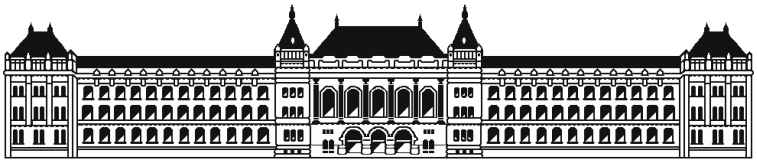


BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKA KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Hierarchikus adatgyűjtő-vezérlő hálózati rendszer otthoni alkalmazásokhoz

TDK DOLGOZAT

Készítette
Németh Péter

Konzulens
dr. Tóth Csaba

2012. október 23.

Tartalomjegyzék

I. Bevezetés

A technológia fejlődésének köszönhetően a különféle elektronikus eszközök egyre kisebb méretekkel és egyre komplexebb belső felépítéssel rendelkeznek. Ennek hatására az áruk is egyre kedvezőbben alakul, ami jelentős mértékben hozzájárult az olyan elosztott felépítéssel rendelkező rendszerek térnyeréséhez, mint például a napjainkban egyre szélesebb körben alkalmazott szenzorhálózatok. A TDK dolgozatomban a beágyazott rendszerek ezen jellegzetes és számos új kihívást tartogató csoportjával foglalkozom.

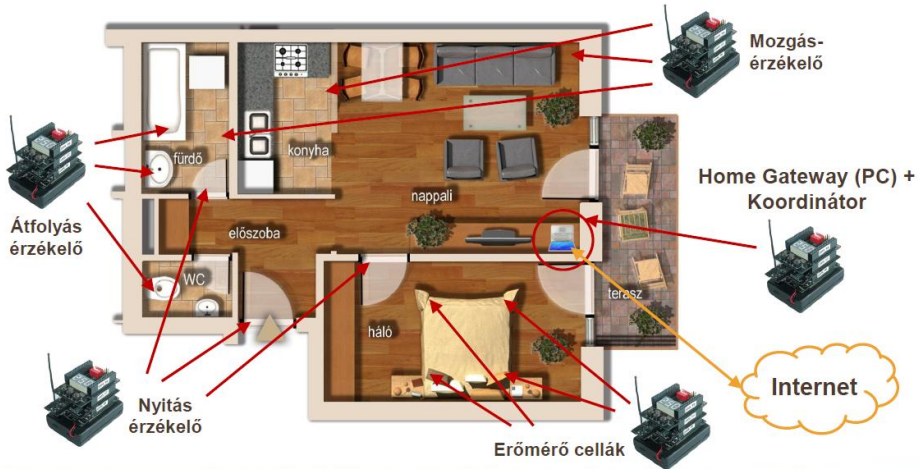
A dolgozatomban bemutatom egy otthoni környezetekbe szánt adatgyűjtő-vezérlő szenzorhálózat rendszertervének elkészítését, hardver- és szoftver-moduljainak megvalósítási lépéseit. A munkámat irodalomkutatással kezdem, amiben többek között az elosztott érzékelés és az intelligens porszem fogalom bemutatására, valamint azok fejlődésének vizsgálatára helyezem a hangsúlyt. A rövid történelmi áttekintést követően, mely rávilágít arra, hogy milyen innovatív ötletek megvalósítását tette máris lehetővé a szenzorhálózatok rohamos fejlődése, bemutatom a saját rendszerem előzetes specifikációját. Ebben elsősorban arra szeretnék rámutatni, hogy milyen kiaknázatlan lehetőségeket is rejt a már meglévő koncepciók előnyös tulajdonságainak egyesítése. Ezt a felvázolt előzetes rendszertervben összegzem, amely tartalmazza, hogy milyen egységekből fog felépülni a megvalósítandó rendszer. Az egységek által használt interfésztípus alapján két alternatívát különböztethetünk meg: vezetékes és vezeték nélküli szenzorhálózatok. Az ezek részletes bemutatásával foglalkozó fejezetben - a megbízhatóságot tekintve elsődleges szempontként - az autóiparban használt vezetékes buszok és az önszerveződés, valamint az önjavítás képességével rendelkező vezeték nélküli hálózati protokollok legelterjedtebb fajtáinak technikai paramétereit elemzem. Egy optimális kombináció kiválasztását követően a dolgozatom további fejezeteiben a hardver- és szoftverfejlesztés témaköreivel foglalkozom. Ismertetem a rendszer végleges hardver rendszertervét és mivel beágyazott rendszerekben a hardver és a hardverközeli szoftver szorosan összekapcsolódik, még az áramkörtervezés megkezdése előtt felvázolom a rendszer szoftvertervét. A hardvertervezés bemutatása a mikrovezérlők alkalmazás-technikájához kapcsolódik szorosan, míg a szoftverfejlesztéssel foglalkozó rész a beágyazott operációs rendszerek témakörére koncentrál. Mivel egy szenzorhálózatnál

kiemelten fontos a begyűjtött adatok a külvilág számára elérhetővé tétele, az utolsó fejezetben egy web alapú felhasználói felület kialakítása is bemutatásra kerül.

A dolgozatomban arra szeretnék rámutatni, hogy egy feladat megoldása során nem az jelenti a nehézséget, hogy a rendelkezésre álló megoldásokat felhasználjuk, hanem, hogy azok közül sikerüljön az optimálisat kiválasztani.

1.1. Előzmények

Az egyetemen részt vettem a BME Egészségipari Mérnöki Tudásközpont által vezetett CCE (Connected Care for Elderly Person) [1] project munkáiban. Ez egy nemzetközi kezdeményezés, melynek elsődleges célja egy az időskorúak életvitelét megkönnyítő nyílt és szabványos európai platform kifejlesztése. A rendszer fő koncepciója, hogy különböző érzékelők segítségével folyamatosan adatokat gyűjtünk a környezetről, majd ezen megfigyelések felhasználásával következtetéseket vonunk le a bekövetkező eseményekről. Ezen információk segítségével riasztásokat küldhetünk a megfigyelt környezet nem várt megváltozásáról vagy esetleg különböző beavatkozók segítségével autonóm módon reagálhatunk ezekre a problémát valószínűsítő váratlan eseményekre. A környezet megfigyeléséről különböző érzékelők gondoskodnak, melyek rádiós szenzorhálózatokba vannak szervezve a könnyű installáció lehetőségének biztosítása érdekében. A szenzoroktól érkező adatok az ún. Home Gateway-ben kerülnek rögzítésre és feldolgozásra.



1. ábra: A CCE rendszer koncepciója MITMÓT[1] alapú szenzorokkal megvalósítva

[1] *A Méréstechnika és Információs Rendszerek (BME-MIT) tanszék moduláris mikrovezérlő oktatási eszköze [2]*

Az én feladatom a tanszéken megvalósított mintarendszer Device Manager szoftvermoduljának implementálása volt. Ez a CCE rendszer egyik alsó szoftverrétegének fontos komponense, melynek feladata a hozzá tartozó szenzorhálózat menedzselése, valamint az érzékelők és a magas szintű szoftverkomponensek közötti információáramlás biztosítása. Ez a feladat kiváló alkalmat biztosított arra, hogy megismerkedjek egy szenzorhálózat által szolgáltatott információkat feldolgozó egység felépítésével. Az itt szerzett tapasztalatok nagy segítségemre lesznek a mostani munkám keretein belül elkészülő rendszer felső szoftverrétegeinek megtervezése során.

A TDK dolgozatomban ismertetett munkám a CCE project tanszéki folytatásának, továbbgondolásának tekinthető. Az alapötlete a következő: tervezzünk egy olyan hierarchikus adatgyűjtő-vezérlő hálózati rendszert, amely ötvözi a vezetékes és vezeték nélküli hálózatok előnyeit.

II. Szenzorhálózatok

2.1. Fejlődési tendenciák

A XXI. század első évtizedének egyik húzóágazata az elektronikai technológia területén az érzékeléstechnika volt. Mára a legkülönbözőbb fajtájú és rendeltetésű érzékelők tevékenykednek a mindennapi élet számos területén. Ezen szenzorok hálózatokba szervezésével kezdődött meg az érzékelésközpontú rendszerek térhódítása. Megjelentek a lakókra, külső körülményekre reagáló intelligens otthonok koncepciói, a különféle környezeti paramétereket monitorozó rendszerek és ma már az ipari gyártósorokon is szenzorvezérelt robotokat állítanak munkarendbe. Az informatikában megfigyelhető trend, miszerint a koncentrált rendszerek helyett egyre gyakoribbak az elosztottak, a beágyazott rendszerek területén is egyre fontosabb koncepciót képvisel. A szenzorhálózat-technológia ugyan még csak gyerekcipőben jár, hiszen egyenlőre „csak" néhány száz érzékelő állítható rendszerbe, melyek jelentősen limitált számítási kapacitással és szerény kommunikációs képességgel rendelkeznek, ugyanakkor ezek kiváló alapját képezik a számítástechnika következő hullámának. Ez a „csendes", környezetünkbe olvadó, annak szerves részét képező technológia kora lesz. A háttérben dolgozó, tárgyakba integrált láthatatlan számítógépek, a környezet-intelligencia megjelenését fogják jelenteni.

2. ábra: A szenzorhálózatok várható fejlődése [3]

2.2. Szenzorhálózatok alapjai

Szenzorhálózatoknak nevezzük a nagy számú intelligens érzékelő egységekből felépített, autonóm működésre képes elosztott hálózatokat, amelyek a tér különböző pontjain megfigyeléseket és esetleg beavatkozó tevékenységeket is

végezhetnek. Az alapvető adatgyűjtő funkción kívül az érzékelő hálózat képes adatfeldolgozás és analízis jellegű feladatok elvégzésére is. A korlátolt sávszélességű hálózati kapcsolatok általában szükségessé teszik az adatok már magán az érzékelők szintjén történő minél magasabb szintű feldolgozást. Az elosztott működés oka a legtöbb esetben az, hogy a fizikai rendszer, melyet mérni vagy befolyásolni szeretnénk, térbeli kiterjedése nem teszi lehetővé, hogy egyetlen központi eszközzel valósítsuk meg a feladatot. Az alacsony árnak és a kis méretnek köszönhetően a szenzorok a megfigyelt térrészben viszonylag sűrűn és nagy számban helyezkedhetnek el.

Ezen hálózatok alapját apró egységek, úgynevezett mote-ok képezik. Minden egyes mote tartalmaz legalább egy szenzort vagy aktuátort, egy relatív kis számítási teljesítményű feldolgozó- és a többi mote-tal folytatott információcseréhez szükséges kommunikációs egységet. Nagyon szemléletesnek tartom, hogy az egyes mote-okat gyakran szokás intelligens porszemeknek is nevezni. Az elképzelés ugyanis az, hogy a jövőben a szenzorhálózatok közel porszemnyi méretű, a környezetükkel szoros kapcsolatban álló, intelligens mobil eszközök ezreinek, esetleg millióinak szoros együttműködésével, azok autonóm módon történő hálózatba szerveződésével fognak létrejönni azzal a céllal, hogy kényelmesebbé tegyék a világunkat. [3] [4]

Az elnevezés valószínűleg a 2001-ben lezárult Smart Dust project hatására ivódott be a köztudatban. A Dust Inc. által elindított kezdeményezésnek mára már számos utóprojectje létezik. A rendszer alapjait a Berkeley egyetemen kidolgozott 1 köbmilliméteres térfogattal rendelkező intelligens szenzorjai adták. Az eredeti elképzelés a könnyen és gyorsan, szinte bárhova telepíthető, az ottani helyzetet kiértékelő hálózatok koncepciójának megalkotását célozta meg. A szenzoregységek áramellátót, analóg áramkört és programozható mikroprocesszort tartalmaznak. Az egységek újraprogramozhatók, vagyis elvégzendő feladatuk könnyen módosítható akár távolról is. Ad-hoc hálózatokat alkotnak, maguktól építik fel az idővel változó struktúrájú rendszert és az adatokat mindig az optimálisnak vélt úton továbbítják. A Dust Inc. elsősorban három alkalmazási területre koncentrál: épület-automatizálásra, ipari monitorozásra és biztonságtechnika. [5] [6]

2.3. Alkalmazási területek

A Proactive Health project, ami az Intel és a Rochester Egyetem egészségügyi központjának kezdeményezése szintén az intelligens szenzorhálózatok alkalmazási területeit kutatja. Az intelligens otthon koncepció részeként, önkéntesek lakásaiban tesztelik parányi, rejtett szenzorokból álló hálózataikat. A szenzorok vérnyomást, testsúlyt mérnek és az adott személy helyváltoztatásáról szolgáltatnak információkat. Az Intel egy másik projektjében vezeték nélküli szenzorhálózat monitorozza egy szőlőskertben a hőmérsékletet, majd tárolja el az adatokat. A távolabbi tervekben az szerepel, hogy az érzékelők a talaj nedvességét ellenőrizve kimutatják, mely területeket szükséges locsolni és melyeket nem. [7]

Ezen utóbbi alkalmazási területhez egy magyar fejlesztés is kapcsolódik: a Szőlőőr project. A Budapesti Műszaki és Gazdaságtudományi Egyetem mérnökei által kifejlesztett döntéstámogató rendszer lényege, hogy az egy ponton történő mérést a négy hektáronként lehelyezett állomások önszervező vezeték nélküli hálózatával váltja fel. Így minden egyes állomás percenként rögzíti a környezeti tényezőket, melyek időben és térben kimagasló pontosságú, táblaszintű előrejelzésekkel szolgálnak a leggyakoribb szőlőbetegségek (lisztharmat, peronoszpóra, szürkerothadás) kialakulási valószínűségéről. [8]

A szenzorok és különösen a szenzorhálózatok elterjedése erősíti a teljesen hálózatba kapcsolt világ jövőképét, ami jelentős mértékben hozzá fog járulni a társadalmunk automatizálódásához. Amint ezen eszközök árképzése eléri a széleskörű forgalomba kerülésükhöz szükséges értéket, a leghétköznapibb alkalmazásokban számíthatunk azok megjelenésére. Az intelligens otthonok, vagyis a környezeti-intelligencia komoly segítséget nyújthat az egészségügyi dolgozóknak, hiszen az általa biztosított információk sokkal gyorsabb orvosi beavatkozást, illetve a betegségek kialakulásának megelőzését teszik lehetővé a rendellenes aktivitások előzetes detektálása által. A lakosság elöregedése miatt minden bizonnyal fel fog értékelődni ennek a fajta egészségügyi mérésnek a lehetősége. Az idősebb embereknek nem kell idő előtt egészségügyi okokból idősek otthonába költözniük, hanem távfelügyelet mellett sokkal tovább élhetnek megszokott környezetükben.

A szenzortechnológia mindennapjaink szerves részévé válásával élhetőbbé, biztonságosabbá teszi világunkat. Az érzékelő- és beavatkozó hálózatok hozzájárulnak a termelési folyamatok automatizálásához, a szervezettebb gyártáshoz és természetesen az általuk precízen összegyűjtött adatokra még az élet számos területén támaszkodhatunk.

III. A rendszer felépítése

3.1. Előzetes specifikáció

Az irodalomkutatással foglalkozó fejezetben rávilágítottam arra, hogy szenzorhálózatok fejlesztésével már több mint egy évtizede foglalkoznak. A nagyobb gyártók számos terméke elérhető a piacon, melyek a legkülönfélébb felhasználási területeket célozzák: épületautomatizálás, egészségügy, biztonságtechnika, autóipar, stb. Ennek ellenére érdemes megvizsgálni az egyes koncepciókat, ugyanis egy jó ötlet bármikor megreformálhatja az eddigi rendszereket.

Alapvetően léteznek vezetékes és vezeték nélküli alternatívák. Ezeknek a dolgozatom szempontjából releváns előnyeit és hátrányait a következő táblázat tartalmazza:

Vezetékes szenzorhálózatok		**Vezeték nélküli szenzorhálózatok**
Magas!	*Telepítési költség*	**Alacsony**
Alacsony	*Kommunikációs periféria költsége*	**Közepes / Magas** *(függ a választott rádiós modultól)*
Hálózati táplálás	*Telepélettartam*	**Korlátos**
Nincs egymásra hatás	*Több különböző hálózat szimultán működése*	**Megosztott csatornakihasználás**

Manapság ha otthoni környezetekbe szánt szenzorhálózatokról beszélünk, akkor szinte azonnal a valamilyen rádiós kommunikációt használó alternatívák kerülnek előtérbe. Ez nem meglepő, hiszen a leggyakrabban nem új építésű házakba kerülnek beépítésre ezek a rendszerek, gondoljunk csak a bevezetőben említett idős emberek felügyeletét biztosító koncepcióra, hanem már meglévő épületekbe. Ebben az esetben pedig a rendszer telepítési költségére kerül a legnagyobb hangsúly. A vezeték nélküli szenzorhálózatok rendkívül rugalmasan és könnyen telepíthetők, hiszen szinte semmilyen kábelezést nem igényelnek. A vezetékes hálózatok esetében viszont a vezetékezés költsége sokszor olyan magas lehet, hogy az alacsonyabb mote ár és az irreleváns telepélettartam ellenére sem éri meg a vezetékes szenzorhálózat kialakítása. A megfelelő kommunikációs csatorna és frekvenciasáv megválasztásával ráadásul a vezeték nélküli mote-ok

legtöbb hátránya is megszűntethető. Ennek ellenére érdemes lenne egy olyan hibrid szenzorhálózat megtervezése mely ötvözné a két alternatíva előnyeit.

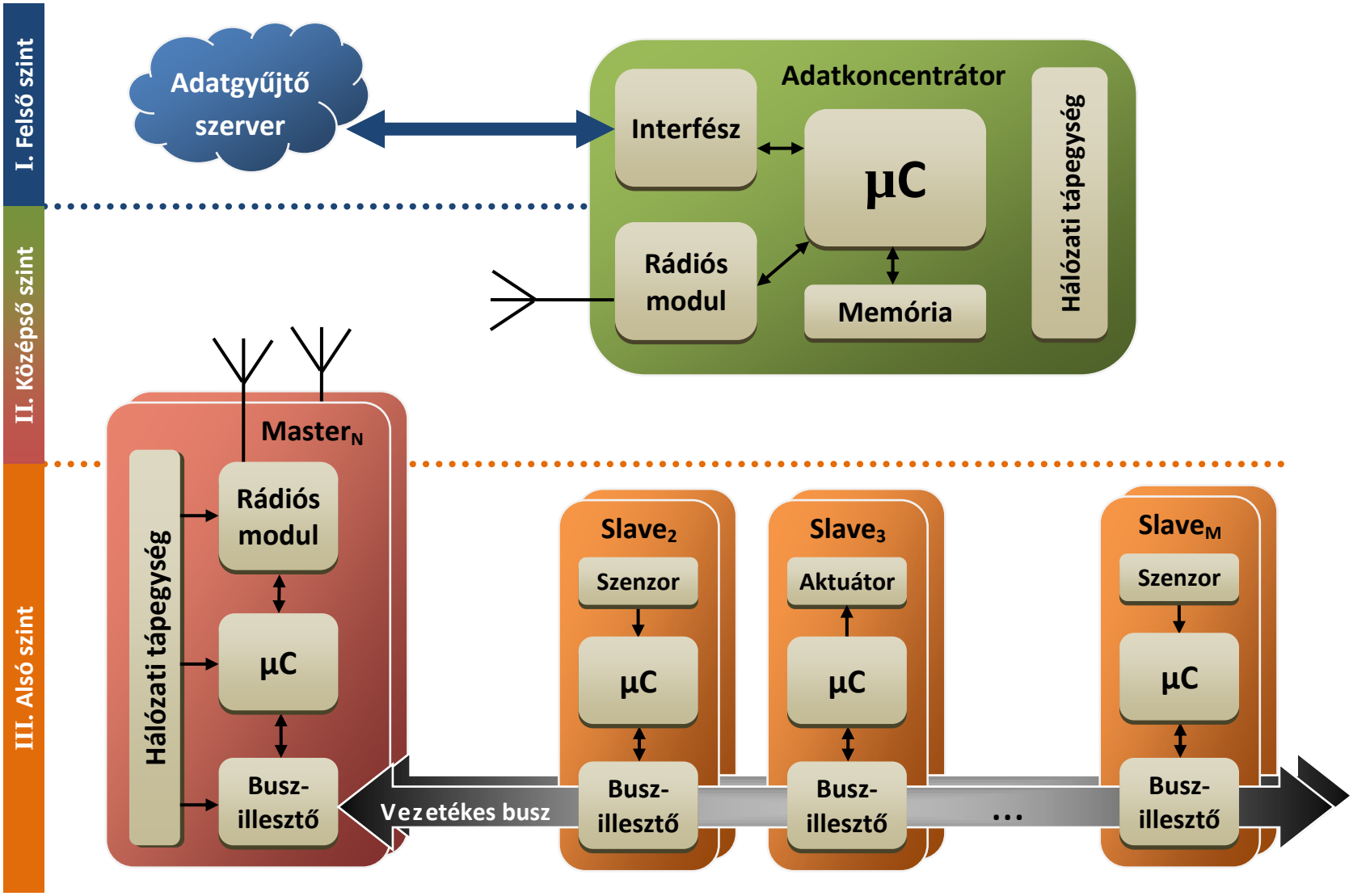
A célkitűzésem, hogy megvalósítsak egy alapvetően vezetékes buszkapcsolatot alkalmazó, hálózati táplálással rendelkező szenzor-, illetve aktuátorhálózatot, mely a könnyebb telepíthetőség érdekében egy vezeték nélküli rendszer alhálózatát képezi. A munkám során kiemelt figyelmet fordítok a rendszertervezés szempontjából rendkívül előnyös hierarchikus elrendezés megvalósításának.

