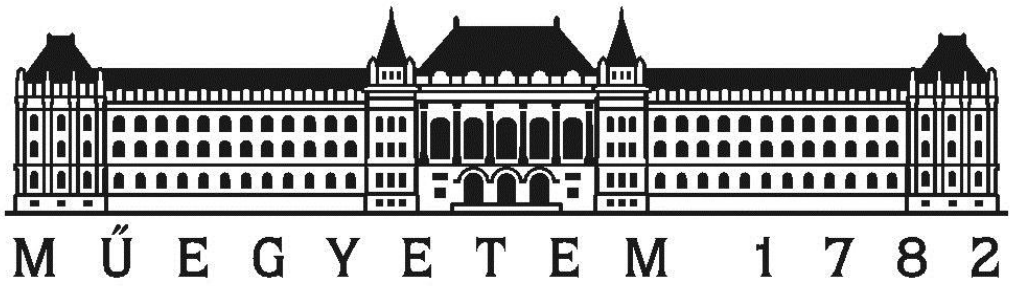


Budapesti Műszaki és Gazdaságtudományi Egyetem

Közlekedésmérnöki és Járműmérnöki Kar

Közlekedés- és Járműirányítási Tanszék

Kommunikációs interfész tervezése vezeték nélküli jármű-ember kapcsolathoz okostelefon platformok felhasználásával

TDK dolgozat

Készítették:

Balázs Gábor, járműmérnöki BSc

Koltai Gábor Soma, járműmérnöki BSc

Konzulens: Dr. Bécsi Tamás, adjunktus

Budapest, 2013. november

Tartalomjegyzék

1 Bevezetés

A TDK dolgozat egy olyan kutatás-fejlesztési projekt első eredményeit ismerteti, amely a Robert Bosch Kft. és a BME Közlekedési- és Járműirányítási Tanszék együttműködéseként jött létre és hallgatók bevonásával zajlik. A feladat egy olyan rendszer –és részelemeinek- fejlesztése, aminek a segítségével egy autó fedélzeti irányítórendszere és egy okostelefon közötti kommunikáció valósítható meg, valamint az okostelefon és a mobil internet képességeit kihasználva új szolgáltatások nyújtására nyílik lehetőség.

A dolgozatban részletesen foglalkozunk a rendszer felépítésével, a használt technológiákkal, valamint a beágyazott és az okostelefonon futó szoftverekkel:

- CAN és LIN hálózatok felépítése, működése, protokollok
- ECU-k, ezek szerepe egy gépjárműben
- Vezeték nélküli átjáró kifejlesztése: a beágyazott rendszer, ami kapcsolatot teremt a gépjármű és az okostelefon között
- Bluetooth technológia elemzése
- Androidos alkalmazás felépítése, fejlesztő környezet lehetőségei, ergonomikus felhasználói felület tervezése.

Az eddig elvégzett fejlesztés eredményeképpen összeállítottunk egy ún. „deszkamodellt” (autóipari szakkifejezéssel élve ún. „A-Sample”-t), amely tartalmaz egy akkumulátor szenzort, amely LIN hálózaton keresztül kommunikál egy Atmel fejlesztői „boarddal”, ami Bluetooth kapcsolaton küldi az adatokat egy Androidos mobiltelefonnak. A kommunikáció megvalósításához kidolgoztunk egy egyedi – JavaScript Object Notation (JSON) szabvány szerinti – protokollt.

Az internetre közvetlenül kapcsolódó autó koncepciója, és a mai okostelefonok képességei együttesen új szolgáltatások egész sorát teszi lehetővé, de egyszerűbbé teheti meglévő megoldások implementálását (hangvezérelt klíma, mainál sokkal olcsóbb flottamenedzsment megoldások). Egy új lehetőség viszont kötelezően új problémákat is felvet: azzal, hogy kaput nyitunk a gépjármű és az internet között, nem hagyhatjuk figyelmen kívül a biztonságot. Munkánk jelenlegi fázisában még nem foglalkoztunk részletesen a kérdéssel, de a későbbiekben megpróbálunk megnyugtató válaszokat adni.

A továbbiakban tervezzük egy iOS alapú megoldás kidolgozását is, valamint egy ergonomikus, és a jármű felszereltségéhez automatikusan alkalmazkodó grafikus felhasználói felület (GUI) kidolgozását is, valamint mélyebben elemezzük a biztonsági kérdéseket.

1.1 A rendszer célja

A Bosch-BME Connected Car K+F hallgatói projekt (CC) keretében fejlesztett, Bosch-BME Connected Car elektronikus vezérlőegység (a továbbiakban CC-ECU) célja, hogy egy személygépjármű fedélzeti hálózataira (CAN, LIN) csatlakoztatva onnan adatokat gyűjtsön, és ezeket az adatokat vezeték nélküli Bluetooth interfészen továbbítsa Android vagy iOS operációs rendszerű okostelefonra, amit az okostelefonra fejlesztett szoftver megjelenít.

2 Technológiák

2.1 Bluetooth

A Bluetooth egy vezeték nélküli rövidtávú adatátviteli technológia. Főleg mobiltelefonok, számítógépek, headsetek és egyéb számítógépes kiegészítők használják a kábelek kiküszöbölésére, de a több milliárd Bluetooth eszköz között megtalálhatók a fogkefék is. Azért ennyire elterjedt, mert kis fogyasztású, és nagy biztonságú az adatátvitel. A svéd Ericsson cég négy másik céggel együtt hozta létre a Bluetooth Special Interest Group-ot (SIG), amely elkezdett a Bluetooth szabvány kifejlesztésével foglalkozni. A SIG tagjainak száma már 10 ezer felett jár.

1. ábra: A Bluetooth logója

2.1.1 Fontosabb Bluetooth verziók (1)

2.1.1.1 Bluetooth v1.2

Frekvenciaugrásos technológia (interferencia elkerülése). Hatótávolság 1, 10 vagy 100 m.

Elméleti maximális átviteli sebesség: 1 Mbps.

2.1.1.2 Bluetooth v2.1

Könnyebb párosítás, nagyobb biztonság, sniff mód.

Elméleti maximális átviteli sebesség: 3 Mbps.

2.1.1.3 Bluetooth v3.0

Megalkotásánál a cél a sebesség nagymértékű növelése volt; ennek elérése érdekében nagyobb terhelésnél WiFi technológiát használ (802.11 PAL, Protocol Adaptation Layer), viszont kis terhelésnél visszakapcsol a kisebb fogyasztású módba.

Elméleti maximális átviteli sebesség: 24 Mbps.

2.1.1.4 Bluetooth v4.0 Low Energy

A szabvány megalkotásánál a cél a nagy sebességen kívül a kis energiafogyasztás volt, így tartalmaz egy ún. Low Energy módot, amely rendkívül kis fogyasztást kölcsönöz a rendszernek. Alapvető célja, hogy az eszközök egy gombelemmel hónapokig, akár évekig működhessenek (főleg az egészségügyi eszközök esetében). Az új technológia miatt a Bluetooth Low Energy visszafelé nem kompatibilis, ezért az ebből eredő fennakadás elkerülése végett sok eszköz a klasszikus Bluetooth-t is támogatja, ezek az ún. Bluetooth 4.0 Dual Mode eszközök. (2)

A Bluetooth Low Energy rendszernek a kis fogyasztás ellenére a hatótávolsága nagyságrendileg megegyezik a klasszikus Bluetooth-éval, viszont nagyméretű fájlok küldésére nem alkalmas, inkább kisebb üzenetek továbbítására tervezték. Ezért hangátvitelre például nem alkalmas. Topológiája megegyezik az eddigi Bluetooth szabványokkal (piconetek, master-slave kommunikáció, részletesen később), viszont a hálózatok (piconetek) közötti összeköttetés (scatternet) nem megoldható vele. Fontos különbség még, hogy a slave-ek száma alkalmazás-implementációtól függő. (3)

A Bluetooth Low Energy és a klasszikus Bluetooth szabványok alapvető összehasonlítása az 1. táblázatban látható.

1. táblázat: A klasszikus Bluetooth és a Bluetooth Low Energy összehasonlítása (4)

Specifikációk	klasszikus Bluetooth technológia	Bluetooth Low Energy technológia
Hatótávolság	100 m	50 m
Frekvenciasáv	2402 MHz – 2480 MHz	2402 MHz – 2480 MHz
Adatátviteli sebesség a levegőn keresztül	1-3 Mbit/s	1 Mbit/s
Adatátviteli sebesség	0,7-2,1 Mbit/s	0,27 Mbit/s
Slave-ek száma	7	alkalmazásfüggő
Energiafogyasztás a klasszikus Bluetooth-t 100%-nak véve	100%	1%-50% (használattól függően)
Hangátvitel támogatása	igen	nem
Profilok támogatása	igen	igen
Eszközfelfedezés támogatása	igen	igen
Maximum fogyasztás	30 mA	15 mA
Késleltetés	100 ms	6 ms
Küldésig eltelt idő	100 ms	3 ms

A Low Energy a klasszikus szabványhoz hasonlóan több profilt támogat (a profilokról részletesen később). Az alapvető profil, ami az adatcsomagok küldését biztosítja, a GATT profil (Generic Attribute Profile). A GATT célja alapvetően az, hogy olyan alkalmazást vagy profilt építhessünk rá, amely egy szerver-kliens kommunikációt valósít meg.

2.1.2 Fizikai megvalósítás (5)

A Bluetooth-t támogató készülékekben egy Bluetooth chip van, ami a vezeték nélküli kommunikációt biztosítja. A 2402 MHz-től 2480 MHz-ig terjedő frekvenciasávot használja (Industrial, Scientific and Medical, ISM frekvenciasáv), 1 MHz-es lépésközökkel, így 79 különböző frekvenciát használhat. Mivel a rádiós adónak és vevőnek nem kell egymást fizikailag "látni", meg kell oldani az interferencia által okozott problémát. Az interferencia elkerülése végett az adaptív frekvenciaugrást (Adaptive Frequency Hopping, AFH) alkalmazza. Eszerint másodpercenként 1600-szor

megváltoztatja az adatátvitel frekvenciasávját, így teljesen valószínűtlen, hogy két eszköz ugyanakkor, ugyanazon a frekvencián kommunikáljon. Ezen kívül a kis fogyasztás és az interferencia elkerülése célját szolgálja az, hogy a Bluetooth eszközök nagyon gyenge jeleket bocsátanak ki. Ez a kommunikációban, mivel kis távolságokra van tervezve, semmilyen problémát nem okoz. A maximális hatótávolság eszköztől függően 1-től 100 m-ig terjed. Az energiafogyasztás/hatótávolság szerint az eszközöket osztályokba soroljuk a következőképpen:

2. táblázat: Bluetooth osztályok hatótávolsága

Osztály	Maximum fogyasztás		Hatótávolság [m]
	[mW]	[dBm]	
1-es osztály	100	20	~100
2-es osztály	2,5	4	~10
3-as osztály	1	0	~1

2.1.3 Hálózati szerveződés, kapcsolatteremtés

A Bluetooth eszközök kisméretű helyi hálózatokba, ún. piconetekbe csatlakoznak, így egy Personal Area Network-öt (PAN-t) hoznak létre. Egy piconetben 8 aktív eszköz lehet: egy master, amelyik a kommunikációt kezdeményezi és hét vele kommunikáló aktív slave eszköz (a nem aktív, virtuális slave eszközökkel együtt 255). A piconet felépítését kezdeményező eszköz lesz automatikusan a master, de az egyes eszközökön külön be lehet állítani, hogy alapértelmezettként masterek vagy slave-ek legyenek. A szerepek idő közben felcserélődhetnek.

A slave-ek kapcsolódáskor szinkronizálják az óráikat a masteréhez, majd a kommunikáció a master Bluetooth órája szerint zajlik. A szinkronizáció a szolgák órajelének eltolásával, azaz ofszettel történik. Ha a hálózat megszűnik, az órajelek visszaállnak az eredeti értékre.

A Bluetooth egyik legfontosabb jó tulajdonsága az, hogy az oda és a vissza irányú kommunikáció egyidejűleg, szimultán történhet, azaz full-duplex. Ez úgy történhet meg, hogy a csatornát időrésekre osztják; egy rés időtartama 625 μs. (Ezt nevezik időosztásos kommunikációnak Time Division Multiple Access.) Általánosságban a master eszköz a

páratlan, a slave eszköz a páros számú időrésekben küld adatcsomagot a másiknak. Egy csomag 1, 3 vagy 5 egymás utáni időrést foglalhat el. Ezzel a módszerrel az egyidejű kétirányú kommunikáció mellett a beszédátvitel is megoldható, mivel a csomagok, amelyekben a beszéd szétbontva érkezik 625 μs-onként követik egymást, és a vevő oldalon könnyen össze lehet őket rakni.

A mestereszköz tud egyszerre több szolgával is kommunikációt létesíteni; mindig a mester dönti el, hogy pillanatnyilag melyik szolgával legyen kapcsolatban. Ezen kívül lehetséges az is, hogy egy eszköz része két pikohálózatnak is, tehát lehet egy hálózatban master, miközben egy másikban slave, viszont egy eszköz csak egy hálózatban funkcionálhat mesterként.

Ha egy hálózatban nemcsak egy szolga van (mint ahogy az általunk elképzelt hálózatban sem), fontos, hogy az adott csomag a megfelelő szolgához érkezzen. Ezért a csomag egy címzést tartalmaz, amely a Header részében van. A csomagok három részből állnak (6):

- Elérési kód (Access Code): mérete 72 bit, időszinkronizálásra, ofszet kompenzációra használják.
- Fejrész (Header): mérete 54 bit, többek között a csomagazonosító információt és a szolga címét tartalmazza.
- Adatrész (Payload): 0-2754 bites, a nyers beszéd- és adatátviteli biteket tartalmazza.

