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

Takács József

CAN RÉSZLEGES HÁLÓZATI ESZKÖZ FEJLESZTÉSE

Szakdolgozat

BELSŐ KONZULENS

Dr. Sujbert László

BME MIT

KÜLSŐ KONZULENS

Dr. Balogh András

ThyssenKrupp Presta Hungary Kft.

BUDAPEST, 2014

MŰEGYETEM 1782

SZAKDOLGOZAT-FELADAT

Takács József (YV22HU)
szigorló villamosmérnök hallgató részére

CAN részleges hálózati eszköz fejlesztése

A modern gépjárművek biztonságtechnikai és kényelmi funkcióinak megvalósításában, környezetvédelmi jellemzőinek javításában stb. egyre jelentősebb szerepet kapnak a számítástechnikai megoldások. Ma egy prémium személyautó gyártójának közel száz elektronikus vezérlőegységből (ECU) és számos fedélzeti kommunikációs sínből kell kialakítani egy megbízhatóan működő elosztott rendszert, amely komoly algoritmus- és kommunikációtervezési, illetve munkaszervezési kihívást jelent. Az így adódó komplexitás uralására alakultak ki különféle szabványok, pl. a megbízható kommunikáció biztosítására a CAN és FlexRay sínek, utóbbi időben pedig ezek kiterjesztéseként a részleges hálózatok létrehozását (partial network), azaz a terepbuszok részbeni lekapcsolását lehetővé tévő szabványok.

A vezérlőegységek fejlesztése nem a járműbe integráltan történik, és a hálózaton jelen levő többi egység nem áll rendelkezésre, ezért nehézkes a hálózati kommunikáció ellenőrzése. Cégünknél korábban elkészült egy terepbusz illesztő (gateway) egység, mely a fejlesztői PC és a vezérlőegység között teremt kapcsolatot. Jelen feladat célja a meglevő hardver kiterjesztése részleges hálózati funkciókkal. A hallgató feladata magában foglalja az alábbiakat:

- TKP Fieldbus Gateway nevű Ethernet-terepbusz illesztő hardver megismerése
- A CAN Partial Networking szabványok megismerése
- Egy kiválasztott CAN PN Transceiver illesztése a GW eszközhöz
- A GW-en a transceiverhez egy eszközmeghajtó készítése
- A PC oldali GW API kiegészítése CAN PN funkciókkal
- Egy egyszerű PC-s demonstrátor alkalmazás készítése, amely képes CAN PN hálózatot kezelni (keretek küldése, GW elaltatása, felébresztése)

A feladat megvalósításához szükséges eszközöket a ThyssenKrupp Presta Hungary Kft. biztosítja.

Tanszéki konzulens: Dr. Sujbert László docens
Külső konzulens: Dr. Balogh András (ThyssenKrupp Presta Hungary Kft.)

Budapest, 2014. szeptember 18.

.....
Dr. Jobbágy Ákos egyetemi tanár
tanszékvezető

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2014. 12. 12.

...
Takács József

Kivonat

Szakdolgozatom témája a ThyssenKrupp Presta Hungary Kft. által kifejlesztett terepbuszillesztő egység (gateway) funkcióinak kiterjesztése a CAN részleges hálózatok kezelésével. Ez a továbbfejlesztés teljes körűen integrálódik a már meglévő gateway egységbe, mind hardveres, mind szoftveres szinteken.

Munkám során először ismertetem a CAN részleges hálózatok megalkotásának mozgatórugóit, működését, valamint az ennek alapjául szolgáló CAN protokollt is.

Majd a továbbfejlesztendő terepbuszillesztő egységet mutatom be, kitérve a felhasználására, valamint alapvető működésére.

Ezek után időbeli sorrendben ismertetem a feladatom megvalósításának fázisait, a hardverrel kezdve. E fejezetben végigkövetem a hardver megvalósításának lépéseit, a tervezési követelményektől kezdve a gyártásig.

A hardver ismertetése után a szoftveres feladatokat veszem számba. A gateway szoftver oldali működésének és fejlesztésének alapjait ismertetem először, majd az elkészített szoftver főbb részeit is. Ezen főbb modulok a gateway-en futtatandó, C nyelven írt beágyazott szoftver, valamint a személyi számítógépen futó, Java nyelven megírt, gateway-t vezérlő szoftver. Ezeket a részeket külön-külön fejezetben dokumentálom.

Az elkészített hardveres, illetve szoftveres komponensek teljes mértékben integrálják a CAN részleges hálózatok kezelését a vállalat jelenlegi tesztelői környezetébe. A továbbiakban így lehetőség nyílik a már elkészült, vagy jelenleg futó projektek kiegészítésére az elterjedés előtt álló CAN részleges hálózatok nyújtotta funkciókkal.

Abstract

The scope of my undergraduate thesis is the function extension of the field-bus gateway – developed by ThyssenKrupp Presta Hungary Kft. – with handling CAN partial networks. This development is fully integrated in the existing gateway module, both in hardware and software “levels”.

First, I describe the aims of the development for CAN partial networks, their operation, and the CAN protocol used for them.

Then I present the field-bus gateway to be developed, and describe its utilization and operation.

Next, the implementation of the project is described step by step, starting with the hardware. In this chapter I am describing the steps of the hardware realization from the design requirements to its manufacturing.

Hardware description is followed by the introduction of the software tasks. First, the operation and development principle of the software are described, followed the main parts of the developed software. These main software modules are the embedded software written in C language to be run on the gateway and the software written in Java language to be run on a personal computer controlling the gateway. These main software modules are described in separate chapters.

The CAN Partial network handling is fully integrated by the developed hardware and software elements into the existing test environment of the Company. It gives opportunity to extend the finished or presently running projects with the functions of CAN partial networks.

1. Bevezetés

Az Európai Unión belül a közúti közlekedés okozza az egyik legnagyobb CO_2 kibocsátást. Jelentős számú személy- és teherautó használja Európa útjait, rengeteg szennyező anyagot kibocsátva. Ezen káros anyagok csökkentésének szükségességét felismerte az EU, így különböző szabályokkal igyekszik redukálni a mennyiségüket. Az autógyártókat a járművenkénti CO_2 kibocsátás korlátozása érinti legszigorúbban. E szabály alapelve az, hogy fokozatosan, évről évre az eladott autók károsanyag kibocsátásának a megadott mértékben csökkennie kell. Ha ezt nem teljesítik az autógyártók, akkor büntetést kell fizetniük. Olyan magas ez az összeg (pl. ha az átlagos CO_2 kibocsátás meghaladja a határértéket, akkor akár €95-t is fizethet a gyártó minden egyes autója után, minden g/km túllépésért), hogy mindenképpen rá vannak kényszerítve a szabály betartására. [1]

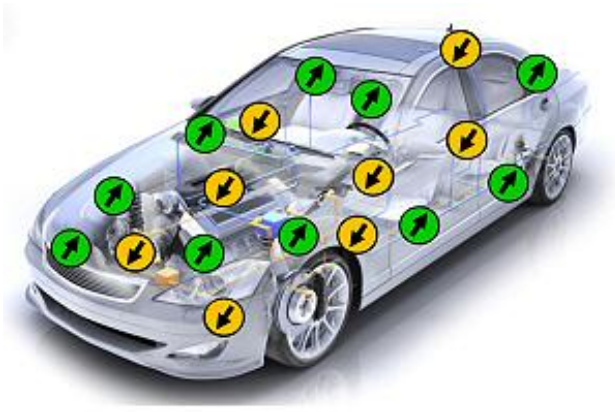
Könnyen belátható, hogy a szabály alapelve a kibocsátott károsanyagok mérséklése elsősorban fogyasztáscsökkentéssel, hiszen a kevesebb üzemanyagot elégető autó kevesebb károsanyagot bocsát ki. Emiatt az autógyártók nagy hangsúlyt fektetnek a járműveik fogyasztására. A mai autók rendkívül összetettek, számos területen, rengeteg módon lehet mérsékelni az üzemanyag felhasználásukat.

A jelenlegi trendek azt mutatják, hogy az autóipari fejlesztések mind nagyobb hányada kapcsolódik az elektronikához. Ezt jól kifejezi az a tény, hogy egy mai prémium kategóriás személyautóban az elektromos vezérlőegységek (Electronic Control Unit, továbbiakban: „ECU") száma megközelíti a 100-at. Mindez megmutatkozik a fogyasztásban is, hiszen a rengeteg ECU energiafelvétele egyáltalán nem elhanyagolható. Az autógyártóknak foglalkozniuk kell a kérdéssel, az ECU-k energiafelhasználásának semmibevételével nem tarthatóak az EU által megszabott irányszámok.

Másrészről viszont a fokozódó felhasználó elvárások/igények (vagyis legyen az autó biztonságos, kényelmes, sokoldalú, adott esetben gyors vagy takarékos) megvalósításához egyre több elektronikus vezérlőegységre van szükséges.

Hogyan próbálnak megfelelni a gyártók ezen ellentétes érdekeknek? Optimalizációval.

Mint említettem, az autókban lévő ECU-k szerteágazó funkciókat valósítanak meg. Ilyen például a parkolóasszisztens, a sebességtartó automatika (tempomat), az ABS vagy az ülésfűtés. A mai autókban ezen egységek többnyire normál állapotukban működnek, ill. normál állapotukban várják, hogy használják őket. Ilyenkor energiafelvételük ugyanakkora, függetlenül attól, hogy elvégzik-e a feladatukat vagy csak várakoznak. Egyáltalán nem optimális az energiafelhasználásuk. Hiszen például 30 km/h sebesség felett semmi szükség nincsen a parkoló automatikára vagy például nyáron nincs szükség az ülésfűtésre. Így ezen - az adott helyzetben szükségtelen - funkciókat vezérlő ECU-kat csökkentett módú állapotukba (úgynevezett sleep módba) kapcsolva, végeredményben üzemanyag takarítható meg jelentős teljesítményvesztés nélkül.



1. ábra: Aktív, ill. passzív állapotú ECU-k adott szituációban

Természetesen van teljesítménycsökkenés, azaz lassabban reagáló vezérlőegység. Ez a plusz idő csupán amiatt keletkezik, mert az ECU-nak, ami a sleep módban volt, érzékelnie kell a keletkezett igényt és át kell térnie normál üzemmódba. Viszont egy adott szituációban (álló helyzet, városi közlekedés, autópálya, tél, nyár, stb.) a megfelelő vezérlőegységek megfelelő beállításával elkerülhető, csökkenthető e késedelmi idő. Természetesen a biztonságkritikus alkalmazásoknál nem élnek e lehetőséggel (pl. egy ABS esetén bármikor fontos lehet a mihamarabbi beavatkozás).

Az autók buszrendszereinél előszeretettel alkalmazott CAN kommunikációs protokoll viszont az addigi formájában ezt nem tette lehetővé, hiszen alapvető

tulajdonsága, hogy minden egység olvassa a busz forgalmát (és ekkor nincsenek energiatakarékos üzemmódban). Emiatt szükségessé vált egy, a fenti működést biztosító, CAN alapokon működő koncepció kifejlesztése. Ez a CAN részleges hálózat, vagy angolul CAN Partial Networking (CAN PN).

A CAN Partial Networking-gel járó előnyök miatt ez a szemléletmód nagy valószínűséggel eljut a sorozatgyártott járművekig. Emiatt az autóipari beszállítóknak is alkalmazkodniuk kell ehhez, integrálniuk kell saját rendszerükbe. Így ThyssenKrupp Presta Hungary Kft. is fejleszt ebben az irányban. Szakdolgozatomban e fejlesztés egy részét valósítom meg, feladatom: CAN PN integrálása a cég által buszmonitorozásra, illetve tesztelésre kifejlesztett terepbusz illesztő egységébe (TKP Fieldbus Gateway). Ez magában foglalja a hardverszintű megvalósítást (hiszen a CAN PN számára egy speciális adó/vevő kell), valamint a meglévő szoftverarchitektúra kibővítését az újítás kínálta funkciókkal.

2. CAN Partial Networking

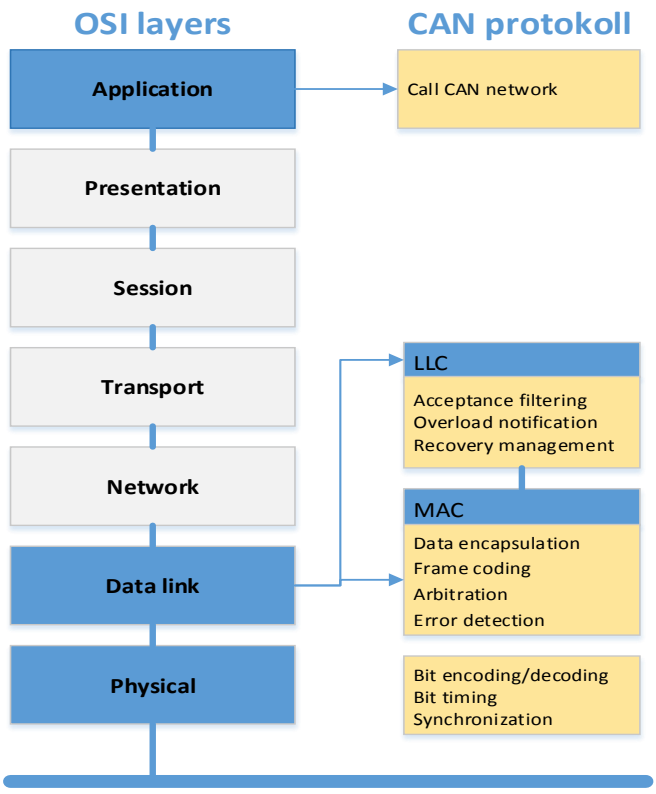
2.1. CAN kommunikációs protokoll

2.1.1. Bevezetés

Az autóiparban alkalmazott elektromos eszközök számának növekedésével, széleskörű elterjedésével szükségessé vált ezen egységeket vezérlő ECU-k közötti szorosabb együttműködés. Ezen ECU-k közötti összeköttetés kezdetben pont-pont alapú volt, viszont az ECU-k számának növekedésével ez a megoldás egyre inkább fenntarthatatlanabbá vált. Túl magas lett a kiépítés költsége, bonyolulttá, nehezen átláthatóvá vált a rendszer, rugalmatlan volt és túlságosan sok helyet foglalt. Ezen problémák orvoslására fejlesztette ki a Robert Bosch GmbH a CAN (Controller Area Network) protokollt az 1980-as években, elsősorban járművön belüli kommunikációs célra. Olyannyira megfelelt az járműgyártás igényeinek, hogy megjelenése óta a leggyakrabban alkalmazott kommunikációs megoldássá vált.

2.1.2. Protokoll ismertetése

Az OSI modell (Open Systems Interconnection Reference Model) egy rétegek szerint csoportosított rendszer absztrakt leírása, amely az elektromos egységek kommunikációjához szükséges hálózati protokollokat határoz meg. E modell létrejöttének célja a nyílt kommunikációs protokollok kifejlesztésének elősegítése. Egy adott protokoll az OSI modell szerinti, egymásra épülő rétegekkel (a rétegek kizárólag az alattuk lévő rétegek által nyújtott funkciókra támaszkodhatnak) tervezhető, jellemezhető. A CAN protokoll is feloszthatóe e rétegeknek megfelelően (lásd 2. ábra), így ismertetni az OSI modell szerint fogom.



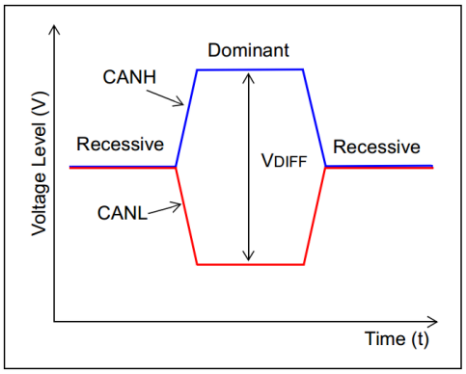
2. ábra: Az OSI modell szerinti CAN protokoll

2.1.2.1. Fizikai réteg (Physical Layer)

A CAN protokoll az összekötendő csomópontok (Node-ok) között egy buszrendszeren keresztül valósítja meg a kommunikációt. A fizikai réteg feladata a bitek továbbítása a csomópontok között, amiket a buszhoz csatlakozó összes egység fogad (szórásos típusú hálózat).

A CAN fizikai rétegeknek több fajtája van, szakdolgozatom során az autóiparban gyakran alkalmazott úgynevezett nagysebességű (High Speed CAN, ISO 11898-2) változatot használtam. Ennek a maximális sebessége 1 Mbit/sec. (Ennek ellenére a járművekben alkalmazott CAN buszok sebessége az adatbiztonság, illetve a zavarokkal, késésekkel szembeni tolerancia fokozása miatt csupán 500 kbit/s). A maximális sebesség függ a kábel hosszától is, alapvetően a bitek ütközése szab ennek határt, azaz bitidőn belül a kibocsátott jel a kábel teljes hosszában végig tud-e haladni. Ennél a sebességnél a kábel szabvány szerinti megengedett maximális hossza 40 méter, így sok, különbözőképpen megvalósított CAN busz (más kialakítás, jelterjedési idő, kábel) teljesíteni tudja az előírt sebességet. Ezenkívül ez a hossz mindig kényelmesen elég az autókban elvárt méretekhez.

A jelek továbbítása a CAN protokoll két vezetéket használ, a jelátvitel differenciális. Tehát a két vezetéken lévő jel különbsége adja meg a buszon éppen küldött bit értékét. Emiatt két állapotot különböztetünk meg: a busz recesszív állapotában a két jelvezeték azonos feszültségszinten van, míg domináns állapotában az egyik vezeték magasabb, valamint a másik vezeték alacsonyabb feszültségszintet vesz fel (lásd. 3. ábra). E kialakítás miatt a CAN busz zavarvédettsége nagyon magas, hiszen az egymás melletti vezetékekre ható külső zavar a jelszintek kivonása során kiesik.



3. ábra: CAN vezetékek jelszintjei

A fizikai réteg feladata a buszon átmenő bitek írásán/olvasásán kívül ezek időzítése is. A CAN buszokon nincs központi órajel (protokoll egyszerűsége végett), így a megfelelő időzítést, szinkronizációt másképp kell kialakítani. Ennek egyik eleme az azonos bitráta használata a buszon belül, azaz minden ECU-nak egységes sebességgel kell kezelnie a buszt. Az azonos bitráta megvalósítása az egy buszon lévő vezérlőegységek megfelelő programozásával érhető el. Ekkor minden egység a saját órajele szerint állítja elő a beprogramozott bitrátát, már csak össze kell hangolni, szinkronizálni kell az egyes bitidők kezdetét, végét, hogy egységessé váljon a busz. Az ECU-k összeszinkronizálása a buszon keletkező lefutó élre történik. Ez először az üzenet kezdetekor történik meg: a forgalommentes busz jelszintje recesszív (logikai 1), viszont az érkező üzenet domináns (logika 0) startbitje a buszt lefele húzza – a buszon fizikailag kialakított logikai ÉS kapcsolat miatt-, így egy lefutó élt generálva. Másodszor pedig szükségessé válik az üzenetküldés közben is a csomópontokat összeszinkronizálni, az üzenetek hosszúsága miatt. Így az üzenetekben előforduló lefutóél-váltáskor is szinkronizáció történik. Előfordulhat, hogy az üzenet hosszú ideig nem tartalmaz 1→0 átmenetet, ezért a megfelelő jelváltások biztosítására a protokoll

minden 5 azonos bit után beszúr egy ellentétes bitet (bit stuffing), aminek az értékét a protokoll majd nem veszi figyelembe.

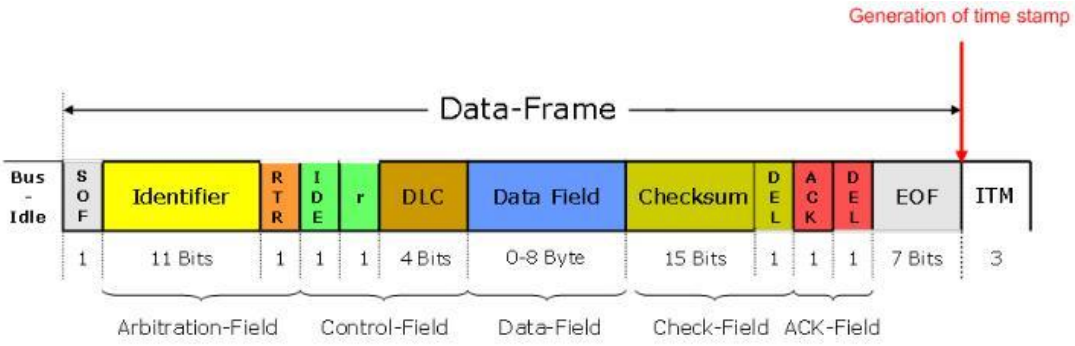
2.1.2.2. Adatkapcsolati réteg (Data Link Layer)

Az adatkapcsolati réteg valósítja meg azokat a funkciókat, amelyek segítségével az adatok átvitele lehetségessé válik két hálózati elem között. Két alréteget különböztet meg az OSI modell: átviteli (Media Access Control, MAC) és objektum (Logical Link Control, LLC) alréteg.

