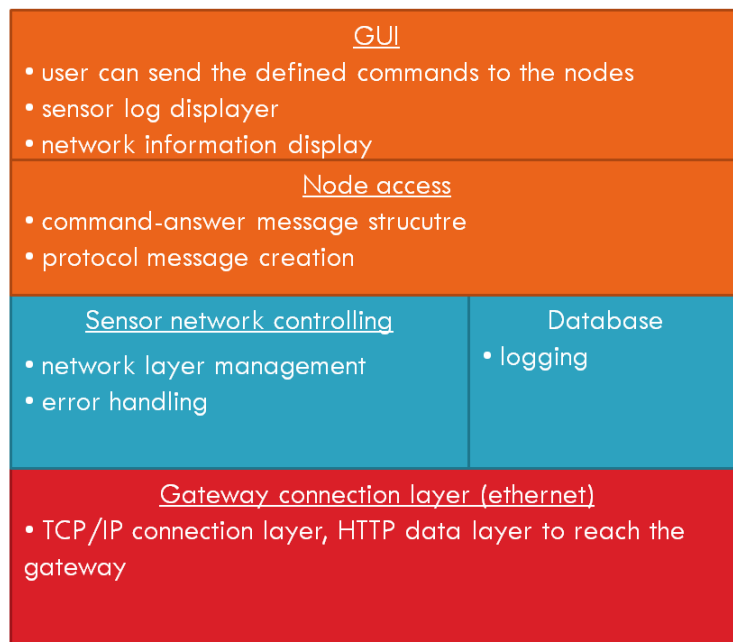
7. ábra A tervezett hálózat réteg felépítése

4.3.1. Szenzorhálózat vezérlő egység

A szenzor hálózat vezérlő egység feladatai:

- a hálózati átjárón keresztül tudjon kommunikálni a hálózat tagjaival
- a hálózati felkonfigurálása
 - node-ok felcsatlakoztatása
 - üzenetek küldése node-onként címezhető módon
- a hálózat felügyelete
 - felcsatlakoztatott node-ok nyilvántartása
 - hálózati térkép készítése
- a node-ok által gyűjtött adatok kezelése

A jelenlegi verzióban a vezérlő egység egy PC-n futó grafikus felhasználói interfésszel rendelkező alkalmazás, ugyanis a fejlesztés során praktikusabb és gyorsabb volt közvetlenül, egy alkalmazásban megvalósítani az alsóbb szintű, hálózat vezérlő rétegeket a grafikus felület funkcióival. A későbbiekben a felhasználói és a hálózat vezérlői funkciók program szinten lesznek elválasztva egymástól, így a hálózat vezérlő akár fizikailag is különválasztható a felhasználói felülettől úgy, hogy az előbbi egy kiszolgáló szerveren fut, amire a felhasználó egy kliens gépen keresztül csatlakozhat.



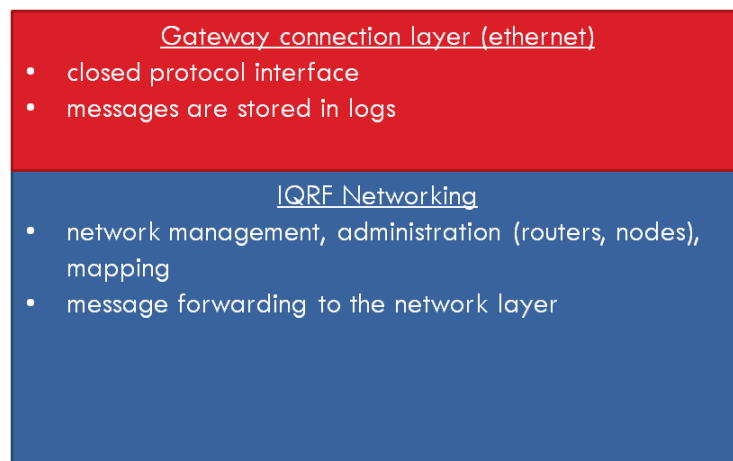
8. ábra A vezérlő alkalmazás blokkvázlata

4.3.2. Szenzorhálózat átjáró és kontroller

A szenzorhálózat átjárójának és kontroller egységének tervezésekor kérdés volt, hogy a két egységet válasszuk-e külön (akár fizikailag, akár logikailag) vagy kezelhetjük őket egy egységben. A kérdést az döntötte el, hogy a választott hálózati kapcsolatot megvalósító modulokhoz használt gateway már tartalmazta a kontrollert is, így célszerűvé vált az egységes kezelése.

Az átjáró és kontroller feladatai:

- közvetlen kommunikáció a vezérlő egységgel (pl. Ethernet interfészen)
- közvetlen kommunikáció a hálózattal (rádiós interfészen)
- két irányban biztosított üzenetküldés biztosítása
- hálózati hibák detektálása



9. ábra A Gateway-Controller blokkvázlata

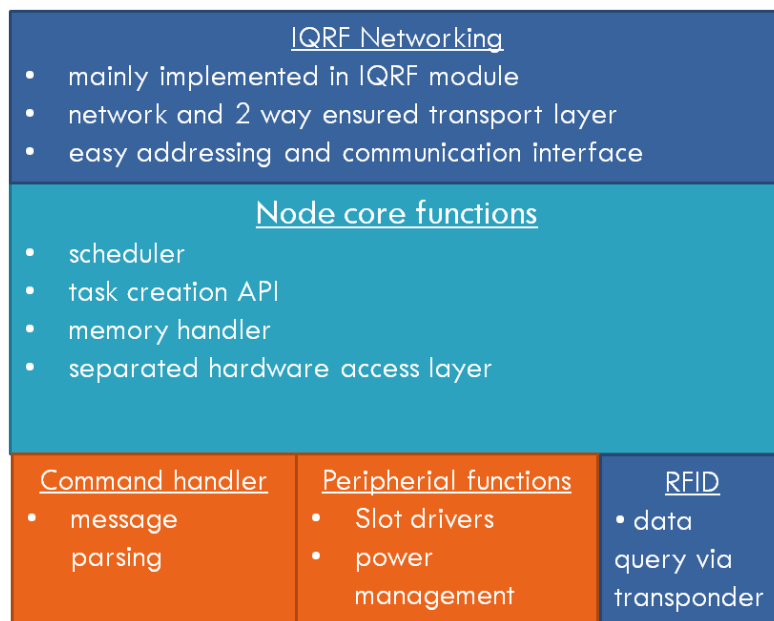


10. ábra Az IQRF Gateway [13]

4.3.3. Szenzorhálózati node

A szenzorhálózati node feladatait már részben megfogalmaztuk a bevezetésben, ezek pontosítva, illetve néhány feladattal bővítve a következők:

- szenzoros mérések végrehajtása konfigurálható mérési időközönként
- szenzoros mérések adatainak mentése konfigurálható logolási időközönként
- univerzális szenzor csatlakoztatási interfészek kezelése
- alacsony energiafogyasztású hálózati kommunikációs kezelése



11. ábra A node egységek blokkvázlata

5. Alaplap

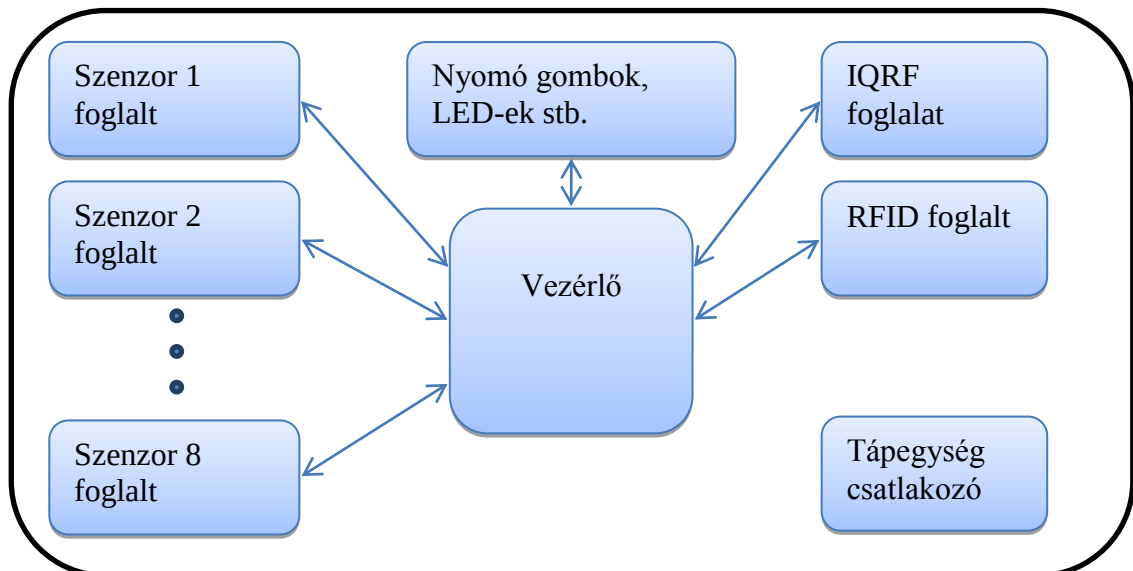
5.1. Szempontok, követelmények

Az alaplapnak a következő követelményeknek kell eleget tenni:

- Csatlakozási pontok biztosítása a tápegység, az IQRF modul, az RFID tag és a szenzorok számára
- A szenzoroknak könnyen cserélhetőnek kell lenni
- Nyolc, univerzális szenzor hely kialakítása
- Tudja fogadni a tápegység esetlegesen stabilizálatlan feszültségét
- Beépített vezérlő (mérésadatgyűjtés és továbbítás IQRF valamint RFID irányba)
- Alacsony fogyasztás, hosszú üzemidő
- A lehető legkisebb méret

5.2. Rendszerterv

Az 12. ábra mutatja az alaplap elvi vázlatát. A központi elem a vezérlő, mely a megfelelő módon kapcsolatban van a modulokkal. Az ábra az áttekinthetőség javítása érdekében nem mutatja, hogy a tápegység csatlakozó minden modullal kapcsolatban van.