A Bluetooth fizikai csatornája rétegekre osztható (ezekből épül fel az ún. Bluetooth stack). A réteges szerveződés beilleszthető az Institute for Electrical and Electronic Engineers (IEEE) által kidolgozott Open System Interconnection (OSI 7) modellbe. Ez 7 réteget tartalmaz; alulról felfelé haladva:

3. táblázat: Az OSI modell rétegei (7)

Réteg	Feladat
fizikai réteg (physical layer)	fizikai és elektromos specifikációk biztosítása

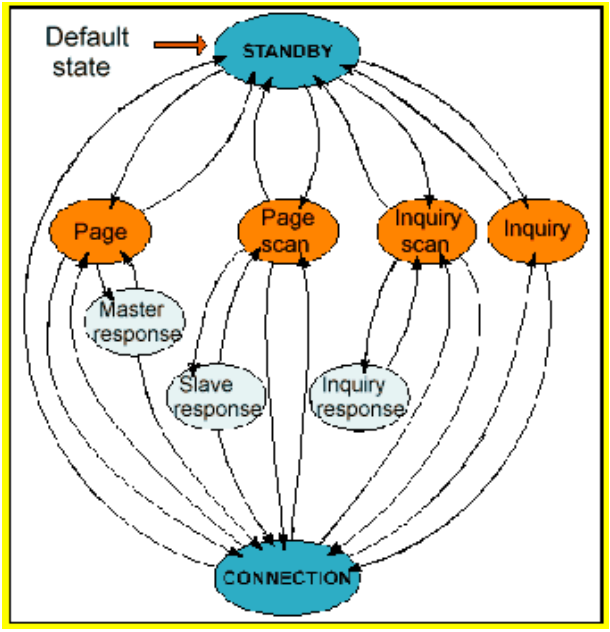
adatkapcsolati réteg (data link layer)	eszközcímzés, adatátvitel biztosítása két eszköz között
hálózati réteg (network layer)	változó hosszúságú adatsorozatoknak a küldőtől a címzetthez való továbbításáért felelős
szállítási réteg (transport layer)	annak biztosítása, hogy minden adat hibátlanul eljusson a címzetthez (viszonylati és hálózati réteggel kommunikál)
viszonylati réteg (session layer)	párbeszéd létrehozása a végfelhasználói alkalmazások között, az adatkapcsolat meglétének ellenőrzése
megjelenítési réteg (presentation layer)	az adatok egységesítéséért, kódolásáért felelős
alkalmazási réteg (application layer)	maga a végfelhasználói alkalmazás rétege, feladata a fogadott adatok feldolgozása

A fizikai rétegnek felel meg a Bluetooth rádió, amely maga az adóvevő, ez valósítja meg a frekvenciaugrálást is. Az adatkapcsolati réteghez tartozik a Baseband Link Controller, amely szolgáltatja az eszköz órajelét és létrehozza az eszközcímzést, illetve a Link Manager Protocol, amely az eszközök közötti kapcsolatokat kezeli. A mi szempontunkból fontos még a viszonylati réteg, amelyen a Bluetooth több protokollt definiál, ezekről a későbbiekben részletesen lesz szó. Az alkalmazási rétegben az általunk megírt alkalmazás protokollja foglal helyet, azaz ami a nyers adatbiteket feldolgozza.

Összefoglalva a csatorna rétegződése úgy képzelhető el, hogy ha egy adatcsomagot, néhány bitet szeretnénk küldeni, akkor erre az adatcsomagra rárakódnak a különböző rétegek protokolljainak bitjei, és így a csomag fizikailag jóval nagyobb lesz.

2.1.4 A Bluetooth egységek állapotai

A piconetben a Bluetooth eszközök különböző állapotokban lehetnek jelen: nyugalmi, lekérdező, kereső és kapcsolt állapot; a kapcsolt állapoton belül még megkülönböztetünk aktív, szaglászó, tartó és parkoló módot (6) (8).



2. ábra: A Bluetooth állapotai I. (5)

Nyugalmi állapot (Standby): Minden eszköz ebben az állapotban van, amikor nem csatlakozik egy pikohálózathoz sem, nincs kapcsolatban más eszközökkel. Ebben az állapotban az eszköz minden 1,28 másodpercben figyel a 79-ből 32 különböző frekvenciát, és várja az esetleges üzeneteket. Az energiafogyasztás alacsony ebben az állapotban.

Lekérdező, Kereső (Inquiry, Page): A kapcsolat megteremtése történik ezekben az állapotokban; Page módban akkor van az eszköz, amikor kapcsolatot akar létesíteni; ilyenkor Page üzeneteket küld 16 frekvencián. Az Inquiry üzenet más eszközök keresésére szolgál; ebben az állapotban hosszabb ideig lehet az eszköz, mivel várakozik a céleszköz válaszára.

A céleszköz egy lekérdezés választ küld vissza, ez után kezdődhet el a hálózat felállítása (Page állapot).

A kapcsolat innentől felépülhet, mert csak a céleszköz címére van szükség hozzá. Ha ismert a céleszköz órajele, a hálózat gyorsabban felépíthető. A keresést indító eszköz lesz automatikusan a master.

A következőkben a kapcsolt állapoton belüli módokat részletezzük:

Aktív mód (Active): A Bluetooth eszköz aktívan részt vesz a kommunikációban. Az aktív szolgák figyelik az időrésekben a mester által küldött csomagokat. Ha egy slave eszköznek éppen nem küld a master csomagot, a következő csomag érkezéséig alvó állapotba kerül.

Szaglászó (Sniff): Ebben a módban egy szolgának csökkentett az adatforgalma, mert csak meghatározott időrésekben küldhet neki a mestereszköz csomagot. Így kevesebbet kell "figyelnie", tehát kisebb lesz a fogyasztása. A meghatározott időrések közötti időintervallumot be lehet állítani, értéke az adott programtól függ.

Tartó (Hold): Az adatkapcsolat megszűnik, viszont az eszköz szinkronizálva marad, és megtartja MAC címét is. Ha az eszköz kilép a Hold módból, a kommunikáció rögtön újra tud indulni. Egy eszköz Hold módba helyezését kezdeményezheti a mester vagy maga az eszköz, hogy a csatorna kapacitását fel lehessen használni más célokra.

Parkoló (Park): Az eszköz órája szinkronizálva marad, de egyáltalán nem vesz részt a hálózati kommunikációban, tipikusan, mert rajta kívül már van hét slave és egy master eszköz a hálózatban. Parkoló módban lehetséges, hogy egy hálózatban 255 Bluetooth eszköz csatlakozzon (virtuálisan).

3. ábra: A Bluetooth állapotai II. (6)

2.1.5 A Bluetooth profiljai

Annak érdekében, hogy két Bluetooth eszköz kommunikálni tudjon, egy kommunikációs interfészt kell definiálni köztük, ez a profil. Ahhoz, hogy a kommunikáció megvalósulhasson, mindkét eszköznek kompatibilisnek kell lennie az adott profillal. Az, hogy milyen profilokat támogat egy eszköz, a gyártón múlik. Ebből a programozás során származhatnak kellemetlenségek, ahogyan ez a mi esetünkben is előfordul (lásd később).

A profilok tehát egyszerűen megfogalmazva előre definiált protokollok, amelyek a bemenő nyers adatot átalakítják valamilyen formába, majd a fogadó oldalon visszaalakítják értelmezhető adatokká.

Az Android alapvetően háromféle profilt támogat: Headset, Advanced Audio Distribution Profile és Health Device, ám a BluetoothSocket implementálásával lehetőségünk van a saját Bluetooth profilunkat definiálni.

Az iOS több profilt is támogat (pl. Hands Free Profile, Advanced Audio Distribution Profile), de itt a Bluetooth socketébe nem nyúlhatunk bele, így nem tudunk saját adatátviteli protokollt definiálni, hanem az adatátvitel céljából az egyik előre meghatározott profilt kell használnunk. (Ez azért van így, mert az iOS fejlesztői inkább a biztonságra törekedtek.) Ennek a megkötésnek kikerülése céljából egy Bluetooth modult használunk (ld. később).

A következőkben az SPP-t ismertetjük:

Az SPP-vel az eszközök között virtuális soros port szerű kommunikációt létesíthetünk, azaz azt tudjuk emulálni, hogy az eszközök egy vezetékkel össze vannak kötve és transzparens módon mennek át az adatok az egyiktől a másikhoz. Erre épülnek a Bluetooth profilok.

2.1.6 A Bluetooth modul

A Bluetooth modul egy olyan egység a rendszerünkben, amely a mikrokontroller által feldolgozott adatoknak a kódolásáért, valamint a Bluetooth-on történő továbbításáért felelős. Benne egy mikrokontroller és egy Bluetooth chip található. A Bluetooth chip felelős a fizikai kommunikációért. A mikrokontroller feladata a fentebb már említett probléma megoldása, azaz a saját Bluetooth-profil megvalósítása. A bejövő adatokat a modul lekódolja az általunk definiált profil szerint, majd átkódolja egy, a telefon által támogatott profil szerint, amit a telefon a vevő oldalon már kezelni képes. Az alkalmazott saját definiálású profil JSON alapú.

A Bluetooth modulunk egy ConnectBlue gyártmányú OEM Bluetooth Serial Port Module OBS421 típusú készülék. Ez a modul képes Bluetooth-os soros adatátvitelre UART interfészen keresztül. Támogatja a Bluetooth v4.0-t, így két alkalmazási módja van: az adatokat küldhetjük a hagyományos SPP-vel vagy a ConnectBlue Low Energy Serial Port Service-ével. Az iOS-es fejlesztés szempontjából fontos, hogy támogatja az iPhone-t is (csak Low Energy módban).

Főbb tulajdonságok:

- beágyazott Bluetooth stack
- dual mode (Low Energy és klasszikus Bluetooth)
- Android támogatás

- iPhone támogatás
- nagysebességű UART
- akár 1,3 Mbps sebességű adatátvitel, akár 300 m-es hatótávolság
- üzemi hőmérséklettartomány -30°C-tól +85°C (ez az autóban uralkodó üzemi körülmények miatt különösen fontos)

2.1.7 A Bluetooth biztonsága

Két Bluetooth eszköz összekapcsolását párosításnak nevezik; csak párosított eszközök kommunikálnak egymással. Az eszközökön beállítható, hogy felfedezhetőek legyenek-e a többi eszköz számára. Ha a felfedezhetőség nincs bekapcsolva, és a két eszköz előzőleg még nem volt csatlakoztatva, a párosítás nem lehetséges. A felfedezhetőség a legtöbb esetben alapbeállítás szerint nem felfedezhetőre van állítva, és a felhasználónak külön engedélyezni kell a felfedezhetőség bekapcsolását, ami általában csak egy előre meghatározott időre valósul meg.

2.1.7.1 A Simple Pairing (egyszerű párosítás) specifikáció (9)

A Bluetooth a 2.1 + EDR (Enhanced Data Rate) támogatja az ún. Simple Pairing (egyszerű párosítás) funkciót, amely jelentősen felgyorsítja, megkönnyíti, és egyben biztonságosabbá teszi a Bluetooth-on történő kommunikációt. Védelmet nyújt a passzív lehallgatás ellen (néhány bit plusz titkosító kódot ad hozzá a csomaghoz), illetve az aktív lehallgatás, az ún. közbeékelődéses támadás ellen. (Ez utóbbi azt jelenti, hogy két kommunikáló eszköz közé beékelődik egy harmadik, amely a kommunikáló felek számára úgy mutatja magát, mintha a másik fél lenne, tehát a kommunikáció rajta keresztül zajlik.) A közbeékelődéses támadás sikerességének egy a millióhoz az esélye.

A Simple Pairing alkalmazásának három alapesete:

4. táblázat: A Simple Pairing alapesetei (9)

Sima párosítás:	olyan esetekben fordul elő, amikor az egyik eszköznek nincsen kijelzője vagy billentyűzete (bemenete vagy kimenete), úgy, mint pl. egy telefon és egy headset kapcsolódása. Ilyenkor a headset a telefon párosítási kísérleteit automatikusan elfogadja.
Numerikus összehasonlítás:	olyankor történik, amikor mindkét eszköz rendelkezik kijelzővel és egy igen/nem beviteli lehetőséggel (pl. mobiltelefon-PC kapcsolat). Ebben az

	esetben egy hat számkarakterből álló kód generálódik, amelyet mindkét eszköz kijelez. Ha ez a számkód egyezik, mindkét oldalon az Igen megnyomásával létrejön a kapcsolat.
Kód megadás:	erre akkor van szükség, amikor az egyik eszköznek nincsen kijelzési lehetősége, viszont a másiknak van (pl. billentyűzet-PC). Ekkor az előző esethez hasonlóan egy hat számból álló kód generálódik, amit a PC kijelez, és a kapcsolódáshoz a billentyűzeten be kell ütni ezt a kódot.

2.1.7.2 Biztonsági módok Simple Pairingben

A párosítás módja szerint hétféle biztonsági mód van definiálva. Alapelv, hogy az újabb Bluetooth verziók támogassák a régebbi titkosításokat is.

5. táblázat: A Bluetooth biztonsági módjai (9)

1. biztonsági mód: Kikapcsolt biztonság	Kikapcsolt biztonság, a párosítási kérés automatikusan el lesz fogadva.
2. biztonsági mód: Bluetooth 2.0 + EDR biztonság	A Simple Pairing kikapcsolása, a Bluetooth 2.0 + EDR szabvány szerinti biztonság jön létre. A visszafelé kompatibilitás miatt fontos, ha egy magasabb szintű Bluetooth eszköz kommunikál egy 2.0 + EDR-essel.
3. biztonsági mód: Fix PIN kód	Kapcsolódás a készülék memóriájában tárolt fix kód alapján.
4. biztonsági mód: Sima párosítás	Ha a készülékek felfedezhetők, a párosítási kérés automatikusan el lesz fogadva.
5. biztonsági mód: Csak kijelző	Egy User Passkey Display Event megy a céleszköznek egy hatjegyű számmal, amit a billentyűzet oldalon kell beírni.
6. biztonsági mód: Igen/Nem bevitel	Egy User Confirmation Event megy a céleszköznek egy hatjegyű számmal, amit elfogadhat vagy elutasíthat.
7. biztonsági mód: Csak billentyűzet	Egy User Passkey Entry Event megy a céleszköznek egy hatjegyű számmal, amit be

kell ütni.

2.2 CAN (10)

A **Controller Area Network** (CAN) egy adatbusz szabvány és hálózati kommunikációs protokoll, melyet a Robert Bosch GmbH kezdett el kifejleszteni 1983-ban, (az első CAN kontroller chipek 1987-ben jelentek meg) első sorban autóipari alkalmazásokhoz. A szabvány célja az volt, hogy az autókban található –egyre növekvő számú- mikrokontroller közötti kommunikációhoz használt kábelezést lecseréljék: a mikrokontrollerek számával a kommunikációhoz használt kábelek száma folyamatosan nőtt, ami drága, helyigényes, és egyre megbízhatatlanabb megoldásnak bizonyult. A CAN busz ezzel szemben két vezetéket használ a több eszköz közötti kommunikáció megvalósításához.

2.2.1 A CAN busz (10)

A CAN busz sodort érpárt használ (ami lehet árnyékolt vagy árnyékolatlan), erre csatlakoznak az állomások. A megfelelő működéshez a buszt mindkét végén ellenállásokkal kell lezárni, melyeknek az szabványos értéke RL=120 Ohm –a tapasztalat azt mutatja, hogy lezáró ellenállás nélkül, bizonyos méret alatt a hálózat lassabb átviteli sebességgel, de működőképes tud maradni-.