Átviteli alréteg (MAC)

Mint korábban említettem, a CAN protokoll szórásos típusú hálózat. Azaz a buszon átmenő adatforgalmat az összes csatlakoztatott vezérlőegység látja. Az üzenetek nem egy-egy csomópontnak vannak címezve, hanem magának az üzenetnek van egy azonosítója, amely alapján felhasználják a vezérlőegységek vagy figyelmen kívül hagyják azt. Az átviteli alréteg gondoskodik a küldendő információ megfelelő becsomagolásáról.

A CAN protokoll által definiált üzenetek keretekből (frame) állnak, amelyet a 4. ábra szemléltet:



4. ábra: CAN adatkeret felépítése

A keret kezdetét a SOF (Start Of Frame) bit jelzi, ami mindig domináns állapotú. Utána következik az ID mező, mely az üzenetet azonosítja. A buszon lévő egységek ezt az ID-t vizsgálják, ha megegyezik a számukra szükséges ID-vel, csak akkor dolgozzák fel a frame-beli adatokat. Az ID mező hossza kétféle lehet: a normál adatkeret esetében ez 11 bitből áll, míg a kiterjesztett adatkeret esetében ezt kiegészítették 18 bittel, azaz összesen 29 bit hosszúságú. Erre azért volt szükség, mert a CAN 2.0A standard által specifikált normál adatkeret (Standard Data Frame) nem

biztosított elegendő számú bitet a megnövekedett számú azonosítók tárolásához, az ID mező 11 bitje kevés lett. Ezért szükség volt a CAN 2.0B által definiált 29 bites kiterjesztett adatkeretre (Extended Data Frame). A variációk száma nagyban nőtt így, ráadásul a normál adatkeret továbbra is használható maradt a CAN 2.0B specifikáció szerint.

A következő adatmező az RTR bit (Remote Transfer Request), amely jelzi, hogy adatkerettől (domináns értékű) vagy távoli keretről (recesszív értékű) van-e szó. Ezután a kontroll mező bitjei következnek. Először az IDE bit (Identifier Extension), amelynek domináns értéke jelzi, hogy standard formátumú a keret, míg recesszív értéke a kiterjesztett keretet jelöli. Utána egy minden esetben domináns bit szerepel, majd a maradék 4 bit jelöli a DLC (Data Length Code) értékét. A DLC a keretben küldendő adatbájtok számát jelzi, amely 0 és 8 közötti lehet.

A CAN protokoll legújabb verziójában az adatbájtok maximális száma már 64 is lehet. Ez a szabvány a CAN FD (CAN with Flexible Data Rate, ISO 11898-7), melynek célja a CAN protokoll sebességének növelése. Az elérhető sebesség 2-8 Mbyte/s is lehet, attól függően mennyire használják ki az új keret adta lehetőségeket. Míg a frame többi részén a sebesség nem változott (max. 1 Mbyte/s), addig az adatmezőben jóval több adatbájt fér el, és e nagyobb mennyiséget gyorsabban (akár 12 Mbit/s) lehet továbbítani. Ezt úgy érték el, hogy küldéskor az adatmezőben a bitidő kisebb értékre vált, míg a többi mezőben a CAN 2.0B-beli értéken marad. A munkám során felhasznált nagysebességű CAN adó/vevő képes tolerálni ezt a keretet, tehát feldolgozni nem tudja, viszont nem érzékeli hibásnak és nem küld hibaüzenetet a buszra. [2]

Az adatmező után következik az 15 bites ellenőrző összeg (Checksum) valamint egy recesszív értékű határoló bit (delimiter). Majd a nyugtázást jelző bit (Acknowledge, ACK bit) következik. E bit esetén nem a küldő hajtja meg a buszt, hanem recesszív állapotban hagyja azt (elengedi), és azt várja, hogy egy, az üzenetet fogadó node domináns bittel jelezze a sikeres vételt.

Végül a keret 7 darab recesszív értékű bittel zárul le.

Az átviteli alréteg definiálja az üzenetek ütközésének esetét is. A CAN protokoll multi-master jellegű, azaz nincs egy kiválasztott busz-vezérlő, hanem minden egység egyenrangú. Ha a buszt más hajtja meg, akkor a többi csomópont veszi a

jeleket, és amint felszabadult a busz, küldést kezdeményeznek. Ha egyszerre több egység szeretné használni a buszt, akkor ütközés lép fel. Ezt a CAN bitenkénti arbitrációval küszöböli ki, azaz az arbitráció minden szereplője bitről bitre adatot küld és közben monitorozza a buszt. Azon csomópontok, amelyek recesszív bitet küldenek, elvesztik az arbitrációt, ha van olyan másik egység, amely a keret ugyanazon bitpozíciójában domináns bitet küld (a domináns bit felülírja a recesszív bitet a huzalozott ÉS kapcsolat miatt). Bitről bitre megismételve a folyamatot kerül ki a győztes node. Ennek az az előnye, hogy nem jelent időveszteséget, hiszen közben a buszon végig a győztes üzenet kerete lesz látható. A keret felépítése miatt tulajdonképpen az üzenet ID-ja dönti el az arbitrációt (hiszen a start bit minden esetben domináns). A fentiek fényében kimondható, hogy minél előbb van az üzenet azonosítójában domináns bit, annál kisebb értékű az ID, és annál nagyobb eséllyel nyer az arbitráció folyamán. Tehát a kis ID-hoz nagy prioritás tartozik a buszon.

A MAC alréteg további feladata a hibakezelés. A CAN protokoll hibadetektáló képessége meglehetősen jó. Minden buszon lévő ECU folyamatosan monitorozza a buszt, ellenőrzi a forgalmat, még az éppen üzenetet küldő node is. Ezenkívül a csomópontok számon tartják, hogy mennyi hibás üzenetet küldtek ki, és ha ez egy bizonyos határt átlép, akkor lekapcsolják magukat. A CAN protokoll 5 féle hibát képes felismerni:

- bithiba

  Az üzenetet küldő ECU folyamatosan ellenőrzi, hogy az éppen elküldött bit megfelel-e a buszról visszaolvasott értéknek. Ez az önellenőrző mechanizmus főleg a lokális hibák kiszűrésére alkalmas. Természetesen az arbitráció folyamán előfordulhat ilyen a szituáció (az adó recesszív bitjét egy másik egység felülírhatja domináns értékkel), de a kommunikáció ezen szakaszában ez nem minősül hibának.

- üzenetkeret hiba:

  A hálózaton lévő üzenetek helytelen formája esetén keletkezik. Ekkor az üzenetkeretek rögzített értékű bitjei eltérnek a protokoll által megkövetelt értékektől.

- CRC bit hibája:

A CRC bit ellenőrző szerepet tölt be a keretben. A fogadó ECU a CRC bit segítségével ellenőrzi le a vett adatok helyességét (nem torzultak-e). Ha a CRC bit értéke nem megfelelő, akkor hiba történt (crc error).

- bitbeszúrás ellenőrzése:

  A fizikai réteg ismertetésekor említettem, hogy a megfelelő szinkronizáció miatt szükséges lefutó élet generálni, amit a protokoll bitbeszúrás módszerével is végrehajt (minden ötödik egyforma bit után). Ha az üzenet egymás után 6 ugyanolyan értékű bitet tartalmaz, akkor nem történt meg a bitbeszúrás, nem volt szinkronizáció sem. Ekkor generálódik a stuff error.

- nyugtázási hiba:

  A normál üzenetkeret tartalmaz egy Acknowledge bitet, amely segítségével a vevő az adó felé jelezheti az üzenet hibamentes fogadását. Ehhez legalább egy csomópontnak sikeresen kell fogadnia az üzenetet. Amennyiben ez nem történik meg, nyugtázási hiba (acknowledge error) generálódik.

Ha a fenti hibák egyikét észleli egy csomópont, akkor az egy hibakeretet küld a buszra. A hibakeret alaptulajdonsága, hogy egymás után minimum 6 darab azonos értékű bitet tartalmaz, amellyel szándékosan megsérti a bitbeszúrás szabályait, így jelezve egyértelműen a hibát. A hibás üzenetet elküldő adó csomópont ezt észlelve újra tudja küldeni az üzenetet.

Objektum alréteg (LLC)

Az objektum alréteg dönti el, hogy buszról vett üzenetet továbbítja-e a felsőbb rétegeknek. Tehát az ECU vagy eltárolja az üzenetet vagy elveti. Ez a funkciót általában a vezérlőegységeken belül különböző szűrési feltételekkel valósítják meg az üzenetek azonosítója alapján. Egyfajta interfészt biztosít az alkalmazási réteg felé.

2.1.2.3. Alkalmazási réteg

Az alkalmazási réteg valósítja meg a magas szintű funkciókat. Ekkor már a vezérlőegység memóriájában van a releváns (küldeni kívánt vagy fogadott) üzenet, ami az adott egység feladatának megfelelően szabadon felhasználható.

2.1.3. Tulajdonságok

Eddig a CAN protokoll felépítését ismertettem, ebben az alfejezetben viszont pontokba szedve kiemelem a főbb tulajdonságait. [3]

Multimaster jellegű

A CAN protokoll multimaster jellegű, azaz egyik node sem vezérli központilag a buszt, hanem mindig azt a node-ot illeti meg a használat joga, amely éppen utoljára elnyerte a busz meghajtásának jogát. Így a buszon lévő ECU-k függetlenek, nincs szükségük a küldéshez más egységre, csak a szabad buszra. Ez nagyon rugalmassá teszi a hálózat kialakítását, a vezérlőegységek nagyon egyszerűen hozzá-, ill. elvehetők. A CAN ezen tulajdonsága emellett ez a hibatűrő képesség is növeli, hiszen egy csomópont kiesésével a rendszer nem válik használhatatlanná. Csupán a teljesítménye csökken, mert a kieső node-nak küldött üzenetek feleslegesen foglalják a buszt.

Broadcast jellegű

Az üzenet küldése szórásos jellegű, ha egy ECU üzenetet küld a buszra, akkor minden csatlakoztatott egység megkapja azt. Így az egyes ECU-k számára fontos adatok egy üzenetküldéssel eljuttathatóak a forrástól. Pl. az autóipari felhasználás során gyakran van szükség a pillanatnyi sebességre a különböző egységeknek. A broadcast felgyorsítja az ilyenfajta információáramlást. Emellett az üzenetszórás a hibadetektálást is elősegíti, mivel minden ECU, amikor olvassa a busz állapotát, ellenőrzi is, hogy a kiolvasott értékek megfelelnek-e a szabványos keretformátumoknak. Ha nem, hibaüzenetet küld.

Üzenetközpontú

Az adat elküldéséhez nem az adó és a vevő azonosítása szükséges, hanem magát az üzenetet azonosítják a buszon lévő ECU-k, az üzenet keretében lévő ID segítségével. Ez a kialakítás segíti elő a broadcast jelleget.

Eseményvezérelt

A kommunikáció egy adott ECU-beli esemény létrejöttekor kezdődik el, melyet ez az adott ECU kezdeményez. Az eseményvezérelt működés összehasonlítva a valamilyen időhöz, periódushoz kötött rendszerekkel, sokkal hatékonyabb, mivel sosem

vész kárba időszelet. Továbbá, ha a buszon nincs kommunikáció, a csatlakoztatott ECU-k akár csökkentett üzemmódba is kapcsolhatóak, spórolva így az erőforrásokkal.

Olcsó

A CAN buszok létrehozási költségei alacsonyak, mert nem igényelnek különleges eszközöket. A CAN rendszerekben használt csavart érpár olcsó, hiszen az iparban több buszrendszerhez is ezt használják. A szükséges vevők szintén nem drágák, mivel ezen eszközök szintén elterjedtek. Ezenkívül a továbbfejlesztése is rendkívül egyszerű, ezáltal olcsó, mert a hálózat bővítése nem igényel összetett tervezési folyamatot.

Mindezek ellenére elfogadhatóan gyors, így remek ár/érték arányú hálózat alakítható ki a protokoll alkalmazásával.

Elfogadható sebesség

A CAN 2.0B szabvány maximális sebessége 1 Mbit/s, amely értéket befolyásolja a kábel hosszúsága. Ez a maximális érték az eddigiekben kielégítőnek bizonyult. Viszont igény jelentkezett az ennél nagyobb bitrátára, melyet a CAN protokoll módosításával (CAN FD) abszolváltak. A CAN FD szabvány sebessége akár már megközelítheti a sokkal drágább és bonyolult FlexRay sebességét (10 Mbit/s).

A fentebb felsorolt tulajdonságok rendkívül kedvezőek az autóipari, sőt egyéb ipari felhasználás szempontjából. Így érthető, hogy a CAN miért terjedt el ilyen széles körben, miért olyan népszerű kommunikációs protokoll. Megalkotásakor számos felmerülő igényt kielégített. Viszont idővel ezen igények is megváltoztak, amiket már nem teljesít maradéktalanul.

Szakdolgozatom központi témája is ezen felmerülő igények egyike: a protokollon lévő eszközök áramfelvételének csökkentése. A CAN eddigi kialakítása ezt nem veszi figyelembe. Az egyik fő tulajdonsága az, hogy minden egyes buszon lévő üzenetet értelmeznek, olvasnak a csomópontok. Ezáltal, ha forgalom van a buszon, akkor nem lehetséges az ECU-k fogyasztáscsökkentő alvó módba kapcsolása, hiszen felélednek. Tehát a CAN 2.0B szabványnak megfelelően két eset lehetséges, hogy növeljük a csomópontok sleep módban töltött idejét, vagy a buszra minél kevesebb

ECU-t csatlakoztatunk és párhuzamosan több buszt alakítunk ki (vezérlőegységek csoportosítása fizikailag), vagy az üzenetek elküldését szabályozzuk (pl. a küldés idejét). Mindkét megoldás során sok hasznos tulajdonságról mondanánk le, és bonyolítanánk a hálózatot az áramfelvétel csökkenéséért, ami nem lenne optimális.

(Az autógyártók a mai autókban egyszerre több párhuzamos CAN hálózatot alkalmaznak, esetenként 3-5 darabot is, de a vezérlőegységek elkülönítésének fő oka nem az ECU-k alvó módjának elősegítése, hanem inkább a rendszer gyorsítása, a buszok keresztmetszetének növelése.)

A CAN Partial Networking (CAN PN) szabvány egyfajta csoportosítást valósít meg, viszont ezt nem a fentebb említett fizikai elkülönítéssel éri el, hanem az üzenetek azonosítóinak felhasználásával alkot csoportokat, megtartva ezzel a CAN 2.0B szabvány előnyeit.

2.2. CAN PN működése

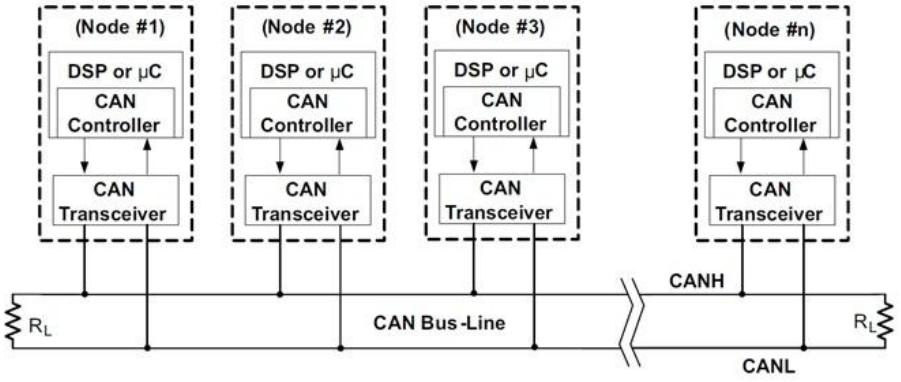
Jelenleg, ha a CAN buszon két ECU egymással kommunikál, akkor a hálózaton megjelenő keretek felébresztik az összes többi csomópontot. Így ilyenkor a többi ECU kénytelen teljesen feleslegesen normál módba lépni, az alacsony áramfelvétellel járó alvó mód helyett. A hálózat teljes fogyasztása akkor lenne ideális, ha csak az aktív egységek (küldő, fogadó) működnének teljes funkcionalitással, míg a többi egység csökkentett módban lenne. A CAN Partial Netwoking (CAN részleges hálózat, ISO 11898-6) pontosan ezt a működést hozza létre.

A CAN PN mindezt az ECU-k úgynevezett szelektív felébresztésével (selective wake-up) éri el. E folyamat lényege: ha szükség van egy ECU-ra vagy azok egy csoportjára, akkor a sleep módból történő felébresztésüket egy külön üzenetkeret (Wake-up frame, WUF) szétküldésével lehet kezdeményezni. A Wake-up frame tulajdonképpen egy normál CAN üzenetkeret (azzal azonos a felépítésű), eltérés csupán a funkciójukban van. A normál adatkeret adatok továbbítására szolgál, míg a WUF egyedüli célja az általa meghatározott vezérlőegység(ek) felébresztése.

A hálózaton lévő ECU-k közül minden egyes egység ellenőrzi, hogy megfelel-e a WUF által támasztott feltételeknek (azaz felébreszteni kívánt ECU-k egyike-e) és ez alapján folytatja a működését. Az újbóli alvó mód elérése már külön-külön a vezérlőegységek

feladata. Általában ez úgy valósul meg, hogy ha egy bizonyos ideig (<1,2 s) nem kommunikál az ECU, akkor automatikusan sleep módba kapcsol.

Mint említettem a standard CAN szabvány szerint az ECU bármely forgalom hatására feléled. Amiatt, hogy a CAN PN ezt a működést megszüntesse és a szabvány implementálása is egyszerű legyen, szükség van egy elkülönített logikára (külön adó/vevőre, transceiverre), amely az előírt működést biztosítja a busz és a mikrokontroller, ECU között. E logika feladata a busz forgalmának elválasztása a mikrokontrollertől (lásd. 5. ábra). A transceiver fogadja a busz jeleit és azokat kiértékelve dönt az ECU felébresztéséről.



5. ábra: Partial networking modul architektúrája

A Partial Networking egyik alapvető tulajdonsága a lehetőség egy külön csoport felébresztésére (eddigi CAN kialakításokban egyszerre lehetet csak az összes ECU-t felébreszteni). Lehetőség van a buszon lévő ECU-k között különböző csoportokat kialakítani, ezzel hatékonyabbá téve a CAN hálózatot. A csoportba rendeződésnek hála egyetlen egy üzenettel lehetséges felébreszteni a vezérlőegységek egy előre definiált halmazát. Így minimális buszhasználattal csak a szükséges egységek kerülnek aktív állapotba. A csoportosításra rengeteg lehetőség van, sok elképzelhető kombináció lehetséges. Az első ehhez szükséges eszköz az üzenetkeret azonosítója. Mint már a normál üzenetkeret ismertetésekor említettem, kétfajta ID-t lehet használni (11 bites standard vagy 29 bites extended ID). A fogadott üzenet ID-ját a transceiver a korábban beállított maszkkal (maszkokkal) összehasonlítja, és ez alapján eldönti, hogy az adott egységnek szól-e a wake-up frame.

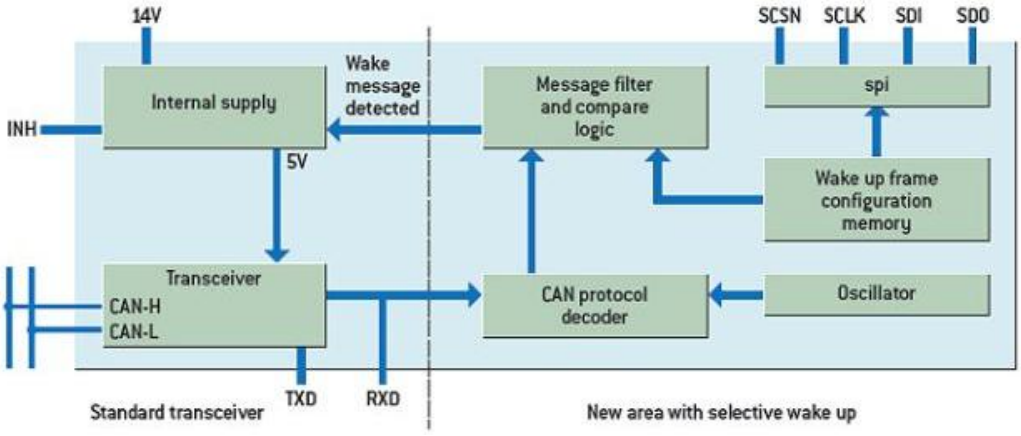
Emellett lehetőség van különböző ECU-kból álló csoportokat előre definiálni. Ezen csoportok azonosítóját a WUF keret adatmezeje tartalmazza. Maximum 8 byte

hosszú lehet (mivel a WUF megegyezik a normál üzenetkerettel), így összesen 64 csoport megalkotása lehetséges. A döntést, hogy az adott ECU a küldött frame által megadott csoporthoz tartozik-e, szintén maszkolás segítségével hozza meg a transceiver, az előre beállított értékek (maszkok) szerint. Felhívnám a figyelmet a nagyszámú variációs lehetőségre, hiszen a maszkolás miatt már egy szűrő feltétel is nagyszámú alternatívát biztosít.

A CAN adó/vevő szűrésének elméleti oldala után az általános felépítését mutatom be.

