SZAKDOLGOZAT-FELADAT

Kiss-Vincze Péter
szigorló villamosmérnök hallgató részére

Valós idejű üzenetkonverziós eszköz autóipari kommunikációs protokollokhoz

Napjaink prémium kategóriás személyautóinak működésében közel száz elektronikus vezérlőegység (ECU) játszik szerepet. Ezek között a kommunikáció *szabványos autóipari protokollokon* zajlik (CAN, FlexRay, LIN). A kommunikáció *legkisebb* (általában fizikai jelentéssel bíró) *adategysége* az úgynevezett *szignál*.

A *szignálok* adatkeretekben *elfoglalt helye*, valamint az általuk reprezentált fizikai mennyiségek *tárolási módja* (skálázás, ofszet) gépjármű típusonként *eltérhet*. Gyakori eset, hogy *prototípus-fejlesztésnél* egy már a korábbiak során másik gépjármű típushoz kifejlesztett vezérlőegységet kell a kész rendszerbe integrálni. Ilyenkor a vezérlőegység szoftverének módosítása, majd teljes újratesztelése helyett, kézenfekvő megoldás lehet egy olyan *üzenetkonverziós eszköz alkalmazása*, mely az adott gépjármű kommunikációs jeleit a vezérlőegység által elvártakra fordítja.

A vállalatnál rendelkezésre áll egy *saját fejlesztésű kommunikációs eszköz* (Gateway), mely rendelkezik minden olyan kommunikációs interfésszel (CAN, FlexRay, LIN és Ethernet csatornák) mely egy ilyen feladat megoldásához szükséges. A hallgató feladata *a Gateway szolgáltatásaira támaszkodva* egy olyan szabadon konfigurálható *üzenetkonverziós eszköz létrehozása* (C nyelven), mely a két oldal kommunikációs konfigurációjának ismeretében képes megvalósítani a fentiekben vázolt feladatot.

A hallgató a feladat megoldását az alábbi szempontok figyelembevételével végezze:

- *Mutassa be az autóipari vezérlőegységek közötti kommunikáció sajátosságait* (idő- valamint eseményvezéreltség, keretek felépítése, biztonságkritikus jelek védelme). Milyen megoldások kínálkoznak a feladat által megkövetelt *biztonságintegritási szint* elérésére?
- *Tervezze meg* és *mutassa be* a megvalósítandó *szoftver felépítését*. Vegye figyelembe, hogy a két oldal *kommunikációja* az időzítések és az üzenetek periodicitását tekintve is jelentősen *eltérhet* egymástól, valamint, hogy az eszköz által kiküldendő szignálok értékének kiszámításához adott esetben *több beérkező szignál értékét* is fel kell használnia.

- *Valósítsa meg* a fenti működést biztosító beágyazott (C nyelvű) szoftvert, és megfelelő tesztkonfigurációkkal *bizonyítsa annak helyes működését*.
- Az eszköz *konfigurációjának* tárolására *egyszerűen szerkeszthető* formátumot (pl.: XML) válasszon, és *készítse el* az ezen konfigurációs állomány feldolgozását és az eszköz eszerint történő felkonfigurálását biztosító *PC oldali keretrendszert* (Java nyelven). Milyen lehetőségek kínálkoznak arra, hogy a későbbiek során az eszköz PC igénybevétele nélkül is használható legyen?

Tanszéki konzulens: Dr. Sujbert László, docens
Külső konzulens: Faragó Dániel (ThyssenKrupp Presta Hungary Kft.)

Budapest, 2016. március 19.

……
Dr. Dabóczi Tamás
tanszékvezető

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

Kiss-Vincze Péter

VALÓS IDEJŰ ÜZENETKONVERZIÓS ESZKÖZ AUTÓIPARI KOMMUNIKÁCIÓS PROTOKOLLOKHOZ

TANSZÉKI KONZULENS

Dr. Sujbert László

BME MIT

KÜLSŐ KONZULENS

Faragó Dániel

ThyssenKrupp Presta Hungary Kft.

BUDAPEST, 2016

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Kiss-Vincze Péter**, 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, 2016. 06. 03.

...……

Kiss-Vincze Péter

Kivonat

Napjaink prémium kategóriás személyautóinak működésében közel száz elektronikus vezérlőegység (ECU) játszik szerepet. Ezek között a kommunikáció szabványos autóipari protokollokon zajlik (CAN, FlexRay, LIN). A kommunikáció legkisebb (általában fizikai jelentéssel bíró) adategysége az úgynevezett szignál.

A szignálok adatkeretekben elfoglalt helye, valamint az általuk reprezentált fizikai mennyiségek tárolási módja (skálázás, ofszet) gépjármű típusonként eltérhet. Gyakori eset, hogy prototípus-fejlesztésnél egy már a korábbiak során másik gépjármű típushoz kifejlesztett vezérlőegységet kell a kész rendszerbe integrálni. Ilyenkor a vezérlőegység szoftverének módosítása, majd teljes újratesztelése helyett, kézenfekvő megoldás egy olyan üzenetkonverziós eszköz alkalmazása, mely az adott gépjármű kommunikációs jeleit a vezérlőegység által elvártakra fordítja.

A ThyssenKrupp Presta Hungary Kft.-nél házon belül fejlesztett kommunkácis eszköz, a Gateway rendelkezik feladat ellátásához minden szükséges kommunikációs interfésszel. A megvalósított szoftver alkalmas a két oldal kommunikációját leíró fájlokból (DBC, fibex) kinyerni a szükséges információkat. Ezek tárolására elkészült egy az Eclipse modellező keretrendszer szolgáltatásaira támaszkodó kényelmesen szerkeszthető modell. Az üzenetek közötti koverziók definiálására egy Xtext alapú domain specifikus nyelv lett implementálva. A nyelv a modellben tárolt szignálok neveit felhasználva különböző műveletek leírására nyújt egy könnyen használható felületet.

A kimenet a Gateway szoftverébe illeszkedő „C” fájlokból áll, amiket egy JAVA nyelven implementált generátor készít el a felépített modell és a definiált konverziók felhasználásával.

Abstract

In today's modern vehicles close to a hundred Electronic Control Units (ECU) are operating. Their communication is realised through standardized automotive protocols like LIN, CAN and FlexRay. The smallest data component that usually contains physical information is the so called signal.

Different vehicle types utilize several unidentical methods for data representation (offset, scale) and signal placement in the frame structure. Integrating an already available control unit into a vehicle is a common task during prototype development. In that case modifying and testing the whole software of the ECU requires great effort. An easily configurable signal converter device that can translate the communication between the incompatible control unit and the vehicle is suitable for accelerating the integration process.

The so called Gateway device is an in-house project of ThyssenKrupp Presta Hungary. It is equipped with all the required communication interfaces. The implemented software is suitable for parsing message descriptor files (like DBC and Fibex) containing detailed information about the communication. For storing the acquired data an easily editable model was created by using the services of the Eclipse modeling framework (EMF). The signal conversions can be defined in an Xtext based domain specific language. It offers an intuitively useable platform for describing operations between stored signals.

The defined conversions and the built model are processed by a code generator implemented in JAVA. The output consist of „C" files compatible with the embedded architecture of the Gateway device.

Bevezetés

A manapság gyártott járművekben egyre nagyobb hangsúllyal vesznek részt elektronikus vezérlőegységek (ECU-k). Ezek vezérlik a motort, a fékrendszert, a menetstabilizációs funkciókat, a kormányrendszert, azaz egyre több és komplexebb feladatot látnak el. Modern személyautókban több mint száz ECU is üzemel. A különböző feladatokat ellátó ECU-k biztonságos kommunikációja elengedhetetlen, amelyre több protokoll is rendelkezésre áll. Az autóiparban a CAN (Controller Area Network), a LIN (Local Interconnect Network) és a FlexRay a legelterjedtebb megoldások, ezeket azonban a különböző gyártók eltérő konfigurációkkal használják. Gyakran előfordul, hogy egy már meglévő, de másik autótípushoz fejlesztett vezérlőegységet kell újra felhasználni, azonban ilyenkor kompatibilitási problémák lépnek fel. Az eltérő típusú ECU-k közötti kommunikáció megoldása sok esetben nagy mértékben meggyorsíthatja a fejlesztési folyamatokat. Prototípus-fejlesztésnél minél előbb biztosítani kell a megrendelőnek egy használható hardvert, hogy a megrendelés tárgyát képező ECU alapvető funkcióit nélkülözni nem tudó fejlesztésekkel haladni tudjon. Gyakorlati példaként megemlíthető egy új partner, ahol az első zártpályás tesztek alatt egy másik típushoz gyártott vezérlőegység lett üzembe helyezve a tesztelt gépjárműben. A probléma központi elemei a kommunikáció fizikai tartalommal bíró adategységei, a szignálok. Ezek eltérő protokollokon keresztül, különböző keretekben (frame-ekben) kerülhetnek elküldésre és más adategységek részeként lehetnek definiálva. Az integrációhoz meg kell oldani mindkét oldal üzeneteinek fogadását és a megfelelő konverzió elvégzése után az adott hálózat által feldolgozható formában továbbítani azokat. A ThyssenKrupp Presta Hungary Kft.-nél házon belül fejlesztett eszköz, a Gateway rendelkezik a szükséges kommunikációs csatornákkal és elegendő számítási teljesítménnyel a feladat megoldásához. Etherneten keresztül újraprogramozható és a rendelkezésre álló PC-s driver segítségével bizonyos funkciói vezérelhetők. Egy konverziós eszköz elkészítéséhez jó kiindulási pontot biztosítanak a gyártóktól kapott úgynevezett üzenetleírók. Ezek tartalmazzák a szignálok minden szükséges paraméterét, beleértve az őket tartalmazó frame(-ek) tulajdonságait, a felhasznált protokoll konfigurációját és a fizikai tartalom értelmezésének módját. A konverzió megadását célszerű magasabb szinten elvégezni. Erre a célra érdemes egy

egyszerű programozási nyelvet definiálni, ami alapvető műveletek gyors leírását teszi lehetővé és könnyen feldolgozható.

A szakdolgozatom elkészítése során a ThyssenKrupp Presta Hungyry Kft.-nél felmerülő vezérlőegység integrációs feladatokhoz törekedtem egy könnyen használható és az alkalmazott Gateway egység mélyebb ismeretét nem igénylő eszközt létrehozni, amely egyszerű PC-s környezetből tetszőleges szignálok közötti konverzió konfigurálását teszi lehetővé.

A továbbiakban ismertetem az autóiparban használatos kommunikációs protokollok működését és kitérek a különböző biztonságintegritási szintekre, majd bemutatom a Gateway eszközt, kitérve annak alapvető funkcióira, használatára és felépítésére. A beágyazott modulok részletezése során fontos szerepet kapnak az üzenetek struktúráját és a kommunikációs beállításokat tároló adatszerkezetek. A magas szintű modulok között bemutatom az üzenetleírókból beolvasott adatok segítségével automatikusan felépíthető modellt és ennek szerkeszthetőségét. Végül pedig az általam definiált szignálkonverziókat leírni képes nyelvet és az ezekből beágyazott kódot generáló implementációt.

1 Szabványos autóipari protokollok

Az autóipariban használatos protokollok között megtalálhatóak esemény és idővezérelt megoldások egyaránt. A bemutatott üzenetkonverziós eszköz a CAN és a FlexRay protokollokon képes üzemelni, továbbá a hardveren adott a LIN támogatás, azonban sem az alacsony sem a magas szintű driver nem áll még készen a használatra.