12. ábra Az alaplap felépítése

Az RFID csatlakozó a választott RFID tag-nek megfelelő kialakítású. Az IQRF csatlakozó szintén adott, illeszkednie kell az IQRF modulhoz. Azonban az univerzális szenzorcsatlakozók kialakítása komoly tervezést igényelt.

Megvizsgálva a szenzorokat kiderült, hogy a legtöbb vagy SPI buszon vagy I2C buszon vagy analóg módon olvasható. Az elvart, univerzális szenzor helyeket úgy kell megtervezni, hogy bármilyen kommunikációt is használ az adott szenzor, az bármelyik helyen használható legyen. Emellett szükség lehet még egy „Wake up” kivezetésre, mely segítségével felkelthetők/alvó állapotba küldhetők az olyan szenzorok, melyeknek nincs külön parancsuk erre a funkcióra. Emellett egy „Interrupt” jel is szükséges lehet, mely segítségével a szenzorok jelezhetnek a vezérlő felé. Mivel egy szenzor nem használ egyszerre többféle interfészt, össze lehet vonni az Analóg kivezetést az SPI - CS-sel. Ennek értelmében az alábbi kivezetésekre van szükség:

1. Tápfeszültség
2. Föld
3. SPI - SDI
4. SPI - SDO
5. SPI - SCK
6. SPI - CS / Analóg
7. I2C - SDA
8. I2C - SCL
9. Wake up
10. Interrupt

Ezt csatlakozó egyszerűen és költséghatékonyan kialakítható 2x5-ös hüvelysorból.

Felmerült, hogy az ilyen univerzálisan kialakított foglalatokba nem csak szenzorokat lehet helyezni, hanem akár egy SD kártyát, GPS-t vagy GSM modult is. Ezzel a lehetőséggel az eszköz szélesebb körben felhasználható válik, és jobban ki tudja elégíteni a felhasználói igényeket.

A tápcsatlakozónak mindenképpen tartalmaznia kell 2 vezetékét: föld és nem stabilizált pozitív tápfeszültség. Emellett szükség lehet még valamiféle jelzés fogadására a tápegység felől. Előre nem meghatározott, hogy milyen jel fog ide kerülni. Lehet például elemfeszültség visszamérés, vagy akár egy digitális jel is, mi szerint lemerülőben van az elem.

Célszerű lehetőséget biztosítani stabilizált tápfeszültség fogadására is. Így a tápcsatlakozó összesen 4 pólusú.

Továbbá az alaplapnak tartalmaznia kell még bizonyos kiegészítő áramköröket.

- Nyomógombok: Reset és általános célra is
- LED-ek: bár relatív sok áramot fogyasztanak, de hasznos lehet, ha jelezni kell valamit. Ha nincs rájuk szükség akár ki is kapcsolhatók
- Feszültség referencia: Az analóg jelek méréséhez szükséges egy stabil referencia feszültség.
- UART kivezetés: a fejlesztés és hibakeresés során hasznos lehet.

A vezérlőnek tudnia kell megszakításokat fogadni az RFID és az IQRF modultól is. Ugyanis, a vezérlőnek reagálni kell, ha érkezik valamilyen kérés az RFID vagy az IQRF csatornán.

5.3. Vezérlő

Egy egyszerű, olcsó mikrokontroller gond nélkül képes ellátni a szükséges feladatokat. A fogyasztás csökkentés érdekében a mikrokontrollernek, ha épp nem végez semmilyen feladatot, akkor alvó állapotban kell lenni. Így választásnak két fő szempontja van: fogyasztás alvó (sleep) állapotban és hogy milyen források tudják felkelteni a mikrokontrollert. Három gyártó – Atmel, Microchip, NXP – mikrokontrolleri kerültek vizsgálatra. Ezek összehasonlítását az 1. táblázat mutatja.

Név	Sleep áramfelvétel	Mi ébreszti fel?	Megjegyzés
ATMEGA168P [6]	4 uA	WDT, fel vagy lefutó él bármely, I2C címzés	
PIC18FX6K22 [7]	< 1 uA	Bármilyen megszakítás, WDT	
LPC122X [8]	30 uA	WAKE_UP láb	Nem őrzi meg a RAM tartalmát

1. táblázat Vezérlők összehasonlítása

Látható, hogy a fogyasztás és felébresztési lehetősége tekintetében is a PIC18FX6K22 a legjobb. Az ára, számítási teljesítménye és memória mérete is bőve megfelel a feladathoz.

5.4. Tervezés

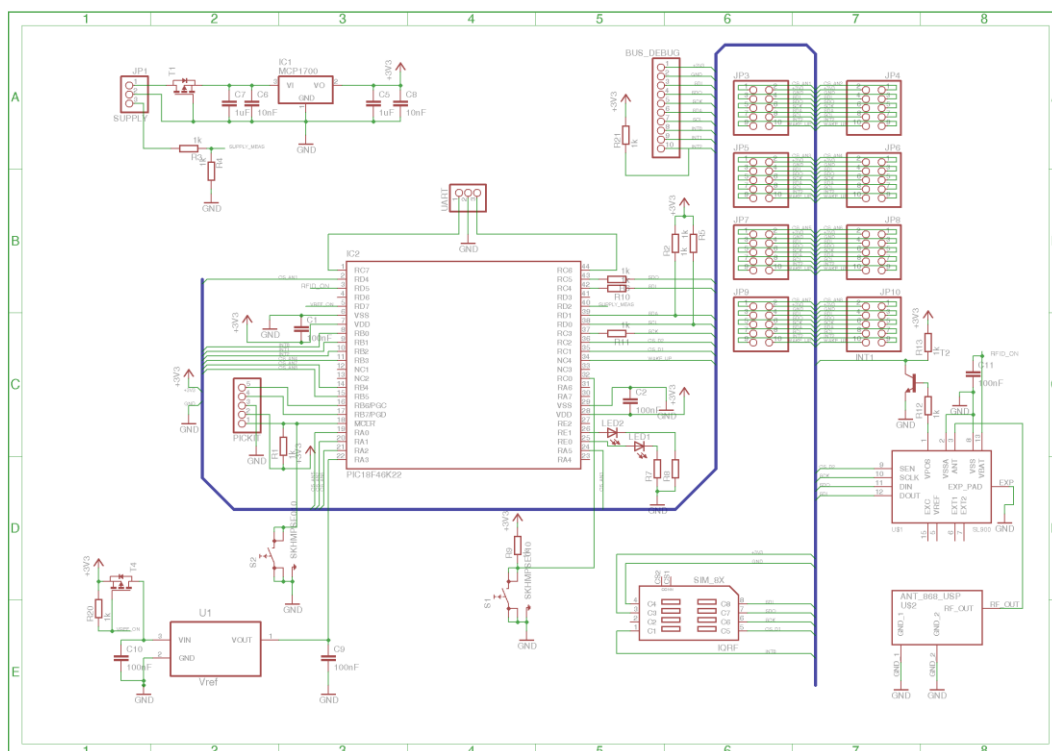
Az áramköri tervek az Eagle Cadsoft programban készültek. A kapcsolási rajzot a 13. ábra mutatja. Középen található a vezérlő, tőle jobbra, a 8 szenzor foglalat, alatta az RFID tag és az IQRF modul. A bal felső sarokban a tápegység, bal alsóban pedig egy feszültség referencia.

Mivel egy RFID tag PCB antennája nagyobb, mint az alaplap várható mérete, ezért inkább egy chip antennát választottunk.

A feszültség referenciára azért van szükség, hogy az Analóg szenzorok mérésekor a PIC belső A/D konverterének stabil referenciája lehessen. Ez 2,5 V-os, így elegendő neki a 3,3 V-os tápfeszültség[10].

A tervekben két nyomógomb és két LED található. Az egyik gomb reset, a másik pedig felhasználói beavatkozást tesz lehetővé.