6. táblázat: CAN adatátviteli sebességek

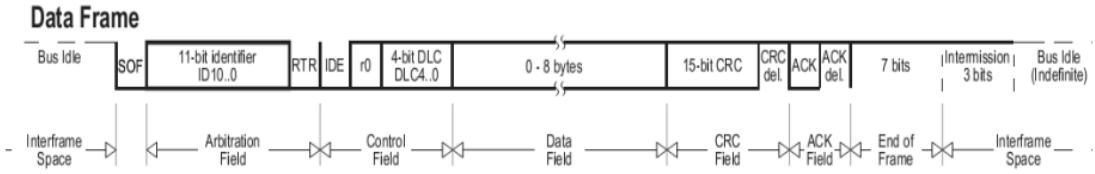
Kábelhossz (m)	Adatátviteli sebesség (kbit/s)
30	1000
100	500
250	250
500	12,5
1000	62,5

A fenti táblázatból látható, hogy ez a felépítés maximum 1 Mbit/s adatátviteli sebességet tesz lehetővé. A busz elektromos paramétereit az ISO11898 szabvány részletezi.

Az állomásokat egy adóvevő (transceiver) illeszti a buszra, mely a kontroller TTL jelszintjét alakítja át a busz differenciális jeleire. A CAN busz a domináns alacsony és recesszív magas jelszintekből adódóan egy logikai ÉS kapcsolatot valósít meg az állomások között, aminek az ütközésdetektálásban van nagy szerepe.

2.2.2 A CAN protokoll

Hasonlóan más komplex hálózatokhoz, a CAN is csomag alapú. A CAN adatkeretet a következő ábra szemlélteti.



4. ábra: CAN adatkeret

A busz alaphelyzetben magas jelszinten van. A kommunikálni kívánó állomás ezt logikai 0 szintre húzza (SOF – Start of Frame), ezzel jelezve az adatkeret elejét. A lefutó élre történik az állomások órájának szinkronizációja. A több résztvevőből adódóan előfordulhat ütközés, ennek kiküszöbölésére szolgál az arbitrációs mező. Több állomás egyidejű forgalmazása esetén a legalacsonyabb ID-val rendelkező üzenet fog „nyerni” az állomások közötti logikai ÉS kapcsolat, valamint a domináns nulla jelszint miatt, azaz prioritása az üzeneteknek van, nem az állomásoknak.

A Control Field első bitje meghatározza, hogy sima, vagy Extended ID-t használ az adatkeret (Extended ID esetén az ID mező további 18 bittel bővül), ezt követi a 4 bites Data Length Counter (DLC), ami megadja, hogy milyen hosszú adatmező következik – ennek szerepe van a hibadetektálásban.

Az adatmező 0-8 byte hosszú lehet, ezt követi a 15 bites CRC mező, ahova az adatmező ellenőrző összege kerül. A vevő kiszámolja a fogadott adat ellenőrző összegét, összehasonlítja a CRC mezővel, egyezés esetén az ACK mezőben jelzi az adónak, hogy fogadta a csomagot, ellenkező esetben hibát kell generálnia.

A keretet 7 bitnyi magas jelszint zárja (End of Frame), majd az adónak várnia kell a következő keret küldéséig, ezzel időt biztosítva a vevőnek az adatok feldolgozására, ennek hossza 3 bitnyi idő (Bus Idle).

A CAN protokollban ezek mellett definiálva van két további adatkeret: Error Frame (hiba) és Overload Frame (túlterhelés).

2.3 LIN (10)

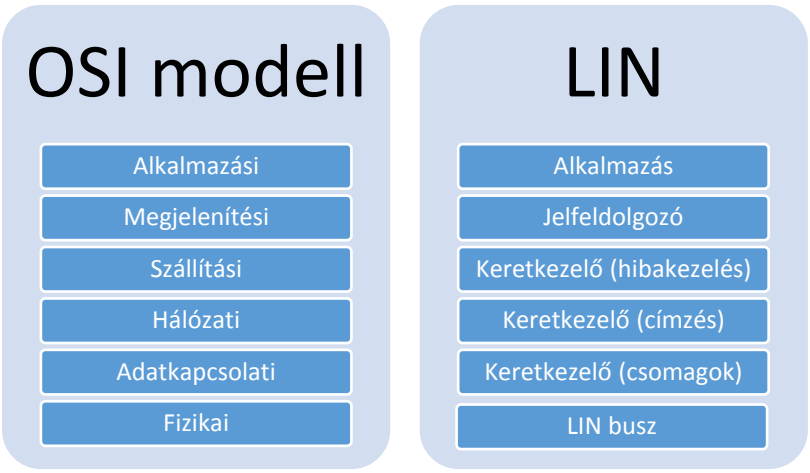
A nyolcvanas évekre elterjedté váltak az UART alapú soros kommunikációs megoldások, és ez beszivárgott a járműiparba. Az UART lényege, hogy az adatot, például egy bájtot, nem párhuzamosan küldünk és fogadunk több vezetéken, hanem mindezt, bitenként "sorosan" tesszük, ezzel az egyik legegyszerűbb, és legkevesebb vezetéket igénylő adatátvitelt megvalósítva. Az egyszerűsége, és ebből adódó alacsony költsége gyors elterjedést tett lehetővé a járműiparban, ez viszont ahhoz vezetett, hogy többféle fizikai megvalósítás és protokoll létezett párhuzamosan, így felmerült a szabványosításra való igény. A szabványosítást a LIN (Local Interconnect Network) Consortium tette lehetővé, melyet az Audi, BMW, DaimlerChrysler, VW, Volvo – Volcano Tech. és a Motorola hoztak létre.

2.3.1 LIN busz

A LIN busz egy master (mester), több slave (szolga) koncepciójú, melyre az állomások alacsony költségű UART alapú IC-vel csatlakoznak. A szolga állomások az órájuk szinkronizációját külön órajel vezeték nélkül képesek végrehajtani. A busz maximum 20 kbit/s sebességű jelátvitelre képes, a jelátvitel determinisztikus, a jelterjedési idő előre számítható. A hálózat tetszőlegesen újrakonfigurálható, bővíthető, az állomások közti interakció jelérzékelés alapú, valamint a busz támogatja a szállítási réteget és diagnosztikát.

2.3.2 LIN kommunikáció

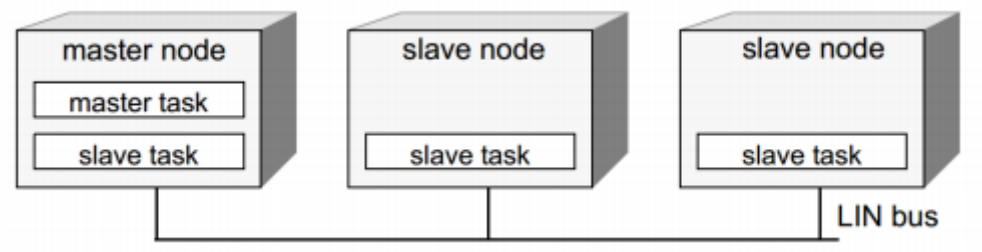
A LIN szabvány a kommunikációt négy rétegre osztja, melyek az OSI modellt valósítják meg (a viszonylati réteg kivételével), mely a következőképp néz ki:



5. ábra: A LIN hálózat rétegei

Sokszor a jelfeldolgozó és a keretkezelő egyben, szoftveresen megvalósítva alkot közös interfészt (API-t) az alkalmazás felé.

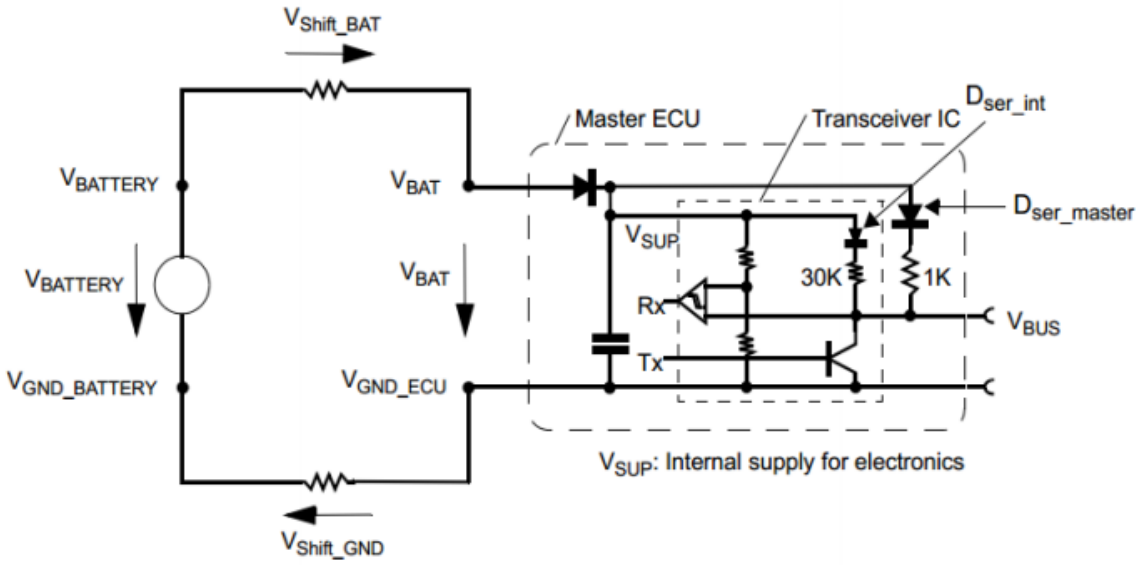
2.3.3 LIN hálózat logikai felépítése



6. ábra: A LIN hálózat logikai felépítése

A LIN hálózaton az állomások folyamatokat valósítanak meg, a hálózatot így egy mester, és több szolga folyamat valósítja meg, azzal együtt, hogy a mester csomópont is rendelkezik szolga folyamattal.

2.3.4 Fizikai réteg

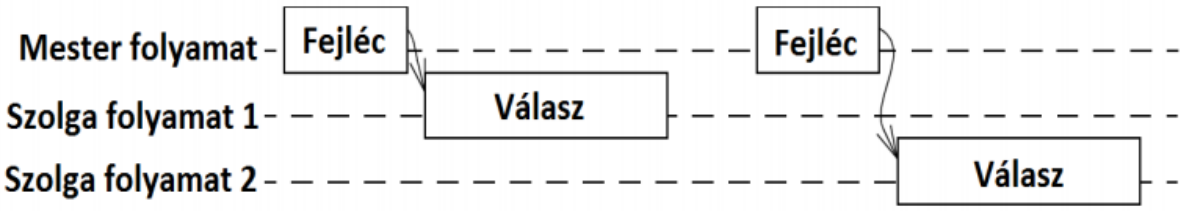


7. ábra: LIN transceiver felépítése

A busz “egy vezetékes”, azaz adatot egy vezeték továbbít, emellett szükség van egy földre is (VGND_ECU). A busz recesszív magas jelszintje jellemzően 12 volt, az alacsony szint a domináns, ami meghajtástól (küldés) és fogadástól függően a busz jelszintjének 20 illetve 40 százaléka.

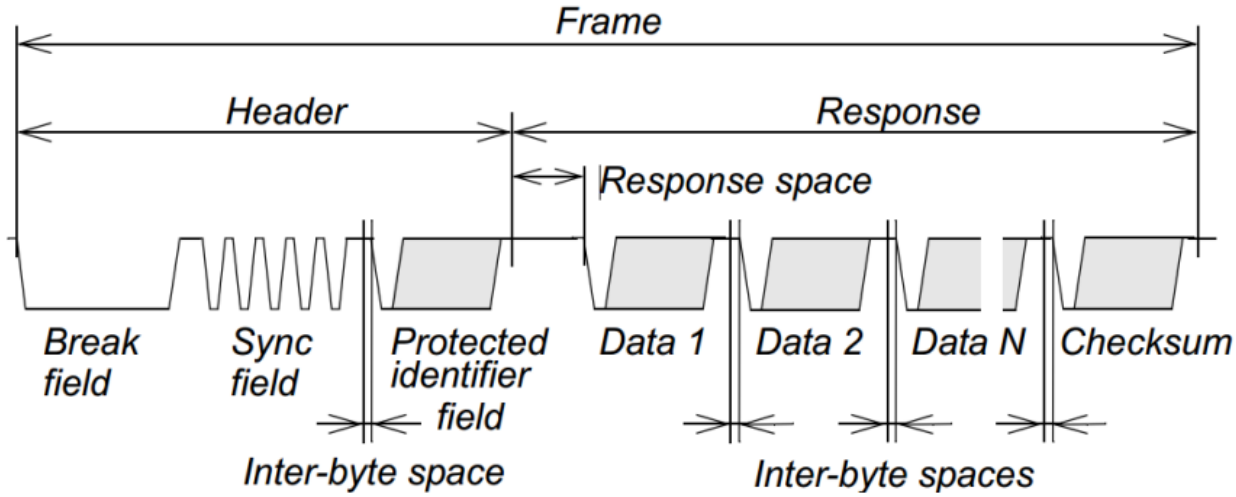
2.3.5 LIN keretek

Egy LIN keret a mester folyamat által szolgáltatott fejlécből (header), és a szolga folyamat által szolgáltatott válaszból (response) áll.



8. ábra: LIN kommunikáció vázlata

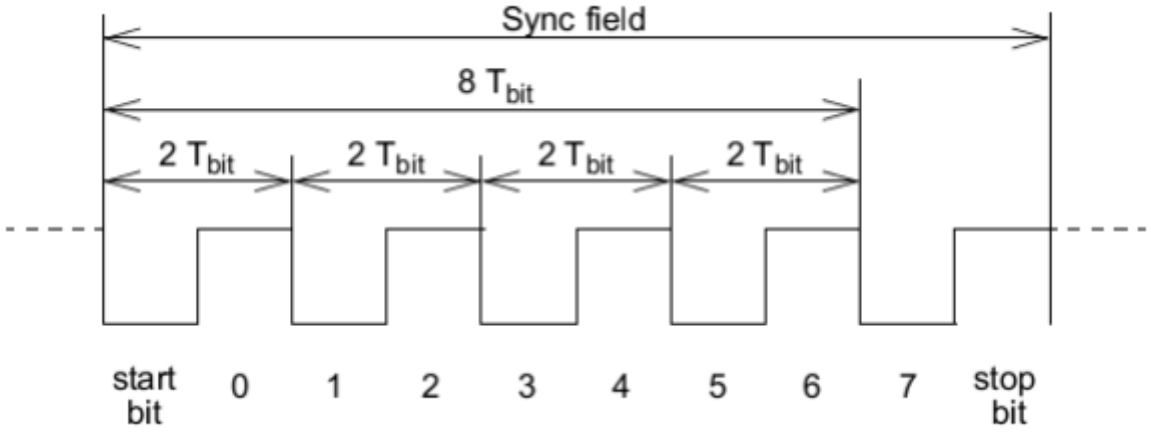
A CAN-hez hasonlóan az állomások itt is címezhetőek, a címzettet a keret azonosítója definiálja. Mivel az állomások közös buszon vannak, így lehetőség van arra is, hogy egy adatot több állomás is felhasználjon (multicast).



