Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Csonka Bálint

AUTÓELEKTRONIKAI ALKATRÉSZEK TESZTELÉSE SZIMULÁCIÓS KÖRNYEZETBEN

Szakdolgozat

KONZULENSEK

Dr. Sujbert László,
docens

Hasprai László,

Audi Hungaria Kft.

BUDAPEST, 2016

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Csonka Bálint**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulensek neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 05. 24.

...……
Csonka Bálint

Összefoglaló

A mai modern autógyártásban, amikor az alkatrészeket külső beszállító cégek szolgáltatják és a készterméket előállító vállalatnál szinte csak az egyes komponensek összeszerelése történik a minőségbiztosítás a gyártás szerves részévé válik. A növekvő felhasználói elvárások maximális kielégítésének érdekében egyre több és egyre precízebb ellenőrzési folyamatra van szükség. A professzionális tesztberendezések beszerzése rendkívül költséges és vannak feladatok, amelyek nem igénylik meg az általuk nyújtott komplexitást, ezért a vállaltok specifikált mérőállomások létrehozására törekszenek.

Szakdolgozatom témájaként egy ilyen mérőállomás át- illetve kialakítását választottam. A feladat megvalósítása során egy olyan tesztkörnyezetet hozok létre, amellyel analízisek végezhetők az Audi TT típusú gépkocsi harmadik generációjának klímavezérlőjén és annak egyik kiszolgáló áramkörén.

A bevezető fejezetekben a modern autók elektromos rendszerének összetevőivel foglalkozom a szenzoroktól és beavatkozó szervektől kezdve, a vezérlőkön át egészen a kommunikációs buszokig. Ezután a téma megértéséhez szükséges információkat közlök az autódiagnosztikát érintően. Ismertetem a klímavezérlő működésével kapcsolatban elvégzett teszteket. Bemutatom a szimulációs környezet tervezésének és megvalósításának lépéseit, majd prezentálom a szimulációs környezethez készített felhasználói felületet. Értekezésemet a jövőbeli fejlődés lehetséges irányainak bemutatásával illetve a kivitelezés során szerzett tapasztalatok összegzésével és továbbfejlesztési lehetőségek ajánlásával zárom.

Abstract

In today's car production quality management is becoming an essential part of manufacturing. It is owing to the fact that spare parts are provided by suppliers and producer undertakings mainly focus on the assembly of single components. In order to meet the growing requirements of the clients more and more precise control procedures are needed. The acquisition of professional testing devices is extremely sumptuous and there are tasks, which do not require the complexity provided by them, so the companies are striving to establish specified measuring stations.

I have chosen the construction and reconstruction of such a station as the subject of my thesis. During the realisation of the task I have established a testing environment with the help of which analyses can be carried out on the climate controller of the third generation Audi TT cars.

The introductory chapters are devoted to components of the electric system of modern cars, from sensors and actuators, through controllers to communication networks. Then I give information referring to car diagnostics, in order to make the topic more understandable. I describe the tests carried out in connection with the functioning of the climate controller. I present the steps of planning and realisation of the simulation environment, and the API designed to the simulation environment. After that I give an outline of the possible directions of future development. I end my work with the summary of the experiences gained during implementation and with the recommendations for development.

Előszó

Szakdolgozatom témájával az Audi Hungaria Motor Kft.-nél töltött gyakornokságom alatt találkoztam. Az előszót ezért általános információkkal kezdem a cégről. Ezután szólók pár szót a minőségbiztosítás vállalat munkájában betöltött szerepéről, majd bemutatom a minőségellenőrzési folyamatot és annak egyik fontos állomását, amelyhez a megvalósítandó feladatom is kapcsolódik.

Audi Hungaria Motor Kft.

A győri illetőségű cég az Audi AG és így a Volkswagen konszern leányvállalata. A gyár 1993-ban nyitotta meg kapuit. Az akkor még kizárólag motorgyártással foglalkozó AHM 1998-tól járműgyárként is funkcionál. A TT Coupe és Roadster, az A3 Limousine és az A3 Cabriolet modellek készülnek itt. Később a vállalat szerszámgyárral is bővült, ami a karosszériaelemeket megmunkáló présgépek számára készít alkatrészeket. A 4 millió négyzetméteren elterülő gyáróriás több mint 12 ezer embernek ad munkát Győrben és a város vonzáskörzetében. Itt működik a legnagyobb motorgyár a világon, évi 2 millió erőforrás hagyja el a gyártósorokat. A győri motorok a konszern minden személygépkocsi márkájában megtalálhatóak a Seat-tól a Bentley-ig.

Minőségbiztosítás az Audi Hungariánál

Modern, globalizált világunkban, egy brand értékét valójában a minőségbiztosítás eredményessége határozza meg. Az autógyárak manapság óriás technológiai integrátorok, melyek a saját terveik szerint megrendelt, külön erre a célra szakosodott cégek által legyártott alkatrészek precíz összeszerelését végzik. Lényeges minőségbeli előnyt a tervezés és a minőségbiztosítás területén lehet szerezni. Nem véletlen, hogy a vállalatnál kiemelt hangsúlyt fektetnek ezekre a területre.

A beszállítók a saját termékeiken elvégzik az előírt vizsgálatokat, melyet a gyártás végén egy „end of line” teszt zár. Az alkatrészt önmagában vizsgálva soha nem lehetünk 100% -osan biztosak abban, hogy az később egy autóba építve, egy elektromos hálózat részeként is hibátlanul fog működik. Több esetben csak a gépjárműben derülnek ki azok a hibák, melyeket a vezérlőket ért egyidejű hatások összessége vált ki. Amint ilyen hálózati anomáliára fény derül, konzerválni kell azt és ezzel ki kell egészíteni a

meglévő vizsgálatokat. Többek között ilyen jellegű a vizsgálatoknál alkalmazható a szakdolgozat keretein belül elkészülő tesztállomás, amint azt a későbbiekben részletezni fogom. A gyártás során folyamatos a mechanikus, elektronikus és optikai ellenőrzés egészen a kész gépjárműig. Ha a folyamat közben bármilyen oknál fogva egy alkatrész elégtelennek bizonyul, egy analízis mérnökökből álló teamhez kerül. Itt a megfelelő vizsgálatok elvégzése után választ adnak a hiba okára és megállapítják annak okozóját.

A mérnökök munkája a minőségbiztosítási részleg különböző mérőszobáiban zajlik. Az itt található high-tech mérőeszközök teszik lehetővé a hatékony és eredményes analízist. Külön mérőszoba működik a mechanikus, az elektromos és elektronikus, az illesztési, lakkozási, az optikai, valamint a gépkocsi hangzásvilágát érintő vizsgálatokra. A cél nem kevesebb, mint a kompromisszumok nélküli minőség és vevői elégedettség.

Az E-labor és a vizsgálópad

Az Elektromos és Elektronikus Eszközök Laboratóriumban (későbbiekben E-Labor) a jármű villamos hálózatát alkotó összes alkatrész analízise elvégezhető.

Az analízis a vizsgált komponens egy speciális mérőpadhoz való csatlakoztatásával indul. A vizsgálópadon az autók teljes elektromos hálózata ki van építve: a legtöbb szenzor, aktor és vezérlő. Ami pedig nincs, annak jelei software-esen vagy hardware-esen szimulálva vannak. Így tehát minden alkatrész úgy viselkedik, mintha egy gépkocsiban lenne. Az autóval szemben egy nagy előnye ennek a rendszernek, hogy minden alkatrész könnyen hozzáférhető, cserélhető és tesztelhető. Ezen kívül lehetőség van az autóban megtalálható kommunikációs buszok (CAN, LIN, MOST) megfigyelésére a kommunikációs interface-ek segítségével. A vizsgálat során a mérőpad megfelelő elemét kicserélik a hibás darabbal, és kielemzik az így módosított rendszer működését. Automatizált funkcióteszteket futtatnak, és online monitorozzák a vezérlők hibatárolóit.

Ha a hiba ilyen körülmények között nem jelentkezik, az alkatrészt hő sokknak vetik alá, vagy rázópadra teszik. Az anomália így már nagy valószínűséggel újra detektálhatóvá válik. Ha még az analízisnek ebben a stádiumában is kérdések vetődnek fel, az alkatrész a Félvezető Laboratóriumba kerül, ahol a belső struktúrák vizsgálatára nyílik lehetőség. Az itt található optikai és elektronmikroszkópokkal, metszetkészítő

berendezésekkel, röntgensugárral és ultrahanggal működő készülékekkel a panelon lévő alkatrészek, forrasztások és vezetékezések ellenőrizhetők.

A hiba azonosítását követően az analízis team javaslatot tesz a probléma megszüntetésére. Ez akár a gyártástechnológiát módosító javaslattétel is lehet a beszállító irányába. Abban az esetben, hogyha feltételezhető, hogy a hiba több alkatrészt is érinthet, elindul a kívánt akció a kérdéses elemek vizsgálatára. Ez jelentheti a beszerelésre váró alkatrészek válogatását és tesztelését, hogy az újonnan épülő autókba már csak jó elem kerülhessen.

1 Bevezető

Dolgozatom bevezetőjében értelmezem a megoldandó feladatot, rávilágítok a téma létjogosultságára és részletezem a specifikációt. Ezen kívül ismertetem a rendszer terveket, amik alapján a téma kidolgozásra kerül.

1.1 A feladat megfogalmazása

A szakdolgozat feladatkiírása pontosan definiálja az elvégzendő munkát. Egy olyan gépjárműtől és helytől független, kompakt tesztkörnyezet létrehozása a cél, mely szimulálja egy vizsgált vezérlő, szakdolgozatom esetén a 3. generációs Audi TT modell klímavezérlőjének működéséhez szükséges hardware és software környezeti paramétereket.

A téma létjogosultságát az adja, hogy vannak helyzetek, amelyekben a korábban bemutatott vizsgálópad használata szükségtelen, felesleges idő és energia befektetést jelent. Léteznek, olyan célzott tesztek, melyek egy kevésbé összetett rendszer segítségével is futtathatók. Egy ilyen vizsgálóállomás tehermentesíti a mérőpadot, meggyorsítja az alkatrész válogatási folyamatokat és mobil volta miatt alkalmazható a gyár bármely területén vagy akár a logisztikai központban is. Ezzel megkönnyíti az analízis mérnökök és a válogatást végző dolgozók munkáját.

1.2 Megoldandó problémakör szűkítése, a folyamat felvázolása

Az előző pontban meghatároztam a kitűzött célt, a kivitelezés elkezdéséhez azonban szükség van a problémakör szűkítésére és az egyes elvégzendő lépések meghatározására.

Amint az előzőekből kiderült nem csak a gépjárműtől, de a vizsgálópadtól is függetleníteni kell a vizsgált vezérlőt. Ha egy hálózatban dolgozó beágyazott rendszert kiemelünk rendeltetési helyéről, akkor önmagában működésképtelenné válik. Azért, hogy mégis a hálózattól függetlenül méréseket lehessen végezni rajta, egy speciális eszközzel szimulálni kell azt a környezetet, amiben a vezérlő eredetileg üzemelt. Az első részfeladat tehát a klímavezérlő működési feltételeinek megteremtése, olyan

mértékig, hogy meghatározott vizsgálatok, például a hibatárak kiolvasása elvégezhető legyen.

A második részfeladat a klímavezérlő egyik kijelző és szabályozó gombjával kapcsolatos. Szükség van egy olyan tesztkörnyezetre, amivel a hálózattól és a klímavezérlőtől függetlenül megvalósítható a hőmérséklet állító klíma vezérlő elem funkcióvizsgálata.

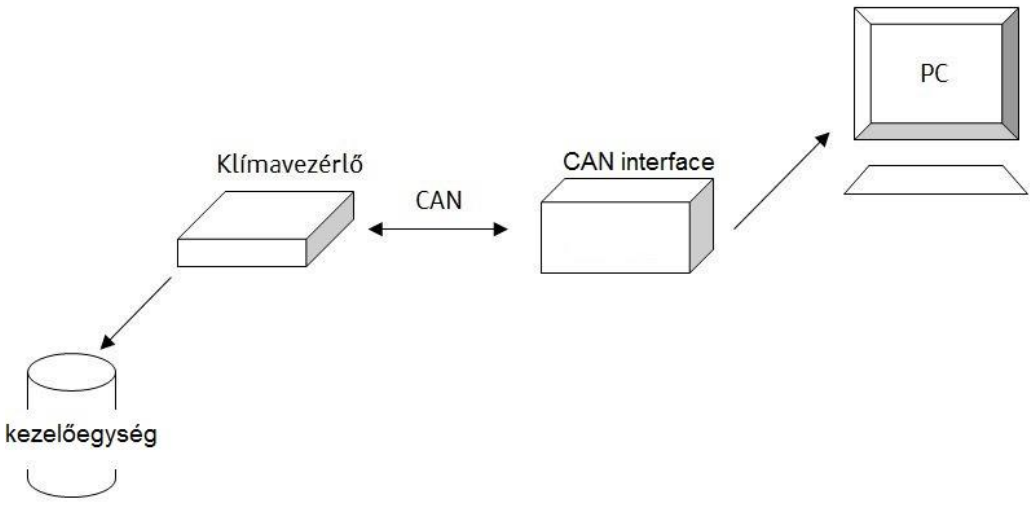
A munkát irodalomkutatással kezdem. Ez esetemben a vizsgált gépjármű elektromos hálózatának megismerését és a felhasználható eszközök feltérképezését jelenti. A dolgozat első felében az autó elektromos alkatrészeiről és a közöttük kapcsolatot teremtő autóipari kommunikációs protokollokról lesz szó. Ezt követően a hálózat szimulációját lehetővé tévő hardware és software eszközökről írok.

A második rész az elvégzendő mérésekről és tesztekről szól. Majd ezután a tervezési és kivitelezési folyamat dokumentálása következik.

Befejezésül értékelem a munkám eredményeit, összegzem a megoldás során szerzett ismereteket és megfogalmazom a lehetséges továbbfejlesztési irányokat.

1.3 Rendszerterv

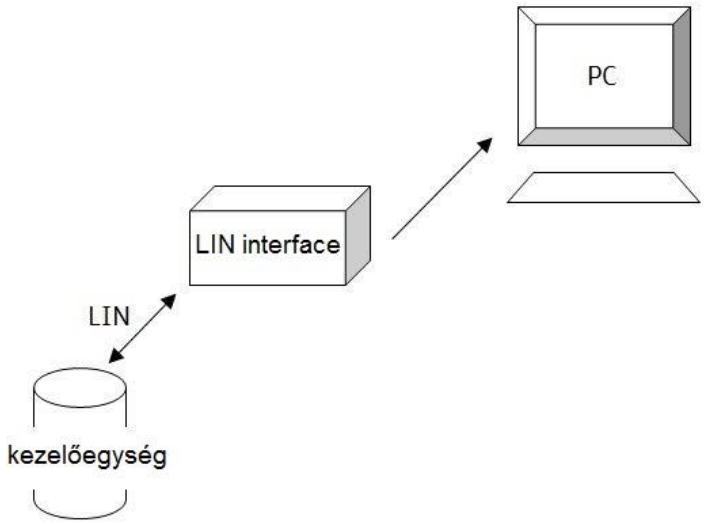
A specifikáció részletezését követően két, jól körülhatárolt feladat megoldását tűztem ki célul. Az alábbi ábrákon a két részegység rendszertervét ismertetem.



1. ábra: Rendszerterv az első részfeladathoz

A számítógép jelen esetben a szimuláció software-es részét végző CANoe programot szimbolizálja. A PC és a szabad perifériákkal maradt klímavezérlő között a

Vector cég által gyártott interface teremt kapcsolatot. Az összeköttetés a számítógép és az interface között USB porton keresztül, az interface és a vezérlő között egy általam elkészítendő CAN kábellel valósul meg. A mérési elrendezés részét képezi a hőmérséklet állító klíma vezérlő elem is, aminek a kijelzője a klímavezérlő működésének egyik indikátora lesz.



2. ábra: Rendszerterv a második részfeladathoz

A feladat második részegységében a kezelőegységé a főszerep. A tekerőgomb egy végtelenített tekerőgombból (Input) és egy folyadékkristályos kijelzőből (output) áll. Az interface itt már nemcsak összeköttetésként szolgál, hanem a tekerőgomb meghajtását végző LIN hálózat Mastereként is. Hiszen eddig ezt a feladatot a klímavezérlő töltötte be, amit eliminántunk ebből a mérési összeállításból. A számítógépen egy általam kivitelezendő API (Application Program Interface) fut a CANoe program keretein belül, amivel a kezelőegység funkciói kényelmesen és áttekinthető formában vizsgálhatók.

A tápellátást mindkét részfeladat esetén nekem kell biztosítanom egy külső feszültségforrásból.

2 Gépjármű elektronikai ismeretek

Ebben a fejezetben ismertetem azokat az információkat, amelyek a modern autó (jelen esetben az Audi TT) elektromos hálózatának megértéséhez és a dolgozat által kitűzött cél eléréséhez szükségesek. Az alfejezetek az egyes komponenseken haladnak végig az egyszerűbb elektromos alkatrészektől egészen a teljes rendszerig.

A külvilág változásait a jármű vezérlői számára feldolgozható jelekké alakító elektromos berendezések az érzékelők. Ezek a korábban analóg jeleket szolgáltató szenzorok manapság digitális jeleket küldenek. A klímavezérlő és érzékelői esetén például LIN buszon történik az adattovábbítás. Az információ áramlás mindkét irányban működik, hisz a klíma állító gomb egyúttal a beállított hőmérséklet kijelzésére is szolgál. A tekerőgomb a kijelzendő információt szintén LIN buszon kapja a Master-től, ami a klímavezérlő.