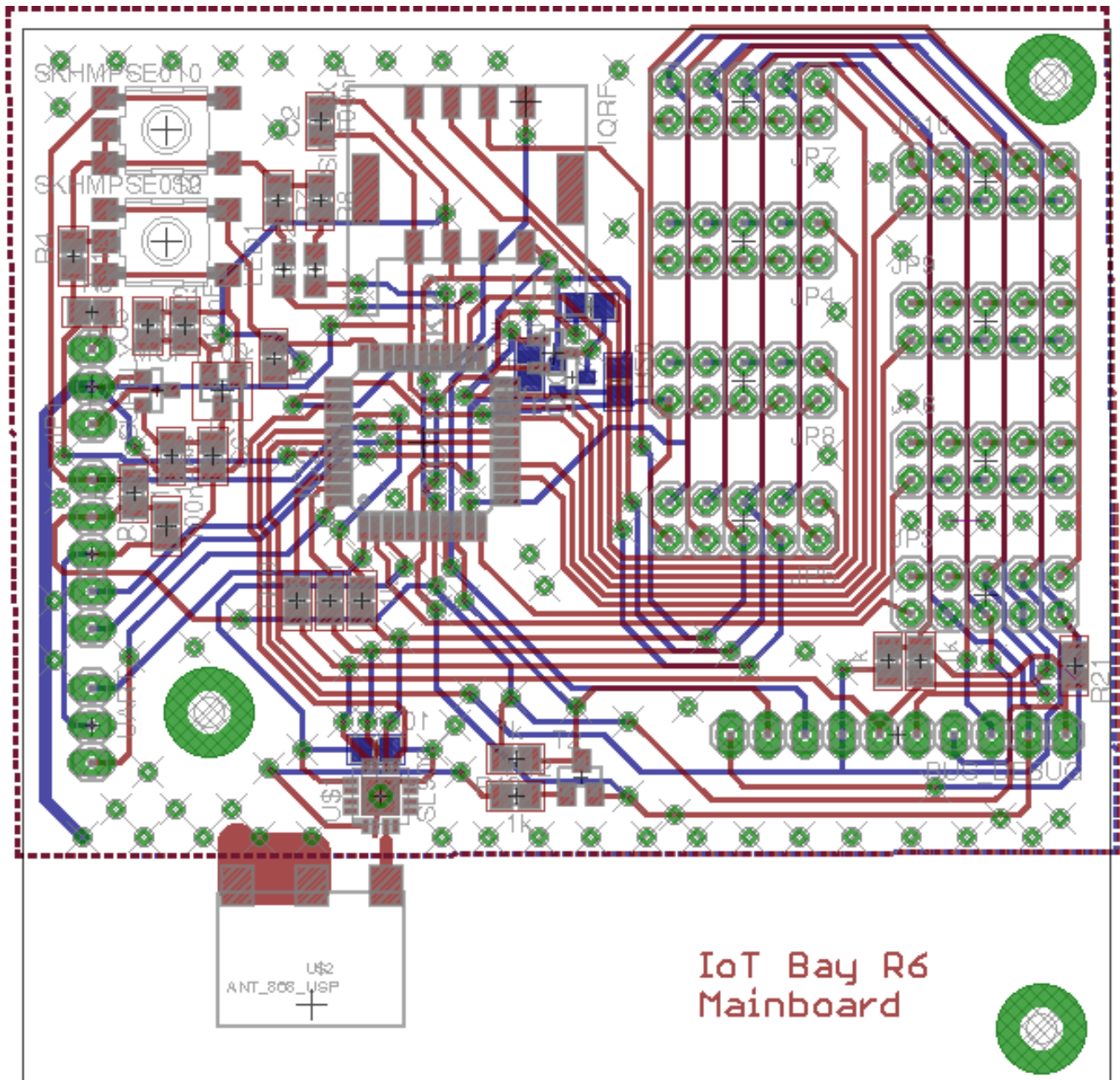
A rajzon látszik még egy BUS_DEBUG csatlakozó. Ennek segítségével könnyen rá lehet csatlakozni a szenzorok kommunikációs interfészeire.



13. ábra Az alaplap kapcsolási rajza

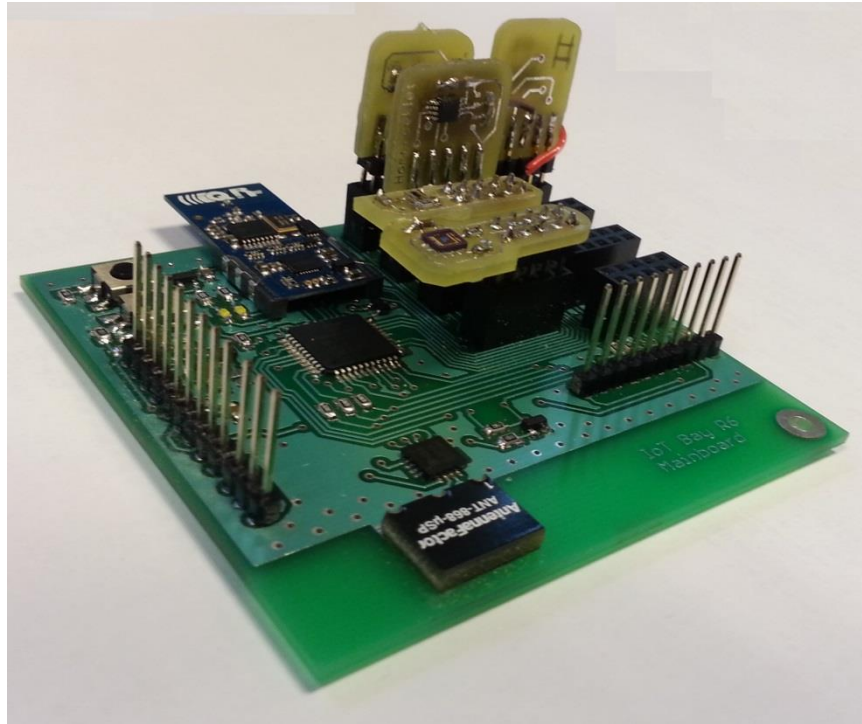
A kapcsolási rajz elkészítési után elkészültek az áramköri tervek is. Ez a 14. ábrán látható. A panel mérete 73 mm x 71 mm lett. Ezt a méretet növeli, hogy az IQRF modul antennájának le kell le kell lógni panelről.

Ez a méret lehetővé teszi akár 60 mm x 60 mm-es napelemek használatát is, melyek akár már 0,5 W-osak is lehetnek. Ez nagyságrendekkel több, mint a teljes hálózat várható fogyasztása (~1 mW).



Moduláris szenzorhálózat tervezése és fejlesztése

Az összeszerelt és működőképes alaplapi panelt a szenzorokkal és az RFID modullal együtt a 15. ábra mutatja.



15. ábra Összeszerelt alaplapi panel szenzorokkal és IQRF modullal

6. Alaplapi firmware

6.1. Áttekintés

6.1.1. Specifikáció

- 8 fizikai és 8 virtuális szenzor slot kezelése
- 8 memória blokk kezelése (FIFO)
- Szenzorok és memóriablokkok összerendelhetőségének biztosítása
- Szenzor mérési idők beállítása 1 sec felbontásban
- Log-olási idők beállítása 15 perc felbontásban
- Mért adatokon a log-olás előtt matematikai műveletek elvégzése (semmi, min., max. és vagy átlag)
- Memóriablokkonként 36 órányi adat tárolása
- Log-olt adatok lekérdezése IQRF-en vagy RFID-n

6.1.2. Felépítés

Az alaplapi firmware egy PIC18F46K22 8 bites mikrokontrolleren fut. A mikrokontroller 3896 bájt adatmemóriával, 64 kBájt program memóriával, 1 kBájt és EEPROM memóriával rendelkezik. Maximálisan 16 MIPS végrehajtásra képes. Nagy előnye, hogy az úgynevezett XLP (eXtra Low Power) család tagja, így sleep módban az áramfelvétele (kikapcsolt perifériákkal) kevesebb, mint 1 uA.

A mikrokontrollerhez csatlakozik az I2C és SPI busz, melyen keresztül a szenzor adatok lekérdezhetők és az IQRF és RFID kommunikáció megvalósítható. Az SPI busz CS kivezetései egyben analóg kivezetések is. Mivel a szenzoroknak vagy analóg vagy digitális interfészük van, nem jelenthet gondot ez az összevonás.

A mikrokontroller a mentett adatokat 2 bájtos előjeles számként tárolja Little endian (kisebb helyi érték a kisebb címen) formában tárolja. Ekkor egy szenzorról 36 óra alatt (a specifikáció alapján) maximálisan 288 bájt keletkezik.

$$\frac{36 \text{ óra} * 60}{15 \text{ perc}} * 2 \text{ bájt} = 288 \text{ bájt}$$

1. egyenlet 36 óra alatt egy szenzorról maximálisan keletkező adat mérete

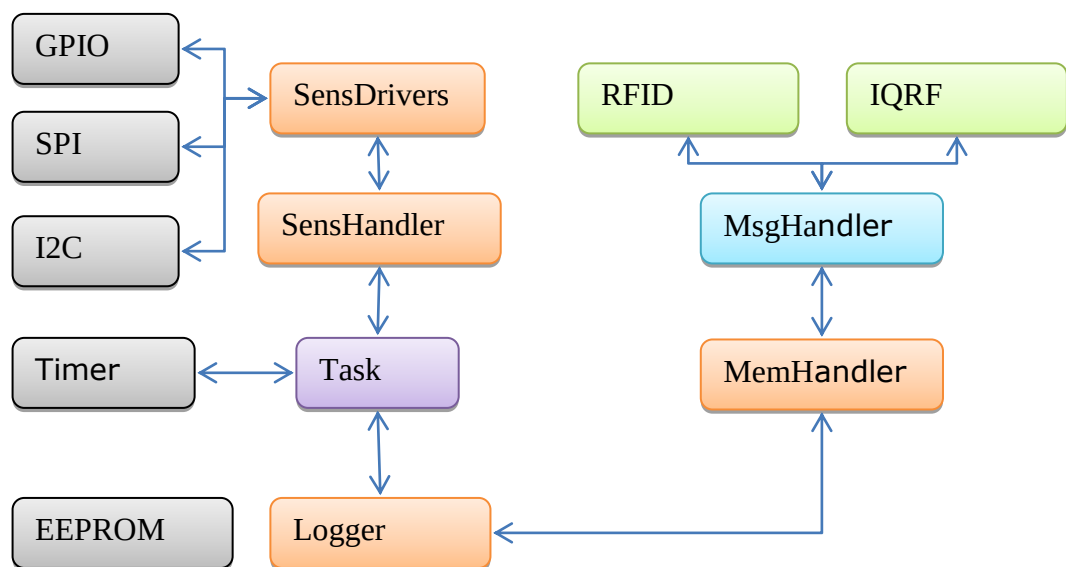
Ennek értelmében 8 memória helyen 2034 bájt keletkezik. Így közel 2 kB adatmemória marad a többi változó és egyéb adatok számára.

$$288 \text{ bájt} * 8 = 2304 \text{ bájt}$$

2. egyenlet 36 óra alatt maximálisan keletkező szenzor adat mérete

6.2. Blokkvázlat

Az alaplapi firmware blokkvázlatát a 16. ábra mutatja.