3.2. Előzetes rendszerterv

A rendszernek a HOLMS fantázianevet adtam, mely mozaikszó az általa ellátott funkció és alkalmazási terület angol megfelelőjéből állt össze. Az otthoni felügyeletre és elő monitorozásra alkalmas rendszer három egymástól jól elkülöníthető szintből fog állni. A következő táblázat a szenzorhálózat hardverkomponenseit rendeli hozzá ezekhez a hierarchiaszintekhez:

	RENDSZERKOMPONENS TÍPUSA	KOMMUNIKÁCIÓ TÍPUSA
I. Felső szint	***(Szerver)***	**Magas szintű hálózat**
	Adatkoncentrátor *1 darab*	
II. Középső szint		**Vezeték nélküli hálózat**
	Master egység *N darab*	
III. Alsó szint		**Vezetékes busz**
	Slave egység *M darab*	

A fenti táblázatból jól látható, hogy a hierarchiaszintek szétválasztása az egyes hardverkomponenseken belül történik, vagyis szoftveres szétválasztás formájában valósul meg. A *Szerver* természetesen nem egy saját komponens, csupán a felső szint célállomását jelöli. Megvalósulhat egy helyi PC-n vagy egy távoli állomáson is.



3. ábra: Előzetes rendszerterv

A rendszer legalsó szintjén találhatók a *Slave* egységek. Ezek a rendszer legkisebb számítási teljesítménnyel rendelkező egységei. Feladatuk a szenzorok vagy aktuátorok kezelése és a *Master* egységek felé az információáramlás biztosítása. A kommunikáció vezetékes buszon keresztül valósul meg, melyen a *Slave*-ek számára a szükséges tápellátás is biztosítva van.

A *Master* egységek valósítják meg az alsó és középső szint közötti átjárást. Vezetékes busz interfészt biztosítanak a *Slave* egységek felfűzéséhez és hálózati tápegységük segítségével gondoskodnak a busz tápellátásáról. Egymással vezeték nélküli kapcsolaton keresztül kommunikálnak, belőlük épül fel az önszerveződő szenzorhálózat.

Az *Adatkoncentrátor* a szenzorhálózat kitüntetett eleme, a koordinátor szerepét tölti be. A rendszerben csak egy lehet jelen, ő kezdeményezi a vezeték nélküli hálózat felépítését. A szenzorhálózat felől érkező információkat egy fejlett magas szintű hálózati kapcsolat segítségével továbbítja a külvilág felé, így megvalósítva a középső és magas hierarchia szintek közötti átjárást. Komplex feladata miatt a rendszer legnagyobb számítási teljesítménnyel rendelkező egysége.

IV. Hálózati rendszer kialakításának alternatívái

A szenzorhálózat megtervezése során az első feladatom a rendszer alsó és középső hierarchia szintjein lévő hálózatok lehetséges kialakítási alternatíváinak vizsgálata volt. Az alsó szinten vezetékes hálózat fogja biztosítani a hálózati táplálásból adódó előnyöket, míg a középső hierarchia szinten vezeték nélküli hálózat garantálja majd a minimális telepítési költséget.

4.1. Vezetékes hálózat

A rendszer alsó hierarchia szintjén vezetékes alhálózatok fogják össze a különféle szenzor- és aktuátor egységeket. Egy alhálózat egy masterből és néhány slave egységből áll. A slave egységek kezelik a szenzorokat és aktuátorokat, míg a master vezérli a saját slave egységeit és gateway szerepet lát el a középső szintű hálózat felé. Szenzorok és aktuátorok tekintetében otthoni környezetben tipikusan a következő egységek fordulnak elő:

Érzékelők:

- Hőmérséklet
- Páratartalom
- Mozgás
- Nyomás
- Átfolyás
- Kontakt / Nyitás

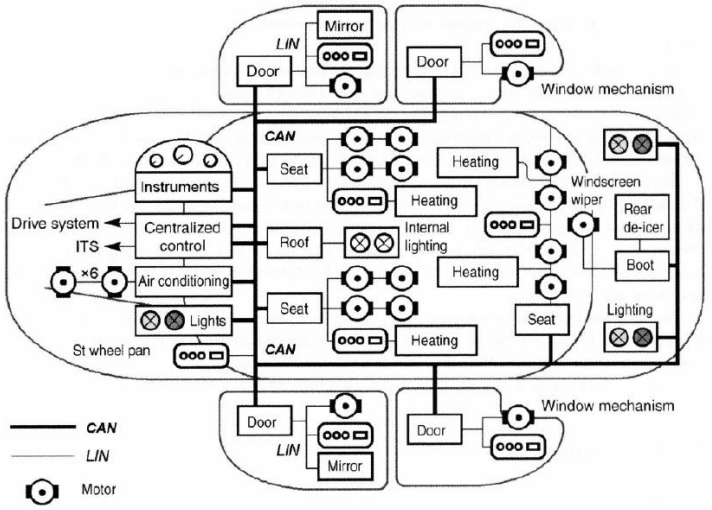
Aktuátorok:

- Lámpák / Jelzőfények
- Szirénák
- Szelepállító motorok
- Elektromechanikus zárak

A megtervezendő szenzorhálózat elsősorban otthonok állapotának monitorozására készül. Egy tipikus alkalmazása lehet majd egy lakás klíma-adatainak rögzítése és ezen információk felhasználásával annak szabályozása. Ezt figyelembe véve megállapítható, hogy a szenzorokat és aktuátorokat kiszolgáló hálózatnak nem szükséges, hogy rendkívül nagy adatátviteli sebességgel rendelkezzen, hiszen a felcsatlakozó eszközök csak viszonylag ritkán fogják igényelni a hálózathasználat jogát.

Olyan vezetékes hálózati alternatívákat kellett keresnem, melyek költség-hatékonyan képesek ellátni a fent ismertetett feladatot. Megbízhatóság szempontjából a legjobb választást az autóipar számára kidolgozott vezetékes buszok jelentik. A gépjárművekben uralkodó szélsőséges körülményekre tervezett áramköri elemek az otthoni környezetekben minden bizonnyal gond nélkül helyt fognak állni. A rendkívül kiforrott protokoll készletnek és a rendelkezésre álló

fejlett fejlesztői eszközöknek köszönhetően ezek választásával egy rendkívül stabil és megbízható hálózat alakítható ki. Ezen érvek miatt én az autóiparban jelenleg legszélesebb körben alkalmazott két busz szabványt, a CAN és LIN hálózatokat vizsgálom meg részletesebben.



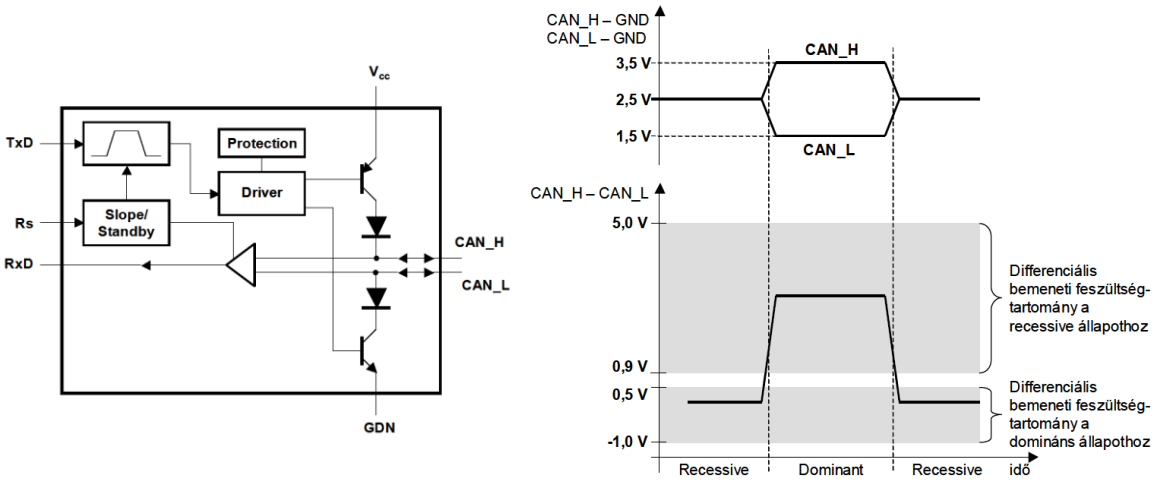
4. ábra: LIN és CAN hálózatok egy gépjárműben [9]

4.1.1. CAN

A CAN (Controller Area Network) protokollt a Robert Bosch cég fejlesztette ki az 1980-as évek első felében. Az azóta megjelent CAN 2.0-ás verzió az autóiparban használt buszkommunikáció egyik vezető szabványa lett, de széles körben alkalmazzák az ipar számos más területein is. [10]

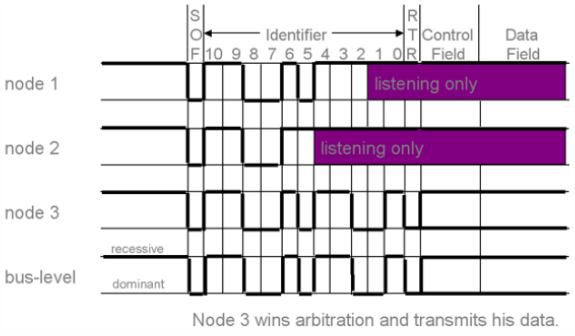
A CAN a helyi hálózatok egy speciális fajtája, az ipari buszok csoportjába tartozik. Topológiáját tekintve lehet normál busz, pont-pont vagy csillag elrendezésű. Szórásos hálózat, tehát a kiküldött kereteket az összes felcsatlakozott eszköz fogadja. A CAN-nek a fizikai rétegét tekintve több változata is létezik, ezek közül a High-Speed (HS) CAN és a Low-Speed (LS) vagy más néven a Fault-Tolerant CAN a leggyakrabban alkalmazott. A HS CAN maximálisan adatátviteli sebessége 1 Mb/s. Ezzel szemben az LS CAN 125 kb/s sebességre képes, azonban annak hibatűrő, hibakezelő szolgáltatásai is vannak. Mindkettő csavart érpárt alkalmaz szimmetrikus adóval és szimmetrikus vevővel. Az adás és a vétel ugyanazon az érpáron történik, tehát half-duplex átvitelről van szó. Az alkalmazott adatkódolás NRZ (Non-Return To Zero), bitbeszúrással kiegészítve. Fontos megjegyezni, hogy a maximálisan elérhető sebességérték függ a felhasznált kábel hosszától.

A busz nem destruktív ütközésérzékelést valósít meg, amihez elengedhetetlen az állomások közötti huzalozott ÉS kapcsolat. Az ehhez szükséges két jelállapot a recesszív (elengedett, 0 bitérték) és a domináns (meghúzott, 1 bitérték) állapot. A busz recesszív állapotában van ha a két vezeték feszültségszintje azonos, illetve domináns állapotban van ha az egyik vezeték magasabb, a másik alacsonyabb feszültségszintre kerül. Ennek köszönhetően a kommunikáció erős zavarvédelemmel rendelkezik, hiszen az egymás mellett fekvő vezetékekre ható bármely közös modusú zavar a különbségképzésnek köszönhetően nem érzékelhető. [9]



5. ábra: CAN buszmeghajtó egyszerűsített rajza és a CAN busz fizikai jelszintjei [9]

Az adatokat a buszon keretekbe ágyazva továbbítódnak. A CAN Multi-Master architektúrájú, vagyis üzenetküldést bármely egység kezdeményezhet. Amennyiben több állomás szeretne egy időben küldeni, a buszhozzáférési probléma a fent már említett CSMA/CA ütközéselkerüléssel, az üzenetek azonosítójának felhasználásával feloldható. Ez azért előnyös, mert így a prioritás nem az egyes állomásokhoz, hanem az üzenetek azonosítójához van hozzárendelve.



6. ábra: Egy tipikus CAN arbitrációs folyamat [11]

Amennyiben egy állomás az azonosító küldése során egy bitnél a buszt recesszív szinten hagyja és ennek ellenére azt érzékeli, hogy az domináns szintre került, leállítja a küldést és majd csak a busz szabaddá válása után próbálkozik újra. Ilyen módon a kisebb azonosítójú üzenetek nagyobb prioritást élveznek és anélkül jutnak érvényre, hogy a küldést le kellett volna állítani, majd újrakezdeni azt.

A CAN fejlett hibakezeléssel rendelkezik. Amennyiben egy állomás bármilyen hibát észlel a kommunikációban, kötelessége azonnal egy hiba keretet küldeni a buszra. Ez nem más, mint a busz azonnali domináns állapotba vitele 6 bit hosszúságig. Normál kommunikáció esetén előírás, hogy minden 5 azonos szintű bit után egy ellenkező értékű bitet be kell szúrni. A hiba keretben található 6 egymást követő domináns bit azonban megsérti ezt a szabályt és pontosan ez teszi lehetővé, hogy amennyiben nem minden állomás érzékelte az eredeti kommunikációs hibát, a bitbeszúrás szándékos megsértését már mindegyik detektálja azt. Minden állomás tartalmaz egy a küldési és egy a fogadási hibák nyilvántartására fenntartott számlálót. Ezek automatikusan inkrementálódnak ha hiba történt, illetve dekrementálódnak ha hibátlan volt az átvitel. Ha a hibák száma meghalad egy bizonyos értéket, az állomás a normál Error Active állapotból átvált Error Passive állapotba. Ha ezt követően a hibák száma ismét meghalad egy bizonyos értéket, az állomásnak be kell szüntetnie az adatátvitelt a busz felé.

A CAN buszprotokoll legfontosabb jellemzői:

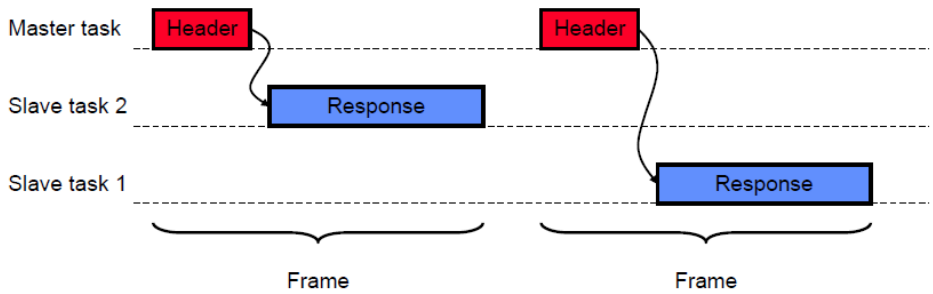
- Multi-Master architektúra
- 1 Mb/s maximális adatátviteli sebesség a kábelhossztól függően
- Maximális kábelhossz: 40-500 m (sebességtől, topológiától függően)
- Gyakorlatilag korlátlan állomásszám
- Fejlett hibadetektálás / hibakezelés
- Nagymegbízhatóságú autóipari alkatrészek rendelkezésre állása

4.1.2. LIN

A LIN (Local Interconnect Network) protokollt a Motorola kezdte el kidolgozni az 90-es évek legvégén. Később további öt nagy európai autóipari vállalat összefogásának eredményeként létrejött a LIN konzorcium, mely az első széles körben elterjedt 1.2-es LIN verziót 2002-ben publikálta. Azóta már megjelent a 2.2-es változat is és a LIN lett a nagyteljesítményű hálózatok elsőszámú költséghatékony alhálózati alternatívája. A szabvány teljes mértékben publikus és bárki számára szabadon letölthető a konzorcium honlapjáról. [12]

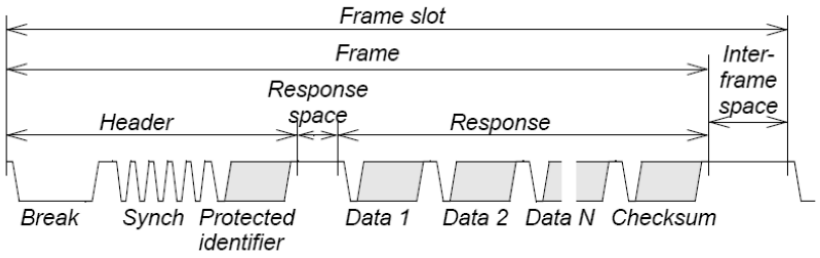
A LIN létrehozásának egyértelmű célja az volt, hogy elsősorban a CAN-nél egy lényegesen olcsóbb, relatív lassú és elegendő megbízhatósággal rendelkező hálózatot specifikáljon. A 2000-es évekre a fejlesztők már rengeteg tapasztalattal rendelkeztek a különféle ipari buszokkal kapcsolatban, így ezekre támaszkodva alkothatták meg ezt az új protokollt. Általánosságban elmondható, hogy a LIN egy nagyon jól specifikált és sok fontos apró részletre kitérő szabvány.

A buszon egy Master és több Slave egység tartózkodhat. Csak a Master egység kezdeményezhet adatátvitelt, ezért nem léphet fel arbitrációs probléma. A LIN egyvezetékes kommunikációt használ, ami half-duplex átvitelt tesz lehetővé. A hálózat szórásos alapú, vagyis az aszinkron soros formában küldött adatokat a buszra kapcsolódó összes állomás fogadja. A Master szabja meg, hogy a buszon milyen üzenet és mikor kerül elküldésre. Minden adatátvitelt a Master indít el és a Slave-ek csak akkor küldhetnek adatot, ha erre felkérik őket. A LIN nagy előnye a determinisztikus működés: a Master tárol egy ütemezési tábla, mely azt tartalmazza hogy mikor melyik Slave-et kell lekérdeznie. Az adatok keretekben (*Frame*) kerülnek továbbításra a buszon. Egy keret egy fejlécből (*Header*) - amit a Master állít elő - és egy válaszból (*Response*) - amit a Slave ad ki a buszra - áll:



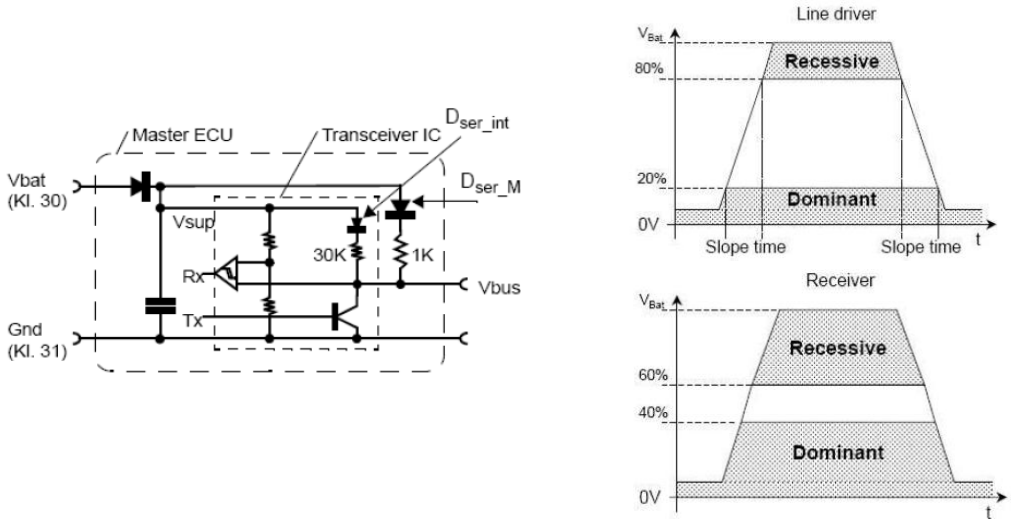
7. ábra: Master-Slave kommunikáció a LIN buszon [13]

Az adatátvitel a LIN buszon a normál UART kommunikációhoz hasonló. Az adatbájtok 8N1 formátumban kerülnek továbbításra, a legkisebb helyi értékű bittel kiindulva. Az üzenetek emellett tartalmaznak egy speciális szinkronizáló mezőt is, ami lehetővé teszi, hogy az állomásoknál a fejlesztők elhagyhassák a külső kvarc használatát és ezáltal költséghatékony rendszert tudjanak kialakítani.



8. ábra: LIN keret felépítése [9]

A kommunikáció maximálisan 20 kb/s sebességű lehet, de általában a szabványos UART sebességeket (pl. 9600 baud) szokták alkalmazni. A maximális kábelhossz 40 méter lehet és az állomásszám is limitálva van a Master-el együtt 16-ra. A bitkódolás a CAN-hez hasonlóan NRZ. Az állomások az egyvezetékes buszt nyitott-kollektorosan hajtják meg, a jelszintek általában a gépjárművek feszültségszintjeihez igazodnak. A buszon a recesszív jelszint jelenti a logikai 1, míg a domináns jelszint a logikai 0 értéket. [9]



9. ábra: Tipikus Master oldali buszmeghajtó és a LIN busz fizikai jelszintjei [9]

A Master és Slave egységek a buszon különböző működési módokban lehetnek. A buszmenedzsment feladata, hogy az állapotváltásokat kezelje. Lehetőség van a buszt, illetve a rá csatlakozó egységeket alvó állapotba küldeni, valamint egy wake-up kerettel felébreszteni.

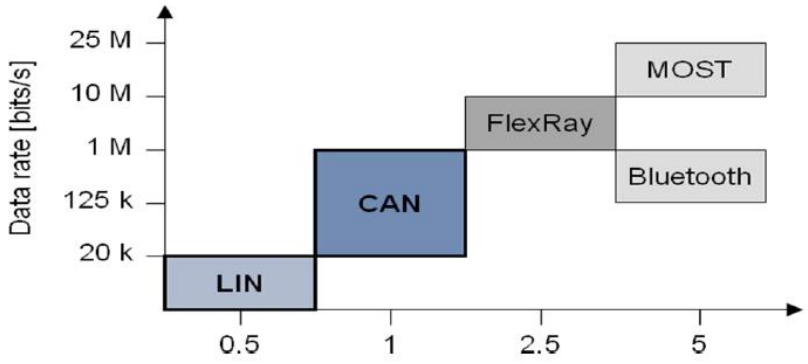
A LIN is tartalmaz hibadetektálási és jelzési funkciókat, igaz nem olyan hatékonyakat mint a CAN. Az üzenetek nyugtázására nincs lehetőség, azonban erre nincs is szükség, hiszen a Master a Slave-től kapott válasza alapján meg tudja állapítani, hogy sikeres volt-e kommunikáció. Az egyes állomások folyamatosan monitorozzák a buszt, hogy ott valóban az általuk kiadott jelszint jelent-e meg. A hibadetektálást emellett a keretek 2 paritás bitje és a CRC mező is biztosítja.