A következőkben szó lesz általánosságban a vezérlőkről és elhelyezem a klímarendszert az autó elektromos hálózatában.

2.1 Az autó, mint elektromos hálózat (Audi TT3) és a vezérlők

Az érzékelőket és beavatkozókat a legkülönfélébb feladatokkal ellátott beágyazott rendszerek kötik össze. Ezek a vezérlők kiegészítő áramkörökből és a beérkező adatokat elemző és a szükséges beavatkozásokat elvégző mikrokontrollerekből állnak. Egy járműgyárban az ilyen alkatrészek hardware-esen készen, software-esen előprogramozva, a beszállító cégektől érkeznek, módosítást a legtöbb esetben már csak a rajtuk futó programok paraméterezésén hajtanak végre. A software a beszállító vagy a fejlesztés saját szellemi terméke, nem publikus, ezért ezeket a modulokat black boxként, működésének részleteiben ismeretlen elemnek tekinthetjük, csak speciális gerjesztések hatására a vezérlő kivezetésein megjelenő jelekből lehet következtetni a programok felépítésre. Ezek a speciális gerjesztések például a vezérlő elektromos környezetének szimulációját jelentik, ami a klímavezérlő esetében szakdolgozatom törzsét képezi. Erről a környezetről, a kontrollerhez kapcsolódó hálózatokról a dolgozat későbbi fejezeteiben lesz szó. Elöljáróban viszont fontos megemlíteni, hogy a vezérlő és szenzorjai illetve beavatkozói között a legtöbb esetben nem egyszerű vezetékek, hanem

autóipari buszok teremtenek kapcsolatot. Ez a klímavezérlő esetén két LIN buszt jelent. A vezérlő a hálózatok Mastere, az utasítások kiosztója és az adatok igénylője.

Az egyik LIN hálózaton öt tekerőgombot találunk, melyekkel a felhasználó a klíma hőmérsékletét, a levegő befújásának erejét és irányát illetve az ülésfűtés intenzitását állíthatja be. Az értékeket a kapcsoló forgatásával vagy megnyomásával lehet módosítani. A másik LIN hálózat tartalmazza a klímarendszer működéséhez szükséges érzékelőket, melyeknek rövid bemutatása következik most.

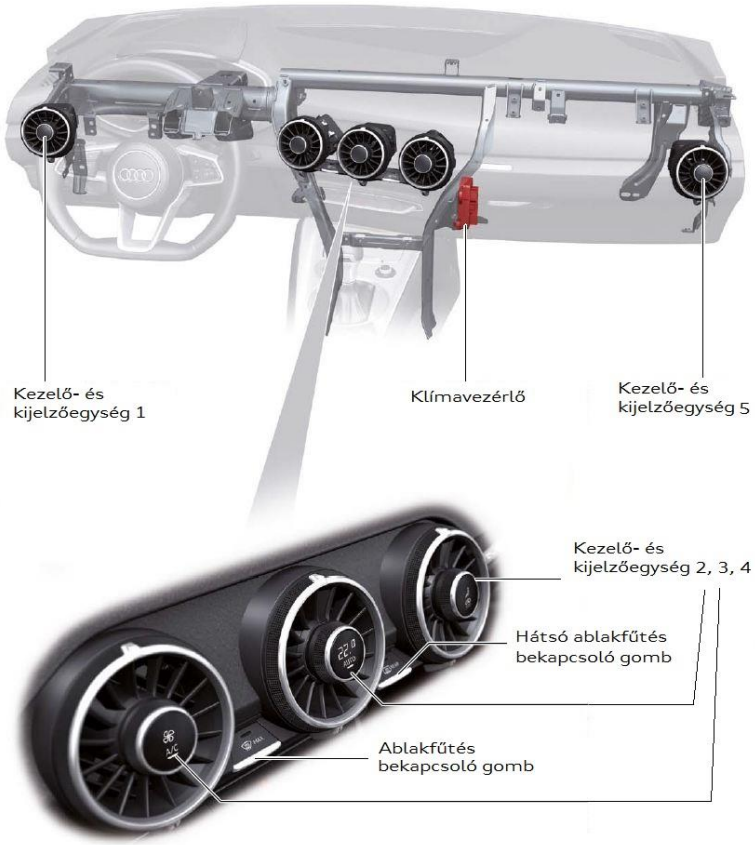
A klímavezérlő szenzorjai a külső és belső levegő összetételét és hőmérsékletét mérik. Ezen kívül fontos, hogy a vezérlő pontos információval rendelkezzen aktorai állapotáról, ezeket a visszajelzéseket is érzékelők küldik.

A klímarendszerek működtetéséhez a kontrollernek ismernie kell a rendszerben lévő nyomásértékeket, amelyekről egy nyomásszenzor biztosít jeleket.

Alacsony külső hőmérséklet esetén a szélvédő bepárásodhat. Ezt megelőzendő a gépjárműveket páratartalom érzékelővel illetve külső és belső hőmérsékletet mérő szenzorral látják el, mivel az, hogy a levegő milyen mennyiségű vízgőzt képes megkötni jelentősen függ a léghőmérséklettől. Ezeknek az adatoknak a feldolgozása is a klímavezérlő feladata.

A belső tér megfelelő minőségű levegőjének biztosítása is a klímavezérlőt terheli. A kívülről beáramló levegőről a légminőség szenzor által adott jelek segítségével dönti el, hogy megfelel-e az elvárásoknak.

A klímavezérlőhöz kapcsolódó aktorok különböző motorok, amelyek a klímafolyadék nyomásának beállítását, a levegő befújását és a különböző terelőlemezek állítását végzik. Az állítómotorok pozíciójáról potenciométerrel megvalósított érzékelő küld visszajelzést a vezérlő számára.



3. ábra: Klímarendszer felépítése az Audi TT-ben

A vezérlőket, az érzékelők hálózatától eltérően, gyorsabb, robosztusabb viszont magasabb költségekkel rendelkező CAN buszok kötik össze egymással. Azért szükséges a többes szám használata, mert egy járműben a különböző funkcionalitások elkülönítésének érdekében több azonos szintű hálózat is dolgozik egymással párhuzamosan. Az Audi TT-ben például öt CAN hálózatot találunk. Ezek egy gateway-be futnak össze, így teljesen egybefüggő a rendszer, ami elengedhetetlen a hatékony működés és a diagnosztika szempontjából. A klímavezérlő a CAN – Komforton kapcsolódik a többi irányító vagy szabályozó elektronikához. Ez a hálózat, ahogy a nevéből is lehet következtetni rá, az autó kényelmi berendezéseit felügyeli. Ezen kívül van még külön a hajtásért, az infokommunikációs és multimédia alkalmazásokért és a diagnosztikáért felelős buszrendszer.

A gépjármű elektromos architektúrája viszont, ahogy azt már korábban is említettem nem tekinthető különálló hálózatok sokaságának. A gateway összekötő szerepén kívül az egységet az is garantálja, hogy az autónak saját „öntudata” van.

Az egyes vezérlők paraméterezhetőek, személyre- illetve autótípusra szabhatóak. Ez feltétlenül szükséges, hiszen a vezérlők flexibilitását és könnyű

konfigurálhatóságát kihasználva ugyanazt a hardware-t más programmal különböző modellekbe építhetjük. Például a TT Roadster és Coupe számos vezérlőjének csak software-ében van különbség. Miután az alkatrészek parametrizálása megtörtént, az egyes komponenseket software-esen hozzárendelik a járműhöz, amiben dolgoznak. Az alkatrészeknek ezt a tulajdonságát komponensvédelemnek nevezzük. A megoldás a gépjármű ellopását is nehezebbé illetve értelmetlenné teszi, hiszen az alkatrészek más autóban nem használhatók fel újra.

2.2 Az autóban előforduló buszrendszerek típusai és a különböző kommunikációs protokollok

Az előző fejezetekben arról volt szó, hogy az autó milyen eszközökkel tudja feldolgozni és értékelni a külvilág változásait és a kapott eredmények alapján milyen alkatrészek segítségével avatkozik közbe. A fentiekben megismert komponensek hatékony munkájához elengedhetetlen a köztük lévő gyors és pontos kommunikáció minél egyszerűbb vezetékezéssel. Ez csak a rendszer alkotóinak hálózatba kapcsolásával érhető el. Az ilyen hálózatot busznak nevezzük. A vezérlők, szenzorok és aktorok közötti információáramlást a TT 3. generációs modelljében LIN, CAN és MOST buszok végzik. A következőben ezeket ismerjük meg részletesebben.

2.2.1 A fejlődési folyamat

Az autókban használt elektromos szabályozási rendszereken a 70-es 80-as évektől kezdődően figyelhető meg jelentős fejlődés. Ekkor még a központosított szabályozás (centralized control system) volt a jellemző. Kevés számú vezérlő vagy egyszerűbb áramkör végezte a szükséges beavatkozásokat. Ahogy azonban az elektronika és ezzel együtt a chipek és az azokon megvalósított beágyazott rendszerek fejlődtek és olcsóbbá váltak, elterjedt az elosztott hálózat (distributed control system). A gépjárművekben egyre több vezérlő jelent meg egyre inkább specifikált feladatkörrel. A korábbi pont-pont vagy csillag összeköttetések helyett pedig megjelentek a buszok.

Térhódításukat az alapozta meg, hogy szinte minden szempontból előnyösebbek a pont-pont vagy csillag kialakítású rendszereknél. Mivel vezetékenként több funkciót látnak el, kevesebb és vékonyabb huzalozás is elegendő, amivel helyet és tömeget lehet megtakarítani. Ebből következik a költséghatékonyság, kevesebb vezeték, kevesebb súly illetve a szerelési idő is jelentősen lecsökken. A szenzorok jelei az összes

vezérlőhöz eljuthatnak, így az érzékelők száma redukálható. A vezérlők erőforrásainak kihasználtsága sokkal jobb, hiszen a megosztott funkciókon keresztül az egyébként holt időben lévő vezérlő is kap feladatot. A buszok alkalmazásával minőségi javulás érhető el, kevesebb kontaktus kevesebb hibalehetőséget hordoz. A teljes rendszer központilag diagnosztizálható, ezért a hibakeresés sokkal hatékonyabb. Egyszerűbb új software-ek és frissítések telepítése és az elektromágneses zavarok elleni védelem is jobb. Optikai szálak használata esetén utóbbi teljesen kiküszöbölhető. Megállapíthatjuk tehát, hogy az ilyen típusú hálózatokat robosztusság, flexibilitás és bővíthetőség jellemzi.

Busz esetén a vezérlők egymáshoz képesti kapcsolata egy számítógépes hálózatéhoz hasonlít. Ennek köszönhető, hogy a kommunikáció felépítése ugyanazon a logikai irány mentén történik. Az elv, ami a struktúrát meghatározza az ISO (International Organization of Standardization – Nemzetközi Szabványügyi Hivatal) által kidolgozott OSI (Open System Interconnection) modell, ami a kommunikációt hét rétegre oszthatja. Ezek a szintek az alkalmazási, a megjelenítési, a szállítási, a hálózati az adatkapcsolati és a fizikai réteg. A szintek közötti kommunikáció az adat megfelelő csomagolásával érhető el, ami kereteken (frame) keresztül valósul meg. Egy adott szint mindig csak a közvetlen alatta és felette lévő réteggel teremthet kapcsolatot. Az adatáramlás a rétegek között vertikálisan folyik. A vezérlők közti közvetlen kapcsolatért pedig a legalsó, fizikai réteg felel. Az autóipari hálózatoknál általában elég csak 3-4 réteget megvalósítani. A különböző szabványok filozófiája ebben eltérhet egymástól, ezt a későbbiekben ismertetni fogom.

Most következzen néhány alapvető információ az Audi TT-t érintő kommunikációs csatornákról.

2.2.2 LIN

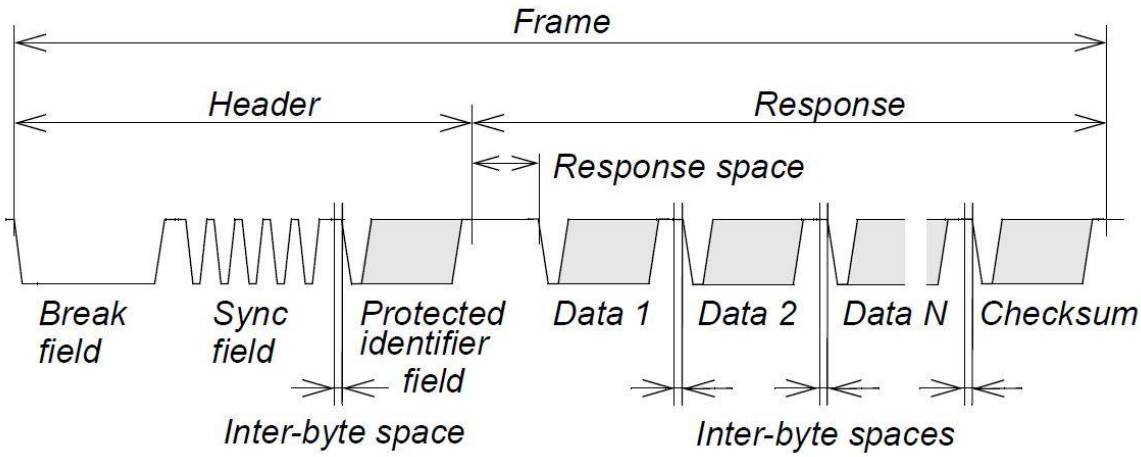
Az autóban található legkevésbé összetett és így a legkedvezőbb költségekkel rendelkező autóipari protokoll a LIN (Local Interconnect Network). Felhasználási területe a vezérlők és egyszerűbb beavatkozók, kijelzők közötti kapcsolatteremtés főként a kényelmi berendezések esetén. A protokollt létrehozó konzorcium 1999-ben jött létre hét autóipari cég együttműködésének eredményeként. Az Audi Group, a BMW, a Daimler-Chrysler, a Motorola, a Volcano, a Volkswagen és a Volvo olyan szabványt hozott létre, amely mintegy a CAN hálózat alrendszereként működve egyszerűbbé és jobban strukturálhatóvá tette a gépjárművek elektromos rendszerét. A

CAN busz költségeinek feléért egy biztonságos Master-Slave elven működő hálózatot más néven klasztert kapunk. A helytakarékosság is komoly érv a LIN protokoll mellett. Ez az egyvezetékes kialakításra és az egyszerűbb kiegészítő áramkörökre vezethető vissza. Az órajel előállításához például elég egy RC rezgőkört használni a költségesebb kvarc helyett.

A Volkswagen konszern maximálisan 16 csomóponttal rendelkező LIN hálózatot alkalmaz, egyébként 63 Slave (plusz 1 Master) áramkörrel működő rendszer is megvalósítható. Sebességét tekintve a protokoll alulmarad a CAN-nel szemben, 1 kBaud és 20 kBaud közötti hálózatok léteznek, de nem is ez a lényeg, hiszen olyan területen alkalmazzák, ahol elég a szerényebb adatátviteli sebesség is.

A LIN busz tehát egy Master és több Slave áramkörből áll. A Master feladata a kapcsolatteremtés a felsőbb szinteken lévő (általában CAN) hálózatokkal, gatewayként funkcionál és Slave-ként is viselkedhet. Kontrollálja a buszt, meghatározza az üzenetek típusát és azt, hogy mikor kerülhetnek a buszra. A teljes hibakezelést elvégzi, és csak ő kezdeményezhet üzenetküldést.

Az üzenetküldés kereteken, frame-eken keresztül történik. A frame-ek bitekből állnak. 0 Volt jelöli a domináns jelszintet, a recesszív jelszint pedig többféle 0-tól különböző feszültség érték is lehet attól függően, hogy a LIN melyik típusáról van szó. Ez azért rendkívül előnyös, mert szakadt vezeték esetén transzparens a hiba megjelenése. Ugyanezt a jelszint logikát használja a CAN hálózat is, csak a feszültségértékek különböznek. A bitek egy általános üzenet esetén a következőképpen csoportosíthatóak: Az üzenet fejléce (header), amit a Master szolgáltat, és a Slave vagy Slave-ek válasza (response), ami adatmezőből (data field), és az ellenőrző összegből (checksum) áll. A Slave-ek csak a Master kérésére válaszolhatnak. Slave-ek közti információcsere is csak a Master utasításán keresztül lehetséges.



4. ábra: LIN frame struktúra

A fejléc tartalmazza a minimum 13 bit hosszúságú szinkronizációs szünetet (sync. break) ezt a részt egy 1 bit hosszúságú recesszív jelszintű határoló (sync. delimiter) zár. Ezután következik a maximálisan 10 bit hosszúságú 01 kódváltásokból álló szinkronizációs mező (sync. field). Itt történik a Masternek és Slave-ek órajeleinek összehangolása. A header utolsó traktusában pedig a 8 bitidő hosszú azonosító (ID) található. 6 bit szolgál az üzenet típusának felismerésére a maradék két bit pedig az ID ellenőrzését végzik. Ezek paritásbitek, amik az ID bitjeinek kizáró vagy kapcsolatából adódnak. 2, 4 vagy 8 adatmező tartalmazza az üzenetet: a megvalósítandó utasítást vagy kérést. Minden adatmező egy domináns startbittel kezdődik, melyet 8 adatbit követ LSB first MSB last formában és végül egy recesszív stopbittel végződik. A frame-et az ellenőrzőösszeg zárja.