A selective wake-up funkcióval ellátott transceiver egy standard CAN adó/vevő továbbfejlesztéséből származik. Gyakorlatilag a standard CAN adó/vevőt az alábbi egységekkel kellett kiegészíteni: dekóder, belső oszcillátor, memória, memória külső hozzáférését megvalósító egység és összehasonlító logika (lásd 6. ábra). A dekóder feladata a fogadott CAN keretek feldolgozása, a szükséges adatok kinyerése az üzenetből (ID, DLC, data). A dekóderből a kinyert adatokat egy összehasonlító logikának kell feldolgoznia, amely az előre beprogramozott, memóriában tárolt feltételekkel összeveti azt. Ha teljesülnek a feltételek, azaz a vett ID megegyezik a várt azonosítók egyikével, és adott esetben az alkalmazott adatok is megfelelőek, akkor a transceiver egy felélesztő (Wake-up) jelet küld az ECU számára.

A dekóder számára szükség van még egy nagy pontosságú oszcillátorra, hogy a CAN üzenetek precíz analizálása kivitelezhető legyen. Mivel (autó)ipari elvárás a kompatibilitás a standard transceiverekkel (jelenlegi rendszer könnyebb leváltása miatt), az új egység számára nem lehetséges külső oszcillátor biztosítása, így egy belső oszcillátort is biztosítani kell ezen adónak/vevőnek. Ezenkívül az áramkörbe integrált oszcillátor mellett szól még a helytakarékosság, kisebb fogyasztás (nem kell külön utakon biztosítani áramot a külső oszcillátornak) és az olcsóbb gyárthatóság.



6. ábra: CAN PN képes transceiver felépítése

Tehát, amikor egy üzenet érkezik a buszon, akkor a transceiver-ben lévő CAN üzenetet vevő egység, a dekóder, az oszcillátor, valamint a szűrő logikák aktiválódnak, és ha a kommunikáció befejeződött (bizonyos ideig nincs forgalom a buszon), akkor ezen elemek deaktiválódnak. Míg ezen a folyamat lejátszódik, addig a mikrokontroller sleep módban marad, a transceiver pedig aktív. Az aktív transceiver becsült áramfelvétele kevesebb mint 1 mA, a sleep módban lévő átlagos ECU fogyasztása körülbelül 50 µA. Míg ha olyan ECU kialakítást használnánk, ami nem támogatja a Partial Networkinget, akkor minden egyes CAN buszon átmenő üzenet hatására normál módba kapcsolnának az ECU-k és 250 mA-es áramfelvétellel működnének. Viszont a CAN PN használatával az ECU csak indokolt esetekben, minimális ideig működik ilyen magas fogyasztást elérő állapotban. Egyes középtávú elemzések szerint ez az energiamegtakarítás körülbelül 2,6 g/km CO_2 kibocsátáscsökkenést, valamint 0,11 l/100 km üzemanyag-megtakarítást jelenthet.[1]

2.3. Konklúzió

A különböző autó- és félvezetőgyártó (Audi, BMW, Daimler, Porsche, PSA, Volkswagen, Elmos, Freescale, Infineon, NXP, STM és C&S) által alapított SWITCH (Selective Wake-up and Interoperable Transceiver in CAN High Speed) csoport elérte a kitűzött célt. A fentiek fényében kijelenthetjük, hogy rendkívül hatékony megoldást találtak az elektronika fogyasztáscsökkentésére. Hiszen az elektronikai módosításokkal elért üzemanyag-fogyasztáscsökkenés egyáltalán nem elhanyagolható. Mindez a felesleges elektronika lekapcsolásával történt meg, ráadásul úgy, hogy érdemi teljesítményromlás nem következett be.

Fontos előnye még a CAN Partial Networking használatának a jelenlegi rendszerekbe való könnyű integrálása. Nincs szükség a jelenlegi CAN architektúra átalakítására, a selective wake-up funkciót támogató ECU-k együtt használhatóak a „régebbi”, nem CAN PN alkalmazására tervezett vezérlőegységekkel, akár egy buszon is. Továbbá a selective wake-up fogadására nem képes ECU-hoz csupán egy megfelelő transceivert kell illeszteni a kívánt működés eléréséhez. Így könnyen belátható, hogy a fejlesztési és a gyártási költségek nagyon alacsonyan tarthatóak.

Ezenkívül, a CAN Partial Networking segítségével az autóban lévő ECU-k normál módú használatának ritkulása jótékony hatással van a hardver élettartamára is. A sleep módban lévő vezérlőegység számos alfunkciót megvalósító áramköri részt egyszerűen lekapcsol (feszültségmentesít). Így e területeken lévő alkatrészek sokszor nincsenek használatban, élettartamuk emiatt nagyobb, mint a rendszeresen aktív módban használt egységeké.

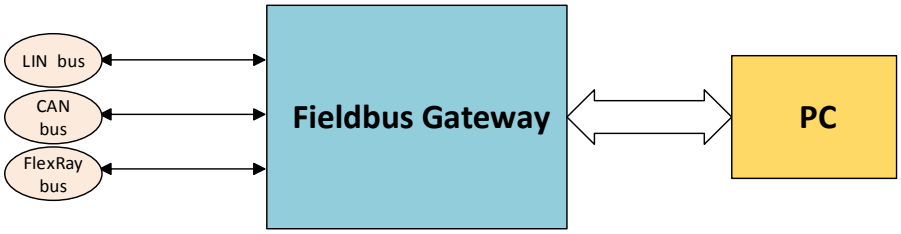
Ezzel szemben a partial networking hátránya jelenleg, hogy még nagyon új ez a kialakítás. Emiatt még viszonylag kevés tapasztalat áll rendelkezésre a hosszú távú gyakorlati alkalmazásáról. De a terjedése mellett szól, hogy kiforrott technikákon alapszik (CAN protokoll, jelenlegi CAN transceiver, jelenlegi vezérlőegységek), valamint könnyen telepíthető és szükség van az általa nyújtott megtakarításra. A trendek azt mutatják, hogy hosszabb távon mindenképpen tovább fog nőni az elektronika aránya az autókban (még több funkció, hibridek, villanyautók), így egyre inkább szükség lesz az általa nyújtott megtakarításra. Tehát az elterjedése csupán idő kérdése.

3. TKP Fieldbus Gateway hardver

3.1. Áttekintés

Az autókban megtalálható buszra csatlakoztatott vezérlőegységek fejlesztése nem a járműbe integráltan történik. Ennek fő oka, hogy többnyire nem az autógyártó fejleszti ki az egyes funkciókat megvalósító alegységeket, hanem ezzel a feladattal külső cégeket, beszállítókat bíz meg. Ezen külső beszállítók a megbízó által definiált követelményeknek megfelelő vezérlőegységeket hoznak létre. Mint a korábbiakban kifejtettem, ezen vezérlőegységek nem függetlenek, hanem többnyire különböző buszokon keresztül (CAN, LIN, FlexRay) kommunikálnak egymással. Így a különböző beszállítóknak úgy kell kifejleszteniük a megvalósítandó rendszerrel kompatibilis terméküket, hogy a fejlesztés során a majdani hálózaton jelen lévő többi egység nem áll rendelkezésükre. Emiatt a hálózati kommunikáció összehangolása, ellenőrzése nehézkes.

A fentebb tagolt nehézségek csökkentése érdekében a ThyssenKrupp Presta Hungary Kft. egy sokoldalú terepbuszillesztő egységet (fieldbus gateway) hozott létre. Ez a gateway egy olyan, fejlesztői számítógépről könnyen programozható, központi egység, amellyel egyszerűen tesztelhetőek a LIN, CAN, FlexRay buszok, illetve a különböző buszok közti kapcsolat (átmenet) is (lásd 7. ábra).



7. ábra: Gateway sematikus felépítése

A vállalat által alkalmazott felhasználási lehetőségek (a teljesség igénye nélkül):

- az AUTOSAR szoftverarchitektúra hardverközeli részeinek (AUTOSAR Basic Software Layer) tesztelése. Vagy más, a buszon lévő ECU üzeneteinek által, vagy a gateway-beli mikrokontroller felprogramozása által.

- buszmonitorozás: a vizsgálat célja megfigyelni, hogy az adott protokollnak megfelelő szabályok érvényesülnek-e a buszon. Tehát például a többi ECU által kibocsátott üzenet megfelelő keretformátumú-e, a bitidőzítések megfelelőek-e, megfelelően működik-e a szinkronizáció.
- kommunikációs tesztek végrehajtása. A gateway a különböző protokollok közti csatlakozást megfelelően hajtja-e végre, helyesen „fordítja le-e" az üzeneteket.
- a beépített SD interfésznek köszönhetően lehetőség van teljes körű ellenőrző tesztek végrehajtására. Erre példa a már autókba beépített egységek hibadetektálása. Az autót tesztpályán vagy forgalomban tesztelik, és a használat közben felmerülő hibák, valamint a hibaesemények bekövetkeztekor fennálló rendszerállapotok automatikusan elmentődnek. Ez mindenképpen elősegíti a hiba javítását.
- a gateway az autóba beépített egységek tesztelését másképp is elősegítheti. A különböző buszokon időnként szünetel a kommunikáció, vagy az adott protokoll tulajdonsága miatt vagy a forgalom hiánya miatt. Így ekkor lehetőség van különböző teszteket végrehajtani a busz segítségével úgy, hogy az nem akadályozza a normál működést (remaining bus simulation). Mindez a gateway vezérlésével bonyolítható le.

A fentiekből látható, hogy a fejlesztéshez a gateway elengedhetetlen segítséget nyújt. Emiatt nem meglepő, hogy a piacon már létezett hasonló tulajdonságokkal bíró eszköz (például a Vector által elkészített VN8970 kódú interfész). Ennek ellenére a cég számára egy saját fejlesztésű eszköz megalkotása előnyösebb: azt maximálisan a cég igényeinek megfelelően lehet kialakítani (és majd továbbfejleszteni), valamint lényegesen olcsóbb, hiszen egy célhardvert kap a cég és nem kell olyan plusz funkciók után fizetni, amelyek amúgy sem lennének kihasználva. Ezen indokok miatt döntött a cég egy vállalaton belüli használtara szánt gateway egység kifejlesztéséről.

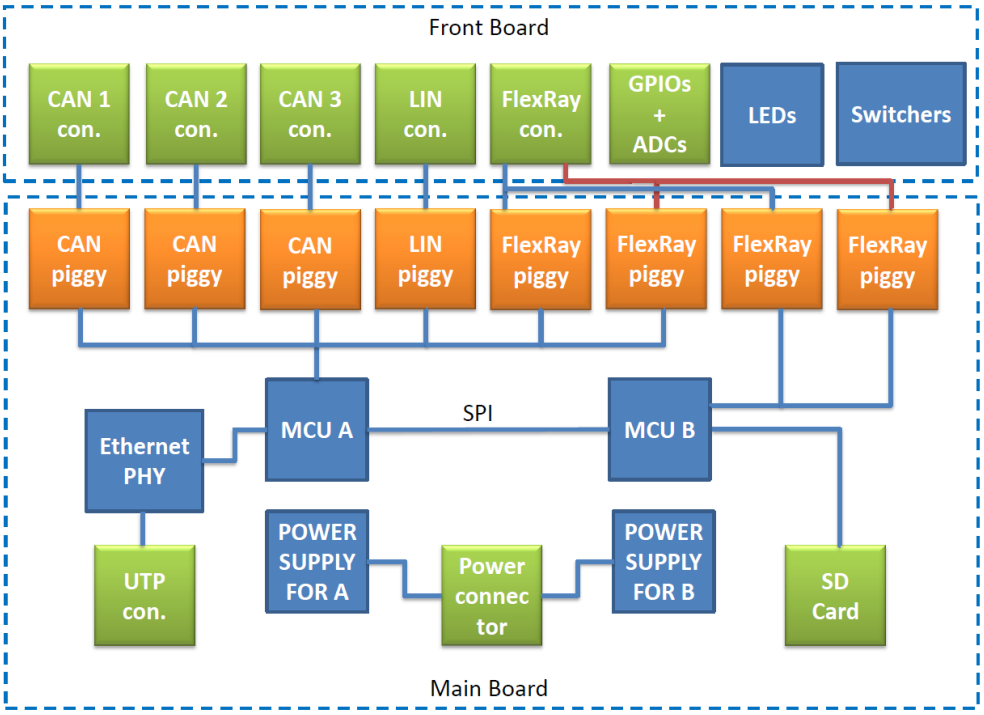
3.2. Felépítés

A gateway felépítését az irodalomjegyzék [4] számú forrása szerint mutatom be.

A TKP Fieldbus Gateway alapját két fő panel (NYÁK – Nyomtatott Áramköri Kártya, vagy angolul PCB – Printed Circuit Board) adja. Az egyik ilyen panel a „Front

Board”, amely a különböző buszok csatlakozási felülete, valamint a felhasználó számára egy külső interfészt (kapcsolók, jelzőfények) adó felület. A másik az úgynevezett „Main Board”, amelynek fő feladata a PC és a busz közötti kommunikáció megvalósítása. Ezt a feladatot két mikrokontroller (MicroController Unit, MCU), ill. az Ethernethez szükséges hardverelemek segítségével látja el.

A két fő panel alkotta eszköz sematikus felépítése a 8. ábrán látható.



8. ábra: TKP Fieldbus Gateway felépítése

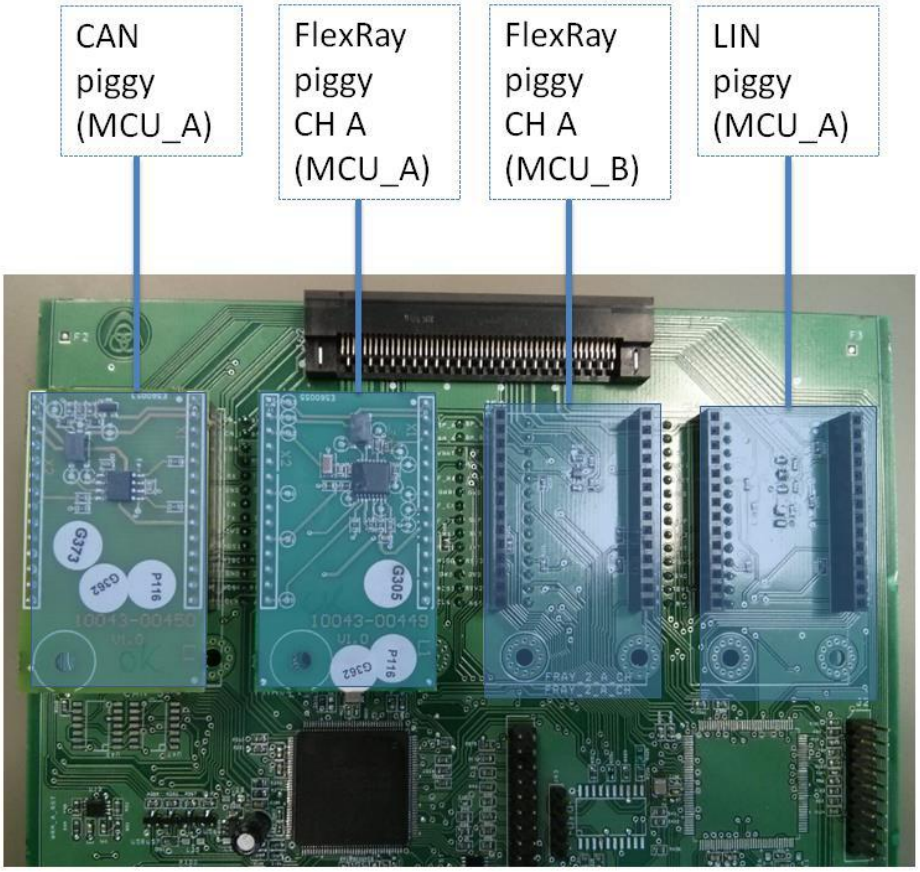
3.2.1. Main Board

A Main Board a gateway központi egysége. Elsősorban a különböző protokollú (FlexRay, CAN, LIN) buszrendszerek kezeléséért, összekapcsolásáért felel. Tehát az autókban lévő, összetett hálózatok gerincét adót FlexRay vagy CAN, valamint tipikusan az alegységek, kisebb alhálózatokat alkotó LIN (Local Interconnect Network Interface) buszok közti kommunikációs eltérések áthidalására alkalmas. Másrészt ez a panel vezérli a buszforgalom monitorozását, és a (teszt)üzenetek küldését.

Az egyes mikrokontrollerek és a különböző buszok közti kapcsolatot az úgynevezett ”piggy” perifériák létesítik. Ezen perifériák tartalmaznak egy, az adott protokollnak megfelelő adó/vevő (transceiver) logikát, amely a két CAN vezetéken

bejövő feszültségszinteket, differenciális analóg jeleket soros, digitális információvá alakítja át. Majd ezt a digitális jelet dolgozza fel a mikrokontrollerbeli megfelelő kommunikációs kontroller. Lehetőség van a mikrokontroller irányításával az összetettebb transceiver logikákat felprogramozni SPI vonalakon keresztül (például, hogy csak bizonyos tartalmú üzeneteket engedjen át az MCU felé).

A gateway hosszabb távú használhatósága miatt a kialakítása során szempont volt a FlexRay, CAN, LIN protokollok később megjelenő, újabb verzióinak támogatása. Emiatt a piggy perifériák flexibilis kialakításúak, nem a Main Board részei, hanem különálló panelek, amelyek csatlakozókon keresztül kapcsolódnak ahhoz (ahogy azt a 9. ábra is szemlélteti). Ennek következtében könnyen kicserélhetőek, az újabb ilyen irányú fejlesztések egyszerűbben implementálhatóak.



9. ábra: Kártyamodulok a Main Board tetején

A blokkvázlatból (8. ábra) egyértelműen látszik, hogy a gateway a buszok különböző lehetséges kombinációja között képes kapcsolatot létrehozni (FlexRay-CAN, FlexRay-LIN, CAN-LIN). Ez a kapcsolat az elsődleges MCU(MCU A)

segítségével épül ki. Az egyes buszok tartalma külön-külön a dedikált vezetékeken eljut a megfelelő transceievertől (piggy-től) a mikrokontrollerhez, amely feldolgozza azt az adott protokoll szabályainak megfelelően, átalakítja a célbusz protokolljának alapján, majd a másik irányba, a megfelelő transceievernek (piggy-nek) továbbküldi. Tehát a gateway alapvető feladatának végrehajtásához szükség van egy mikrokontrollerre (main MCU).
A másodlagos MCU (MCU B) feladata a FlexRay hidegindításának elősegítése. A FlexRay protokoll szerint a kommunikáció elindulásához legalább két buszon lévő csomópont szükséges, mert a hidegen induló (coldstart) csomópontok csak akkor fejezik be az indítási folyamatot, amikor stabil kommunikáció alakult ki egy másik coldstart egységgel. E folyamat miatt szükség van egy másik FlexRay egységre is, amelyet egy FlexRay protokollt kiszolgáló piggy valósít meg. A FlexRay buszon kialakított 2 csomópont viszont a FlexRay hidegindítása mellett használható a kommunikáció tesztelésére is. Így például egy gateway használatával tesztelhető a FlexRay protokoll megfelelő használata vagy AUTOSAR hardverközeli rétegeinek funkciói. Mindezek mellett az MCU B a terhelés elosztására is használható, az MCU-k működésének koordinációja miatt van a két egység között SPI összeköttetés.

A fentiekben ismertetett különböző kialakítási lehetőségek alkalmasak mind a fejlesztendő hardver paramétereinek (pl.: csatlakozás, hibatűrő képessége, pontossága), mind a fejlesztendő szoftver részeinek (pl.: AUTOSAR alacsony, hardverközeli moduljai, a Basic Software rétegei) tesztelésére, ellenőrzésére. A mikrokontrollerek hálózati kialakítása adata lehetőségek után bemutatom magát az MCU-t.

A terepbusz gateway a Texas Instruments TMS5703137 típusú ([5]) mikrokontrollerjein alapszik. Ezt a típusú mikrokontrollert kifejezetten biztonságkritikus rendszerekre fejlesztették ki, mint például fékrendszer, elektromos szervókormány, vezetést segítő aktív rendszerek. A különböző hibák elkerülésére, megelőzésére több megoldást is alkalmazott a gyártó. Az egyik ilyen az MCU-ban alkalmazott két központi feldolgozó egység (Central Processing Unit, CPU), amelyek úgynevezett lockstep rendszert alkotnak. Ezt redundancián alapuló rendszert két, egymással párhuzamosan működő CPU alkotja, amelyek egyszerre ugyanazokat a műveleteket végzik el, majd a kapott eredményeket összevetve egyértelműen

megállapítható, hogy hibamentesek voltak-e a különböző számítások, ill. a busz jeleinek feldolgozása.

További hibadetektáláson alapuló védelmet ad az ECC (Error Correction Code) flash és az ECC RAM memória, amely az írt és olvasott adat azonosságát bithelyesen ellenőrzi, valamint a CPU-beli belső öntesztelő modul, kiegészítve külön hibát jelző kimenettel. Ezenkívül a MCU tartalmaz egy a feszültséget, illetve órajelet monitorozó modult is.