A LIN buszprotokoll legfontosabb jellemzői:

- Csak 1 Master egység lehet jelen a buszon
- 20 kb/s maximális adatátviteli sebesség
- Maximális kábelhossz: 40 m
- Limitált állomásszám (1 Master + 15 Slave)
- Nagymegbízhatóságú autóipari alkatrészek rendelkezésre állása
- Költséghatékony

4.1.3. Értékelés

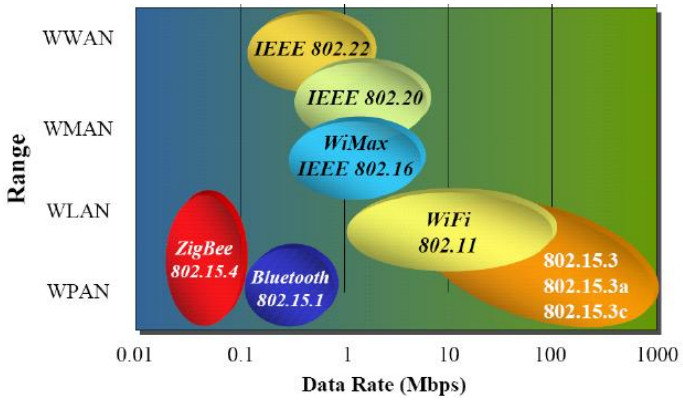
A feladat jellemzőinek vizsgálata után arra a következtetése jutottam, hogy a megvalósítandó szenzorhálózat nem használná ki a CAN rendkívül fejlett és sokrétű szolgáltatásait. Az adott feladathoz a LIN hálózat illik a legjobban. A buszon nem lesz szükség több Master egységre, valamint a maximálisan elérhető 20 kb/s-os adatátviteli sebesség és a 15-ben korlátozott Slave állomásszám is tökéletese megfelel a célkitűzés elvárásainak. Mivel a költséghatékonyság is fontos tervezési szempont, az alábbi összehasonlító diagram csak megerősített a döntésemben: a LIN busz az ideális választás a szenzorrendszer alsó hierarchia szintje számára.



10. ábra: Autóipari buszok CAN-hez viszonyított relatív költsége [9]

4.2. Vezeték nélküli hálózat

A rendszer középső hierarchia szintjét az alhálózatok Master egységei alkotják, melyek vezeték nélküli kapcsolatban állnak egymással. A rádiós kapcsolatot biztosító hálózati struktúra kiválasztásánál arra kellett ügyelnem, hogy olyan alternatívát válasszak, mely rendelkezik a szenzorhálózatokkal kapcsolatos egyik legfontosabb tulajdonsággal, az önszerveződés képességével. Ebből kifolyólag olyan megoldásokat kerestem, melyek valamilyen magas szintű hálózati protokollt futtatnak.



11. ábra: Vezeték nélküli kommunikációs szabványok [14]

A fenti ábra jól szemlélteti a legelterjedtebb vezeték nélküli átviteli szabványok viszonyát a hatótávolság és adatátviteli sebesség szempontjából. A szenzorhálózatok tipikusan kis adatrátájú kommunikációt igényelnek, ezért én az ebben a szegmensben található Bluetooth és ZigBee szabványokat vizsgáltam meg részletesebben.

4.2.1. Bluetooth

A Bluetooth koncepcióját az Ericsson alkottam meg 1994-ben azzal a céllal, hogy egy olyan rádiós szabványt készítsen amely egységesítheti a mobileszközök közötti drótnélküli adatcserét. Elsősorban kábelek vezeték nélküli alternatívájának szánták, ezért ennek megfelelően viszonylag alacsony hatótávolsággal és a soros vonalakhoz képest magas adatátviteli sebességgel rendelkezik. Fogyasztása és a protokoll futtatásához szükséges számítási-teljesítménye igénye a WiFi-hez képest alacsonyabb, de továbbra is magas. A Bluetooth stack több, különböző alkalmazási területekre szánt profilt tartalmaz, melyek közül a felhasználó szabadon választhat. Ez egyik legelterjedtebb az SPP (Serial Port Profile), mely egy egyszerű soros port emulációját valósítja meg.

Az egyes készülékeket adóteljesítményük (1-100 mW) alapján 3 osztályba sorolják, a velük elérhető hatótáv 5 és 100 méter közé tehető. A 2,4 GHz-es frekvenciasávot használja és a 2.0-s szabvány már akár 1 Mb/s-os adatátviteli sebességet is képes biztosítani. Az egymáshoz csatlakoztatott eszközök egy PAN (Personal-Area Network) hálózatot, vagyis egy piconet-et hoznak létre, melynek 1 master és 7 további készülék lehet a tagja. A tervezés során fontos szempont volt a biztonság kérdése, ezért a Bluetooth-hoz egy háromszintű (nincs, szerviz szint, adatkapcsolati szint) biztonsági rendszert dolgoztak ki. A biztonságos adatcserét a kapcsolat kiépítéskor szükséges hitelesítési folyamat garantálja, melynek alapját a két eszköz közti, előre kiosztott kulcsot használó eszközkapcsolat jelenti. [15]

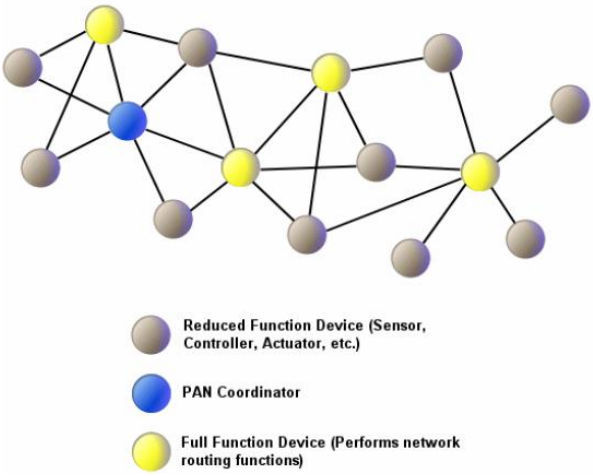
A Bluetooth mára egy rendkívül kiforrott szabványnak tekinthető, sok hardver- és szoftvertámogatással. A technológia megtalálható a „buta mobiltelefonoktól" kezdve, egészen az intelligens vérnyomásmérőkig. Ennek köszönhetően a különböző rendszerek, különböző készülékei könnyen kapcsolatba léphetnek egymással. Bár természetesen nem szabad megfeledkezni arról, hogy a közös profil támogatása így is elengedhetetlen. Az alapítók egységesítő célkitűzése megvalósulni látszik, pont úgy mint annak az északi törzseket egyesíteni kívánó, áfonyát kedvelő dán királynak a terve, akiről a technológia a nevét kapta. A feladatom szempontjából nagy előnyt jelentene egy Bluetooth-ot használó szenzorhálózat, hiszen az bármilyen további hardverelem felhasználása nélkül lenne képes például a felhasználó mobiltelefonjához kapcsolódni és arra tájékoztató üzeneteket küldeni. A szabvány azonban elsősorban a két eszköz közötti egyszerű adatcserét hivatott biztosítani és nem azok hálózatba szervezését. Ennek a funkciónak a magasabb szintű támogatása csak a legújabb 4.0-s verzióban kezd megjelenni, melyhez egyenlőre rendkívül drágák és nehezen elérhetők a fejlesztőeszközök. A jövőben minden bizonnyal nagyobb szerephez fog jutni a Bluetooth a szenzorhálózatok területén is, de jelenleg nem tartom jó választásnak a feladatom megvalósításához.

4.2.2. ZigBee

A ZigBee egy magas szintű vezeték nélküli hálózati szabvány, melynek segítségével alacsony fogyasztású, olcsó, kis számítási kapacitású eszközökből (node-ok) lehet ad-hoc, önszervező és önjavító hálózatot létrehozni. A szabvány

rendszeres karbantartását és frissítését a ZigBee Alliance végzi. Azzal a céllal kezdték meg 1998-ban a fejlesztését, hogy kitöltsék a szenzorok és vezérlő egységek speciális igényeit kielégítő vezeték nélküli kommunikációs szabvány hiányát jelentő űrt. Ezek a rendszerek nem igényelnek nagy sávszélességet, azonban fontos a rövid várakozási idő, az alacsony energia felhasználás és a biztonságos kommunikáció, valamint természetesen az alacsony alkatrészköltség biztosítása is. [16]

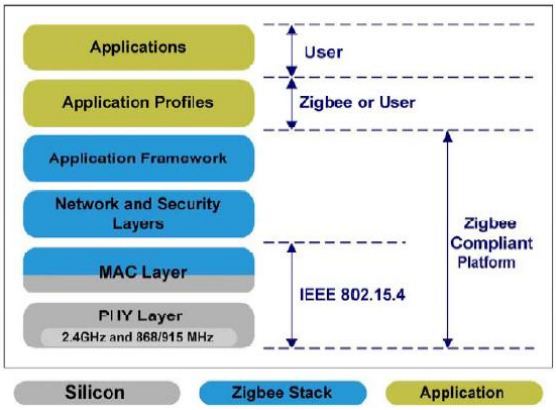
A ZigBee alapvetően két eszközosztályt különböztet meg: Full Function Device (FFD) és Reduced Function Device (RFD). RFD kizárólag csak Végpont lehet, ami nem vehet részt az útválasztásban. Ez azzal a hatalmas előnnyel jár, hogy nem kell folyamatosan rendelkezésre állnia és periodikusan alvó állapotba kerülhet. Tipikusan csak a saját szenzorjaik és aktuátoraik kezelésével foglalkoznak. FFD ezzel szemben lehet Koordinátor vagy Router. A Koordinátor kitűntetett szereppel rendelkezik a hálózatban, ő végzi annak menedzselését, ezért csak 1 darab lehet jelen a rendszerben. Természetesen az FFD-k is rendelkezhetnek saját szenzorokkal. Ezen eszközosztályokból felépítve a ZigBee három topológiát definiál (Csillag, Klaszter, Mesh). Ezek közül talán a Mesh a legjelentősebb, ugyanis segítségével a szenzorhálózat hatótávolsága jelentősen megnövelhető. [18]



12. ábra: Mesh hálózati struktúra [14]

A Router-ek átjátszó szerepet töltenek be és egészen addig továbbítják az üzenetet a többi Router felé amíg az célba nem ér. A csomag által bejárt útvonal előre definiált, a hálózatba kapcsolt eszközök saját maguk építik fel és jegyzik meg a legelső kommunikációs kísérlet idején. A hálózat önjavító, mivel ha az egyik Router kiesik, akkor a többi hozzá közeli Router átveszi annak feladatát.

A ZigBee egy magas szintű vezeték nélküli hálózati szabvány, annak alacsony szintű alapjait az IEEE 802.15.4 szabványon definiálja. Ez utóbbi tartalmazza az alsó kettő fizikai (PHY) és adatkapcsolati (MAC) rétegeket. A ZigBee specifikáció az ezek felett elhelyezkedő hálózati (NWK) és alkalmazási rétegeket írja le.



13. ábra: A ZigBee és az IEEE 802.15.4 kapcsolata [14]

A fizikai réteg a 818 MHz / 915 MHz és 2,4 GHz sávok használatát definiálja. Az alacsonyabb frekvenciasávok akadálytalanabb terjedést biztosítanak, ennek ellenére a 2,4 GHz-es ISM használata a legelterjedtebb, amelyen 16 csatorna valamelyikén kommunikálhatnak az eszközök. A hálózati réteg feladata a topológia kialakítása és az útválasztás. Az alkalmazási réteg az alsóbb rétegek szolgáltatásait teszi elérhetővé a felsőbb rétegek számára egy kezelhető formában, tehát gyakorlatilag egy API-t nyújt a felhasználónak. [17]

A ZigBee szabvány összegyűjtve tartalmazza a tipikus alkalmazási területeket. Ilyenek például a világítás- és hűtés/fűtés vezérlés, valamint a mérési adatgyűjtés. Ezeket hívják úgynevezett Functional domain-eknek. A Cluster-ek pedig ezen domain-ek szorosan összetartozó egységekké történő további lebontásával születtek. Ilyen Cluster például a mérési adatgyűjtés domain-ben a hőmérséklet vagy légnyomás mérés. A Cluster-ek tartalmazzák a feladatukhoz kapcsoló attribútumokat és parancsokat, így azok ezek szabványos gyűjteményének tekinthetők. A ZigBee minden egyes Functional domain-nek, Cluster-nek, attribútumnak és parancsnak meghatároz egy egyedi azonosítót.

A Functional domain-eken kívül alkalmazásprofilok is definiálásra kerültek a ZigBee szabványban. Ez elsősorban a különböző gyártók azonos alkalmazási területet célzó termékeinket együttműködését hívatott biztosítani. Ilyen például a ZigBee Home Automation profil. Itt fontos megemlíteni a végpontok fogalmát.

Segítségükkel lehetőség van több különböző logikai eszköz (pl. szenzor) megkülönböztetésére egy fizikai berendezésen belül. Az alkalmazás profilokat sem eszközhöz, hanem végpontokhoz kell hozzárendelni. Ebből következik, hogy egy eszköz akár több alkalmazás profilt is támogathat. [16]

4.2.3. Értékelés

A ZigBee tökéletesen alkalmas relatív lassú adatátvitelt igénylő, alacsony energia-felhasználású beágyazott eszközök vezeték nélküli kommunikációjának kialakítására. A szabvány által is megcélzott tipikus alkalmazási területe a szenzorhálózatok, itt a teljesítménye és a relatív költsége is sokkal jobb mint Bluetooth alapú hálózatoké.

	ZigBee	Bluetooth
Új egység felvétele a hálózatba	**30 ms**	**20 s**
Aktív állapotba váltás alvó állapotból	**15 ms**	**3 s**
Aktív egység tipikus elérési ideje	**15 ms**	**2 ms**

A fenti táblázat [14] nagyon jól szemlélteti, hogy a Bluetooth nem elsősorban önszerveződő és önjavító hálózatok kialakítására lett specifikálva. A kutatómunkámat összegezve arra jutottam, hogy minden kétséget kizárólag a ZigBee protokoll alkalmazása az optimális választás a TDK dolgozatomban bemutatott adatgyűjtő-vezérlő hálózati rendszer középső hierarchia szintjének szánt vezeték nélküli hálózat számára.

V. Rendszerterv

Miután kiválasztottam a rendszer számára legmegfelelőbb vezetékes és vezeték nélküli hálózati megoldásokat, a korábban bemutatott előzetes rendszerterv alapján elkészíthettem a végleges hardver és szoftver rendszertervet.

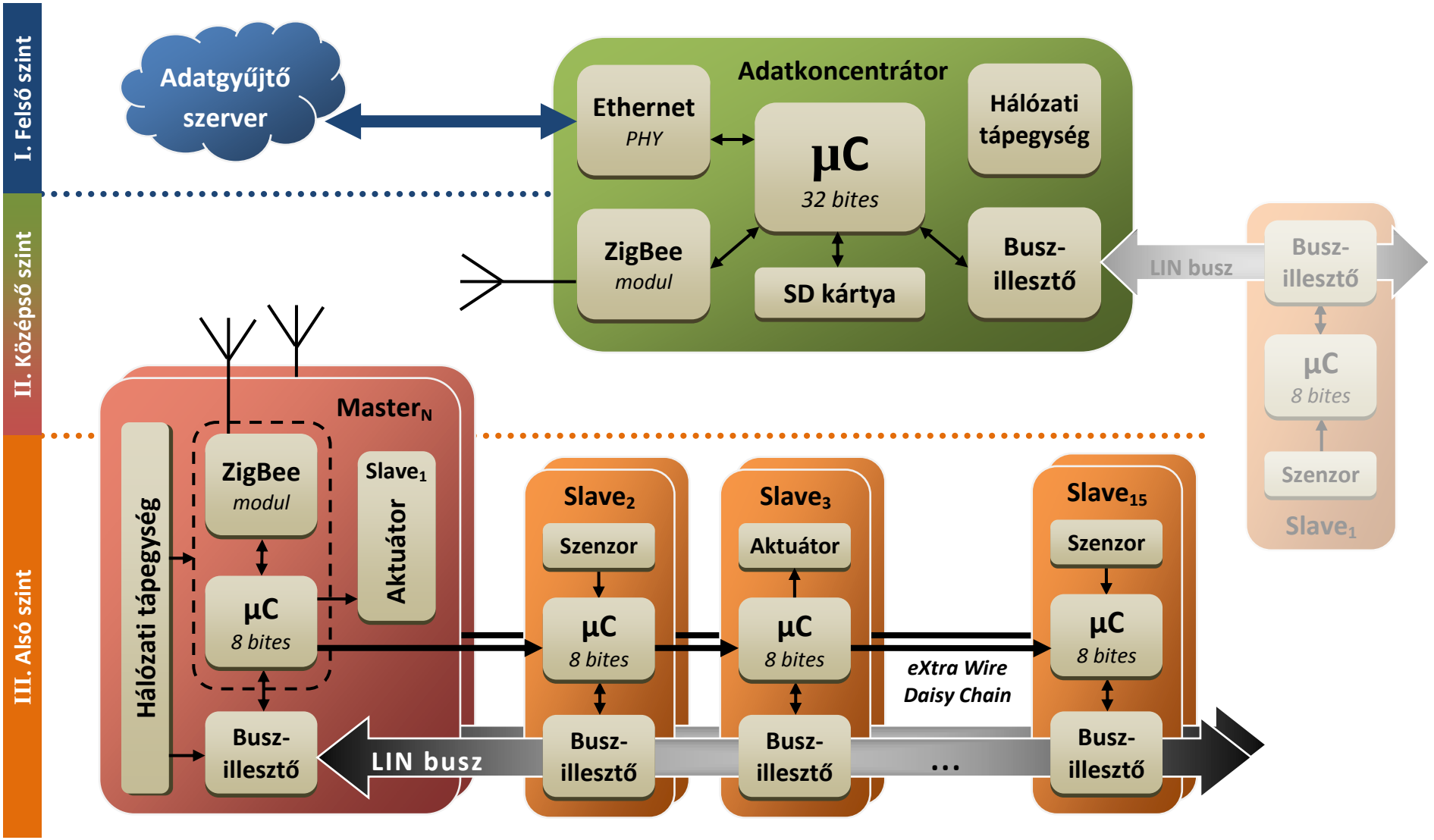
5.1. Hardver rendszerterv

Az *Adatkoncentrátor* egy saját hálózati tápegységgel rendelkező hardver-komponens. Alapvetően egy nagy számítási teljesítményű, fejlett 32-bites mikrovezérlő köré felépített egység, amire elsősorban a magas szintű hálózati kapcsolat kiszolgálása miatt van szükség. Én a rendszer az Ethernet hálózathoz történő csatlakoztatása mellett döntöttem, mivel az ilyen fejlett kontrollerek tartalmaznak olyan belső perifériát, mely mellé már csak egy az Ethernet fizikai rétegéhez való csatolást megvalósító külső szintillesztő áramkörre van szükség. Az egységben egy SD kártya látja majd el a külső nem felejtő memória szerepét, ami arra az esetre, ha a távoli szerver nem lenne elérhető, biztosítja a szenzorhálózat felől érkező adatokat ideiglenesen tárolását. Az *Adatkoncentrátor* egység is ki lett egészítve egy további vezetékes busz interfésszel, ezáltal hozzá közvetlenül *Slave* egységek csatlakoztathatók. Ez a tervezési döntés elsősorban a költséghatékonyság miatt született, mert így a rendszer legalapvetőbb kivitele csak az *Adatkoncentrátor* és a *Slave* egységeket igényli. Ennek köszönhetően ha a telepítendő rendszerben nincs szükség a vezeték nélküli adatátvitel előnyeire, a viszonylag nagy beruházási költséget jelentő *Master* egységek egyszerűen kihagyhatók. A rádiós kapcsolatot egy ZigBee modul biztosítja a rendszer egységei között, melynek magas szintű protokollja képes megvalósítani a hálózattól elvárt önszerveződési képességet. Ezek a modulok általában többféle hardveres kommunikációs lehetőséget is biztosítanak a host processzor felé. A jelen alkalmazáshoz mind sebesség, mind protokoll szempontjából megfelelő választás az egyszerű soros adatátvitel, ami a rendszer élesztése során a hibakeresést is jelentős mértékben megkönnyítheti.