9. ábra: LIN keret felépítése

A keret fejléce három részből áll: Break Signal, mely legalább 13 bit hosszúságú kell, hogy legyen, és a kommunikáció kezdetét jelzi, ezt követi a szinkronizációs mező és a keretazonosító. A szolga folyamat által küldött válasz az adatmező(k)ből és egy ellenőrzőösszegből állnak.

A kereten belül a byte mezőknek is adott a formátuma: az LSB-MSB bitsorrendű bájtot egy logikai alacsony start és egy logikai magas stop fog közre, valamint ezt követi egy rövid bájtközi szünet.



10. ábra: LIN szinkronizációs mező

A szinkronizációs mező technikailag egy 0x55 byte, ami egy négyszögjelet eredményez, ebből pedig a szolga eszköz képes meghatározni egy bit hosszát a lefutó

élek távolságainak mérésével, ezzel megvalósítva minden keret kezdetekor a szinkronizációt.

Az egy bájtos azonosító mező (védett azonosító) két további almezőből áll: keretazonosító, és a hozzá tartozó paritás, az azonosító így 0..63 értéket vehet fel, mely alapján három csoportba sorolható:

- 0..59 (0x3B) a jeltovábbításra szolgáló keretek esetén
- 60 (0x3C) mester igény (master request) és 61 (0x3D) szolga válasz (slave response) a diagnosztikai keretek azonosítói
- a 62 (0x3E) és 63 (0x3F) azonosítók további protokoll bővítéshez vannak fenntartva.

Az adatmező hossza egytől nyolc bájtig terjedhet, de a keret hossza előre definiált kell, hogy legyen. A keret utolsó eleme az ellenőrzőösszeg, ami két féle lehet:

1. Klasszikus ellenőrzőösszeg, mely a diagnosztikai keretek és a LIN 1.x szabványok esetén alkalmazandó
2. Továbbfejlesztett ellenőrző összeg (Enhanced Checksum), amely a védett azonosítót is beleveszi a számításba, amelyet a LIN 2.x protokollt alkalmazó eszközök használnak.

2.3.6 Időzítés

A keretek időzítéséért a mester folyamat a fellelős, mely folyamatosan frissíti az időzítési táblát (schedule table), amiből adódóan minden keret egy meghatározott pozícióban (frame slot) keletkezik. A keretpozíciók hossza megegyezik a leghosszabb keret elméleti hosszával (plusz a keretközi idő). Mivel a keretek időzítéséért egyedül a mesterfolyamat felelős, ütközés csak speciális esetben alakulhat ki, illetve az ütemezési tábla garantálja a már említett előre számolható jelterjedési időt.

2.3.7 LIN keretek fajtái

A feltétel nélküli keret (Unconditional frame) egyszerű jelek szállítására szolgál, azonosítója 00..59 lehet.

Az eseményvezérelt keret (Event triggered frame) feladata a LIN hálózat sávszélességének optimalizálása, valamint segítségével több szolga figyelése is megoldható. A mester folyamat elküldi a feltételes keret fejlécét, melyre az egyik szolga

válaszol. Mivel ebben az esetben nem rögzített, hogy melyik állomás fog válaszolni, ezért ütközés léphet fel. Ez esetben a mester folyamat címzett keretekben lekéri az állomásoktól az adatokat.

A szórványosan megjelenő keretek (Sporadic frame) olyan adatokat hordoznak, amelyeknél nem feltétel, hogy ütemesen jelenjenek meg a hálózaton. Több ilyen keret foglalhatja el ugyanazt a keret pozíciót az időzítési táblában.

A diagnosztikai keretek (Diagnostic frame) a szállítási réteg információit hordozzák. Ez lehet mester igény (Master Request) vagy szolga válasz (Slave Response).

A fenntartott keretek (Reserved frames), melyek azonosítója 62 és 63, nem használhatóak LIN 2.x hálózatban, mivel ezek későbbi protokollbővítésre vannak fenntartva.

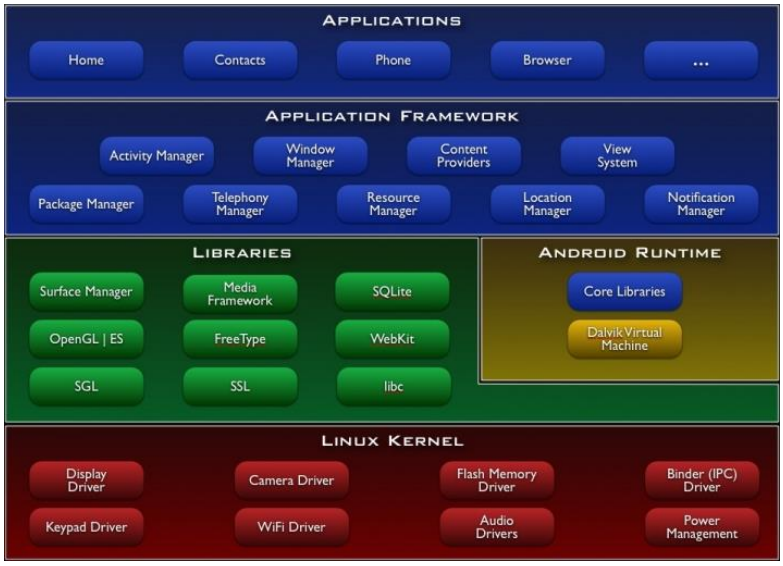
2.3.8 Jelek kezelése

Egy keretben két típusú adat kerülhet továbbításra: Jel (Signal), ami lehet skalár vagy byte tömb, és a keret adatrészébe kerül, valamint diagnosztikai üzenetek, amelyek a már említett, erre a célra fenntartott keretekben közlekedhetnek.

A skalár jelek hossza 1-16 bit lehet; az 1 biteset boolean jelnek nevezzük, a 2-16 bitesek unsignedként (előjel nélküli számként) kezelendők. Minden jelnek egy szolgáltatója van, de azt több keretben is küldheti, azaz a jelek átnyúlhatnak a kerethatárokon. Minden jelhez tartozik egy kezdeti érték, amely a jel első írásáig érvényesnek tekintendő. A byte tömb jelek 1-8 byte hosszúak lehetnek, és minden esetben kötöttek a keret bájt határaihoz.

2.4 Android fejlesztés

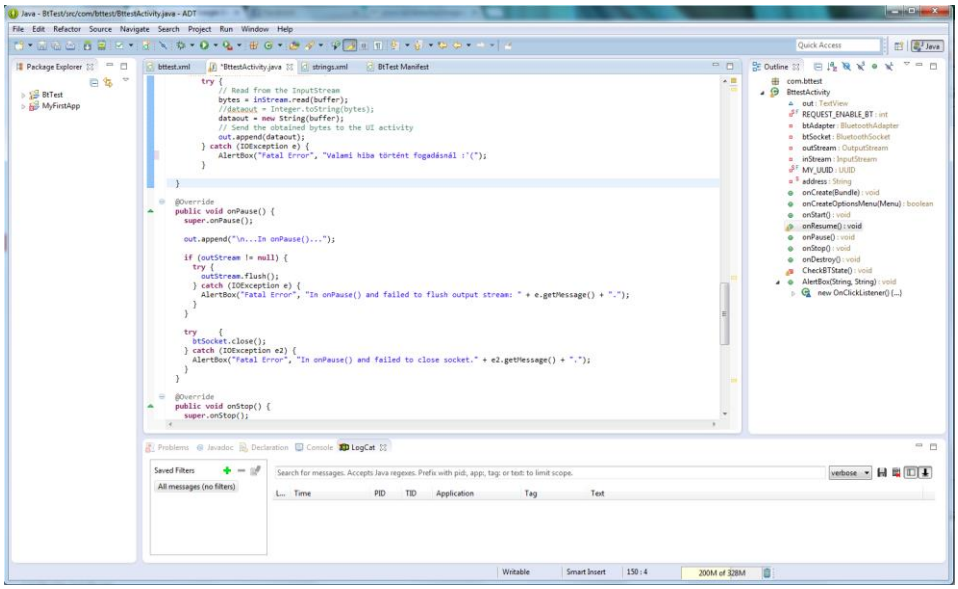
A projekt specifikációja szerint a mobilalkalmazás fejlesztés egyik támogatott platformja az Android operációs rendszer. Az Android egy, a Google által fejlesztett, mára igen népszerű mobil operációs rendszer. Alapja a gyári Linux kernel ARM alapú mikroprocesszorokra szánt változata, kiegészítve azt rengeteg szolgáltatással, amik gyorssá és hatékonnyá teszik az Androidra történő fejlesztést és a szoftverek használatát.



11. ábra: Android szoftver architektúra (18)

Az operációs rendszer lelke a Dalvik virtuális gép, egy Java futtató környezet, ami sok szempontból eltér az Oracle eredeti megoldásától, annak érdekében, hogy az okostelefonok relatív kis processzor- és memóriakapacitása mellett is megfelelően működjön.

Az Android rengeteg eszközt és szolgáltatást nyújt a fejlesztő számára, és érezhető, hogy a rendszer tervezésekor a fő koncepció az egyszerű használhatóság legyen.



12. ábra: Eclipse IDE, Android SDK

A fejlesztés fő eszköze az Android SDK (Software Development Kit - fejlesztő készlet) és az Eclipse IDE-hez (Integrated Development Environment - Integrált fejlesztői környezet) tartozó kiegészítések. Új Android projekt esetén az IDE biztosít egy program vázat, a kész programot pedig lehetőségünk van emulátoron futtatni, így még a mobil készülékre telepítés nélkül tesztelhető a program.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bttest"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.bttest.BttestActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

1. kódrészlet: AndroidManifest.xml

Egy Androidos alkalmazás legfontosabb eleme az AndroidManifest.xml (11), főbb szerepei:

- tartalmazza az alkalmazás Java csomag nevét, ami az alkalmazás egyedi azonosítójául szolgál
- leírja az alkalmazás komponenseit: Activity-k, háttérben futó folyamatok, rendszerüzenetek fogadása, tartalom szolgáltatók (adatkezelők). Itt találhatóak az osztályok nevei, amik a rendszerfunkciókat megvalósítják.

- leírja, hogy a programnak milyen jogosultságokra van szüksége bizonyos rendszerkomponensek eléréséhez (pl.: Bluetooth rádióhoz való hozzáférés)
- leírja, hogy más programoknak milyen jogosultságokkal kell rendelkezniük, hogy kommunikálhassanak egymással
- leírja a legalacsonyabb Android API szintet, ami szükséges a program futtatásához
- tartalmazza azon könyvtárak listáját, amik a program futtatásához elengedhetetlenek

Az Android alkalmazások megkövetelik az MVC (Model-View-Controller – Modell-Nézet-Vezérlő) felépítést, aminek a lényege, hogy a program megjelenése, a felhasznált erőforrások kezelése, és az ezek közötti kapcsolatért felelős vezérlés teljes mértékben elkülönüljön. Ez a szoftvertervezési minta lehetővé teszi, hogy az adatok érkezése, feldolgozása és megjelenítése egymástól teljesen függetlenek legyenek, betartása esetén pedig bármelyik komponens könnyen lecserélhető, fejleszthető.

Esetünkben a Modell része a Bluetooth kapcsolat létrehozása, az adatfolyam fogadása és feldolgozása. A Nézet felel a feldolgozott adatok megjelenítéséért a grafikus felhasználói felületen (GUI), amit a Bosch UX (User Experience – felhasználói élmény) részleg segítségével fogunk megtervezni. A Vezérlő fő feladata a felhasználói interakciókra (pl.: gomb megnyomása a felületen) való válaszadás, illetve szükség esetén a modellbe történő beavatkozás.