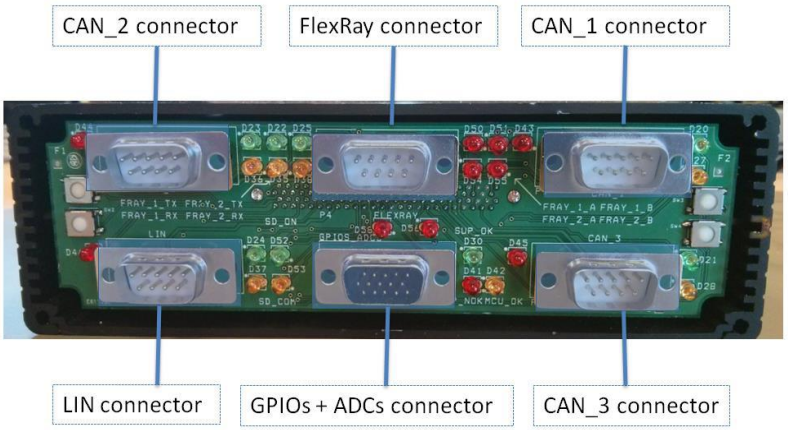
A mikrokontrollerbe integrált CPU egy ARM magos, 32 bites, RISC architektúrájú egység (ARM Cortex-R4F). Működési frekvenciája 1,66 DMIPS/MHz. Emellett az MCU 3 MB programozható flash memóriát, valamint 256 KB RAM memóriát tartalmaz. Az MCU debuggolását JTAG interfész (IEEE 1149.1) illetve az ARM CoreSight komponens támogatja.

A TMS5703137 kontroller kommunikációs interfészek terén elég sokoldalú:

- 3 db SPI kivezetés
- I^2C (Inter-Integrated Circuit)
- SCI (Standard Serial Communication Interface)
- LIN 2.0 standardot támogató kontroller
- 3 db CAN 2.0 (A és B) verziójú szabványt támogató kontroller
- 1 db, kétcsatornás FlexRay
- 10/100 Mbps Ethernet MAC (EMAC)

3.2.2. Front Board

A Front Board szerepe egyrészt a csatlakozási felület biztosítása különböző buszok számára. A nyák felső oldala 3 db CAN, 1 db FlexRay, 1 db LIN csatlakozót tartalmaz, valamint még egy csatlakozót, ami vagy általános célú ki-, ill. bemenet, vagy analóg-digitális konvertert (ADC) csatlakozójaként funkcionálhat.



10. ábra: Beépített Front Board

A buszok mind 9 lábból álló D-SUB (DE-9) csatlakozókon keresztül kapcsolódnak a gateway-hez. Ezt a csatlakozót széleskörűen alkalmazzák a CAN, LIN és FlexRay buszok esetében. Ezeken kívül az ADC és GPIO lábak számára egy több lábbal rendelkező csatlakozó szükséges, hogy a több funkciót megvalósító összes vezeték elférjen. Emiatt ebben az esetben egy 15 lábú D-SUB (DE-15) konnektor szükséges.

A Front Board másrészt egy felhasználói felületet is nyújt a tesztelő számára, amelyet a csatlakozók mellett elhelyezett visszajelző LED-ek és különböző kapcsolók biztosítanak. A LED-ek mindig az aktuális csatlakozó mellett találhatóak meg, főbb funkcióik az alábbiak:

- A csatlakozón bemenő jelet a megfelelő piggy fogja feldolgozni, ezért minden esetben egy, az adott piggy elérhetőségét jelző LED szükséges.
- Az adott csatlakozón történik-e kommunikáció, külön TX (küldést), RX (fogadást) jelző LED
- Tápellátás elegendő-e
- Akkumulátorvédelem be van-e kapcsolva
- A mikrokontrollerek megfelelően működnek-e
- A mikrokontroller hibát észlelt-e

Ezenkívül a felhasználónak lehetősége van az egyes mikrokontrollereket újraindítania, resetelnie. Ez lehet úgynevezett hideg reset (cold reset), amikor az

energiaellátás egy rövid leállás után újraindul, valamint meleg reset (warm reset), ekkor csak a belső egységek kerülnek alaphelyzetükbe. Így a két kontroller e funkcióinak 4 kapcsoló van elhelyezve a board tetején.

3.2.3. Gateway ház

A fentebb megismertetett alapegységek egy közös házba vannak beszerelve. A ház elején, függőleges helyzetben található a Front Board, hogy a csatlakozók, kapcsolók és jelzőfények jól hozzáférhetőek legyenek. A házon belül, mechanikailag védett helyen, vízszintesen helyezkedik el a Main Board, a gateway központi egysége. A két egység közti kontaktust csatlakozók biztosítják: a Front Board hátulján egy 100 lábú csatlakozó található a megfelelő magasságban, amihez ki vannak vezetve a nyák elején lévő busz vonalai, valamint a felhasználói interfészek. Ez a konnektor közvetlenül kapcsolódik a Main Board tetején elhelyezett csatlakozóhoz. Az autóiparban ez a fajta csatlakozási mód a preferált a vezetékes kialakítással szemben. Ennek oka az, hogy a csatlakozók esetében kevesebb hibalehetőség adódhat, mind gyártás (pontatlan összeszerelés), mind használat közben (kevésbé sérülékeny).

11. ábra: Fieldbus Gateway ház

A TKP Fieldbus gateway tervezésekor szempont volt a hordozhatóság, hiszen akár laboratóriumban, akár mozgó járművekben is használhatónak kell lennie. Emiatt szükséges volt felkészíteni az eszközöket a különböző mechanikai behatásokra. Ezért vált szükségessé egy ilyen kompakt házba implementálni a hardvert. Tervezésekor az ütődések mellett ügyelni kellett a rázkódásra is. Emiatt a nyákokban több furat van kialakítva a rögzítő csavaroknak. Az általam tervezett CAN piggy-n is található e célból kialakított furat.

3.2.4. PC-Gateway kapcsolat hardveres megvalósítása

A fejlesztői számítógép és a mikrokontrollerek közti kaput beépített Ethernet kontroller biztosítja (a ház hátoldalán hozzáférhető csatlakozó segítségével), így UTP kábel segítségével könnyen létrehozható a kapcsolat. Ez az egység támogatja mind a 10 Mbps (10Base-T), mind a 100 Mbps (100Base-TX) sebességű szabványokat, amelyek közt automatikusan vált, ha kell. Az átvitel fél-duplex, azaz egy időben csak egy irányú vagy duplex, tehát egyszerre kétirányú lehet.

4. CAN PN implementálása, projektterv

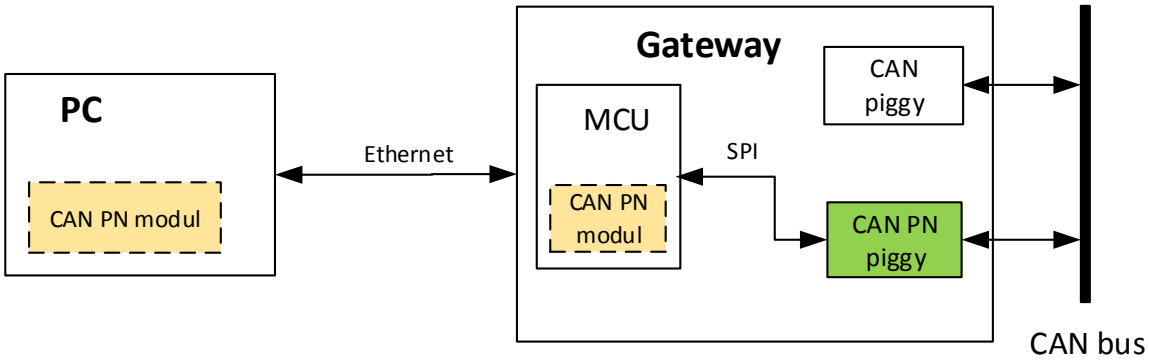
A fejezet célja, hogy áttekintést adjon az elkészített projektről, megismertesse az elkészült modulok közti összefüggéseket.

Munkám célja a korábbiakban bemutatott TKP Fieldbus Gateway egység továbbfejlesztése volt. A gateway egy összetett fejlesztői eszköz, amely segítségével különböző buszok működése egy külső számítógép igénybevételével tesztelhető, ellenőrizhető. E komplikált művelethez szükség van a tesztelendő buszt illesztő áramkörre, valamint a buszbeli kommunikációt feldolgozni képes mikrokontrollerre. A felsorolt hardver elemek viszont önmagukban nem elegendőek a teszteléshez, a gateway működését vezérlő szoftver is elengedhetetlen. A gateway működését befolyásoló szoftver két elemre bontható: a mikrokontrollert működtettető, úgynevezett beágyazott szoftver oldalra, illetve a mikrokontrollert kívülről vezérlő számítógép oldalra (e két szoftverrész természetesen nem független egymástól).

Ahhoz, hogy ezen összefüggő struktúrába a CAN Partial Networking jelentette funkciók integrálhatóvá váljanak, biztosítani kell a megfelelő hardveres kiegészítést (ez elkerülhetetlen, mert az új szabványt az eddig alkalmazott transceiverek nem képesek kiszolgálni), valamint biztosítani kell az új funkciókat irányító, kezelő szoftveres kiegészítést is.

A hardveres fejlesztés tulajdonképpen magában foglalja egy új transceiver illesztését a gateway egységbe. Az illesztést pedig az úgynevezett CAN piggy platformok újratervezése jelentette (lásd 12. ábra, folytonos vonalú, zöld blokk).

A szoftveres fejlesztés magában foglalja a mikrokontrolleren, illetve a számítógépen futó programok módosítását. Ez azért szükséges, mert az új transceiver az eddigiekkel ellentétben programozható, így jóval összetettebb műveletek végezhetőek el vele. Az új műveletekhez tartozó többlépcsős vezérlést (számítógép, mikrokontroller által) pedig létre kellett hozni a szoftverstruktúrában (lásd 12. ábra, szaggatott vonalú, sárga blokkok).



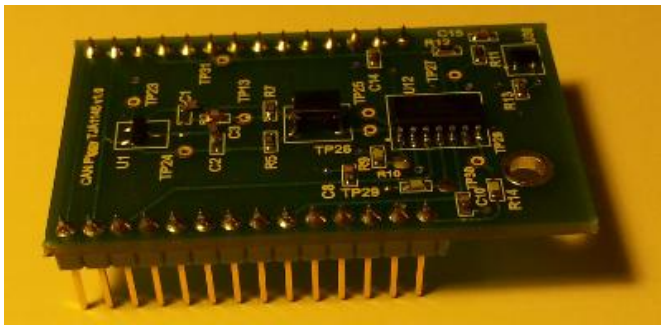
12. ábra: Projekt vázlat

5. CAN PN transceiver hardveres illesztése

5.1. Áttekintés

A szakdolgozatom első tervezést igénylő része egy hardver elkészítése. A cél egy meghatározott vezérlőegységhez, egy speciális, CAN protokollú buszt illesztő platform létrehozása. Az általam tervezett platform a cég keretein belül korábban létrehozott TKP Fieldbus Gateway (3. fejezet) eszközének egy kiegészítő modulja.

A gateway elsősorban a LIN, CAN, FlexRay protokoll szerint működő buszokat kapcsolja össze. A különböző kommunikációs vonalakat külön-külön, egy-egy egyszerűen cserélhető elektronika (piggy platform) kezeli, amely - a szükséges szűrés és védelem mellett - az analóg busz jeleit digitális adatokká alakítja át a mikrokontroller számára.



13. ábra: Az elkészült CAN Partial networking képes piggy

A gateway tervezésekor szempont volt a kommunikációs protokollokat kezelő áramkörök könnyű cserélhetősége, moduláris felépítése. Ennek az az oka, hogy a technológiai fejlődés nem áll meg, folyamatosan finomítják vagy bővítik a már létrehozott szabványokat. A cég pedig ezzel a fejlődéssel természetesen lépést szeretne tartani, úgy, hogy ne kelljen mindig teljesen új gateway-t tervezni, illetve legyártani. A piggy-k használatával elegendő csupán ezeket újragondolni, ahogy azt a szakdolgozatban is tesztem.

A gateway-en lévő, eddig használt CAN buszt illesztő áramkör feladata a következő volt: feltétel nélkül fogadni a CAN üzenetet, és továbbítani a mikrokontroller felé. Ezt a működést a gateway a CAN 2.0B (hosszú ideje alkalmazott és rendkívül népszerű) szabványa szerint valósította meg. A CAN 2.0B protokollú

üzeneteket az NXP által gyártott TJA1042T/3 transceiver dolgozta fel az eddigi piggy modulon.

A CAN protokoll jelenlegi legfrissebb újítása a partial networking (lásd 2.1. fejezet) használata. A CAN PN összetett működésének megvalósításához viszont alkalmatlan az eddigiekben használt transceiver, szükséges egy új, speciális transceiver alkalmazása.

5.2. Transceiver bemutatása hardver oldalról

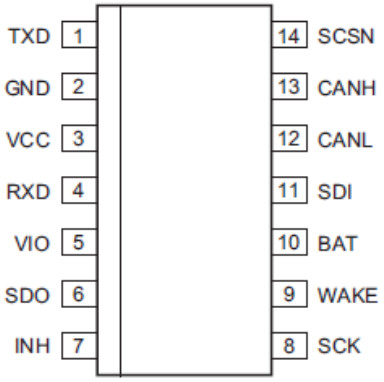
Az általam használt CAN adó/vevő az NXP TJA1145 ([6]). Kiválasztásában szerepet játszott az, hogy kifejezetten az autóipari felhasználású, nagysebességű CAN hálózatokhoz tervezték, illetve a gateway egységben a korábbi transceiverek is az NPX hasonló felépítésű termékei voltak.

A korábban használt CAN transceiverhez képest a TJA1145 sokkal fejlettebb, összetettebb funkciók megvalósítására képes. Természetesen alapvetően ez is a CAN busz jeleit továbbítja a mikrokontroller felé, de működése sokféleképpen szabályozható, állítható. E konfiguráláshoz szükség van egy kétirányú kommunikációs csatornára, amelyen keresztül a mikrokontroller vezérelheti a transceivert. Ezt a csatornát a beágyazott rendszerekben előszeretettel használt SPI (Serial Peripheral Interface) vonalak biztosítják.

A transceiver oldalon a különböző beállítások, feltételek alkalmazásához szükség van az ezekhez tartozó adatok tárolására is, amit a transceiver belső regiszterei biztosítanak. A transceiveren elvégezhető beállítások kezelése szoftveres úton történhet (az SPI vonalak segítségével), emiatt ezeket majd később részletezem a Transceiver funkciók használata című fejezetben. A továbbiakban az adó/vevő hardveres szempontból releváns tulajdonságaival folytatom.

A CAN adó/vevő a busz analóg jeleit kezelő CANH, illetve CANL lábak segítségével csatlakozik a buszhoz. A transceiver fő feladata az e lábakon megjelenő jelek fogadása/küldése, valamint a jelek és a mikrokontroller által használt digitális adatok közti átalakítás. A mikrokontroller és a transceiver közti digitális adatok cseréje a TXD és RXD lábakon keresztül valósul meg.

CAN Partial Networking szabvány alkalmazása esetén, ha a fogadott üzenet megfelel az előre konfigurált feltételeknek, akkor a transceivernek fel kell ébresztenie a mikrokontrollert. Ez az ébresztés kétféleképpen történik. A külön tervezést nem igénylő esetben, a transceiver a fogadott üzenetek továbbítására (a mikrokontroller felé) szolgáló RXD vonalat 0 értékűre állítja be (domináns értékre), amelyet a mikrokontroller egy bejövő CAN üzenet kezdőbitjének érzékel, és felébred. Ezzel párhuzamosan, ha a mikrokontroller tápellátása külön szabályozható (a 16. ábra szerinti felépítés szerint), akkor ennek kihasználásával is felébreszthető az eszköz. A transceiver rendelkezik egy, a mikrokontroller külső tápellátását vezérelni hivatott dedikált lábbal (INH), amely a BAT lábon beérkező feszültség felhasználásával felébresztheti a mikrokontrollert, ha az szükséges.



14. ábra: TJA1145 lábkiosztása

Akárcsak a mikrokontroller, a transceiver is többféle üzemmóddal rendelkezik. A különböző üzemmódok sajátosságai miatt (pl. alvó módból való kilépés) a transceiver használata folyamán szükség lehet a külső, egyszerűen vezérelhető felébresztésére is. Az eszköz oldalán megtalálható WAKE láb e célt szolgálja. [6]

5.3. Tervezési követelmények

Az általam tervezett CAN piggy egy kész hardverstruktúra (TKP Fieldbus Gateway) része, ebből adódóan a főbb paraméterek kötöttek voltak, nem lehetett eltérni azoktól.

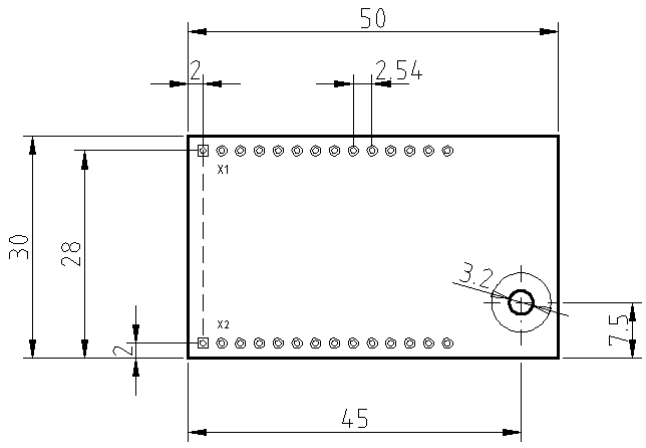
5.3.1. Huzalozás

A paraméterek teljesítése a nyomtatott áramköri panel kialakításakor is nélkülözhetetlen volt. Mivel a tervezett hardver a gateway egy cserélhető eleme,

szigorúan követnem kellett a fizikai paramétereket. A gateway gondos tervezése miatt elfértem a megszabott területen, nem kellett a vezetékezés kialakítása során sok, kompromisszumos megoldást használnom. Ebben közrejátszott az is, hogy nagyméretű elemeket nem kellett alkalmaznom, pl. elektrolitkondenzátor. A transceiver számára előírt egy ilyen, az akkumulátoros működés esetén használt tápszűrő elektrolitkondenzátor. Ez a védelem viszont a Main Boardon van egységesen kialakítva, nem pedig az egyes piggy-ken, így a nyák kialakításakor nem kellett ügyelnem a helykihasználásra.

A gateway számára az előírt méreteket a 15. ábra mutatja. Az ábrán látható, hogy a méret mellett a különböző célú furatok pozíciójára is figyelnem kellett. A piggy-t rögzítő csavar számára szükség van egy szabványos méretű furatra (M3). A gateway-t mozgó járművekben is használni fogják, a rázkódások ellen muszáj rögzíteni a Main Board tetején lévő platformot, a csatlakozók erre a célra nem felelnek meg. A nyákon két csatlakozósor helyezkedik el, mindkettő 14 elemből álló, egysoros, 100 mil-es pitch konnektor. A két külön konnektorsor, valamint a csatlakozó fajtája a piggy elemet egyértelműen pozicionálja a Main Board tetején. Sajnos a csatlakozó lábainak kiosztási sorrendje, illetve a használt transceiver lábkiosztása nem volt szinkronban, a megfelelő összekötéshez sok kereszteződést kellett feloldanom. Ezt a problémát elsősorban a transceiver SPI interfésze idézte elő, amelynek az egyes kivezetései az IC mindkét oldalán szét voltak szórva. Míg a csatlakozó esetében teljesen más sorrendben, de amúgy logikusan egymás mellett helyezkedtek el. Az ilyen fajta nehézségeket csak a transceiver megfelelő pozícionálásával, illetve egyes vezetékek hátoldalon való elvezetésével (via segítségével) oldottam meg.

A vezetősávokkal ellentétben az alkatrészeket csupán a felső oldalon lehetett használnom, ennek oka a piggy modul minél egyszerűbb kialakítása, gyártása és az, hogy ezt az elemet a Main Board tetejére kell csatlakoztatni.



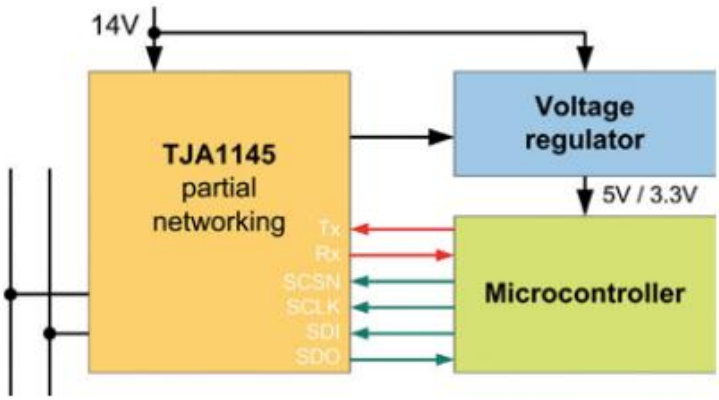
15. ábra: Piggy platformok méretei mm-ben (LIN, CAN, FlexRay)

Az előre megtervezett, cserélhető piggy kialakításnak több előnye mellett van egy hátránya is. A Main Boardon kialakított csatlakozók az előre megtervezett fix funkciókhoz tartozó vezetősávokat szolgálnak ki. Tehát a piggy perifériák főbb jellemzői már gateway tervezésekor eldőlnek, amelyeket nem lehet megváltoztatni, csak a Main Board újratervezésével. Viszont a kommunikációs protokollok folyamatos fejlődésével az egyes szabványok olyan új tulajdonságokkal is bővülhetnek, amelyek megvalósításához a buszt illesztő áramköröknek szükségük lehet teljesen új részegységek hozzáféréséhez. A munkám során felhasznált, a CAN Partial Networking szabványt is már megvalósítani képes transceiver az előző eszközhöz képest két új elemmel bővült: SPI kommunikációval, illetve a mikrokontroller tápellátását vezérlő egységgel.