16. ábra Az alaplapi firmware blokk vázlata

A **GPIO**, **SPI**, **I2C**, **EEPROM** és **Timer** blokkok az alacsony szintű driverek. Ezekben a részekben található a mikrokontroller specifikus részek kezelése. A mikrokontroller cserje esetén csupán ezeket a fájlokat kell lecserélni, mivel általuk a hardver teljesen elrejtésre kerül.

A **SensDriver** blokkban történik a szenzorok által mért adatok lekérdezése az alacsony szintű driver-ek segítségével. Minden szenzorhoz tartozik egy Init, egy Read, és egy Info függvény. Az Init függvény inicializálja az adott szenzort, a Read függvény kiolvassa a mért adat, az Info függvény pedig a szenzorra jellemző információkat szolgáltat, ha léteznek (ilyenek például gyári kalibrációs adatok)

A **SensHandler** blokkban történik a szenzorok lekérdezésének vezérlése. Ebből a blokkból történik a SensDriver-ben található Read függvények hívása a megfelelő mérési

időnként, és a kapott értékek megfelelő kezelése. Ez a blokk biztosít egy interfészt is a memória blokk és szenzor hely beállításokhoz és lekérdezésekhez.

A **Logger** blokk a megfelelő log-olási időnként a SensHandler által gyűjtött adatokat beírja a memóriába. Biztosítja megfelelő függvényeket a memóriába történő íráshoz és olvasáshoz, bár a memóriakezelést nem ez a blokk végzi. Ezzel elfedi a tényleges memóriakezelést, melyet így rugalmasan változtatható. Például könnyen lehetne akár SD kártyára is menteni az adatmemória helyett.

A **MemHandler** blokkban történik a memóriák kezelése, az írási és olvasási műveletek. Itt történik a mentéskori műveletvégzés is (például átlagolás). Ez blokk biztosítja meg a memória blokkok FIFO jellegét.

A **MsgHandler** blokk egy általános interfészt jelent melyen keresztül üzeneteket küldhetnek az egyes blokkok. A MsgHandler képes megkülönböztetni bizonyos parancskódokat és az azoknak megfelelő kiértékelő függvényt meghívni.

Az **IQRF** blokkban történik az IQRF kommunikáció vezérlése. Az IQRF-en vett adatokat némi formázás után az MsgHandler-nek továbbítja, mely elvégzi az adatok megfelelő kiértékelését és választ szolgáltat.

Az **RFID** blokkban történik az RFID kommunikáció vezérlése. Az RFID-n olvasott adatokat némi formázás után az MsgHandler-nek továbbítja, mely elvégzi az adatok megfelelő kiértékelését és választ szolgáltat.

A **Task** blokk egy egyszerű ütemezőt valósít meg. Képes a beállított függvényeket periodikusan meghívni. A szenzorokat olvasó és log-oló feladatok, mind külön egy-egy taszkot jelentenek. Így 8 fizikai és 8 virtuális szenzorhelynek 16 taszkja van, míg a 8 memória helynek megfelelően 8 log-oló taszk fut. Az egyes taszkok hívási idejének állításával valósítható meg a mérési idők és log-olási idők változtatása.

6.3. Működési mechanizmus

A szenzorokat a kialakított 8 slot valamelyikébe lehet elhelyezni az alaplapi panelen.

Léteznek olyan szenzorok, melyek egyszerre több paramétert is képesek mérni:

- A látható fény szenzor infra fényt
- A páratartalom szenzor hőmérsékletet
- A nyomás szenzor hőmérsékletet

Ahhoz, hogy ezek másodlagos mérési eredményei is elérhetőek legyenek, azokat úgynevezett virtuális slot-okba kell helyezni. Ezek nem fizikai slot-ok, csupán a működtető

programnak kell tudni, hogy a másodlagos szenzorok is léteznek és hogy melyik szenzornak melyik a másodlagos szenzora.

Egy virtuális slot-ot úgy lehet elérni, hogy fizikai slot azonosítójához (0 .. 7) hozzá kell adni 8-at. Így a virtuális szenzor helyek 8-tól 15-ig tartanak.

Minden slot-hoz tartozik egy mérési időköz, melynek felbontása 1 másodperc. Az adott slot-ban ennyi időnként történik mérés a hozzárendelt szenzor segítségével. A megfelelő paranccsal az az idő módosítható. Alapbeállításban minden szenzor mérési időköze 1 másodperc. Ha ez az idő 0 másodperc, akkor az adott slot-on nem történik mérés, így az kikapcsolható.

A programban található 8 memória blokk, melyekbe egyenként 144 mérési adat tárolható. A memória blokkokat össze kell rendelni a slot-okkal. Például a 3. memória blokkba a 4. slot mérési kerüljenek. Virtuális slot-okat is lehet rendelni a memória blokkokhoz, például a 6. memória blokkhoz a 12-es slot-ot.

Minden memória blokkokhoz tartozik úgynevezett log-olási időköz, melynek felbontása 15 perc. A mért eredmények ennyi időnként kerülnek mentésre a megfelelő memóriarekeszbe.

A FIFO-ba mentés előtt az előző mentés óta mért eredményeken 4 féle művelet elvégzésére van lehetőség: semmi (ekkor az utolsó mért érték kerül mentésre), minimumkeresés, maximumkeresés és átlagszámítás.

Mivel a műveletvégzés a memória blokkokhoz kapcsolódik az is megvalósítható, hogy két memória blokk azonos slot-hoz van rendelve, de az egyik minimumot a másik pedig átlagot tárol.

6.4. Parancsok

Az alaplappal előre definiált formájú parancsokon keresztül lehet kommunikálni:

Parancsazonosító	Paraméterek
1 bájt	0 .. 10 bájt

2. táblázat Parancs felépítés

Az alaplappal az ismert parancsra választ fog adni. A válasz formátuma függ az adott parancstól, de az biztos, hogy az első bájtja az a parancskód, melyre válaszol.

Az alaplappal soha nem kezdeményez kommunikációt, mindig csak válaszol.

6.5. RFID kommunikáció

Az RFID tag-gel történő kommunikáció a tag User memory-ján keresztül történik. A User memory logikai felosztását a 3. táblázat mutatja.

Név	Tartomány
Közös memória terület	0x0000 ... 0x0001
Olvasói memória terület	0x0002 ... 0x000A
Node memória terület	0x000B ... 0x04C1

3. táblázat Tag User memory logikai felosztása

Egy parancs beviteléhez az olvasónak a Közös memória területre be kell írni a 0xCC kódot. Ha mikrokontroller azt kódot olvassa ezen a címen, akkor tudja, hogy parancs került a teg-be.

Az olvasói memóriába be kell írni a Parancsok fejezetben található leírásnak megfelelően összeállított parancsot.

A node válasza a Node memória területre fog kerülni pár másodperccel később. Ezzel együtt a Közös memóriában található 0xCC kódot 0x00-ra állítja, ezzel jelezvén az olvasó felé, hogy a parancs sikeresen végrehajtásra került.

Egy üzenetváltás lépésről lépésre:

1. Üzenet összeállítása. (egy lehetséges összeállítás pl: [0xCC 0x10 0x01] log lekérdezés az 1. memória blokkból)
2. A tag User memory-jának elejére az összeállított parancs üzenetet írása
3. Legalább 2 másodperc várakozás, amíg a node észleli az RFID transzponderbe írt üzenetet, és végrehajtja a parancsot
4. Preamble byte visszaolvasása a Közös memória területéről. Ha 0x00, akkor parancs végrehajtva, ha 0xCC maradt, akkor várni kell még pár másodpercet.
5. Node memória terület olvasása (a parancs alapján elegendő a válasz várható hosszának megfelelő mennyiségű bájt olvasása)

7. IQRF vezérlő program

7.1. IQRF áttekintése

Az IQRF egy jól használható, egyszerű vezeték nélküli MESH hálózatot megvalósító protokoll tipikusan sok Node-ot tartalmazó hálózatokhoz.

7.1.1. Az IQRF protokoll paramétere

- Frekvencia tartomány: ISM 433MHz, 868 MHz és 916 MHz
- Beépített operációs rendszert tartalmaz
- C nyelven programozható
- Csomag orientált kommunikáció (64 bájt / csomag)
- Néhány 100 m adó távolság és legfeljebb 240 hop / csomag
- Legfeljebb 65.000 node
- Alacsony fogyasztás: 380 nA készenléti állapotban és 25 uA vétel közben
- 19,2 kb/s adatátviteli sebesség
- Ingyenesen használható a protokoll
- Könnyen implementálható

7.1.2. Az IQRF protokoll

Az IQ MESH-be kapcsolt eszközök 2 félé szerepet tölthetnek be[12]:

1. Controller (csak egy lehet belőle és nem változtatható, hogy melyik eszköz)
2. Node

A MESH-en az üzenetek eszközről eszközre kerülnek továbbítása, így olyan eszközök is megkapják az üzeneteket, akik a küldő hatótávolságán kívül esnek. Azonban nem minden Node képes részt venni a csomagok továbbításában. Ahhoz, hogy egy Node ilyen funkciót meg tudjon valósítani Router-nek kell konfigurálni. Amennyiben minden Router elérhető legalább egy másik Router vagy a Controller számára, akkor a küldött üzenetek el tudnak jutni minden Router-hez és a Controller-hez is. A nem Router Node-ok csak akkor kapják meg az üzeneteket, ha egy Router vagy a Controller hatótávolságán belül vannak.