Ötféle kerettípus létezik. A Feltétel nélküli keret (unconditional frame) a legáltalánosabb, a headerben megadott összes résztvevő megkapja és végrehajtja a benne foglalt utasítást. A kerettel történik a Slave-ek közötti közvetett kommunikáció megvalósítása is. Az Eseményvezérelt keret (event triggered frame) feladata hogy egyrészt optimalizálja a hálózat sávszélességét és egyben megakadályozza a busz túlterhelését a Slave-ek egyszerre adásának kiküszöbölésével. A Szórványosan megjelenő keretben (sporadic frame) olyan adatok szerepelnek, amelyek nem periodikusan jelennek meg a buszon. A Diagnosztikai keret (diagnostic frame) hordozza a szállítási réteg információit. Mindig 8 adatbájtot tartalmaz. A Fenntartott keret (reserved frame) a LIN jelenlegi verziójában nem használható.

A Master és az egyes Slave-ek közötti kapcsolat pontos leírására az LDF (LIN Description File) szolgál. Minden egyes LIN eszközhöz hozzá van rendelve egy ilyen file. Többek között szerepel benne a beállított adatátviteli sebesség, a kommunikációt végrehajtó jelek nevei és a lekérdezés gyakorisága.

A LIN hálózat fontos jellemzője, hogy képes magát standby üzemmódba helyezni. Ezt az állapotot Sleep Mode-nak nevezzük. A Master utasítására minden résztvevő befejezi a kommunikációt, a busz nyugalmi állapotba kerül. Sleep Mode alatt recesszív jelszint van a buszon, ez a megoldás az energiafelhasználás miatt kedvező, mert így elenyésző a nyugalmi áramfelvétel, ami a gépjárműveknél kitüntetett fontossággal bír (akkumulátor lemerülése). Az alvó állapotot bármelyik résztvevő megszakíthatja domináns jelszint kiadásával.

A protokoll az OSI rétegek közül négyet használ. Amikor a mintavételezés és a szinkronizáció megvalósításáról, a jelek típusáról, a busz meghajtó és fogadó egységeiről van szó, valójában a protokoll fizikai rétegéről beszélünk. Az adatkapcsolati réteget a protokoll specifikáció valósítja meg. Meghatározza a klaszteren küldhető információegységek és üzenetek típusát és kezeli az ütemező táblázatot. A szállítási réteg és a diagnosztikai specifikáció közösen alkotja a következő szintet. Leírja a diagnosztikai üzenetek (csomagok) szerkezetét és például azt is, hogy hogyan küldhető 8 adatbájnál nagyobb összetett üzenet a buszon. A legfelső, alkalmazási réteg egy API (Alkalmazási Program Interface), amely a felhasználó számára láthatatlanná teszi a klaszter konfigurációs részleteit és lehetőséget biztosít a hálózatot elérő alkalmazási programok futtatására. A szabvány mankót ad a LIN hálózatra C nyelven történő software fejlesztéshez, példakódokat tartalmaz és meghatározza a megvalósítandó interface-eket.[4][6]

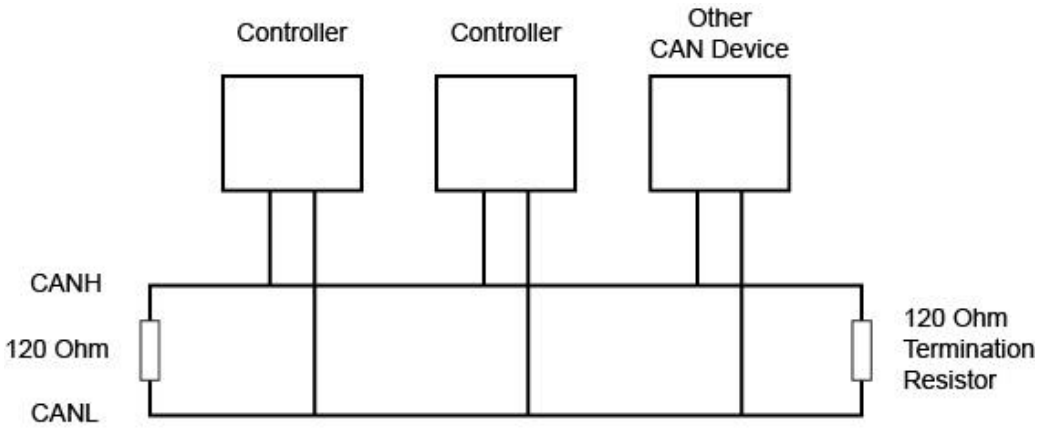
2.2.3 CAN

A LIN-nél összetettebb és sokoldalúbban felhasználható autóipari hálózat a CAN (Controller Area Network). A protokollt a Robert Bosch GmbH fejlesztette ki a nyolcvanas években. A specifikáció 1985-ben készült el, 1987-ben pedig az Intellel való együttműködés következtében megjelent az első CAN vezérlő chip. A protokoll később szabvánnyá vált.

Az aszinkron multi-master topológiát alkalmazó CAN hálózat is az OSI modellre épül, esetében viszont csak három réteg definiált. Az alkalmazási réteget olyan

magasabb rendű protokollok írják le, mint a CANOpen, CAN Application Layer (CAL) vagy a J1939. Ezek nincsenek szorosan a CAN specifikációba foglalva. A klaszter adatkapcsolati és fizikai rétege viszont igen. A protokoll adatkapcsolati rétege a CSMA/CD with NDA (Carrier Sense Multiple Access/ Collision Detection with Non Destructive Arbitration) alapelveit használja. Ha szabad a busz bármelyik résztvevő (multi-master busz) küldhet és mindenki fogad. A rendszer ütközésfigyeléssel van ellátva, minden vezérlő visszaolvassa a buszt, ebből kiderül, hogy adhat-e tovább. A buszhasználat jogának eldöntése vagy más néven arbitráció úgy van megoldva, hogy az üzenet akkor sem sérül, ha a vezérlők egyszerre próbálnak adni. Az arbitrálás akkor kezdődhet, ha a busz nyugalmi állapotban van (minden bit recesszív). Valójában nem a csomópontok, hanem az üzenetek versenyeznek egymással. Az aktuálisan küldésben lévő üzenetek azonosítója bitenként kiolvasásra kerül és a hexadecimális számmal megadott legkisebb ID-val rendelkezőnek a vezérlője élvez prioritást.

A csomópontok közti tényleges jeltovábbítást a fizikai réteg végzi. A jelátvitel a hatékony zavarvédettség elérésének érdekében csavart érpáron történik, melyet a káros reflexiók kiküszöbölésére 120 Ohmos hullámellenállások zárnak. A vezetékre a csomópontok párhuzamosan kapcsolódnak.

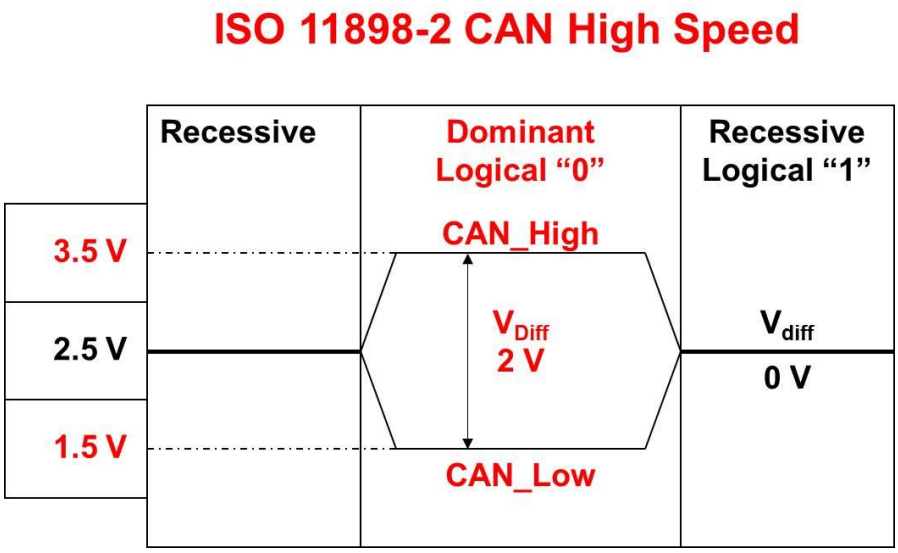


5. ábra: CAN hálózat felépítése

A hálózat átviteli sebessége 100 kBit/s-tól 1 Mbit/s-ig terjedhet. Alacsonyabb sebesség esetén lehetőség van egyvezetékes rendszer használatára is. A buszon kétirányú kommunikáció folyik.

A protokoll jelszintjeinek elve a LIN-éhez hasonlít. A 0 logikai érték a domináns, az 1 pedig a recesszív. Ez a feszültségértékeket tekintve eltérően néz ki, több, sebességfüggő szabvány is létezik. A legelterjedtebb azonban, amikor a 0 értéknél

mindkét vezetéken (CAN High és CAN Low) 2.5 Volt jelenik meg, az 1-es logikai értéknél pedig 3.5 illetve 1.5 Volt.



6. ábra: CAN High Speed jelszintek

Az adatforgalom a CAN hálózaton is, mint a soros hálózatokon általában keretek használatával történik. Négy típusú keret létezik. Az adatkeret (Data Frame) 1 domináns (0 logikai értékű) SOF (Start of Frame) bittel kezdődik, ehhez szinkronizálnak a kommunikáció résztvevői. Ezt követi a 12 bit hosszúságú arbitrációs mező. Az első 11 bit az éppen aktuálisan küldött üzenet azonosítója (ID) MSB – LSB sorrendben. A 12. záró bit az RTR (Remote Transmission Request) meghatározza, hogy adattovábbításról (0 logikai érték) vagy adatkérésről (1 érték) van szó. Utóbbi esetben a keretet Remote Frame-nek nevezzük. Ebből az adatkérő üzenetből hiányzik az adatmező. A Data Frame az ellenőrző mezővel (control field) folytatódik, melynek tartalma az IDE (Identifier Extension) és a DLC (Data Length Code). Előbbi az adatkeret típusát határozza meg. 0 esetén standard, 1 esetén extended azaz kiterjesztett adatkeretről beszélünk. Használata esetén a 11 bites ID 29 bitre bővül, tehát több üzenet definiálható. A DLC 4 biten az adatmező hosszúságát határozza meg: egy kereten belül maximum 8 bájt küldhető. A tényleges információ a 8-64 bit hosszú adatmezőben (Data Field) foglaltatik. A bitek MSB – LSB sorrendben követik egymást. A keret további részében a CRC (Cyclic Redundancy Code) mező kap helyet, amely az ellenőrző funkciókat látja el. Felismeri és jelzi a SOF-től kezdve az adatmező végéig az átviteli hibákat. A keretet a nyugtázó bitmező (Acknowledge Field) zárja. Az 1 bites alapállásban recesszív Acknowledge Slot-ot a vevők 0-ba állítják, ha az üzenetet

hibátlannak találják. A mező második bitje, az Acknowledge Delimiter mindig recesszív értékű, és a nyugtázó mező végét jelzi. A résztvevők először jelzik, hogy megkapták az üzenetet majd utána mérlegelik, hogy szükségük van-e rá. Ezt egy a memóriájukba töltött, üzenet ID-kat tartalmazó adatbázis segítségével döntik el. A keret utolsó 7 bitje mindig recesszív, EOF-nak (End of Frame) nevezik. Két keret között az előző üzenet feldolgozására illetve a következő adás előkészítésére szánt minimum 3 bitidő hosszúságú recesszív IFS (Interframes Space) található, viszont ha ennél több idő is eltelik a következő keret érkezéséig, akkor a busz recesszív, Idle állapotba kerül.

A Data és Remote Frame-en kívül A CAN szabvány még két kerettípust definiál. Hibakeretet (Error Frame) hibát észlelő résztvevő küld a buszra. Az éppen adásban lévő üzenet ekkor érvénytelennek minősül. Az Error Frame minimum 6 maximum 12 domináns és 8 recesszív bitet tartalmaz.

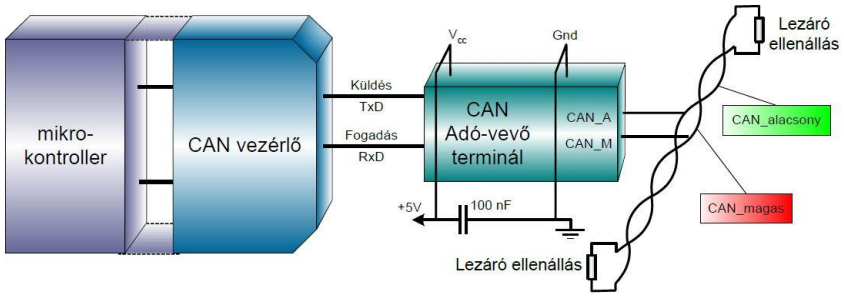
Ha valamelyik vezérlőnek nem sikerült az IFS alatt feldolgoznia a fogadott üzenetet, vagy hibásnak érzékeli az IFS valamely bitjét, akkor kiadja a túlterheltségi keretet (Overload Frame). A keret az Error Frame-hez hasonlóan 6 domináns és 8 recesszív bitet tartalmaz, buszra bocsátása pedig az első ISF bit helyén történik.

A CAN protokollban a szinkronizáció úgy zajlik, hogy minden vezérlőnek a saját erre kialakított áramköre (BTL – Bus Timing Logic) előállítja az órajelet majd a résztvevők ezt a jelet minden felfutó élre összehangolják. Ehhez segítség a SOF illetve a kötelező bitbeültetés vagy Bit Stuffing. A szabály azt írja elő, hogy kötelező módon minden 5. bitnek paritást kell váltania, akkor is, ha a bitsorozat az üzenetben nem így következne. Ha például 10 domináns bit követi egymást, akkor az ötödik után a rendszer automatikusan beszúr egy recesszív bitet. Ezt a plusz bitet a vezérlők automatikusan kiszűrik. Ez a szabály az SOF-tól egészen a CRC mezőig tart, kivéve az Error és az Overload Frame-et.

A CAN klaszter is rendelkezik wake up funkcióval. Ebben az esetben is a gazdaságos energiafelhasználás a cél.

Vizsgáljuk meg, hogy a protokoll rendszerét alkotó összetevők melyik OSI rétegnek felelnek meg! A hálózat működéséhez szükség van magán a vezérlőn kívül egy CAN Controllerre. Ez az alkatrész a mikroprocesszortól megkapott adatokat a CAN Transceivernek továbbítja a megfelelő átalakítások elvégzése után. Feladatköre az alkalmazási réteg elvárásait elégíti ki. A CAN Transceiver képviseli a protokoll

adatkapcsolati szintjét. A Controllertől érkező adatokat a busz számára értelmezhető jelekké alakítja és kiküldi a buszra. A buszról érkező jeleken pedig ugyanezt a metódust végzi el fordított sorrendben. A klaszter fizikai rétegét a csavart érpár és a lezáró ellenállások képezik.



7. ábra: CAN csomópont felépítése

Ezzel megismertük az autóiparban legtöbbet használt protokoll alapjait, a továbbiakban egy olyan hálózatról lesz szó, ami költséges mivolta miatt egyelőre csak a prémium alkalmazásokban jelenik meg.[5]

2.2.4 MOST

A MOST (Media Oriented System Transport) egy szinkron általában gyűrű felépítésű és maximum 64 eszközt foglalkoztató hálózat. Mivel plug & play képes, ezért rendkívül egyszerű az eszközök hozzáadása és eltávolítása a buszról. A rendszer rendelkezik a csillag topológiához megfelelő feltételekkel, így ennek használata is lehetséges. Különösen biztonságkritikus alkalmazások esetén pedig a kettős gyűrűs topológia is elképzelhető. A kommunikáció optikai szálon keresztül zajlik, ezért a rendszer teljesen zavar érzéketlen. A rézvezeték kiváltása miatt a kábelezés tömege is kedvező.

A hálózatban az ún. Timing Master folyamatosan küldött alapjeléhez szinkronizál a többi eszköz, a Timing Slave-ek. Ez a megoldás feleslegessé teszi a puffer használatát és a mintavételi idők egymáshoz hangolását, nincs szükség kiegészítő áramkörökre. A technológia hasonló ahhoz, amit a telefonhálózatoknál használnak. A szinkron alap adatjelében többszörös streamelt adatcsatornák és a vezérlő csatorna is kiküldésre kerül. Utóbbival beállítható, hogy melyik streamelt adatcsatornát kell az adónak és melyiket a fogadónak használnia. Ha megindult a kommunikáció utána már nincs szükség további címzésre. A streamelt adatcsatorna sávszélessége mindig elérhető

és a beállított stream részére le van foglalva, ezért nincsenek ütközések, megszakítások, vagy lassulás az átvitelben. Ez az olyan alkalmazások számára előnyös, ahol az információ folyamatosan áramlik. Ilyenek például az audio vagy video alkalmazások, a MOST buszt ezeknek az adatfolyamoknak a magas minőségű és költséghatékony szállítására fejlesztették ki.

Az internetforgalom vagy a navigációs rendszerből érkező információ tipikusan rövid aszinkron csomagokban kerül a buszra. A MOST ezért rendelkezik az ilyen típusú adatok hatékony kezelésére szolgáló mechanizmussal, ami teljesen el van választva a szinkron adatforgalomtól.

A MOST specifikáció nem csak a fizikai és az adatkapcsolati réteget definiálja, lefedi mind a hét OSI modellben szereplő szintet. Az egységes interface-ek egyszerűvé teszik a protokoll implementálását multimédia eszközök számára. Az alkalmazás fejlesztőknek a MOST elsősorban egy protokoll definíció. A felhasználók egy egységes objektum orientált interface-t, API-t kapnak, amivel hozzáférhetnek az eszközhöz.