A Partial Networking alkalmazásához a CAN adó/vevőnek szüksége van előre beállított feltételekre, amelyek alapján szűrni képes a busz forgalmát. A feltételek beprogramozását a mikrokontroller vezérli, a szükséges adatokat SPI vonalakon juttatja el a transceivernek. A gateway tervezésekor a mikrokontrollerek és a különféle buszokat illesztő áramkörök között be lettek kötve az SPI vonalak. Ez a döntés az SPI beágyazott rendszerekben való elterjedtségével indokolható, nagy valószínűséggel felhasználható majd a jövőbeli újabb piggy-k tervezésénél (például az általam tervezett hardver esetében is).

Továbbá a TJA1145 transceiver rendelkezik egy a mikrokontroller tápellátását vezérlő kimenettel, INH lábbal (inhibit). A Partial Networking lényege, hogy az ECU-t

minél inkább kis áramfogyasztású állapotban tartsa a hálózat, ez nemcsak a vezérlőegység belső működésével érhető el, hanem a külső tápforrás korlátozásával is kiegészíthető (lásd 16. ábra).



16. ábra: A transceiver akár az ECU tápját is vezérelheti

Az e funkciót elősegítő kialakítást viszont nem tartalmazza a gateway. Ennek több oka van:

- Nincs szükség a gateway fogyasztásának csökkentésére. Ezt az eszközt vagy irodai környezetben használják, ahol a tápellátás állandó, vagy autókban, ahol az autót meghajtó rendszer biztosítja az elegendő energiaellátást. Az utóbbi esetben az autó belső rendszeréhez csatlakozik a gateway, így a jármű kikapcsolásakor a terepbuszillesztő is kikapcsol, nem merítve az autó belső tartalékait. Illetve, normál működés közben sem kiugróan magas a fogyasztása, csupán teszteléshez használt eszközként ez nem releváns. Így nincs szükség e funkcióra magához a gateway használatához. Viszont szükség lesz rá mint tesztelendő funkcióra a későbbiekben.
- A gateway több CAN buszt illesztő egységet tartalmaz (és több más buszt illesztő egységet is). Ha minden egység esetén lehetővé tették volna a közös mikrokontroller tápjának szabályozását, akkor sokszor ellentétes érdekek küzdöttek volna a számukra megfelelő szabályozás elérésért.
- Az utolsó ok pedig az, hogy a tervezés folyamán még nem számítottak, nem számíthattak erre a konkrét lehetőségre. Viszont a nagy valószínűséggel bekövetkező hasonló esetek (nem várt új funkciók megjelenése) kezelése miatt a csatlakozókon szabad lábak lettek kialakítva, amelyeknek nincs kijelölt

funkciójuk. Ezen lefoglalt lábak pedig a Main Boardon elhelyezett mikrokontrollerek általános célú ki- és bemeneteire lettek bekötve (General Purpose Input OutputO, GPIO). A MCU így hozzáférhet ezekhez és feldolgozhatja a transceiver speciális kimeneteit, még ha azok így nem is az eredeti feladatukat végzik el. Ez segíti a bármilyen, jelenlegi gateway által nem kiszolgálható funkció megismerését, tesztelését. Majd ha ez megtörtént, akkor az új funkcióval szerzett tapasztalatok segítségével továbbfejleszthető a Main Board.

A fentieknek megfelelően a transceiver INH lábát a mikrokontroller GPIO lábainak fenntartott csatlakozók egyikébe kötöttem be, ezáltal az adó/vevő nem befolyásolja a mikrokontroller tápellátását.

A CAN piggy hardver tervezésekor különösen nagy figyelmet kellett szentelnem a megfelelő huzalozás kialakítására. Az általam elkészített hardver, az autóipari környezetben gyakran elterjedt nagysebességű CAN szabványt használja (max. 1 Mbit/s), így ennek megfelelően kellett ügyelnem az érintett vezetősávokra. Ezek a vezetősávok a CAN protokoll kommunikációs vonalai, amelyek a busz és a transceiver közötti (transceiverbeli CANH, CANL lábak), valamint a transceiver és a mikrokontroller között (transceiverbeli TxD, RxD lábak) helyezkednek el. A hardverterv elkészítésének kezdetekor e nagyfrekvenciás vonalak ideális elhelyezése volt az elsődleges szempont. Hogy minél kevesebb zavart hasson a vonalakra, illetve minél kevésbé érje külső befolyás a CAN vezetékeit, igyekeztem a pályájukat a lehető legrövidebbre tervezni. Továbbá a vezetősávok töréseit (kanyarjait) minimalizáltam, így csökkentve a vonalakon fellépő késleltetést. További késleltetéscsökkenést szolgál a kialakított vezetősávok oldalág-mentessége. Oldalágat például akkor kellene használni, amikor egy-egy alkatrészt (tipikusan via-t, teszt pontot) a már sűrűn felhasznált területre akarunk bekötni. Ekkor az alkatrészt csak távolabb tudnánk elhelyezni, és vezetékkel a megfelelő helyhez vezethetnénk. Ez egyáltalán nem optimális, előrelátó tervezéssel elkerülhetőek a hálózat e felesleges ágai. A hardver vezetékezését éppen az említett okok miatt újra kellett kezdeni. Ez egy NYÁK tervezésekor lényeges szempont, amely nem hagyható figyelmen kívül, főleg nagysebességű vonalak esetében.

5.3.2. Alkatrészek

A tervezési követelmények ismertetésének végén pedig az alkatrészek által támasztott néhány követelményt ismertetem.

A tervezett piggy központi egysége a transceiver, amelynek kiválasztása meglehetősen egyszerű volt, mert a CAN Partial Networkinget megvalósító transceiverekből a tervezés kezdetekor szerény választék volt elérhető. Ennek oka az, hogy – mint ahogy korábban említettem – CAN PN szabványa még újnak tekinthető, emiatt korlátozott a választási lehetőség.

A többi alkatrész teljesen szabványos, általánosan használt, a csatlakozók kivételével felületszerelt elemek (Surface Mounted Device - SMD). Egyedül a méretükre kellett ügyelnem, mert a panelre nem géppel kerülnek beültetésre, hanem kézzel beforrasztva. Emiatt egy kicsi, de mégis jól látható, kézzel megfogható méretet (főleg az ellenállások és kondenzátorok esetén) alkalmaztam, amely a 0603 tokméret volt.

5.4. Tervezés kivitelezése

A tervezés különböző részfolyamatait időbeli sorrendben mutatom be, egészen a gyártás megrendeléséig. Miután az előzőekben ismertetett követelmények főbb pontjait egyeztettem a céggel, szükség volt egy megfelelő tervezőprogram kiválasztására.

5.4.1. A tervezőszoftver

A NYÁK elkészítéséhez mindenképpen szükséges egy tervezőprogram használata. Nemcsak amiatt, mert kényelmes, gyorsítja a munkát, illetve segíti az elkövetett hibák kiszűrését, hanem mert elengedhetetlen a gépi gyártáshoz. A gyártó számára szükséges a megtervezett áramkör digitalizált formája, amelyből minden apró információt felhasznál a gyártáshoz, kezdve a vezetősávok paramétereivel, egészen az alkatrészek pozíciójáig. A nyomtatott áramkörgyártásban elterjedtté vált az úgynevezett Gerber-fájl formátum, amely segítségével a legtöbb géppel előállítható a kívánt NYÁK, annak paramétereit tartalmazza. E Gerber-fájl formátumot a legtöbb tervező szoftver képes generálni, miután a szoftver keretein belül az áramkör elkészült.

A munkám során használt programot a konzulensem javasolta. Ez a program egy népszerű online tervezőprogram, az Upverter ([7]). Kiválasztásában a fő szempont

az volt, hogy minden szükséges funkciót képes kezelni, azaz megtervezhető vele a kapcsolási rajz (schematic), valamint a konkrét legyártandó nyomtatott áramköri elem egésze is (layout). Emellett természetesen feltétel volt az ingyenessége is, valamint előnyére vált az online hozzáférhetősége.

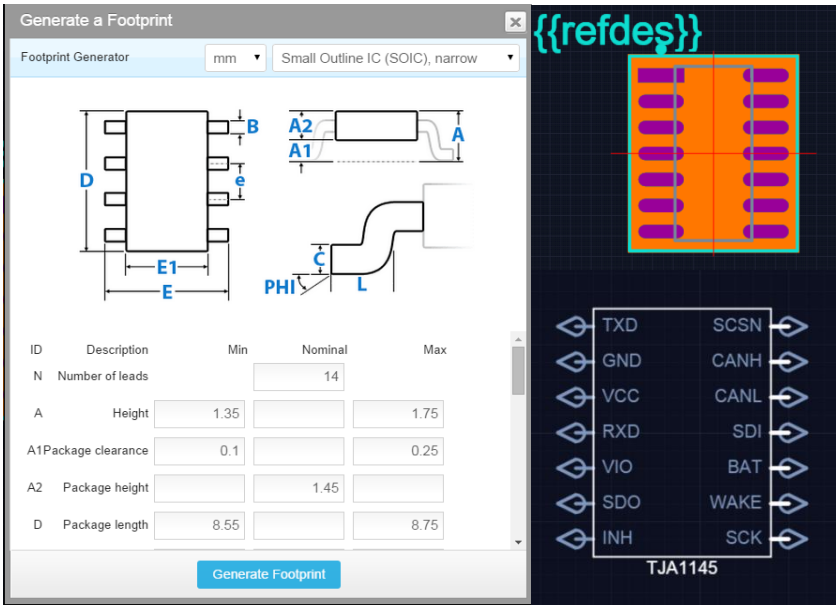
Az Upverter által kezelt program erőssége a nyíltsága. Ez a nyíltság megmutatkozik többek között a projektek hozzáférhetőségén. Az itt elkészített kapcsolások elérhetővé válnak mások számára, sőt, ha engedélyezve van, akkor akár más is szerkesztheti a projektet. Számomra ez teljesen megfelelt, az általam elkészített hardver nem titkos. Ezenkívül, az alkatrészek adatbázisa is bárki számára elérhető, szerkeszthető. Az különböző elemeket tároló adatbázist akárki szabadon bővítheti egy-egy alkatrésszel, akár az alkatrészhez tartozó különböző információval is (ábra, lábkiosztás, alkatrész rajzolat, alkatrész dokumentációja stb.). Emellett a már a rendszerben lévő elemek módosíthatóak is, újabb verzió szerkeszthető belőlük, könnyítve ezzel a bázis naprakészen tartását.

5.4.2. A szükséges alkatrészek

Bármennyire is egyszerűen növelhető az Upverter adatbázisa, a feladatomhoz hiányzott pár szükséges alkatrész, amelyekkel ki kellett bővíteni az adatbázist.

A hardver központi magját adó transceiver nem szerepelt az Upverter rendszerében, csak korábbi verziók. Ez valószínűleg azzal függhet össze, hogy még nagyon új a transceiver, nem rég jelent meg a piacon. Tehát létrehoztam az adatbázisban egy új bejegyzést, kitöltöttem az információs mezőket (lábkiosztás, funkció leírása, gyártó dokumentációja). Majd az elemhez tartozó ábrát is megszerkesztettem, a gyártótól származó dokumentáció alapján a lábkiosztást is berajzoltam. Ezek után kész volt a kapcsolási rajzhoz szükséges ábra. Viszont hiányzott még az alkatrész rajzolata (footprint), ami mindenképpen szükséges a megfelelő layout előállításához. A footprint (lábnyom) egy az adott alkatrészhez tartozó, forraszpasztából kialakított alkatrész lenyomat, amely a kontaktus felületeket alakítja ki a forrasztandó elem és a felület között. Emiatt nagyon pontosan kell azt létrehozni. A gyártó honlapján megtalálható az alkatrész pontos dokumentációja, amelyben minden szükséges adat fellelhető (pl.: alkatrész hossza, szélessége, magassága, lábak közti távolság, tokozás). A pontosan ugyanolyan tokozás kiválasztása és az adatok (bizonyos tűréshatárral) való

megadása után lehet generálni az alkatrészhez tartozó footprintet és felhasználni a projektben (lásd. 17. ábra).



17. ábra: Alkatrész létrehozásainak lépései

A transceiveren kívül szükséges volt a piggy szélén elhelyezkedő csatlakozósorok kialakítása is. Köszönhetően az Upverter adatbázisának, ez már a rendelkezésemre ált, a ThyessnKrupp egyik munkatársa már korábban létrehozott egy hasonló alkatrészt (feltételezem a korábban megtervezett piggy-hez). A csatlakozók eleme úgy lett kialakítva a furatok pontos pozícionálása miatt, hogy egy nagy, a teljes panellel megegyező méretű, azt lefedő alkatrész lett létrehozva, amin megtalálhatóak a megfelelően kimért helyeken a furatok. Ezt a többi piggy projektben egy közönséges alkatrészként kell használni, ahol a más áramköri elemekkel az átfedések megengedhetőek, kivéve természetesen a furatok helyén. Tehát egyszer kellett csak pontosan beállítani a furatok pozícióját, és felhasználva ezt az elemet, többé már nem kellett ezzel foglalkozni. Ehhez a projekthez is megfelelt ez a megoldás, egyedül a csatlakozók neveit kellett átszerkeszteni, kreálva így egy újabb verziót az elemből.

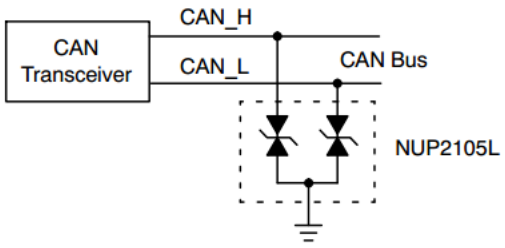
5.4.3. Kapcsolási rajz (schematic) elkészítése

A főbb elemek létrehozása után, a kapcsolás kialakítása következett. A kapcsolási rajz különböző funkciókat megvalósító részeit egyesével mutatom be.

5.4.3.1. CAN busz szűrő

A CAN busz szűrő egység feladata a CAN buszon beérkező, a transceiverbe befutó analóg jelek szűrése, illetve a transceiver védelme a különböző feszültségtranziensektől.

A szűrő áramkörbe befutó CAN vonalakat (CAN_H és CAN_L) először egy velük párhuzamosan kapcsolt, nagy feszültségimpulzusokat elvezetni képes integrált áramkör szűri. Ez az IC mindkét CAN vonalra egy-egy úgynevezett szupresszor diódával (Transient Voltage Suppression, TVS diode) csatlakozik (lásd 18. ábra). E diódák feladata a nagyobb feszültségtüskék (pl. elektrosztatikus kisülés okozta tranziensek) földbe vezetése, védve így az érzékenyebb mögöttes áramköröket. A szupresszor diódákat kizárólag erre a célra tervezték, a hosszabb ideig tartó nagy feszültséget már nem viselik el. A karakterisztikájukból adódik, hogy egy bizonyos feszültségszint alatt záróirányúak, nagy ellenállású eszközként viselkednek, ezáltal nem befolyásolják az áramkör működését. Viszont, ha a vonalon megjelenő feszültség eléri az úgynevezett „megszólalási értéket”, akkor nyit a dióda, így korlátozva egy elfogadható szintre a feszültséget. Miután a vonal feszültségszintje újra normál értékűre redukálódott, a szupresszor dióda ismét zár, függetlenítve magát az áramkörtől [8]. Az áramkörömben a direkt nagysebességű CAN buszokra specializált ON Semiconductor által gyártott NUP2105LT1G szűrőt választottam.



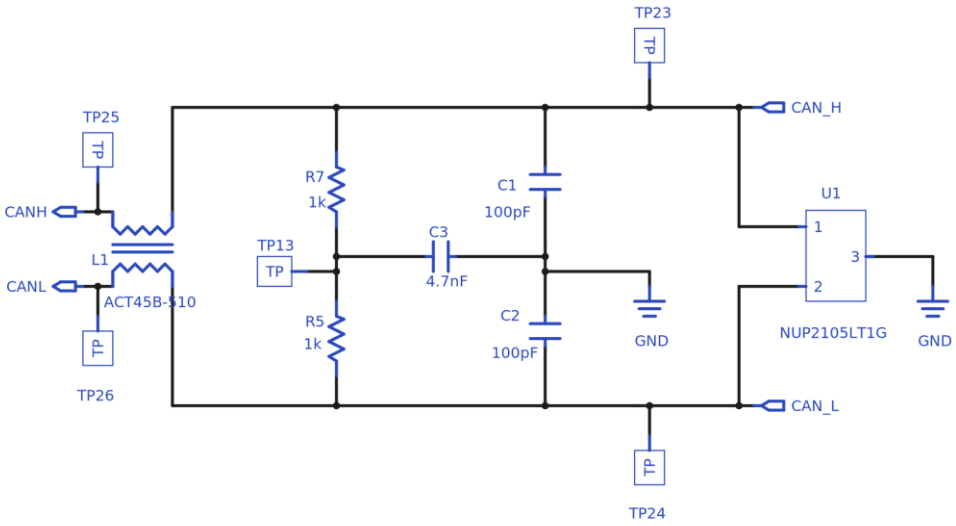
18. ábra: TVS diódás szűrő

A transceiver védelmét nem kizárólag a TVS diódák végzik el, hanem ezt a feladatot megosztják több kondenzátorral, növelve a rendszer a pontosságát, megbízhatóságát. A kondenzátorok feladata olyan kisebb feszültségtranziensek kiszűrése, amelyek a szupresszor diódás szűrő megszólalási értéke alatt vannak.

Végül mielőtt a CAN vezetékek csatlakoznának a transceiverhez, sorosan kapcsolódnak egy közös módosú szűrőhöz. Ez a szűrő a vonalon jelenlévő közös

módosú zajokat képes eltávolítani, azaz mindkét jelvezetéket egyformán befolyásoló zajokat. A működése egy közös vasmagon alapul, amire a vezetékek fel vannak tekercselve. A vezetékek úgy vannak irányítva, hogy a rajtuk futó áram ellentétes irányú fluxusokat hozz létre. Az ellentétes fluxusok egymást gyengítik, különbségük adja a zajmentes jelet, így a vezetékeken egyformán jelenlévő közös módosú zajok kiesnek. Ez az egyszerű megoldás hatékonyan szűr, ráadásul az áramköri elem ellenállása sem számottevő. [9][9] A hardveremben az Epcos cég ACT45B típusú, CAN buszokhoz használható SM alkatrészét használtam fel.

Az előzőekben részletezett elemek felhasználásával a CAN busz szűrőt a ábra szerint készítettem el.



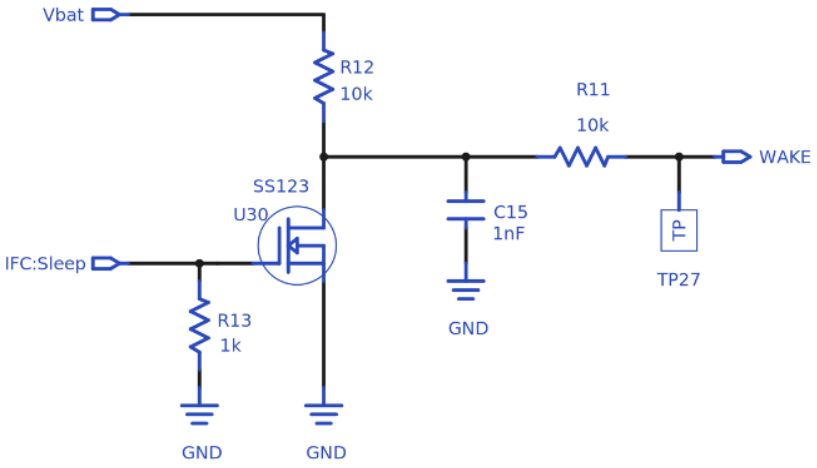
19. ábra: CAN busz jeleit szűrő kapcsolás

5.4.3.2. Local Wake-up bekötése

A transceiver nem csupán a busz forgalmától függően képes felébredni, hanem lehetőség van az MCU által vezérelt felébresztésre, lokális wake-up-ra is. A felébresztés parancsa egy dedikált vezetéken érkezik a mikrokontroller felől, a piggy egységhez a csatlakozókon keresztül jut el. A csatlakozó IFC:SLEEP lábának értéke határozza meg, hogy a transceiver (a WAKE lábán keresztül) kap-e felébresztést kiváltó parancsot. A szükséges feltétel az eszköz programozásával állítható, azaz a vezetéken jövő 1 vagy a 0 érték váltson-e ki felébresztést.

A transceiver WAKE bemenetének vezérlését egy kapcsolóként működő FET látja el (vezérlésre a FET gyakran használt, mert nagy a bemeneti ellenállása, és egyszerűen előállítható). Ahogy azt az általam elkészített kapcsolásban látható (20. ábra), a mikrokontrollertől jövő IFC:SLEEP vonal értéke dönti el, hogy a kapcsoló nyitva van-e, vagy sem. Ha nyitva van, akkor a WAKE lábra nem kerül feszültség, viszont ha zárva van, akkor WAKE lábat az áramkör az akkumulátor feszültségszintjére húzza fel.