A *Master* szintén egy saját hálózati tápegységgel rendelkező egysége a rendszernek. Ennek köszönhetően képes a *Slave* egységek buszon keresztüli megtáplálására és ebből adódóan az elemmentes koncepció megvalósítására. Mivel az egységet egy hálózati aljzathoz kell majd csatlakoztatni, a végleges hardver rendszerterv elkészítése során úgy döntöttem, hogy érdemes azt egy

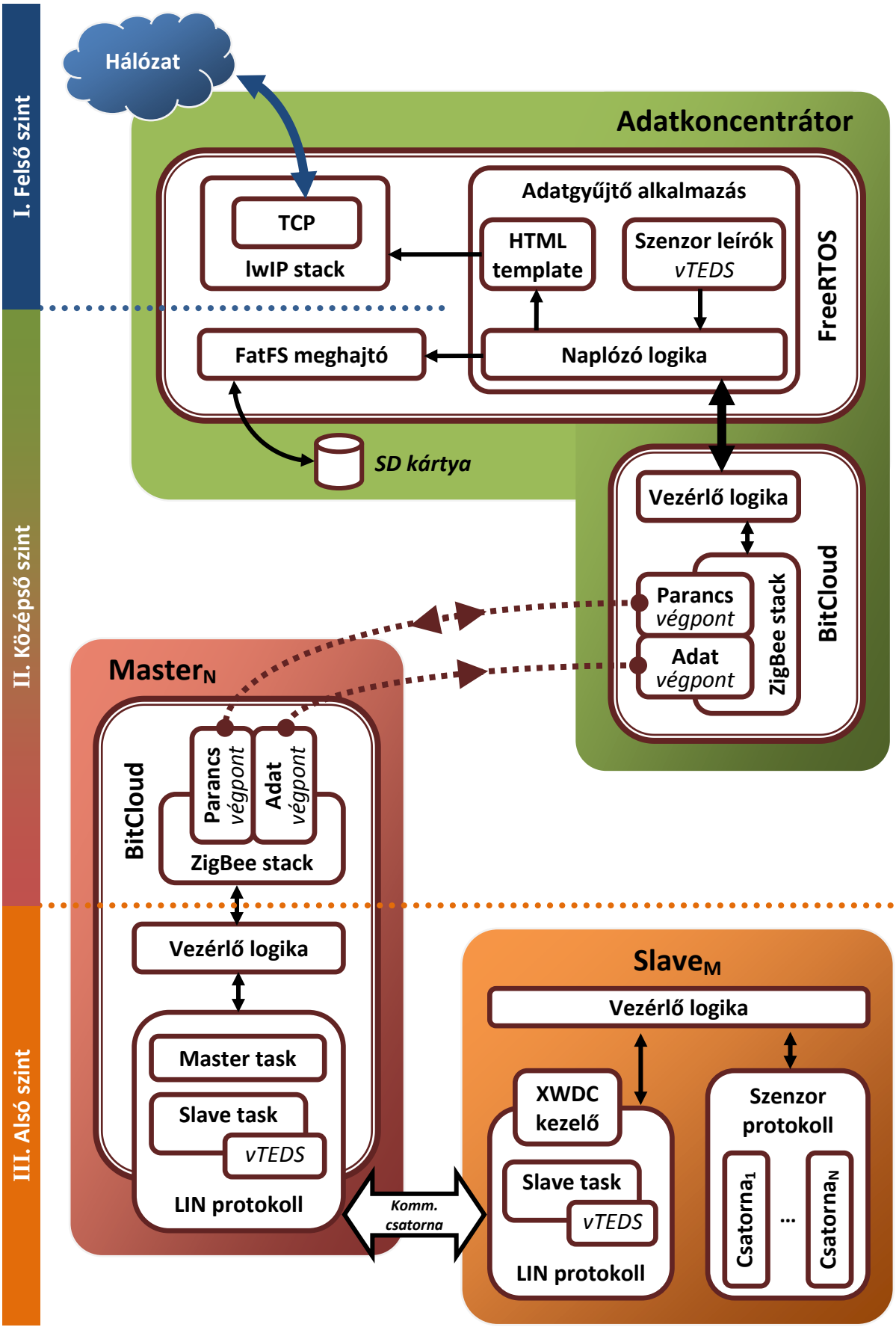
aktuátor funkcióval is felruházni. Megfelelő műszerdobozt választva az egység így képes lesz egy hálózati fogyasztó kapcsolására. A *Master* egységek és az *Adatkoncentrátor* közötti vezeték nélküli összeköttetést egy ZigBee modul biztosítja. Mivel ezek a modulok önállóan végzik a hálózati protokoll kezelését, minden esetben tartalmaznak egy viszonylag nagy számítási teljesítménnyel rendelkező mikrovezérlőt. A piacon találhatók olyan rádiós modulok, melyeknél ezen mikrovezérlők a hálózati kommunikáció által nem használt erőforrásai a felhasználó számára is szabadon elérhetők. Én is egy ilyen modult választottam a *Master* megvalósításához, ezáltal nincs szükség egy további mikrovezérlő alkalmazására a LIN protokoll kiszolgálásához. Ez jelentős alkatrész- és költségmegtakarítást tesz lehetővé. A fizikai LIN interfész kialakítása rendkívül egyszerű és olcsó áramköri szempontból, csupán egy külső buszmeghajtó perifériát igényel.

A *Slave* egységek a rendszer legprimitívebb egységei, a buszillesztésen és az adott szenzoron vagy aktuátoron kívül csak egy kis teljesítményű, a busz- és szenzorprotokoll kiszolgálására képes mikrovezérlőt tartalmaznak. A rendszerterven jól látható, hogy a LIN buszon kívül ezek az egységek egy úgynevezett *eXtra Wire Daisy Chain*-re (XWDC) is fel vannak fűzve. Ez a megoldás a LIN protokoll egyik javasolt kiegészítése, mely lehetővé teszi a felcsatlakozó moduloknak automatikus címkiosztását. A rendszer indulásakor egy token fut végig ezen a láncon egységről-egységre. A Slave-ek így ezáltal és a buszforgalom figyelésével automatikusan megállapíthatják a saját címüket. A módszer nagy előnye, hogy az egységek címei egyben a buszon elfoglalt fizikai pozíciójukat is jelöli. Ez a plusz információ nagy segítséget jelent a rendszer diagnosztikai szempontjából, például a hibás modulok keresésekor. Megvalósítása egy normál bemenet-kimenet páron kívül semmilyen különleges hardverelemet nem igényel. A blokkvázlaton nem tűntettem fel ezen egységeknél külön tápegység modult, melynek oka a következő: A LIN-t általában nagyon egyszerű és kis energiafelhasználású eszközök hálózatba szervezésére használják. Ebből kifolyólag a buszmeghajtó áramköröket gyártó cégek olyan típusokat is gyártanak, melyek integrálva tartalmaznak egy néhányszor 10 mA leadására képes egyszerű lineáris tápegységet. Mivel a megvalósítandó szenzorhálózatban nem lesznek nagy teljesítményfelvétellel rendelkező *Slave* egységek, én is egy ilyen buszmeghajtóba integrált tápegység megoldás alkalmazása mellett döntöttem.



14. ábra: Hardver rendszerterv

5.2. Szoftver rendszerterv



15. ábra: Szoftver rendszerterv

A szoftver rendszerterv bemutatását a legegyszerűbb belső logikával rendelkező *Slave* egységekkel kezdeném. Alapvető feladatuk a hozzájuk csatlakozó szenzor (vagy aktuátor) kezelése, az általa szolgáltatott mérési információk beolvasása és azoknak a szenzorhálózatba való juttatása. Nagyobb pontosságuknak és megbízhatóságuknak köszönhetően az analóg kimenetű szenzorokat egyre nagyobb mértékben váltják fel a digitális interfésszel rendelkező típusok. Ennek következtében a mérőberendezések szoftvereiben is a jelfeldolgozás helyett, a szenzorprotokoll kiszolgálásának a feladata került előtérbe. A *Slave* egységekben implementálásra kerül a hozzájuk csatlakozó szenzort kezelni képes meghajtó szoftver. Egy szenzor akár több csatornával is rendelkezhet, a fejlettebb klimatikai szenzorok például hőmérséklet és páratartalom mérésére is alkalmasak. Ezen csatornák független kezelése a meghajtó által biztosított, a *Slave* által kezelt csatornaszám a felsőbb rétegek számára lekérdezhető. Az információcsere a rendszer többi részegységével egy LIN alapú kommunikációs csatornán keresztül valósul meg. Az egységek ezért rendelkeznek egy a LIN protokollt kiszolgáló szoftverkomponenssel. A LIN specifikáció ennek megvalósításához tartalmaz egy API ajánlást, mellyel a busz kommunikációs és az egység vezérlő logikája egy szabványos interfészen keresztül elválasztható egymástól. A *Vezérlő logika* teremti meg a kapcsolatot a szenzor meghajtó és a kommunikációs szoftverkomponens között. A busz irányítása nem a *Slave*-ek feladata, ezért ezek az egységek csak az úgynevezett *Slave task*-ot futtatják, mely a *Master*-től érkező kérésekre ad automatikus választ. Már a tervezés korai szakaszában elhatároztam, hogy a szenzor-egységeket fel fogom ruházni TEDS képességgel. Ez egyfajta digitális adatlap, melynek segítségével az adott szenzor képes a rendszert informálni az általa biztosított funkciókról (mért paraméterek, mérési tartomány, pontosság, stb.). Sajnos egy ilyen adatlap nagy memóriaigényt támaszt az egységgel szemben, ami azonban ellent mond a költséghatékonysági célkitűzésemnek. Ezért az iparban is egyre gyakrabban alkalmazott virtuális TEDS (*vTEDS*) megoldást választottam. Ennek az a lényege, hogy az adott szenzoregység csupán egy gyártó és funkció azonosítót tárol, melyek kiolvasása után az általában hálózati kapcsolattal rendelkező feldolgozó egység az Internetről tölti le a beazonosított szenzorhoz tartozó adatlapot. A *Slave* egységek azonosításához én a LIN diagnosztikai protokollja által biztosított lehetőségeket használom ki, vagyis ez a megoldás

semmilyen további szoftver-komponenst nem igényel. Az egységeknek a LIN buszon való fizikai elhelyezkedésük automatikus megállapítást lehetővé tevő XWDC a LIN protokoll egy ajánlása, azonban sajnos egyenlőre nem szabványosított eleme. Ennek következtében az implementálásához a LIN vezérlő logikát egy saját komponenssel kell kiegészítenem, ami a LIN diagnosztikai protokolljával együttműködve fogja megvalósítani a kívánt funkciót.

A *Master* egység a ZigBee protokollnak megfelelő vezeték nélküli hálózati kommunikációt fog folytatni. Ez egy magas szintű protokoll, melynek kiszolgálása egy akár egyszerűbb beágyazott rendszerekbe szánt operációs rendszer használata nélkül csak jelentős kompromisszumok árán lenne megvalósítható. Nem meglepő, hogy a választott ZigBee modul is egy egyszerű, a gyártó saját fejlesztésű esemény vezérelt operációs rendszerét futtatja, mely a *BitCloud* fantázianevet kapta. Az operációs rendszer réteges szerkezetű, melynek elemi része maga a ZigBee protokoll. Erre ráépülve a legfelső szinten találhatók a felhasználói alkalmazások, jelen esetben az általam megírt vezérlő logika és az általam port-olt LIN meghajtó. A *Master* felel a LIN busz vezérléséért, ezért a meghajtó szoftver-komponensben már az ütemezési táblák kezeléséért és az üzenetek fejléceinek generálásáért felelő *Master task* is megtalálható. A hardver rendszertervben bemutattam, hogy ez az egység is fel lett ruházva egy aktuátor szerepkörével, ezért ennek megfelelően egy *Slave task*-ot is futtat és természetesen *vTEDS* képességekkel is rendelkezik. A *Vezérlő logika* biztosítja az adatáramlást a LIN buszra csatlakozó eszközök és a vezeték nélküli hálózat között, vagyis itt valósul meg az alsó és a középső hierarchiaszint szoftveres szétválasztása. A ZigBee protokollban az adatcsere nem az egyes egységek között, hanem az azokban szoftveresen definiált végpontok között történik, adatátvitelkor ezek virtuális összekapcsolódnak. Nekem kettő definiálására volt szükségem a szenzorhálózathoz: egy *Parancs* és egy *Adat* végpontra. A *Parancs* egy kétirányú végpont, mely a felhasználótól az *Adatkoncentrátor*-on és a *Master*-en keresztül a *Slave* egységekhez küldött parancsok (pl. mérési felbontás beállítása) eljuttatására és a vezetékes hálózat inicializálásakor a szenzoregységek bejelentésére szolgál. Az *Adat* végpont ezzel szemben csak egyirányú, a mérési adatok eljuttatására szolgál az *Adatkoncentrátor* egységhez. A kommunikáció lebonyolítása teljes mértékben a ZigBee protokoll feladata, az adatátvitel a felhasználó számára transzparens módon megy végbe.

Az *Adatkoncentrátor* két különálló központi egységgel rendelkezik. Az egyik a *Master* egységben is megtalálható ZigBee modul, mely saját operációs rendszert futtat és jelen esetben csak a vezeték nélküli kommunikáció lebonyolításáért felel. A másik központi egység egy 32-bites nagy számítási teljesítményű mikrovezérlő. A több párhuzamos bonyolult feladat kiszolgálása, a hatékony erőforrás kihasználás és a gyors reakcióidő biztosítása csakis egy operációs rendszer alkalmazásával volt megvalósítható. Én a kimondottan beágyazott rendszerekbe szánt, nyílt forráskóddal rendelkező *FreeRTOS* rendszert választottam. Ez az ingyenes kernel minden olyan funkciót biztosít, ami különféle task-ok hatékony párhuzamos futtatásához szükséges (szemafor, mutex, queue, stb.). A rendszer egy szenzorhálózat, tehát legfontosabb feladata a szenzoroktól érkező adatok fogadása, naplózása. Ezt az *Adatgyűjtő* szoftverkomponens fogja megvalósítani. Fogadja a vezeték nélküli perifériától érkező adatokat és a szenzor leírók segítségével a megfelelő formátumban rögzíti azokat. Ez az utóbbi adatbázis tartalmazza a szenzorok egyedi funkció azonosítóinak és a részletes adatlapjainak az összerendelését (*vTEDS*). Mivel a tárolni kívánt adatmennyiség jelentős mértékű is lehet, ez egy külső adattárolón valósul meg (*SD kártya*). Annak érdekében, hogy a rögzített adatok a későbbiekben más rendszereken is könnyedén feldolgozhatók legyenek, azok nem nyers formában, hanem FAT fájlrendszer használatával kerülnek elmentésre. Ehhez a szintén szabadon hozzáférhető FatFS meghajtó csomagot választottam. Ez gyakorlatilag az SD kártya alacsony szintű kezelésétől a fájlrendszer legfontosabb funkcióinak biztosításáig minden feladatot ellát. A rendszer fontos követelménye volt, hogy a begyűjtött adatok egy magas szintű hálózat felé is továbbítani lehessen. Ethernet-en keresztül az Internetre történő adattovábbítás számos protokoll implementálását megköveteli. Szerencsére ilyen protokollkészletből több is szabadon hozzáférhető, én a kimondottan beágyazott rendszerekbe szánt *lwIP stack* használata mellett döntöttem. Ez csak a legszükségesebb protokollokat (TCP, UDP, ICMP, DHCP, stb.) tartalmazza, így rendkívül kedvező memóriaigénnyel rendelkezik. Ha egy kliens szeretné lekérni a naplózott adatokat, akkor az adatgyűjtő alkalmazás feltölt egy HTML csontvázat az aktuális szenzorinformációkkal és továbbítja azt az *lwIP stack*-nek. Ez végül TCP protokoll használatával eljuttatja azt a célállomáshoz.

VI. Hardvertervezés

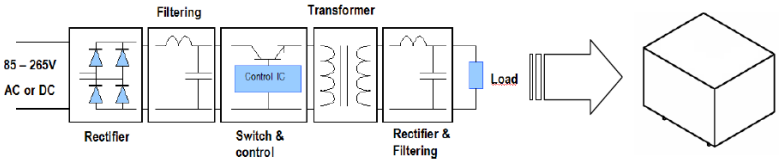
Ebben a fejezetben az adatgyűjtő-vezérlő hálózati rendszer számára megtervezett hardverkomponenseket szeretném bemutatni. A komponenseket részegységekre bontom és úgy ismertetetem az alkalmazott áramköri megoldásokat.

6.1. Master egység

A hálózati rendszer középső és alsó hierarchia szintjei közötti átjárást biztosítja. A középső szint felé egy ZigBee alapú vezeték nélküli interfésszel rendelkezik. Ebben egy Router szerepkörét látja el, vagyis a saját adatainak továbbításán kívül jelismétlő eszközként is funkcionál a hálózati hatótávolságának kiterjesztése érdekében. Hozzá a különféle szenzor és aktuátor egységek a hálózati rendszer alsó szintjén egy LIN alapú buszon keresztül csatlakozhatnak. Ebben a vezetékes hálózatban a Master szerepét tölti be, innét az egység elnevezése is.

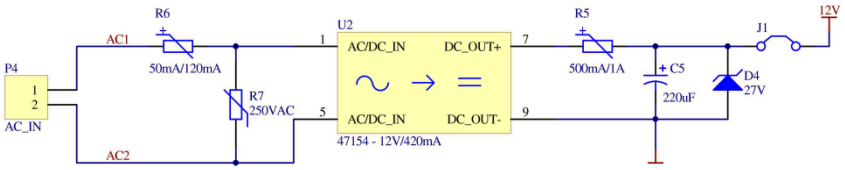
6.1.1. Hálózati tápegység

A rendszer elemmentes kialakítása érdekében a Master hálózati táplálással rendelkezik. A tervezés korai szakaszában mérlegelnem kellett, hogy mi a legjobb módja az egység a 230V-os hálózatra történő csatlakoztatásának. Az egyik lehetőség, hogy csak egy kisfeszültségű tápcsatlakozóval láttam volna el az egységet és ezáltal az egy külső gyári tápegységet igényelt volna a működéshez. Minden bizonnyal egy kísérleti eszköznél ez lett volna az optimális megoldás, azonban a cél egy piacképes termék megtervezése volt. Az esztétikai szempontokat is figyelembe véve, végül egy közvetlenül a hálózati aljzatba csatlakoztatható műszerdobozzal rendelkező egységet képzeltem el. A kérdés azonban ebben az esetben is felmerült, hogy milyen tápegységet célszerű kialakítani, hiszen méretbeli és életvédelmi szempontok is figyelembe kellett venni. Körülnéztem a piacon, hogy milyen tápegység megoldások érhetők el és így találtam rá a *myrra* cég kompakt hálózati tápegység kockáira.



16. ábra: myrra gyártmányú tápegység kocka blokkvázlata [20]

Ezek a Flyback topológiájú kapcsoló üzemű tápegységek minden elemet integrálva tartalmaznak ami szükséges a váltakozó áramú hálózati feszültségből a stabilizált egyenfeszültség előállításához. Többféle kimeneti teljesítménnyel és feszültséggel rendelkező változatban érhetők el. Rendelkeznek kimeneti túláram és túlmelegedés elleni védelemmel is. Az áruk viszonylag magas, azonban ha megvizsgáljuk, hogy körülbelül mekkora költségből hozható ki egy hasonló képességű, diszkrét komponensekből felépített tápegység, rögtön megfizethető alternatívává válnak ezek az alkatrészek. Nem is beszélve az általuk garantált kompakt design előnyeiről. Én is egy ilyen tápegység kockát választottam a Master egység tápegységeként.

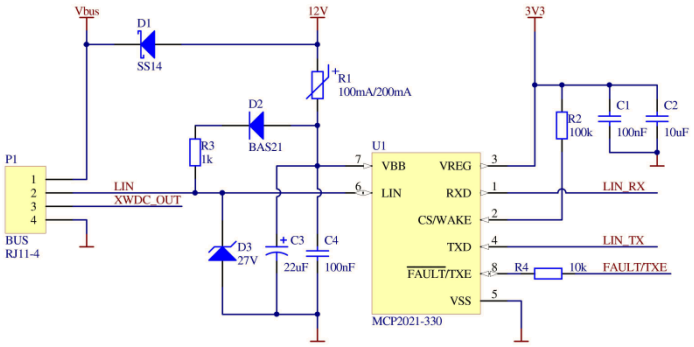


17. ábra: A Master egység hálózati tápegysége

A tápegység kocka bemenetének túlfeszültségvédelme érdekében vele párhuzamosan kötöttem egy 250V-os névleges feszültségű varisztort. Ennek ellenállása túlfeszültség esetén jelentősen lecsökken, aminek hatására a bemenettel sorba kapcsolt biztosíték leválasztja azt a hálózatról. A választott biztosíték nem egy normál olvadó biztosító, hanem egy PTC típusú, vagyis automatikus regenerálódó fajta. Ezek az eszközök szobahőmérsékleten viszonylag kis ellenállással rendelkeznek, azonban túláram esetén a bennük keletkező hő hatására megváltozik az anyagi szerkezetük, ami az ellenállásuk drasztikus megugrásához vezet. A beavatkozási sebességük attól függ, hogy a hibaáram hányszorosa a névleges áramértéküknek, ezért a megfelelő típus kiválasztására kiemelt figyelmet kell fordítani. A fenti kapcsolásban két ilyen regenerálódó biztosíték is található. A bemeneti PTC 50 mA-es áramot képes üzemszerűen átengedni, efölött fokozatosan elkezd melegedni és a védelem működésbe lép. Az úgynevezett trip, vagyis levágási áramértéke 120 mA. A kimeneten is található egy ilyen eszköz, azonban az 27 V-os Zener-diódával kiegészítve az csak egy plusz védelmi vonalat jelent, hiszen a tápegység kocka tartalmaz kimeneti áramkorlátot és túlfeszültségvédelmet is. A nagy értékű puffer kondenzátor gondoskodik a kimeneti feszültség szűréséről, a jumper pedig az egység élesztése során a tápegység leválaszthatóságáról.

6.1.2. LIN busz illesztés

A Master-hez a Slave szenzor és aktuátor egységek LIN buszon keresztül csatlakoznak. Számos nagy chipgyártó kínálatában találhatók LIN busz illesztő áramkörök, én a kedvező árfekvése és elérhetősége miatt az MCP2021 [21] típusjelzésű Microchip gyártmányú IC-t választottam. Különlegessége, hogy integrálva tartalmaz egy lineáris feszültség stabilizátort is, ami szükségtelenné teszi egy további ilyen áramköri elem felhasználását. Kimeneti feszültség szempontjából fix 5 és 3,3V-os változatban érhető el és 50 mA-es áram leadására képes.