Android környezetben a grafikus felület leírására egy XML fájl szolgál, melyben a HTML-hez hasonlóan fa struktúrába szervezett blokkokkal írhatjuk le a felület kinézetét úgy, hogy a blokkjaink a standard építőelemek (gomb, szövegmező, címke, alakzatok, stb.), az attribútumok pedig a blokk paramétereit határozzák meg.

```
<TextView
    android:id="@+id/out"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

2. kódrészlet: TextView elem

A képen látható példában a TextView elem egy szövegdobozt jelöl, aminek az azonosítója @+id/out így később a programkódban a tartalmát az R.out változón

keresztül érjük el, szélessége és magassága pedig kitölti a szabad területet (layout_width, layout_height = ”wrap_content”).

2.5 iOS fejlesztés

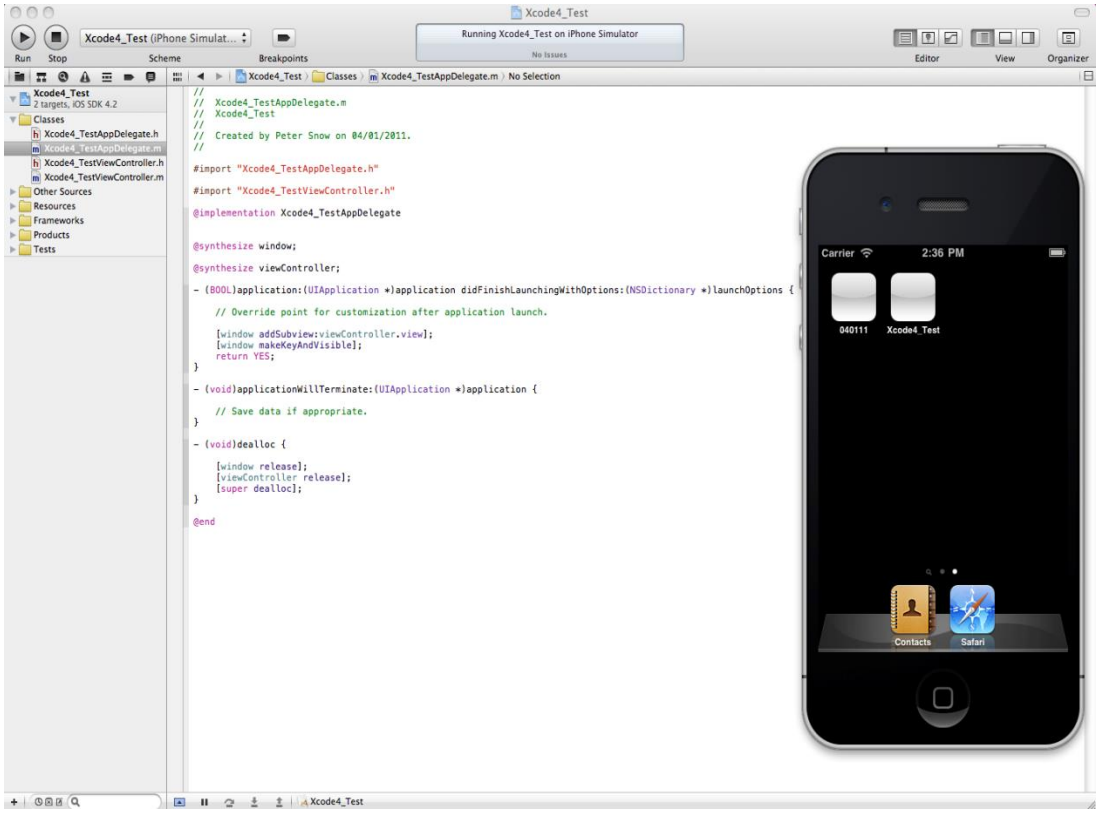
Az iOS fejlesztői környezete az Xcode, hivatalos programozási nyelve az Objective-C. Ez tulajdonképpen egy objektum orientált vékony réteg a hagyományos C nyelv felett, azonban a C++-hoz hasonló programozási nyelvektől eltérően nem a függvényhívásokon alapul, hanem az objektumok közötti üzenetküldésen.

A különböző Apple termékekre való fejlesztés, valamint a fejlesztéshez szükséges előkövetelmények hamar eltántoríthatják a különböző egyéb fejlesztői környezethez szokott felhasználókat. A legelső lépés, mivel az Xcode-ot csak Macintosh gépen lehet elindítani egy ilyen típusú számítógép beszerzése. A második nagyobb lépés az a fejlesztői licenszek megvásárlása. Itt két különböző típus közül választhatunk attól függően, hogy mint magánszemély, vagy mint cég szeretnénk beregisztrálni; mindkét esetben az éves díj 99 dollár. Van lehetőség a fejlesztői környezet ingyenes használatára is, azonban ilyenkor elveszítjük a készülékekre való telepítés lehetőségét és csak a beépített szimulátorral fogunk tudni valamit is kezdeni, valamint a magasabb szintű online támogatási lehetőségek is megszűnnek.

2.5.1 Az emulátor

Az Xcode beépített emulátora egy igen hasznos és praktikus segédeszköz a megírt programok tesztelésére, azonban vannak komoly limitáltságai is, amelyeket figyelembe kell vennünk. A legfontosabb, hogy a szimulátor az aktuális számítógép memóriájából gazdálkodik, ami természetesen jóval nagyobb, mint amivel egy iPhone/iPad rendelkezhet. Így például egy létrehozott program pontos teljesítmény, vagy memória igényének a becslésére nem teljesen alkalmas. Szintén ide tartozik, hogy az emulátor hiába rendelkezik rengeteg funkcióval, természetesen egy Macintosh gépben nem mindig lesz megtalálható pl. egy giroszkóp, GPS modul, vagy egy gyorsulásszenzor, amelyek a mai okos készülékekben már szinte mindenütt jelen vannak. Vagyis azon programok, amelyek ezen szenzorokra támaszkodnak szintén nem fognak megfelelően működni. Ami számunkra jelen pillanatban különösen fontos, hogy hiába rendelkezik számos Macintosh gép belső Bluetooth LE kommunikációs eszközzel, ezeket az

emulátor alapesetben sajnálatos módon nem tudja felhasználni. Ezt a problémát egy külső, pl. USB-s Bluetooth LE modul csatlakoztatásával ki lehet küszöbölni.



13. ábra: Xcode fejlesztői környezet iPhone emulátorral

2.5.2 Bluetooth az iOS-ben

Az iOS az Androiddal ellentétben a hagyományos Bluetooth profilok többségét nem támogatja. A nem támogatott profillal rendelkező modulokra való fejlesztést csak új, saját gyártmányú Bluetooth eszköz/kellék legyártásával lehetett eddig megoldani. Kisebb papírmunka után a fejlesztők csatlakozhattak az Apple MFi (Made for iPod/iPhone/iPad) programjához, amelynek során a kézhez kaphatták többek közt a hardware létrehozásához szükséges beépített eszközillesztőkről és komponensekről szóló dokumentumokat, és a szükséges belső kommunikációs protokollok részletes leírását.

Erre a hosszadalmas folyamatra az operációs rendszer 6-os vagy a feletti verziójával, a Bluetooth LE (Low Energy) eszközök megjelenésének, és a személyre szabható profilok létrehozhatóságának köszönhetően már nincs szükség.

2.5.3 Fejlesztői profilok

Az aktuális készülékre való fejlesztésnek (Development), vagy az AppStore-ra való feltöltésnek (Distribution) három feltétele van:

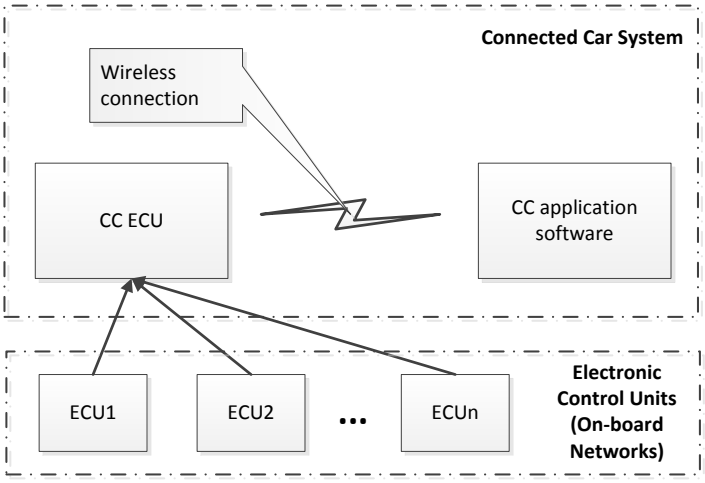
- az adott fejlesztőnek rendelkeznie kell egy az Apple által kibocsájtott az adott számítógépre specifikus tanúsítvánnyal,
- az applikációnak rendelkeznie kell egy egyedi AppId-val, ez szabadon választott, jelentősége inkább az online boltba való feltöltésnél van,
- legvégül szükséges egy egységes Provisioning Profile, vagy szolgáltatói profil, ez összefoglalva tartalmazza a különböző felhasználókról és a hozzájuk rendelt készülékekről szóló engedélyeket.

3 Architektúra

Ebben a fejezetben bemutatjuk a CC-ECU hardverével és szoftverével szemben támasztott követelményeket, valamint ismertetjük a főbb komponensek kiválasztásának szempontjait, valamint technikai paramétereit.

3.1.1 Rendszerarchitektúra

A magas szintű rendszerarchitektúra a lenti ábrán látható. A CC ECU összegyűjti a szükséges adatokat a CAN és LIN hálózatokról, majd az előfeldolgozott információt továbbítja az okostelefon(ok)nak (Android vagy iOS) Bluetooth 4.0 Serial Port Profile-on keresztül egy saját alkalmazásszintű protokollt használva.



14. ábra: Rendszerarchitektúra

Az okostelefonos alkalmazás fogadja, feldolgozza és megjeleníti az információt egy grafikus felhasználói felületen (GUI) az aktuális konfigurációnak megfelelően.

A CC ECU hardverét és szoftverét egyaránt ki kell fejleszteni. Az okostelefonok esetében a szoftvert és az ergonomikus GUI-t az adott operációs rendszernek megfelelően kell megtervezni és implementálni.

3.1.2 Rendszerállapotok

7. táblázat: A főbb elektromos komponensek állapotai az adott rendszerállapotokban

	uC	BT modul	CAN trans.	LIN trans.
INIT	N/A	N/A	N/A	N/A
NORMAL	Aktív	Aktív	Aktív	Aktív
FLASH	Aktív	N/A	N/A	N/A
CAN/LIN Készenlét	Aktív	Aktív	Inaktív	Inaktív
CAN/LIN + uC Készenlét	Inaktív	Aktív	Inaktív	Inaktív
ECU Készenlét	Inaktív	Inaktív	Inaktív	Inaktív

3.1.2.1 INIT (kikapcsolás)

A CC-ECU kezdeti állapota bekapcsolás vagy reset után.

3.1.2.2 NORMAL

A CC-ECU rendes állapota, miután az inicializálás/flashelés megtörtént. A CC-ECU aktiválja a kommunikációs interfészeit, és megpróbál kapcsolatot teremteni a rendszer elemeivel.

3.1.2.3 CAN/LIN STANDBY

Ebben az állapotban a CC-ECU deaktiválja a vezetékes kommunikációs interfészeit. A CC-ECU adóvevői készenléti (alacsony fogyasztású) módban vannak, amíg az adott busz inaktív. (Az adóvevők egymástól függetlenül deaktiválhatóak.) A CC-ECU jelez a csatlakoztatott okostelefonnak a vezeték nélküli interfészen, hogy nincs ECU csatlakoztatva. A CC-ECU aktiválja a vezetékes kommunikációs interfészeit,

amennyiben valamely buszon található csomópontnak vagy az okostelefonnak adat küldési vagy fogadási igénye van.

3.1.2.4 CAN/LIN + uC STANDBY

Ebben az állapotban a CC-ECU deaktiválja a vezetékes kapcsolatait és a mikrokontrollert. Ebben az állapotban a CC-ECU képes Bluetooth párosítási kérelmeket fogadni. (A Bluetooth modul a párosítási kérelmeket képes önállóan feldolgozni.)

3.1.2.5 ECU STANDBY

Ebben az állapotban a CC-ECU deaktiválja (készenléti állapot) az összes rendszer összetevőt. A CC-ECU nem tudott csatlakozni egy okostelefonhoz sem a vezeték nélküli interfészen, vagy az összes közelben található és párosított telefonon bezárják az alkalmazást. A CC-ECU képes aktiválni a vezeték nélküli interfészt, ha egy, már előzőleg párosított okostelefon a hatótávolságon belül van, és azon fut az alkalmazás. A CC-ECU képes aktiválni a vezeték nélküli interfészt, amennyiben párosítási kérelem érkezik. (Csak a Bluetooth modul kerül aktiválásra a párosításhoz.)

3.1.2.6 FLASH

Ebben az állapotban a CC-ECU alkalmas a szoftver frissítések letöltésére. A CC-ECU normál állapotba kerül, amint a flashelés sikeresen befejeződött. Amennyiben a folyamat sikertelen volt, a CC-ECU egy belső reset után újra normál módba kerül.

3.2 Hardver architektúra

Az alábbiakban bemutatjuk a CC-ECU-val szemben támasztott fő hardver követelményeket, valamint a hardver felépítését és a fő moduljait.

3.2.1 A CC-ECU fő követelményei

- CAN 2.0B interfész
- 2 db LIN 2.0 interfész
- Bluetooth 4.0 Serial Port Profile támogatás (és lehetőség 8 készülékes hálózatra)
- Automotive Grade 1 működési hőmérséklettartomány (–40 °C-tól +125 °C-ig) az univerzális felhasználhatóságért az autó motorterében
- IP65-ös por és pára elleni védelem és olajálló burkolat (Az IP – Ingress Protection Rating – az elektronikus szerkezet borításának por, víz és egyéb szennyeződés bejutása elleni védelméhez rendel egy mérőszámot. Az IP kód meghatározása szabványos vizsgálatok alapján történik. Az első szám a szilárd

szennyeződések elleni zárást jelzi, a 6-os a lehető legjobb, tökéletesen záró borítás. A második szám a vízállóságot jelzi, az 5-ös a folyamatos vízsugár ellen is védelmet biztosít.) (12)

- az autóval kompatibilis csatlakozási lehetőségek
- direkt és biztonságos csatlakozás az autó elektronikus rendszeréhez

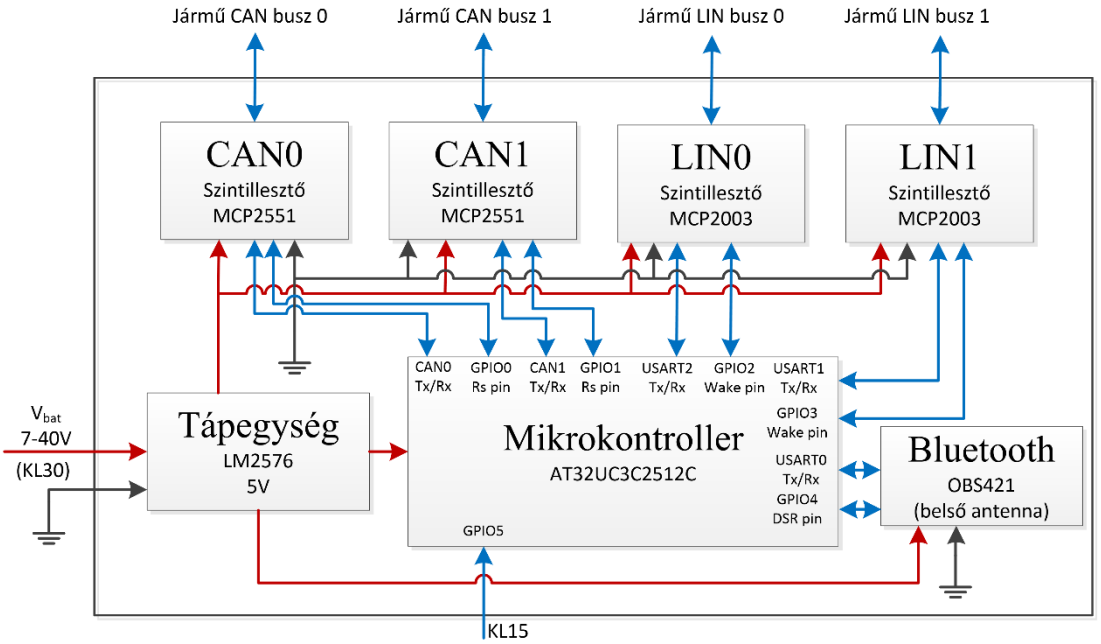
3.2.2 A CC-ECU hardvere

A járműmechatronika szakirányon az Atmel AVR mikrokontroller család tagjai használatosak. A mikrokontroller kiválasztásánál ezek közül két lehetőség jött szóba.

Az egyik a 8 bites megaAVR-ek közül az Atmel AT90CAN128-alapú ECU. Ez teljesíti az Automotive Grade 1 követelményeit, és rendelkezik egy CAN kontrollerrel, ellenben nincs LIN interfésze. A megaAVR család termékeit áttekintve csak olyan mikrokontrollereket találhatunk, amelyek csak egy CAN és egy LIN interfésszel rendelkeznek. Ebben az esetben az interfészek számát megnövelhetnénk egy MCP2515SPI-CAN modul mikrochippel, és egy UART fizikai interfészen LIN-t implementáló szoftverrel, esetleg további kisebb megaAVR kontrollerekkel egy közönséges SPI buszon keresztül. A legfőbb előnye ennek a felépítésnek a bővíthetőség, viszont sokkal komplikáltabb szoftvert igényel.

A másik lehetőség a 32 bites Atmel AVR család volt, amely a következő autóipari termékeket kínálja: AT32UC3C0512C, AT32UC3C1512C, AT32UC3C2512C. Ezeknek mindegyike rendelkezésünkre bocsát 2 független CAN kontrollert és legalább 4 LIN-képes USART modult, emellett erősebb processzoruk és több perifériájuk van.

A két lehetőség közül a második mellett döntöttünk az egyszerűbb szoftver, és az erősebb hardver miatt. A 15. ábrán a hardver felépítés vázlata látható: az Atmel AT32UC3C2512C mikrokontroller, Atmel CAN és LIN vevők és egy Texas Instruments LM2576 autóipari tápegység. Ezek mindegyike teljesíti az Automotive Grade 1 követelményeit. A Bluetooth modul egy ConnectBlue OBS421 típusú készülék beépített antennával. Ez az elem feltétlenül szükséges az Apple iOS-szel való kompatibilitás érdekében, viszont ez nem teljesíti az Automotive Grade 1 hőmérsékletkövetelményeit, az adatlapja szerint, csak –25 °C-tól +80 °C-ig garantálják működőképességét.



15. ábra: A hardver architektúra

(Az ábrán a tápegységbe csatlakozó KL30 az akkumulátor pozitív pólusát jelenti, a KL15 a gyújtáskor működő áramkört.)

3.2.3 A hardver komponensei

3.2.3.1 A központi mikrokontroller egység

A 32 bites Atmel AVR család három típusa a kommunikációs, az ADC, és az időzítő/számláló csatornák számában tér el egymástól. Ezek közül az AT32UC3C2512C-re esett a választás, ennek a főbb tulajdonságait ismertetjük a következőkben.

- nagy teljesítményű, alacsony fogyasztású mikrokontroller
  - 32 bites működés (gyorsabb műveletvégzés 4 bájtos adatokkal)
  - kompakt egykörös RISC utasításkészlet (A RISC – Reduced Instruction Set Computing – csökkentett utasításkészletű processzor: utasításkészlete korlátozva van egy kisebb körre, viszont ezeket az utasításokat sokkal gyorsabban képes elvégezni; ez gyakran használt a mikrokontrolleres környezetben a kis erőforrások gazdaságos kihasználása miatt. Az egykörös azt jelenti, hogy az utasítások elvégzése és következő utasítás beolvasása egyszerre történik, így minden órajelciklusban elvégezhető egy művelet.)

    - DSP utasításkészlettel rendelkezik (Digital Signal Processor; néhány bonyolultabb utasítást tartalmaz olyan speciális feladatokra, amelyeket a processzor működése során sokszor végez. Ezzel az extra utasításkészlettel, a jelfeldolgozással kapcsolatos feladatait sokkal gyorsabban tudja elvégezni.)
    - a működési frekvenciától függően MHz-enként 1,49 DMIPS teljesítmény (Dhrystone alapján mért (fixpontos) millió utasítás másodpercenként)
- 512 Kbyte-os belső flash memória
    - 15 éves tervezett adattárolási képesség, 10000 írási ciklus (kb. 10000-szer írható a memóriája)
- 64 Kbyte-os belső nagy sebességű SRAM (Static Random Access Memory)
- egy kétcsatornás Controller Area Network (CAN) vezérlő
    - a CAN2A és CAN2B protokollnak megfelelő
    - magas szintű üzenetkezelési képesség
    - két független csatorna, csatornánként egyszerre 16 üzenetes feldolgozási kapacitás
- 4 Universal Synchronous/Asynchronous Receiver/Transmitter (USART) csatlakozás mindegyikre független bitráta generátor, LIN támogatás
- 64 lábú TQFP áramkör elrendezés
- két üzemeltetési feszültségsáv: 3,3 és 5 V
- autóipari minősítés
    - fejlesztés és gyártás az ISO-TS 16949 legszigorúbb követelményei szerint
    - teljes Automotive Grade 1 hőmérséklet-tartomány (–40 °C-tól +125 °C-ig)
- fogyasztás (3,3 V-nál)
    - Aktív: 10,24 mA (20MHz-nél)
    - Készenlét: 73µA (Stop módban)

3.2.3.2 CAN szintillesztő

Sok különböző típusú CAN és LIN szintillesztő érhető el a piacon, amelyik rendelkezik a szükséges csatlakozással. Mi az MCP2551 nagy sebességű, nagy hőmérséklet-tartományú CAN szintillesztő mikrochipet választottuk, ennek a fizikai paraméterei a következők:

- 1 Mbit/s adatátviteli sebesség
- teljesíti az ISO-11898 standard fizikai réteg követelményeket (OSI modellben a fizikai réteg)
- a négyszögjel lejtésének külső vezérlése a rádióhullám kibocsátás csökkentésére (Magyarázat: a négyszögjel gyors feszültségugrásai rádióhullámokat generálnak; ezt a jelenséget a feszültségugrás meredekségének csökkentésével kezelhetjük.)
- földelési hiba felismerés, nagy feszültségesés elleni védelem, alacsony tápfeszültség elleni védelem, rövidzár okozta károsodás és nagyfeszültségű tranziensek elleni védelem, túlmelegedés következtében automatikus kikapcsolás
- ha energiaellátási probléma adódik, nem zavarja a CAN buszt
- alacsony áramú standby működés
- érzéketlen a külső elektromágneses hatásokra a CAN differenciális busz rendszerének köszönhetően
- energiafogyasztás
  - Aktív: 75 mA domináns és 10 mA recesszív busz állapot esetén
  - Készenlét: 465 uA
- teljes Automotive Grade 1 hőmérséklet-tartomány (–40 °C-tól +125 °C-ig)

3.2.3.3 LIN szintillesztő

Erre a célra az MCP2003 LIN szintillesztő mikrochipet választottuk. Tulajdonságai:

- megfelel a Local Interconnect Network (LIN) 1.3, 2.0 és 2.1 busz specifikációknak
- adatátviteli sebességek támogatása 20 Kbit/s-ig
- jól ellenáll az elektromágneses interferenciás hatásoknak és rádióhullámok hatásának