Mivel egy ilyen bonyolult topológiában az üzenetek több úton is eljutnak egy feladótól a címzettig, ezért definiálni kell egy routolási szabályt, mely megadja, hogy milyen logika szerint haladjanak a csomagok a hálón. A routolás történhet az eszközök felcsatlakoztatása során definiált címek alapján, ekkor az SFM (Static Full Mesh) routing alkalmazható vagy a Controller által kiosztott logikai címek (VRN – Virtual Routing Number) alapján, ekkor a DFM (Dynamic Full Mesh) algoritmus használható. Célszerűbb a DFM használata, mert SFM esetén az eszközök elhelyezése az azonosítójuk függvényében akár a kommunikációból való kizáródást is eredményezheti. Mivel DFM esetén a címek az adott elrendezésben kerülnek kiosztásra, ez a probléma nem merül fel. A logikai címek kiosztása a Discovery folyamat során történik, melyet a Controller indít el. Mivel a VRN-eknek csak a routolásban van szerepe, ezért csak a Router-ek kapnak ilyen címet. Az üzenet továbbítása során ezek a címek veszik át a felhasználó által adott címek szerepét. A VRN-t az OS kezeli, így a felhasználónak nem kell foglalkoznia vele, a címezés a felhasználó által adott címek alapján történik. A Controller-hez közeli eszközök kapják a kicsi VRN-eket. A kommunikáció során meghatározó szerepe van a VRN-nek, mert a küldés sorrendje azoknak megfelelően történik. A kommunikáció időosztásos elven működik. Minden Router a saját időszakban adhat, így elkerülve az ütközéseket.

Üzenetküldés előtt be kell állítani, hogy az üzenet maximum hány Router-en haladhat át, azaz hány hop-ot tehet meg. Ha ez a szám pl. 2, akkor 3 üzenet kerül elküldésre (küldő + 2 Router). Ezen kívül be kell állítani egy time slot hosszát is. (lásd később)

Controller által küldött üzenet

Pl. 5 hop és 100 ms time slot esetén, a Controller 0 ms-nél elküldi az üzenetet, amit több Router is vehet. Az ütközések elkerülése érdekében minden Router csak a saját VRN-jének megfelelő időszeltnél adhat. Ennek megfelelően a példában a következő időszakok, 100 ms-tól 200 ms-ig az 1-es VRN-ű eszközé, 200..300 ms-ig a 2-dik eszközé, és így tovább, míg az 5 hop meg nem történik. A küldés sorrendje a VRN-ek szerinti növekvő sorrend. Ha a címzett egy Router és VRN-je 8, akkor jelen a példában ő soha nem fogja megkapni az üzenetet, mert az üzenet előbb hal el, mint eljutna hozzá (feltételezve, hogy nem vette az üzenetet korábban, pl a 3-mas VRN-ű eszköztől).

Node által küldött üzenet

A Node által küldött üzenetek továbbítása nagyon hasonló a Controller -> Node kommunikációhoz. Az egyetlen különbség, hogy az első időszület a küldő Router-é és az utolsó a Controller-é. Más szóval a sorrend a VRN-ek szerinti csökkentő sorrend.

A közeghozzáférésre nincs konkrét protokoll az IQRF szabványban. Annak érdekében, hogy mégsem történjenek ütközések be kell tartani azt a szabályt, hogy csak a Controller kezdeményezhet kommunikációt. A Controller üzenetet küldhet egy Node-nak, amire az válaszolhat. Így nincs lehetőség arra, hogy egy Node kérés nélkül adni kezdjen. Ha mégis ilyen történne, akkor annak kezelése a felhasználó feladata.

7.2. IQRF továbbfejlesztés

7.2.1. Szállítási réteg

Az IQRF az OSI modell alsó 3 rétegét valósítja meg. Definiálja a fizikai réteget (jelszintek, frekvencia sávok stb.), az adatkapcsolati réteget (egy keret formátumát) és a hálózati réteget (a címezést és routolást). Azonban szállítási réteg feladatait nem képes elvégezni a protokoll, így ad garanciát arra, hogy csomagok valójában célba is érjenek. Sőt visszajelzést sem képes adni csomagvesztés esetén.

Ennek kiküszöbölése érdekében olyan mechanizmust kellett kidolgozni mind a Controller mind pedig a Node oldaláról, mely valamilyen módon képes visszajelzést adni csomagvesztés esetén. A megoldást egy olyan megállapodás jelentette, miszerint a Controller üzenetére a Node-nak mindenképpen válaszolni kell. Ez válasz lehet nyugtázás is (például csak 1 karakter) vagy lehet akár egy teljes válasz egy lekérdezésre. A Controller az általa küldött üzenet függvényében eldöntheti, hogy kiértékeli-e a válaszban kapott adatokat vagy megelégszik azzal, hogy jött valami. A rugalmas és praktikus használhatóság érdekében az alábbi paraméterek megadása szükséges egy üzenet elküldéséhez:

- Üzenet és annak hossza
- Cél
- Időkorlát
- Újrapróbálkozások száma

A küldést végző függvény a Cél Node-nak elküldi az Üzenetet majd vár a válasza az Időkorlátban megadott ideig. Ha nem érezik válasz, akkor újraküldés történik maximálisan az Újrapróbálkozások számaként megadott alkalommal. A függvény visszatérési értéke a maradék Próbálkozások száma. Tehát, ha 0 értékkel tér vissza, akkor nem érkezett válasz az üzenetre.

7.2.2. STD és XLP közti váltás

Egy másik megoldandó probléma abból fakadt, hogy a rendszerszintű üzeneteket csak STD módban lehet küldeni, míg az általános felhasználói üzeneteket XLP módban lenne célszerű küldeni a fogyasztás minimalizálása érdekében. Ezeket a feladatokat általánosan, a felhasználó számára rejtve lenne érdemes kezelni. A megvalósítandó funkciók:

- Start up: általános indulási folyamatok és lépések
- Bond node: Node felcsatlakoztatása a hálózathoz a Controlleren keresztül
- Prebond node: Node felcsatlakoztatása a hálózathoz másik Node segítségével
- Sleep Node: adott Node XLP-be küldése
- Wake up Node: adott Node STD-be küldése
- SPI handler: SPI kezeléss

A felsorolt funkciók sikeresen implementálásra kerültek mind a Controller mind pedig a Node oldaláról.

7.2.3. Új API

A számos új funkció jobb átláthatósága érdekében célszerű ezeket valamilyen egységes formában egy API ként kezelni. A cél az volt, hogy a lehető legtöbb rendszerszintű funkció kerüljön elrejtésre és csak néhány függvényt kelljen feltölteni a specifikus adatokkal. Ez meg is valósult.

A Controller-ben a Start up függvény után már nem kell mást tenni, mint üzeneteket küldeni és elemezni válaszukat. Minden más automatikusan megtörténik. Az új függvények a következők:

- IQRFSStartUp
- IQRFSend
- IQRFSleepNode

- IQRFWakeUpNode
- IQRFAAddNode

A Node programja szintén a Start up függvénnyel kezdődik. Utána egy végtelen ciklusban kell hívni az adat fogadó függvényt. A felhasználó feladata, hogy egy általános előre definiált függvényben megírja a kapott üzenetek elemzését és elindítsa válaszküldést. A rendszer szintű tevékenységek így teljesen láthatatlanok a felhasználó számára. Az új függvények a következők:

- IQRFSStartUp
- IQRFRReceive
- UserReceive (ebbe írhatja a felhasználó a kapott üzenetek értelmezését)

7.3. Megvalósítás

A megvalósítás során a DFM algoritmust egy módosított változata került alkalmazásra, a DFM2B. A „2B” a 2 bájtos címekre utal, ugyanis ebben routolásban a Node-ok címei 1 és 65000 között lehetnek. (A Routers száma továbbra is maximálisan 240 lehet.)

A végleges szenzor hálózati alkalmazásban a Controller az ethernet gateway-ben található. A felhasználói PC-s programból etherneton kapott üzeneteket a gateway SPI-on továbbítja a Controllernek. A lehetséges parancsok listája a Parancsok fejezetben leírtakhoz képest kiegészül néhány olyan parancssal, melyek csak az IQRF-re specifikusak, így ezek parancsok nem jutnak el más Node-ok alaplapi firmware-éig. Az új parancsokat a mutatja.

Az üzenetek felépítése annyiban módosul, hogy a Controller-nek küldött üzenetek a cél Node címével kezdődnek (2 bájt). A 0. cím jelenti a kontrollert.

A megvalósított lekérdező mechanizmus lépései a Controller szempontjából a következők:

1. A Controller kap egy SPI üzenetet az ethernet vezérlőtől
 - a) Ha az üzenet a Controllernek szól, akkor feldogozza
 - b) Ha nem a Controllernek szól, akkor továbbítja az üzenetet a megfelelő címre.