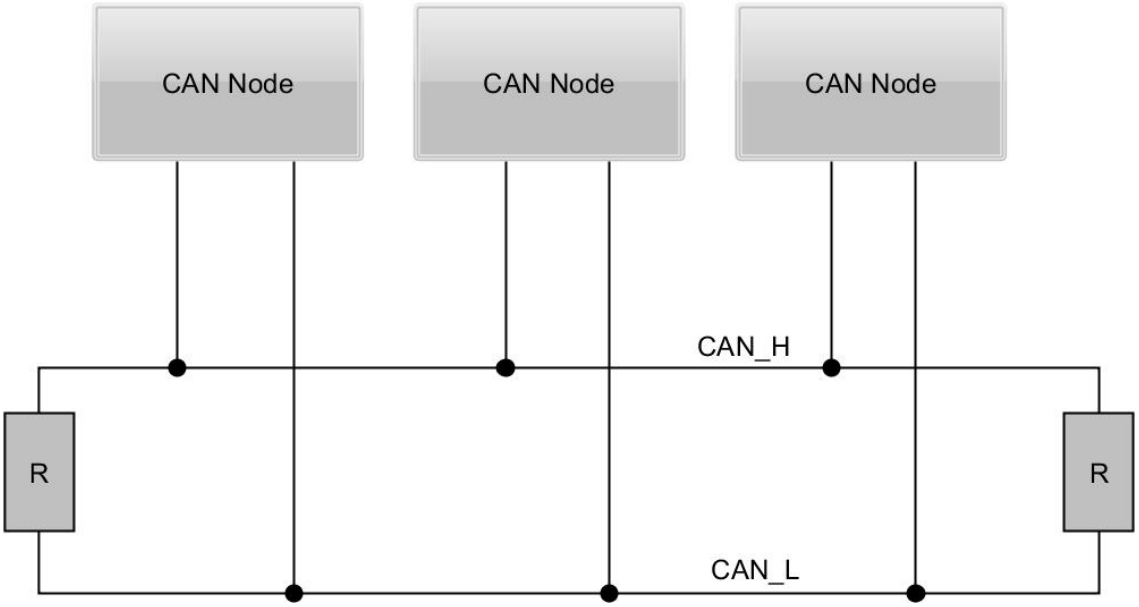
1.1 CAN (Controller Area Network)

A modern autókba egyre nagyobb számú vezérlőegységet építenek be. A motor, a sebességváltó, a kormányzás, a különböző biztonsági funkciók mind külön ECU-val rendelkeznek. A különböző alrendszerek és szenzorok közötti kommunikáció elengedhetetlen. Ezt az igényt szolgálja ki a CAN, ami az alacsony kiépítési költségének és megbízhatóságának köszönhetően hamar elterjedt az autóiparban. A CAN egy széles körben alkalmazott soros, eseményvezérelt protokoll, megtalálható orvos diagnosztikai eszközökben ipari gyártóberendezésekben és automatákban egyaránt.

A CAN szabványt 1983-ban kezdte el fejleszteni a Robert Bosch GmbH. 1986-ban mutatták be először és 1988-ban került sorozatgyártású autóba (BMW 8-as széria). 1993-ban szabványosították (ISO 11898). A CAN-t folyamatosan fejlesztik, 2012-ben mutatták be a CAN FD-t lehetőséget biztosítva ezzel magasabb sávszélességű adatátvitelhez. [2]

1.1.1 A CAN hálózat

A CAN egy multimaster soros busz, amely csomópontokat köt össze. A csomópontoknak elektronikus vezérlőegységek felelnek meg. Minden csomópont egy kétvezetékes hálózatra van csatlakoztatva (árnyékolatlan csavart érpár). Autóipari alkalmazásokban a vezeték ellenállása nem haladhatja meg a 60 mΩ-ot és a csomópontok maximális száma 32 lehet.



1.1. Ábra: CAN topológia

Az elérhető maximális sebesség 1 Mbit/s. Az áthidalható maximális távolság 40-500 m, azonban a távolság növekedésével jelentősen csökken az átviteli sebesség (500 m-es távolság esetén 125 Kbit/s). Az autóiparban egy hálózaton belül 40 m a maximális megengedett buszon mért távolság két csomópont között.

A CAN szabvány különbségi jeleket használatát írja elő a közös módusú zavarok elkerülése érdekében.

1.1.2 Arbitráció

A CAN buszra csatlakozó vezérlőegységek „huzalozott ÉS" kapcsolatban állnak egymással. Ebből következően a 0 a domináns és az 1 a recesszív érték, azaz ha egy 0 és egy 1-es érték találkozik, akkor a 0 jut érvényre. Ennek köszönhető az üzenetek azonosítóján alapuló prioritás. Minden adó folyamatosan olvassa a buszon lévő értéket és ha az általa kiadottal nem megegyezőt tapasztal, megszakítja a küldést. Ebből következik, hogy az alacsonyabb azonosítóval rendelkező üzenet a magasabb prioritású és a kiadott legnagyobb prioritású üzenet azonnal kikerül a buszra. További előnye ennek a kialakításnak, hogy valamelyik keret mindig kikerül a buszra, azaz az ütközések nem roncsolják a magasabb prioritású keretet, nem destruktívak.

1.1.3 A CAN keretek

Egy CAN hálózaton kétféle üzenetformátum létezik „Standard", ahol 11 bit és „Extended", ahol 29 bit hosszúságú lehet az egyes üzenetek prioritását is meghatározó

azonosító. A szabvány 4 kerettípust definiál: Adatkeret, Távoli keret, Hibakeret, Túlcsordulás keret. Az első három kerül bemutatásra ebben a fejezetben.

1.1.3.1 Adatkeret (Data frame)

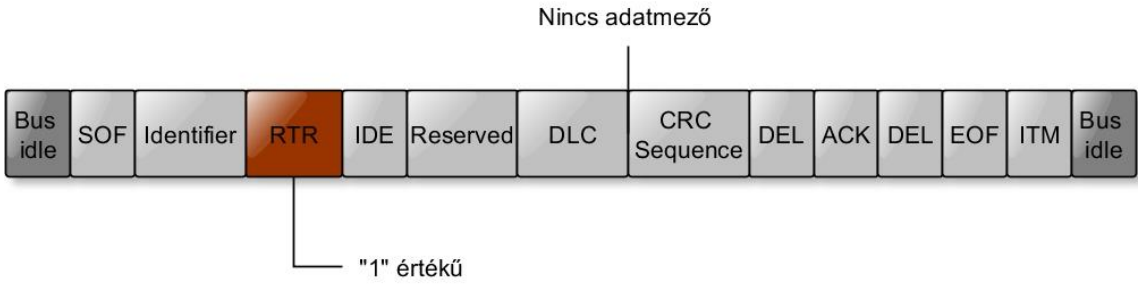
A hasznos információt hordozó adatkeret maximum nyolc byte hosszú adatot tartalmazhat. Az alábbi táblázat összefoglalja egy adatkeret felépítését.

1.2. Ábra: Az adatkeret

Mező neve	Hossz (bit)	Leírás
Start of Frame	1	Jelzi a keret kezdetét.
Identifier	11	Egyedi azonosító, ami meghatározza a prioritást is.
Remote Transmission Request (RTR)	1	Távoli adáskérésnél 1 értékű egyébként 0.
Identifier extension bit	1	11 bites azonosító esetén 0 értékű.
Reserved bit	1	0 és 1 érték is elfogadott.
Data length code	4	Az adat byte-ok számát tartalmazza.
Data field	0-64	Adatmező, maximum 64 bit hosszú.
CRC	15	Ellenőrző összeg.
CRC delimiter	1	1 értékű jelzőbit.
ACK slot	1	Az ACK bitet a vevő sikeres vétel után 0-ba állítja.
ACK delimiter	1	1 értékű jelzőbit
End of frame (EOF)	7	A keret végét jelző 1-es értékű bitek.

1.1.3.2 Távoli keret (Remote frame)

Az adatküldés általában a küldők ütemezik, úgy hogy a forrás, amikor szükséges kiküldi az adatkereteket a buszra. Lehetséges azonban egy fogadó csomópontnak kérni az adatküldést, erre a célra használható a távoli keret. A távoli keretet az adatkerettől az RTR bit különbözteti meg: ilyenkor 1-es értékkel szerepel.



1.3. Ábra: A távoli keret

Abban az esetben, amikor egyszerre kerül ki a buszra egy adat és egy távoli keret az adatkeret a nyeri meg az arbitrációt, (RTR=0) mivel a 0 a domináns érték.

1.1.3.3 Hibakeret (Error frame)

A hibakeret a kommunikáció során fellépő hibák jelzésére hivatott. Egy hibásnak ítélt adat küldése azonnal megszakad és egy hibaüzenet váltja fel a buszon. A keret felépítése jelentősen eltér az adat és a távoli keretektől. Két részre bontható fel. Az „Error Delimiter”-re, ami nyolc 1-es értékű bitből áll és az „Error Flag”-re, ami további 6-12 bitet tartalmaz. Abban az esetben, ha egy csomópont hibát észlel, a hálózaton aktív hibaüzenetet küld. Ez az „Error Flag” keretszakaszban hat 0 értékű bitet jelent. Aktív hibaüzenet fogadásakor a csomópontok passzív hibaüzenetet küldenek a buszra, ahol az „Error Flag” hat darab 1-est tartalmaz.

1.4. Ábra: A hibakeret

1.1.4 Hibadetektálás és hibakezelés

A CAN szabvány az alábbi hibajelzési módszereket definiálja:

- Bit Stuffing: A kellően sűrű jelváltás érdekében (amennyiben nem hibakeret van a buszon) maximálisan öt azonos értékű bit követheti egymást. Amennyiben ez a feltétel nem teljesülne az adó beszúr egy ellentétes értékű bitet. A vevő „Stuff Error" hibát jelez, ha valamilyen okból 6 egyforma bitet érzékel.
- Bit Monitoring: Az adó küldés közben folyamatosan olvassa vissza a buszról a kint lévő értékeket. Az arbitrációs fázis kivételével mindig meg kell egyeznie a küldött és a buszon található biteknek, amennyiben ez nem teljesül „Bit Error" hibáról beszélünk.
- Frame Check: A CAN szabványban definiált keretek felépítésétől eltérni nem lehetséges. Ezt a vevők folyamatosan figyelik és hiba esetén „Frame Error" keletkezik.
- CRC: A Ciklikus redundancia ellenőrzés az adatmezőt egy polinomként használja fel és elosztja egy előre meghatározott polinommal. Az eredmény alkotja a CRC mezőt. A vevő elvégzi ugyanezt a műveletet a fogadáskor és eltérő eredmény esetén „CRC Error" hibát ad.
- Acknowledgement check: Az ACK bitet felhasználva a vevő jelezni tudja, hogy megtörtént a fogadás. Amennyiben az ACK bitet senki nem állítja 0 értékűre „Acknowledgement Error" keletkezik.

1.2 LIN (Local Interconnect Network)

A LIN egy soros kommunikációs protokoll, amit az autóipar költségérzékeny és nem biztonságkritikus területein használnak. A drágább kommunikációs technológiákkal (pl.: CAN, FlexRay) kiépített rendszerek alhálózataként használatos.

Az 1990-es évek közepén több gyártó is elkezdett költséghatékony technológiákat fejleszteni, de ezek nem voltak elég gazdaságosak. 1998-ban a BMW, a Daimler-Benz, a Motorola és a VTC közösen egy új buszrendszer fejlesztésébe kezdtek, majd az Audi a Volvo és a Volkswagen csatlakozásával megalapították a LIN konzorciumot 1999-ben.

A LIN 1.0 szabvány 1999-ben, majd 2002-ben a LIN 1.3 és 2003-ban a 2.0 került bemutatásra. A legfrissebb LIN 2.2-es revízió 2010 óta elérhető. Ingyenes, mindenki számára hozzáférhető, tovább csökkentve ezzel egy LIN hálózat kiépítési költségét. [3]

1.2.1 A LIN kommunikáció

Egy LIN hálózat egy Master és több Slave csomópontból állhat. Csak a Master kezdeményezhet adatátvitelt, és a többi Slave közötti kommunikációt is ő vezérli így arbitrációs problémák nem merülnek fel. A kommunikáció egy vezetéken történik és a kiküldött adatot a hálózatra kötött összes résztvevő képes fogadni. A szinkronizáció az üzenetekben található speciális mező segítségével történik.

Az általános LIN keretek két részből állnak. A Master küld először egy fejlécet, majd erre egy Slave (vagy maga a Master) egy Response-al válaszol. Csakúgy, mint a CAN esetében, itt is a logikai alacsony, azaz a 0 érték a domináns.

A Gateway hardver fel van készítve LIN kommunikációra, azonban a szoftveres támogatás még nem készült el, ezért ennek a protokollnak a bemutatása rövidebb terjedelmű.

1.2.1.1 A Header (kérés avagy fejléc)

- SYNC Break Field: Legalább 13 domináns bitből áll amelyet, minimum 1, maximum 4 db recesszív bit követ.
- SYNC Field: A start és a stop bittel együtt 10 bitnyi 1-0 sorozat, azaz egy recesszív és egy domináns bit közötti byte 0x55 értékkel. Ennek a mezőnek a segítségével valósul meg a szinkronizáció és az adatátviteli sebesség beállítása. A beépített oszcillátorral ellátott Slavek frekvenciájának fluktuációja 14%-os is lehet, az ilyennel nem rendelkezőknél 1,5%-os tolerancia megengedett.
- PID Field (azonosító mező): A mező tartalmaz egy egyedi azonosítót, aminek köszönhetően a megcímzett Slave tudni fogja, hogy neki kell válaszolni. A 6 bites azonosítóból 2 bit a Slave válaszának a hosszát jelzi. A 6 bitből 64 különböző érték adódik (0-63), amiből a 60-as és a 61-es diagnosztikai kérést és választ jelöl. A 62-es és a 63-as azonosítók későbbi fejlesztések számára vannak fenntartva. További 2 paritásbit

gondoskodik az adatátvitel biztonságosabbá tételéről. A felsorolt 8 bit egy start és egy stop bit között helyezkedik el.