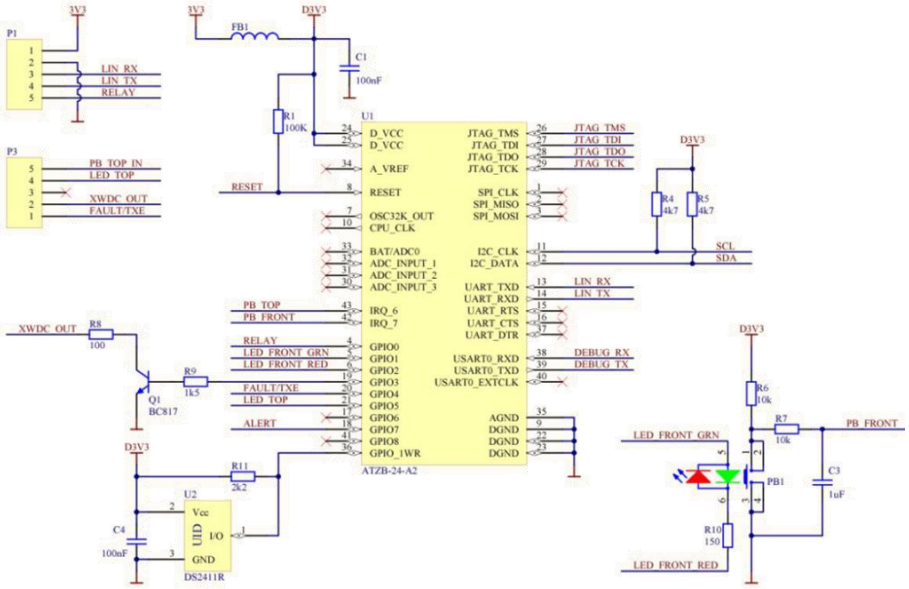
18. ábra: A Master egység LIN buszillesztő áramköre

A buszillesztő adatlapja számos kiegészítő védelmi elemet javasol, én ezek közül csak az aktuális alkalmazás számára relevánsakat valósítottam meg. A hálózati tápegység kimenete megtáplálja a buszt és egy regenerálódó biztosítékon keresztül magát a Master egységet. A kapcsolásban megfigyelhető a LIN szabvány által előírt Master oldali buszvonal felhúzó ellenállás. A buszillesztő 3 jelet biztosít a központi egység felé: küldés, fogadás és hibadetektálás. Ez utóbbi kétirányú, ugyanis a nyitott-drain-es meghajtásnak köszönhetően a központi egység is lehúzhatja ezt a vonalat, ezáltal letiltva a buszmeghajtó adó fokozatát. Az IC-ben található feszültség stabilizátor kimenete minimális szűrést követően rácsatlakozik a Master egység tápvonalára.

6.1.3. ZigBit modul

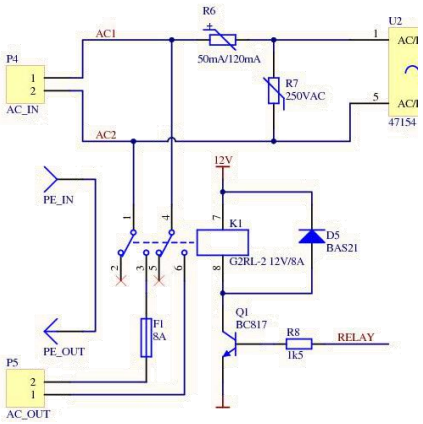
A Master egység számára választott ATMEL gyártmányú ATZB-24-A2 [22] típusú ZigBee modul különlegessége, hogy a hálózati protokollt kezelő mikrovezérlő nem kihasznált erőforrásai a felhasználó által szabadon hozzáférhetők.

A modulban egy 8 bites ATmega1281 található, mely rendkívül gazdag perifériakészlettel, nagy program- és adatmemóriával rendelkezik. Annak érdekében, hogy a Master egység a rádiós periféria tekintetében rugalmasabb legyen, ahogy a fenti képen is látszik a ZigBit modul egy külön panelon került elhelyezésre. Mivel a LIN buszmeghajtó és a tápegység áramkör egy csatlakozósoron keresztül kapcsolódik ehhez a kommunikációs perifériához, a későbbiekben akár pl. WLAN alapú vezeték nélküli szenzorhálózat is megvalósítható ugyanazon alapelemekkel felhasználásával.



19. ábra: A Master központi egysége és rádiós interfésze

A ZigBit modul két UART egységgel rendelkezik. Az egyiket a LIN buszon történő kommunikációhoz használtam fel, míg a másik az áramkör élesztése során nyújt majd egy hibakeresési csatornát. A kivezetett számos általános célú I/O portok közül csak néhányat láttam el funkcióval. A kapcsolási rajzon látható egy relé vezérlő jel. A Rendszerterv ismertetése során már említettem, hogy a Master aktuátorként is funkcionál majd. A műszerdobozán található hálózati ajzat érintkezőire egy relén keresztül kapcsolja rá a hálózati feszültséget. Az érintésvédelmi követelményeket szem előtt tartva a relé mind a fázis, mind a null vezetőt leválasztja az aljzatról. A védőföld megszakítás nélkül átkötésre kerül.



20. ábra: A Master aktuátor egysége

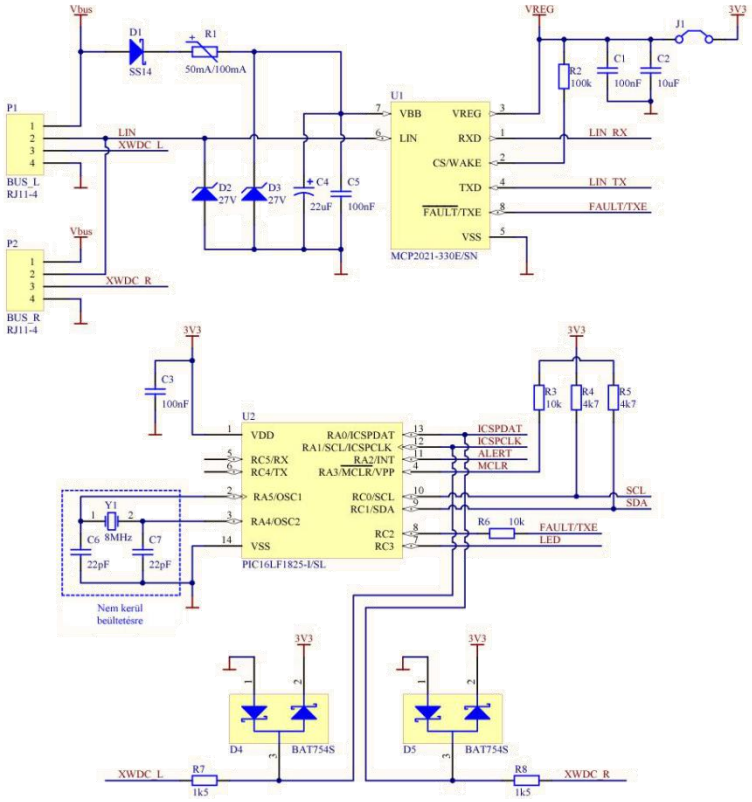
A Master egység emellett tartalmazni fog két-két darab nyomógombot és LED-et is. Ezek az aktuátor állapotának változtatására és a vezeték nélküli hálózat állapotának visszajelzésére fognak szolgálni. A ZigBee hálózatban fontos, hogy minden felcsatlakozó modul egyedi hálózati azonosítóval rendelkezzen. Ez az azonosító egyedileg megadható az egyes modulok felprogramozásakor, de sorozatgyártásra szánt elemeknél sokkal praktikusabb megoldás egy egyedi sorszámot tartalmazó chip (UID) alkalmazása. Mivel a ZigBit modul az 1 vezetékes kommunikációs interfészén keresztül erre közvetlen támogatást tartalmaz, a vezeték nélküli modulra egy ilyen chip is rátervezésre került, annak ellenére, hogy ez a fejlesztési fázisban nem feltétlenül kerül majd beültetésre.

6.2. Slave egység

Feladata a különböző szenzorok és aktuátorok a hálózati rendszerbe történő illesztése. Vezetékes buszkapcsolattal rendelkezik a felsőbb szintek felé és csak minimális számítási teljesítménnyel rendelkezik a digitális szenzorok és a busz-protokoll kiszolgálásához.

6.2.1. LIN busz illesztés és a központi egység

A Slave egységekben is a Master-nél bemutatott MCP2021-es áramkör végzi a LIN busznak megfelelő fizikai jelszint illesztést. A központi egység egy Microchip gyártmányú PIC16F1824 [23] típusjelzésű 8 bites mikrovezérlő. UART egységet is tartalmazó periféria készlete, alacsony lábszáma és kedvező árfekvése tökéletes választássá tette az egyszerű Slave-ek számára. Program- és adatmemóriájának mérete megfelelő a buszprotokoll kiszolgálásához, valamint a későbbi továbbfejlesztés lehetőségét is biztosítja. Emellett nem elhanyagolható az a tény sem, hogy jelentős tapasztalattal rendelkezem a PIC mikrovezérlőket alkalmazó áramkörök fejlesztése területén.



21. ábra: Slave egységek LIN busz illesztése és központi egysége

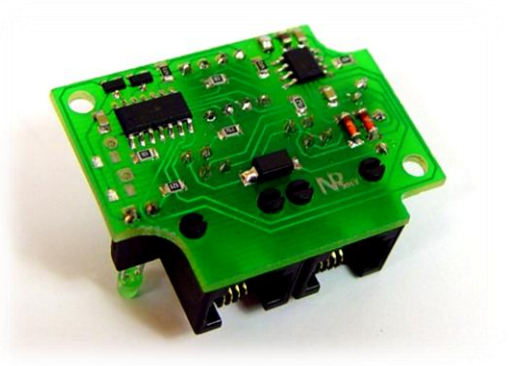
A Slave egységek automatikus címkiosztásához választott megoldást a hardver rendszerterv bemutatásakor már ismertettem. Az egységek busz-csatlakozóiként a széles körben alkalmazott RJ11-es csatlakozótípust választottam. Ez nem csak a fordított bedugást akadályozza meg, hanem a hozzá tartozó kábel szerelése is rendkívül egyszerű. Mivel ez a csatlakozó 4 érintkezővel rendelkezik, a LIN busz kiépítéséhez pedig összesen csak 3 jel szükséges (táp, jel, föld), a 4. érintkező felhasználhattam az eXtra Wire Daisy Chain kialakításához. Nem akartam, hogy az egységeken dedikált bemeneti és kimeneti csatlakozó legyen, ez ugyanis egy szignifikáns hiba-forrássá válna a rendszer telepítésekor. Ez szerencsére némi plusz programkóddal elkerülhető volt. A mikrovezérlő a rendszer indulásakor mind a két XWDC vonalat bemenetként konfigurálja és figyeli, hogy melyik oldalról érkezik meg hozzá a token. Ezzel megállapítja, hogy melyik oldalán található a Master egység, majd a másik vonalát kimenetként konfigurálva továbbküldi a token-t.

Az áramkörtervezés során kiemelt figyelmet fordítottam arra, hogy az egységek összes külvilág számára is elérhető jelei a lehető legnagyobb mértékben védve legyenek a hibás jelszintek ellen. Hibás működés esetén például az XWDC vonalain található a soros áramkorlátozó ellenállások védik

meg az egymást szembehajtó mikrovezérlők kimeneti fokozatait, a túlfeszültség védelemről pedig a kis nyitófeszültségű vágó Schottky-diódák gondoskodnak.

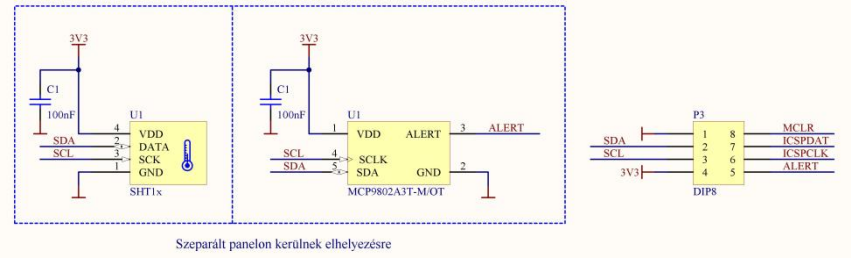
Megfigyelhető a fenti kapcsolási rajzon, hogy a mikrovezérlő külső kvarc oszcillátora csak opcionális elemként szerepel a tervben. Annak footprint-je ugyan rá lesz tervezve a panelre, de az alkatrészek nem kerülnek majd beültetésre. Erre a LIN busz által biztosított szinkronizációs funkció miatt van lehetőség. A mikrovezérlő belső RC-oszcillátora ugyan nem olyan pontos mint egy külső kvarc, de rövid távon a szinkronizáció után képes biztosítani az aszinkron kommunikációhoz szükséges stabilitást.

6.2.2. Hőmérséklet érzékelő



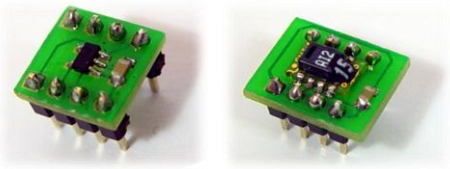
A HOLMS rendszer elsősorban lakások klímájának monitorozására készül, ezért a legelső Slave amit megterveztem egy hőmérséklet, illetve páratartalom mérésére szolgáló egység volt.

Hőmérséklet érzékelő szenzorokból is hatalmas a választék a piacon, én a minél egyszerűbb és megbízhatóbb működés érdekében mindenképp egy digitális kimenettel rendelkező változatot kerestem. A választásom végül az MCP9800 [22] típusjelű szenzorra esett, mely rendkívül kedvező ára ellenére szobahőmérsékleten 0,5°C-os pontosságot biztosít 12 bites felbontás mellett. I^2C kommunikációt használ, aminek kezelése nem jelent problémát a választott mikrokontroller számára.



22. ábra: Hőmérsékletérzékelő szenzorok

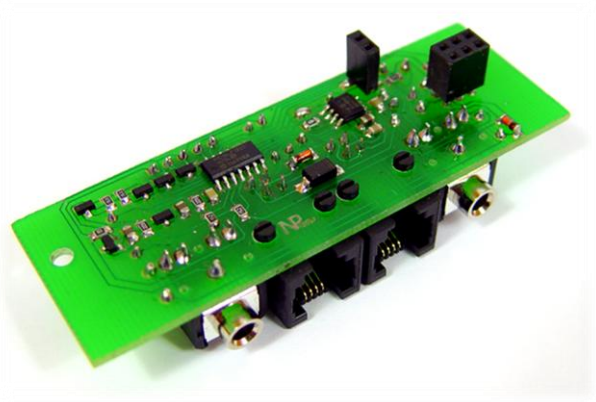
Annak érdekében, hogy a hőmérséklet érzékelő minél pontosabb mérési eredményeket szolgáltathasson egy külön panelon került elhelyezésre. Ez biztosítani fogja az áramkör többi elemétől történő termikus leválasztását.



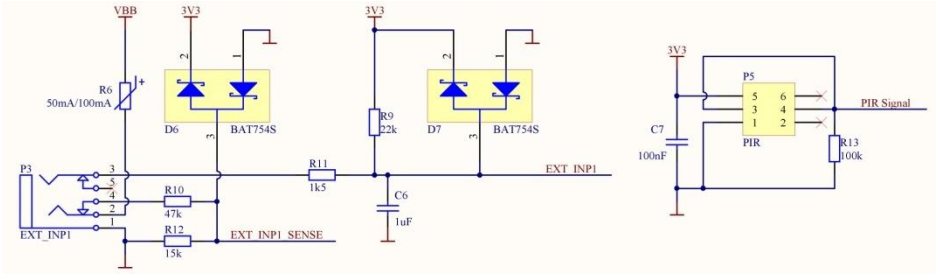
23. ábra: MCP9800 és SHT1x szenzorokhoz készített adapter panelek

A megvalósításhoz definiáltam egy 8 érintkezős (DIP8) csatlakozót, melyen a táp, föld és I^2C jeleken kívül egy további általános célú vonal is helyet kapott. A rendszerhez ezáltal bármilyen I^2C kommunikációs interfésszel rendelkező szenzor illeszthetővé vált. A munkám során a MCP9800-as hőmérséklet érzékelő mellett a tanszéken gyakran használt SHT1x hőmérséklet és páratartalom szenzorhoz is készítettem adapter panelt.

6.2.3. PIR mozgásérzékelő



Lakásmonitorozó rendszerek szempontjából egy másik fontos szenzorcsoportot képviselnek a mozgásérzékelők. Ezeknek a manapság legjobban elterjedt fajtái a PIR, vagyis passzív infravörös szenzorok. Előnyük, hogy csak a hőforrások - mint például az emberi test - mozgását érzékelik, ezáltal elkerülve a például a szél által megmozdított függöny által generált téves riasztást. Én is egy ilyen szenzort választottam, bár mivel a kimenetük a legtöbb mozgásérzékelőnek azonos, az gyakorlatilag bármilyen más típussal helyettesíthető.



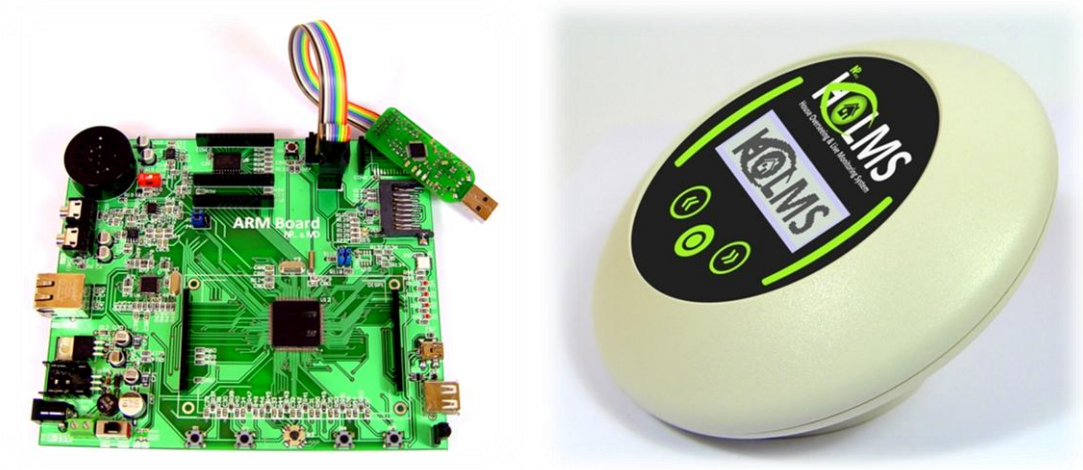
24. ábra: PIR mozgásérzékelő és a modul egyik külső kontaktus bemenete

Hasznosnak találtam ha kiegészítem a mozgásérzékelő Slave egységet két további külső kontaktus bemenettel is. Ezekhez bármilyen külső szenzor, például ajtó vagy ablak nyitásérzékelő csatlakoztatható, ezáltal a felhasználó nem

kényszerül csak ezért további egységek beszerzésére és telepítésére. Annak érdekében, hogy ezen bemenetek táplálást igénylő szenzorokat is fogadhassanak a tápfeszültség is kivezetésre került. Minimum 3 érintkezős csatlakozóra volt szükségem. A gyors és egyértelmű installáció miatt semmiképpen sem akartam ugyan olyan RJ-11-es csatlakozót használni mint a buszcsatlakozásnál, ezért végül egy audio készülékekben gyakran alkalmazott sztereó jack aljzatot választottam. A telefoncsatlakozóhoz hasonlóan ez is könnyen beszerezhető és szerelhető. Létezik olyan kialakítású változata, melyben a két jelvezeték a dugó bedugásakor automatikusan átvált a belső forrásról a külső jelforrásra. Ezt kihasználva sikerül egy csatlakoztatás-észlelés funkciót megvalósítanom. Így a legfelső szinten az egység csak pontosan annyi csatornáját fogja bejelenteni, mint amennyi éppen használatban is van. Mivel a felhasználó ezekhez a bemenetekhez is könnyedén hozzáfér, gondoskodnom kellett azok megfelelő védelméről. Regenerálódó biztosítékok segítségével védve vannak rövidzárlat és a túláram ellen, valamint a túlfeszültség védelmük is biztosított.

6.3. Adatkoncentrátor egység

A rendszer legnagyobb számítási teljesítményű egysége egy ST gyártmányú 32 bites Cortex-M3 maggal rendelkező ARM architektúrájú STM32F207-es mikrovezérlő [25] köré felépített egység. Azért esett a választásom erre a kontrollerre, mert egy korábbi félévben az egyik csoporttársammal közösen terveztünk ehhez a mikrovezérlőhöz egy demó kártyát. Ezen minden olyan periféria megtalálható, mint amit az aktuális rendszertervben is rögzítettem, ezért a tervezés során ezt a kártyát használhattam referenciaként.



25. ábra: ARM demó kártya és a kompaktabb kialakítású Adatkoncentrátor egység látványterve

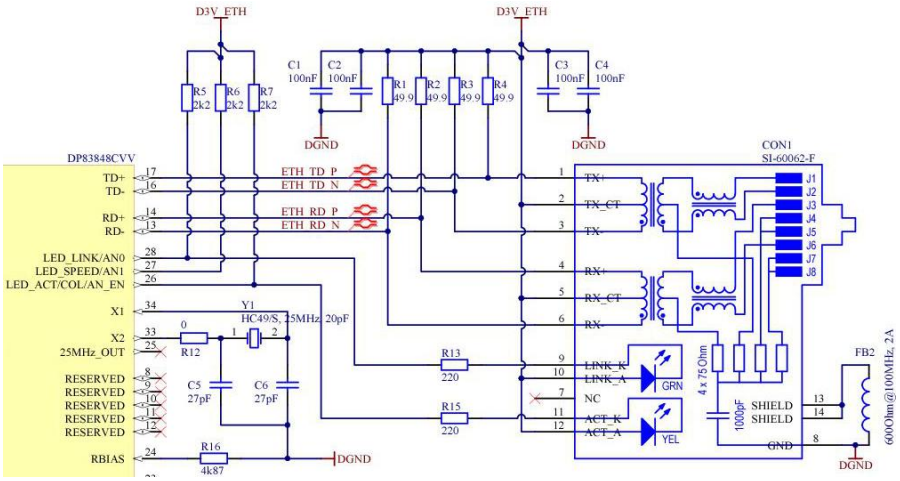
A következőkben az Adatkoncentrátor a feladatom szempontjából fontos két részegységét, az Ethernet és az SD kártya interfész áramkörét szeretném bemutatni. Ezek mellett az Adatkoncentrátor rendelkezik egy egyszerű lineáris tápegységgel, LIN busz illesztő áramkörrel és a felhasználói interfészt megvalósító monokróm grafikus kijelzővel és nyomógombokkal.