Természetesen szükséges az akkumulátoros táp szűréséhez egy kondenzátor, valamint a FET és a transceiver védelme érdekében különböző védőellenállások is nélkülözhetetlenek. Mindehhez a transceiver dokumentáció adott útmutatást, amely többek között tartalmazza WAKE láb ajánlott bekötését. [6]

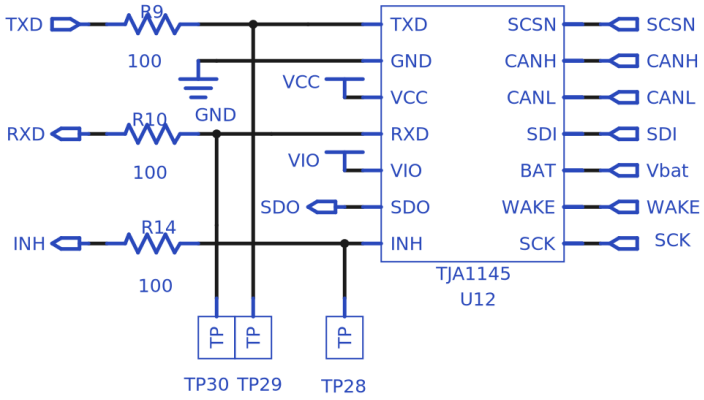


20. ábra: Lokális felébresztés vezérlő áramkör

5.4.3.3. Transceiver lábainak bekötése

Ebben a fejezetben a TJA1145 transceiver további lábainak bekötését ismertetem (eddig a CANH, CANL és a WAKE láb bekötése szerepelt a szakdolgozatomban).

Az elkészített kapcsolásból látható (21. ábra) látható, hogy az SPI kommunikáció által megkövetelt vezetékek, azaz az SDO (SPI data out), SDI (SPI data input), SCSN (SPI chip select input) és SCK (SPI clock input) bekötése a csatlakozók megfelelő lábaival való összekötésén kívül nem igényelt egyéb tervezést. A felsorolt vezetékek az SPI kommunikációs protokoll szabványos vonalai, a piggy csatlakozóin is ugyanezek találhatóak meg.



21. ábra: TJA1145 lábainak bekötése

A transceiver és a mikrokontroller közti kommunikáció (digitális CAN üzenetek formájában) az RXD (receive), valamint a TXD (transmit) vonalakon keresztül történik. Ezek a vezetékek a csatlakozón szintén megtalálható, hasonló elnevezésű lábakba futnak. A csatlakozók e lábai – mint az SPI esetében is – közvetlenül a mikrokontrollerhez csatlakoznak. A transceiver nagyon fontos részei ezek a vezetékek, hiszen annak alapvető funkcióját valósítják meg, digitalizált formában továbbítják (ha a feltételek megengedik), illetve fogadják a CAN üzeneteit.

A transceiver védelme érdekében 100 ohmos védőellenállásokat alkalmaztam egyes lábaknál. Ezeknek az a feladatuk, hogy egy esetleges hiba esetén (pl. rövidzár) fellépő nagy áramerősség ellen védje az IC bemeneteit, hiszen ebben az esetben feszültség esik rajta.

Az INH láb bekötését a korábbiakban már részletezett módon valósítottam meg, a csatlakozó egy általános célú kimenetére vezettem ki (lásd 22. ábra).

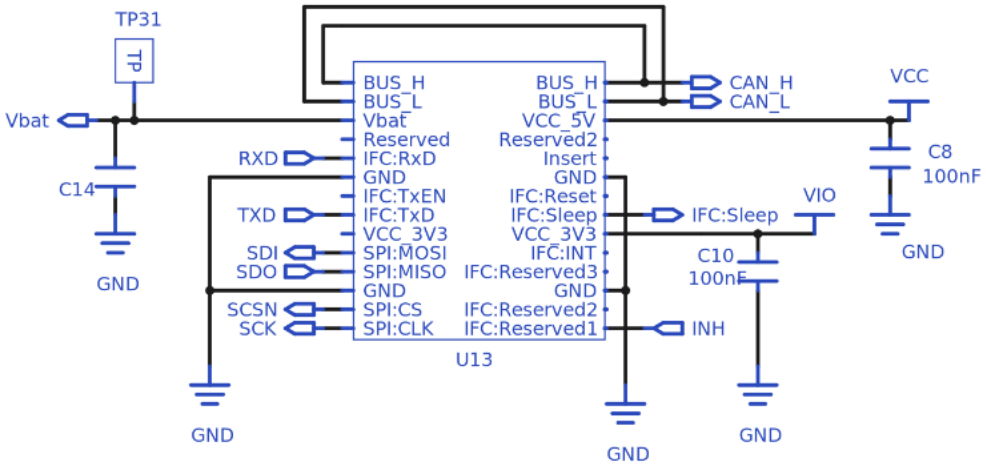
A különböző tápfeszültségek kezelését (szűrését) rögtön a csatlakozó környezetében valósítottam meg, ezzel is minél inkább védve a nagyobb feszültségek elől a transceivert.

5.4.3.4. Csatlakozók bekötése

A csatlakozó lábainak bekötésekor a CAN busz vezetékeinek átkötésén kívül nem kellett semmi újabb részletre odafigyelni. A CAN vonalak átkötését, azaz a transceiver párhuzamos csatlakoztatását a hálózathoz a busz jelleg indokolja.

A transceivertől kijövő vonalakat egyszerűen be kellett kötni a csatlakozó megfelelő lábához (amelynek ráadásul az elnevezése is hasonló). A tápok szűrése pedig semmiben sem különbözik a CAN busz jeleinek szűrésétől, az oka és a megvalósítása is hasonló.

A csatlakozók bekötését a 22. ábra tartalmazza.



22. ábra: Csatlakozók bekötése

5.4.4. Layout elkészítése

Miután a schematic elkészült, azaz az elemek és a vezetékhálózat rendelkezésre áll, elkezdhető a hardver konkrét tervezése.

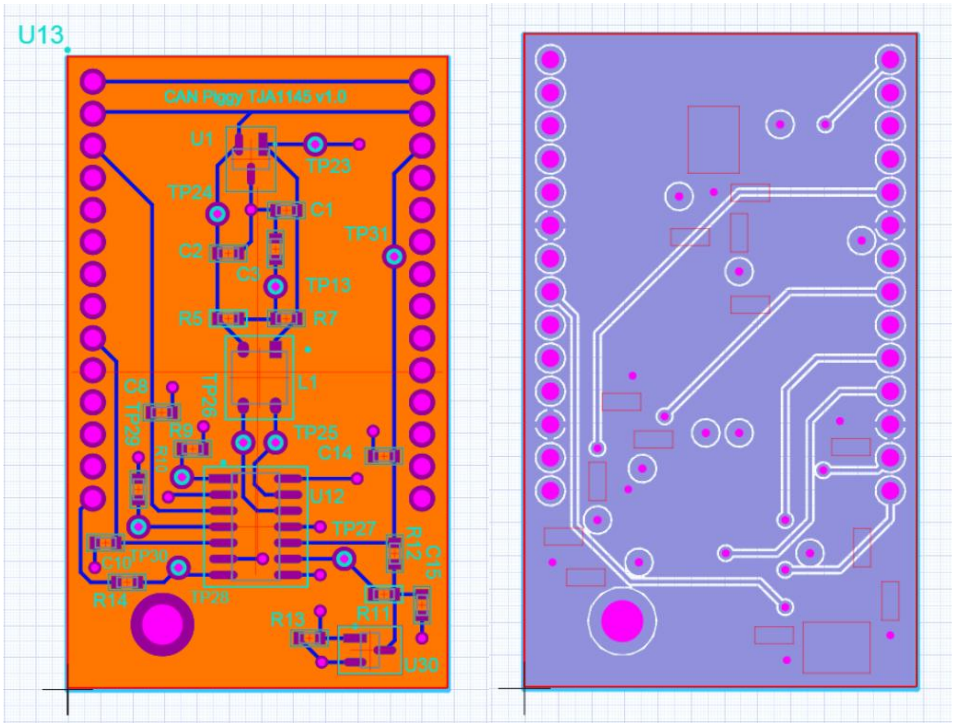
Legelőször a nyák felületén elhelyezkedő áramköri elemek elhelyezését (pozíció, forgatás) kell meghatározni, amit a kapcsolási rajzban foglalt összeköttetések nehezíthetnek, majd ha a koncepció elkészült, akkor a vezetősávok összekötése elkezdhető. Úgy gondolom, hogy a layout elkészítésének ez az igazi nehézsége, hiszen úgy kell számos követelményt kielégíteni, hogy mindkét oldalon futnak vezetősávok, vannak kétoldalas elemek (via, teszt pont), illetve a technológiai korlátok miatt bizonyos távolságokat kell tartani az alkatrészek/vezetősávok között.

Miután elkészültek az alkatrészeket összekötő vezetősávok, a layout befejezése előtt már csak a NYÁK hátsó oldalán kellett dolgozni. A NYÁK teljes hátsó oldala földpotenciálon van, ez egy bevett tervezési fogás, ami az alkatrészek földelését megkönnyíti. Így földelés létrehozásához elegendő csak a földelendő elemet egy vián keresztül összekötni a panel hátoldalával.

A hátoldal földelése úgy néz ki, hogy a hátsó vezetősávokat, teszt pontokat, valamint a nem földpotenciálú viákat és csatlakozókat kivéve minden felület (bizonyos távolságot tartva ezekhez képest) azonos potenciálú réteggel van bevonva (pour). E réteg pedig a csatlakozó föld potenciálú lábaival kontaktusban van, ezáltal válik a hátoldal földeltté (lásd 23. ábra).

Mivel ez a réteg a teljes hátoldalt kitölti (ahol lehet), kézzel nehézkes lenne kialakítani. Az Upverter programja viszont képes ezt legenerálni, csupán a más potenciálú felületek közti minimális távolságot és a felület potenciálját kellett beállítani. Ezek után legenerálható a hátsó földréteg, amely így automatikusan elkészül.

A 22. ábra prezentálja az elkészült layout alsó és felső oldalát.



23. ábra: Az elkészült layout felső és alsó rétege

5.4.5. Gyártatás

E lépés kezdetekor már digitalizált formában elérhető a megtervezett hardver, az Upverter program keretein belül, egy kész projektben. Az egyedi gyártást a ThyssenKrupp Presta Kft. egy külső partnere végezte el.

A gyártó által megvalósítható gyártási adatokat még a layout készítése folyamán tisztázni kellett, mivel ez befolyásolja a NYÁK tervezését. Ilyen információ a

huzalozás elérhető minimális vastagsága, a minimális szigetelő távolság (ez a vezetékek közti legkisebb távolságot, valamint a hátoldali föld és a hátoldali vezetékek minimális távolságát határozza meg). A layout elkészítésekor még egy dologban kell előre gondolkodni a nyák későbbi gyártásának megkönnyítéséhez. Ez pedig a metrikusan egész átmérőjű furatok használata, ha lehetséges. Sokszor nincs szükség egy konkrét furatátmérőre (például viák esetében), így kompromisszumok nélkül megkönnyíthető a gyártás. Ekkor nem szükséges több különböző fúrással kialakítani azt a furatot, elegendő csak eggyel.

A gyártás megkezdéséhez csupán a kész projekthez tartozó Gerber-fájlokat kell legenerálni és eljuttatni a gyártónak. A külső partner csupán a nyák gyártását csinálta meg, azaz egy szabványos típusú platformra a vezetősávokat, a forrasztandó felületeket és a különböző furatokat készítette el. Ezek a részeket gépekkel alkották meg. Az alkatrészek beültetését már a ThyssenKrupp-on belül végezték el, kézi forrasztással.

6. Gateway szoftver oldali felépítése

6.1. Áttekintés

Ismerve a gateway szerkezetét, belátható, hogy sok különféle funkció megvalósítására képes. Hogy e multifunkcionális jelleg kihasználható legyen, valamint igazán hasznos fejlesztőeszközzé váljon, szükség van egy olyan szoftverarchitektúrára, amely segítségével az eszköz perifériái gyorsan kiszolgálhatóak (párhuzamosan kezelhetőek) és az egyszerű fejleszthetőség miatt moduláris felépítésű. Ezen követelmények teljesítése szükségessé teszi egy operációs rendszer használatát.

A gateway sokoldalúsága miatt könnyen előfordulhat, hogy a különböző perifériákon több párhuzamos, adott időn belüli vezérlést igénylő esemény játszódik le. A gateway-beli mikrokontroller ezekre csak akkor tud a megfelelő időn belül reagálni, ha az idő szempontjából legszükségesebb lépéseket elvégzi egyenként az összes periférián. Ehhez az operációs rendszer a különböző modulok, perifériák kezelését elkülöníti (ezeket az elkülönített egységeket, feladatokat nevezik taszkoknak). Ezen taszkok közötti gyors váltásokkal érhető el a feladatok párhuzamosnak látszódó, kis késleltetésű végrehajtása. Ehhez szükséges a különböző taszkok megfelelő ütemezése is. A beágyazott operációs rendszer ezeket a funkciókat valósítja meg. Az ütemezés mellett az operációs rendszer kezeli még a közös erőforrásokat is, valamint az egyes részfolyamatok közötti információ megosztását is irányítja.

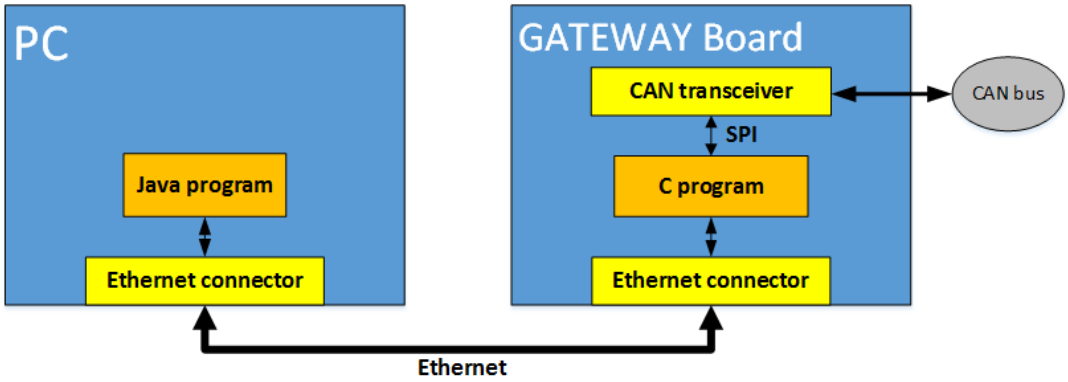
Emiatt a gateway alapvető működését a beágyazott operációs rendszer határozza meg. Az eszköz folyamatosan futtat egy beágyazott operációs rendszert (FreeRTOS), amely fogadja és feldolgozza a kívülről érkező információkat. A bejövő adatok származhatnak az egyes perifériákon keletkező eseményektől (pl.: CAN üzenet érkezése), valamint a fejlesztői PC-ről (pl.: CAN üzenet kiküldésének parancsa).

A gateway hardverén futó operációs rendszert ki kell egészíteni különböző, feladatspecifikus modulokkal (pl.: transceiver felügyelete, kezelése). Beágyazott rendszerről lévén szó, a gateway programozása természetesen C nyelven történik.

A számítógép oldalán pedig létre kell hozni egy olyan programot, amellyel Ethernet kapcsolaton keresztül utasítások adhatóak ki a gateway-en futó programnak, amely így

működés közben vezérelhetővé válik. A PC-beli program pedig a futtatókörnyezettől való függetlenséget nyújtó Java nyelven íródott.

Szakdolgozatomban a fentebb taglalt, már meglévő szoftverarchitektúrát egészítettem ki az új CAN transceiver vezérléséhez szükséges funkciókkal.



24. ábra: A CAN transceiver vezérlése

Mivel az új CAN transceiver – az eddigiekkel ellentétben – már programozható, kiegészült a fentebb leírt vezérlési folyamat egy újabb lépéssel. Ha a CAN PN megvalósítására képes transceivert szeretnénk elérni, akkor ahhoz a személyi számítógépen kell a megfelelő parancsot kiadnunk. A számítógép feldolgozza azt, a szükséges paramétereket a paranccsal együtt Ethernet segítségével elküldi a gateway-nek. A gateway-en futó beágyazott operációs rendszer ezt feldolgozza és ez alapján meghatározott SPI üzenetekkel felprogramozza a transceivert. Látható, hogy a CAN PN integrálása mind a beágyazott szoftver szintjén, mind a PC szintjén megvalósult, teljesen beolvadt a már használt szoftveres környezetbe.

6.2. PC-Gateway kapcsolat szoftveres megvalósítása

Mint azt korábban részleteztem (PC-Gateway kapcsolat hardveres megvalósítása című fejezetben) a kommunikáció az Ethernet adatkapcsolati protokollon keresztül történik. A kommunikáció során használt üzenetek felépítését, illetve sorrendjét az IPv4 hálózati protokoll szabja meg. A kommunikáció egyszerűsége miatt bőven elegendő az IPv4 nyújtotta címzés.

Az üzenetküldést az UDP (User Datagram Protocol) szállítási protokoll végzi el. Azért az UDP használatára esett a választás, mert a kommunikáció a két egység között meglehetősen egyszerű felépítésű. Csupán arra van szükség, hogy gyorsan, előre látható

időn belül történjen az adatátvitel. Nincs szükség kapcsolat kiépítésére, nem követelmény a hibamentesség vagy veszteségmentesség biztosítása. Fejlesztéshez használt rendszerről van szó, ha bármilyen probléma lép fel az adatátvitelben, akkor a felhasználó egyszerűen újraküldi az üzenetet. Ezenkívül, a kommunikáció is csak két csomópont között történik, nem kell útvonalat tervezni, adatokat darabolni. Az UDP pontosan ezt nyújtja, az adatcsomagokat kiküldi, és a továbbiakban nem foglalkozik velük.

A továbbiakban bemutatom az általam létrehozott beágyazott és PC modulok közti információcsere felépítését, működését.

Habár mindkét irányban az UDP szállítási protokoll szerinti forgalom van, az adatcsomagokban lévő adatstruktúra eltérő felépítésű. A gateway PC oldali vezérlése úgy valósul meg, hogy a számítógép elküldi a feladatot végrehajtó függvény azonosítóját és a függvény futásához szükséges paramétereket a gateway-nek. Ezeket a gateway fogadja, az egyes paramétereket sorban feldolgozza, és ezek alapján futtatja a kiválasztott függvényt. Miután a gateway-ben meghívott függvény lefutott, visszaadja a megfelelő visszatérési értékeket. A PC-nek küldött válasz eltérő felépítésű, csupán a visszatérési értéket tartalmazza, ezenkívül semmi egyéb azonosítót (nincs is rá szükség, hiszen ismert, hogy mi futott le a gateway-en).

Az adatok pontos feldolgozását, illetve kódolását a beágyazott oldali szoftver működésének leírásában (7.1 fejezetben) ismertetem.

6.3. Debugger, tesztelés

Munkám során az elkészült programelemeket, modulokat folyamatosan tesztelnem kellett. Ez szükséges volt, hiszen egymásra épülő egységekkel dolgoztam, munkám teljesen átöleli a gateway vezérlésének lépéseit, a felhasználó döntésétől kezdve a transceiver felprogramozásáig minden folyamatot. Így a fejlesztés során szükség volt a magas szintű függvényhívások (PC oldal), a kódolt/dekódolt üzenetek, valamint a transceiver megfelelő felprogramozásának folyamatos ellenőrzésére is. Mindezeket a különböző függvények műveleteinek és a különböző változók értékeinek rendszeres kiolvasásával valósítottam meg.

PC (azaz Java) oldalon ez nem igényelt különösebb előkészületet. Viszont a beágyazott oldal működésének nyomon követése igen. A fentebb sorolt műveletek követéséhez szükség volt egy külső debuggerre, amely megmutatta az éppen aktuálisan futó függvényt, annak változóit, értékeit. Habár a gateway állapotának megfigyelésével a felprogramozott transceiver regisztereinek értékeit közvetlenül nem tudtam kiolvasni, mégis az SPI üzenetek, illetve a transceiver dokumentációjának segítségével könnyen lekérdezhettem a regiszterek értékeit.

A gateway-en működő beágyazott rendszer tulajdonképpen a gateway-be épített egyik mikrokontrolleren fut. Ez a mikrokontroller a Texas Instruments (TI) terméke, amely komplett infrastruktúrát kínál eszközeihez, így a debuggoláshoz elengedhetetlen komponensek kiválasztása, előteremtése zökkenőmentes volt. A mikrokontroller debuggolásához szükséges volt egy Eclipse alapú, TI által módosított szoftver keretrendszerre (Code Composer Studio), amelynek feladata a felhasználói interfész biztosítása, illetve a debug eszköz illesztése. Ezenkívül szükség volt egy JTAG (Joint Test Action Group) alapon működő eszközre is (a gateway-en JTAG van kialakítva), ami a mikrokontroller memóriáját olvassa ki. Sok eszköz megfelel a célnak, én egy viszonylag olcsó, ARM alapú mikrokontrollereket kezelni képes eszközt használtam, amely az XDS100v2-ARM volt.