1.2.1.2 A Response (válasz)

- Data Field (adatmező): Az adatmező 2, 4 vagy maximum 8 byte információt tartalmazhat, ezt a Headerben kapott azonosító mező által definiálva. A küldés a legkisebb helyiértékű byte-tal kezdődik.
- Checksum: A hasznos adatot védő 8 bit ellenőrzőösszeget tartalmazó mező.

1.3 FlexRay

Az autóipari vezérlőegységek fejlődésével nőtt a kommunikáció sávszélesség igénye is. A CAN által biztosított maximum 1 Mbit sebesség már nem minden esetben volt elegendő. A prémium kategóriás járművek esetében a gazdaságosság kevésbé fontos szempont, azonban a szolgáltatások száma egyre inkább bővül. Előtérbe kerültek olyan funkciók, mint az elektromos fékrendszer és kormányzás, ahol a kezelőszervek nincsenek fizikailag összekötve a kerekekkel, illetve a fékekkel. Az egyre fejlettebb felfüggesztés, a menetstabilizációs eljárások és az önvezető autó koncepciója további adatátviteli sebességet, valamint nagyobb megbízhatóságot követel.

A FlexRay konzorcium által kifejlesztett FlexRay protokoll sokkal gyorsabb és megbízhatóbb megoldást jelent, mint a CAN. A FlexRay szabvány két csatornát definiál („A” és „B”), ezek használhatóak különböző üzenetek küldésére vagy redundáns módon is. Idővezéreltségének és redundanciájának köszönhetően sokkal megbízhatóbb, így ideális protokoll a különösen biztonságkritikus illetve valósidejű alkalmazások számára is (pl.: Steer-by-Wire, Break-by-Wire). Először 2007-ben került sorozatgyártású autóba (BMW X5), de a 2008-ban bemutatott X6-ban használták ki teljesen a szabvány nyújtotta lehetőségeket.

A FlexRay konzorcium 2010-ben publikálta a 3.0.1-es specifikációt. Mára az ISO[1] vette át a szabvány gondozását, mely az ISO 17458 számon érhető el. [1]

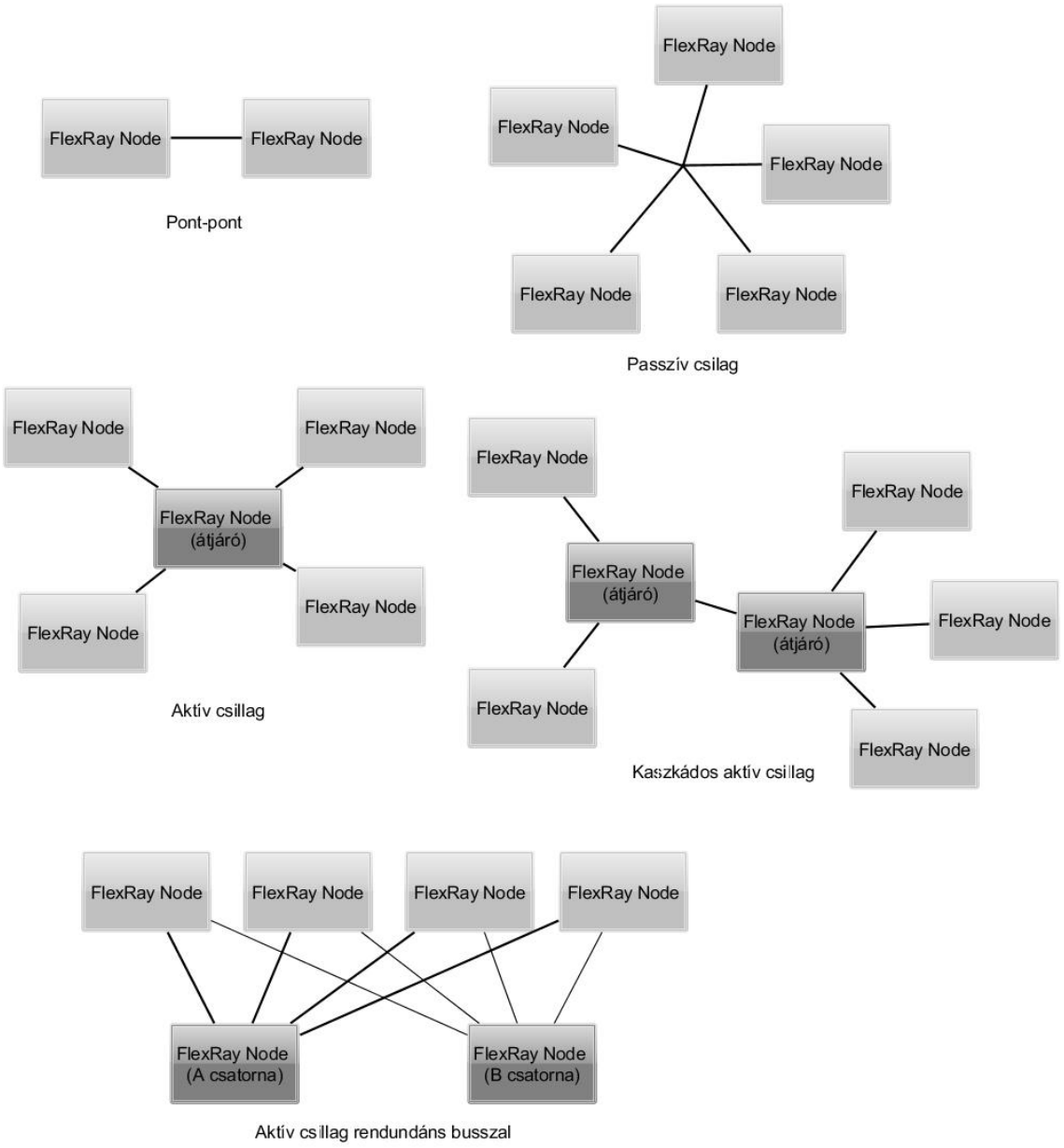
[1] International Organization for Standardization

A FlexRay kommunikáció

A FlexRay kommunikáció a CAN-hez hasonló módon árnyékolatlan csavart érpáron zajlik az induktív zajok csökkentése érdekében. Szintén a CAN-nél alkalmazott módon különbségi jeleket használ a közös módusú zavarok kiszűrése végett. Egy FlexRay klasztert csomópontok (node-ok) alkotnak. A csomópontokat a FlexRay busz köti össze, ami többféle topológia szerint épülhet fel.

1.3.1 Busztopológiák

Az alapvető topológiák a következők:



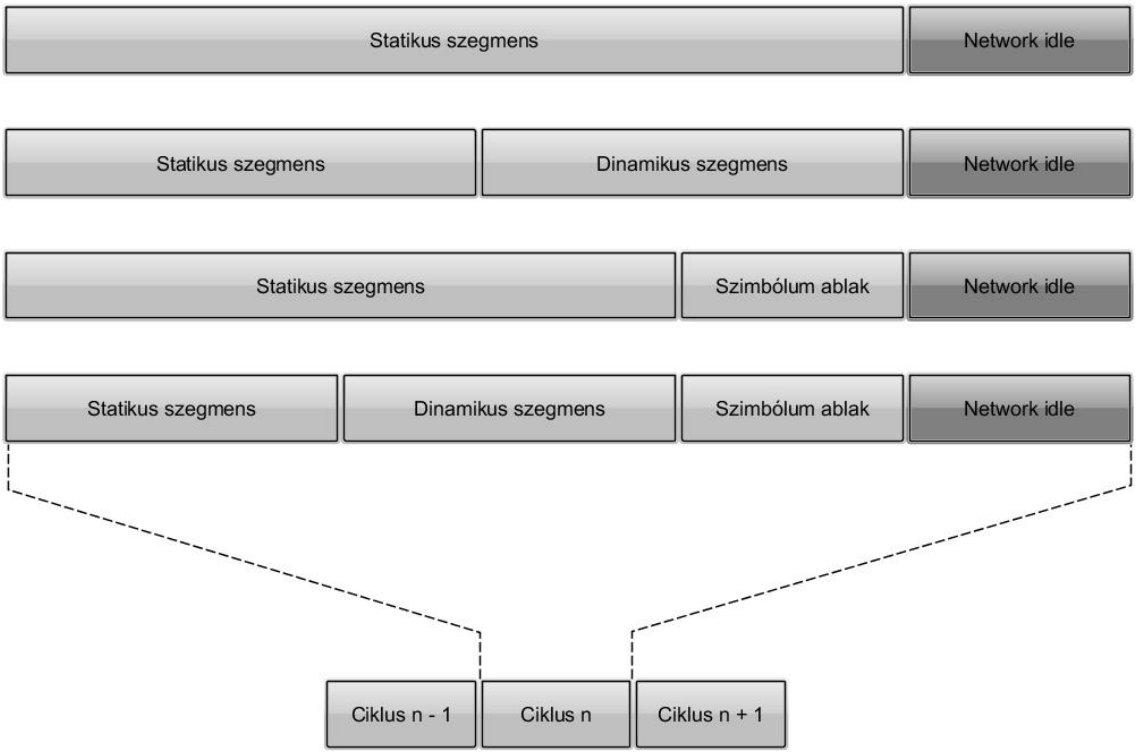
1.5. Ábra FlexRay topológiák

- Pont-pont kapcsolat: Két FlexRay node-ot tartalmazó topológia, ahol egy vezetékkel közvetlenül össze vannak kötve a csomópontok.
- Passzív csillag: Kettőnél több node alkotja, amelyek egy közös pontban vannak összekötve.
- Lineáris passzív: Nagyon hasonló a passzív csillaghoz. A különbség mindössze annyi, hogy az összekötött vezérlőegységek nem egy pontban találkoznak, hanem egy összekötő vezeték különböző pontjaihoz csatlakoznak. Redundáns adatátvitel esetén két összekötő vezeték van.
- Aktív csillag: Egy központi node átjáró szerepet tölt be, ehhez kapcsolódik a többi, lényegében pont-pont összeköttetéssel. Minden kommunikáció a központi vezérlőegységen keresztül történik.
- Aktív csillag redundáns busz használata mellett: Az aktív csillagtól az különbözteti meg, hogy két központi node van a FlexRay „A" és „B" csatornájához külön-külön. Minden ECU egyik csatornája az egyik átjáróhoz a másik a másikhoz van kötve ezeken keresztül folytatva a kommunikációt.
- Kaszkádos aktív csillag: A FlexRay klaszterben a szignálok integritását megőrzendő limitálva van a maximális távolság a két legtávolabbi pont között. A kaszkádosított csillag esetében ezt kicsit meg lehet növelni azzal, hogy két aktív csillag központi átjáró csomópontjait összekötjük.

1.3.2 A protokoll

Egy FlexRay klaszterben ütemezett kommunikáció zajlik. Emellett a protokoll lehetőséget biztosít eseményvezérelt működést megkövetelő üzenetek kezelésére is.

