SZAKDOLGOZAT-FELADAT

Kanál Attila Károly (KP0D48)
szigorló villamosmérnök hallgató részére

AUTOSAR hálózatmenedzsment-modulok megvalósítása

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, a valós idejű feladatok futtatására az OSEK operációs rendszer vagy a futási idejű monitorozást támogató XCP protokollcsalád. A vezető autógyártók által 2002-ben életre hívott AUTOSAR konzorcium célja az, hogy ezen szakterületi szabványokra építve specifikáljon egy (i) *alapvető szolgáltatásstruktúrát,* amely eltakarja a hardver sajátosságait és támogatja az alkalmazási szoftver hordozhatóságát (base software stack, BSW), (ii) egy *modellezési nyelvet* az ECU-kon futó alkalmazási szoftver szabványos leírására (software component template), és (iii) az alkalmazások és BSW-k ECU-n belüli és ECU-k közti *transzparens kommunikációját* lehetővé tevő elosztott runtime szolgáltatást (RTE):

- A *base software stack* magában foglalja az alacsony szintű eszközmeghajtókat (pl. EEPROM és Flash driverek), az ezeket eltakaró absztrakciós rétegeket (pl. memória absztrakciós felület) és az ezekre ültetett magas szintű funkciókat (pl. perzisztens adattárolás).
- A *modellezési nyelv* lehetővé teszi, hogy precízen specifikáljuk az adattípusokat, illetve az alkalmazást alkotó komponensek interface-eit és belső felépítését.
- Az *RTE* egy generált glue kód réteg, amely eltakarja az alkalmazáskomponensek elől, hogy az általuk fogadott vagy küldött információ pontosan hogyan jut el a forrástól a célig, potenciálisan ECU-k közötti kommunikációs buszok igénybevételével.

A konzorcium jelentős hangsúlyt fektet az *API-k szabványosítására,* de kifejezetten támogatja a versengést az egyes szolgáltatások *megvalósításában* („Cooperate on standards, compete on implementation"). Az AUTOSAR egy élő, aktívan fejlesztett szabvány, amelynek a közelmúltban jelent meg a 4.3-as verziója.

A jelölt feladata egy régebbi AUTOSAR Base Software Stack megvalósítás (4.0 szabvány verzió) hálózatmenedzsment funkcióinak továbbfejlesztése a legfrissebb szabványnak megfelelően az alábbiak szerint:

- *A szabvány kapcsolódó részeinek megismerése:* (i) ismertesse az AUTOSAR rétegzett BSW struktúráján belül a *hálózati kommunikációért felelős modulok szerepét,* különös tekintettel a hálózatmenedzsmentre, (ii) vázolja ezen modulok együttműködését egy olyan *forgatókönyv bemutatásával*, amelyben a vezérlőegység elindítja a hálózati kommunikációt, végül (iii) foglalja össze a hálózatmenedzsment modulok 4.0-s és 4.3-as szabványban tapasztalható különbségeit.
- *Szoftvertervezés és -megvalósítás:* (i) módosítsa a rendelkezésre álló 4.0 szabványverziónak megfelelő megvalósítás designját és megvalósítását, úgy, hogy az megfeleljen a 4.3-as szabványnak (érintett modulok: CAN Network Management,

FlexRay Network Management, illetve Network Management Interface). A szabvány a modulok megvalósítását egy *statikus* (kézzel írt, minden konfigurációban azonos) és egy *dinamikus* (konfigurációtól függő, tipikusan generált) részre bontással javasolja és megadja a konfigurációs adatok modelljét egy osztálydiagrammal. A dinamikus kódrészletek előállítása a szabványos adatmodellből az aktuális megvalósítás feladata úgy, hogy az illeszkedjen a statikus részhez és megfeleljen az alkalmazási terület erőforrás-használatra vonatkozó követelményeinek (pl. minimális ROM használat). A 4.0-ról 4.3-ra továbbfejlesztés igényelheti a generált adatszerkezetek megváltoztatását, ebben az esetben *tervezze meg* az új generátumot. A 4.0-hoz készült kódgenerátor egy mára elavultnak tekinthető kódszintézis technológián (Apache Velocity) alapul, amelyet célszerű korszerűsíteni, ezért *tervezze és valósítsa meg* a fenti modulok *kódgenerátorait* Java nyelven.

- *A megvalósítás tesztelése:* a vállalatnál rendelkezésre áll egy autógyártó *megfelelőségi teszt készlete*. Az Ön által fejlesztett modul helyességének vizsgálatához (i) illessze megvalósítását a tesztfuttató környezethez, ennek érdekében valósítsa meg a szükséges adaptereket és függvénycsonkokat, (ii) futtassa a teszteket, és győződjön meg arról, hogy megvalósítása megfelel a szabvány által elvártaknak, illetve (iii) szükség esetén javítsa a megvalósítást.

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

Budapest, 2017. szeptember 22.

……
Dr. Dabóczi Tamás
tanszékvezető

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

Kanál Attila Károly

AUTOSAR hálózatmenedzsment-modulok megvalósítása

KONZULENSEK

Hegyi Balázs
(thyssenkrupp Presta Hungary Kft.)

Dr. Sujbert László

BUDAPEST, 2017

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Kanál Attila Károly**, 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, 2017. 12. 08.

…….