2. Mielőtt a Controller válaszolna SPI-on az ethernet Controller-nek elküld egy 0xFE-vel kezdő kód után elküldi, hogy melyik címről fog érkezni a válasz. Ez segíti a további kiértékelést. Például: 0xFE 0x04 0x01, a 260-as című Node válasza.
3. Az elküldött parancs után a válasszal kapcsolatban két eset lehetséges:
 - a) A Node válaszolt: a Controller a válaszul kapott üzenetet továbbítja az ethernet kontroller felé
4. A Node nem válaszolt: a nincs választ jelző üzenetet továbbítja a Controller az ethernet kontroller felé.

A megvalósított lekérdezi mechanizmus lépései a Controller szempontjából a következők:

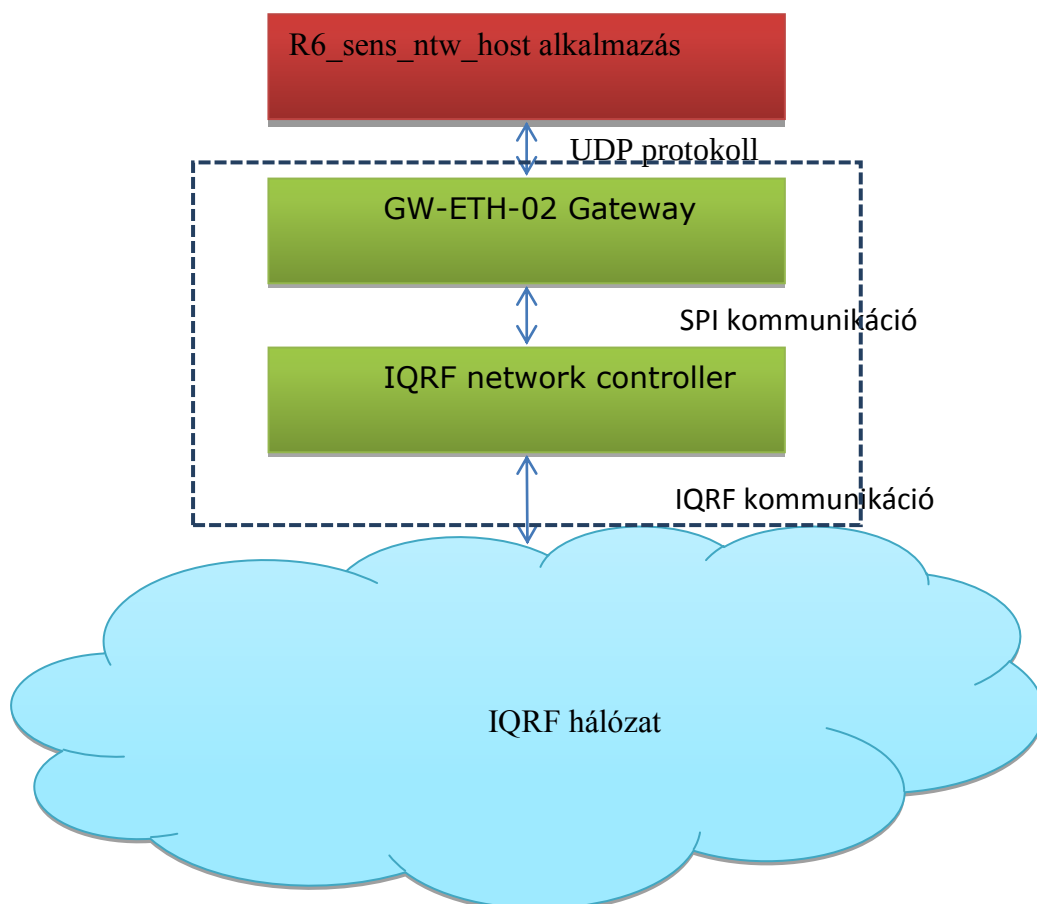
1. A Node kap egy üzenetet IQRF-en.
2. a) Ha az üzenet IQRF specifikus, akkor feldogozza és válaszol
b) Ha általános üzenetről van szó, akkor SPI-on elküldi az üzenetet az alaplap mikrokontrollerének.
3. Ha SPI-on érkezett válasz az alaplapi mikrokontrollertől, akkor azt IQRF-en továbbítja a Controller-nek.

8. Vezérlő PC-s alkalmazás

8.1. Áttekintő

A szenzorhálózat konfigurálásához, működtetéséhez és az adatok megjelenítésére létrehoztunk egy grafikus felhasználói interfésszel rendelkező alkalmazást, mely . Az alkalmazás az IQRF alapú szenzorhálózathoz IQRF GW-ETH-02 típusú ethernet gateway-en keresztül kapcsolódik a hálózatot vezérlő Controller egységhez (ez fizikailag a gateway-ben található), melyen keresztül a hálózat összes többi egysége elérhető. Az alkalmazásra R6_sens_ntw_host néven hivatkozunk, mivel a kutatási projektben így lett elnevezve.

A hálózat több kommunikációs csatorna együttműködésével jön létre, ennek felépítését mutatja a 17. ábra.



17. ábra A hálózat kapcsolati felépítés-hierarchiája

Az alkalmazásban grafikus felületén keresztül a szenzorhálózat fejlesztésekor definiált parancsokat küldhetünk a hálózat node egységeinek (itt a node egységen az IQRF kapcsolaton keresztül elérhető vezérlő egység jelenti, az ábrán látható IQRF node-ok, mint a kommunikációs kapcsolat egységei szerepelnek), és láthatjuk az egyes parancsokra érkező válaszüzeneteket is.

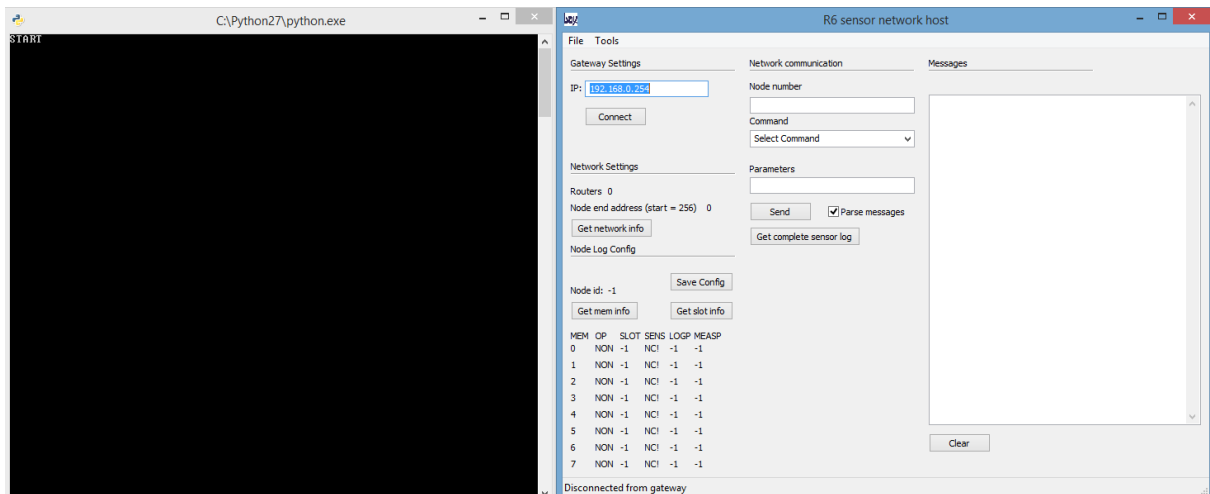
Ezenkívül a node-októl beérkező üzenetek automatikusan mentésre kerülnek, ezek megtalálhatók a programkönyvtár `node_logs` mappájában. A logok fájlneve a node-ok sorszámai alapján kerülnek megadásra, pl. a `node_260.txt` fájl a 260-as azonosítójú node üzeneteit fogja tartalmazni.

8.1.1. Az alkalmazás futtatásához szükséges követelmények

- telepített Python 2.7 környezet/programcsomag (ajánlott 32 bites verzió)
- telepített wxPython programcsomag (ajánlott 32 bites verzió)
- az alkalmazás keresztplatformot támogató eszközökkel készült, de eddig csak Windows operációs rendszeren volt tesztelve
- helyesen konfigurált ethernet kapcsolat a Gateway-jel, azaz a futtató PC legyen egy alhálózatban a Gateway-jel

8.1.2. Az alkalmazás futtatása

Amennyiben a 8.1.1-ben foglalt követelmények teljesültek, a programot az `r6_sens_ntw_host/src` mappában található `app.py` fájl python környezetben történő futtatásával tudjuk elindítani, indítás után két ablak jelenik meg, az egyik az alkalmazás interfész ablaka, a másik pedig egy konzol kimeneti ablak, ezt mutatja a 18. ábra.



18. ábra Az elindított alkalmazás, konzol kimeneti ablakkal

8.2. Az alkalmazás felhasználói áttekintése

8.2.1. Csatlakozása az IQRF gateway-hez

A hálózatra csatlakoztatott, működő IQRF gateway-hez az IP címének ismeretében tudunk csatlakozni, de ehhez előtte a futtató PC-n is szükséges a megfelelő hálózati konfiguráció.

Ha a gateway IP címe helyesen be lett konfigurálva, írjuk be a címét az IP jelzésű beviteli mezőbe, majd kattintsunk a Connect feliratú gombra. Ha minden megfelelően megtörtént, az ablak állapotsorában megjelenik a Connected to gateway felirat.

Hiba esetén ellenőrizzük a gateway csatlakozását, működését valamint a hálózati beállításokat, és próbáljuk újra a csatlakozást.

A hálózatba a csatlakoztatott gateway-en keresztül küldhetünk üzeneteket.

8.2.2. Válaszüzenet konzol értelmezése

Hálózati kommunikáció esetén minden, a hálózathoz érkező válaszüzenet megjelenik a Messages szekcióban található kimeneti konzolon.