A kommunikáció működését a MOST System Services garantálja: Az alsó és középső réteg rendszer szolgáltatásai illetve az Application Socket. Az alsó réteg rendszer szolgáltatásai a modell 2. rétegének felelnek meg. A MOST transceiverében vannak implementálva és a klaszter fizikai rétegén alapszanak. A transceiver és az API (ami a modell 7. szintjét képviseli) között a MOST NetServices található, ami a középső réteg rendszer szolgáltatásaiból (3., 4. és 5. réteg) és az Application Socketből (6. réteg) áll.[9][11]

2.2.5 A K-Mátrix

A fent ismertetett protokollok definiálnak minden eszközt, ami szükséges a kommunikációhoz. A felhasználó vállalatoknak nincs más feladatuk, mint hogy meghatározzák az üzeneteket és jeleket, amik a hálózatokon előfordulhatnak. A Volkswagen konszern esetén ezt az adatbázist K-, azaz Kommunikációs Mátrixnak hívják. A táblázat minden lehetséges üzenet azonosító esetén definiál valamilyen funkciót. Meghatározza, hogy mely vezérlők használják küldőként, melyek fogadóként az adott csomagot és leírja annak pontos tartalmát bitről bitre. A kommunikációt megfigyelve egy adott időintervallumban a táblázat lehetőséget nyújt annak megfejtésére, hogy milyen információk kerültek ki a buszra.

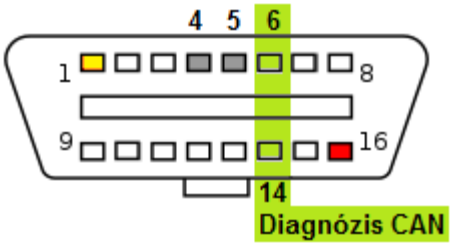
3 Az autó elektromos hálózatának diagnosztikai lehetőségei

A gépkocsik elektromos rendszerének fejlődésével a vezérlők, hibamentes működés esetén, a jármű szinte teljes egészét felügyelet alatt tudják tartani. A legtöbb fontos jellemzőt folyamatosan mérik, és ebből következtetéseket vonnak le a helyes működést illetően. Ma már egy szervizbe behajtó tulajdonos azt tapasztalja, hogy autóját számítógépre csatlakoztatva legtöbb esetben meg tudják mondani a rendellenességet, anélkül, hogy akár a motorháztetőt felnyitnák, vagy megemelnék a gépkocsit. Laboratóriumi körülmények között azonban ennél mélyrehatóbb analitikai és diagnosztikai technikákkal is dolgoznak. Nem csak a vezérlőkből érkező hibakódokat és üzeneteket lehet értelmezni, de az autó különböző hálózataira való kapcsolódással a teljes, akár bitszintű kommunikáció megfigyelhető és vizsgálható. Jelen fejezetben azt tárgyalom, hogy milyen modern megoldások léteznek az autó hálózatának diagnosztizálására, melyek azok az eszközök, amivel elemezni lehet a kommunikációt, és amelyek később a feladatmegoldás során is segítségemre lesznek.

3.1 Az OBD (On Board Diagnostic)

Ma már az eredetileg emissziós értékek ellenőrzésére használt szabvány kommunikációs portján történik a gépkocsi belső hálózatainak teljes diagnosztikája és programozása. Azonban az adatok feldolgozásához más standardokat használnak. A VW konszern jelenleg az UDS kommunikációs protokollt használja, ami ODX (Open Diagnostic eXchange format) fájlokat alkalmaz. Erről a következő fejezetben lesz szó.

Az OBD II hibakódok (DTC – data trouble code) segítségével ad információt az egyes komponensek állapotáról és működéséről. A PID-nek (parameter ID) is nevezett kódok kiolvasása és az élő adatok megfigyelése standardizált digitális kommunikációs porton történik a Data Link Connector segítségével. Ennek a D-alakú 16 pines szabványos hardware interface-nek két típusa létezik, az A és a B típusú csatlakozó.



8. ábra: OBD II /A csatlakozó pin kiosztása

Az OBD II többféle kommunikációs protokollt is elfogad. A Volkswagen konszern autóiba a CAN protokollal ellátott rendszer épül. Ezeknek a csatlakozóknak a lábkiosztása a következő: test (szürkével jelölt 4-es és 5-ös pin), KL15 tápfeszültség (sárgával jelölt 1-es pin), KL30 tápfeszültség (pirossal jelölt 16-os pin), CAN (6-os és 14-es pin). A diagnosztikai szolgáltatások megkönnyítése érdekében különböző felhasználói felületeket alkalmaznak. Ezek közül most a VW konszernnél használtnak a rövid bemutatása következik.[7][10]

3.2 Vezérlők diagnosztikai lehetőségei, az iDEX

Az OBD II előnyös tulajdonságait felhasználva az autóipari vállalatok olyan szabványok fejlesztésén dolgoztak tovább, amely az OBD II-ből ismert DTC-ket felhasználóbarát formában képesek megjeleníteni a teszt berendezéseken vagy akár PC-n. Egy elterjedt irány az ISO szabványok közé is felvett UDS protokoll és annak meghatározó formátuma az ODX. [8] Az ODX adatok konvertálás nélkül is ember által értelmezhető információt szolgáltatnak. Az ODX valójában egy adatbázis, amely magában foglalja a vezérlők közti kommunikáció adatkapcsolati rétegét, az ECU-k konfigurálásával és felprogramozásával (flashelésével) kapcsolatos információkat, ezen kívül ismerteti az interface-ek és csatlakozók pin kiosztását és a tulajdonságait.[2]

Mindezt azért volt fontos megismerni, mert a Volkswagen konszern autóinak teszteléséhez, így az Audi TT-hez használt diagnosztikai program is az ODX modellre épül.[1] Az software neve iDEX (Intelligent Diagnostics Environmnet for ODX).

A kívánt autótípus kiválasztása után lehetőség nyílik a vezérlőkben tárolt hibakódok kiolvasására, mérőblokkok megfigyelésével az autót jellemző, különböző fizikai mennyiségek értékeinek elemzésére. Egyszerűbb funkciótesztek és makrók futtathatók, vagy akár az ECU-k újraprogramozása is megtörténhet. Ez jelentheti a vezérlőn futó software frissítését vagy új software telepítését. Korábban említettem,

hogy a vezérlőt adott autótípusban elvárt működéséhez be kell kódolni. Az ECU-t ennek a software-es beavatkozásnak a során látják el a szükséges paraméterekkel, ami alapján dolgozni fog. A kódolást és parametrizálást is el lehet végezni a program segítségével.

9. ábra: Az iDEX kezdőképernyője

A program diagnózis menüjébe belépve lista jelenik meg, amely tartalmazza a vizsgált autóban szereplő, gateway által elérhető összes vezérlőt. Ezek közül egyet kiválasztva megjelenik az adott ECU memóriájában lévő hibakódok listája, jelentésükkel és tulajdonságaikkal. A lekérdezett adatok XML (Extensible Mark-up Language) formátumú fájlban tárolódnak. Ha megvan a hiba forrása, és a rendellenesség javítása megtörtént, a vezérlő hibatároló memóriája ugyanebben a menüpontban törölhető. Számomra a szakdolgozat feladatának megoldása szempontjából ez a funkció lesz az egyik legfontosabb.

A programmal előre beállított tesztparancsok is kiadhatók. Ezekkel diagnosztikai szempontból értékes feladatok elvégzésére bírható a rendszer. Elérhető például, hogy egy kijelző berendezés egyszerre működtesse minden szegmensét. Ez a másik szükséges funkció, amit használni fogok.[3]

3.3 A Vector cég és a 1630A-s interface

A fejezetnek ebben a részében érkeztem el annak a hardware és software eszközökből álló diagnosztikai- és fejlesztőcsomagnak az ismertetéséhez, amelynek

segítségével a szakdolgozatom feladatkiírásában megfogalmazott tesztberendezést megvalósítom.

A Vector céget 1988-ban alapították, abban az évben, amikor az első Intel CAN Controller is piacra került. A vállalat a beágyazott rendszerek tervezéséhez és fejlesztéséhez kínál különböző hardware-es és software-es szolgáltatásokat. Ezeknek az eszközöknek a segítségével valós időben figyelhető meg az autóipari hálózatokon folyó kommunikáció, ezzel hatékonyan támogatva a rendszer analízisét.

10. ábra: VN 1630A interface

Az általam használt VN 1630A-s típusú busz interface 4 csatornával rendelkezik. A számítógéphez USB-n keresztül csatlakozik, a működéshez szükséges tápellátást is innen kapja. CAN és LIN kommunikáció megfigyelésére alkalmas. Képes a többi résztvevő számára észrevétlenül belehallgatni az adatfolyamba, de keretek küldése és fogadása is lehetséges a segítségével. Felismeri a kommunikációban előforduló hibákat, Error Frame-eket, checksum eltéréseket vagy szinkronizációs problémákat. Méri a buszra jellemző adatokat: a sebességet, a terheltséget, a keretek hosszát. A hálózat konzisztenciájának vizsgálatára is képes, figyelmeztet, ha valamelyik résztvevő hiányzik, vagy hibás az egyik rendszer kapcsolatokat leíró fájl. Az interface önmagában, annak ellenére, hogy viszonylag széles spektrumú tudást mutat fel, nem hatékony. Szükség van egy software-re, amivel a hardware által szolgáltatott adatok kielemezhetők, a beavatkozások véghezvihetők, tesztek futtathatók. Ez a programcsomag esetemben a CANoe, ami szintén a Vector cég terméke.

3.4 Buszrendszerek analíziséhez használt CANoe program

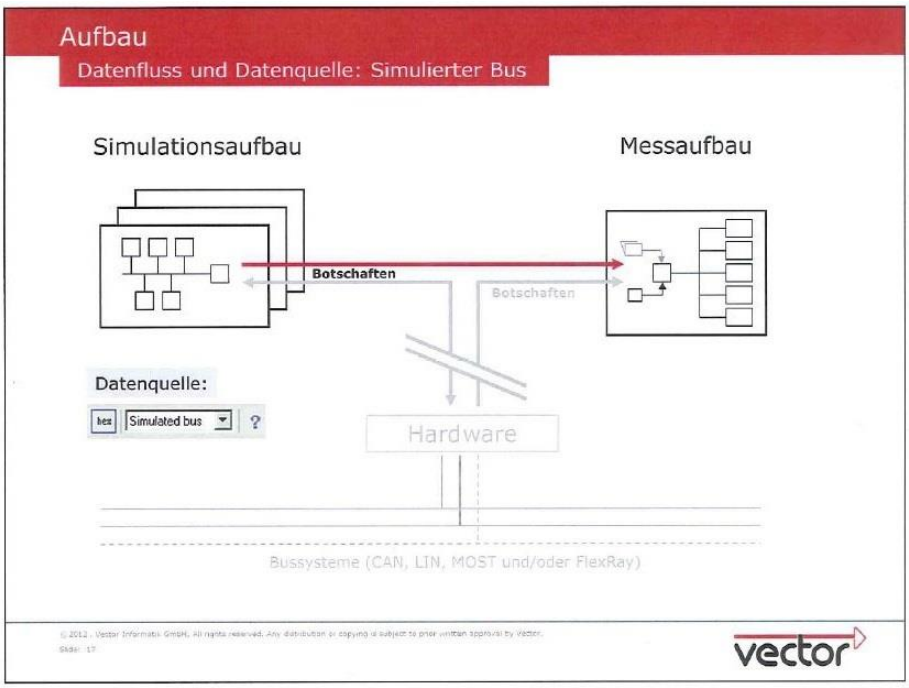
A software meglehetősen bőséges eszköztárral rendelkezik. A probléma megoldása során nagy részét használom ezeknek a szolgáltatásoknak, ezért pár szóban összefoglalom működésük jellemzőit.

A program használata valós hálózatok analízisére csak egy licenccel védett interface közbeiktatásával lehetséges. A hardware eszköz a hálózat egyik node-jaként viselkedve hajtja végre a software által kitűzött feladatokat.

Egy konfiguráció létrehozása során két alapvető rendszert kell felépíteni. Az egyik a mérési elrendezés, a másik pedig a szimulációs környezet. Utóbbi esetén lehetőség van egy tetszőleges hálózatot leíró adatbázis programhoz rendelésére. A rendszer ennek alapján virtuálisan felépíti a teljes kommunikációs környezetet. Meghatározza az adatbázisban szereplő csomópontok egymás iránti kapcsolatát és a hálózatra küldhető üzenetek listáját. Ezen kívül definiálja az időzítéseket és a protokoll többi jellemzőjét. A szimulációs környezetet kiegészíthetjük visszajátszó blokkokkal, interaktív generátorblokkokkal, vagy CAPL programot futtató virtuális node-okkal. A fejezet során részletezem, hogy mire használhatók az előző felsorolás szereplői. Ha a szimuláció felépítése teljes, akkor akár el is indítható a mérés, hiszen létrejött egy virtuális hálózat, aminek tagjai képesek adatforgalmat bonyolítani egymással.

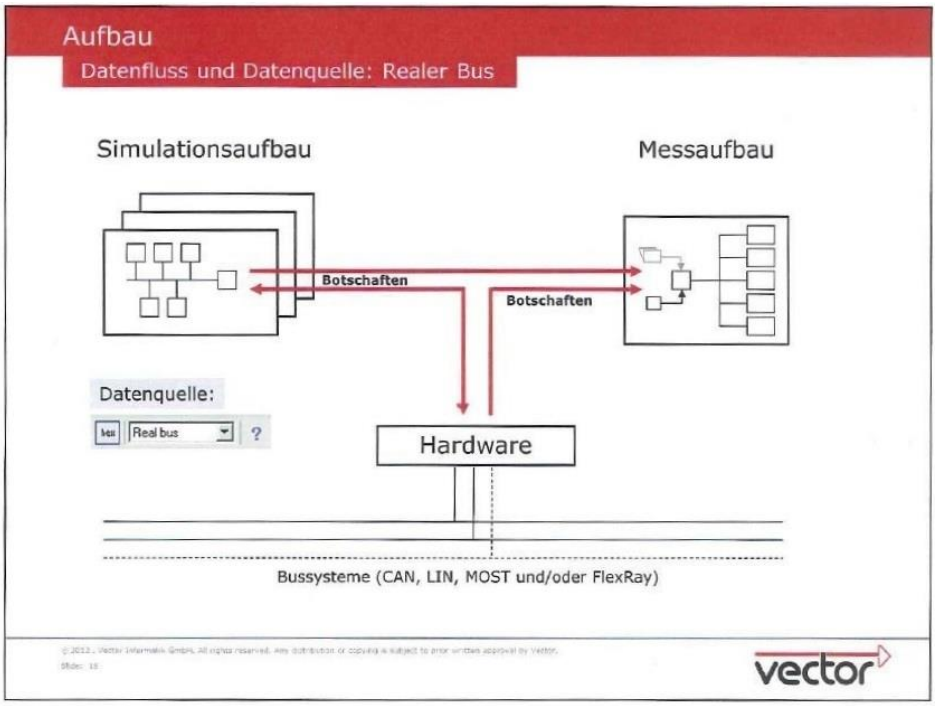
A mérési elrendezésre azért van szükség, hogy a szimulációs környezetben elindított kommunikációt minél precízebben meg lehessen figyelni. A mérési struktúra különböző mérőablakokat tartalmaz, amelyek működésére később kitérek. Ebben a logikailag elválasztott részben az üzenetek bemenetként jelennek meg, innen üzenet kiküldés nem történik. Míg a szimulációs blokkban az üzenetek áramlása kétirányú.

Ha a hálózatot csak a szimulációs környezet és a mérési elrendezés alkotja, akkor szimulált buszról beszélünk.



11. ábra: Szimulált környezet a CANoe-ban

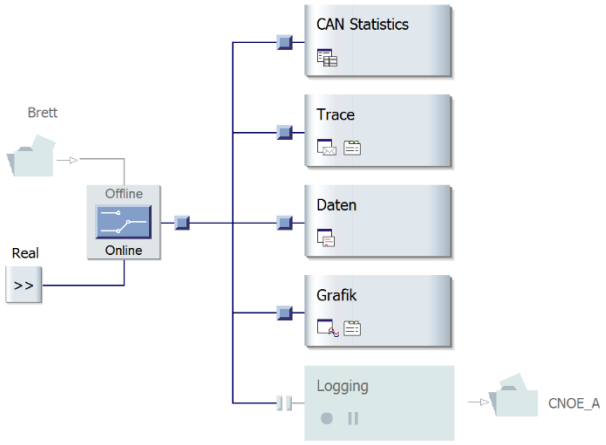
Ebbe az összeállításba azonban bekapcsolható egy valós, fizikailag is megvalósított, az interface-en kapcsolódó buszrendszer. Ilyenkor az üzenetek küldése a szimulációs blokk és a valós busz között oda-vissza történik, a mérési blokk pedig megfigyelőként kapja meg a kereteket, ahogy ez az ábrán is jól látható



12. ábra: Részben szimulált környezet a CANoe-ban

Mielőtt elkezdeném bemutatni a különböző blokkokkal elvégezhető műveleteket, fontos még szót ejteni a write ablakról. Ezen a felületen jelenik meg minden méréssel kapcsolatos alapvető információk. Ilyen például az aktuálisan használt protokoll, annak sebessége, hibaüzenetek és figyelmeztetések a mérés során eltelt idő függvényében. Hibakeresésnél nagy segítség ez a szolgáltatás. Ha a később ismertetésre kerülő CAPL programnyelven write() parancs kiadása történik (ez a C-ben használt printf()-hez hasonló függvény) a kimenetre írt szöveg is ebben az ablakban jelenik meg.