Kanál Attila Károly

Kivonat

Napjaink modern gépjárműi egyre több elektronikus vezérlőegységgel rendelkeznek, amik egyes funkciókat szolgáltatnak. Ilyen funkciók lehetnek biztonság kritikusak, mint például a fék és a kormányrásegítő rendszer, vagy kényelmi szolgáltatásokért felelhetnek mint az ablaktörlő és az ülések fűtésszabályozója.

Ezek az elektromos vezérlőegységek (Electrical Control Unit, ECU) kommunikációs buszhálózatokon keresztül kommunikálnak egymással és hajtanak végre komplex feladatokat. Ennek a szabványosítására fejlesztették ki az AUTOSAR rétegzett szoftverarchitektúrát, ami a különböző beszállítók által gyártott vezérlőegységek közti kommunikációt teszi egységessé.

Az AUTOSAR szabvány 4.3-as verziója 2016 végén jelent meg. A feladatom az AUTOSAR szabványrendszer megismerése, valamint három hálózatmenedzselésért felelős 4.0-as szabvány szerint megvalósított modul implementációjának a módosítása és tesztelése, hogy megfeleljen a 4.3-as verzió követelményeinek. Ezen kívül a modulok konfigurációs kódjának elkészítését segítő konfiguráció generátorokat is el kell készítenem JAVA nyelven, amik az eddigi Apache Velocity sablonos kódgenerátorokat helyettesítik majd.

A szakdolgozatomban először az AUTOSAR szabványt mutatom be, annak réteges felépítését és magukat a rétegeket. Ez után a hálózatmenedzsment felépítéséről és működéséről írok, majd bemutatok egy kommunikációs forgatókönyvet, amiben kitérek annak hálózatmenedzsmenttel kapcsolatos részeire.

Következőnek bemutatom a három hálózatmenedzsment modult, a Network Management Interface-t, CAN Network Management-et és FlexRay Network Management-et. Itt részletezem a modulok funkcióit, a változásokat és azok implementálását.

Mindezek után a modulok konfigurációját ismertetem. Bemutatom miként készülnek el az új konfiguráció generátorok, és miért hasznosabb ez a megoldás.

A dolgozat legvégén a modulok helyes működését ellenőrző tesztelési lehetőségeket ismertetek, és ezekből kettőt, amit a dolgozat során felhasználok, részletesen bemutatok.

Abstract

Today's modern vehicles tend to have more and more electrical control units (ECUs), as time goes. These units are responsible for functions that control safety features like breaking and steering assisting systems. They also control the vehicle's comfort system, for example the windshield wipers and the seat heating system.

These control units are linked with communication buses, that utilize AUTOSAR, which is a software architecture, that was developed to make communication between the products of suppliers standardized.

The AUTOSAR version 4.3 was released in late 2016, and my task is to study it's architecture, communication stack and learn how network management works in this standard. After that I need to modify three network management modules' 4.0 version implementation to fit the specifications of version 4.3.

First, I will present AUTOSAR, it's architecture, communication stack, and show it's network management functions in details.

Then I describe the Network Management Interface, CAN Network Management and FlexRay Network Management modules' functions, the differences between version 4.0 and 4.3 and how these changes can be implemented.

Next chapter is about how AUTOSAR defines the configuration of these modules, and the process of making a configuration generator, that creates the configuration files.

Lastly I present the methods that are used to test these modules, and to make sure that everything fits the described specifications. I will go into details about two testing methods that I use to be certain that my modules function correctly.

1 Bevezető

2016 novemberében megjelent az AUTOSAR szabvány 4.3-as változata. A feladatom a 4.0-s és a legújabb verzió kommunikációs stack-jének, valamint a hálózatmenedzsment működésének a megismerése és összehasonlítása volt. Három hálózatmenedzsmenttel kapcsolatos modulra ki kell térnem részletesebben is. Ezek a Network Management Interface, CAN Network Management és FlexRay Network Management. Az első feladatom az ezeken történt változások implementálása a már elkészített 4.0-val kompatibilis megvalósítások segítségével.

A két verzió közötti változások lehetnek új funkciók megjelenései, amiket meg kell valósítani a szabványdokumentumok követelményeinek megfelelően (például egy új API), vagy esetleg már meglévő funkciók működésében történhettek változások, amikhez már meglévő elemeket kell módosítanom. Mindkét esetben előfordulhat az, amikor a modul adatszerkezetetét újra kell tervezni a változott funkciónak megfelelően.

A szakdolgozatom a thyssenkrupp Presta Hungary Kft-nél készítettem el, ahol rendelkezésemre álltak a feladat megvalósításához szükséges modulok 4.0-s implementációi, valamint a kódgeneráláshoz és teszteléshez szükséges fejlesztőkörnyezetek.

Először ismertetem az AUTOSAR 4.3-as verziójú szabványát, annak réteges felépítését, részletesen kitérve a kommunikációs stackre és az azon belüli hálózatmenedzselési funkciókra. Ez után a három hálózatmenedzsment modul 4.3-as verzióban leírt funkcióit részletezem és leírom hogyan módosultak azok a 4.0-hoz képest, valamint azt folyamatot amit a változások implementálásához követtem.

A három hálózatmenedzsment modul konfigurációját generáló egységeket is frissítenem kell, amihez bemutatom a modulok konfigurációs felépítését, és a régi megvalósítását a konfigurációgenerátornak. Leírom miért kell a konfigurációgenerátorokat is módosítanom, miért előnyösebb az új generátor, és azt hogyan lehet megvalósítani az új, más elven készített kódgenerátort.