A kiírt üzenet paraméterei:

- Node ID: az üzenetet küldő node egység azonosító száma
- MSG: az üzenet gateway log száma (0-65535 értékű lehet)
- TIME: az üzenet gateway log-ba érkezési ideje

- adatmező: 64 byte, az IQRF kommunikáció maximális méretű alapegysége alapján van meghatározva a mérete

8.2.3. Hálózati alapadatok lekérdezése

A hálózati alapadatok, mint például az IQRF hálózatban található router-ek és node-ok száma lekérdezhető manuálisan beállított paranccsal, vagy a Get network info feliratú gombra kattintva.

Ha a lekérdezés teljesül, a Network Settings szekcióban a Routers felirat után olvasható a routerek, a Nodes felirat után pedig a node-ok száma.

8.2.4. Parancsok küldése

A parancsok a hálózat olyan üzenetei, melyeket a vezérlő alkalmazás küldhet a node-oknak, egy végrehajtandó utasítást és azok paramétereit tartalmazza. A felcsatlakoztatott node-ok minden parancsra válaszolnak, ezek lesznek a válaszüzenetek, melyeket a konzolon és a log-okban olvashatunk.

A parancsküldés módja:

- írjuk be a node címét (0. cím a kontroller, 1-től 255-ig router, 256-tól 65536-ig egyszerű node), melynek a parancsot küldjük
- válasszuk ki a command lenyíló menüből a küldendő parancsot
- adjuk meg az üzenet paramétereit a Parameters beviteli mezőben. A paraméterek megadásánál 1 vagy 2 byte méretű paramétereket adhatunk meg vesszővel vagy szóközzel elválasztva, egyszerű decimális, vagy C nyelvben ismeretes hexadecimális formában (pl. 0x55)

8.2.5. Válaszüzenet log-ok felépítése

A válaszüzenet log-okat a programkönyvtár node_logs mappájában találhatjuk. A log fájlok az alkalmazás futtatása során keletkeznek, illetve módosulnak a válaszüzenetek beérkezésekor.

Egy adott node-hoz tartozó log egy ASCII karaktereket tartalmazó szövegfájl, tehát megnyitható valamilyen szövegmegjelenítő alkalmazással. A node válaszüzeneteit soronként láthatjuk hexadecimálisan ábrázolva (a beérkezési idő kivételével, mivel az decimális), és szóközzel vannak elválasztva az üzenet különböző mezői. Az első mező az üzenet gateway log sorszáma (4 karakter, 2 byte hexadecimális), a második mező az üzenet gateway-be

érkezési ideje (12 karakter, nap,hónap,év,óra,perc,másodperc decimális ábrázolásban), a harmadik mező pedig maga az üzenet, mely 64 byte-ot tartalmaz hexadecimálisan ábrázolva.

Egy ilyen sor a következő módon néz ki:

[illegible]

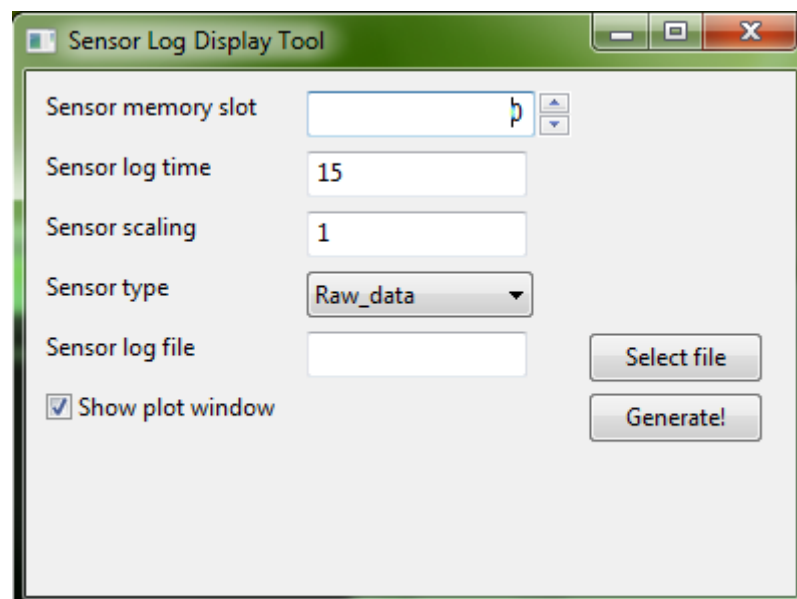
Itt az üzenet sorszáma hexadecimálisan 0x0023, dátuma 2014.09.24.09:39:11, adatmezője pedig a 0x10 értékű byte-tal induló karaktersorozat.

8.2.6. Adatok megjelenítése

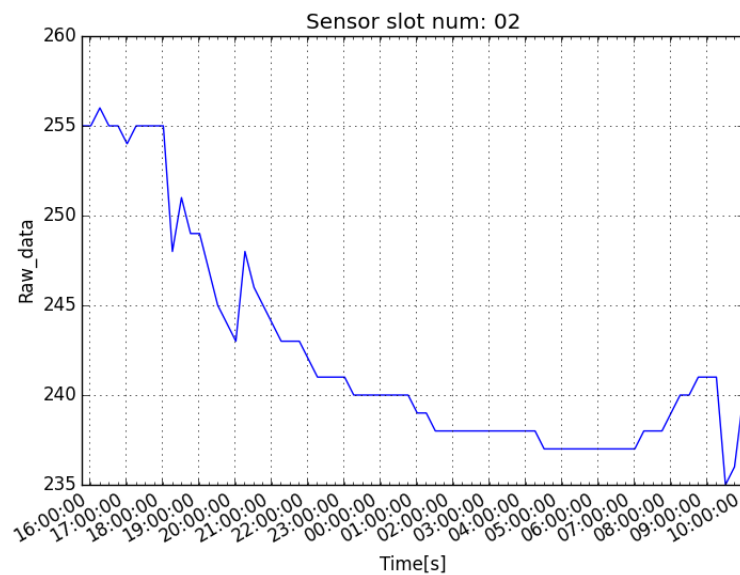
A szenzorhálózat által gyűjtött adatok megjelenítését a vezérlő alkalmazás Tools/Sensor Log Display menüből indítható alkalmazással tehetjük meg.

Az alkalmazásban ki kell választani a programkönyvtár `node_logs` mappájában az adott azonosítójú node-tól érkező log-okat tartalmazó log-fájlt, meg kell adni a megjeleníteni kívánt memória slot azonosítóját, a szenzor log-olási idejét és skálázását, valamint a szenzor típusát (opcionális).

A Generate! gombra kattintva a `node_logs/plots` mappába mentésre kerül a generált diagram, ami a Show plot window checkbox kipipálása esetén egy külön ablakban megjelenik.



19. ábra Az adatmegjelenítő alkalmazás ablaka

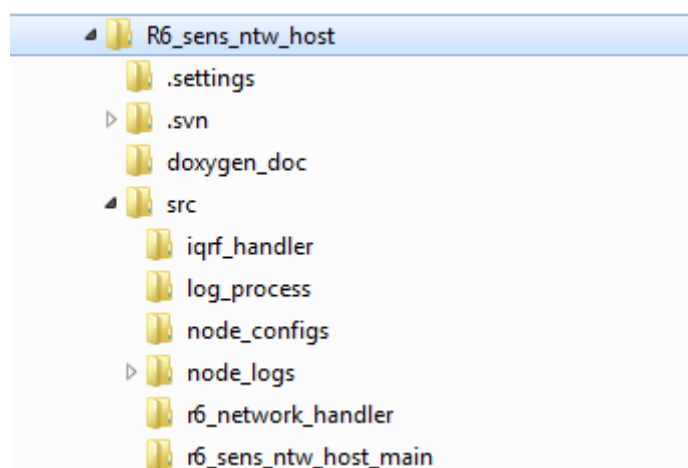


20. ábra A megjelenítő alkalmazás által generált diagram

8.3. Az alkalmazás fejlesztői áttekintése

Az alkalmazás fejlesztése Python 2.7 programozási nyelven történt, a felhasználói interfész pedig Wx Widgets keretrendszer Python könyvtárainak (wxpython) felhasználásával lett kialakítva.

A forráskönyvtár szerkezetét a 21. ábra mutatja.