7. Beágyazott oldali szoftver elkészítése

7.1. Működés

A gateway-en futó, beágyazott oldali szoftver feladata a perifériák kezelése. Működésének célja a perifériákon történő események megfelelő időn belüli fogadása, feldolgozása, illetve megválaszolása. Mivel az ezt megvalósító szoftverarchitektúrának a gateway-be épített mikrokontrollerek egyikén kell futnia (lásd 3.2.1. fejezet), célszerű egy olyan magas szintű programozási nyelvet használni erre a célra, amely közvetlen memóriakezelést biztosít (valamint amely természetesen széleskörben elterjedt). Ennek a kritériumnak tökéletesen megfelel a C programozási nyelv.

A külső események megfelelő kiszolgálása miatt a gateway-en egy beágyazott operációs rendszer fut. Az operációs rendszer a perifériákon fellépő, akár párhuzamos eseményeket különböző taszkok segítségével szolgálja ki. A CAN PN implementálása során nem volt szükség külön taszk létrehozására, mert a taszkok között egy külön, a PC-ről érkező utasításokat fogadó és végrehajtó taszk már korábbról rendelkezésre állt. Ebbe a taszkba kellett beilleszteni a kifejlesztett modult, illetve a modulhoz tartozó speciális PC parancsok értelmezését, megválaszolását is. Az elkészített, PC parancs feldolgozását végző szoftverrész a következőképpen működik:

A számítógépen kiadott parancs és a hozzá tartozó paraméterek egy strukturált felépítésű, 8 bites tömbként kerülnek elküldésre. A gateway ezen 8 bites elemekből álló tömböt fogadja, majd a feldolgozást végző taszk által alkalmazott modul (Commander) először különböző, egymásba ágyazott függvények láncolatával meghatározza, dekódolja a kiadott parancsot és a paramétereit. A dekódolás a kapott tömb elemein végigmenve pontosítja a végrehajtandó feladatot:

1. A Commander először megállapítja (`Commander_ProcessCommand()` függvény segítségével), hogy a parancs a gateway-en belül mely területre vonatkozik (a tömb 0. bitje tartalmazza ezt az információt). Ez lehet a PC-gateway kommunikációjára vonatkozó (*COM_COMM*), egyszerű CAN adó/vevőkre vonatkozó (*COM_CAN*), FlexRay-re vonatkozó (*COM_FR*), a második kontrollerre vonatkozó (*COM_MCU_B*), vagy az munkám során

használt, magasabb funkcionalitású CAN transceiverre vonatkozó (*COM_CAN_TRCV_TJA1145*) egyike. Miután egy switch-case szerkezettel a Commander dekódolta az érkezett tömb első elemét a fenti lehetőségek egyikeként (a zárójelbeli elnevezések enumokkal létrehozott azonosítókat takarnak, amelyek 8 bit hosszúak), akkor az adott területre vonatkozó újabb segédfüggvény hívódik meg, megkapva a parancsot tartalmazó tömb hátralévő elemeit. Prezentálva a saját munkámat a *COM_CAN_TRCV_TJA1145* esetet feltételezem a továbbiakban. Ez esetben a tömb feldolgozását, és a feladat további szűkítését a `commanderCAN_TRCV_1145Command()` függvény fogja elvégezni.

2. A fogadott tömb hátralévő elemeit megkapva a `commanderCAN_TRCV_1145Command()` kiválasztja a konkrét feladatot végrehajtó függvényt. Az ehhez szükséges információt a kapott kisebb tömb első eleme adja, amely – az előzőekkel azonosan – egy azonosítót tartalmaz. Ezen azonosító dekódolása szintén egy switch-case szerkezettel történik, a 8 bites kódnak esetemben 24 darab, különböző lehetőséget kell lefednie, mert 24 darab transceivert működtető függvényt hoztam létre (lásd Transceiver funkciók használata fejezet).
3. Miután sikerült a pontos feladat meghatározása, a hozzá tartozó paramétereket kell feldolgozni (a tömb maradék elemei alapján), illetve futtatni a parancsot végrehajtó függvényt. A paraméterek feldolgozása még a `commanderCAN_TRCV_1145Command()` függvényben, a switch-case szerkezet case ágában történik, értelemszerűen a parancs végrehajtása előtt. A paraméterek a legtöbb esetben egy adott értéket képviselnek, például logikai igazat/hamisat, egy-egy azonosítót, vagy számsort. Ezek esetében mindössze arra kell ügyelni, hogy a 8 bites tömb elemeiben elférjenek. Ha az adott szám nagyobb, mint 8 bit, akkor darabolással több tömbelembe kell helyezni az adatot, és mind a kódoló, mind a dekódoló oldalon helyes sorrendben kell kezelni őket.

   Előfordul viszont olyan parancs is, amikor a végrehajtó függvény paraméterként összefüggő adatokat, valamilyen definiált struktúrát vár. Szemben az egyszerűbb, önálló adatokkal, ahol csupán a tömb egyes

elemeit kellett elmenteni és átadni a végrehajtó függvénynek, itt a tömb (előző lépések után megmaradó) összes eleme valamilyen kombinációban adatokat tartalmaz. Ezeket egy további segédfüggvény kezeli (`commanderParseStruct()`), amely egy előre definiált struktúrát hoz létre és ismerve az elemek méretét, a tömbből sorrendben véve az adott méretek szerint kitölti a struktúrát, amivel majd visszatér. A létrehozandó struktúrákat előre definiált leíró (segéd)struktúrák írják le (pl.: `commanderCANTrcvDataDescriptor`), amelyek tartalmazzák: a létrehozandó/kitöltendő struktúra méretét (amely elemek méreteinek összegétől függ), az elemek típusait, illetve az elemek számát. Ezek alapján előre létrehozható és számítható, hogy a bejövő adatnak mit kell tartalmaznia.

Miután sikeresen végrehajtódott a felhasználói parancsot közvetlenül megvalósító függvény, szükség van a visszatérési érték (válasz) eljuttatásához a számítógépre. Amíg az előbb taglalt folyamatok lejátszódnak, a függvények, amelyekből az egyes segédfüggvények meghívódtak, várakoznak a segédfüggvények futásának befejezésére, esetleges visszatérési értékeikre. A számítógépen futó program szintén válaszra vár ez idő alatt. A Commander modul válaszának felépítése minden esetben azonos: a `commanderCAN_TRCV_1145Command()` függvény egy olyan struktúrát ad vissza, ami a végrehajtás sikerességét (`E_OK`, illetve `E_NOT_OK` enumok által), a visszatéréskor átadandó adatok hosszát, valamint az átadandó adatok számára egyben lefoglalt memóriaterület kezdőcímét tárolja. Így a válasz elküldése uniformizálttá válik, küldéskor egyértelműen kiolvashatóak a memóriából a szükséges adatok. Az adatátvitel során csupán arra kell ügyelni, hogy az elküldendő adatokat a lefoglalt memória kezdőcímétől sorrendben küldi ki a gateway, így fogadáskor is arra a sorrendre kell számítani, amellyel elhelyeztük az adatokat gateway memóriájában.

7.2. Transceiver funkciók használata

Ebben a fejezetben ismertetem a transceiver által biztosított funkciók megvalósítását, ehhez felhasználva az eszközhöz tartozó dokumentációt ([6]). Sok ilyen funkció létezik, hiszen az új, integrált CAN adó/vevő programozható regiszterei sok beállítást, információt hordoznak. Mindezek előtt viszont nem a konkrét

regisztermű veleteket (és az ezáltali beállításokat) mutatom be, hanem a transceiver regisztereinek elérését.

A gateway mikrokontrollere a CAN adó/vevővel SPI vonalakon tartja a kapcsolatot. Ezért a modul működéséhez elengedhetetlen volt egy, az SPI kommunikációt irányító másik modul (SPIDriver) használata. Az SPIDriver egységből csupán két, de nagyon fontos funkciót használtam fel: az SPI kommunikáció inicializálását, illetve természetesen az SPI üzenetküldését.

A kommunikáció inicializálásakor be lehet állítani, hogy a küldés/fogadás az adat legfelső bitjével kezdődjön-e vagy a legalsóval; a küldés/fogadás az órajel fel- vagy lefutó élére történjen-e; mekkora lesz az átküldendő adat mérete, valamint természetesen a küldést vezérlő órajel frekvenciája is állítható. A TJA1145 transceiverrel való SPI kommunikáció csak bizonyos, előre létrehozott paraméterekkel képes működni, amelyet a transceiver dokumentációja tartalmaz, ezáltal az SPI megfelelő inicializálása elvégezhető.

Ha az inicializálás megtörtént, akkor lehetővé válik az üzenetküldés. Mint arról korábban szó volt, a gateway egyszerre több transceivert (piggy-t) is képes kezelni. A transceiverek könnyed variálhatósága miatt minden piggy-hez tartozik egy külön SPI vonal (úgynevezett channel), emiatt küldés előtt ki kell választani, hogy melyik eszköznek szeretnénk elküldeni az adott üzenetet. Ezt az SPI szabványa által megkövetelt chip select vonal határozza meg. A chip select értékét a felhasználó dönti el. Ez egy olyan fix paramétere lesz minden függvénynek, amelyet már a PC parancs kiadásakor meg kell adni, így a felhasználó eldöntheti melyik, a gateway-be helyezett adó/vevőt szeretné felprogramozni.

Emellett az üzenetek elküldéséhez természetesen szükség van magára az elküldendő adatra is. A transceiver az érkezett üzeneteket kétféleképpen értelmezheti: a transceiver regiszterét kiolvasó vagy a transceiver regiszterét író parancsnak. Mindkét esetben az üzenet 8 bittel kezdődik. Az üzenet első 7 bitje egy, az adó/vevőn belüli regiszter címét tartalmazza (íráskor e regiszter tartalma íródik felül, olvasáskor pedig visszaadódik), a 8. bit (úgynevezett read-only bit) pedig azt jelöli ki, hogy az aktuális üzenettel írni vagy olvasni szeretne-e a küldő. Ha a read-only bit értéke 1 (olvasás), akkor a transceiver az üzenet további részét figyelmen kívül hagyja (mivel nincs szüksége semmilyen más paraméterre), és visszaküldi a kapott címen lévő regiszter tartalmát (ezt 8 biten, mivel a

transceiver összes regisztere 8 bit méretű). Ha a read-only bit értéke 0 (írás), akkor az ezután beérkező adatokat a transceiver beírja az előzőleg kapott címen található regiszterbe. Lehetőség van egyszerre 8, 16 vagy 24 bit hasznos adatot is beírni (de 16 és 24 bit esetén ez csak egymás utáni címeken lévő regiszterekbe lehetséges).

Az egyszerűség és átláthatóság kedvéért mind olvasásnál, mind írásnál egyszerre egy 16 bites üzenet jut el a transceiverhez. Az üzenet felső 8 bitje (az üzenetküldés a felső bitekkel kezdődik) mindig a címet és a read-only bitet tartalmazza, az alsó 8 bit pedig olvasásnál csupa nullát, írásnál pedig a 8 bitnyi beírandó adatot foglalja magában. Olvasáskor nem zavaró az üzenet alsó 8 bitje, mert azt a transceiver mellőzi. Így gyakorlatilag ugyanazzal a paraméterezéssel valósul meg az SPI-on keresztüli üzenetküldés az összes függvényben.

7.2.1. Transceiver állapotára vonatkozó funkciók

Ebben a pontban bemutatom a transceiver általános viselkedésére vonatkozó különböző beállítási lehetőségeket. Minden egyes állítható tulajdonságnak külön beállító függvényt (Set), illetve minden egyes jelenleg érvényes tulajdonságnak külön kiolvasó függvényt (Get) készítettem. (Ez a felépítés egyúttal az ellenőrzést is elősegítette.)

7.2.1.1. Rendszer módjának (System mode) állítása

A System mód feladata a transceiver belső funkcióinak és regisztereinek menedzselése. Az egyes módok meghatározzák a transceiver egészének állapotát, fogyasztását, funkcionalitását. Alapvetően a transceiver 3+2 darab System módba kerülhet. Normál működés közben ezek a következőek:

- NORMAL mode: ez a mód a transceiver teljes funkcionalitását biztosítja. Minden hardvertől függő funkció elérhető ekkor. A CAN üzenetek küldése, fogadása (mikrokontrollernek való továbbítása) csak ebben a módban lehetséges. Az eszköz ekkor veszi fel a legtöbb energiát.
- STANDBY mode: a transceiver egy alacsony fogyasztású módja. A transceiver ebben a módban az INH lábán keresztül folyamatosan

feszültséget ad a hozzá kapcsolódó mikrokontroller tápszabályozójának, ezáltal bekapcsolva tartva a mikrokontrollert (lásd 1516. ábra).

STANDBY módban a transceiver ezenkívül CAN üzeneteket képes vizsgálni (küldeni és fogadni nem), és ez alapján különböző wake-up eseményeket detektálni. Képes a közönséges CAN üzenetek hatására vagy akár az előre megadott feltételeknek megfelelő CAN üzenetek hatására wake-up eseményt generálni.

- SLEEP mode: a transceiver első számú alacsony energiafelhasználású módja. E módban a transceiver kis fogyasztású állapotba kerül, és a transceiver INH lába magas impedanciájúvá válik, azaz a hozzá kapcsolódó mikrokontroller tápjának vezérlését megszünteti. SLEEP módban a transceiver csupán a CAN üzeneteket képes vizsgálni (CAN 2.0B vagy CAN PN szabvány szerint is), és ennek hatására akár wake-up eseményt generálni és STANDBY módba váltani.

  A transceiver felébredésének biztosítása miatt csak akkor állítható SLEEP üzemmódba az eszköz, ha előtte valamely wake-up esemény engedélyezve volt.

A Partial Networking által nyújtott fogyasztásszabályozás a SLEEP mód (és a STANDBY mód) segítségével valósul meg. A mikrokontroller tápellátása minimalizálható, amíg a transceiver SLEEP módban van. Amikor a busz üzeneteket forgalmaz, a transceiver vagy SLEEP állapotában marad vagy ”találat” esetén felébred és STANDBY módba vált. STANDBY módban pedig már a mikrokontroller feszültséget kap, és így mindenképpen felébred. Ezáltal a mikrokontroller lekapcsolása lehetővé válik, hiszen megfelelő CAN üzenetekkel egyszerűen felkelthető a fentebb leírtak szerint.

Az SPI üzeneteket mindhárom mód képes kezelni, így a transceiver vezérlése előtt nincs szükség módváltásra. A fenti módokon kívül a teljesség igénye miatt még említést teszek az OFF módról (amely a bekapcsolási folyamat egy állomása), valamint az OVERTEMP módról (amely az eszköz túlmelegedésekor jut érvényre).

```
ReturnType            CANTrcv_TJA1145_System_SetMode(SPIDriver_ChannelType
channelIndex, uint8 mode);
```

```
ReturnType            CANTrcv_TJA1145_System_GetMode(SPIDriver_ChannelType
channelIndex, CANModeReturnType* modeReturn);
```

A transceiver System módjának kiolvasása, valamint állítása ugyanazon regiszter ugyanazon bitjei szerint történik. A módot állító függvény paraméterként kap egy azonosítót, amely a beállítandó módot tartalmazza, valamint kap egy SPI vonalat azonosító ID-t, amely a felprogramozandó transceiver helyétől függ, az ahhoz vezető vonalat azonosítja. Minden függvény megkapja ezt az azonosítót (emiatt a továbbiakban nem térek ki rá), ezáltal egyszerre több különböző transceiver is programozható. A System mód olvasásakor a megfelelő függvény egy struktúrát ad vissza (technikailag a kapott struktúrát tölti ki), amelyben a kiolvasott mód azonosítója van, valamint egy időbélyeg (timestamp). A transceiverhez tartozó System mód futás közben változhat, akár 2 egymás utáni lekérdezés különböző eredményt adhat vissza, ezért szükséges a lekérdezett értékhez egy időpillanatot rendelni.

A függvények belső működése nem igényel semmifajta ismertetést, egyszerű regiszter kiolvasásokból/beírásokból áll, különféle maszkolásokkal végrehajtva. Ez a további függvényekre is igaz.

7.2.1.2. Kontroller módjának (Controller mode) állítása

A transceivernek nemcsak az általános szintű üzemmódja állítható, hanem lehetőség van a CAN busszal szoros kapcsolatban lévő kontroller módjainak konfigurálására is. Az egyes kontroller módok a CAN üzenetek kezelését, fogadását szabják meg. Ezek az üzemmódok természetesen összefüggésben vannak a transceiver System módjaival, nem minden Controller mód érhető el például a SLEEP System módban. A transceiver 3+1 Controller módot különböztet meg:

- ACTIVE mode: aktív módban a transceiver a CAN buszról jövő üzeneteket képes fogadni, illetve elküldeni. Ebben a módban teljes értékű CAN kontrollerként működik a transceiver.
- LISTEN-ONLY mode: szó szerint fordításban: csak hallgatás. Ez jól kifejezi az elérhető opciókat e módban. A transceiver csupán a buszon folyó kommunikációt figyelheti meg, fogadhatja (továbbküldheti a mikrokontrollernek) az üzeneteket, de nem küldhet. A LISTEN-ONLY

módot kifejezetten a fejlesztéshez készítették, a busz monitorozására kiválóan alkalmas.

- OFFLINE mode: ebben a módban a transceiver csupán a wake-up, vagy selective wake-up eseményeket keresve monitorozza a buszt. Az előbbi eseményt, egy közönséges CAN üzenet (buszon lévő forgalom) is kiválthatja, míg az utóbbit csak a megfelelő feltételeket teljesítő üzenetek.

Ezenkívül még az OFF Controller módor említeném meg , amely a transceiver normál működése közben mellőzött, egyfajta védelmet biztosító mód. A Controller módba a transceiver akkor kerül, amikor a tápvonalán (V_{BAT}) a megengedett szintnél nagyobb feszültséget van jelen.

```
ReturnType        CANTrcv_TJA1145_Controller_SetMode(SPIDriver_ChannelType
channelIndex, uint8 mode);
ReturnType        CANTrcv_TJA1145_Controller_GetMode(SPIDriver_ChannelType
channelIndex, CANModeReturnType* modeReturn);
```

A függvények paraméterei, illetve működése megegyezik az előzőekben tárgyalt System módokat kezelő függvényével.

7.2.1.3. Transceiver CAN FD tolerancia

A CAN with Flexible Data Rate protokollt már korábban bemutattam (lásd Protokoll ismertetése fejezet). Azon CAN transceiverek, amelyek nem képesek kezelni a CAN FD üzenetkereteket, minden CAN FD üzenetet egy hibás CAN keretnek látnak. Ha bizonyos számú ilyen hibát észlelnek, akkor hibát jeleznek, hibaüzenetet küldenek ki a buszra. Az általam alkalmazott transceiver képes tolerálni a CAN FD üzeneteket, így habár fogadni és feldolgozni nem tudja, de nem tekinti hibás üzenetnek azt. Ez a tolerancia engedélyezhető, illetve tiltható. Ehhez készítettem egy konfiguráló függvényt, amely a paraméterben megkapott logikai érték szerint állítja be ezt a tulajdonságot (logikai IGAZ esetén engedélyezi, HAMIS esetén tiltja). A továbbiakban is konzisztensen, e módon történik az engedélyező, tiltó függvények paraméterezése.

```
ReturnType            CANTrcv_TJA1145_SetFDTolerance(SPIDriver_ChannelType
channelIndex, boolean enable);
boolean               CANTrcv_TJA1145_GetFDTolerance(SPIDriver_ChannelType
channelIndex);
```

7.2.1.4. CAN sebesség beállítása

A CAN busz megfelelő kommunikációjához szükséges, hogy az egyes csomópontok azonos sebességre legyenek konfigurálva. A transceiver sebességének beállításához és lekérdezéséhez az alábbi függvényeket alkottam meg:

```
ReturnType CANTrcv_TJA1145_SetBaudrate(SPIDriver_ChannelType channelIndex,
uint8 baud);
uint8 CANTrcv_TJA1145_GetBaudrate(SPIDriver_ChannelType channelIndex);
```

A függvények az állítandó/visszaadandó sebességet 8 biten kódolva kezelik.

7.2.2. CAN PN funkciói

7.2.2.1. CAN PN engedélyezése, tiltása

A Partial Networking használatához engedélyezni kell a transceiver számára a selective wake-up lehetőségét, azaz hogy a buszon lévő üzenet csak az előre beállított feltételek teljesítése esetén ébressze fel az eszközt. A selective wake-up engedélyezéséhez a transceiveren lehetővé kell tenni a közönséges wake-up detektálását is.

```
ReturnType CANTrcv_TJA1145_Set_SelectiveWakeup(SPIDriver_ChannelType
channelIndex, boolean enable);
      boolean      CANTrcv_TJA1145_Get_SelectiveWakeup(SPIDriver_ChannelType
channelIndex);
```

7.2.2.2. CAN PN azonosító beállítása

A Partial Networking alapvető tulajdonsága a CAN üzenetek feltételek szerinti szűrése. Az egyik ilyen (állandó) feltétel az üzenetek azonosítója. Az eszközön csupán egy darab azonosító állítható be (amely standard vagy extended formátumú lehet), viszont mellete egy maszk is megadható, amivel az érvényes üzenetek száma növelhető. Amely biteken a maszk 1 értékű, ott a selective wake-up bekövetkezéséhez nem kell egyeznie a beállított ID-nak és a buszon lévő üzenet azonosítójának.