Mindezek után meg kell győződni a modulok helyes viselkedéséről. Ehhez bemutatom a rendelkezésemre álló tesztelési lehetőségeket, azok elméleti hátterét, és magukat a tesztelési folyamatokat.

2 Az AUTOSAR elméleti háttere

A következő fejezetben az AUTOSAR 4.3-as verziójának a réteges szoftverarchitektúráját és a rétegek felépítését mutatom be. Ki fogok térni a rétegzett struktúrájú alapszoftverre (Basic Software), amiben ismertetem a kommunikációs és hálózatmenedzsment stackeket.

Az AUTOSAR (AUTomotive Open System Architecture) egy világszerte eltejedt autógyártó, elektronika, szoftver és félvezetők ágazatából származó cégek által létrehozott és fejlesztett szabványcsalád. 2002-ben alapították a Bosch, BMW, Continental, DaimlerChrysler és Volkswagen cégek közösen.

Az egyik célja az AUTOSAR-nak az alapfunkciók a szabványosítása. Ez teszi ehetővé, hogy az AUTOSAR-partnerek a járművek fejlesztésénél azok funkcióit integrálni, cserélni tudják a jármű hálózatán belül, valamint frissítsék vagy fejlesszék azt.

Az AUTOSAR alapú fejlesztés főbb előnyei közé tartozik az, hogy lehetővé válik a szoftverek újrafelhasználhatósága, rugalmasabbá válik a szoftverdizájn, és ezekre egyértelmű integrációs szabályok vonatkoznak. A szoftverfejlesztés ára így csökken hosszútávon, valamint a szoftver minősége növekszik, mivel ugyanannak a modulnak a tesztelésével több cég is foglalkozik. [1]

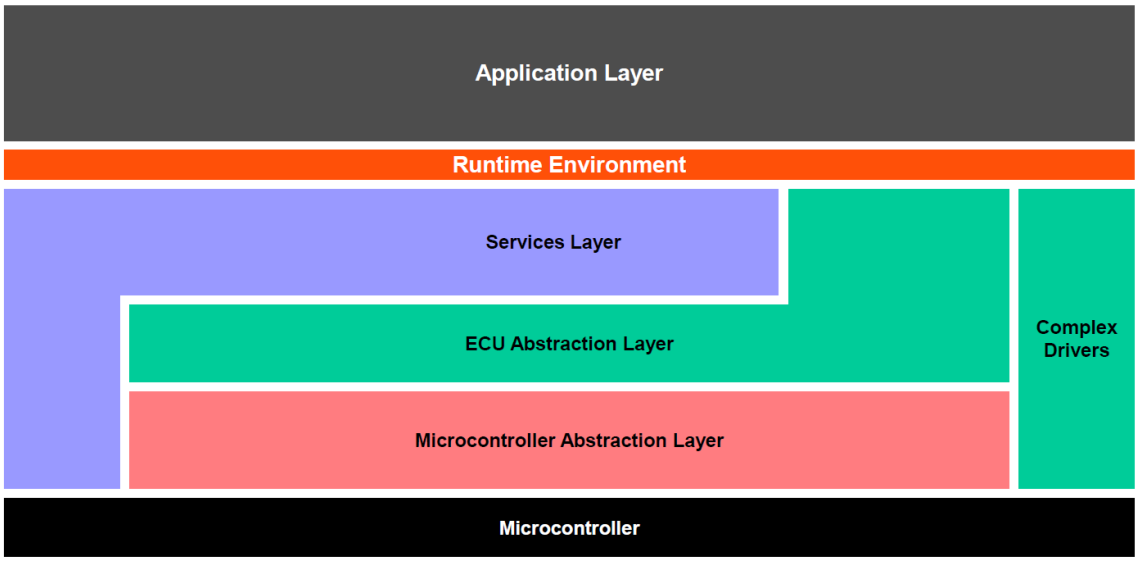
2.1 Az AUTOSAR réteges szoftver architektúrája

Az AUTOSAR architektúra a legmagasabb absztrakciós szinten három rétegre bontható:

- Alkalmazási réteg (Application Layer)
- Futtató környezet (Runtime Environment (RTE))
- Alapszoftver (Basic Software (BSW))

Az alapszoftver réteg további négy rétegre bontható:

- Szolgáltatási réteg (Services Layer)
- ECU absztrakciós réteg (ECU Abstraction layer)
- Mikrokontroller absztrakciós réteg (Microcontroller Abstraction Layer)
- Komplex driverek (Complex Drivers)



2.1. ábra AUTOSAR réteges szoftver architektúra [2]

Az alkalmazási réteg tartalmazza az AUTOSAR szerint megvalósított alkalmazási szintű szoftverkomponenseket, amik hardverfüggetlen elemek. A komponensek egymással az RTE rétegen keresztül tudnak kommunikálni.

Az RTE kommunikációs szolgáltatásokat nyújt az alkalmazási rétegnek, és függetlenné teszi az alkalmazás réteg szoftverkomponenseit a specifikus ECU-tól. Az RTE réteget ECU-tól függően egyénileg kell generálni, így a komponensek megvalósítása a hordozó ECU-tól teljesen függetlenné válik.