21. ábra A forráskönyvtár szerkezete

Az src forráskód mappa tartalma a következő (a nem használt fájlok kihagyásával):

- **app.py**: alkalmazás indító fájl

- **iqr_handler**: iqr hálózat kezelő driver modulok
 - **iqr_gateway_handler.py**: iqr gateway driver modul
 - **iqr_network_handler.py**: iqr network driver modul
- **r6_sens_ntw_host_main**: felhasználó felület modulok
 - **main_win.py**: felhasználói felület elemeit tartalmazó modul
- **log_process**: az adatmegjelenítéssel kapcsolatos modulokat tartalmazza
 - **sens_log_plot.py**: grafikon generáló modul
 - **sens_log_window.py**: a grafikon generáló modul grafikus interfésze
- **node_logs**: az elő-feldolgozott node üzenet log-okat, valamint a generált grafikonokat tároló könyvtár

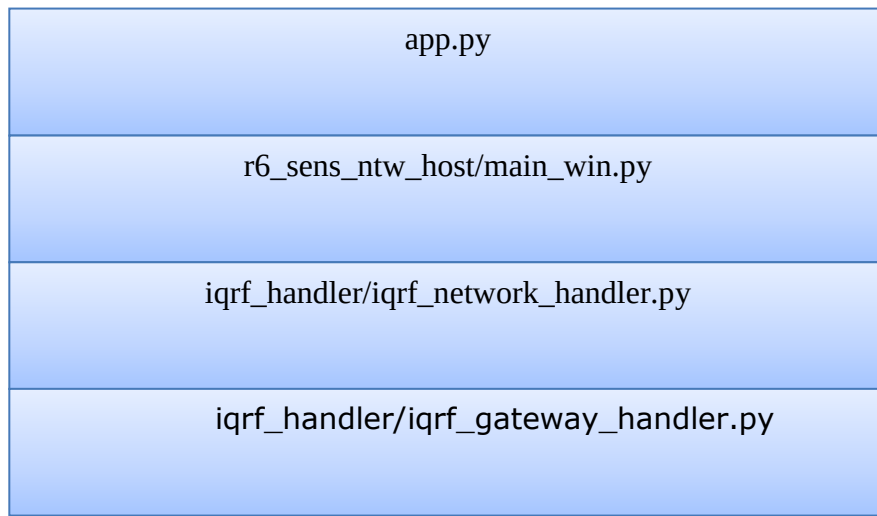
8.3.1. Az alkalmazás modulok hierarchiája

A 22. ábra az alkalmazás modul hierarchiáját ábrázolja egymásra épülés alapján. Az app.py modul indítja el az alkalmazást, a main_win.py modulban található grafikus interfész osztályok példányosításával.

A main_win.py modulban már megtalálhatók az alkalmazás grafikus és logikai rétegének elemei, melyek az iqr_network_handler.py modulon keresztül kapcsolódnak a szenzorhálózathoz.

Az iqr_network_handler.py modul a hálózat magasabb szintű hozzáférését biztosítja, mint például az egyedi (adatkapcsolati) kommunikációs protokoll ismeretével történő üzenetküldés és üzenetelérés (üzenetet queue-ból) illetve az üzenetek log-olása. A hálózat alsóbb rétegeihez az iqr_gateway_handler.py modulon keresztül fér hozzá.

Az iqr_gateway_handler.py biztosítja a hozzáférést az IQR gateway eszközben található HTTP szerverhez LAN kapcsolaton keresztül. A modulban található iqr_gateway osztály példányosításával és tagfüggvényeinek használatával tudjuk elérni az IQR gateway különböző funkcióit.



22. ábra Modul hierarchia vázlat

9. További lehetőségek és célok

További fejlesztésekkel a rendszer szélesebb körben felhasználhatóvá válna. Leginkább olyan dolgok merültek fel továbbfejlesztési lehetőségként, melyek jól beleilleszkednek a mostani rendszerbe, így megvalósításuk nem igényel nagy volumenű változtatásokat, illetve új vagy ismeretlen technológiák, megoldások alkalmazását.

9.1. Adat tömörítés

Jelenleg az alaplapon található mikrokontroller memóriája szolgál a mért adatok átmeneti tárolására. 8 x 36 órás log tárolásánál többre a jelenlegi konfigurációban nem lenne lehetőség. Azonban a hardver változtatása nélkül is elérhető bizonyos kapacitásnövekedés, ha az adatokat tömörítve tárolja az alaplapi mikrokontroller. Tekintve, hogy a szenzorhálózatok esetében tipikusan lassan változó jelekről van szó a különbség alapú letárolás célra vezető lehet.

Elképzeltető egy olyan kódolás, mely az adatok 2 bájtban ábrázolja, a változást pedig 1 bájtban. Ha az aktuális adat olyan nagy mértékben tér el az előzőtől, hogy nem ábrázolható 1 bájtban az eltérés, akkor egy új 2 bájtban, abszolút adatot kerül letárolásra. (Így a méretcsökkenés legjobb esetben is kevesebb 50 % lehet.) Legalább egy bitet fel kell használni annak jelzésére, hogy az adat 2 bájtban abszolút vagy 1 relatív érték. Emiatt az ábrázolható tartomány -16384-től +16383-ig tart, a változás tartománya pedig -64-től +63-ig.

Abból kiindulva, hogy például minden 10. adatot kell majd abszolút módon 2 bájtban tárolni kiszámítható, hogy mennyire nő tárolható adatok mennyisége. Ez mutatja a 3. egyenlet.

$$\frac{\text{tömörített méret}}{\text{eredeti méret}} = \frac{1 * 2 \text{ bájt} + (10 - 1) * 1 \text{ bájt}}{10 * 2 \text{ bájt}} = \frac{11 \text{ bájt}}{20 \text{ bájt}} = 55 \%$$

3. egyenlet Adat tömörítés hatásának becslése

Ezzel az eredeti 36 órás log-olási kapacitás 65 órára növelhető.

9.2. Mentés memória kártyára

Ha még több memóriára lenne szükség, mert valamilyen az adatok nem kérdezhetők le néhány naponként, akkor lehetőség van az egyik szenzor slot-ba memória kártyát helyezni.

Tekintve, hogy a memóriakártyákat lehet SPI-on is kezelni ez viszonylag könnyen megvalósítható lenne, csupán egy FAT fájlrendszer kezelő réteg kell hozzá.

Egy 4 GBájtos memóriakártyára több ezer év mérési adat elférne, ha 8 darab 2 bájtos adatot 15 percenként ment rá a Node.

$$\frac{4Gbajt}{\frac{60\ perc}{15\ perc} * 8 * 2\ bajt} \approx 67\ millió\ óra \approx 7660\ év$$

4. egyenlet Eltárolható log hossza 4 GBájtos memória kártyán

9.3. Teljesen univerzális alaplapp

A szenzorokhoz jelenleg SPI és I2C busz is ki van vezetve. Így a szenzorokon túl akár az IQRF és az RFID (vagy bármilyen egyéb kommunikációs eszköz) is a slot-okba helyezhető lehet. Ezzel az alaplapp olyannyira leegyszerűsödik, hogy csak mikrokontroller, a slot-ok és a kiegészítő áramkörök kapnak rajta helyet. Ez egy nagy lépést jelentene az univerzális felhasználhatóság és a modularitás terén.

9.4. Dobozolás

Ahhoz, hogy a szenzor Node-ok a gyakorlatban is használhatóak legyenek természetesen szükség van megfelelő készülékházra. A doboznak időjárás állónak kell lenni. az elektronika védelme mellett meg kell valósítani, hogy a mért hőmérséklet, páratartalom vagy fény a készüléken kívüli értékeknek feleljenek meg.

9.5. Általánosított API

Némi módosítással és további általánosítások bevezetésével a magasabb szintű részeknek egy olyan API, illetve rendszerterv készítése válna lehetségessé, mely segítségével tetszőleges szenzorhálózat tervezése, konfigurálása és menedzselése is megoldható.

9.6. Két részre bontott vezérlő alkalmazás

A PC-s vezérlő alkalmazás tervezése során nagy hangsúly került arra, hogy az egyes szoftver rétegek hierarchikusan és egymástól szeparáltan kerüljenek implementálásra. Ezt kihasználva megvalósítható lenne a grafikus kezelői felület és a szoftver magját képező vezérlő alkalmazás teljes szétbontása.

A mag részt egy szerveren futtatva a grafikus felülettel web socket-eken keresztül képes kommunikálni. Ezzel a mag és grafikus felület web alapokra helyezhető. Ez magas fokú platform függetlenséget biztosít az egész vezérlő alkalmazás számára, mivel a grafikus felület lehet egy tetszőleges operációs rendszerrel rendelkező PC-n vagy laptopon, de akár okos telefonon is.

10. Összefoglalás

Az elkészített tervek alapján megvalósításra került egy teljes szenzorhálózat elektronikájának hardvere és hozzá kapcsolódó firmware és szoftver programok. Az elkészült eszközök kielégítik a specifikációban támasztott követelményeket. Ahhoz, hogy a rendszer a gyakorlatban is alkalmazható legyen még mindenképpen szükséges egy megfelelő készülékház létrehozása.

Mind a hardver, mind pedig a szoftver tervezés során nagy hangsúlyt fektettünk a modularitás biztosítására. Ez hardver szinten az univerzális szenzor slot-okat, míg szoftver (és firmware) szinten a jól definiált interfészekkel ellátott szoftver modulokat jelenti. Ennek révén a az egyes alkotóelemek könnyen testre szabhatók egy speciális feladat esetén. Nem jelent problémát például az IQRF kommunikáció lecserélése valamely másik protokollra.

Összességében a létrehozott rendszer stabilan, megbízhatóan az elvártnak megfelelő paraméterekkel üzemel. Ez lehetővé teszi, hogy a jövőben sikerrel alkalmazásra kerüljön, így segítve a tudományok más területén dolgozó kutatók munkáját.

11. Irodalom jegyzék

- [1] ISO18000-6: Information technology —Radio frequency identification for item management — Part 6: Parameters for air interface communications at 860 MHz to 960 MHz
- [2] TR-52D IQRF modul
- [3] TPS60502 Step-down konverter adatlap
- [4] MCP1700 feszültség stabilizátor adatlap
- [5] TR-52D IQRF modul
- [6] ATmega128P mikrovezérlő adatlap
- [7] PIC18FX6K22 mikrovezérlő adatlap
- [8] LPC122X mikrovezérlő adatlap
- [9] SL900A RFID tag adatlap
- [10] AD1582 feszültség referencia adatlap
- [11] ANT-868-uSP chip antenna adatlap
- [12] www.iqrf.org
- [13] <http://www.iqrf.org/weben/index.php?sekce=products&id=gw-eth-02&ot=gateways&ot2=gw-eth-02>