- túlmelegedés hatására automatikus kikapcsolás
- nagy elektrosztatikus feltöltődés elleni védelem
- teljes Automotive Grade 1 hőmérséklet-tartomány (–40 °C-tól +125 °C-ig)
- standard USART interfész
- alacsony fogyasztású mód
  - csak fogadja az üzeneteket, a küldési funkció ki van kapcsolva

- energiafogyasztás
  - Aktív: 150µA
  - Készenlét: 15µA

3.2.3.4 *Bluetooth modul*

A Bluetooth modul előkövetelményei voltak, hogy támogatnia kell a Bluetooth 4.0 Low Energy-t, az iOS-t és a Serial Port Profile-t. A Bluetooth modulunk a ConnectBlue OBS421-es típus beépített antennával, bővebb információ róla a Bluetooth részben található.

3.2.3.5 *Tápegység*

A mikrokontrollerek egység táplálását egy feszültségszabályozó végzi. Ez az autó akkumulátorától kapja a táplálást, feladata, hogy a bemenő feszültségtől és a terheléstől gyakorlatilag függetlenül egy adott értékű állandó kimeneti feszültséget biztosítson. A mi mikrokontrollerünkhöz egy széleskörűen elterjedt típust, a Texas Instruments LM2567 3A típusú feszültségszabályozót használjuk. Ezt az autóiparban sok helyen használják a fedélzeti egységekben (pl. flottamenedzsment rendszerek). Főbb jellemzői:

- állandó 3,3 V-os és 5 V-os kimeneti feszültség, 3 A kimeneti áram
- széles bemeneti feszültségtartomány (40V-ig)
- egyszerű referencia áramkör (beszereléshez nem igényel sok áramköri elemet), külső jelre történő kikapcsolási lehetőség
- 77%-os hatékonyság 12 V és 3 A-nél
- túlmelegedés hatására kikapcsolás és áramhatároló védelem
- teljes Automotive Grade 1 hőmérséklet-tartomány (–40-től +125 °C)

3.3 Szoftver architektúra

Az okostelefonos szoftverrel szembeni követelmények

- a szoftvernek támogatnia kell az Android 4.0 (és újabb) valamint az Apple iOS6 (és újabb, iOS7 előnyben) mobil platformokat.
- a szoftver a Bluetooth interfészt Serial Port Profile-on keresztül kezelje
- a szoftvernek a Bluetooth Protocol Stack-en belül kell megvalósítani az alkalmazásszintű kommunikációs protokollt

- a szoftvernek a gyűjtött adatokat az okostelefon felhasználói felületén kell megjelenítenie. A gyűjtött adatok ergonomikus, modern felhasználói felületen jelenjenek meg.
- a felhasználói felületnek újrakonfigurálhatónak kell lennie, és ezek a beállítások visszaállíthatóak kell, hogy legyenek
- a szoftvernek kezelnie kell a gépjárművek konfigurációs adatait, valamint a CAN és LIN üzenet leírókat.

4 Prototípusrendszer

4.1 A kommunikációs protokoll

Ebben a fejezetben bemutatjuk a CC-ECU és az okostelefon közötti kommunikáció felépülését és menetét, illetve szó lesz a küldött üzenetek tartalmáról és az ilyenformán megvalósult kommunikáció lehetőségeiről, valamint további lehetőségekről. Bevezetésnek ez utóbbiakat tárgyaljuk.

4.1.1 Az autó-telefon kapcsolat tervei, lehetőségei

Amikor a felhasználó az okostelefonjával az autóhoz kellő közelségbe ér, bekapcsolja az autó gyújtását és a mobilalkalmazást, a telefon és az ECU között automatikusan létrejön az adatkapcsolat. Ettől kezdve a telefon kéréseket intéz az ECU-hoz, hogy milyen üzeneteket szeretne kapni. Az ECU folyamatosan szűri az üzeneteket az autó hálózatán, és az adott azonosítójú, éppen aktuális üzenetet a lentebb ismertetett formában elküldi Bluetooth-on az okostelefonnak. Az ECU-hoz tartozik egy konfiguráció arra vonatkozóan, hogy milyen üzeneteket szűrjön a CAN hálózatról, továbbá a telefon is rendelkezik egy konfigurációval, amely az egyes jelek értelmezéséhez szükséges információkat tartalmazza.

A vezeték nélküli hálózathoz több okostelefon is csatlakozhat (7db), ezek mint slave-ek kommunikálnak a masterrel különböző csatornákon, az ECU pedig egyenként küldi nekik az üzeneteket (csak annak a telefonnak küldi ki a lekérdezett adatot, amelyik lekérdezte). Az ECU a memóriájában tárolja, hogy melyik csatornán melyik telefon található.

A tervek szerint az ECU első beszerelésekor, mivel még nem tudja, milyen üzeneteket kell szűrnie, feltérképezi, hogy milyen ECU-k vannak a hálózaton (megszerzi az ECU-k azonosítóját, gyártóját, stb.), és ezt elküldi az okostelefonnak. A telefon a kapott adatokat interneten keresztül felküldi a Bosch szerverére, és letölti onnan az ECU-khoz tartozó specifikációkat, hogy milyen adatokat szűrjön le tőlük a CC-ECU; ezeket az információkat továbbítja a CC-ECU-nak. Ezen kívül letölti magának, hogy az egyes ECU-któl leszűrt adatokat hogyan dolgozza fel. Ezek a konfigurációk tárolódnak mind az okostelefon, mind a CC-ECU memóriájában, így a következő csatlakozás már lényegesen rövidebb idő alatt fog lezajlani.

4.1.2 A kommunikáció alapja (13)

Ahogy a kommunikáció hardveres megvalósításának tárgyalásakor már említettük, a CC-ECU által a CAN hálózatból kiszűrt üzeneteket a Bluetooth modul küldi tovább az okostelefonnak, amely feldolgozza azokat. (A kommunikáció felépülése során más üzenetek is cserélődnek, ezeket később részletesen ismertetjük.) A folyamat pontos leírásához először a Bluetooth modul képességeit tárgyaljuk.

A ConnectBlue OBS421 Bluetooth modulnak háromféle működési módja van: AT Mode (konfigurációhoz), Data Mode és Extended Data Mode. Data módban minden adat, amit a mestereszköz küld, eljut az összes eszközhöz, amelyik a hálózat része; továbbá a szolgák által küldött adatok eljutnak ugyan a mesterhez, de az nem tudja beazonosítani, hogy melyik szolgától jött. A mi projektünk szempontjából fontos, hogy az ECU egyenként tudjon kommunikálni a telefonokkal, ezért jó lenne, ha a kommunikációs protokollunkban nem kéne foglalkozni az eszközcímzéssel. Erre jelent megoldást az Extended Data Mode, amely mindezeket a lehetőségeket képes nyújtani.

Ez a mód a modul és az ECU közötti kommunikációt terjeszti ki. Extended Data módban az ECU az üzenetet ellátja címzéssel, és így a Bluetooth modul irányítottan képes kiküldeni azt az egyik telefonhoz, illetve a másik irányból érkező üzenetekhez a modul hozzárendeli a küldő azonosítóját, így az ECU tudni fogja, hogy melyik telefontól érkezett. Az Extended Data Mode által az üzenetből előállított csomag a következőképpen néz ki:

Start: 0xAA	Lefoglalt terület (4 bit)	Adatcsomag hossza (12 bit)	Adatcsomag	Stop: 0x55

16. ábra: Extended Data Mode csomag

Mint látható, minden csomag 1 bájton tárolt „AA” hexadecimális kóddal kezdődik és az „55” hexadecimális kóddal záródik le. Ezek között van 4 bit későbbi felhasználásra lefoglalt terület, illetve 12 bit-en tárolva az érkező adatcsomag hossza, amely adatcsomag rögtön ezután található. Így tehát ha pl. az ECU felé küldünk egy üzenetet, akkor az az „AA”-tól kezdve fogadja a biteket egészen, amíg megkapja az „55”-öt. Ezután kezdi a fogadott adatok feldolgozását.

Ezen belül maga az adatcsomag még 3 részre tagolódik:

Azonosító (Identifier) (12 bit)	Típus (4 bit)	Üzenet (az adatcsomag hossza – 2 byte)

17. ábra: Az Extended Data Mode adatcsomagja

Az üzenet többféle lehet: esemény, jelzés, válasz, kérelem, megerősítés vagy parancs. A típus adja meg 4 biten, hogy ezek közül melyik kategóriába tartozik az üzenet, az azonosító pedig azt mondja meg, hogy a kategóriákon belül milyen üzenetről van szó (pl. csatlakozási esemény – Connect Event). Az azonosító és a típus együttvéve 2 byte, ezért maga az üzenet a teljes adatcsomagnál 2 byte-tal rövidebb. A Bluetooth modul 9-féle azonosító-típus kombinációt definiál, a mi szempontunkból fontosaknak egy rövid leírását a következő táblázat tartalmazza:

8. táblázat: Az Extended Data Mode típusai

Azonosító + típus	Név	Leírás
0x0011	Csatlakozási esemény (Connect Event)	Egy csatlakozás létrejöttekor küldi a modul az ECU-nak.
0x0021	Lecsatlakozási esemény (Disconnect Event)	Egy csatlakozás megszakadásakor küldi a modul az ECU-nak.
0x0031	Adat esemény (Data Event)	Adat érkezésekor küldi a modul az ECU-nak.

0x0036	Adat parancs (Data Command)	Utasítás adatok küldésére, az ECU-tól érkezik a modulhoz.
0x0056	Parancs a csatlakozási események újraküldésére (Resend Connect Events Command)	Parancs a modulnak, hogy küldje újra a csatlakozási eseményeket a meglévő kapcsolatokról; pl. rögtön az ECU beindulása után vagy újraindításakor használható.

A fentiek jobb érthetősége kedvéért ismertetjük a CC-ECU és a Bluetooth modul közötti kommunikációt a rendszer működése során.

Amikor egy kapcsolat létrejön, a Bluetooth modul küld egy csatlakozási eseményt az ECU-nak benne foglalva a csatorna azonosítóját, amelyre a csatlakozás történt, a használt Bluetooth profillal és a csatlakozó eszköz Bluetooth címével. Tehát egy csatlakozási esemény adatrésze a következőképpen néz ki:

Azonosító + típus: 0x0011	Csatorna (1 byte)	Csatlakozás típusa: 0x01	Profil: 0x00 SPP esetén	Bluetooth cím (6 byte)	Keret mérete (2 byte)

18. ábra: Egy csatlakozási esemény adatrésze

A csatlakozás típusa Bluetooth-os csatlakozás esetén 0x01 hexadecimális kódú, a Bluetooth profil típusa SPP esetén 0x00, a keret mérete pedig azt adja meg, hogy maximálisan mekkora üzenet küldhető a csatlakozott eszköznek adat parancsban, illetve mekkora fogadható tőle adateseményben.

Amikor egy kapcsolat megszakad, a modul egy lecsatlakozási eseményt küld, amelynek az egyik mezőjében az azonosító + típus, a 0x0021 található, a másikban pedig a csatorna azonosítója, amelyen véget ért a kapcsolat.

Adat érkezésekor a Bluetooth modul egy adat eseményt továbbít az ECU-nak, benne a küldő csatorna azonosítójával és a fogadott adatokkal. A telefonok irányába adat kiküldéséhez az ECU-nak egy adat parancsot kell küldenie a modul felé a küldeni kívánt adatokkal és a címzett csatornájának azonosítójával.

Ha az ECU éppen bekapcsol vagy újraindul, a telefonoknak még lehetnek aktív kapcsolatai a modullal. Az aktív kapcsolatokról információt kell szereznie az ECU-nak,

ennek érdekében küldhet a Bluetooth modulnak egy parancsot a csatlakozási események újraküldésére.

4.1.3 Az okostelefon és az ECU közti kommunikáció folyamata

Az okostelefon és az ECU közötti kommunikációval kapcsolatban az alábbi megkötéseket tettük.

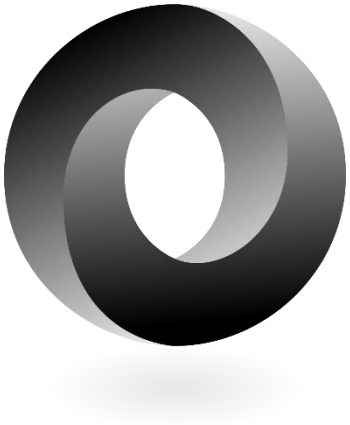
- Az mobilalkalmazás indulásakor az okostelefon kezdeményezi a Bluetooth kapcsolatot.
- A telefon egy login üzenetet küld és kéri az ECU konfigurációját. Ha még nincs ilyen, vagy már elavult, a telefon küld neki egyet (a rövid távú tervek szerint ezt az okostelefon a memóriájában tárolja, a hosszú távúak szerint az internetről tölti le).
- A CC-ECU kérésre küld információt, de az okostelefon azt is kérheti, hogy meghatározott időközönként folyamatosan küldje az adatokat.
- Az alkalmazás kikapcsolási eseményének hatására a telefon kijelentkezik az ECU-tól, de az ECU a véletlen szétkapcsolódást is kezeli.

4.1.4 Üzenetek

Az alábbiakban ismertetjük az általunk kifejlesztett, JSON-alapú protokollban használatos üzeneteket.

4.1.4.1 JSON – a leíró protokoll

Ahhoz, hogy az ECU-k által a mikrokontrollerhez küldött adatokat feldolgozás után küldhető és a mobiltelefonok számára értelmezhető formára hozzuk, szükségünk van egy leíró nyelvre. Egy szabványosított leíró nyelvet, a JSON-t választottuk erre a célra. Eszerint a protokoll szerint rendszerezzük az adatokat egy általunk meghatározott struktúrába, amelyet majd a mobilkészülék oldalán könnyen a felhasználó által érthető formára alakíthatunk.



19. ábra: A JSON logója

4.1.4.1.1 Miért a JSON? (14) (15)

Több választási lehetőségünk volt a leíró nyelvek között, a JSON mellett az Extensible Markup Language (Kiterjeszthető jelölő nyelv, XML) jött komolyan számításba (saját protokollt nem akartunk kidolgozni, hiszen a szabványosak is teljesen a célnak megfelelőek). Alapvetően azért döntöttünk a JSON mellett, mert mikrokontrolleres környezetben könnyen kezelhető, ami fontos szempont, illetve sok library, forráskód érhető el hozzá. (A library-k kódokat tartalmaznak, amelyek az adott nyelvben segítik a programozó munkáját a JSON sztring felépítésében.)

A következőkben összehasonlítjuk a JSON-t az XML-lel:

Mindkét leíró nyelvben közös, hogy tökéletesen alkalmasak adatok leírására úgy, hogy két különböző alkalmazás számára értelmezhetők legyenek, könnyen olvashatók és értelmezhetők az ember és a program számára, és képesek az adatokat valamilyen struktúrába szervezni.

Az adatok leírása szempontjából mindkettő szöveg alapú és elrendezéstől független (azaz csak az adatok és a köztük lévő szintaktikai jelek számítanak a kód értelmezésekor).

Általánosságban az XML fő hátránya a JSON-nel szemben – amely a mi szempontunkból különösen fontos –, hogy adatátviteli célra nem a legmegfelelőbb. Ez abban nyilvánul meg, hogy a JSON-hoz képest eléggé „bőbeszédű”, jelentős benne a redundancia (lásd <valami> </valami> szintaktika). Ennélfogva ugyanazt az

adatmennyiséget csak nagyobb méretben tudjuk lekódolni, ami nem hatékony, adattranszfer szempontjából gazdaságtalan. Ezen kívül a JSON-nel az adatok valamilyen bonyolultabb struktúrába szerveződését egyszerűbben le lehet írni. Az XML további hátránya, hogy az általunk fontos nyelveken (Java, Objective-C) több library elérhető, míg XML-es library-t nagyrészt csak C++-hoz találtunk, bonyolult kódolással. (Az Android alkalmazás nyelve Java, az iOS-é Objective-C; a mikrokontroller nyelve C, amelyben mindkét leíró nyelvhez találhatók forráskódok.) Ezen felül az XML nyelvnek több verziója van, így az ebből adódó kompatibilitási problémákat külön le kellene kezelnünk, amihez megint a library-k kellenek.

A JSON hátránya, hogy csak néhány adattípus áll rendelkezésre, ám a mi projektünkben a JSON adattípusai tökéletesen elégségesek.

4.1.4.1.2 A JSON-ról

A JSON egy adatcserére kifejlesztett leíró nyelv, amellyel tetszőleges struktúrájú adatokat tömöríthetünk egy sztringgé. Az adatok rendszerezése a szabvány jelöléseivel fa struktúrában történik, azaz egy gyökérből ágaznak szét csúcsokká, majd ezekből további csúcsokká stb. Ezt írjuk bele egyetlen sztringbe, amit később bitenként küldünk a telefonnak. A mobil érzékeli, amikor teljes egészében átment egy JSON sztring, és ezután kezdi el az adatok feldolgozását.

A JSON által leírható alap adattípusok: (15)

- szám
- karakterlánc
- boolean (true vagy false érték)
- tömb
- objektum (kulcs-érték párok)
- üres (null)

4.1.4.1.3 Az eddig elkészült tesztprogram és JSON minta

A fejlesztés jelenlegi állapotában megvalósult a kommunikáció az ECU-tól egészen a mobilkészülékig. Az ECU által küldött jelek a LIN hálózaton keresztül eljutnak a mikrokontrollerig, ahol feldolgozásra kerülnek. A mikrokontroller az érkező adatokból előállítja a JSON-kódot, és meghatározott időközönként küldi tovább a

mobiltelefonnak, amely értelmezi, és a felhasználónak kijelzi azt. (A mikrokontrolleren C-ben, az Androidos telefonon Java-ban, az iPhone-on Objective-C-ben írjuk a JSON író/olvasó kódot.) A rendelkezésünkre álló ECU az akkumulátorhoz tartozó érzékelő (Electronic Battery Sensor, EBS), így a lent bemutatott C-s tesztprogramban az ehhez tartozó JSON kódot mutatjuk be:

```
uint8_t conf_string[]="{"
"\"ECU\":\"EBS\","
"\"voltage\":"
"{"
"\"value\":\"%2.3f\","
"\"status\":\"%d\","
"},"
"\"current\":"
"{"
"\"value\":\"%2.3f\","
"\"current_range\":\"%d\","
"\"current_div\":\"%d\","
"\"current_status\":\"%d\","
"},"
"\"temperature\":"
"{"
"\"value\":\"%d\","
"\"status\":\"%d\","
"},"
"\"SOC\":"
"{"
"\"value\":\"%d\","
"\"status\":\"%d\","
"}"
"}";
sprintf(write_string,conf_string,EBS.voltage,EBS.voltage_status,EBS.current,EBS.current_range,EBS.current_div,EBS.current_status,EBS.temp,EBS.temp_status,EBS.SOC,EBS.SOC_status);
USART_Transmit(write_string)
```

```
{
   "ECU": "EBS",
   "voltage"  :
   {
      "value": "12.123",
      "status"        : "1",
   },
   "current"  :
   {
      "value": "123123",
      "current_range" : "1",
      "current_div" : "1",
      "current_status" : "1",
   },
   "temperature"  :
   {
      "value": "25.1",
      "status" : "1",
   },
   "SOC"  :
   {
      "value": "56",
      "status" : "1",
   }