Az alapszoftver réteg feladata alkalmazási réteg szoftverkomponenseinek a kiszolgálása az RTE-n keresztül. Ez alapvetően egy moduláris felépítésű függvénykönyvtár, ami szabványos interfészekkel rendelkezik és feladata a mikrokontroller funkcióinak az elérhetővé tétele az AUTOSAR alkalmazás részére. [3]

Az alapszoftver réteget további rétegekre lehet bontani, ezekből a főbb elemek:

Mikrokontroller absztrakciós réteg: olyan belső driverek alkotják, amik közvetlen hozzáférést biztosítanak a mikrokontroller belső perifériáihoz. Feladata az, hogy a magasabb szintű szoftverrétegeket függetlenné tegye a mikrokontrollertől. Ez a réteg erősen függ a használt mikrokontrollertől.

ECU absztrakciós réteg: a mikrokontroller absztrakciós réteg driverjeinek hoz létre egy interfészt azok pozícióitól függetlenül (legyen az mikrokontrolleren belül vagy kívül). Ezt az interfészt a magasabb rétegek funkciói használják fel, hogy elérjék a mikrokontroller szolgáltatásait az ECU hardveres elrendezéstől függetlenül. A réteg maga mikrovezérlő független, de ECU hardverelemektől függő.

Komplex vezérlők: közvetlen elérést nyújt a szoftverkomponenseknek a hardverhez. A réteg lehetővé teszi, hogy speciális funkcionalitásokat integráljanak a rendszerbe, mint például az egyes driverek, amik nincsenek specifikálva az AUTOSAR keretein belül. Konkrét példa egy motorvezérlő periféria. A réteg lehet mikrovezérlő- vagy ECU hardver függő.

Szolgáltatási réteg: ez az alapszoftver réteg legmagasabban elhelyezkedő alrétege. Feladata, hogy elérje az alatta lévő ECU absztrakciós réteg I/O jeleit és továbbítsa a felette elhelyezkedő RTE rétegnek vagy másik alapszoftver rétegbeli modulnak. Ezen kívül a réteg rendelkezik: operációs rendszer funkcionalitással, kommunikációs és hálózatmenedzselési funkciókkal, memória-szolgáltatásokkal (NVRAM menedzselés), diagnosztikai szolgáltatásokkal, ECU állapot- és módmenedzseléssel valamint logikai programműködés megfigyeléssel (watchdog menedzser). [2]

2.2 Az alapszoftver réteg szolgáltatásai, moduljainak típusai

Az alapszoftver réteg a benne található modulok funkciói szerint további csoportokra, szolgáltatásokra is bontható. Ezek a szolgáltatások:

Input/Output (I/O): szabványosított hozzáférést nyújt a szenzorokhoz és más ECU-n található perifériákhoz.

Memory: ez a réteg a rendszer külső/belső nem felejtő memóriáját kezeli.

Crypto: a Crypto réteg kezeli a kriptografikus algoritmusokat, valamint a külső/belső hardver gyorsítókat.

Communication: a jármű kommunikációs hálózatához nyújt szabványos hozzáférést. Ebbe beletartoznak az ECU fedélzeti és belső szoftveres rendszerei.

Off-board Communication: szabványos hozzáférést nyújt a járművek közötti kommunikációhoz, a vezeték nélküli hálózatokhoz és a nem fedélzeti kommunikációs hálózatokhoz. Ide tartoznak a diagnosztikai kommunikációval foglalatos egységek.

System: az operációs rendszerrel, időzítővel és hibamemóriával kapcsolatos modulokat tartalmazza ez a réteg. Ide tartoznak még az ECU specifikus szolgáltatások is.

Az alapszoftver moduljait viselkedésük és szerepük alapján típus osztályokba tudjuk sorolni:

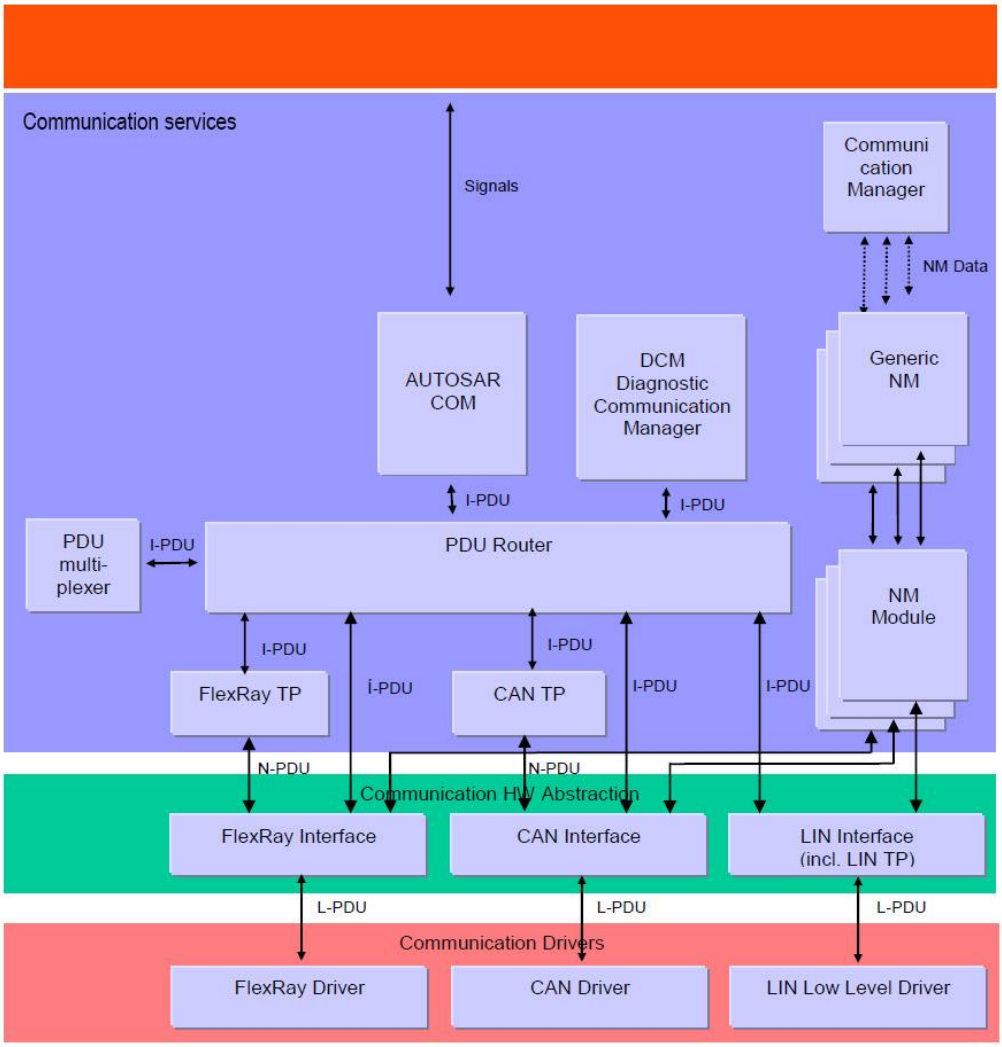
Driverek: fő feladatuk ezeknek a moduloknak, hogy hozzáférést biztosítsanak a szükséges hardver funkcióihoz, és hardverfüggetlen API-t biztosítsanak az interface moduloknak. Ebben a rétegben foglalnak még helyet a transciever driverek, amik a transciever hardvert vezérlik (Ez a hardver felelős a jelszintillesztésért a kommunikációs periféria (pl.: CAN) és mikrovezérlő között.), és irányítják ezáltal a busz állapotát. (normál üzemmód, alvó mód). [5] A driverek kezelik a belső és külső eszközöket. A belső eszközök a mikrovezérlőn belül helyezkednek el. Ilyen lehet például a belső analóg-digitális átalakító vagy a belső CAN szabályozó. Ezt a vezérlőt internal driver-nek hívjuk és a mikrokontroller absztrakciós rétegben helyezkedik el. A külső driverek az ECU absztrakciós rétegben találhatóak és a mikrokontroller egységén kívül az ECU hardverén helyezkednek el. Lehet ilyen a külső EEPROM vagy a külső flash memória. Ez a driver hozzáférést nyújt a külső egységekhez a mikrokontroller absztrakciós réteg driverei segítségével. A működésre ilyen példa az, amikor a driver egy SPI interfésszel rendelkező EEPROM-hoz fér hozzá annak SPI busz driverével.[2]

Interface: ezek a modulok képesek a strukturálisan alattuk helyezkedő modulok funkcionalitásait kihasználni . Egy generikus API-t nyújt, aminek a segítségével az adott eszköz hardveres realizációjától képesek leszünk elvonatkozni. Az interfész modulok nem változtatják meg az adatokat, és az ECU absztrakciós rétegben találhatóak. Példa a működésükre az, amikor az általános hálózat menedzsment interfész (NmIf) modul egy általános hozzáférést nyújt a különböző buszfüggő hálózat menedzsment modulok funkcióihoz (CAN, FlexRay).

Handler: a handler modulok kezelik az egyidejű, többszörös és aszinkron hozzáférést egy vagy több kliens által a driverekhez. Ezek a modulok foglalkoznak a pufferelléssel, sorbaállással, arbitrációval és multiplexeléssel.

Manager: a menedzser specifikus szolgáltatásokat nyújt több kliensnek egyszerre. Ezek a modulok ott szükségesek, ahol a kizárólagos handler funkcionalitás nem elég ahhoz, hogy több klienstől elvonatkozzon. A handler funkcionalitáson kívül egy manager képes az adatokat megvizsgálni, és azok tartalmát a funkcionalitásának megfelelően megváltoztatni. A manager modulok a szolgáltatás rétegben foglalnak helyet.

2.3 A kommunikációs stack



2.2. ábra AUTOSAR kommunikációs stack [6]

Ez a stack tartalmazza az AUTOSAR architektúra kommunikációs működését megvalósító moduljait. A stack felbontható a kommunikációs protokollok szerint CAN, LIN, FlexRay, J1939 és Ethernet stackekre, valamint az alapszoftver rétegbeli pozíciójuk szerint alrétegekre. Ezek a különböző protokolloknál nagyon hasonló felépítésűek:

Communication drivers

Itt találhatóak a protokoll által meghatározott, alacsony szintű driver modulok, amik a perifériát kezelik. Ebben a rétegben található meg az SPI Handler driver is, ami az SPI-t kezeli.

Communication hardware abstraction

Ez a réteg tartalmazza azokat az interface modulokat, amik az adott típusú periféria szerint egy felületet nyújtanak az adott drivernek. Ugyanitt vannak azok a driverek is, amik a külső perifériák irányításáért felelősek, például a transciever driverek.