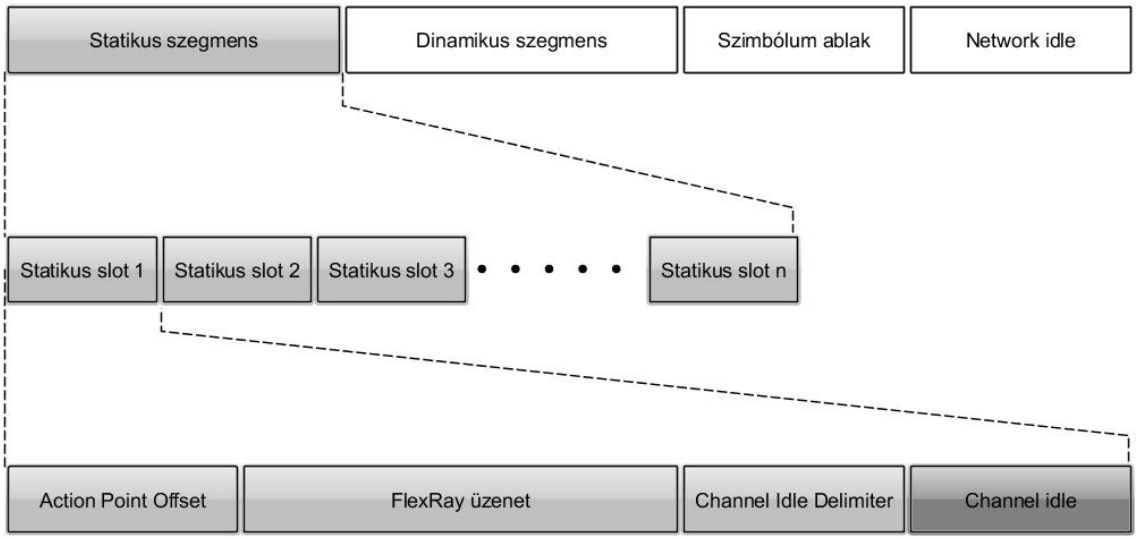
A kommunikációs ciklus minimum 2 maximum 4 szegmensből áll. Minden ciklusban jelen van az ütemezett üzeneteket tartalmazó statikus szegmens és az üresjárási idő, ami időt biztosít a szinkronizáció számára. A dinamikus szegmens az eseményvezérelt adatok átviteléért felel. A negyedik szegmens a szimbólum ablak a működéshez szükséges szimbólumok továbbítását végzi.



1.6. Ábra: FlexRay ciklus

A statikus szegmensben kerülnek elküldésre az úgynevezett statikus üzenetek. Minimum 2 maximum 1013 egyenlő hosszúságú időrést tartalmazhat, melyekhez egy-egy üzenet társítható. Ebből következően egy számláló érték (slot azonosító) egyértelműen azonosítja a küldő node-ot is. A az időrések minimális száma azért kettő, mert legalább két node szükséges a központi idő szinkronizáltságának fenntartásához. A FlexRay vezérlők szinkronizálják a számlálóikat így követve nyomon éppen hányas számú időrésnél tart a kommunikáció. A rendelkezésre álló két csatornát lehetséges nem redundáns módon, különböző üzenetek küldésére is használni, ezzel megduplázva az adatátviteli sebességet.

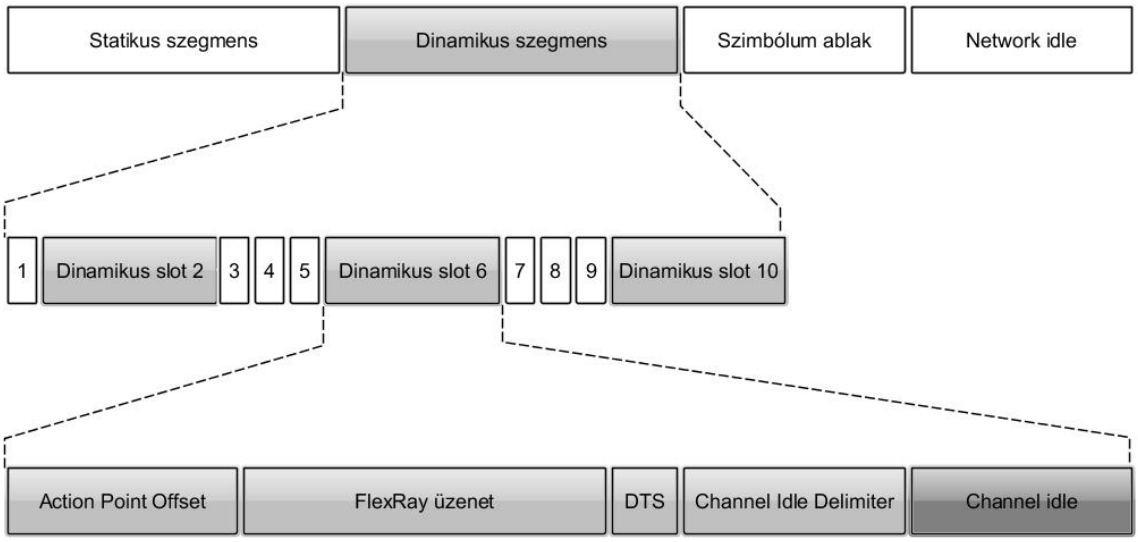
Egy statikus időrés elegendően hosszú, hogy a megengedhető legnagyobb szinkronizációs hiba, és a leghosszabb szükséges adat is beleférjen. Négy további részre bontható a statikus időrés. „Action Point Offset”-tel kezdődik és „Channel Idle”-lel végződik, amelyek az elcsúszás és az üzenetek hosszúságának kompenzálására szolgálnak. A kettő között helyezkedik el a FlexRay keret, ahol a hasznos adatátvitel zajlik és a 11 recesszív bitből álló „Channel Idle Delimiter”.



1.7. Ábra: A statikus szegmens

A dinamikus szegmens használata opcionális. Alapvetően az eseményvezérelt üzenetek küldésére alkalmas. A dinamikus szegmens a statikushoz hasonlóan adott számú időrésből (minislot) épül fel. A szegmens kezdetekor minden FlexRay csomópont eggyel növeli a számlálóját, és amennyiben az adott dinamikus időréshez tartozó node nem kezdeményez küldést, egy minislot hosszúságú idő után tovább növekednek a számlálók. Küldés esetén a dinamikus időrés hosszúsága az elküldött adat függvényében változik, így előfordulhat, hogy egy ciklusba nem minden csomópont üzenete fér bele. Ebben az esetben a következő ciklusban - amennyiben a nagyobb prioritásúakkal nem telik be ismét a dinamikus szegmens - kerülnek továbbításra.

A dinamikus időrés felépítése nagyon hasonló, mint a statikusé. „Action Point Offset”-tel kezdődik majd az „action point” és az üzenet küldésével folytatódik. A dinamikus keret mérete változó ezért követi egy „Dynamic Trailing Sequence” (DTS) nevű mező, ami garantálja, hogy a következő „action point”-nál végződjön. A DTS maximum 1 minislot hosszú lehet.

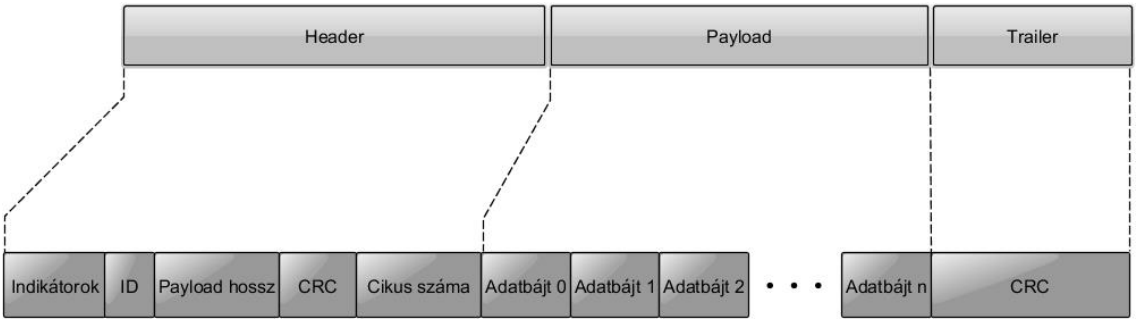


1.8. Ábra: A dinamikus szegmens

1.3.3 A FlexRay keret

Az adatküldés egységes felépítésű keretekben történik. Egy üzenet három részre osztható:

- Fejléc (header): 40 bitből áll, amiből 11 bit az azonosítót tartalmazza, meghatározva ezzel az üzenethez tartozó időrést. Az azonosító előtt 5 indikátor bit foglal helyet, amik a keret főbb tulajdonságaira vonatkozó információkat tárolnak. Itt derül ki, hogy „Startup" , „Sync", vagy „Null" frame érkezik. Ezek egymástól független tulajdonságok, annyi megkötéssel, hogy minden „Startup" frame „Sync" frame is kell legyen. A következő mező tartalmazza a hasznos adat hosszát 7 biten, majd ezt követi egy 11 bites CRC-t tartalmazó mező és a 6 bit szélességű ciklusszámláló.
- Hasznos adat (payload): A küldött információ hossza maximum 254 byte lehet, ami a FlexRay-en értelmezett 16 bites szóhosszt tekintve 127 szót jelent. „Null frame" esetén csak 0 értékű bájtot tartalmazhat a payload.
- Trailer: Egy 24 bites CRC mezőt tartalmaz az üzenet integritásának megőrzése érdekében.



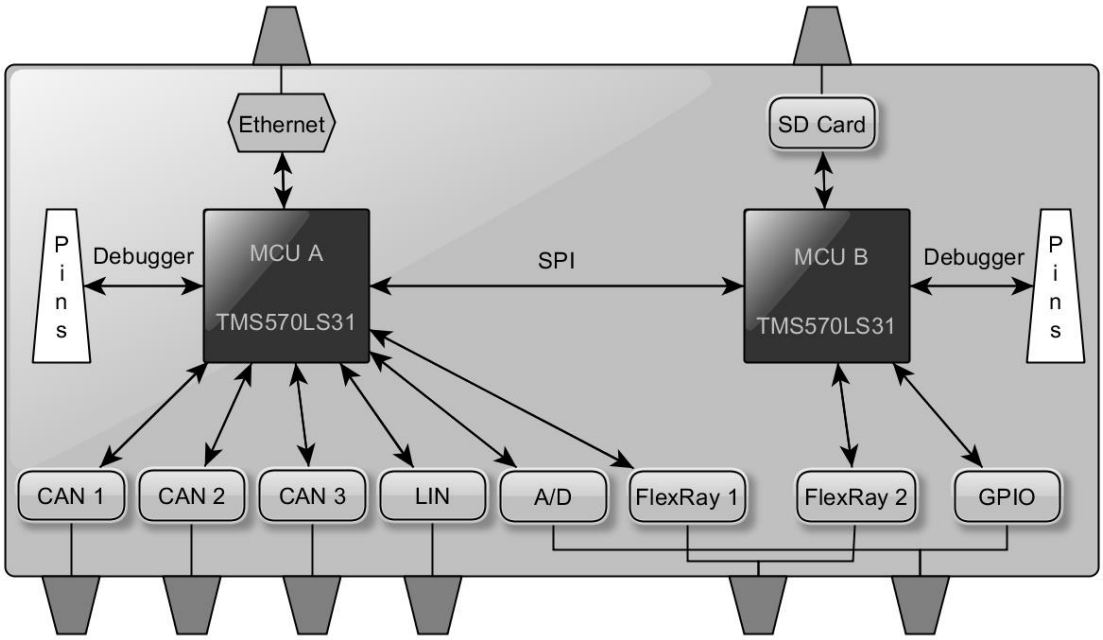
1.9. Ábra: FlexRay keret

2 A Gateway eszköz

A Gateway a ThyssenKrupp Presta Hungary Kft.-nél házon belül fejlesztett eszköz, amely az autóipari vezérlőegységek tesztelése során felmerülő kommunikációs igények kielégítésére készült. Képes analóg és digitális jelek feldolgozására, valamint generálására. Rendelkezik három CAN csatlakozóval, önállóan képes FlexRay hálózatot indítani (coldstart) és üzemeltetni, valamint LIN kommunikációs port-tal is el van látva. A PC-hez Etherneten kapcsolódik, és a funkcióknak eléréséhez a felhasználó számára készült egy magas szintű JAVA API. Konfigurációja akár SD kártyáról is beolvasható. Az eszköz folyamatos fejlesztés alatt áll mind a szoftver, mind a hardver tekintetében.

2.1 Általános felépítés

Az eszközön két Texas Instruments TMS570LS3137 típusú mikrokontroller található. A két vezérlő egymással SPI-on keresztül tud kommunikálni. Az „A” processzor a Master, így a kommunikáció irányítása ezen a kontrolleren zajlik. Az általános működés tekintetében is alárendelt szerepben van a „B” vezérlő, mivel az Ethernet kapcsolat is az „A”-ról van kivezetve, így ide érkeznek meg PC felől érkező utasítások, és itt állítódnak össze az UDP protokoll által használt csomagok. Mindkét processzor két maggal rendelkezik, melyek lockstep üzemmódban képesek egymás működését ellenőrizni. Ez lehetőséget biztosít ugyanazon utasítás kétszeri párhuzamos elvégzésére és nem konzisztens működés észlelése esetén hibajelzésre. A kontrollereken a FreeRTOS nyílt forráskódú beágyazott operációs rendszer üzemel. A FreeRTOS kezeli a létrehozott taszkokat, timereket és szemaforokat. További előnyei, hogy kevés memóriát használ és ingyenesen hozzáférhető csökkentve ezzel a fejlesztési költségeket.