6.3.1. Ethernet interfész

Az *Adatkoncentrátor* egyik legfontosabb áramköri eleme az Ethernet interfészt biztosító egység. A választott mikrovezérlő rendelkezik beépített Ethernet perifériával (MAC), az interfész kialakításához csak egy külső fizikai jelszintillesztő áramkörre (PHY) van szükség. Én a National Semiconductors által gyártott DB83848C típusjelzésű transcievert [26] választottam, ami egy rendkívül széles körben alkalmazott típus. Ez nagy előnyt jelent, hiszen a legtöbb TCP/IP stack számára elérhető az ehhez az IC-hez elkészített alacsony szintű driver, többek között az általam választott lwIP stack alá is. Minimális számú külső komponenst igényel, a részegység minél kompaktabb kialakíthatósága érdekében én csatlakozóból és leválasztó transzformátorból is egy egybeintegrált típus választottam. A DP83848C és a mikrovezérlőben található MAC között kétféle interfésztípus kialakítására van lehetőség. Ezek a Media Independent Interface (MII) és Reduced Media Independent Interface (RMII). Funkcionalitás tekintetében a két típus teljesen azonos, kizárólag a szükséges jelszámban van eltérés. Az IC-vel elérhető maximális 100 Mb/s kommunikációs sebességhez a MAC és PHY között 25 MHz-es órajelű adatátvitelre van szükség. Mivel az RMII 4 RX és 4 TX jel helyett csak fele annyit használ, a szükséges órajelet is meg kell duplázni. Ez a nagyobb jelsebesség azonban már jelentős nagyfrekvenciás zavarforrást is jelent, ezért rendkívül gondos tervezést igényel az áramköri lap huzalozása során. A választott mikrokontrolleren az összes periféria bekötéséhez elegendő I/O port-tal rendelkezik, ezért nem láttam szükségességét az RMII interfész alkalmazásának. Az MII mellett a MAC és a PHY között egy Serial Management Interface is található, ami az induláskor biztosítja, hogy a két egység „megegyezhessen” a működési üzemmódban.

Egy ilyen nagysebességű periféria érzékeny a megfelelő huzalozásra, ezért erre kiemelt figyelmet kellett fordítanom a NYÁK tervezés során. Mivel az

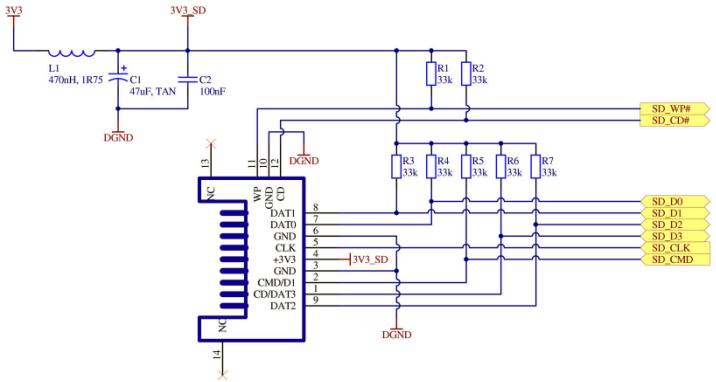
Ethernet a fizikai rétegében differenciális érpárokat használ (TD+/- és RX+/- vonalpárok), biztosítani kellett ezen vezetékek 100 Ohm-os impedancia illesztését a csatlakozó és a PHY között. A szükséges vezeték-távolság és vastagság meghatározásához a gyártás során is használt FR4-es panel paramétereire támaszkodtam. Szerencsére a differenciális huzalozáshoz a tervezés során használt Altium Designer program natív támogatást nyújtott.



26. ábra: Differenciális Ethernet adatvonalak a PHY és a leválasztó transzformátor között

6.3.2. SD kártya interfész

A memóriakártyát a mikrovezérlőhöz szabványos 4-bites interfészen keresztül illesztettem, mert ezt biztosítja a leggyorsabb adatátviteli sebességet. Az SD kártya alacsony szintű kezeléséhez ez a nagy tudású kontroller tartalmaz belső perifériát. A DMA-val együttműködve például képes nagyméretű adatblokkok automatikus átvitelére a kártya és a kontroller között. Az *SD_CD* és *SD_WR* alacsony-aktív jelek a kártya behelyezéséről és írásvédettségéről informálják a processzort.



27. ábra: 4-bites SD kártya interfész

VII. Szoftverfejlesztés

A HOLMS rendszer elkészítése közben számos különféle szoftverplatformra kellett fejlesztenem. Kezdve az egyszerű 8-bites mikrovezérlők hardverközeli programozásától egészen a több ütemezett feladatot futtató prioritásos beágyazott operációs rendszerekig. A HTML struktúra definiálásakor emellett a web programozás területét is érintettem. Ebben a fejezetben nem szeretnék minden szoftverfejlesztési lépést részletesen ismertetni, helyette inkább a kigondolt algoritmusok és a szoftverplatformok sajátosságainak bemutatására helyezném a hangsúlyt.

7.1. LIN protokoll megvalósítása

A munkám során a LIN specifikáció csomag 2.1-es verziójával [8] dolgoztam. A protokollt megvalósító meghajtó program egy német cég [17] jóvoltából a rendelkezésemre állt. Tőlük hallgatói licence alapján kaptam egy olyan programot, amely képes a LIN buszon üzemelő egységek paramétereit tartalmazó LDF (LIN Description File) fájl felhasználásával a szükséges C nyelvű forráskód legenerálására. A meghajtó által biztosított funkciókat a LIN szabványban rögzített API-n keresztül érhettem el.

A LIN protokoll segítségével egy rendkívül megbízható, garantált reakcióidővel rendelkező hálózati kommunikáció alakítható ki. Ez elsősorban a determinisztikus működésnek köszönhető, ugyanis minden üzenet (frame) előre rögzített időkerettel rendelkezik. Ezen ütemezés definiálásához szükséges a buszt használó teljes rendszer előzetes ismerete, melynek adatait az LDF fájl tartalmazza. Ebben megtalálhatók a buszra csatlakozó egységek adatai, az általuk generált frame-ek és az összes ezekben továbbított szignál (különböző bithosszúságú adatmezők). Ez a statikus működési szemlélet rendkívül komoly kihívást jelent a feladatom szempontjából. Az én célom éppen egy dinamikusan bővíthető szenzorhálózat megvalósítása, melyhez bármikor egy új szenzoregység csatlakoztatható vagy éppen eltávolítható. Nem tudom előre, hogy összesen hány egység fog csatlakozni a buszra. A szignálok előzetes rögzítésére sincs lehetőségem, hiszen azt sem tudom előre, hogy melyik egység lesz például csak egy 1-bites vezérlőjelet igénylő aktuátor vagy egy több bájtos adatstruktúrát elküldő hőmérséklet szenzor. Szerencsére a 2.0-s verziótól kezdve a LIN ki lett

bővítve különféle diagnosztikai funkciókkal, melyek számos paraméter dinamikus beállítását teszik lehetővé. A diagnosztikához a Master oldalon külön ütemezési tábla tartozik, melyben a rögzített Frame ID-val rendelkező Master Request és Slave Response Frame-ek biztosítják az egységek közötti adatcserét. A buszra csatlakozó eszközök ilyenkor nem a Frame ID-k alapján állapítják meg, hogy ők-e az adott üzenet címzettjei, hanem ehhez saját egyedi hálózati címazonosítójukat (NAD) használják. A LIN diagnosztikai rétege által a feladatom szempontjából legfontosabb funkciók a következők [12]:

- **Assign NAD via SNPD** (Slave Node Position Detection)
  → Az eszköz egyedi NAD azonosítója akár a működés során is megváltoztatható, ha a hálózat ki van egészítve a LIN által ajánlott egyik automatikus Slave pozíció detektálást biztosító megoldással
- **Assign frame identifier range**
  → Az adott Slave-hez tartozó Frame-ek az ID-k megadásával módosíthatók
- **Read by Identifier**
  → Ez a parancs lehetővé teszi a Slave-ek egyedi paramétereinek (pl. gyártó- és funkcióazonosítójának) lekérdezését

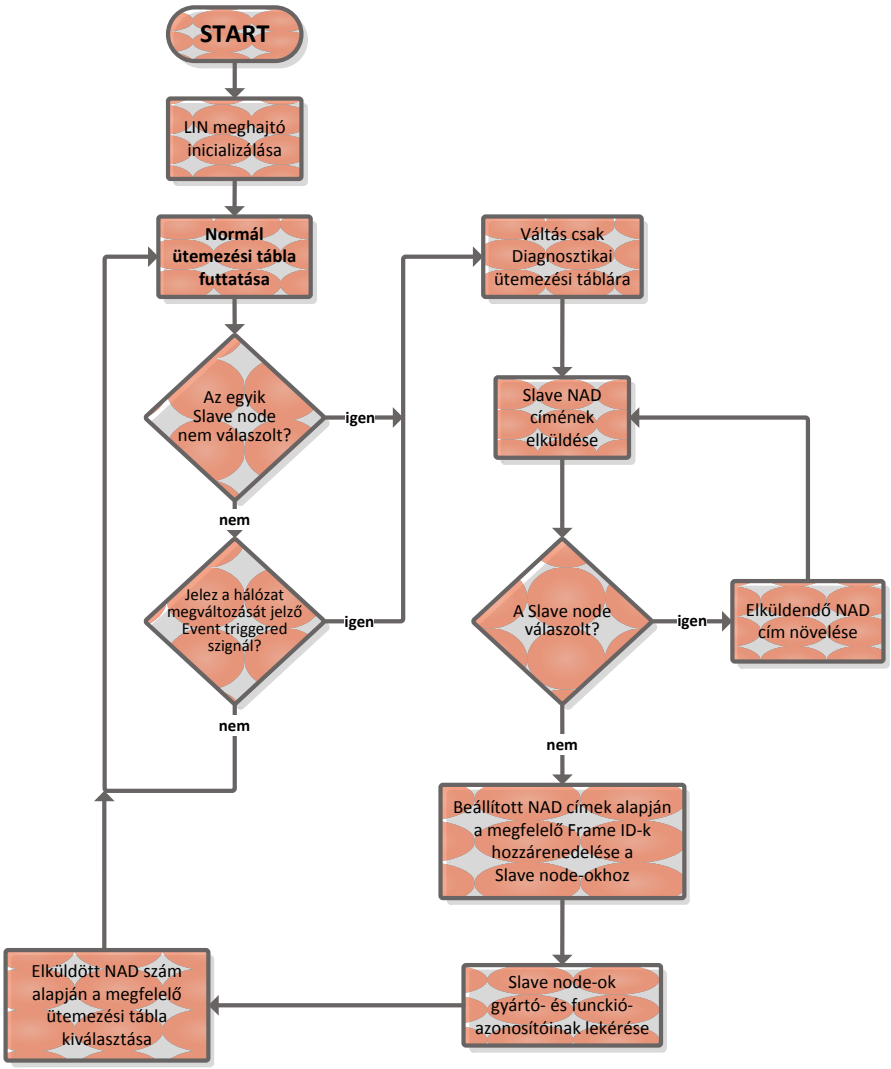
A feladatom szempontjából a LIN protokollal kapcsolatos legnagyobb kihívás tehát az volt, hogy a rendszer indulásakor hogyan tudom a statikusként elvárt paramétereket a diagnosztikai szolgáltatásokkal dinamikusan beállítani. A következő pontokban a Master és Slave egységek működésének folyamatábráit szeretném bemutatni, a rendszer bekapcsolásakor végbemenő konfigurációs teendőkre koncentrálva.

7.1.1. A Master egység inicializálásának folyamata

A Master feladata a busz vezérlése, ezáltal a diagnosztikai funkciók elvégzése is. Alapvetően egy normál ütemezési táblát futtat, melyben az adott Frame-ek előre rögzített időkeretekben kerülnek lekérdezésre a Slave-ektől. Azt nem tudjuk előre, hogy egy adott Slave milyen szenzorral vagy aktuátor funkcióval rendelkezik, ezért úgy döntöttem, hogy minden Frame a LIN által maximálisan biztosított 8 adatbájtot fogja tartalmazni, melyek tartalmát és

adatstruktúráját a szenzor típusa fogja meghatározni. A feldolgozásra majd csak az Adatkoncentrátor egységben kerül sor, ahol viszont már ismert lesz az adott szenzor típusa és ezáltal a tőle érkező adat struktúrája is. Sajnos nem ismert előre a Slave-ek és így a szükséges Frame-ek száma sem. Erre az jelentette a megoldást, hogy minden lehetséges Slave számmal rendelkező hálózati konfigurációhoz saját ütemezési táblát definiáltam. A LIN protokoll 15-ben limitálja ezt a számot, vagyis összesen ennyi ütemezési táblát kellett megadnom. Ez a szükséges memóriaméret szempontjából nem jelentett problémát.

A következő folyamatábra szemlélteti a Master egység LIN szoftverkomponensének működését. A hálózat újrainicializálását vagyis a paraméterek dinamikus beállítását egy Slave node elmulasztott válasza (kiesett a hálózatból) vagy egy új egység csatlakoztatásakor megválaszolt esemény-vezérelt Frame megválaszolása válthatja ki.

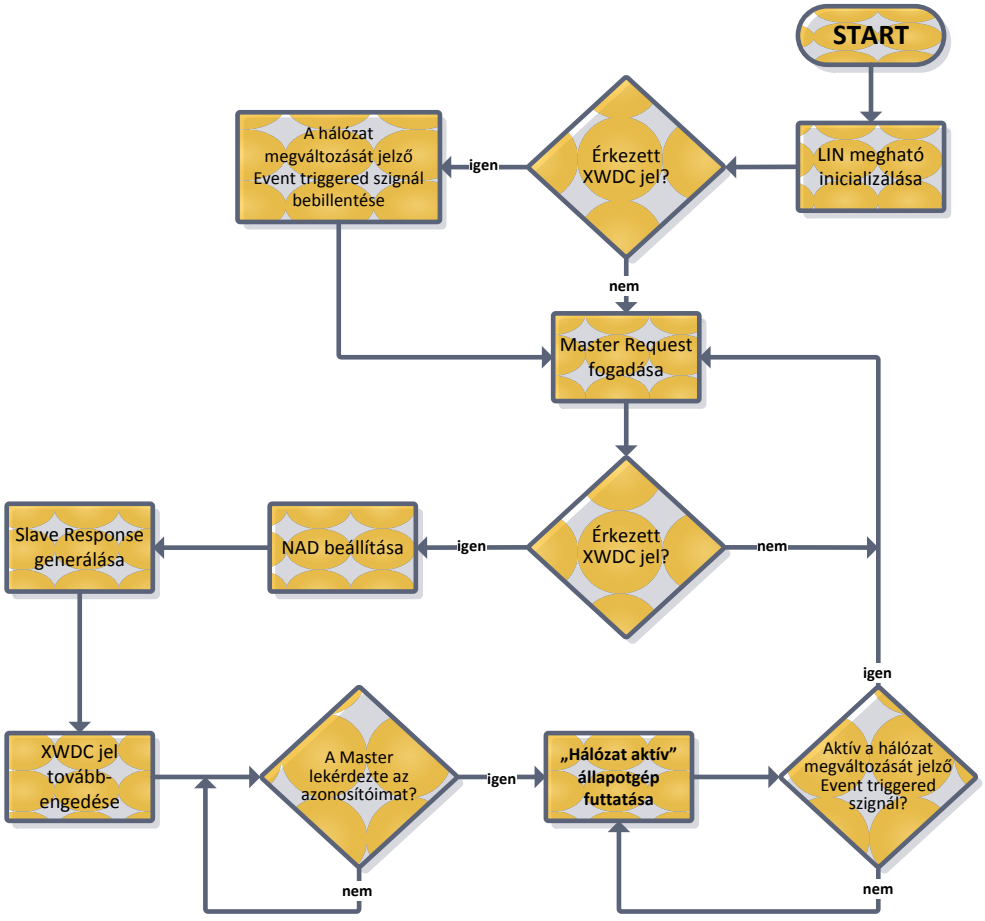


28. ábra: A Master egység LIN inicializálásának folyamatábrája

Újrainicializálás esetén a Master automatikusan átvált a diagnosztikai üzenetek továbbításához szükséges ütemezési táblára. Egy számlálót értékét fokozatosan növelve egymás után küldi ki a soron következő Slave-hez hozzárendelni kívánt egyedi hálózati címet (NAD). Arról, hogy ezt mindig a megfelelő Slave dolgozza fel, az XWDC jel végigfutása biztosítja a hálózaton. Ha egy címbeállítási kérelemre nem kapott a Master választ, akkor ez azt jelenti, hogy nem csatlakozik több Slave a hálózathoz (vagy a soron következő Slave hibás, így az megszakítja a láncot). A címkiosztást követően a megfelelő Frame ID-k hozzárendelése az adott Slave-hez és a megfelelő számú Frame-et tartalmazó ütemezési tábla kiválasztása elvégezhető.

7.1.2. A Slave egység inicializálásának folyamata

A Slave egységek felelnek a hálózat megváltozásának detektálásáért és az automatikus címkiosztás biztosító XWDC jel megfelelő időben történő továbbításáért. LIN szoftverkomponensük működését a következő folyamatábra szemlélteti:



29. ábra: Egy Slave egység LIN inicializálásának folyamata

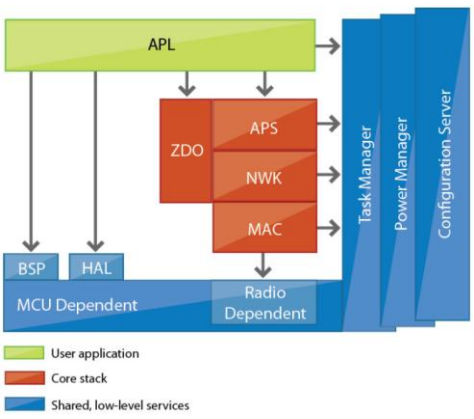
Egy egység elindulását követően, első feladata annak ellenőrzése, hogy ő-e a buszra (újonnan) csatlakoztatott eszközök közül a legelső inicializálatlan elem. Ha ő a soron következő akinek továbbítania kell az XWDC jelet, akkor ez a feltétel teljesül és az egység jogosult lesz a hálózat újrainicializálást kiváltó szignál bebillentésére. Ez a szignál egy minden ütemezési táblában megtalálható esemény-vezérlet Frame-ben kapott helyet annak érdekében, hogy a Frame megválaszolása ne legyen kötelező. Mivel mindig csak az XWDC láncon soron következő elem válaszolhatja meg ezt a Frame-et, az ütközés veszélye nem áll fent. A szignál jelzése (illetve a „hálózati aktív" állapotban lévő Slave-ek a szignál bebillentésének detektálása) hatására konfigurációs módba lépnek. Az XWDC jel által kiválasztott soron következő Slave egység fogadja a Master NAD beállító parancsát, nyugtázza azt és várakozó állapotba kerül. Ha a Master minden Slave-et ellátott egyedi NAD-el és beállított az általuk figyelendő Frame ID-ket, sorra lekérdezi azok egyedi azonosítóját is, melynek hatására azok a normál működést megvalósító „Hálózat aktív" állapotba lépnek.

7.2. Beágyazott operációs rendszerek programozása

A 8 bites mikrokontrollerekre történő fejlesztés során megszokott szekvenciális programozástól eltérően, egy beágyazott operációs rendszer használata teljesen más nézőpontot igényel. Számos konkurens szoftver-komponens dolgozik együtt a rendszerben, melyek egymásra hatását a programozónak folyamatosan szem előtt kell tartania.

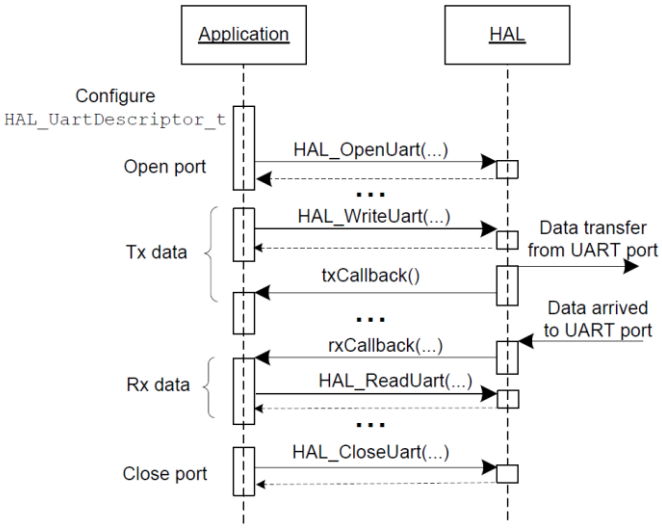
7.2.1. BitCloud

A szoftver rendszerterv bemutatásakor már ejtettem szót arról, hogy a BitCloud egy rendkívül jól strukturált réteges szerkezettel rendelkezik.



30. ábra: A BitCloud software stack réteges szerkezete [22]

Ezen szerkezetben a ZigBee protokollt megvalósító rétegek teljesen önállóan, a felhasználó számára transzparens módon működnek. Minden réteg saját taszk manager-rel rendelkezik, melyek rendszerhívással jelezhetik a kiszolgálási igényüket a rendszer ütemezője felé. Eseményvezérelt operációs rendszer lévén, programozás technikai szempontból kitűntetett szereppel rendelkeznek a Callback függvények. A rendszer legtöbb függvényhívása aszinkron működésű, vagyis nem blokkolja a szekvenciális programvégrehajtást. Talán ez az egyik legnagyobb előnye az ilyen egyszerűbb operációs rendszerek használatának, hiszen leveszi a műveletek befejeztének figyelésével járó terhet a programozó válláról.



31. ábra: Soros vonal kezelése BitCloud stack alatt [20]

A fenti UML diagram nagyon jól szemlélteti két réteg szoros együttműködését a BitCloud stack alatt. Az is nagyon jól látszik, hogy az alkalmazás réteg egyik kérésnél sem blokkolódik. Az új adatok beérkezését sem kell lekérdeznie, erről az eseményről automatikusan értesül egy Callback függvényének a HAL (Hardware Abstraction Layer) réteg által történő meghívásával.