Communication services

Ebben a rétegben történik meg a kommunikációs adatcsomagok (PDU-k) kezelése, és szállítása a modulok számára. Ennek a rétegnek a főbb moduljai:

Transport Layer: ezek a PDU Router és Interface modulok között helyezkednek el. A fő feladatuk, hogy a feladandó I-PDU-kat (például amelyek 8 (vagy 64 FD esetén) bytenál hosszabbak CAN protokoll esetén) szegmentálják, majd a fogadó oldalon a protokollnak megfelelően újraalkossák. Erre azért van szükség, mert a legtöbb esetben a két kommunikáló félnek nincs információja a köztük lévő csatorna típusáról és az egyszerre átvihető adatmennyiségről. A Transport Layerek ezen kívül irányítják az adatáramot és észlelik a szegmentációs hibákat is. [6] Adott perifériák kommunikációs stackjében több transport protokol is definiálva van, de csak egy van használatban. Példa erre a FlexRay stack, amiben a FlexRay ISO Transport réteg és a FlexRay AUTOSAR transport layer is megtalálható. Utóbbi feladata az hogy a buszt kompatibilissé tegye az AUTOSAR R3.x szabvánnyal. Más kommunikációs protokollnál, mint például LIN estében nincs külön transport layer modul. Ezt a funkcionalitást itt a LIN Interface valósítja meg.[2]

State Manager: az AUTOSAR BSW stack minden egyes kommunikációs busznak specifikál egy annak megfelelő állapot manager-t. Ezek a modulok implementálják az adatáramlás vezérlését amiknek megfelelően vált a kommunikációs hálózat az állapotai között. Ilyen kérések, amik állapotváltást eredményeznek a ComM modul felől érkeznek. Ha az adott modul Controller vagy Transceiver állapotában változás történik, az interface modul értesíti erről a state manager-t.[7]

PDURouter (PduR): a PDURouter feladata a stack rétegei közötti I-PDU-k irányítása. Ez a modul minden AUTOSAR ECU-ban telepítve van, és ő az AUTOSAR kommunikációs stack középpontja. A PDU Router modul két fő részből épül fel [8]:

- PDU Router routing paths: Egy kapcsolási táblát (routing table) valósít meg, aminek a segítségével eldönti a modul, hogy az üzeneteket melyik másik modulnak kell továbbítania.
- PDU Router engine: maga a kód, ami az útvonalválasztó folyamatokat valósítja meg.

ECU-k szerint a PDU Router három feladatot láthat el:

- PDU-k fogadása helyi moduloktól: a modul I-PDU-k fogadására képes az alacsonyabb rétegek moduljaitól, amiket egy, vagy több felsőbb réteg moduljaihoz juttat el.
- PDU-k küldése helyi moduloktól: I-PDU-k küldése egy vagy több alacsonyabb réteg moduljai számára felsőbb rétegbeli modulok kérésére.
- PDU Gateway: gateway (átjáró) szerepet kétféle képpen tölthet be. Interface vagy transport protocol modultól érkező I-PDU-kat fogad és továbbít ugyanannak, vagy más interfész / transport protocol modulnak, ezzel kommunikációs hálózatokat összekötve.

I-PDU Multiplexer (IpduM): az I-PDU multiplexelés azt jelenti, hogy kibővíthetjük az I-PDU-kat úgy, hogy azok multiplexálhatóak legyenek (eltérő a tartalmuk, de az azonosítójuk megegyezik a buszon). Léteznek olyan PDU-k amikhez több, mint egy SDU (Service Data Unit) tartozik, de azonos PCI-el (Protocol Control Information) rendelkeznek. A PDU multiplexelésnél egy kiválasztó mezőt használnak, ami része feldolgozandó SDU-nak. Küldő oldalon az I-PDU Multiplexer modul feladata a COM modultól érkező megfelelő I-PDUk kombinálása egy új, multiplexált üzenetté,

majd annak továbbítása a PDU Router-en keresztül. Fogadó oldalon a beérkező multiplexált I-PDUk értelmezése, és a COM modulnak a megfelelően szeparált I-PDU-k szállítása a feladata a kiválasztó mező tartalmának megfelelően. [9]

Communication (Com): az AUTOSAR kommunikációs modul az RTE és a PDU Router között helyezkedik el, és a köztük lévő kommunikáció lebonyolításáért felelős. Ez a modul helyezkedik a BSW rétegben a legfelül. Feladata hogy az RTE virtuális buszán keresztül, szoftverkomponensektől érkező üzeneteket, szignálokat I-PDU-k ba csomagolja, és ezeket továbbítsa a PDU Routernek. A feladata még ezen PDU Router irányából érkező I-PDUk kicsomagolása, és továbbítása az RTE-nek. További funkciói közé tartoznak még: kommunikációs transzmissziószabályozás (I-PDU csoportok indítása és leállítása), szűrő mechanizmusok, szignál alapú gateway funkció és dinamikus hosszúságú adattípusok támogatása [10]

Diagnostic Communication Manager (Dcm): a DCM modult a külső diagnosztikai eszközök veszik igénybe fejlesztés, gyártás és szervizelés során. Feladata a kommunikáció megvalósítása a szervízelő eszköz és az ECU között. [11]

A következő modulok vesznek részt az egyes buszok hálózatmenedzsmentjében:

Busz specifikus hálózatmenedzsment modulok: ezek a modulok felelnek az adott busz hálózatmenedzmenti állapotának a vezérléséért. Ilyen állapot lehet például az alvó mód és a normál mód. Ezen kívül, ha a buszon és a hozzá kapcsolódó *klaszteren* nem történik aktivitás, kommunikációs szempontból nincs az egyik node-ra sem szükség, ezek a modulok felelnek azért, hogy ezt az információt továbbítsák az NmIf modulnak.[12]