2.1. Ábra: A Gateway

A CAN csatornák közül mind a három az „A” processzorról van kivezetve, ezért az alapvető funkciók működéséhez szükséges CAN driver csak ezen a kontrolleren foglal helyet. Ugyanez igaz az A/D átalakítóra, ami maximum 12 bites felbontásra képes. FlexRay driver mindkét mikrokontrolleren található, lehetővé téve ezzel a legalább két node-ot igénylő önálló működést. Általában a „B” processzoron működő FlexRay kontroller csak a hidegindításban vesz részt, és ugyan küldésre alkalmas, értesítést nem áll módunkban küldeni sem a fogadott keretekről (rxIndication), sem a küldés (txConfirmation) sikerességéről. Ennél fogva az adatkommunikáció egészét az „A” kontroller végzi. Rendelkezik még az eszköz LIN kivezetéssel (szintén az „A” processzoron), azonban erre a funkcióra még nem volt szükség így a driver jelenleg nincs implementálva. A szakdolgozat központi témáját képező üzenetkonverzió is csak CAN és FlexRay protokollokon működőképes. Szintén az „A” mikrovezérlő felel az Etherneten érkező UDP csomagok feldolgozásáért, valamint a szükséges adatok visszaküldéséért, valamint a „B” processzornak érkező utasítások továbbítása is a feladatai közé tartozik. Továbbá lehetőség van egy erre a célra kifejlesztett bootloaderrel PC-ről frissíteni a Gateway szoftverét, ezzel minimálisra csökkentve a fejlesztéshez és későbbi használathoz szükséges eszközigényt.

A „B” processzorról kerültek kivezetésre a GPIO kimenetek, amelyek szintén segítséget tudnak nyújtani az autóipari vezérlőegységek teszteléséhez (pl.: gyújtáskapcsoló). A nyolc általános célú láb egyszerű jelek küldésére és fogadására

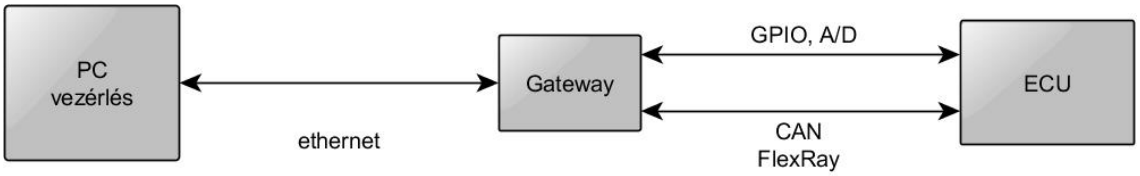
egyaránt konfigurálható. A „B” kontrolleren kapott helyet a fentebb említett főleg indítási célokat ellátó második FlexRay vezérlő, továbbá az SD kártya elérése SPI kapcsolat segítségével is ennek a processzornak a feladata. Az SD kártya felhasználásával PC nélkül konfigurálhatóvá tehető a Gateway, így teljes mértékben hordozható eszközként van lehetőség a használatára.

2.2 Felhasználási lehetőségek

A Gateway széles körben alkalmazható autóipari vezérlőegységek tesztelésére. A tesztelés során szükség lehet a tápfeszültség, a gyújtáskapcsoló automatikus ki és bekapcsolására, valamint bizonyos analóg jelek mérésére is. A vezérlőegységet körülvevő többi ECU kommunikációs jeleinek szimulálása gyakori feladat, ez az úgynevezett Restbus szimuláció. A Restbus szimuláció esetében szükséges lehet egy PC-től független felhasználói interfész biztosítása a tesztelés befolyásolására. ECU-k tesztelésénél és kommunikáció szimulálásánál az A/D és a GPIO csatlakozók lehetővé teszik a jelek mérését, és ha megkívánja a tesztelési folyamat, akkor lehetőség van beavatkozni. Szemléletes példa egy autó műszerfalát szimuláló hardver, ahol potenciométerekkel lehet állítani a fordulatszámmérő, a sebességmérő és egyéb analóg funkciók jeleit. Hasonlóképpen kapcsolókkal és gombokkal az egyéb funkciókat akár automatizáltan is lehet szimulálni és LED-ek segítségével visszajelzést is kaphatunk a tesztelt ECU-tól (pl.: hibajelző lámpa, gyújtáskapcsoló állása).

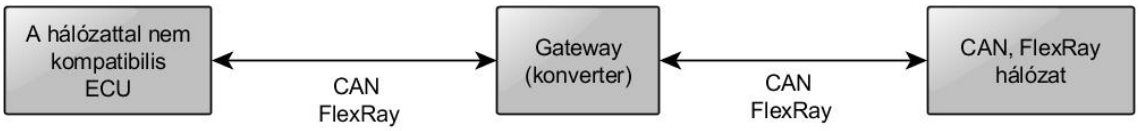
2.2. Ábra: PC nélküli alkalmazás

Restbus szimuláció megvalósítására is lehetőséget biztosít a Gateway. CAN és FlexRay protokollokon tetszőleges üzenetek ütemezett küldése egyszerűen konfigurálható a magas szintű API segítségével. Miközben a buszon folyó kommunikáció is monitorozható, megkönnyítve a tesztelés folyamatát.



2.3. Ábra: Restbus szimuláció

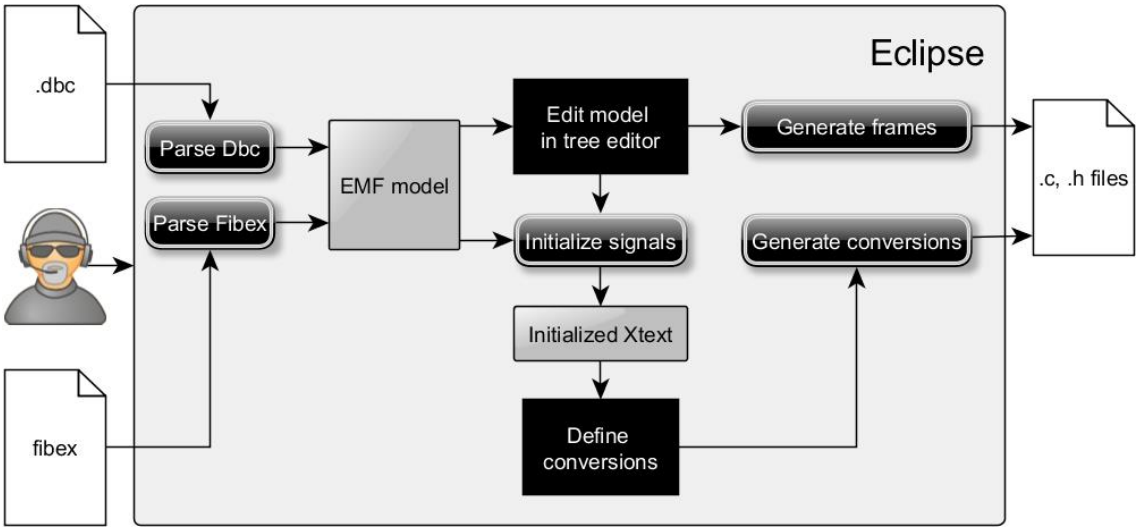
A nagyteljesítményű kontrollereknek köszönhetően lehetséges a fogadott és az elküldött üzenetek által tartalmazott adatok között különböző műveletek végrehajtása. A következő fejezetben egy erre a célra szolgáló szignál konverter implementáció kerül bemutatásra.



2.4. Ábra: Szignál konverter alkalmazás

3 A konverter implementálása

A szakdolgozat témája egy PC segítségével használható, a Gateway architektúrájának mélyebb ismerete nélkül konfigurálható konverziós eszköz megvalósítása. Gyakran felmerülő igény, egy már meglévő hardver integrálása egy új gépjármű hálózatába. Az eszköznek képesnek kell lennie fogadni mindkét oldal üzeneteit és a megfelelő konverzió elvégzése után az adott hálózat által feldolgozható formában továbbítani azokat. Eddig ez egy tisztán beágyazott kódot jelentett, ami nehezen volt változtatható ezzel jelentősen lassítva az integrációt. A következő fejezetben bemutatásra kerülő szoftvercsomag képes üzenetleírókból a Gateway architektúrájához jól igazodó modell feltöltésére. A modell leírja a szignálok, adatkeretek és a protokollok konfigurációit. Tetszőlegesen és könnyedén szerkeszthető, minden paraméter kényelmesen, gyorsan megváltoztatható és az esetlegesen hiányzó paraméterek pótolhatók. A szerkesztett modellből generálható Gatewayen futtatható kód tartalmazza az ütemezett küldés konfigurációját és a várt üzenetek minden szükséges beállítását. A szignálok közötti kapcsolatok leírására egy domain specifikus nyelv szolgál. A felépült modellből megkapjuk a szignálok nevét és a nyelv használatával egyszerűen tudunk konverziós kapcsolatokat leírni, majd ebből szintén „C” kódot generálni, a háttérben épülő modell feldolgozásával. A kimenet fordítható és közvetlenül feltölthető a Gateway-re, akár a rendelkezésre álló Ethernet kapcsolaton keresztül is. Az így kapott eszköz valósidejű üzenetkonverzióra képes, a gépjármű kommunikálni tud az eredetileg nem kompatibilis vezérlőegységgel.



3.1. A szoftver használatának menete

3.1 A beágyazott oldal

A Gateway-en futó szoftver induláskor inicializálja a szükséges modulokat. Alapesetben ez azt jelenti, hogy ezek lefutása után lehetséges Ethernet kapcsolatot használva a PC-vel kommunikálni. Használhatóvá válik a CAN, a FlexRay, a GPIO pinek, az AD átalakító és a többi periféria. Induláskor a két processzor egymástól függetlenül indul és a „B” kontroller csak az „A”-tól kapott utasításokra reagál.

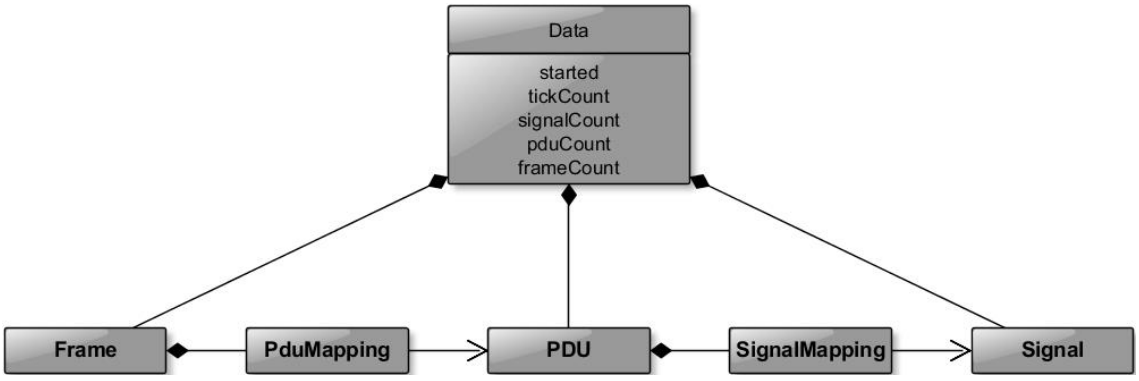
Az üzenetkonverzió esetében azonban a „B” passzív, nem kap utasítást. Egyetlen feladata szükség esetén a FlexRay elindítása. Más perifériát egyáltalán nem használ. Az „A” mikrovezérlő is csak a szükséges modulokat inicializálja. A működéshez nélkülözhetetlen modulokon felül csak a CAN és a FlexRay kontrollereket vezérlő CANDriver és FlexRayDriver válik aktívvá. Az ebben a fejezetben bemutatásra kerülő fogadott üzenetekből szignálokat kinyerő FrameDisassembler és az ütemezett küldésért felelős Restbus modulok a drivereket használják fel a működésükhöz. Ezekre támaszkodik a szoftverbe illesztett generált kód.

3.1.1 A fogadott üzenetek kezelése

3.1.1.1 A FrameDisassember modul

A FrameDisassembler modul végzi a beérkező üzenetek feldolgozását. Inicializálás után meg kell adnunk a feldolgozandó üzenetek egyes paramétereit és mindenekelőtt fel kell iratkoznunk a megszakításokat kezelő EventManager modul jelzéseire.