Azért ennek a példának a bemutatását választottam, mert nekem is ezen két réteg (APL, HAL), illetve a HAL-ra épülő BSP (Board Support Package) megvalósításával kellett foglalkoznom. A BitCloud ajánlásának megfelelően a feladatom egy-egy taszk manager elkészítése volt, melyek gyakorlatilag nagyon egyszerű állapotgépet valósítanak meg. Ezen állapotgép megfelelő ága fog lefutni minden egyes alkalommal, ha a réteg kiszolgálást kér az operációs rendszer ütemezőjétől.

7.2.2. FreeRTOS

Az Adatkoncentrátor egységhez választott nagyteljesítményű mikrovezérlő már egy sokkal szélesebb funkciókészlettel rendelkező operációs rendszer használatát tette lehetővé. A választásom végül nem csak azért esett a FreeRTOS-re, mert már korábban szereztem tapasztalatokat a használatával kapcsolatban, hanem mert elérhető a port-olása a használt STM32F207-es mikrokontroller alá. Ennek implementációja viszont sajnos függ a választott fordítótól is, ezért teljes mértékben nem kerülhettem el az operációs rendszer új platformon való elindításával járó feladatok elvégzését. Ez többek között a kontroller helyes elindulását biztosító (pl. PLL beállításokat tartalmazó) Startup fájl assembly nyelvű korrekcióját jelentette. A programot a „CrossWorks for ARM" fejlesztőkörnyezetben készítettem.

A FreeRTOS a BitCloud-al ellentétben lehetővé teszi új taszkok futás idejű elindítását, azzal a korlátozással, hogy a lehetséges prioritási szinteket azonban már előre, fordítási időben definiálni kell. A megvalósított szoftverstruktúra megegyezik a szoftver rendszertervben bemutatottakkal. Az operációs rendszer két párhuzamos taszkot futtat:

- **Adatgyűjtő alkalmazás + FatFS meghajtó**
  - → A soros vonal eseményeire épülő taszk
  - → Fogadja a rádiós perifériától érkező szenzoradatokat és naplózza azokat
- **lwIP stack**
  - → Ethernet periféria kezelése
  - → Webszerver megvalósítása
  - → Az Adatgyűjtő alkalmazással együttműködve a naplózott szenzoradatok továbbítása a kliens felé

Az lwIP stack és az Adatgyűjtő alkalmazás programszálai közötti fellépő adatcsere esetén az adatintegritást az operációs rendszer egyik szinkronizációs elemével, egy bináris szemaforral valósítottam meg.

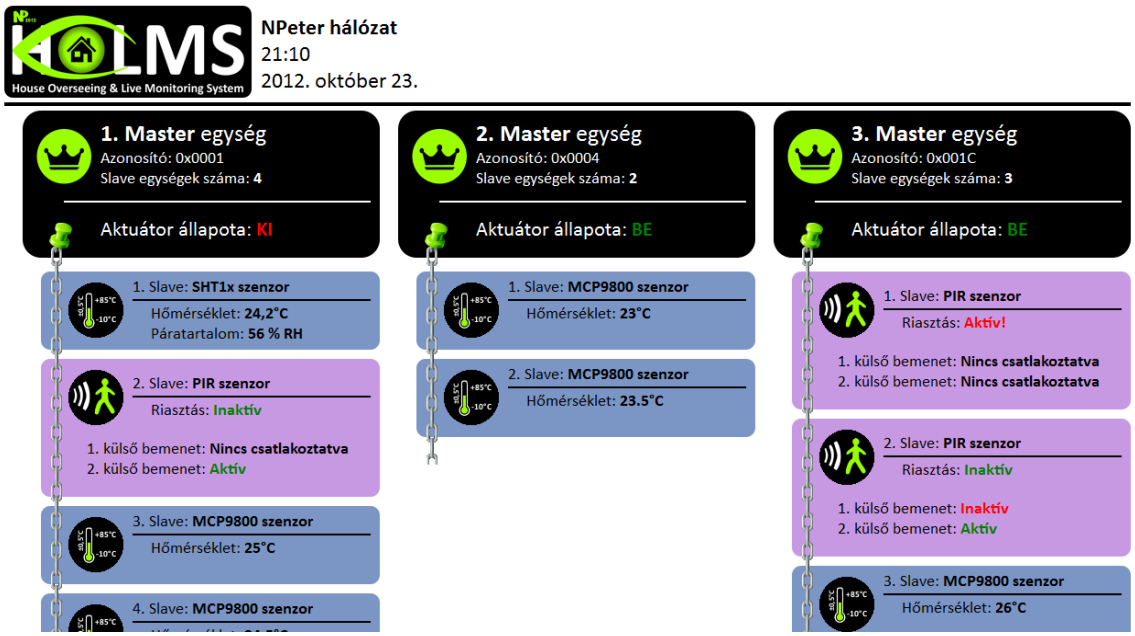
Az Adatkoncentrátor hardver egysége nem lett kiegészítve külső memóriával, ezért a hatékony memóriakihasználtság biztosítása kiemelten fontos volt. Szerencsére a FreeRTOS egy rendkívül jól konfigurálható OS, a felhasználó beállíthatja, hogy mely szolgáltatásait fordítja bele a programba.

7.3. Megjelenítés WEB felületen

Egy adatgyűjtő rendszer különösen fontos részét képezi a begyűjtött adatok megjelenítéséért felelős felület. Ezzel találkozik a felhasználó rendszeresen, ezért nagy mértékben hozzájárul a rendszer minőségéről alkotott képéhez. Igyekeztem ezt a felületet minél esztétikusabbra megalkotni.

Alapvetően két választási lehetőségem volt az adatok tárolását és az ezzel szorosan összekapcsolódó webes megjelenítéssel kapcsolatban. Vagy egy szerveren található adatbázisba töltöm fel az adatokat és onnét például egy PHP kiszolgálón keresztül kérdezheti le őket a felhasználó vagy egy helyi webszerveren keresztül teszem elérhetővé a szenzoradatokat. Mind a két lehetőség megvalósítását támogatja az lwIP stack, én az utóbbit választottam. Ennek egyszerűen az volt az oka, hogy szerver oldali alkalmazásokat már készítettem, de egyszerű 32 bites mikrovezérlőn megvalósított webszervert még soha, ezért egyfajta kihívásnak tekintettem a dolgot. Természetesen későbbi továbbfejlesztési lehetőségként a másik megoldás is bármikor implementálható a rendszerbe, akár egy párhuzamos szolgáltatásként.

Az lwIP stack által megvalósítható webszerver erőforrásai jelentős mértékben limitáltak, ezért például a komplexebb interaktív tartalmakat (pl. Flash) mellőzni kellett. Ennek ellenére én megpróbáltam egy minél látványosabb megjelenítési felületet megalkotni pusztán a HTML+CSS kombináció nyújtotta eszközkészlet felhasználásával.



32. ábra: HOLMS rendszer webes megjelenítő felülete

7.3.1. HTML skeleton

Annak érdekében, hogy az előző ábrán bemutatott megjelenítési felület erőforrás kímélő módon feltölthető legyen a szenzorhálózat aktuális adataival, egy gondosan strukturált HTML szerkezetet kellett megvalósítanom. Ez a szerkezet egyfajta skeleton-nak, vagyis csontváznak felel meg, melyben a különféle adatok számára előre fenntartott adatmezők vannak definiálva. A weboldal lekérésekor a webszerver alkalmazásnak egyszerűen ezekbe a mezőkbe kell behelyettesítenie az aktuális információkat, ezáltal visszaadva a felhasználónak a kvázi dinamikusan generált weboldalt. Az előző oldalon bemutatott mintaoldalhoz tartozó kitöltetlen struktúra a következő:

```
<div id="Header">
     <div id="HOLMSLogo"></div>
     <div id="NetworkDetails"></div>
</div>
<div id="Sensordata">
     <ul id="Masters">
          <li>
               <div class="Master">
                    <div class="MasterHeader"></div>
                    <div class="MasterActuator"></div>
               </div>
               <ul class="Slaves">
                    <li class="SlaveTemp">
                         <div class="SlaveTempHeader"></div>
                         <div class="SlaveData"></div>
                    </li>
                    <li class="SlavePIR">
                         <div class="SlavePIRHeader"></div>
                         <div class="SlaveData"></div>
                         <div class="SlaveLongData"></div>
                    </li>
                    <li class="SlaveTemp">
                         <div class="SlaveTempHeader"></div>
                         <div class="SlaveData"></div>
                    </li>
                    <li class="SlaveTemp">
                         <div class="SlaveTempHeader"></div>
                         <div class="SlaveData"></div>
                    </li>
               </ul>
          </li>
          <li>
               <div class="Master">
                    <div class="MasterHeader"></div>
                    <div class="MasterActuator"></div>
               </div>
               <ul class="Slaves">
                    <li class="SlaveTemp">
                         <div class="SlaveTempHeader"></div>
                         <div class="SlaveData"></div>
                    </li>
                    <li class="SlaveTemp">
                         <div class="SlaveTempHeader"></div>
                         <div class="SlaveData"></div>
                    </li>
               </ul>
          </li>
          .
          .
          .
     <ul>
</div>
```

A fenti struktúra feltöltése nem igényel nagy számítási teljesítményt, azért ezzel a megoldással egy rendkívül jó válaszidejű megjelenítési felületet sikerült implementálnom a HOLMS szenzorhálózat számára.

VIII. Összefoglalás

A TDK dolgozatomban egy komplex, hierarchikus szerkezettel rendelkező adatgyűjtő-vezérlő hálózati rendszer megtervezését tűztem ki célul. Megpróbáltam egy közös szenzorhálózatban egyesíteni a jelenleg a piacon elérhető vezetékes és vezeték nélküli alternatívák legjobb tulajdonságait. Ennek sikeres és hatékony megvalósításához először megvizsgáltam az ezekhez kapcsolódó legelterjedtebb szabványokat és protokollokat. Mérlegeltem ezek előnyeit és hátrányait, majd egy optimális kombináció kiválasztása után megkezdtem a rendszer tényleges implementálását.

A rendszerterv elkészítésekor már körvonalazódott, hogy egy rendkívül sokrétű hardverarchitektúrával rendelkező rendszer kerül megvalósításra, ezért a dolgozatomban is ezen terület bemutatására helyeztem a hangsúlyt. A munkám során megterveztem több egyszerű 8 bites mikrovezérlőn alapuló szenzor és aktuátor egységet, egy LIN busz és ZigBee alapú hálózat összekapcsolására alkalmas átjátszó elemet és az egész hálózat koordinátoraként funkcionáló, fejlett 32-bites ARM magú központi egységgel rendelkező adatgyűjtő állomást. Természetesen a számos hardverarchitektúra sokféle szoftverarchitektúra megismerését is lehetővé tette a számomra. A normál egyszálas, szekvenciális futással rendelkező program-fejlesztéstől, egészen a beágyazott rendszerekbe szánt prioritásos operációs rendszerek használatával is foglalkoztam. Mivel a begyűjtött adatok végső célállomása egy Ethernet alapú hálózat, így a feladat megoldásakor web alapú technológiát is alkalmaztam.

33. ábra: HOLMS szenzorhálózat komponensei

A munkám során számos érdekes és a szakmai fejlődésem szempontjából különösen hasznos kihívást kellett leküzdenem. Büszke vagyok arra, hogy a megtervezett hardveregységeket akár a piacon is életképes termékeknek megfelelő minőségben sikerült elkészítenem. Egészen a kezdeti vázlatoktól, a műszerdobozokon végzett utolsó simításokig minden részletre megpróbáltam aprólékosan odafigyelni. Természetesen fejleszteni való, mint minden elkészült rendszernél, a HOLMS szenzorhálózatnál is van, de úgy érzem, hogy a munkám végeztével a célkitűzéseimet sikerült maradéktalanul teljesítenem.

Németh Péter

IX. Irodalomjegyzék

[1] **BME-EMT: Connected Care for Elderly person**
2012
http://emt.bme.hu/emt/hu/cce

[2] **BME-MIT: mitmót**
2012
http://bri.mit.bme.hu/?l=mitmot&p=what is

[3] **Kömlődi Ferenc: Szenzorrendszerek**
2008. december
http://www.nhit-it3.hu/images/tagandpublish/Files/it3-2-1-11-u.pdf

[4] **Simon Gyula: Szenzorhálózatok**
2012. május
http://www.dcs.vein.hu/~simon/SH/

[5] **Berkeley mote: Smart dust**
Autonomous sensing and communication in a cubic millimeter
2012
http://robotics.eecs.berkeley.edu/~pister/SmartDust/

[6] **Dust Networks**
2011
http://www.dustnetworks.com/about/company_overview

[7] **Intel's Proactive Health Lab**
2007
http://www.intel.com/Assets/PDF/Article/Intel_Proactive_Health_Lab_article.pdf

[8] **Szőlőőr**
2012
http://szoloor.hu/index.php/hu/a-szoloor

[9] **Scherer Balázs, dr. Tóth Csaba:**
Autóipari kommunikációs hálózatok vizsgálata
2008. február
http://www.mit.bme.hu/system/files/oktatas/targyak/8604/BEARLab_M78.pdf

[10] **BOSCH: Controller Area Network**
2012. május
http://www.semiconductors.bosch.de/en/ipmodules/can/can.asp

[11] **CAN Bus Arbitration Method**
2012. május
http://www.softing.com/home/en/industrial-automation/products/can-bus/more-can-bus/communication/bus-arbitration-method.php

[12] **Local Interconnect Network**
2012. május
http://www.lin-subbus.de/

[13] **Scherer Balázs, dr. Tóth Csaba:**
Autóipari beágyazott rendszerek: LIN
2011. február
http://www.mit.bme.hu/system/files/oktatas/targyak/vedett/8616/ABR_LIN.pdf

[14] **Ballagi Áron: ZigBee**
2007. október
http://publikacio.uni-miskolc.hu/data/ME-PUB-16513/Ballagi_Zigbee_Poster.pdf

[15] **Bluetooth**
2012. május
http://en.wikipedia.org/wiki/Bluetooth

[16] **ZigBee Alliance Homepage**
2012. május
http://www.zigbee.org/Specifications/ZigBee/Overview.aspx

[17] **Kovács Balázs, Vida Rolland: A ZigBee technológia**
2012. május
http://w3.tmit.bme.hu/~vida/cv/zigbee.pdf

[18] **Turi Gábor: Rádióhálózatok ZigBee-adatátvitel alapján**
2008
http://www.macrobp.hu/WEBSET_DOWNLOADS/620/%C3%A1ltal%C3%A1nosan%20a%20Zigbee-r%C3%B3l.pdf

[19] **ihr: LIN Driver Configuration Tool**
2008
http://www.ihr.de/ihr/index.php?option=com_content&view=article&id=127&Itemid=556&lang=en

Felhasznált adatlapok:

[20] **myrra: 47000 series - Electronic transformers**
2012. május
http://www.farnell.com/datasheets/606238.pdf

[21] **Microchip: MCP2021**
2012. május
http://ww1.microchip.com/downloads/en/DeviceDoc/22018F.pdf

[22] **ATMEL: ATZB-24-A2**
2008. június
http://www.atmel.com/Images/doc8226.pdf

[23] **Microchip: PIC16F1824**
2011
http://ww1.microchip.com/downloads/en/DeviceDoc/41419C.pdf

[24] **Microchip: MCP9800**
2010
http://ww1.microchip.com/downloads/en/DeviceDoc/21909d.pdf

[25] **ST: STM32F207**
2012
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00237391.pdf

[26] **National Semiconductors: DP83848C**
2008
http://www.ti.com/lit/ds/symlink/dp83848c.pdf

X. Ábrajegyzék

XI. Függelék

11.1. LIN Description file (*HOLMS.LDF*)

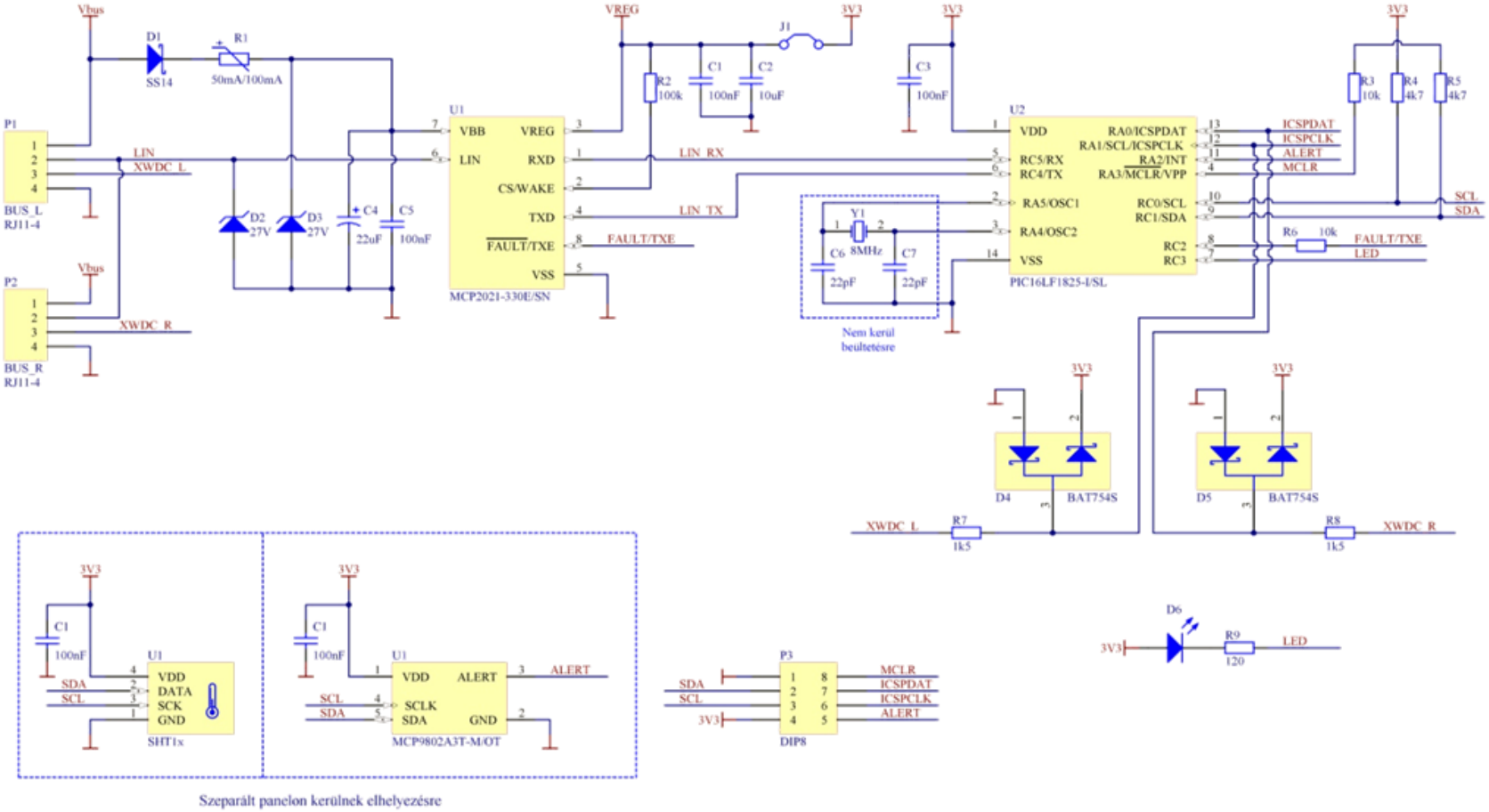
```
LIN_description_file;
LIN_protocol_version = "2.1";
LIN_language_version = "2.1";
LIN_speed = 9.6 Kbps;

Nodes {
	Master : HOLMS_Master, 10 ms, 0.1 ms;
	Slaves : SlaveMaster, Slave2, Slave3, Slave4, Slave5, Slave6, Slave7, Slave8, Slave9,
		 Slave10, Slave11, Slave12, Slave13, Slave14, Slave15;
}
Node_attributes {
	SlaveMaster {
		LIN_protocol = "2.1";
		configured_NAD = 1;
		initial_NAD = 1;
		product_id = 18, 1;
		response_error = CommErrorMaster;
		configurable_frames {
			SlaveMasterCMD;
			SlaveMasterData
		}
	}
	Slave2 {
		LIN_protocol = "2.1";
		configured_NAD = 2;
		initial_NAD = 2;
		product_id = 18, 2;
		response_error = CommError2;
		configurable_frames {
			Slave2CMD;
			Slave2Data;
		}
	}
	.
	.
	Slave15 {
		LIN_protocol = "2.1";
		configured_NAD = 15;
		initial_NAD = 15;
		product_id = 18, 15;
		response_error = CommError15;
		configurable_frames {
			Slave15CMD;
			Slave15Data;
		}
	}
}
Signals {
	SlaveMasterCMD_S  : 8, {0,0,0,0,0,0,0,0}, HOLMS_Master;
	SlaveMasterData_S : 7, {0,0,0,0,0,0,0}, SlaveMaster;
	CommErrorMaster   : 1, 0, SlaveMaster;

	ReinitRqst_S      : 1, 0, Slave2;

	Slave2CMD_S       : 8, {0,0,0,0,0,0,0,0}, HOLMS_Master;
	Slave2Data_S      : 7, {0,0,0,0,0,0,0}, Slave2;
	CommError2        : 1, 0, Slave2;
	.
	.
	Slave15CMD_S      : 8, {0,0,0,0,0,0,0,0}, HOLMS_Master;
	Slave15Data_S     : 7, {0,0,0,0,0,0,0}, Slave15;
	CommError15       : 1, 0, Slave15;
}
Diagnostic_signals {
	MasterReqB0 : 8, 0;
	MasterReqB1 : 8, 0;
	MasterReqB2 : 8, 0;
	MasterReqB3 : 8, 0;
	MasterReqB4 : 8, 0;
	MasterReqB5 : 8, 0;
	MasterReqB6 : 8, 0;
	MasterReqB7 : 8, 0;
	SlaveRespB0 : 8, 0;
	SlaveRespB1 : 8, 0;
	SlaveRespB2 : 8, 0;
	SlaveRespB3 : 8, 0;
	SlaveRespB4 : 8, 0;
	SlaveRespB5 : 8, 0;
	SlaveRespB6 : 8, 0;
	SlaveRespB7 : 8, 0;
}
```

```
Frames {
	ReinitRqst : 0x00, Slave2, 2 {
		ReinitRqst_S, 0;
	}
	SlaveMasterCMD : 0x02, HOLMS_Master, 8 {
		SlaveMasterCMD_S, 0;
	}
	SlaveMasterData : 0x03, HOLMS_Master, 8 {
		SlaveMasterData_S, 0;
		CommErrorMaster, 56;
	}
	Slave2CMD : 0x04, HOLMS_Master, 8 {
		Slave2CMD_S, 0;
	}
	Slave2Data : 0x05, Slave2, 8 {
		Slave2Data_S, 0;
		CommError2, 56;
	}
	.
	.
	Slave15CMD : 0x1D, HOLMS_Master, 8 {
		Slave2CMD_S, 0;
	}
	Slave15Data : 0x1E, Slave15, 8 {
		Slave15Data_S, 0;
		CommError15, 56;
	}
}
Event_triggered_frames {
	ReinitRqstEvent : Collision_resolver, 0x01, ReinitRqst;
}
Diagnostic_frames {
	MasterReq : 60 {
		MasterReq0, 0;
		MasterReq1, 8;
		MasterReq2, 16;
		MasterReq3, 24;
		MasterReq4, 32;
		MasterReq5, 40;
		MasterReq6, 48;
		MasterReq7, 56;
	}
	SlaveResp : 61 {
		SlaveResp0, 0;
		SlaveResp1, 8;
		SlaveResp2, 16;
		SlaveResp3, 24;
		SlaveResp4, 32;
		SlaveResp5, 40;
		SlaveResp6, 48;
		SlaveResp7, 56;
	}
}
Schedule_tables {
	Empty {
	}
	Diag {
		MasterReq delay 100 ms;
		SlaveResp delay 100 ms;
	}
	SlaveNum1 {
		SlaveMasterCMD  delay 20 ms;
		SlaveMasterData delay 20 ms;
		ReinitRqstEvent delay 10 ms;
	}
	SlaveNum2 {
		SlaveMasterCMD  delay 20 ms;
		SlaveMasterData delay 20 ms;
		ReinitRqstEvent delay 10 ms;
		Slave2CMD       delay 20 ms;
		Slave2Data      delay 20 ms;
	}
	SlaveNum15 {
		SlaveMasterCMD  delay 20 ms;
		SlaveMasterData delay 20 ms;
		ReinitRqstEvent delay 10 ms;
		Slave2CMD       delay 20 ms;
		Slave2Data      delay 20 ms;
		Slave3CMD       delay 20 ms;
		Slave3Data      delay 20 ms;
		Slave4CMD       delay 20 ms;
		.
		.
		Slave15CMD      delay 20 ms;
		Slave15Data     delay 20 ms;
	}
}
```