Ezen paraméterek állíthatóak be a transceiveren az üzenetek ID-jával kapcsolatosan. Mivel ezek nagyon is összefüggő paraméterek, struktúraként, egyben kapja meg a konfiguráló függvény és a kiolvasó függvény is struktúrába menti a kiolvasott értékeket.

```
ReturnType CANTrcv_TJA1145_SetId(SPIDriver_ChannelType channelIndex,
CANpn_IdType* param);
```

```
ReturnType CANTrcv_TJA1145_GetId(SPIDriver_ChannelType ChannelIndex,
CANpn_IdType* idReturn);
```

7.2.2.3. CAN PN adat beállítása

Az azonosító beállítása mellett a CAN üzenetek az adattartalmuk alapján is szűrhetőek (még több variációs lehetőséget nyerve ezzel). Az adatok figyelembevétele viszont csak a megfelelő CAN ID fogadása után lehetséges. Mivel ez egy választható opció, a Partial Networking használata mellett akár külön tiltható, vagy engedélyezhető is – szemben az azonosító használatával.

```
ReturnType CANTrcv_TJA1145_PN_SetDataEnable(SPIDriver_ChannelType
channelIndex, boolean enable);
boolean CANTrcv_TJA1145_PN_GetDataEnable(SPIDriver_ChannelType
channelIndex);
```

Egy CAN üzenetkeret maximum 8 byte-ot tartalmazhat, azaz 64 bitet. Így a CAN PN adatok szűrésekor szintén maximum 64 bit adható meg. A definiált adatfeltétel méretét (egész bájtokban) az adatok mellett külön át kell adni a transceivernek, amit a DLC (Data Length Code) értéke képvisel. A DLC egy egyfajta előszűrő, segítségével az adatok vizsgálata nélkül látható már, ha az érkezett CAN üzenet például több/kevesebb adatot tartalmaz.

Az adatok szerinti szűrés bitenként történik. Minden egyes bitpozíció egy csoportot jelöl. Találat (érvényes CAN üzenet) akkor van, ha az érkezett adat és az előre beprogramozott adat legalább egy azonos bitpozícióban 1-est tartalmaz. Ez azt jelenti, hogy van legalább 1 közös csoportjuk. A csoportosítással 1 azonosító egyszerre akár 64 darab különböző csoport tagja lehet.

```
ReturnType CANTrcv_TJA1145_SetData(SPIDriver_ChannelType channelIndex,
CANpn_DataType* param);
ReturnType CANTrcv_TJA1145_GetData(SPIDriver_ChannelType ChannelIndex,
CANpn_DataType* dataReturn);
```

Mivel a DLC és a hozzá tartozó adat értelemszerűen összetartozik, ezeket is egy struktúrában adom át a megfelelő függvényeknek. A DLC egy 8 bites, az adatot – a legrosszabb esetet figyelembe véve – 64 bites változóban tárolom. Megjegyzem, felhasználói (PC) oldalon a könnyebb kezelhetőség miatt (mind a felhasználó, mind az átküldés számára) 8 bites tömbként kezelem az adatot.

7.2.3. Eseményeket kezelő funkciók

A transceiver működését a bekövetkező külső és belső események (eventek) határozzák meg. Az eszköz a különböző események bekövetkeztét jelző bitekkel (úgynevezett flagekkel) tartja számon. A transceiver a bekövetkező eventeket négy kategóriába sorolja, mindegyik csoport külön regisztert foglal el:

- System event status register: e csoport tárolja a transceiver-re vonatkozó általános eseményeket. Konkrétan a megfelelő bekapcsolást, túlmelegedést és SPI hibát jelző bitek olvashatóak ki ebből a regiszterből.
- Transceiver event status register: a CAN busszal, kommunikációval összefüggő jelzéseket tárolja e csoport. Ilyen a CAN PN kerethibát, busz aktivitást, egyéb hibás CAN működést, valamint wake-up eseményt jelző flag. Ez utóbbinak fontos szerepe van mind a CAN, mind a CAN PN üzenetek fogadása során.
- Wake pin event status register: a transceiverhez történő külső wake-up beérkezését detektálja ez a regiszter.
- Global event status register: ez az események kiolvasását elősegítő csoport. Azt tárolja, hogy történt-e valamiféle esemény a fentebb felsorolt csoportok egyikében vagy sem. A pontos eseményt csak az adott csoport kiolvasásakor ismerhetjük meg, ez a regiszter csak jelzi, mely területen történt event.

Az egyes eseményeket jelző bitek nem törlődnek maguktól. Addig fennállnak (és adott esetben akadályozzák a transceiver további működését), amíg ki nem töröljük a jelzéseket. Ehhez felül kell írni az adott flageket 1 értékű bittekkel. Modulomban ezt az event ”törlést” közvetlenül az események kiolvasása után kell elvégezni, mert ezek összetartozó műveletek, és elvégzésük is hasonló és könnyen összevonható.

```
ReturnType CANTrcv_TJA1145_SetWakeup(SPIDriver_ChannelType channelIndex,
boolean enable);
boolean CANTrcv_TJA1145_GetWakeup(SPIDriver_ChannelType channelIndex);

ReturnType CANTrcv_TJA1145_Set_EventsEnable(SPIDriver_ChannelType
channelIndex, uint32 parts);
ReturnType CANTrcv_TJA1145_Set_EventsDisable(SPIDriver_ChannelType
channelIndex, uint32 parts);
```

```
ReturnType
CANTrcv_TJA1145_Get_Events_And_ClearStatusBits(SPIDriver_ChannelType
channelIndex, CANEventsType* eventsReturn);
```

Az események (időbélyeggel való) kiolvasása mellett lehetőség van külön engedélyezni, tiltani az egyes események detektálását. Az ezeket megvalósító függvények közvetlenül egy regisztermaszkot kapnak (3*8 bit értékes adattal, mert a globális események detektálása nem tiltható le), amelyekben lévő 1-esek (engedélyezés esetén), valamint 0-sok (tiltás esetén) íródnak be az engedélyező/tiltó regiszterekbe.

Továbbá felhívnám a figyelmet arra, hogy a wake-up esemény engedélyezése a többi eseménytől nincs elkülönített regiszterben. Viszont ez egy gyakran használandó funkció, emiatt külön függvénnyel is állítható – a regisztermaszkok használata mellett.

8. Számítógép oldali szoftver elkészítése

8.1. Működés

A PC oldalon futó program feladata egy olyan (felhasználói) interfész biztosítása, amely segítségével a fejlesztői számítógépen keresztül vezérelhető a gateway. Ez tulajdonképpen egy olyan programstruktúrát foglal magában, ami – az Ethernet kommunikáció kezelésén kívül – a gateway különböző feladatait ellátó függvényeket szolgálja ki. A kiszolgálás alatt az egyes gateway-beli függvények meghívását, megfelelő paraméterek átadását, illetve a gateway által küldött válasz fogadását/feldolgozását (parse-olását) kell érteni.

A PC oldalon mindezeket egy Java programozási nyelven írt programstruktúra valósítja meg. A magas szintű Java nyelv használta e feladathoz ideális, hiszen az objektumorientált Java nyelv segítségével az azonos paraméterek, illetve feladatok egységbezárása egyszerűen kivitelezhető. Így bármely felhasználó számára egyértelművé válik a program használta, a gateway vezérlése. A Java nyelv használatának másik előnye a platformfüggetlensége, ami tovább egyszerűsíti a PC oldali program felhasználhatóságát.

A Java nyelvre – mint minden objektumorientált nyelvre – jellemző az intenzív osztályhasználat. A programban minden egyes elem egy osztályhoz kapcsolható (annak vagy paramétere, vagy éppen egy osztály példánya). Emiatt a továbbiakban a PC oldal megismertetését a különböző célból létrehozott osztályokon keresztül mutatom be.

8.2. Transceiver kezelését megvalósítotó osztály

A TJA1145 transceiver közvetlen vezérlését megvalósító összes funkciót (Java nyelv esetén metódust) egy osztályon belül (`class CANTrcvTJA1145`) hoztam létre. A PC oldali program alapját képezi ez az osztály. Az ezen osztályon belül megtalálható metódusok a gateway-en futó, transceivert kezelő függvények számítógép oldali megfelelői. Mindegy egyes `CANTrcvTJA1145` osztálybeli metódus feladata önállóan futtatni (azaz meghívni a megfelelő paraméterekkel) a gateway-beli megfelelőjét, valamint a gateway-től érkező választ fogadni.

Az osztály működése tulajdonképpen abból áll, hogy a felhasználó az osztály egyik metódusát meghívja (ezzel egyúttal kiválasztva a transceiver műveletet is), és a metódus által megkövetelt paramétereket kitölti (amiket tulajdonképpen majd a gateway-en meghívandó függvény fog felhasználni). A metódus ennek hatására a kapott információkat feldolgozza és egy 8 bites elemekből álló tömbbe helyezi – olyan sorrendben, amelyet a fogadó oldalon a gateway elvár – majd elküldi. Amíg a gateway az üzenetet fogadja, és az alapján a szükséges elemeket futtatja, addig a PC oldalon lévő, meghívott `CANTrcvTJA1145` osztálybeli metódus futása nem ér véget, a gateway válaszára várakozik. A válasz megérkezésekor, ha van visszatérő adat (ez tipikusan lekérdezések esetén fordul elő), akkor a metódus feldolgozásra átadja a fogadott adatokat egy másik, speciális tároló osztálynak, majd visszatér annak egy példányával.

8.3. Kiegészítő komponensek

Hogy elősegítsem az átláthatóságot és a függvények használatát, a különböző paraméterek számára tároló osztályokat, illetve enumokat hoztam létre. Ezen kiegészítő elemek biztosítják `CANTrcvTJA1145` osztálybeli metódusok paraméterezésének, valamint a visszatérési értékeik értelmezésének félreérthetetlenségét.

Az adott funkciókat meghívó metódusok paramétereinél használtam fel a Java-beli enumokat. A Java programozási nyelvben az enumra egy olyan osztályként tekinthetünk, amelynek van egy, csak előre definiált értékeket felvehető, kitüntetett változója (value). Azonban minden value értékhez csupán egy enum példány létezhet. Ennek használatával a metódusok csak az előre definiált értékek közül kaphatnak paramétereket.

Az gateway által visszaküldött adatok tárolása viszont nem volt enumokkal megvalósítható. Ennek az az oka, hogy nem lehet több adatot tárolni egy enum érték alatt. Emiatt kizárólag az adott feladatokkal összefüggő tároló osztályokat hoztam létre. A `CANTrcvTJA1145` osztálybeli metódusok fogadják a visszaérkezett adatokat, és ezeket a tároló osztályok egy példányának adják át. A különböző tároló osztályok végzik az adatok feldolgozását, és ezzel együtt a saját példányuk kitöltését. Másképpen megfogalmazva: a tároló osztályok konstruktora végzi el egyben a kapott adatok visszafejtését is.

8.4. Demo program

Miután elkészült a CAN Partial Networking funkciót integráló program készítettem két egyszerű demonstrátor alkalmazást a számítógépre. Az alkalmazások a Partial Networking szabályai, valamint a transceiver előírása szerint először konfigurálják az SPI kommunikációt és a transceivert. Ez utóbbi magában foglalja a CAN PN bemutatásához szükséges wake-up események (wake-up és selective wake-up) engedélyezését és a SLEEP System mód, illetve ID feltételek beállítását. A két applikáció az üzenet adatainak kezelésében tér el, az egyik esetben a transceiver nem veszi figyelembe azt, míg a másik esetben igen.

Az adatok esetleges beállítása után a demo programban a két transceiver közreműködésével forgalom generálódik a CAN buszon (így biztosan lesz olyan transceiver, amely fogadja az üzenetet és nem generálódik hiba a küldőben). A transceiver-nek időre van szüksége, amíg nyugalmi állapotából a CAN üzeneteket vizsgálni képes állapotába kerül. Azaz, ha a busz forgalommentes állapota után csak 1 üzenet jelenik meg, az hiába felel meg a CAN PN nyújtotta feltételeknek, nem fogja felébreszteni a transceivert. Ennek oka, hogy a transceivernek idő kell, amíg felismeri az elkezdődő forgalmat, és amíg belső működése lehetővé teszi az üzenetek újbóli vizsgálatát. Emiatt a tesztek során minimum 2 üzenet egymás utáni kiküldésére került sor, ahol az első üzenet csupán a busz forgalmát hozta létre, tartalma közömbös volt, szemben a második, tényleges tesztre használtával.

A transceiver e működése nem okoz gondot, hiszen a gateway használatakor (fejlesztéskor), valamint a járműbe beépítve (tényleges alkalmazásakor) kis időn belül számos, gyakran periodikusan küldött üzeneteket észlel a CAN buszon.

A forgalom generálása után a program folyamatosan lekérdezi a transceiver System módját, valamint kiolvassa a wake-up eseményt tároló flageket. A transceivertől érkező válaszok pedig rögtön a képernyőre íródnak ki, ezáltal ellenőrizhetővé téve a megfelelő működést.

9. Összefoglalás, továbbfejlesztés

Szakdolgozatom készítése során egy már elkészült gateway modul tulajdonságait bővítettem ki: a CAN Partial Networking funkciót integráltam a gateway struktúrájába.

Mindezt először (a még nem elterjedt) CAN részleges hálózat működésének megismerésével kezdtem. A CAN PN egy nagyon ígéretes, CAN hálózati optimalizációt szolgáltat, amelynek elterjedése valószínű a jövőben. Viszont jelenleg ez még nem történt meg, kevés dokumentáció lelhető fel e témában, illetve elenyésző az evvel szerzett tapasztalat. Szakdolgozatommal bővítem a rendelkezésre álló CAN részleges hálózatok megvalósítására vonatkozó ismereteket.

Miután működésének alapjait megismertem, hardverszinten kezdtem el a feladat megvalósítását. Ehhez a jelenlegi hardver rendszerbe illesztettem egy Partial Networking megvalósítására képes CAN adó/vevőt. A kötött hardveres környezet miatt sajnos nem tudtam az új transceiver (és a CAN PN) minden újítását implementálni. A projektem jelenlegi felhasználásához ez nem szükséges, csupán a részleges hálózatok nyújtotta egyik lehetőség (mikrokontroller tápjának vezérlése) nincs hardveresen támogatva, viszont vizsgálható, mérhető állapotban van. A továbbiakban e területen van továbbfejlesztésre lehetőség, habár ehhez a hardveres környezet jelentős átalakítása is szükséges.

Szoftveres szinten minden, az adott transceiver nyújtotta új funkciót implementáltam a rendelkezésre álló szoftverarchitektúrába. Az új adó/vevő vezérlésével szereztem újabb tapasztalatokat, amikből jelenleg szintén hiányosság van, új termék lévén.

Összességében elmondható, hogy a CAN PN integrálása sikeres volt a meglévő rendszerbe. Az integrált adó/vevő tökéletesen ellátja feladatát, a részleges hálózatok funkció úgy működik, ahogyan azt a szabvány megköveteli. A továbbiakban lehet a munkámra építeni, felhasználni az avval szerzett tapasztalatokat.

10. Irodalomjegyzék

[1] Cornelius Butzkamm, David Bollati: *Partial Networking for CAN bus systems: Any saved gram CO2/km is essential to meet stricter EU regulations.*

CAN in Automation (CiA), http://www.can-cia.org/fileadmin/cia/files/icc/13/butzkamm.pdf (2012)

[2] Robert Bosch GmbH: *CAN FD, CAN with Flexible Data-rate*

http://can-newsletter.org/uploads/media/raw/30e0cfdc05de5f831221487388087eb8.pdf (2012)

[3] dr. Fodor Dénes: *Autóipari kommunikációs protokollok – a CAN*, Pannon Egyetem (2012)

[4] TKP Fieldbus gateway HW guideline - Gergely Horváth

(Version: 1.0 - 03 April 2014)

[5] Texas Instruments - 16- and 32-Bit RISC Flash Microcontroller (SPNS230B, October 2013)

[6] NXP Semiconductors N. V.: *TJA1145 High-speed CAN transceiver for partial networking* (Rev. 2 — 18 April 2014)

[7] https://upverter.com/

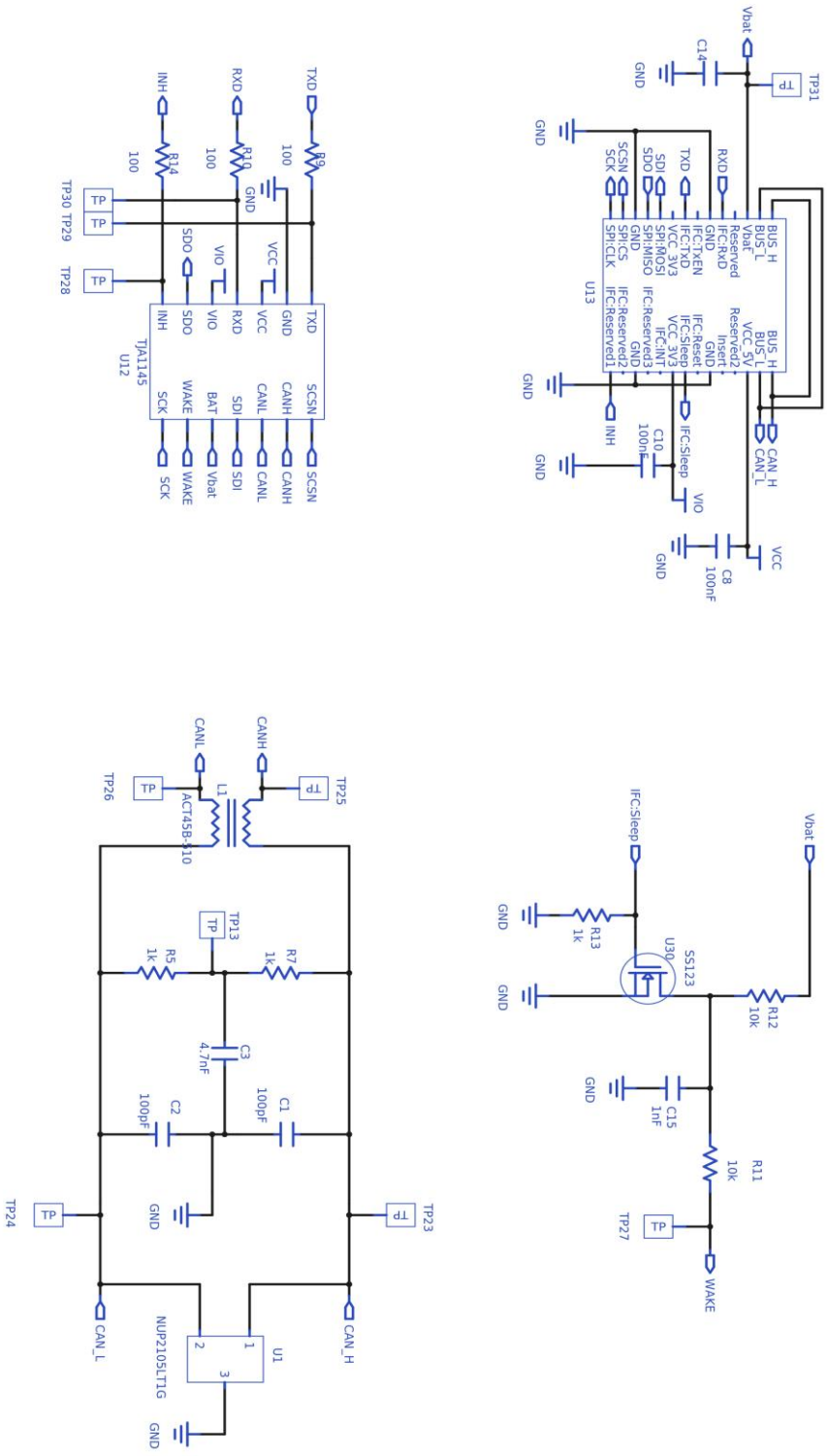
[8] Jim Lepkowski: *AND8230/D Application Hints for Transient Voltage Suppression Diode Circuits*
http://www.onsemi.com/pub_link/Collateral/AND8230-D.PDF

(Rev. 1 − October, 2009)

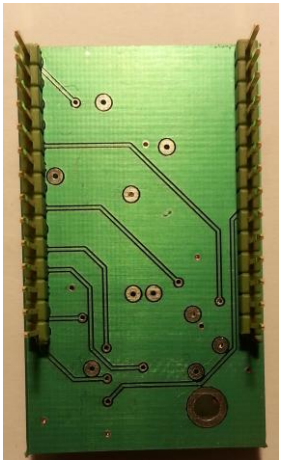
[9] Murata: No.TE04EA-1.pdf 98.3.20
http://www.murata.com/~/media/webrenewal/products/emc/emifil/knowhow/26to30.ashx

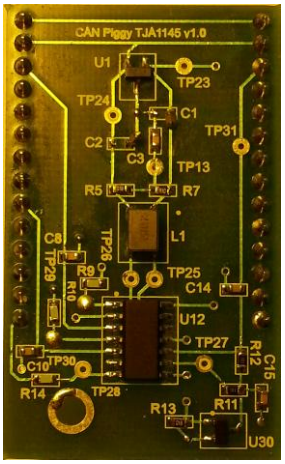
11. Függelék

Az elkészített panel kapcsolási rajza



A kész panel





Forráskód:

A forráskódokat a szakdolgozathoz melléklet CD tartalmazza.