 }
sprintf(write_string,"{""ECU"":""EBS"", %2.3fV %fA %dC° \n Voltage_Status: %d \n Current_Status: %d \n Temp_Status: %d \n SOC: %d \n SOC_Status: %d \n \n",EBS.voltage,EBS.current,EBS.temp,EBS.voltage_status,EBS.current_status,EBS.temp_status,EBS.SOC,EBS.SOC_status);
```

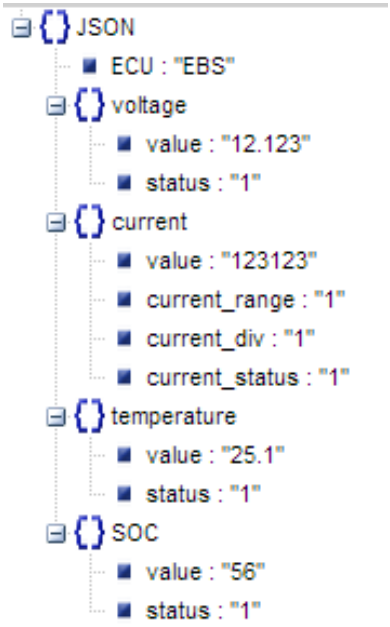
3. kódrészlet: Akkumulátor ECU adatai JSON-ben

Itt jól látszik a JSON fa struktúrája, a csoportokra tagolás és a kulcs-érték párok. Ahogy látható, a JSON kódot az ember is ránézésre tudja értelmezni.

Ebben a példában a struktúrában tárolt adatokat a következőképpen értelmezhetjük:

- négy tulajdonság van: az akkumulátor feszültsége, árama, hőmérséklete és töltöttségi szintje
- a tulajdonságokon belül találhatók a kulcsok és az ezekhez tartozó értékek (pl. a töltöttségi szintnek a "value" kulcshoz tartozó értéke 56 [%]).

A JSON struktúrájának könnyebb megértéséhez készültek grafikusan megjelenítő programok; a fenti program képe az Online JSON Viewerrel:



20. ábra: JSON fa struktúra

4.1.4.2 Az üzenetek típusai

A következőkben bemutatásra kerülnek a konkrét üzenetek, amelyeket JSON nyelven küldünk a telefontól az ECU-hoz és viszont. Ezek az üzenetek már nem egy vagy több specifikus ECU-hoz készültek, hanem univerzálisan alkalmazhatók a CAN és LIN hálózaton forgalmazott jelek továbbítására.

4.1.4.2.1 Login

A login üzenetet az okostelefon alkalmazás küldi a Bluetooth kapcsolat létrejöttekor. Mezői:

- az okostelefon egyedi hardver azonosítója
- az alkalmazás szoftververziója

4.1.4.2.2 Config Info

Ezt az üzenetet válaszként küldi a CC-ECU a Login üzenetre. Benne a CC-ECU konfigurációjáról találhatók információk:

- a CC-ECU egyedi hardver azonosítója
- firmware verzió
- a konfiguráció verziója
- az autó hálózatán elérhető ECU-k

4.1.4.2.3 Config

Ezt az üzenetet a telefon küldi válaszul a Config Info üzenetre. Ez a szűrendő jeleket tartalmazza:

- jel azonosító
- üzenet típus (CAN/LIN)
- üzenet azonosító
- kezdő bit
- befejező bit

4.1.4.2.4 Request for Signals

Kérelem az információk küldésére; a telefonos alkalmazás küldi az ECU-nak, amennyiben valamilyen információra van szüksége. Egyetlen típusú mezője van, amelyből több is szerepelhet egy üzenetben:

- jel azonosító

4.1.4.2.5 Signal Data

Ezt az üzenetet a kérelemre válaszul küldi az ECU a megszerzett információval. Mezői minden egyes jelre vonatkozóan:

- jel azonosító
- a nyers adatok

4.1.4.2.6 Request for Periodic Sending

Kérelem a periodikus üzenetküldésre, amelyet a telefon intéz az ECU-hoz.

- jel azonosító
- időköz (értéke 100-tól 60000 ms; 0: üzenetküldés leállítása)

Ennek hatására az ECU periodikusan ismétlődően Signal Data üzeneteket kezd küldeni.

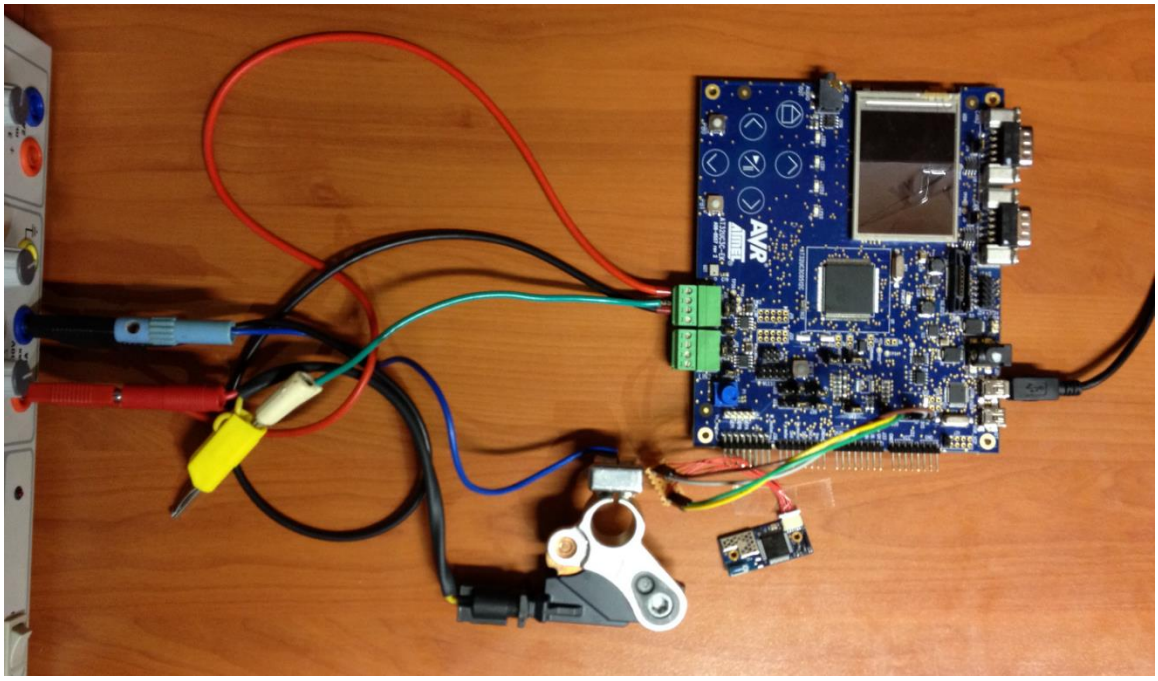
4.2 Fejlesztői környezet

Az Atmel mikrokontrollerekre történő fejlesztés fő eszköze az Atmel AVR Studio. Integrált fejlesztő környezet, mely tartalmaz szerkesztőt, fejlesztés segítő funkciókat (code highlighting, szöveg kiegészítés, egyszerű navigáció a hivatkozott fájlokban), emulátort, debuggert, amivel megfelelő programozó eszköz (pl.: JTAGICE III) segítségével akár regiszter szinten tudjuk figyelni a program futását magán a mikrokontrolleren.

Az AVR Studio támogatja az Assembly, C és C++ nyelven történő fejlesztést, a CC-ECU szoftvere C-ben íródott.

Az AVR Studio fontos eleme az Atmel Software Framework (ASF), ami egyrészt egy kész szoftver sablont, másrészt rengeteg előre megírt könyvtárat nyújt (pl.: LIN, CAN), ezzel megkönnyítve a fejlesztést.

4.3 Prototípus hardver



21. ábra: A rendszer asztali modellje

A projekt első fázisában elkészült a rendszer asztali modellje, amely a végleges rendszer minden lényeges részét tartalmazza. A fejlesztői rendszer részei a következők:

- Bosch EBS - elektronikus akkumulátor szenzor
- Atmel AT32UC3C-EK mikrokontroller fejlesztői készlet
- ConnectBlue OBS421 Bluetooth modul

A rendszerben felhasznált elemek típusa megegyezik a végleges rendszerben történő felhasználásra szántakkal, azzal a különbséggel, hogy így nem volt szükség új nyomtatott áramkör tervezésére. Természetesen a véglegesnek szánt kommunikációs eszköz PCB (Printed Circuit Board) tervezése folyamatban van, de a felépített rendszer ezen formájában már funkciófejlesztésre alkalmas.

4.3.1 Bosch EBS

A rendszerben használt Bosch EBS autóipari vezérlő célja, hogy mérje az akkumulátor feszültségét, áramát, hőmérsékletét, illetve ezekből kiszámítja az akkumulátor töltöttségi szintjét (SoC - State of Charge), ezeket az adatokat pedig LIN hálózaton továbbítja.

4.3.1.1 Bosch EBS LIN keretei (16)

Az EBS által küldött LIN keretek az alábbi információkat tartalmazzák:

- Akkumulátor feszültség
- Akkumulátor áram tartománya
- Akkumulátor áram
- Akkumulátor hőmérséklet
- Áram státusz
- Feszültség státusz
- Hőmérséklet státusz
- Töltöttségi szint (SoC)
- SoC státusz

4.3.2 ConnectBlue OBS421 Bluetooth modul

A Bluetooth modul SPI buszon kapcsolódik a mikrokontrollerhez, a kommunikáció a modul gyártója által előírt protokoll szerint történik. A Bluetooth kommunikációval és a modul működésével részletesen a Bluetooth fejezet foglalkozik.

4.3.3 Atmel AT32UC3C-EK (17)

A fejlesztői board központi eleme AT32UC3C0512C mikrokontroller, viszont szemben a nyers chippel, a fejlesztői board tartalmazza majdnem az összes mikrokontroller által támogatott fizikai interfészt és port kivezetést, többek közt 2 CAN és 2 LIN interfészt, valamint a Bluetooth modulhoz szükséges SPI busz portjainak kivezetését.

A jelenlegi összeállítás alkalmas a feszültségérték kiolvasására, az érték feldolgozására majd Bluetooth-on keresztüli küldésére a projekt folyamán kidolgozott protokollnak megfelelően.

4.4 Prototípus szoftverek

4.4.1 A mikrokontroller szoftvere

A mikrokontroller szoftvere jelenleg az alapfeladatokat látja el: adatokat fogad az akkumulátor szenzortól LIN hálózaton, az így érkezett adatokat pedig Bluetooth-on továbbítja a már említett JSON formátumban. A LIN kezelése az Atmel Software Framework által szolgáltatott könyvtár segítségével történik. A Bluetooth modul és a

mikrokontroller között az adatcsere USART-on történik, a modul gyártója által előírt protokoll szerint. Az alvó módok és a kétirányú kommunikáció megvalósítása a későbbi tervek közt szerepel.

4.4.2 Az okostelefonos szoftver (Android)

A CC-ECU-ból érkező adatok fogadására és feldolgozására szolgáló tesztprogramnak a következő feladatokat kell megvalósítania:

- csatlakozás a CC-ECU-hoz Bluetooth-on keresztül
- a Bluetooth-on érkező adatok fogadása
- a beérkezett adatok feldolgozása
- a feldolgozott adatok megjelenítése

Az Android Bluetooth eszközei az android.bluetooth csomagon keresztül érhetőek el, miután a felhasználó jóváhagyta, hogy az alkalmazás hozzáférjen a Bluetooth rádióhoz (a jogosultság igényt az AndroidManifest.xml-ben kell jelezni). Az Android program alap építőeleme az Activity (egyben ez az ősosztály), ami legegyszerűbben a program egy képernyőjeként jellemezhető, viszont az Activity eseményei szorosan összefüggenek a program életciklusával: az onCreate függvény akkor hívódik meg, amikor az Activity elindul, így a kapcsolat létrehozásáért felelős rész ide kerül.

A Bluetooth rádió bekapcsolása után a mBluetoothAdapter.getBondedDevices(); függvényhívással lekérdezhetjük a párosított eszközök listáját, ellenkező esetben egy külön felületen meg kell jeleníteni a közelben lévő eszközök listáját, és párosítani az okostelefont a CC-ECU-val.

A megfelelő inicializálás után az Andorid létrehoz egy socketet, amin keresztül fogadhatjuk a bejövő adatokat a getInputStream() függvény segítségével. Mivel az Android Bluetooth kapcsolata alapértelmezetten soros portként viselkedik, a következő lépés az adatfolyamban az üzenethatárok megállapítása. Mivel az üzenetek a Java által támogatott JSON formátumban érkeznek, az adatfolyamból kinyert üzenetek visszaalakítása könnyen megoldható a JSONObject segítségével.

Az Android által megkövetelt MVC modellnek köszönhetően ez után a megfelelő változóknak értéket adva frissülnek a grafikus felületen megjelenített adatok.

5 Összefoglalás

A dolgozatban a Bosch-BME Connected Car hallgatói projekt eddig elért eredményeit ismertettük, ide értve a használt technológiák leírását (Bluetooth, CAN, LIN, Android, iOS), a felhasznált eszközök specifikációit (fejlesztői környezetek, alapértelmezett protokollok), illetve az általunk felépített rendszer architektúráját és szoftvereit.

5.1 Eddigi eredményeink

A fejlesztés jelenlegi fázisában a kommunikáció működőképesen összeállt a CC-ECU és az okostelefon között. Az autó LIN hálózatát a „deszkamodellünkben” egy széria Bosch akkumulátorérzékelő (EBS) reprezentálja, amely LIN technológiával van összekötve a mikrokontrolleres egységgel. Maga az egység egy 32 bites Atmel AT32UC3C2512C típusú mikrokontrollerből és egy ConnectBlue OBS421 típusú Bluetooth modulból áll. A JSON üzenetekkel történő kommunikációt egy, a Play Store-ról ingyenesen letölthető Androidos Bluetooth SPP programmal sikerült megvalósítani (egy egyszerű üzenetet küldtünk és megjelenítettük a telefon képernyőjén), valamint elkészültek az Android és iOS szoftverek kommunikációs alapjai.

5.2 Közeli tervek

A közeljövőben a következő fejlesztési feladatokat szeretnénk elvégezni:

- CAN/LIN hálózatra való univerzális csatlakoztathatóság (az ECU-któl származó adatok feldolgozása),
- egységes protokoll, üzenet-felépítés (JSON) kialakítása a mikrokontroller és a Bluetooth modul, illetve a Bluetooth modul és az okostelefon között,
- az Androidos szoftver teljes kifejlesztése ergonomikus felhasználói felülettel,
- a szoftver átültetése iOS-re,
- a CC-ECU végleges tervezése és megépítése, IP65-ös védelemmel, kompakt méretben.

5.3 Távlati tervek

Az alapkiépítésen túl a rendszer rengeteg új lehetőséget rejt, főleg, ha kihasználjuk az okostelefon és az internet közötti kapcsolat kínálta lehetőségeket, illetve a telefontól a CC-ECU, ezáltal az autó felé történő kommunikáció lehetőségeit. Ezekből néhány konkrét terv:

- Az okostelefonok hangfelismerő és hangvezérlő képességét felhasználva megoldható, hogy a vezető beszéddel vezérelje pl. a klímát, napfénytetőt stb.
- Ha túl alacsony akkumulátor-töltöttséget, vagy egyéb hibát érzékel a rendszer, akkor a telefon képes térképen megmutatni a legközelebbi szervizeket, vagy megjeleníteni az autómentő telefonszámát.
- Ha az autót többen is használják, lehetséges sofőr-profilokat definiálni és a telefon memóriájában tárolni, így az erre alkalmas autókban automatikusan beállítható az előre elmentett konfigurációk alapján az aktuális sofőrnek megfelelő ülésmagasság, kormánymagasság, támla-dőlésszög, stb.
- Az autó hálózatára való csatlakozás miatt a hálózaton futó adatokból statisztikákat lehet készíteni; ezeket lehet tárolni a telefonon, vagy pl. a telefon által a Bosch szerverére elküldeni.
- Könnyen készíthető a járműről diagnosztika, mivel közvetlenül hozzáférhetünk minden, az autó hálózatán futó adathoz.
- Egyszerűvé válik egy cég autóinak flottamenedzsmentje: az autókhoz konfigurált céges telefonok folyamatosan tudják küldeni a jelet a jármű hollétéről (a GPS vevő által), az üzemanyagtartály töltöttségéről, stb.
- A CC-ECU az első beszereléskor a telefon által könnyen konfigurálható a konkrét autóra úgy, hogy a telefon lekéri az internetről az autó ECU-inak adatait.
- Használaton kívül a rendszer energiatakarékossági célból alvó módban van, a telefon közeledésére felépül a kapcsolat és megtörténnek a szükséges frissítések, így mire a járműbe beszállnak, a rendszer máris használatra kész.

6 Irodalomjegyzék

1. **Ferenc László.** Mobilport. *Mi tud az új Bluetooth?* [Online] http://www.mobilport.hu/hirek/20100504/mit_tud_az_uj_bluetooth/.

2. **SIG, Bluetooth.** Bluetooth. *Low Energy.* [Online] http://www.bluetooth.com/Pages/Low-Energy.aspx.

3. **Balogh András.** Kommunikáció Bluetooth Low Energy alapú piconetek között.

4. **Wikipedia.** Bluetooth Low Energy. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Bluetooth_low_energy.

5. **Oláh Zoltán.** MacMagazin. *Így működik az új Bluetooth.* [Online] http://www.ijoe.hu/macmagazin/howbluetoothworks/howbluetoothworks.php.

6. **Najmul, Islam.** *Bluetooth: Overview of architecture, PHY and MAC.* Tamperei Műszaki Egyetem : ismeretlen szerző, 2007.

7. **BME Közlekedés- és Járműirányitási Tanszék.** *Irányító és kommunikációs rendszerek III. előadás.*

8. **Johnson Consulting.** Bluetooth - An Overview. *How networks are formed and controlled.* [Online] http://www.swedetrack.com/images/bluet10.htm.

9. **AB, connectBlue.** *Bluetooth Serial Port Adapter Security.* 2011. 10. 21..

10. **BME Közlekedés- és Járműirányitási Tanszék.** *Járműfedélzeti rendszerek II. - Oktatási segédlet.* 2013.

11. **Google.** The AndroidManifest.xml File. *Android API Guides.* [Online] http://developer.android.com/guide/topics/manifest/manifest-intro.html.

12. **Wikipedia.** IP Code. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/IP_Code.

13. **AB, connectBlue.** *Serial Port Adapter - Extended Data Mode.*

14. **JSON.org.** Introducing JSON. *JSON: The Fat-Free Alternative to XML.* [Online] http://www.json.org/xml.html.

15. **Wikipedia.** JSON. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Json.

16. **Robert Bosch GmbH.** *EBS - Electronic Battery Sensor - adatlap.*

17. **Atmel.** *AT32UC3C-EK User Guide.*

18. Android (operating system). *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Android_(operating_system).

7 Ábrajegyzék

8 Táblázatjegyzék