Network Management Interface (NmIf): ez a modul nyújt egységes interfészt a ComM modul számára a különböző busz specifikus hálózatmenedzsment (FrNm, CanNm) moduloktól. Ezen kívül az azonos ECU-hoz kapcsolódó hálózatok szinkron alvó üzemmódba váltását valósítja meg, amit NM Coordinator funkcionalitásnak hívnak. A modul vagy az alap funkciókat valósítja meg (adaptációs réteg a busz specifikus hálózat menedzsment modulok és a ComM között) vagy mind alap és NM Coordinator funkciót. [13]

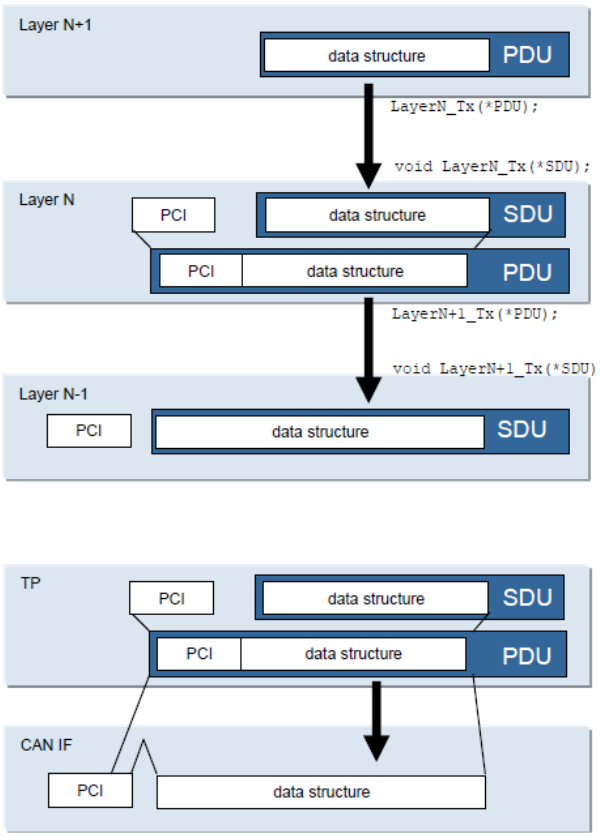
Communication Manager (ComM): a kommunikációs menedzsment modul feladata, hogy a kommunikációs buszok elérési kéréseit fogadja, majd irányítsa a kommunikációhoz szükséges erőforrások elosztását. Az RTE felé egységes interfészt nyújt, ezzel a szoftverkomponenseket egyszerűbben kiszolgálva. A modul egyszerre több kommunikációs buszt is kezelhet egy ECU-n belül úgy, hogy minden egyes csatornának egy csatorna állapotgépet hoz létre. Olyan szolgáltatást is nyújthat a ComM, amivel kérésre egy ECU-t, ami már csak egyedül tartja ébren az egyik csatornáját „nincs kommunikáció” módba váltja. [14]

Default Error Tracer (Det): az AUTOSAR szabvány 4.2-es verziója előtt ezt a modult Development Error Tracer-nek nevezték. A basic software moduljai a default error tracer-nek jelentik a fejlesztés és futás idejű hibáikat. Ezeket a hibaüzeneteket a Det fogadja és feldolgozza vagy hibatárolóba menti. A modulok szabványai leírják milyen hibákat kell észlelni, és azokat milyen hibakóddal kell továbbítani a DET-nek.

2.4 A Protocol Data Unit

Az AUTOSAR rétegek közti kommunikációban, az adatokat PDU-nak (Protocol Data Unit, Protokol Adat Egység) nevezzük. A szolgáltatási rétegből RTE-n keresztül érkező adatot, *szignált* a COM modul kapja meg, és épít belőle PDU-t. A hasznos információt, amit a PDU szállít, az adott rétegbeli protokoll jellemzi . A PDU-k a BSW rétegeken lefelé haladva az adott protokollra jellemző, kiegészítő információkkal egészülnek ki, amit a PCI (Protocol Control Information) mező tartalmaz. A PCI mező figyelembe vételével történik az adatok feldolgozása, az átvételt követően, és a továbbítása is. A küldő rétegtől kapott, később tövábbítandó adatokat a Service Data Unit (SDU) tartalmazza.

A kövekező ábrán látható hogyan történik a PDU-k rétegek közti átvitele. Az N+1-ik réteg PDU-ja SDU ként jelenik meg az N-ik rétegben. Ez a réteg a kapott SDU alapján kiegészíti a saját PDU-ját egy PCI mezővel, majd továbbítja azt az N-1-ik rétegnek. A folyamat folytatódik egészen driver rétegig ami után a PDU-t már közvetlenül a buszon keresztül küldik tovább a fogadó ECU driver rétegének.



2.3. ábra PDU felépítése [2]

Fogadás folyamata során a rétegeken lentről felfelé haladunk. A réteg a kapott PDU-ból kicsomagolja az SDU-t a PCI mező segítségével, és továbbítja az a felette lévő rétegnek. Ez így folyatatódik egészen a legfelsőbb rétegig, ahol a PDU-ból az RTE réteg által feldolgozandó szignálok kerülnek kibontásra, ami végül eljuttatja a szignálokban tárolt adatokat a megfelelő szoftverkomponensekhez. [16]

Egyes modulok közti kommunikáció során a PDU-k különböző előtagokat kapnak annak megfelelően, hogy melyik rétegben foglalnak helyet. Ilyenek lehetnek I-PDU (Interaction), N-PDU (Network Layer) és L-PDU (Data Link Layer). [2]

3 Az AUTOSAR hálózatmenedzsment

A mai korszerű járművekben közel 100 elektronikus vezérlőegység foglal helyet, amik a gépjárművek biztonságtechnikai és kényelmi funkciói megvalósításában vesznek részt. A vezérlőegységek különböző kommunikációs protokoll szerint működő buszrendszereken kommunikálnak egymással.