3.4.1 A mérési felépítés blokkjai



13. ábra: Egy mérés általános felépítése

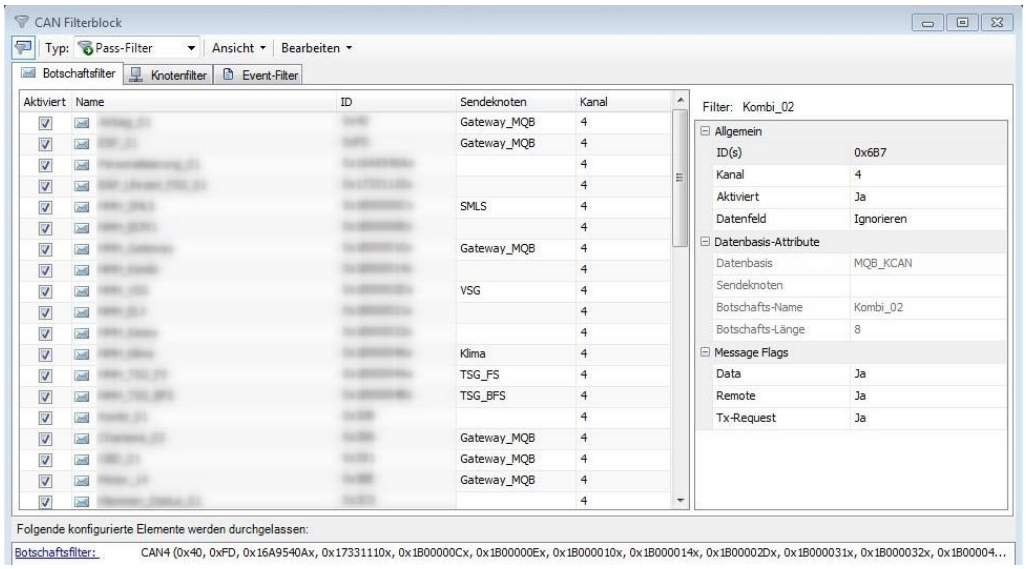
3.4.1.1 Üzenetek értelmezése: ki küldi, kinek és mit

A Trace ablak mutatja meg a buszon folyó tevékenységet a mérés során. Az összes Trace blokkba érkező üzenet és az ezeket felépítő szignál kiértékelődik és táblázatos formában kiírásra kerül. A táblázat sorai maguk a keretek, oszlopai pedig a rájuk vonatkozó információk: az üzenet neve, a csatorna, amin érkezett, az időpont, amikor érkezett, az üzenet hossza, adatmezőjének tartalma hexadecimálisan megadva és az, hogy küldött vagy fogadott frame-ről van-e szó. Beállítható, hogy a több alkalommal frissülő üzenet egyszer szerepeljen a felsorolásban vagy pedig a valós érkezési sorrend kerüljön megjelenítésre.

14. ábra: A későbbi mérés Trace ablaka

3.4.1.2 Szűrési lehetőségek

Egy kommunikációt megfigyelve rengeteg olyan üzenettel találkozik az ember, ami az adott mérés szempontjából nem tartalmaz lényegi információt. Azért, hogy a Trace ablakban látható frame-ek valamilyen szisztéma szerint csoportosíthatók legyenek szűrőket lehet tenni az üzeneteket megjelenítő blokk elé. Egész csatornák ignorálása is megvalósítható, de a nem szükséges üzenet külön-külön szűrése is lehetséges, vagy akár egy esemény bekövetkezésétől is függővé tehetjük az adatok átengedését. Alkalmazhatunk blokkoló vagy áteresztő szűrőt. Az első esetben azokat a frame-eket kell kiválasztani, amire nem vagyunk kíváncsiak, a másodikban pedig azokat, amelyeket meg szeretnénk figyelni.



15. ábra: A későbbi méréshez használt, szűrt üzenetek listája

3.4.1.3 Grafikus megjelenítés

Módunkban áll a szignálok grafikus megjelenítése. Ilyenkor a program egy adott üzenetet vagy annak részét függvényként rajzolja a képernyőre. Egyik tengelyt az idő,

másikat pedig a jelhez hozzárendelhető fizikai mennyiség alkotja. Szemléletes megoldás az adatok megjelenítésére és nagyban megkönnyíti a kiértékelést.

3.4.1.4 Adatrögzítés

A valós vagy szimulált buszforgalmat rögzíthetjük egy erre definiált blokk segítségével. Elengedhetetlen ez a szolgáltatás a kommunikáció kis részletének analízisénél, amikor később is rekonstruálni szeretnénk az adatfolyamot. A program egy bináris, blf formátumú fájlban tárolja a felvételt, amit később egy Replay blokkban, a szimulációs környezetben játszhatunk vissza.

3.4.2 A szimulációs környezet blokkjai

A szimulációs környezet is, a mérési felépítéshez hasonlóan blokkokból áll. Legfontosabb ilyen egység magának a busznak a megjelenítése. Ez gyakorlatilag a protokollt és a fizikai vezetékezést szimbolizálja. Ehhez kapcsolódnak a hozzáadott adatbázis alapján létrehozott, korábban már említett node-ok. A szimulációs környezetbe azonban a felhasználó által definiált blokkok is kapcsolható, ezek bemutatása következik most.

3.4.2.1 Szimulációs lehetőségek

A feladat megoldása során három szimulációs blokktípust használtam, az interaktív generátorblokkot, a Replay blokkot és a virtuális ECU-t. A Replay blokkról korábban említettem, hogy a felvett kommunikáció visszajátszására használható. Ennél kifinomultabb megoldás az interaktív generátorblokk használata. Ennek az eszköznek a segítségével tetszőleges üzeneteket lehet kiküldeni a buszra. A kellő paraméterek megadása után beállítható, hogy a frame adott eseményre, például egy gomb megnyomására vagy bizonyos időközönként kerüljenek a hálózatra. Így elérhető, hogy a felhasználó is aktívan részt vehessen a kommunikációban. Legintelligensebb megoldás talán a virtuális ECU használata. Ilyenkor egy hálózatra kapcsolódó, node-ként viselkedő vezérlőt szimulálunk, amin egy felhasználó által írt program fut. Ez az alkalmazás képes a buszról érkező üzenetek fogadására, hatásukra műveletek elvégzésére, lokális vagy rendszerváltozók értékének módosítására és frame-ek kiküldésére. A virtuális ECU-n futó program nyelve a CAPL, amit a következő pontban ismertetek.

3.4.2.2 CAPL programnyelv megismerése

A vezérlők működésének leírására a CANoe környezetben és általában a Vector cég által fejlesztett software-ekben a CAPL (Communication Access Programming Language) nyelvet használják. Ez az eseményvezérelt programnyelv szintaktikáját és felépítését tekintve a C-re hasonlít leginkább. Fejlesztőkörnyezete az ún. CAPL Browser. Az eseményvezérelt felfogás annyit tesz, hogy a programrészletek egy bizonyos változás bekövetkeztekor futnak le. Ez a változás lehet a program elindítása, egy gomb megnyomása vagy például egy adott típusú üzenet érkezése. A CANoe-ban egy-egy konfigurációhoz definiálhatunk rendszer szintű változókat, amelyeket a programban bárhol, így egy CAPL alkalmazásban is elérhetünk. Ezek használata rendkívül előnyös, hiszen bárhol módosítható, vizsgálhatók és kijelezhetők minden nehézség nélkül és ezek biztosítják a programok közti átjárást.

A CAPL támogatja a legtöbb megszokott típust (int, double, char), azonban olyan típusok használata is lehetséges, ami kimondottan buszrendszerek esetén előnyös. Ilyen például a message, amivel egy már definiált üzenet mintájára hozhatunk létre változót.[12]

4 A klíma vezérlő tesztelése és működésének ismertetése

A második fejezet második szakaszában elhelyeztem a vizsgált vezérlőt, a klímavezérlőt és a hozzá kapcsolódó kiegészítő áramköröket az autóban, és bemutattam a hálózatban elfoglalt szerepkörüket. Jelen fejezetben ismertetem azokat a méréseket és teszteket, amelyek a végső cél eléréséhez, tehát a klímavezérlő és kiegészítő áramkörének a gépjárműtől független működtetéséhez vezetnek.

A kitűzött feladatot, mint azt a bevezetőben is kifejtettem, két részre lehet osztani. Elsősorban a klímavezérlőt kell a hálózaton kívül működésre bírni és tesztelhetővé tenni. Fontos továbbá, hogy a vezérlőhöz tartozó kijelző és szabályozó gombok funkcionalitása is vizsgálható legyen. Az egyik kezelőegység tesztkörnyezete is a dolgozat részét képezi, ezért a LIN Slave viselkedését is fel kell térképezni a működésének megismerésére irányuló tesztek elvégzésével.

A kontrollerrel kapcsolatban legfontosabb megvalósítandó elvárás a vezérlő hibatárának kiolvasása volt. Ez azt jelenti, hogy egy beszállítótól, sorról vagy gyáron kívüli szervizből érkező klímavezérlőt az általam megvalósított mérőállomásra csatlakoztatva meg lehessen mondani, hogy milyen hibabejegyzések szerepelnek az ECU memóriájában. Az ezzel kapcsolatos legnagyobb kihívást az jelentette, hogy a mérőállomás csak kiolvassa, és ne befolyásolja ezeket a hibaüzeneteket, általa ne jelenjenek meg újabb hibák. Az újabb hibák alatt azt értem, hogy a vezérlő tesztelése során a perifériáinak döntő többsége hiányzik. Ezeket a szabadon maradt lábakat a vezérlő normál esetben diagnosztizálja és amennyiben szakadást észlel az áramkörben, több hibatár letárolásával jelzi azt. A vizsgálataim szerint a kontroller különböző tápfeszültségekre különbözőképpen reagál. Feladatom ezeknek a reakcióknak a felderítésével kezdődött. A cél az volt, hogy a vezérlőt egy olyan „relaxációs állapotban” vizsgáljam, mely során a perifériáit nem hiányolja, de a korábban beíródott hibatárakat engedi kiolvasni. Így megspórolható a vezérlő számtalan hőmérséklet mérő és állító motorjának szimulációja.

4.1 Vezérlő működésének vizsgálata különböző tápfeszültségek esetén

Az E-Laborban található mérőpadon, ahogy korábban beszámoltam róla az Audi TT 3. generációjának teljes elektromos rendszere kiépítésre került. Ezen kívül az analízist segítendő, a hálózatot több mérési funkcióval is ellátták. A mérőpadhoz kapcsolódik egy stabilizált laboratóriumi tápegység. Megoldható, hogy az egyes vezérlők ne az autó akkumulátoráról kapják a tápot, hanem erről a stabilizált forrásról lehessen táplálni őket. Méréseimet a klímavezérlő labortápegységre való csatlakoztatásával kezdtem. A kezelőegység reakcióit is figyelembe véve a következő eredményeket kaptam:

	működési feszültségtartomány alsó határa (feszültség csökkentése esetén)	működési feszültségtartomány alsó határa (feszültség növelése esetén)	Hibakód vagy iDEX megjegyzés
Hibatár olvashatósága	6,1 V	6,6 V	Negative Response von STG Der Vorgang wird abgebrochen
kijelző- és szabályozó gomb működési tartománya	5,5 V	9,1 V	Versorgungsspannung Spannung zu niedrig 20002 Hex, Funktionsbeschränkung durch Unterspannung 20004 Hex, 5,5 V alatt bekapcsol a befúvó motor
LIN kommunikáció működési tartománya	8,7 V	9,1 V	

1. táblázat: A klímavezérlő és a kezelőegység működési tartományai

Amint látható a vezérlő és a beavatkozó szerv különböző tápfeszültségekre adott válasza hiszterézises jelleget mutat. Ha 12 V-ról indulunk, a feszültséget csökkentve egészen 6,1 V-ig képes kommunikálni a vezérlő a fentiekben megismert iDEX diagnosztikai programon keresztül. Ilyenkor ugyan megjelenik két hibaüzenet, de ezeket ismerve és ezután figyelmen kívül hagyva a vezérlő működése tesztelhető. A táblázatban németül olvasható hibatár bejegyzések fordítása: „A tápfeszültség túl alacsony.” és „Funkciókorlátozás alacsony tápfeszültség miatt.” Amikor a magasabb feszültségértékek felől indulva elérjük a 6,1 V-ot a kommunikáció megszűnik, az utolsó hibaüzenet a „Negatív válasz a vezérlőtől, a folyamat megszakad.” Ez alatt a feszültség alatt a vezérlő áramkörei nem funkcionálnak. Ha elértük ezt a szintet és újra működésbe

akarjuk hozni a kontrollert, a feszültséget egészen 6,6 V-ig kell emelnünk. Ezt jelenti a hiszterézises jelleg: a rendszer nem azonnal reagál a feszültségemelkedésre, késleltetéssel észleli csak, hogy a működéséhez szükséges érték már előállt. A működésbe lépési határ tehát függ az előzményektől. Ugyanezt tapasztaltam a kezelőegységgel kapcsolatban is. Itt két részre lehet osztani a megfigyelést. A táblázat második sorában a kijelzés működésével kapcsolatos értékek láthatók. Ebben a tartományban a kijelző egy korábban beállított értéket mutat, azonban nem reagál az őt érő változásokra így például a gomb forgatására. A kijelzett érték csak abban az esetben változik, ha a LIN kommunikáció is fennáll. Ennek feltétele a táblázat harmadik sorában olvasható. Ezzel a méréssel megállapítottam, hogy ha a vezérlőt 6,1 V-nál nagyobb viszont az akkumulátor aktuális feszültségénél (13,6 V – 14,4 V) kisebb feszültségértékek között üzemeltetjük, akkor hibatárai autótól és a vizsgálópadtól függetlenül kiolvashatók anélkül, hogy új hibákat okoznánk. A tekerőgombnál is hasonló a helyzet. Ott viszont a feszültségértékeknek 8,7 V-nál kell nagyobbnak lenniük.

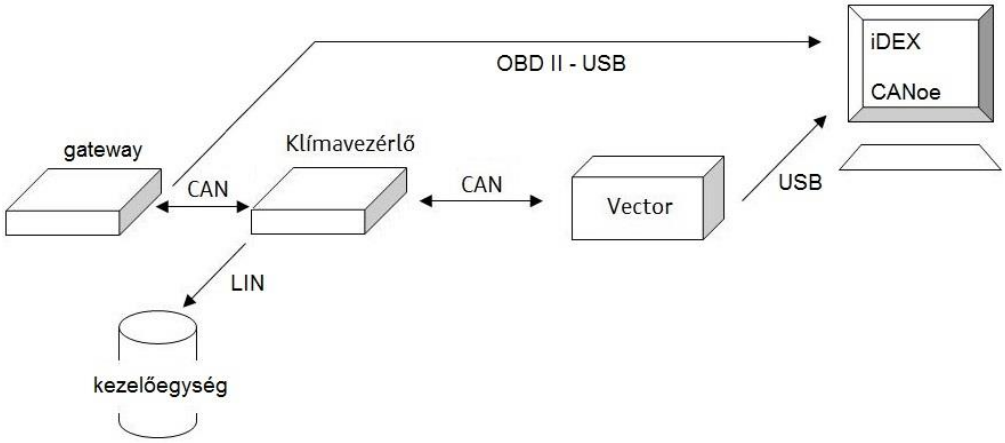
Kijelenthető tehát, hogy a szimuláció során alkalmazott ideális feszültségérték 8,9±0,1 V. Ezt a felismerést ki lehet használni az analízisek során.

4.2 Hálózati (CAN / LIN) kommunikáció felderítése

Következő lépésként a CAN hálózaton folyó, a klímavezérlőt érintő kommunikáció részleteit vizsgáltam. A koncepció az volt, hogy ha egy speciális helyzetben meg tudom figyelni a buszon lévő üzeneteket, akkor azokat rekonstruálva a gépjárműtől független környezetben is működtetni lehet a vezérlőt.

A vizsgálatot a vezérlőt érintő CAN üzenetek kigyűjtésével kezdtem. Ehhez a művelethez a K-Mátrix adatbázisa volt segítségemre. Az előző fejezetekben kitértem arra, hogy ebben a táblázatban rendelkezésre áll valamennyi üzenet, amely a hálózaton megjelenhet. Az adatbázis a frame-ek rövid leírását is tartalmazza. Azokra az üzenetekre koncentráltam, amiket a klímavezérlő fogad. Ezeket a későbbiekben szimulálva a klímavezérlő a perifériákból és a többi vezérlőből érkező jelként fogja értékelni őket és ezáltal úgy fog viselkedni, mintha egy valódi autóban dolgozna. Miután táblázatos formában megvoltak az üzenetek, fizikai szűrésbe kezdtem a CANoe program segítségével. Ez még mindig a vizsgálópadon zajlott. A mérőpad beépített CAN-interface-eivel a gépjármű teljes kommunikációja valós időben nyomon

követhető. Számomra elég volt a Komfort-CAN monitorozása, hiszen a klímavezérlő ehhez a hálózathoz kapcsolódik. A többi csatorna tartalmát ezért kiszűrtem. Ezután következett az üzenetek egyesével való kiválogatása, mert a Komfort-CAN üzenetein belül sem volt mindegyikre szükség. Miután végeztem ötvennyolc típusú frame maradt. Ezek közül külön ki kell emelni a gateway-ből érkező üzenteket. Ezekre azért van szükség, hogy az iDEX diagnosztikai program szolgáltatásait használni lehessen. A mérés a később ismertetett szegmenstesztt kapcsolatos. Az alábbi ábrán nyomon követhető ennek a vizsgálatnak a felépítése.



16. ábra: A diagnosztika folyamata gateway-en keresztül

A CAN hálózatra természetesen a többi vezérlő is csatlakozik a gateway kiemelését csak a diagnosztika szempontjából tartottam fontosnak.