CAN vagy FlexRay csatornán fogadott üzenet egy megszakítást okoz, így a modul megkapja a keretet, amelyből ki tudjuk nyerni a szignálokat. A keretekben úgynevezett PDU-k foglalnak helyet. PDU-k szintjén definiálható három féle típusú „E2E" védelem, amely hozzárendel az üzenetekhez egy számlálót és küldés esetén kiszámítja a CRC mező értékét. Fogadáskor a CRC ismételten kiszámítása után ennek , illetve az üzenetszámlálónak felhasználásával hitelesítésre kerül a üzenet. CAN esetében egy, FlexRay esetében általában egy, de esetenként több PDU is található egy frame-ben. Ezek pontos elhelyezkedésének tárolására szolgálnak a PDU leírók (PduMapping). A szignálok a legkisebb értelmezett adategységek, amelyeket a PDU-k tartalmaznak. Tetszőleges hosszúságúak lehetnek és tetszőleges ponton helyezkedhetnek el az őket tartalmazó PDU-ban. Ezeket az információkat szintén leírók tartalmazzák (SignalMapping). A leírókhoz mindig egy tartalmazott PDU vagy szignál tartozik, ennek ellenére szükséges szétválasztani bizonyos paramétereket, mert lehetséges hogy ugyanaz a szignál vagy PDU két különböző üzenetnek is részét képezi, máshol elhelyezkedve a frame-ben. A sikeres feldolgozáshoz ismernünk kell, hogy hány üzenet fog érkezni, melyik csatornán fogadjuk, milyen típusú védelemmel van ellátva és pontosan tudnunk kell a szignálok hosszúságát és az üzenetben elfoglalt helyét.



3.2. Ábra: FrameDisassembler adatszerkezet

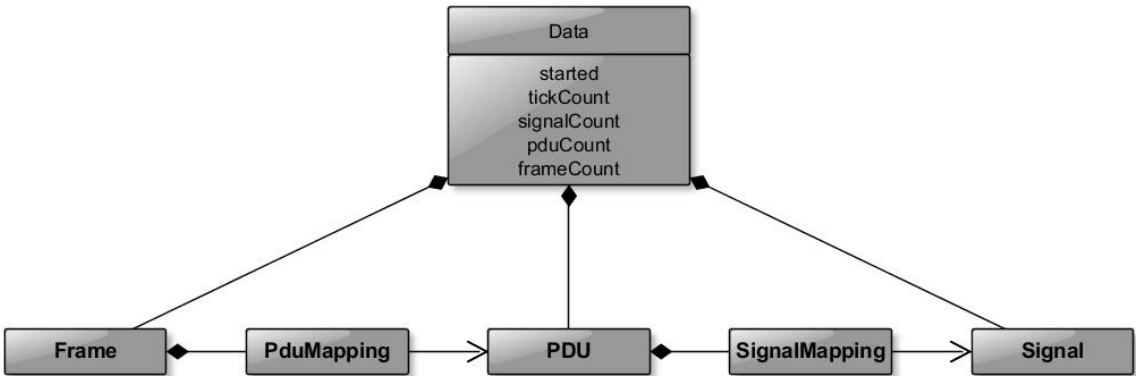
A konfiguráláshoz használt API függvények (minden függvény FrameDisasselmler prefixummal kezdődik). A bemutatásra kerülő függvények célja a fenti adatszerkezet manipulálása:

- AddSignal: Megadjuk egy szignál hosszát, kezdeti értékét és egy „timeout" paramétert, amivel jelezzük az érték validitását. A függvény hatására létrejön egy szignál entitás.

- AddPdu: Egy PDU-nál szükséges tudnunk a CRC számítás módját, hogy ellenőrizzük az integritást (nem kötelező) és a méretét. Létrejön a fenti ábrán látható PDU egy példánya.
- AddFrame: Egy FrameDisassembler modul által használt keret esetében tudnunk kell melyik beérkező üzenethez tartozik, melyik csatornán érkezhet és mekkora a hossza. Ennek a függvénynek a segítségével jön létre egy Frame.
- AddPduToFrame: A fenti ábrán látható PduMapping létrehozására szolgáló függvény. Egy frame és egy PDU közötti kapcsolat esetében szükség van a PDU pozíciójára, továbbá a tartalmazott szignálok frissítésének szükségességét jelző bit helyére. Szintén fontos paraméter az MSB és az LSB elhelyezkedésére utaló beállítás.
- AddSignalToPdu: A szignálok és a PDU-k közötti kapcsolatot leíró entitás létrehozására alkalmas. A szignál PDU-hoz rendelése teljesen azonos paramétereket igényel, mint a fentebb leírt, azzal a különbséggel, hogy a pozíciót bitben kell megadni.

3.1.2 Az üzenetek ütemezett küldése

A Restbus modul végzi a felkonfigurált üzenetek küldését. A felépítése és a használata majdnem teljesen megegyezik a Disassembler modulnál leírtakkal. A használt adatszerkezetet ugyanúgy framek, PDU-k és szignálok konfigurációjának tárolására használjuk, illetve ezek összerendelésére leírók szolgálnak a frame-PDU és a PDU-szignál kapcsolatok esetében. A számunkra legfontosabb adategységek a szignálok. Ezek hordozzák a kommunikáció a fizikai tartalommal bíró információkat. Minden szignálnak saját periódusideje van, ehhez igazodik az értékük frissítése. A kiszámíthatóság érdekében egy időzítő segítségével ütemezzük mind a CAN, mind a FlexRay protokoll küldését. Ezt egy 1 milliszekundumos időzítővel tesszük, szükség esetén utóosztókkal módosítva.



3.3. Ábra: Restbus adatszerkezet

A Restbus által biztosított konfigurációs függvények, amelyek a Disassembler-nél is látható, ugyanolyan felépítésű adatszerkezet feltöltését végzik:

- AddSignal: A függvény létrehoz egy példányt az ábrán látható a Sigal „osztályból". Megadjuk egy szignál hosszát, kezdeti értékét és lehetőség van egy konverziós függvény hozzáadására. Az utóbbi funkciót nem használjuk, mert magas szinten kényelmesebb átskálázni a szignálok értékét.
- AddPdu: A függvény egy PDU entitást hoz létre. A PDU-k esetében beállíthatunk három féle CRC számítási módot és megadjuk a hosszt.
- AddFrame: Ennek a függvénynek a segítségével adjuk meg, az üzenet periódusát és az általa használt csatornát, illetve protokollt. Minden frame esetében lehetőség van az ütemezett küldés kezdetének késleltetésére is. Meghíváskor egy Frame-et példányosít.
- AddPduToFrame és AddSignalToPdu: A paraméterek és a működés tekintetében is megegyeznek a Disassembler által használtakkal.

3.1.3 Az üzenetkonverzió

Két azonos funkciót ellátó, de más gyártótól származó ECU általában ugyanazok a fizikai paraméterek használatát igényli, azonban a kommunikáció adott esetben használhat más protokollt, más hosszúságú szignálokat, más kereten belüli elhelyezkedést és más értékek hordozhatják ugyanazon információt. Az üzenetkonverzió első sorban a különböző vezérlőegységek hasonló, vagy azonos fizikai tartalommal bíró szignáljainak a feldolgozhatóvá tétele, a fogadó számára. Az üzenetleírókból nyert információkat több lépésben szerkesztheti a felhasználó. A

folyamat vége egy generált kód, ami magas szintű funkciók használatával készíthető el, mivel a beágyazott „C” nyelvű implementáció időigényességének leküzdése a cél.

A Gateway elindulás után inicializálja a szükséges modulokat és létrehoz egy taszkot, ami a kommunikáció megszakadása esetén újraindítja a FlexRay-t. Az újraindítás kellően gyorsan történik ahhoz, hogy ez kívülről ne legyen észlelhető. Ezután fut le a generált kód azon része, ami a modellből készül. Itt történik a szükséges CAN és FlexRay üzenetek létrehozása. A különböző küldő és fogadó kereteket felhasználva a 3.1.1 és a 3.1.2 fejezetekben bemutatott függvények segítségével elkészül a Restbus és a Disassembler konfiguráció, így válik lehetségessé a megfelelő szignálok küldése és fogadása. A szignálok manipulálásához mindegyikhez eltárolunk egy egyedi azonosítót és az ezeket tartalmazó struktúrát használja fel a konverter.

A konverziókat megvalósító függvények minden esetben ellenőrzik a feldolgozott szignálok validitását, azaz megvizsgálják, hogy a kapott érték értelmezhető (pl.: járműsebesség esetén nem fogadunk el negatív értéket) és nem elavult. Ennek függvényében végrehajtódnak az előírt műveletek. Mivel egyszerre több szignál is érkezhet és vannak olyan összetartozó szignálcsoportok, amiknek az értékeit csak együtt lehet módosítani, kölcsönös kizárást kell biztosítani a konverziós függvények között. Erre a célra szemaforok szolgálnak. A konverziós függvényeket egy taszk hívja meg előre beállított, kellően gyors ütemezéssel, azaz garantálni kell, hogy minden szignál időben küldésre kerüljön.

3.2 PC oldali konfigurációs eszköz

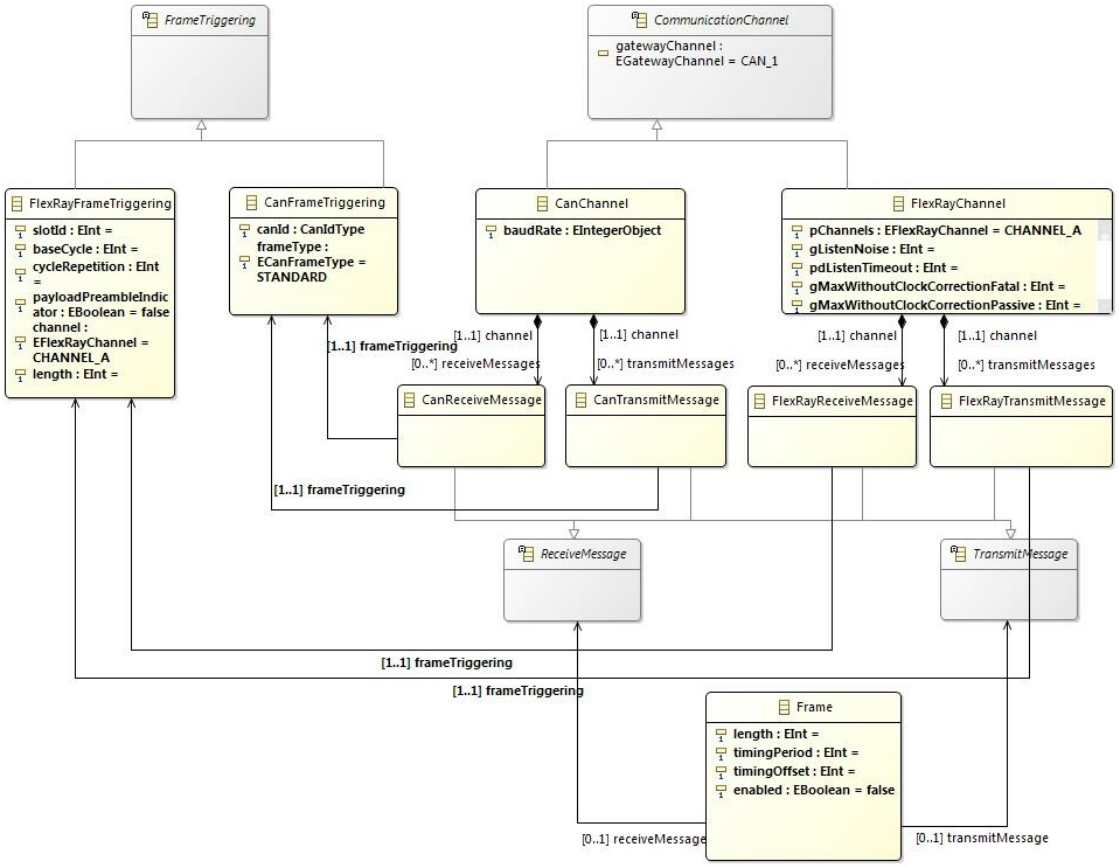
3.2.1 EMF modellezés

Az Eclipse modellező keretrendszer (Eclipse Modeling Framework) egy adatszerkezet alapú fejlesztést támogató eszköz. Képes UML, JAVA vagy XML nyelvek segítségével specifikált modelleket importálni, ezzel egységes felületet biztosítva kezelésükhöz. [4] A Gateway CAN és FlexRay kommunikációjának és az ezekhez kötődő szolgáltatásokhoz szükséges adatszerkezetek modellezése grafikusan, EMF editor használatával történt. Átlátható és könnyen, gyorsan szerkeszthető a rendelkezésre álló grafikus felület segítségével. [6]