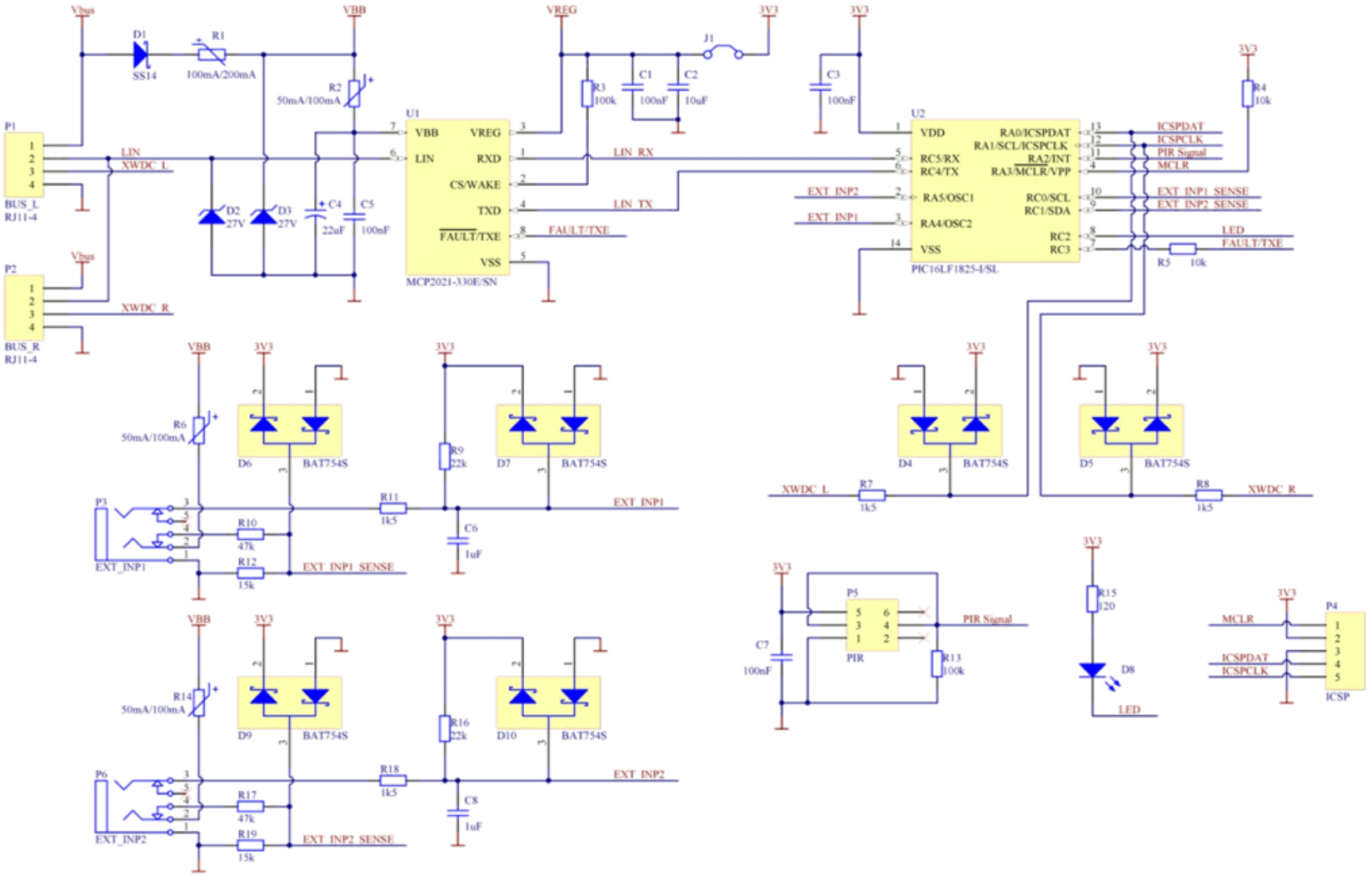
11.2. Kapcsolási rajzok

11.2.1. Slave egységek

11.2.1.1. Hőmérséklet érzékelő

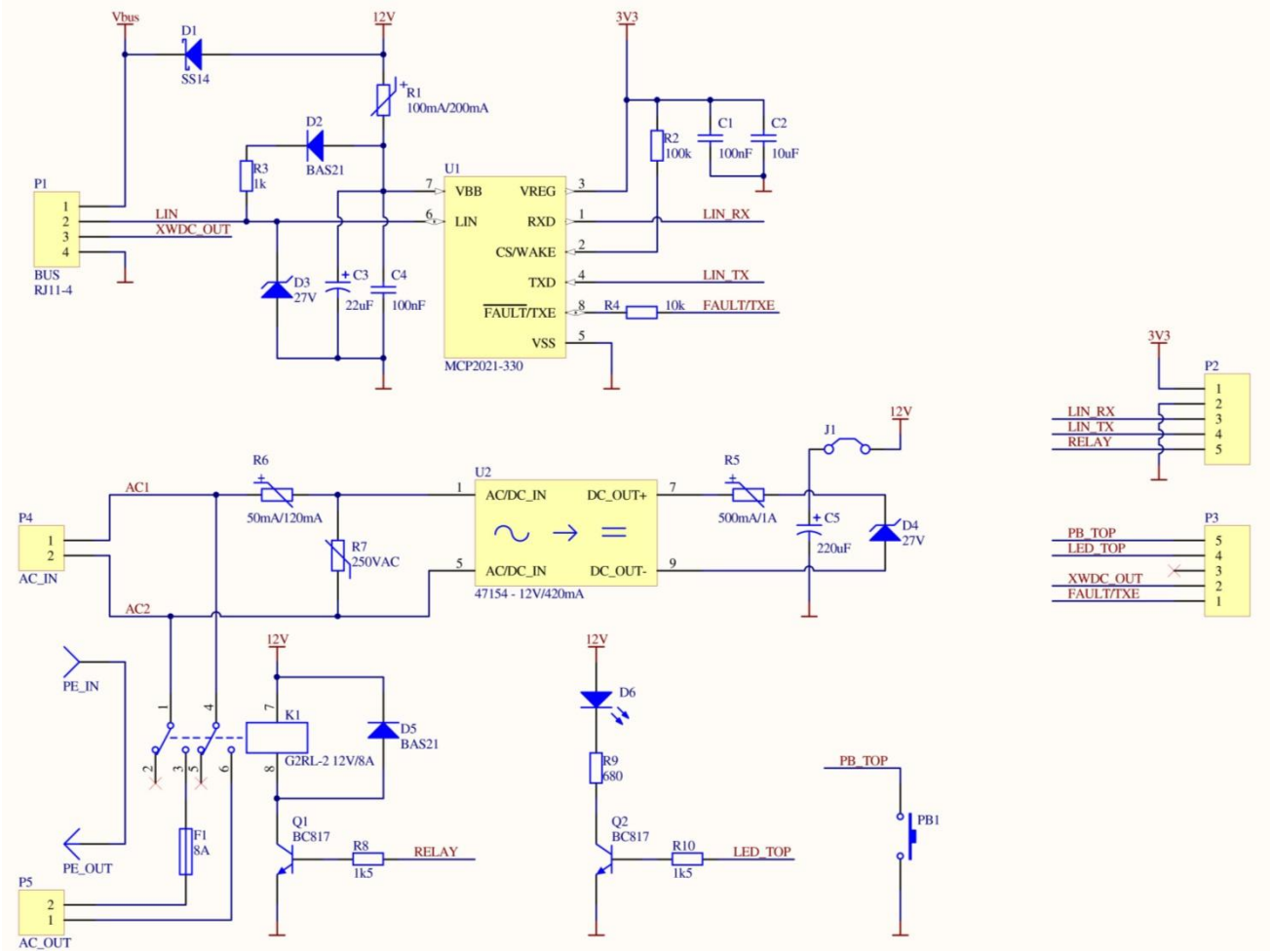


11.2.1.2. PIR mozgásérzékelő

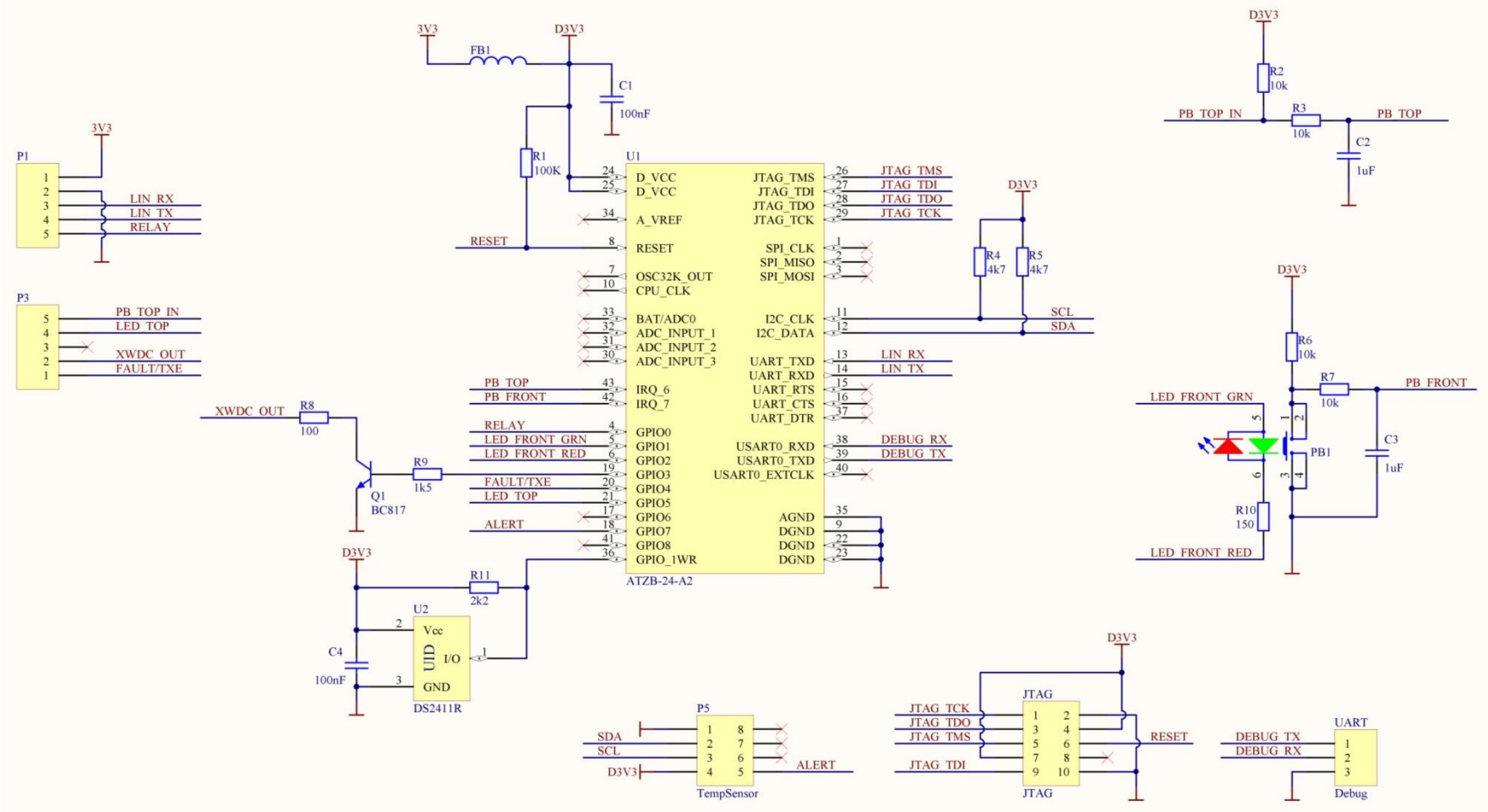


11.2.2. Master egység

11.2.2.1. Tápegység és LIN busz illesztés

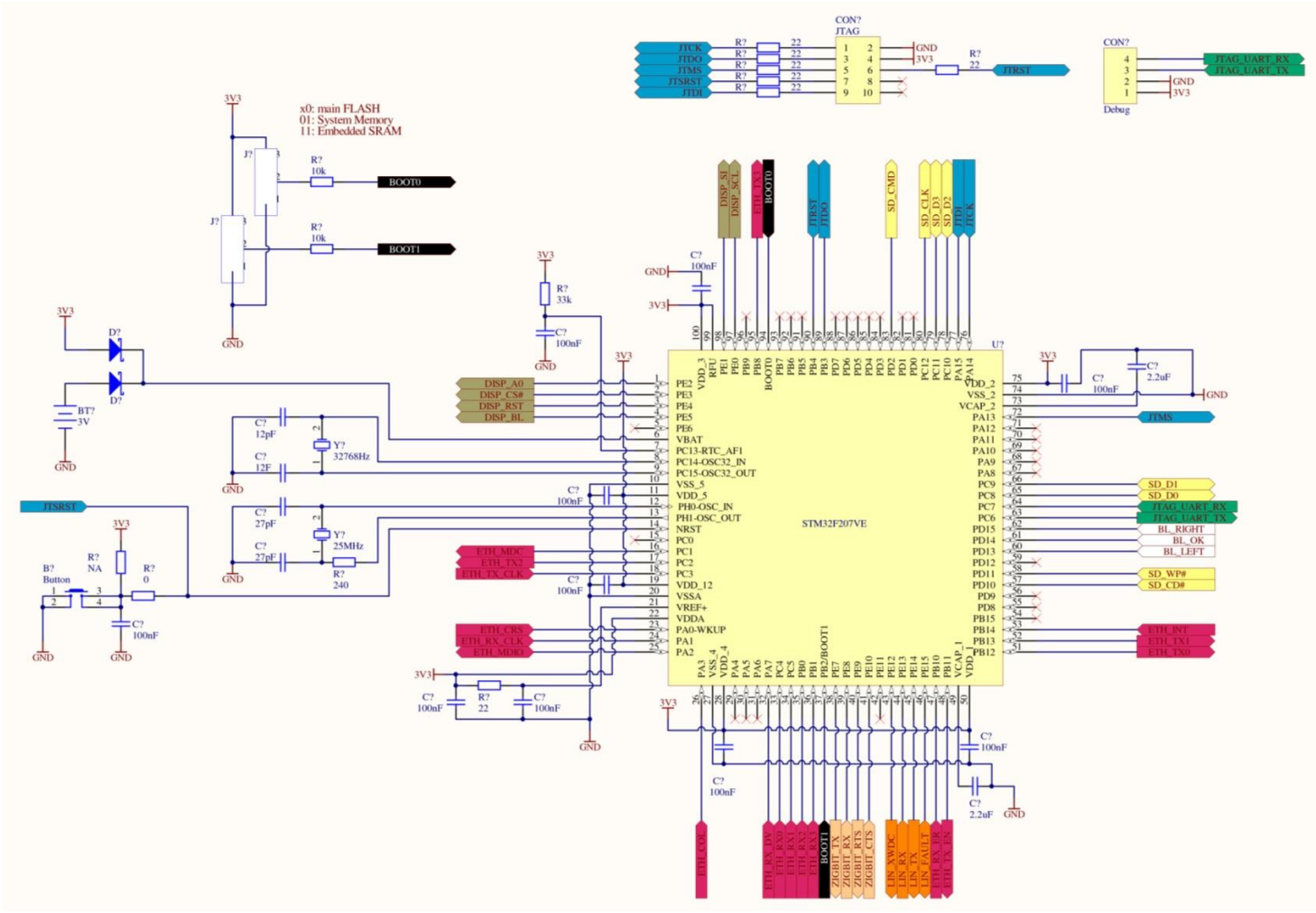


11.2.2.2. Központi egység és rádiós interfész

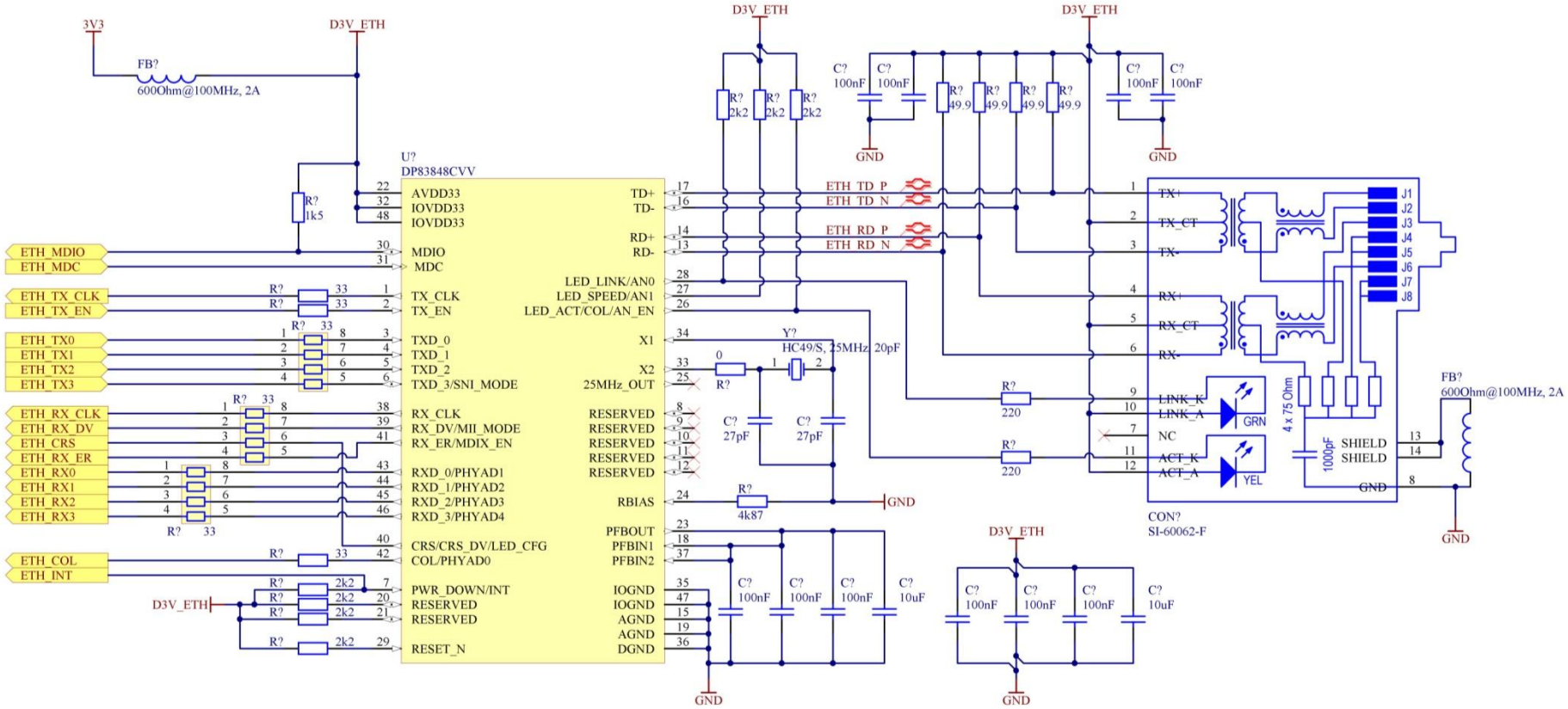


11.2.3. Adatkoncentrátor egység

11.2.3.1. Processzor



11.2.3.2. Ethernet



11.2.3.3. Grafikus kijelző és nyomógombok