A CANoe program segítségével felvételt készítettem a már megszűrt CAN hálózat kommunikációjáról és ezt a későbbi visszajátszás céljából elmentettem. Az adatok elemzése során sikerült megtalálnom azokat a szignálokat, amik Roadster típusú autótető nyitásáról szolgáltattak információt. Ezeket a jeleket egy grafikus ablakban megjelenítettem a CANoe programban. Ezen adatok birtokában elkezdhettem a szimuláció megtervezését a mérőállomásra.

5 A szimulációs környezet megvalósítása

A klímavezérlő szimulációs környezetét egy már létező mérőállomás, a feladat elvárásainak megfelelő módosításával valósítottam meg. A rendszert a mérési elrendezést összekötő kábelezéssel egészítettem ki és felépítettem a „Restbus" szimulációhoz szükséges szoftveres konfigurációt.

5.1 Meglévő mérőállomás bemutatása

Az átalakításra szánt eszköz, melyet a diplomamunka gyakorlati megvalósításához felhasználhattam, egy korábban új software verziók vezérlőre töltésére, flashelésre használt állomás.

A mobil mérőasztalon egy PC, egy labortápegység, egy a TT-ben is használatos gateway és egy csatlakozókat rögzítő tábla kapott helyet. A PC-re az általános programok mellett a korábban bemutatott, külön licenccel rendelkező CANoe program is fel van telepítve. A mérőállomáson diagnosztika szempontjából elengedhetetlen szabványos OBD II csatlakozó is megtalálható.

5.2 Tápfeszültség és kábelezés megvalósítása

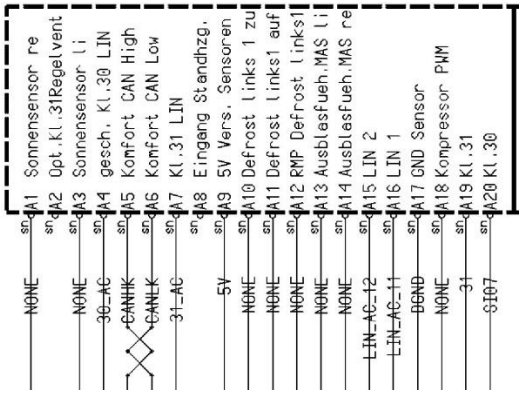
Értekezésem bevezetőjében a rendszerterv keretein belül már vázlatosan bemutattam, hogy a szimulációs környezet egyes elemei hogyan kapcsolódnak egymáshoz. Előrevetítettem, hogy a klímavezérlőt a Vector interface-szel és a tekerőgombbal összekötő kábelezéssel még részletesebben foglalkozom, mert nem szabványos vezetékezésről van szó, az alkatrészek összekötése is a feladat részét képezi.

Az autóba kerülő elektromos alkatrészek csatlakozói fizikai kódolással vannak ellátva. Csak az összetartozó aljzatok és csatlakozók illenek egymásba, így a szerelésnél adódó hibalehetőségek közül a rossz bekötés kiküszöbölhető.

A kábel elkészítéséhez két speciális és egy 9-pólusú D-SUB csatlakozóra volt szükségem. A klímavezérlő négy csatlakozóval kapcsolódik a hálózat többi részéhez.

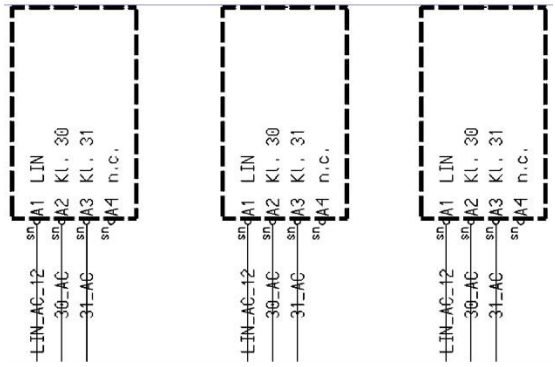
17. ábra: A klímavezérlő és csatlakozói

Az egyik speciális csatlakozó a klímavezérlőhöz kapcsolódó legnagyobb, húsz pólusú konnektor. A másik három aljzat be- és kimenetei a fenti mérés eredményeként a tápfeszültség lecsökkentésével ignorálhatók, hiányuk az általam vizsgált funkciókat nem befolyásolja. A téma szempontjából fontos csatlakozó pin kiosztása az alábbi ábrán látható.



18. ábra: A klímavezérlő 20 pólusú csatlakozója

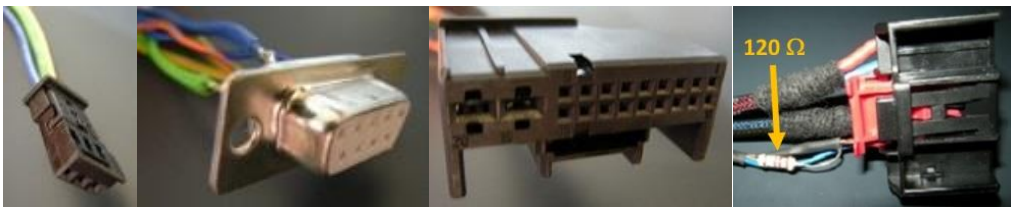
Ezen a 20 pólusú érintkezőn belül sincs minden pinre szükség. Elég, ha a vezérlő tápfeszültséget kap a 19-es (Kl. 31 – test) és a 20-as (Kl. 30 – pozitív táp) kivezetésen keresztül illetve kapcsolódik a Komfort CAN hálózatra az 5-ös (CAN High) és 6-os (CAN Low) pineken keresztül. Ahhoz, hogy a kontroller a tekerőgombbal is kommunikálhasson egy LIN hálózaton keresztül be kell kötni a protokoll jelvezetékét a 16-os kivezetésre (LIN 1). Ezen felül mivel a LIN Slave számára a Master szolgáltatja a tápellátás a 4-es (Kl. 30 LIN) és 7-es (Kl.31 LIN) pinek kivezetése is szükséges. Az utóbbi három vezeték a tekerőgombhoz az alábbi csatlakozóval kapcsolódik.



19. ábra: A LIN Slave-ek pin kiosztása

A kezelőegység vezetékezése ipszilon elágazást tartalmaz, és így csatlakozik 9-pólusú D-SUB csatlakozón keresztül az interface LIN csatornájához a busz adatforgalmának megfigyelése és a későbbi szimuláció céljából. A vezérlőből induló CAN vezetékek is ugyanehhez az RS-232-es csatlakozóhoz vannak forrasztva, mivel az interface egy csatlakozón két csatornát is tud egyidejűleg kezelni. A csatornák kiosztása és pinekhez rendelése előre meghatározott, megtekinteni a CANoe-n keresztül a hardver eszközök menüjében lehet. Az ebben a kontextusban használt Komfort CAN hálózaton már természetes a klímavezérlő és az interface közti kapcsolatot kell érteni, mivel már a szimulált buszrendszer előkészítése folyik.

A vizsgált vezérlő a tápfeszültséget a mérőállomás csatlakozótáblájáról kapja, ami össze van kötve egy stabilizált tápegységgel. A csatlakozókról készült képek:



20. ábra: A csatlakozók balról jobbra: LIN Slave, 9-pólusú D-SUB, klímavezérlő szemből, klímavezérlő oldalnézetből

A képekkel kapcsolatban két dolgot érdemes feljegyezni. A sorban harmadikként következő képen megfigyelhető, hogy a bal szélső pinek robosztusabbak. Ez a klímavezérlő 20 pólusú csatlakozója a kiemelt pinek pedig a kontroller tápellátásáért felelnek. A másik részlet, amire fel szeretném hívni a figyelmet, a negyedik képen szereplő ellenállás. Szintén a 20 pólusú konnektort látjuk. Az ellenállás a CAN vezetékekhez kapcsolódik. Ez a protokoll specifikációjában definiált, reflexiók ellen védelmet nyújtó 120 Ohm-os hullámellenállás. Ismertettem az egyes résztvevőket

összekötő vezetékezést, ezzel a hardware-es átalakításokat ki is merítettem. A soron következő fejezetekben a software-es megvalósítás bemutatása következik.

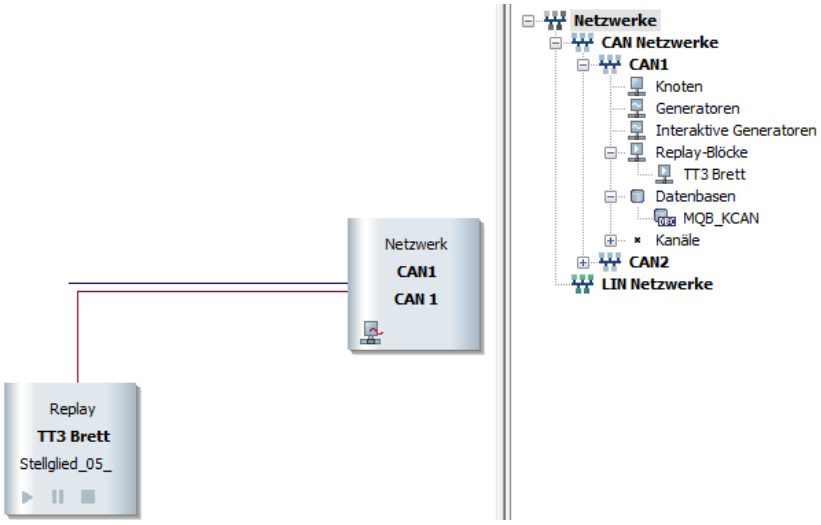
5.3 A „Restbus” szimuláció

Restbus szimuláció esetén egy vagy több valódi node egy interface-en keresztül a szimulációs környezethez, jelen esetben a CANoe-hoz van kapcsolva. A hálózat további elemeit (Rest - maradék) a software szimulálja. A CANoe program segítségével a klímavezérlőn és a tekerőgombon Restbus szimulációt hajtottam végre. A software-rel kapcsolatban folytatott tevékenységem részletei következnek.

5.3.1 A vezérlő élesztése szimulált környezetben és vizsgálható funkcióik felderítése

Belépéskor a CANoe programba meg kell határozni az alkalmazandó protokollt. Az első részfeladat megvalósítása a klímavezérlővel és így a CAN protokollal kapcsolatos. Egy új konfiguráció létrehozásánál ellenőrizni kell a hardware beállításait is. Ki kell választani a használni kívánt csatornát és meg kell bizonyosodni róla, hogy a csatorna elvárásainak megfelelően csatlakoztunk-e a pinekhez. Ezen kívül be kell állítani a protokoll verzióját és sebességét. Ha ezeket az előkészületeket elvégeztük jöhet a mérési elrendezés specifikálása.

Ennek a konfigurációnak a mérési felépítése teljesen általános, megegyezik a program ismertetésénél bemutatottal. A szimuláció felépítése egy Replay blokkot tartalmaz.



21. ábra: Az első részfeladat szimulációjának felépítése

A visszajátszott adatforgalom a vizsgálópadon korábban felvett kommunikáció.

Ha megvan a hálózaton előforduló valamennyi üzenettípus adatbázisa, akkor ezt a méréshez rendelve a program automatikusan felismeri a buszon megjelenő üzeneteket. Ez a szolgáltatás nagyon megkönnyíti az üzenetek kielemzését és nemcsak az üzenetekről szolgál információval, a hálózat egész felépítését leírja. Meghatározza egyebek mellett a node-ok közötti kapcsolatot, a hálózat sebességét és az időzítéseket. Rendelkeztem a Komfort CAN-hez tartozó K-Mátrixal. Ezt a dfc típusú fájl a programhoz rendelve a software felépítette a virtuális hálózatot. Ezt szimbolizálja a Netzwerk CAN1 „blokk". Ehhez lehet csatlakoztatni a valódi node-okat. Mérési összeállításomban, ahogy már a kezdeti rendszertervben is megjelent a klímavezérlő szerepel reális csomópontként.

A gateway-re ebben az esetben már nincs szükség. Az iDEX programból a klímavezérlő már közvetlenül a Komfort-CAN-en keresztül elérhető. Ez azért fontos, mert az iDEX programot használják a hibatárak kiolvasásához, hiányában nem lenne kényelmes lehetőség erre a vizsgálatra.

A mérés összeállt, kérdés már csak az volt, hogy hogyan reagál a klímavezérlő a vizsgálópadon felvett kommunikációra. A tapasztalatok azt mutatták, hogy a kontroller a szimulációnak köszönhetően minden szükséges jelet megkapva rendeltetésszerűen, hibajelzés nélkül működött. A működést az sem befolyásolta, hogy a kommunikáció egy körülbelül húsz másodperces felvétel loopban történő ismétlése volt. A vezérlő hibatárai olvashatók voltak, új hiba nem jelent meg. A LIN Master funkciót is kiválóan betöltötte, hiszem a kezelőegységet tekerve a kijelzett hőmérsékletérték változott, a klímavezérlő reagált a külső beavatkozásra.

Ezzel a CAN szimulációval és a diagnosztikai hozzáférési pontok kialakításával a vezérlő „sziget üzemben" is életre kelthető és diagnosztizálható. A perifériáit a visszajátszott CAN adatforgalomnak megfelelően kivezérli, pontosan úgy, mint valós környezetében, a gépjárműben.

5.4 LIN Slave élesztése és tesztelése szimulált hálózaton keresztül

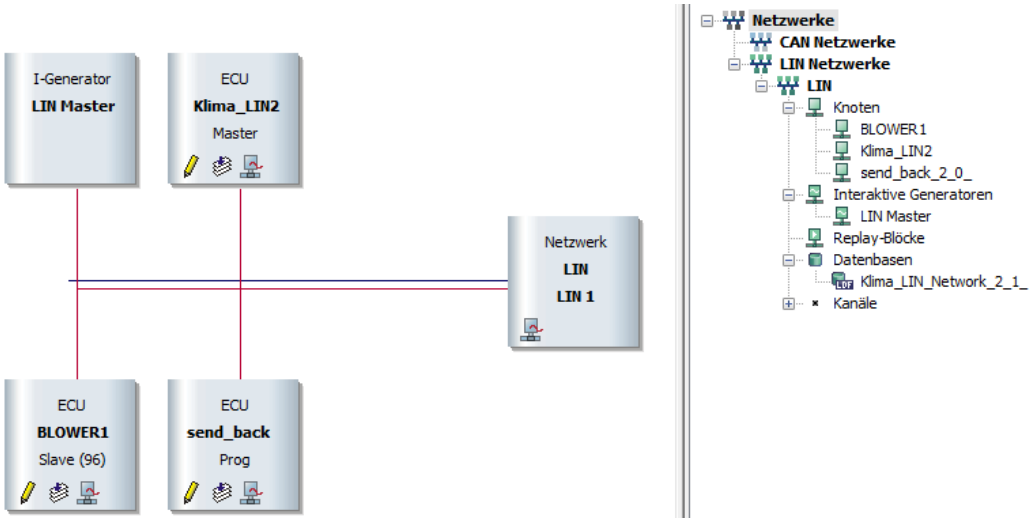
Mint utóbb kiderült a második elvárás megvalósítása bizonyult összetettebbnek. A LIN hálózat szimulációjára először egy új kevert CAN-LIN konfigurációt hoztam

létre. Abban bíztam, hogy itt is a megfelelő K-Mátrix adatbázisokat a programba illesztve vissza tudom fejteni a kommunikációt. Elsődleges célom azoknak a CAN vagy LIN üzeneteknek a megtalálása volt, melyek értékének változtatásával a tekerőgomb kijelzőjén változást lehet elérni. Ezzel kapcsolatos CAN üzeneteket egyáltalán nem találtam, ebből arra következtettem, hogy a kijelző értékét változtató jelek kizárólag a LIN kommunikációra korlátozódnak, csak azon a hálózaton jelennek meg. A LIN hálózathoz viszont nem volt K-Mátrix adatbázis. A Trace ablakban megjelenő LIN üzeneteknek csak az ID-ja és az adatmező volt látható. Nem lehetett tudni, hogy melyik résztvevő küldi őket, és kinek szólnak. Ezért elkezdtem egyesével figyelni az LIN üzenetek reakcióját a gomb elforgatásának hatására. Ezzel az intuitív technikával sikerült összefüggést találnom két üzenet között. További vizsgálatokkal megállapítottam, hogy az egyik üzenet a gomb elforgatásával kapcsolatos. Ezt a Master header-jére válaszolva a Slave küldi, ami ez esetben a tekerő gomb. Az üzenet egyik 1 bájtos szignáljának értéke jobbra tekerés hatására nőtt, balra tekerés hatására pedig csökkent. A másik üzenet a kijelzésért felelt. A Master az előző üzenet értékéből kiszámolva visszaküldte azt az értéket, amit a Slave-nek a kijelzőn meg kell jelenítenie. Ez is egy 1 bájtos szignál volt értéke 0x00 és 0x1A között változott. Mint később kiderült 0x1A-nál nagyobb érték is definiált. A hőmérséklet értéke ilyenkor °C helyett °F-ban kerül kijelzésre. Miután megvoltak a kijelzést érintő üzenetek elkezdhettem azon dolgozni, hogy befolyásolásukkal programból lehessen beállítani a kijelzett értéket.