3.2.1.1 A modell bemutatása

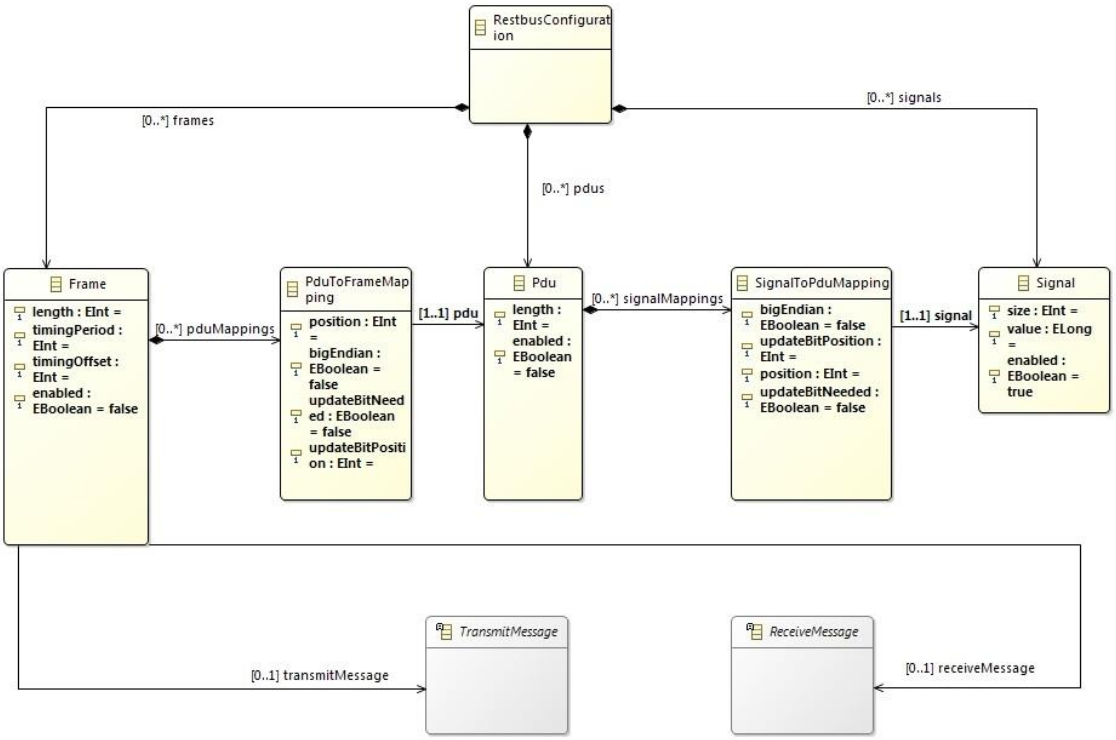
A modell tervezésekor több fontos szempontot kellett szem előtt tartani. Az üzenetleírókhoz hasonló módon kell tárolni a paramétereket, ugyanakkor a Gateway szoftverkomponenseit érintő részek esetében kézenfekvőbb a beágyazott szoftver működését alapul venni. A modell alapvetően egy köztes állapot, a fő célja a konfiguráció egyszerűbbé és gyorsabbá tétele. Ahogy a 3.1.-es ábrán látható, az üzenetleírókból közvetlenül generálható, majd ebből készül a generált kód egyik fele és innen töltjük be a szignálok neveit a konverziós műveletek leírásához. Jelenleg csak az üzenetkonverzióhoz szorosan kötődő elemeket tartalmazza, de később minden a Gateway-en elérhető funkció valamilyen szintű leírását is képesnek kell lennie tárolni. Azaz a jövőbeni bővíthetőség is szerepet játszott a megalkotásakor.

A modell lényegében két részre osztható, amelyek között természetesen több kapcsolat is található. Az egyik fő rész tartalmazza a különböző kommunikációs csatornák driver szintű beállításait, a protokollok pontos konfigurációját és az üzenetek létrehozásához szükséges alapvető információkat. A létrehozott üzeneteket tartalmazó objektumok pontosan leírják, hogy az adott kontroller milyen paraméterekkel üzemel és ez az információ felhasználható a kódgeneráláshoz is. Ezek az üzenetek úgynevezett „FrameTriggering” típusokat hivatkoznak. Ezek mutatják meg egy adott alacsonyszintű, küldést vagy fogadást lehetővé tevő üzenetobjektum paramétereit.



3.4. Ábra: Driver szintű beállítások és FrameTriggering-ek modellezve (részlet)

A másik önállóan bemutatható modellrészlet nagy mértékben követi a beágyazott résznél tárgyalt adatszerkezetek felépítését. Mivel a bejövő üzeneteket feldolgozó és az ütemezett küldést ellátó modulok adatszerkezete csak kismértékű eltérést mutat, kézenfekvő volt a modellben egy közös leírást biztosítani. A „Frame” által hivatkozott küldendő vagy fogadandó üzenet egyértelműen meghatározza annak irányát. Erre azért van szükség, hogy az alacsonyszintű driveren belül azonosítani lehessen az üzenetobjektumot.



3.5. Ábra: Egy tárolt üzenet meta-modellje (részlet)

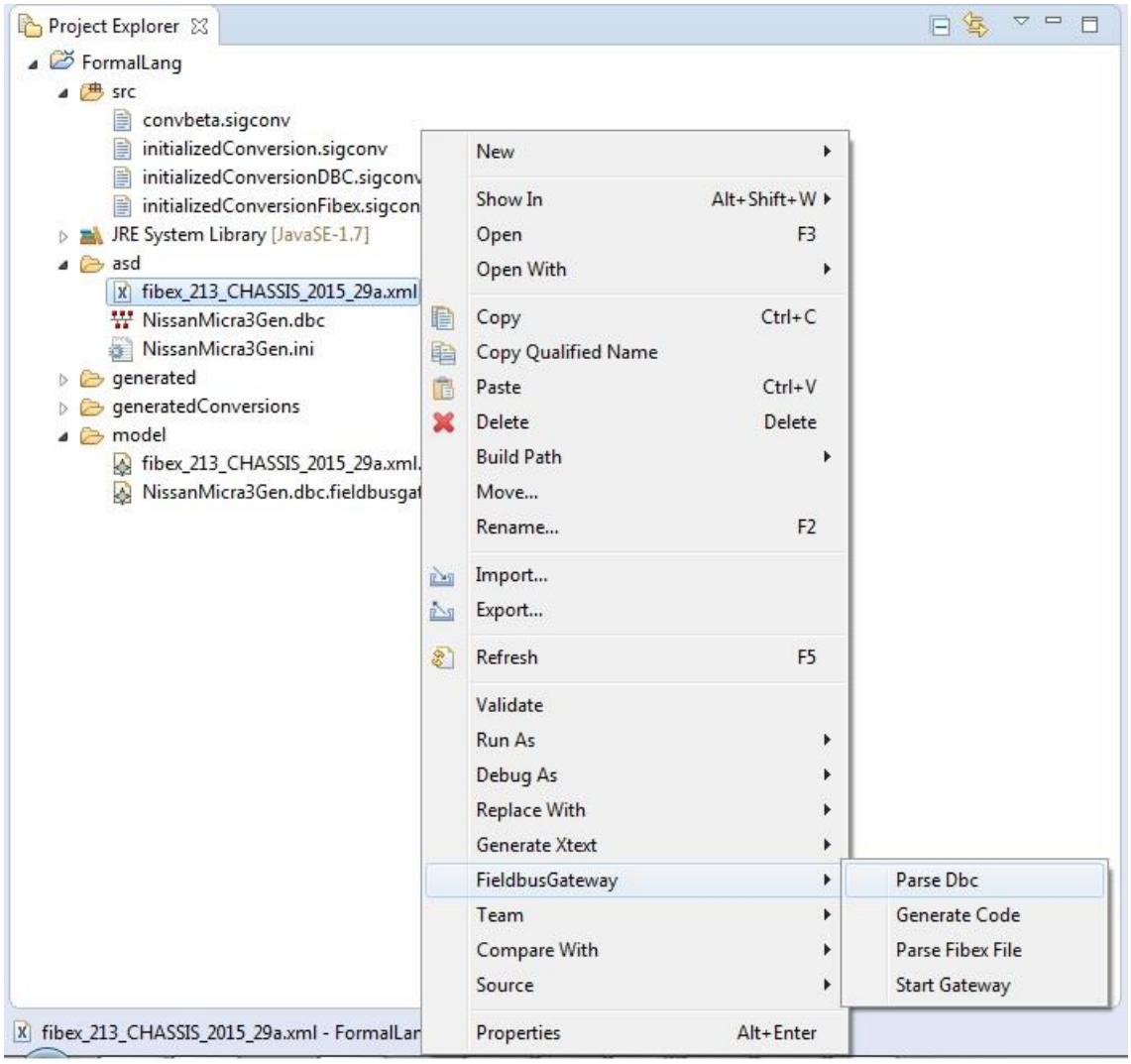
3.2.1.2 A modell bejárása (feltöltés XML fájlból)

A vezérlőegységek kommunikációját tartalmazó üzenetleírók (CAN kommunikáció esetén jellemzően DBC, FlexRay-nél fibex) beolvasására rendelkezésre állt egy plugin, ami a fentebb említett EMF modell feltöltéséhez elegendő információt szolgáltat. Ez a megoldás azonban egy más felépítésű adatszerkezetben tárolja a beolvasott adatokat. Ennek ellenére célszerű volt ebből kiindulni, mivel ez egy kész és könnyen kezelhető megvalósítás.

A bejárást végző metódus a gyökérelemből kiindulva halad végig a modellen és ezzel párhuzamosan vizsgálja a beolvasott adatszerkezetet. A feladat nagyon hasonlít szövegbeolvasó működéséhez, lényegében egy adatstruktúra konverzió került megvalósításra. Az üzenetleírókból minden hasznos információ betöltődik a modellbe, így az Eclipse modellező keretrendszer minden szükséges szolgáltatása a rendelkezésünkre áll. Ez a megvalósított szoftverben - Eclipse környezetben - egy alkalmas fibex vagy DBC fájlra kattintáskor egy almenüpontból nagyon egyszerűen és gyorsan kivitelezhető.