Már szükségtelen volt a CAN hálózat megfigyelése, ezért egy kizárólag LIN típusú konfigurációt hoztam létre, amibe később a végleges implementáció is került. Mint korábban említettem a CANoe program lehetőséget ad tetszőleges üzenetek buszra küldésére. Ehhez vagy interaktív generátorblokk vagy CAPL script használható. A LIN adatbázist tartalmazó fájlt (LDF – Lin Description File) ez esetben nem állt rendelkezésemre, ezért azt először létre kellett hoznom a kutatásom során szerzett információkból. Az LDF fájl létrehozásában a CANoe egyik kiegészítő programja az LDF Editor volt segítségemre. A programmal az LDF szerkeszthető kód formában vagy párbeszédablakokon keresztül. Utóbbiak alkalmazása esetén az üzenetek bármelyik paramétere áttekinthető formában megjelenítésre kerül és módosítható. Megadható az üzenetek azonosítója, hossza és tartalma, sőt még a küldéssel kapcsolatos időzítések is beállíthatók.

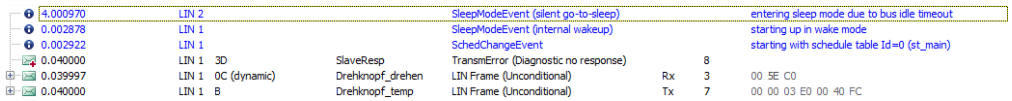
Az LDF létrehozása a hálózat szereplőinek definiálásával kezdődött. A szimulált klímavezérlő a hálózat Mastere, a tekerőgomb pedig a kommunikációban résztvevő egyetlen Slave. Ez után a vizsgálatokból ismert 0x0B illetve 0x0C azonosítóval rendelkező két jel meghatározása következett. Megadtam a jelekhez tartozó szignálokat a korábbi kutatásaim alapján. Ezen kívül megneveztem az üzeneteket publikáló illetve fogadó résztvevőket. A 0x0C üzenetet a Slave küldi a Master felé, azaz a Slave a tekerés mértékét meghatározó adattal válaszol a Master header-jére. A 0x0B ID-val rendelkező üzenet esetén pedig a Master publikálja a keret adatmezőjét is, hiszen ez tartalmazza azt a számított értéket, amit a kijelző egységnek meg kell jelenítenie. Az üzentek kiküldésének időzítését a schedule table határozza meg. Itt mindkét üzenet esetében 20 ms-ot állítottam be, ilyen gyakorisággal kerülnek az üzenetek a buszra. Ez azt a sebességet jelenti, amit a klímavezérlő még könnyűszerrel feldolgoz, és amivel a tekerésre való reagálás és a kijelzés is az elvártaknak megfelelően frissül.

Miután mindent elvégeztem a következő szimulációs felépítést kaptam:



22. ábra: LIN Slave szimuláció felépítése

A node-ok közül a klímavezérlő, mint LIN Master (Klima_LIN2) és a tekerőgomb, mint LIN Slave (Blower1) maradt. A klímavezérlő ebben az összeállításban már csak virtuálisan van jelen. Az általa ciklikusan kiküldendő headerek közlését már a hálózatba általam illesztett interaktív generátorblokk végzi. A 24. ábrán látható, hogy az elrendezésben még egy utólag beillesztett ECU szerepel. Ez nem más, mint az a CAPL script, amivel a kijelzett szimbólum programból módosítható. Ennek részletes bemutatása később következik. Az így létrehozott konfigurációval már sikerült a frame-eket a buszra juttatnom. A Trace ablakban a következő üzenetek jelentek meg:



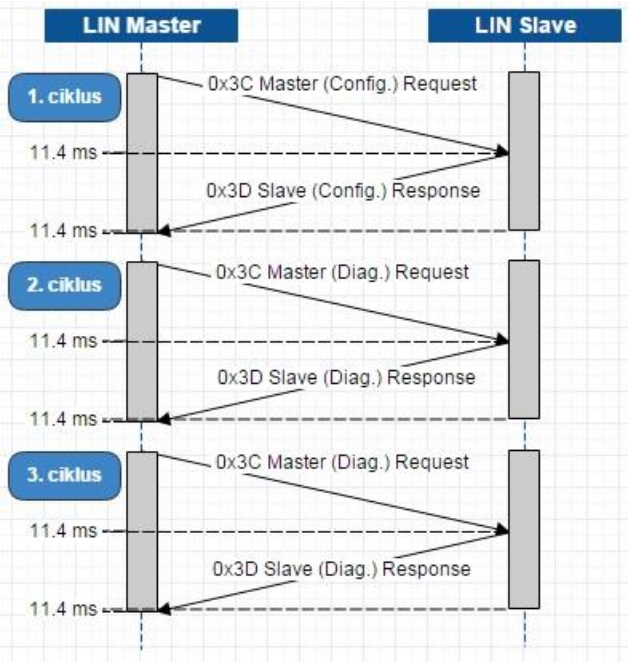
23. ábra: LIN szimuláció Trace ablaka

Mint már korábban említettem a 0x0C ID-val rendelkező üzenet hordozza a gomb eltekeréséről szóló információt, míg a 0x0B ID-val rendelkező a kijelzett értéket határozza meg. Látszik, hogy szerepel egy 0x3D azonosítóval rendelkező diagnosztikai üzenet, ennek a későbbiekben, a szegmensteszttel kapcsolatban lesz jelentősége.

A kijelző értékét már módosítani tudtam, de még nem voltam tisztában azzal, hogy a két megismert üzenet egyes szignáljai milyen funkciókért felelnek még, hiszen az üzenetek tartalmát az LDF-ben a megfigyeléseimre hagyatkozva állítottam be. Munkám következő szakaszaként ezen megfigyelések kiterjesztése, a többi szignál feltérképezése következett. Alapvető elvárás volt a feladat megfogalmazásakor a kijelzett hőmérsékletérték változtatásának lehetősége, de ezen kívül más követelmények is megfogalmazódtak. Szerettem volna elérni a kijelző fényerejének módosítását és egy szegmensteszt kivitelezését, amivel a kijelző összes szegmense egyszerre villantható fel. A Dimmung, azaz fényerő beállításához kapcsolódó szignált könnyedén megtaláltam annak az üzenetnek a részeként, amely a kijelzendő értéket is hordozta, viszont a szegmensteszt kivitelezése összetettebbnek bizonyult. Mivel ez egy egyébként iDEX-ből indítható vizsgálat, vissza kellett térnem a korábbi, valódi klímavezérlőt és gateway-t tartalmazó konfigurációhoz. Megint a vizsgálópadot használtam. Az iDEX-ből kiadtam a tesztet indító parancsot, miközben rögzítettem a busz adatforgalmát. A felvétel megfigyelése után markáns változást ismét csak a LIN üzeneteket érintően tapasztaltam. Itt azonban már nem csak két üzenet egy-egy szignálja változott, hanem egy egész üzenet szekvencia futott le az alábbi ábrán látható módon.

24. ábra: Szegmenstesztet indító üzenet szekvencia

Az üzenetváltások a hálózat diagnosztikai frame-jeit érintették. A szekvencia egy Master kéréssel (Request) indult, amit a Slave válasza követett. A teszt három ilyen üzenetpár lefutása hatására indult:



25. ábra: Szegmensteszt üzenetküldési szekvenciája

A kommunikációt CAPL-ben leprogramozva nekem is sikerült a szegmensteszt indítása. A vizsgálat kikapcsolásához még egy plusz üzenet elküldésére volt szükség, ennek megvalósítási módját a CAPL program ismertetésekor részletezem.

26. ábra: A tekerőgomb az aktív szegmensteszt állapotában

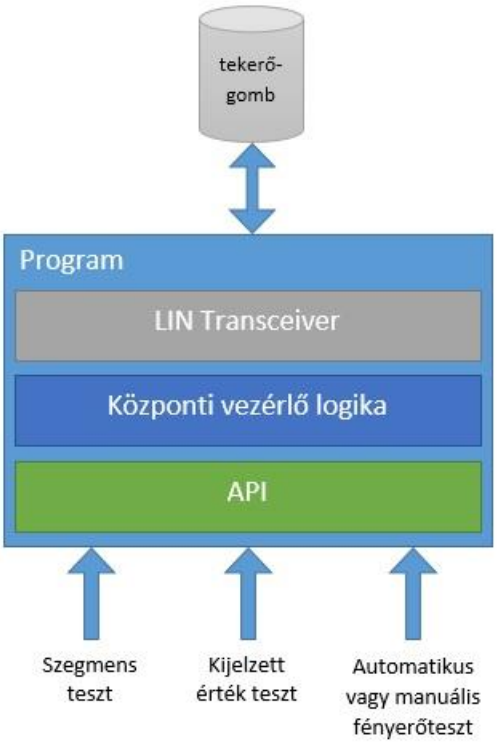
6 Felhasználói felület tervezése és implementálása a CANoe programba

Az előző fejezetben említettem, hogy a LIN Slave környezetét megvalósító szimulációban helyet kapott egy virtuális ECU. A látszólagos vezérlő szerepét egy CAPL program töltötte be. A kódban határoztam meg az interaktív generátorblokk által buszra küldött üzenetek tartalmát és innen indítottam a szegmenstesztet is. Ebben a szakaszban a program és a hozzá kapcsolódó felhasználói környezet felépítésének ismertetése következik.

6.1 A CAPL program

A CAPL bemutatása alkalmával utaltam arra, hogy egy eseményvezérelt nyelvről van szó. Egyéb tulajdonságait tekintve a programnyelv szinte megegyezik a C-vel. A Függelékekben mellékelt programkód, ezért nem meglepő módon olyan blokkokból áll, melyek egy-egy bizonyos esemény bekövetkeztekor futnak le. Egy blokk szintaktikája az on <esemény> {}. A programot öt ilyen blokkra lehet osztani. Ebből három egy-egy timerrel kapcsolatos a negyedik egy üzenet megjelenésével az ötödik pedig a program indításával.

A programkód az include-okkal (nekem erre nem volt szükségem) és a változók deklarálásával és/vagy definiálásával kezdődik. Fontos tudni, hogy a CANoe-ban lehetőség van rendszerváltozók használatára. Ezeket a CANoe és annak összes segédprogramja, így a CAPL program editora is „látja”. A rendszerváltozók a konfigurációban tárolódnak. Az általam elkészített program öt ilyen rendszerváltozót és számos lokális változót használ. A rendszerváltozókkal az éppen aktuálisan folyó tesztről kap a program visszajelzést (Dimmung – képernyő fényereje, zahlen – futtatása esetén a hőmérsékletérték folyamatosan nő, Segment - szegmenstesztthez), illetve a felhasználói környezet számára szolgálnak információval (i – fényerősség értéke, j – aktuális hőmérsékletérték). Lokális változók a korábban említett timer típusok. Ezeken kívül három üzenettípus is definiálásra került. A két kijelzéssel kapcsolatos illetve a szegmensteszt indításáért felelős frame típus. Ezekről az előző fejezetben már volt szó. A többi segédváltozó használatát a program logikája követeli meg. Az alábbi ábrán látható a program logikai felépítése.



27. ábra: Program logikai felépítése

A program három fő egységre osztható. Ezek közül az egyik feladata a LIN kommunikációval kapcsolatos események illetve ki és bemenetek kezelése. Különböző inputok (teszt indítás, kijelzett hőmérséklet érték változtatás, fényerő módosítás) érkezhetnek a felhasználó felől is, az ezekkel kapcsolatos műveleteket egy API végzi el. A két programblokkot a központi vezérlő logika kapcsolja össze a szükséges számítások elvégzésével és a megfelelő változók módosításával. Nézzük végig a korábban említés szintjén már ismertetett egyes blokkok feladatát!

on linMessage 0x0C

Ez a blokk 0x0C azonosítóval rendelkező üzenet érkezésekor fut le. A vizsgált frame a gomb eltekerésekor jelenik meg a buszon. A blokkban a program megvizsgálja, hogy mekkora mértékű volt az eltekerés és előkészíti a kijelzésért felelő 0x0B ID-jú üzenetet (msgTemperatur) a frissített adat kiküldésére.

on timer tmrZyklus

Az első timert a számláló és fényerő tesztek során használja a program. Gyakorlatilag egy for ciklus kiváltása a feladata. Kommunikációs buszok esetén ez a megoldás elegánsabb, a ciklusok megvalósítása bonyolultabb és nagyobb körültekintést

igényel. A timer blokk az i és j változók értékét növeli és beállítja saját maga várakozási idejét addig, amíg a kiválasztott tesztek futnak.

on timer tmrZyklus2

Fényerő és számláló tesztek során ebben a blokkban történik a kijelzett értéket tartalmazó 0x0B ID-jú üzenet buszra küldése. A timer értékével állítható be, hogy milyen időközönként frissítsen a rendszer.

on timer tmrZyklus3

A szegmensteszt összetettsége miatt megvalósításához külön blokk definiálására volt szükség. Ez a programrészlet küldi ki a szegmestesztet elindító Master headereket. Figyeli a Slave válaszát, és ha az az elvártnak megfelelő folyatatja a kérések küldését, amíg a szegmensteszt el nem indul. Ha a Segment rendszerváltozó hamissá válik, egy lezáró Master üzent kiküldésével kikapcsolja a vizsgálatot.

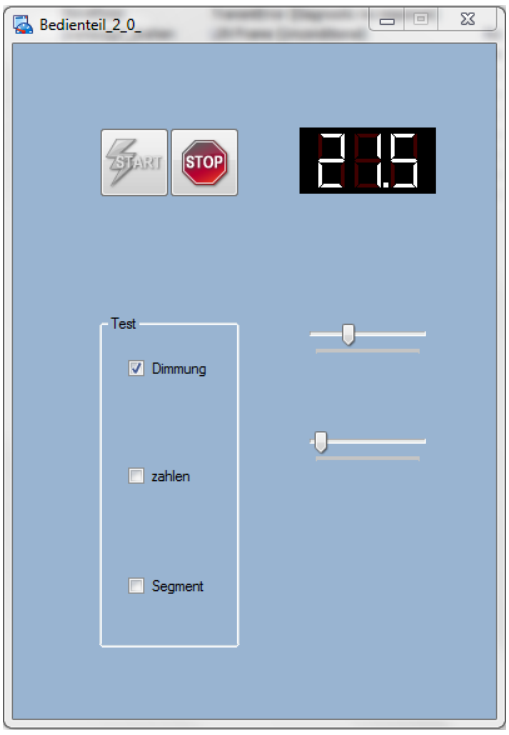
on start

A tmrZyklus3 timer indításárért felel.

Ez a program működésének összefoglalása, most a felhasználói környezet bemutatása következik.

6.2 A felhasználói környezet

A CANoe Panel Editor segédprogramjával a megvalósított konfigurációhoz lehetséges egy API létrehozása. A LIN Slave szimulációjához készített felhasználói interface látható az alábbi ábrán.



28. ábra: A szimulációhoz készített API

Az ablak bal felső sarkában a mérést indító és leállító gombok találhatók. A jobb felső sarokban található kijelzőn látható az éppen aktuálisan beállított hőmérsékletérték. Ezt a zahlen teszt indításával vagy a második csúszka mozgatásával lehet változtatni. Ilyenkor természetesen a valós tekerőgomb kijelzőjén is módosul a kijelzett érték. Hasonló elven működik a fényerő befolyásolása. Itt is vagy a Dimmung teszt futtatásának hatására vagy a csúszka mozgatására változik a képernyő fényessége. A Dimmung és zahlen teszt egyszerre futtatható, a szegmensteszt esetén azonban nincs lehetőség párhuzamos vizsgálatok végzésére. A Segment mező bepipálására a valós tekerőgomb összes szegmense kigyullad. Ez az állapot mindaddig fennmarad, míg a felhasználó ki nem veszi a jelölőt a mezőből.

Mint az sejthető a Test panelben szereplő három mező a három rendszerváltozó értékét módosítja 0 vagy 1 értékre. Az i és j rendszerváltozók a csúszkákhoz vannak rendelve. A kijelző értékét a j rendszerváltozó határozza meg.

6.3 A kész környezet és alkalmazási területeinek bemutatása

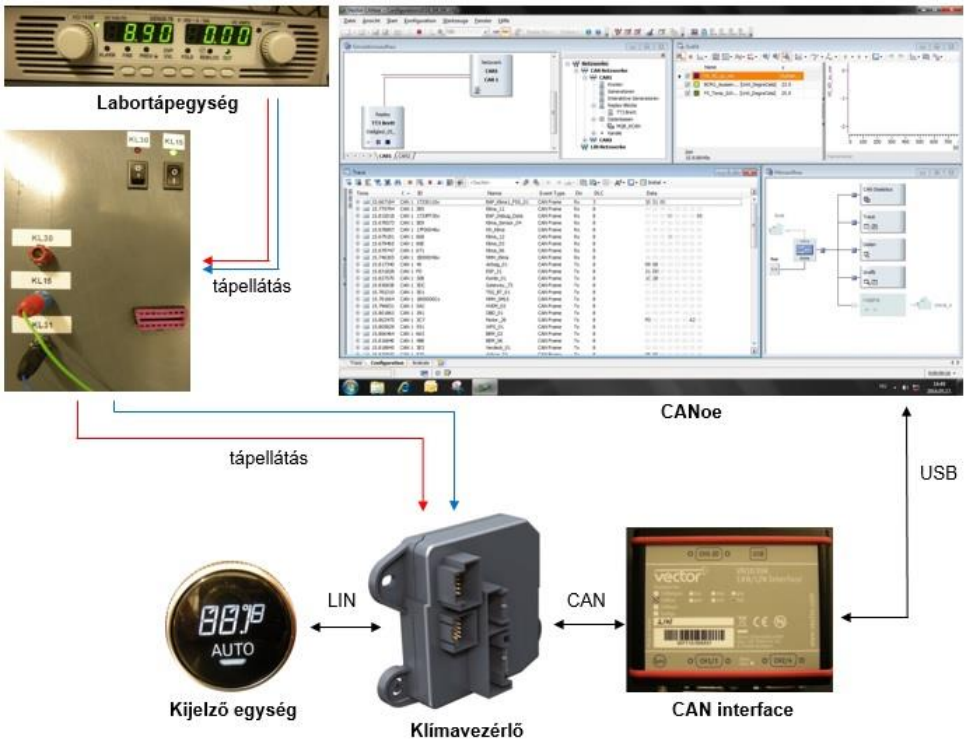
A megvalósított mérőállomás tehát egy CAN-es és egy LIN-es tesztkörnyezetet is magában foglal. Az alábbiakban röviden összefoglalom az egyes tesztkörnyezetekkel elérhető vizsgálatokat és szemléltetem a kész mérőállomás működési logikáját.