3.2.1.3 A modell módosítása, kiegészítése Eclipse alatt

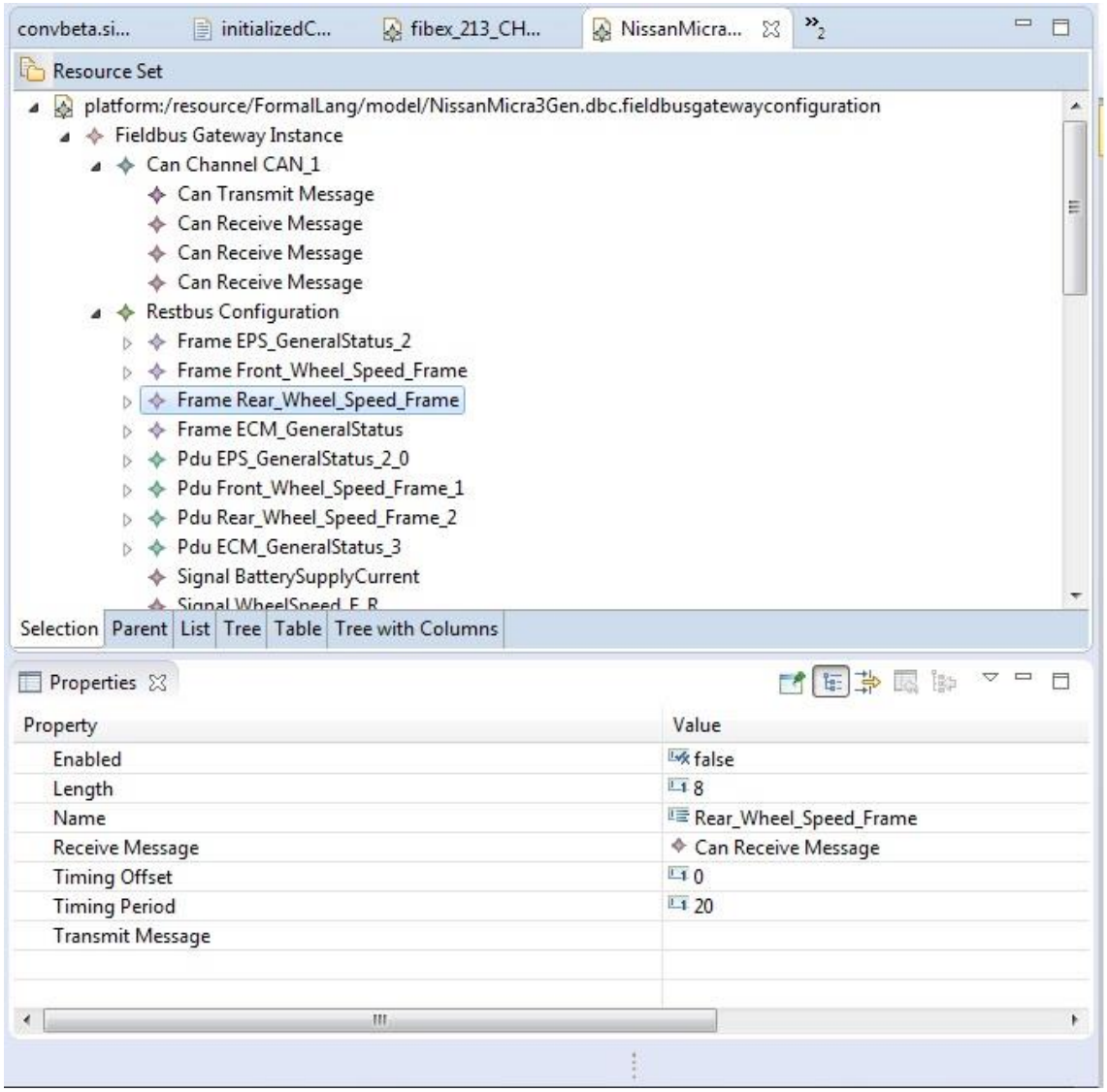
A felépült modell gyakran kiegészítésre szorul, vagy a szükségesnél jóval több üzenetet tartalmaz. Egy konverziós feladathoz általában nincs szükség az összes feldolgozott frame-re, mivel az alapvető funkciók biztosítása a leggyakoribb cél. Előfordulhat olyan eset is, amikor a felhasznált üzenetleíró hibákat tartalmaz. Egy Eclipse alapú keretrendszer biztosítja számunkra a kényelmesen kezelhető környezetet. A „Parse fibex” vagy a „Parse dbc” menüpontok egyikét kiválasztva kapunk egy faszerkezetű adatbázist.



3.6. Ábra: Kezelés Runtime Eclipse segítségével: választható menüpontok

Ennek a szerkesztéséhez, rendelkezésre áll egy „Tree editor” nézet, amit az EMF biztosít. Minden létrejött objektum összes paramétere megváltoztatható, a hivatkozások módosíthatók és felvehető tetszőleges új elem a modellbe. Üzenetleíró hiányában akár

üres modellből is kiindulhatunk, azaz tetszőleges konfiguráció felépítésére is van lehetőség.



3.7. Ábra: Egy leíróból feltöltött modell, szerkesztés Tree editorral

3.2.2 A konverzió leírása Xtext modellező eszközökkel

Az Xtext egy nyílt forráskódú keretrendszer programozási nyelvek és domain specifikus nyelvek fejlesztéséhez. Nagy erőssége, hogy a szövegértelmezőn felül generál egy osztálymodellt az absztrakt szintaxis fa reprezentálására. [5] Mivel a tervezett nyelv segítségével megírt konverziók „C” kód formájában fognak hasznosulni, a háttérben épülő modell könnyű bejárhatósága nagy segítséget nyújt.

A szignálok közötti kapcsolatok leírására alkalmas nyelv három típust támogat. A legegyszerűbb adattípus a szignál (signal), ez mindössze egy név, amire később

tudunk hivatkozni, így a kód-kiegészítés funkciót kihasználva a nyelv használata rendkívül gyorssá válik.

```
Signal:
	'signal' name = ID
;
```

A kód-kiegészítést az Xtext támogatja. A második típus a konstans (const), ami szintén kényelmi funkciót tölt be. Egy névvel hivatkozható értéket lehetséges definiálni a használatával.

```
Const:
	'const' name = ID value = INT
;
```

Funkciója megegyezik egy „C” nyelvből ismert makróéval. A legfontosabb típus maga a konverzió (conversion). Minden konverziónak kötelező nevet adni és a törzsét kapcsos zárójelek között kifejteni. Tartalmazhat szignálokra vonatkozó műveleteket és if-else állításokat amelyeken belül további műveletek definiálhatók.

```
Conversion:
	'conversion' name = ID
	'{'
		(conversionExpression += ConversionExpression ';')*
		(ifStatement += IfStatement)*
	'}'
;
```

Egy ilyen kifejezés egy szignál nevével és ezt követően egy egyenlőségjellel kezdődik, majd tetszőleges módon zárójelek és a négy alapművelet felhasználásával írható le a kívánt konverzió. Ezekhez felhasználhatóak az előzetesen definiált konstansok és szignál nevek, illetve egész számok is. Egy „if” feltétel megadásakor az első operandus egy név vagy szám lehet, amit egy egyenlő, nem egyenlő, kisebb, kisebb-egyenlő, nagyobb, nagyobb-egyenlő operátor követ majd a jobb oldali operandus, ami egy kifejezés is lehet. A jobb oldal minden esetben zárójelek között kell szerepeljen az egyszerűbb olvashatóság érdekében.

```
IfStatement:
	'if' '(' operandLeft = Universal operator=OPERATOR operandRight =
WithParenthesis  ')'
	'{'
		(conversionExpression += ConversionExpression ';')?

		(ifStatement += IfStatement)*

	'}'
	(elseStatement = ElseStatement)?
;
```

Üres törzsű „if” feltétel nem hozható létre, azonban az „else” ág nem kötelező. A komplexebb konverziók leírását elősegítendő tetszőleges számú feltétel ágyazható egymásba és ezek bármennyi (egynél több) kifejezést tartalmazhatnak.

Alább látható egy példa a nyelvtan használatára:

```
signal WheelSpeed_F_R
signal VehicleSpeed_284
signal MessageCounterRear
signal WheelSpeed_R_R

const CONST2 2
const CONST5 5

conversion
Conv3
{
	WheelSpeed_R_R = ( VehicleSpeed_284 * CONST2 / 54 + ( 45 + ( CONST2 +
( VehicleSpeed_284 ) ) ) * ( 500 / ( CONST2 - MessageCounterRear ) ) );

	if ( MessageCounterRear  != (CONST5))
	{
		VehicleSpeed_284 = 45 * WheelSpeed_F_R ;
	}

}
```

3.2.3 A kódgenerátor

A kódgenerátor elsődleges feladata a Gateway szoftverébe illeszthető, annak szolgáltatásaira támaszkodó szignálkonverzió létrehozása. A beolvasott adatokat felhasználva kimenetként kapott fájlok fordíthatók és feltölthetők az eszközre.

A legegyszerűbb funkciója a konverziók leírásának inicializációja. Ez lényegében a modellben tárolt szignálok kigyűjtését jelenti, amik alapján létrehozza az összes „signal” típusú hivatkozható elemet, így csak a konverziók és a konstansok maradnak a felhasználóra.

Az EMF modell képes tárolni a CAN és a FlexRay kontrollerek beállításait így ezek alapján az alacsony szintű driverek konfigurálása generálható. A kommunikációban résztvevő üzenetek paraméterei szintén kinyerhetők a feltöltött modellből. Az generátor egyik fő metódusa egy gyökérelemet kap és ebből kiindulva készíti el a kódot. Létrehozza a kontroller beállításokat, az include-okat majd az üzenetobjektumokat. Ezután bejárja a fogadott üzeneteket és kitöltött FrameDisassembler függvényeket generál minden frame, PDU és szignál kapcsolatot figyelembe véve. Ezt követően a küldendő üzenetekhez is elkészül a kód. Végül a fájl

végére kerülnek a konverziót inicializáló, valamint FlexRayt a Disassemblert és a Restbust elindító függvényhívások.

A konverterhez kapcsolódó fájlok generálása a másik fontos funkció. Az Xtext segítségével definiált nyelv használatakor a háttérben modell épül. Ebben a modellben megtalálható minden információ, amit a felhasználó a konverziókhoz definiált. Ezt bejárva generálódik a „SignalCoversion_Priv.h” a „SignalConversion.h” és a „SignalConversion.c”. A „Priv” posztfixel ellátott fájl a konverziós taszk ütemezését tartalmazza. Ez alapértelmezett módon 1 milliszekundum. A „SignalConversion.h” egy struktúrát definiál, ami az összes használt szignál azonosításához használható, valamint a függvények prototípusait tartalmazza. A forrásfájl az include-ok és a globális változók után a konstansokat makróként írja le, majd a szignálok nevével létrehozott statikus „SignalType” változók foglalnak helyet. Ezek után következnek a konverziókat megvalósító függvények. A nevük tartalmazza az Xtext-ben megadott egyedi elnevezésüket, visszatérési értékkel és bemeneti paraméterekkel nem rendelkeznek. Törzsükben helyezkednek el a definiált műveletek és „if-else” kifejezések, kiegészítve a szignálok validitásának ellenőrzésével. Ez lényegében a kapott adat érvényességének és hibamentességének a vizsgálata. A fájl végén kapnak helyet a konvertert működtető függvények. Az inicializáló függvényben létrejön az ütemező taszk, a start függvény pedig megfelelteti a Restbus és a Disassembler szignáljait a konverter által használtakkal. A stop függvény lehetőséget nyújt, a konverzió leállítására, azonban ez a funkció általában nincs használva.

4 Összegzés

A szakdolgozatban bemutatott eszköz segítségével összeköthetőek különböző módon kommunikáló vezérlőegységek. Az implementációhoz szükség volt a Gateway hardver megismerésére, a beágyazott kód megértésére. Meg kellett valósítani az üzenetek beolvasását végző szoftvermodult és ugyancsak alacsony szinten a szignálok közötti konverziót. Elengedhetetlen volt alaposan áttanulmányozni a CAN és a FlexRay hálózatok működését, hiszen a driverek használata elengedhetetlen volt a feladathoz. Meg kellett ismerni az üzenetleíró adatbázisok tartalmát és az ezekből beolvasott információkat szerkeszthető formában eltárolni. Szükséges volt elsajátítani az EMF modellezés módszereit és ennek segítségével meg kellett tervezni egy a Gateway segítségével végezhető CAN és FlexRay kommunikáció minden paraméterét tartalmazó adatszerkezetet. A konverziók kényelmes leírásához az Xtext ismeretek elsajátítása volt szükséges. Implementáltam egy nyelvtant, ami ehhez nyújt megfelelő környezetet. Továbbá JAVA programozási nyelv használatára is szükség volt a modellek bejárásának elvégzésekor és a fordítható kód generálását végző program megvalósításakor. Az elkészített szoftverkomponensek használatát megkönnyítendő a kurzor használatával elérhető menüpontok kiválasztásával futtathatóak a magas szintű funkciók. Ez további jártasság szerzését igényelte Eclipse és JAVA ismeretek terén.

Továbbfejlesztésre több szempontból is szükség lehet. Még inkább könnyítené és gyorsítaná egy működő konverter elkészítését egy olyan funkció hozzáadása, ami gombnyomásra lefordítja a generált kódot a Gateway többi szükséges komponensével egyetemben. Létezik egy önállóan futtatható Bootloader alkalmazás, melynek integrációjával még kényelmesebbé válna a szoftver kezelése. A modellt ki lehet egészíteni további perifériákhoz kapcsolódó osztályokkal így bővülne a felhasználási terület és a megvalósított nyelvtan is tovább bővíthető, finomítható a felmerülő igényeknek megfelelően.

Irodalomjegyzék

[1] Vector Informatic GmbH. Introduction to FlexRay, https://elearning.vector.com/vl_flexray_introduction_en.html

[2] Vector Informatic GmbH. Introduction to CAN, https://elearning.vector.com/vl_can_introduction_en.html

[3] Vector Informatic GmbH. Introduction to LIN, https://elearning.vector.com/vl_lin_introduction_en.html

[4] Wikipedia: *Eclipse Modeling Framework*, https://en.wikipedia.org/wiki/Eclipse_Modeling_Framework (revision 11:05, 21 September 2015)

[5] Wikipedia: *Xtext*, https://en.wikipedia.org/wiki/Xtext (revision 04:44, 23 May 2016)

[6] Vogella GmbH. Eclipse Modeling Framework (EMF) – Tutorial, http://www.vogella.com/tutorials/EclipseEMF/article.html (02 September 2015)