6.3.1 CAN szimuláció

Az elkészült mérőállomás egy gépjármű elektromos hálózatát szimulálja a klímavezérlő számára. Ebben a szimulációs környezetben a klímavezérlő önállóan, perifériái nélkül tesztelhető. Mindez úgy történik meg, hogy a vizsgálat a kontroller kezdeti állapotát nem módosítja, hibatárolójában nem keletkeznek újabb bejegyzések, köszönhetően a vizsgáló berendezésen beállított tápfeszültségnek. A tápfeszültség megválasztása a szakdolgozat mérési eredményei alapján történik, melyek szerint a vezérlőt 8.9 V-on kell működtetni a szimulációs környezetben végzett sikeres diagnosztikához.

A szakdolgozat eredménye továbbá, hogy a klímavezérlőt a mérőpadhoz csatlakoztatva egyéb alkatrész (gateway) közbeiktatása nélkül lehetőség nyílik

- a kontroller diagnosztikájára a vállalatnál használt PC-s környezetből
- a mérési blokkok kiolvasására és
- a hibatárak kiolvasására és törlésére.



29. ábra: A megvalósított CAN-es tesztkörnyezet működés közben

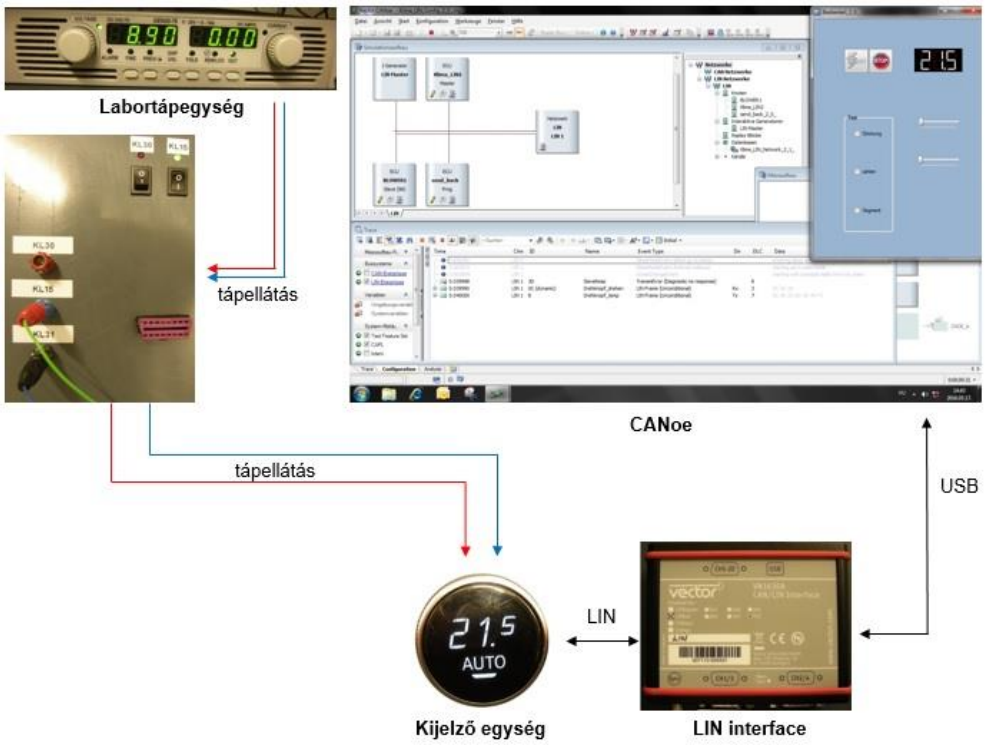
6.3.2 LIN szimuláció

A mérőállomással, a szakdolgozat részeként megvalósított második konfiguráció során a klímavezérlő szimulációja történik az egyik kijelző és beavatkozó egység számára. A kezelő berendezés a belső tér hőmérsékletét állító tekerőgomb. A megvalósított környezettel ugyanazok a funkciók vizsgálhatók a felhasználó számára, mint amik egy autóba épített alkatrészen is működnek.

A konfigurációhoz kapcsolódik egy interaktív software-es alkalmazás, amiből a felhasználó manuálisan vagy automatikus teszt formájában változtathatja

- a kijelzett hőmérsékletértéket
- a fényerő értékét és
- speciális diagnosztikai tesztet, egy szegmensteszt is futtathat.

Utóbbi vizsgálat során a kijelző valamennyi pixele aktívvá válik, így kiszűrhetők a kijelzést érintő hibák.



30. ábra: A megvalósított LIN-es tesztkörnyezet működés közben

A mérőállomás tehermentesíti az Elektromos és Elektronikus Laboratórium vizsgálópadját és megkönnyíti az analízismérnökök munkáját.

7 Az eredmények összegzése és ötletek a továbbfejlesztéshez

A záró fejezetben egy kis ismertető kapott helyet az autóipari hálózatokban rejlő jövőbeli lehetőségek bemutatására. Majd a szakasz és a szakdolgozat befejezéseként összegzem az elért eredményeket és rávilágítok a témában rejlő további kihívásokra.

7.1 Nyitás a jövő felé

A bevezető fejezetekben a modern autók elektromos rendszerének legkisebb építőelemétől eljutottam a gépkocsi, mint feladat specifikus beágyazott rendszereket összekötő hálózat bemutatásáig. A lehetséges komplexitási fok, mint ahogy szakdolgozatom eredményeiből is kitűnik, nem áll meg ezen a szinten. Egy prémium termékeket gyártó vállalat esetén kötelező elvárás a folyamatos innováció. A jövőbe mutató törekvések számos irányt vesznek. Még nem tudhatjuk, pontosan melyek lesznek azok, amelyek valóban áttörést hoznak a gépjárművekkel kapcsolatos felfogásunkban. Vannak azonban megoldások, amelyek jelen pillanatban megszabják a fejlődési folyamatot, most ezeknek a rövid ismertetése következik.

A modern autóipari trendek fontos kulcsszavakra építenek. Ilyenek a biztonság, a környezettudatosság és kényelem. Az olyan megoldások, mint például az ESP, az ABS vagy az egyre fejlettebb légzsák rendszerek már több évtizeddel ezelőtt a minél biztonságosabb közlekedés szolgálatába állították az elektronikát. A nagy adatmennyiségek gyors feldolgozásának lehetőségével új utak nyíltak ADAS (advanced driver assistance systems) rendszerek fejlesztéséhez. Ma már számos példa létezik ezeknek a rendszereknek az aktív jelenlétére. Vegyük például az autópályán haladást. Az alkalmazkodó sebesség- és távolságtartó automatika (ACC – adaptive cruise control) már számos autóban meglévő szolgáltatás. Segítségével beállítható a kívánt utazósebesség, viszont ha az előttünk közlekedők lassítanak vagy fékeznek, a rendszer ezt egy radar segítségével érzékeli és automatikusan hozzáigazítja a sebességet az előttünk haladókhoz. Külön software és elektronika gondoskodik arról, hogy ne hagyjuk el a sávot, vagy ha el akarjuk hagyni tartózkodik-e valami a holttérben, illetve azt is figyeli, hogy elég éberen vezetünk-e. Éjszaka intelligens fényszórók gondoskodnak arról, hogy kanyarban is jól lehessen látni az utat és ma már a szembe

jövők elvakításának problémája is megoldott. A járműben dolgozó elektronika nem csak az utasokat igyekszik megvédeni. Az ütközés előtti vagy ütközést megelőző (precrash) rendszerek a balesetek elkerüléséért vagy enyhítéséért felelnek. Radarokkal és kamerákkal monitorozzák a közlekedési szituációt és vészhelyzetben be is avatkozhatnak fékezéssel vagy a kormány szögének módosításával. Olyan szolgáltatás is létezik, amikor egy kiválasztott parkolóhelyre az autó önállóan beáll helyettünk. Ezeket a megoldásokat egytől egyig az autó érzékelői és vezérlői teszik lehetővé.

Ahhoz, hogy a fent felsorolt alkalmazások még hatékonyabban működhessenek a közlekedés résztvevőinek hálózatba kapcsolása a következő lépés. GPS-es navigációt és internetet már ma is találunk a gépkocsikban, az autók egymással való kommunikációja (C2C – car to car) illetve a jármű és környezete közti információcsere (V2I – Vehicle to infrastructure) azonban még nincs kiépítve. Ezekkel a megoldásokkal még biztonságosabbá és hatékonyabbá válhatna a közlekedés. Gondoljunk csak bele, mennyivel egyszerűbb lenne a parkolóhely keresés, ha a hálózatról az autó megkapná a szabad helyek listáját. Ha tudnánk, hogy merre érdemes menni, melyik irányban várható zöldhullám, vagy információ állna rendelkezésre a forgalmi fennakadások elkerülésének módjáról. Az autó hiba érzékelése esetén, automatikusan bejelentkezne a legközelebbi szervibe és számos ehhez hasonló előny származna abból, ha létezne egy ilyen hálózat. Innen már csak egy lépés a prototípus szinten már ma is működő autonóm közlekedés, mely a vezérlők még nagyobb számítási kapacitását követeli meg.

A második kulcsszó, amit korábban kiemeltem, a környezettudatosság. A fosszilis energiahordozók csökkenésével muszáj alternatív üzemanyagot alkalmazni a gépjárművekben. Viszont ezzel az újfajta üzemanyaggal szemben az is komoly elvárás, hogy környezetbarát legyen. A klímaváltozás korunk egyik legnagyobb problémája. Az üvegházhatású gázok kibocsátásának csökkentése alapvető követelmény ahhoz, hogy a folyamatot megakadályozzuk. Ezért az újfajta erőforrások esetén a zéró emisszió a kitűzött cél. Ezt az elvárást az elektromos hajtás képes leghatékonyabban teljesíteni. A jövőben az elektromos autók számának jelentős növekedése várható, ami újabb kihívásokat támaszt a gépjárművek elektromos hálózatával szemben, hiszen teljesen új vezérlésre kell optimalizálni őket.

7.2 Összegzés és továbbfejlesztési lehetőségek

A kapott eredményeket tekintve kijelenthetem, hogy a szakdolgozat feladatkiírásában megfogalmazott követelményeket teljesítettem, a kitűzött munkát elvégeztem. Sikerült egy olyan tesztállomás megvalósítása, aminek segítségével az Audi TT klímavezérlője és a hozzá tartozó tekerőgomb gépjárműtől és az Elektromos és Elektronikus Laboratóriumban található próbapadtól függetlenül analizálható. A kidolgozott folyamat lépéseit követve a mérőállomás univerzálissá, más vezérlők vizsgálatára is alkalmassá tehető. Ezt, mint továbbfejlesztési irányt számba is lehet venni. Végső cél lehet egy olyan univerzális mérőberendezés kialakítása, amelyen a különböző elektronikai egységek Plug & Play módon vizsgálhatók.

Irodalomjegyzék

[1] ASAM: *MCD-2 D* https://wiki.asam.net/display/STANDARDS/ASAM+MCD-2+D, (2016. máj.)

[2] AutomotiveIQ: *Automotive Diagnostic Systems: From OBD to Open Diagnostics Exchange format (ODX)* https://automotiveiq.wordpress.com/2011/06/03/automotive-diagnostic-systems-from-obd-to-open-diagnostics-exchange-format-odx/, (2016. máj.)

[3] Brucklacher, U.: *iDEX Programm und Funktionsbeschreibung*, 2008.

[4] Dr. Bécsi, T.: *Járműipari hálózatok* BME, Közlekedési- és Járműirányítási Tanszék, 2012.

[5] Dr. Fodor, D.: *Autóipari kommunikációs protokollok - a CAN*, Pannon Egyetem, 2012.

[6] Dr. Fodor, D., Dr. Szalay, Z.: *Autóipari kommunikációs rendszerek*, Pannon Egyetem, 2014.

[7] EPA: *OBD Basic Informations*, https://www3.epa.gov/obd/basic.htm, (2016. máj.)

[8] ISO: *Road vehicles -- Open diagnostic data exchange (ODX) -- Part 1: Data model specification, Abstract,* www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41207, (2016. máj.)

[9] MOST Cooperation: *MOST Network*, http://www.mostcooperation.com/technology/most-network/, (2016. máj.)

[10] SparkFun Electronics: *Getting Started with OBD-II.*, https://learn.sparkfun.com/tutorials/getting-started-with-obd-ii (2016. máj.)

[11] Vector Informatik GmbH: *Media Oriented Systems Transport (MOST),* http://vector.com/vi_most_en.html (2016. máj.)

[12] Vector Academy: *CANoe Grundlagen Workshop*, 2012

Függelék

CAPL programkód

```
/*@!Encoding:1250*/
includes
{

}

variables
{
  int voriges, aktual = 0x0;
  int drehZhl;

  linMessage 0xB msgTemperatur;

  int cntrSeg = 1;
  linMessage 0x3C msgStellglied;
  linMessage 0x3D msgStellgliedHeader;

  msTimer tmrZyklus, tmrZyklus2, tmrZyklus3;
}

on linMessage 0x0C {  //ha kézzel tekerjük: figyeli, hogy melyik irány és
mennyi a különbség az előző értékhez képest
      if (voriges < this.Dreh_dir)
      {
        drehZhl = this.Dreh_dir - voriges;
        aktual = aktual + drehZhl;
      }

      else
      {
        drehZhl = voriges - this.Dreh_dir;
        aktual = aktual - drehZhl;
      }

    voriges = this.Dreh_dir;
}

on timer tmrZyklus
{
    if (@Testing::Dimmung)
    {
      ++@Cntr::i;
    }

    if (@Testing::zahlen)
    {
      ++@Cntr::j;
    }

    setTimer (tmrZyklus, 100);
}
```

```
on timer tmrZyklus2
{
  if(!(@Testing::Segment))
  {
    if (@Testing::zahlen)
    {
        msgTemperatur.byte(0) = @Cntr::j;
    }

    else
    {
      msgTemperatur.byte(0) = aktual + @Cntr::j;
    }

    msgTemperatur.byte(4) = @Cntr::i;

    msgTemperatur.byte(1) = 0x00;
    msgTemperatur.byte(2) = 0x03;
    msgTemperatur.byte(3) = 0xE0;
    msgTemperatur.byte(5) = 0x40;
    msgTemperatur.byte(6) = 0xFC;
  }

  if((aktual + @Cntr::j) < 26)
  {
    @disp = 15.5 + 0.5*(aktual + @Cntr::j);
  }

  else if (((aktual + @Cntr::j) > 27) && ((aktual + @Cntr::j) < 53))
  {
    @disp = 32 + aktual + @Cntr::j;
  }

  else
  {
    @disp = 28;
  }

  output (msgTemperatur);
  setTimer (tmrZyklus2, 20);
}

on timer tmrZyklus3 //0x3c üzenet küldése
{
    if (@Testing::Segment)
    {
      switch (cntrSeg)
      {
        case 1:
          msgStellglied.MasterReqB0 = 0x01;        //01 06 B0 42 00 0F 00
A0
          msgStellglied.MasterReqB1 = 0x06;
          msgStellglied.MasterReqB2 = 0xB0;
          msgStellglied.MasterReqB3 = 0x42;
          msgStellglied.MasterReqB4 = 0x00;
          msgStellglied.MasterReqB5 = 0x0F;
          msgStellglied.MasterReqB6 = 0x00;
          msgStellglied.MasterReqB7 = 0xA0;
```

```
          ++cntrSeg;

          output(msgStellglied);

          setTimer (tmrZyklus3, 200);
          break;

        case 2:
          msgStellglied.MasterReqB0 = 0xA0;        //A0 01 63 FF FF FF FF
FF
          msgStellglied.MasterReqB1 = 0x01;
          msgStellglied.MasterReqB2 = 0x63;
          msgStellglied.MasterReqB3 = 0xFF;
          msgStellglied.MasterReqB4 = 0xFF;
          msgStellglied.MasterReqB5 = 0xFF;
          msgStellglied.MasterReqB6 = 0xFF;
          msgStellglied.MasterReqB7 = 0xFF;

          ++cntrSeg;

          output(msgStellglied);

          setTimer (tmrZyklus3, 200);
          break;

        case 3:
          msgStellglied.MasterReqB0 = 0xA0;        //A0 02 08 01 FF FF FF
FF
          msgStellglied.MasterReqB1 = 0x02;
          msgStellglied.MasterReqB2 = 0x08;
          msgStellglied.MasterReqB3 = 0x01;
          msgStellglied.MasterReqB4 = 0xFF;
          msgStellglied.MasterReqB5 = 0xFF;
          msgStellglied.MasterReqB6 = 0xFF;
          msgStellglied.MasterReqB7 = 0xFF;

          ++cntrSeg;

          output(msgStellglied);

          setTimer (tmrZyklus3, 200);
          break;
        }
      }
    else
    {
      msgStellglied.MasterReqB3 = 0x00;
      output(msgStellglied);
    }
}

on start
{
  setTimer (tmrZyklus, 0);
  setTimer (tmrZyklus2, 0);
}
```