Az elektromos vezérlőegységeknél működési állapotokat definiálhatunk azok viselkedésének és energia felhasználásának megfelelően. Gyújtás állapotban két mód van: egyik az, amikor áram alatt vannak a vezérlő rendszerek, de még nem kaptak engedélyt a működésre. Például az ablaktörlőt nem kapcsoltuk be, de a gépjármű gyújtás alatt van. Ekkor az áramfogyasztásuk elhanyagolható, néhány mikro amper. Ezt az állapotot *alvó* állapotnak hívjuk. Másik mód, amikor ezek engedélyt kapnak, aktívan üzembe kerülnek és a működésüknek megfelelően fogyasztanak energiát. Ez a *normális* üzemmód.

A hálózatmenedzslés szintjei szerint három módszert különböztethetünk meg. Az első, egyeben a legegyszerűbb megoldás, a fizikai tápelvonás. Itt a gyújtás állapota szerint egy relé segítségével lehet szabályozni az elektromos vezérlők ki- és bekapcsolását. Második szint, amikor minden eszköz megkapja a tápot, de a be- és kikapcsolásról minden elektromos vezérlőeszköz maga dönt. A harmadik megoldás a klasszikus hálózatmenedzsment példája, amikor a gyújtásjel vezetéket nem külön az összes ECU-ba, hanem egy közösen vezérelt klaszterba kötjük, amiben a vezérlőegységeket hálózatmenedzslési szempontból közösen vezérlünk. Ennek a megoldásnak az egyik változata a részhálózat kezelés funkciója, amire későbbi fejezetekben térek ki.

Az elektromos vezérlőegységek hálózatmenedzselési szempontokból Network Management klaszterekbe rendezhetők. Ezek az elrendezések az egész rendszer szintjén adottak. Egy csomóponthoz akár több különböző protokoll szerint működő busz is tartozhat, és ezekből egyszerre több is része lehet egy hálózatmenedzsment klaszternek. A hálózatmenedzsment funkciók mindig egy azonos klaszterbe tartozó buszokra vonatkoznak. A klaszterek a gateway (átjáró) elektromos vezérlőegységek esetén jelentősek, mivel azok úgy lettek konfigurálva, hogy több hálózatot csatolnak össze egymással, így üzenetet továbbíthatnak a többi hálózatnak is. A klaszterek egy

topológiát alkotnak magasabb szintről az alacsonyabbakig. Az átjáró ECU-k fontos szerepet töltenek be a koordinált kikapcsolás esetén, mert a magasabb rétegből érkező lekapcsolási/felébresztési utasítást ezeken az ECU-kon keresztül kapják meg az alhálózatok. Klaszterek esetén a buszok csak akkor kapcsolhatók le, amikor a buszra csatlakozó összes vezérlőegység készen áll az alvó állapotra. Amíg egyetlen egy node igényt tart a buszra, addig a klaszter összes többi egysége ébren marad. Ezzel kerülendő el az az eset, amikor az egyik vezérlőegység már nem tart igényt a buszra, viszont egy másik még igen. [12][13][15]

3.1 A hálózatmenedzsment üzenetek

A hálózatmenedzsment folyamata a rendszerben résztvevő vezérlőegységek által adott időközönként *broadcast* jellegűen kiküldött network management üzeneteken, NM-PDU-kon alapul. Eseményvezérelt hálózatokon, mint például a CAN, ezek az üzenetek előre megadott időközönként kerülnek ki a kommunikációs buszra. Idővezérelt kommunikációs protokoll esetén, mint a FlexRay, a kiküldendő NM Üzenetek adott periódusonként kerülnek kiküldésre az idővezérlésnek megfelelően. Ezekben az üzenetekben adja át a vezérlőegység a kommunikációs csatornán keresztül a többi ECU-nak azt az információt, hogy szüksége van-e még a buszra. Ha adott ideig az ilyen üzenetek szerint nincs szükség a buszra, akkor a hálózatmenedzselési alhálózat elemei kikapcsolnak. Felépítésükben ezek az hálózatmenedzsment PDU-k igen hasonlóak. Általában 8 byte-ból állnak, amiből kettőt a Control Bit Vector (CBV) és a Source Node Identifier (SNI) foglal el. A CBV tartalmazza azt az információt, ami a hálózatot szabályozza, és azt, ami megadja hogy a küldő készen áll-e hogy alvó módra. Az SNI tartalmazza a küldő fél információit és azonosítóját. Ezután a két byte után felhasználói adat byte-ok jöhetnek, amik többek között a részleges hálózatkezelés irányítására használhatóak. [2]

A szakdolgozatomban a CAN és a FlexRay hálózatmenedzsment moduljaival foglalkozom, ezért bemutatom a két protokoll NM-PDU-inak a felépítését.

3.1.1 CAN NM-PDU

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 7	User data 5							
Byte 6	User data 4							
Byte 5	User data 3							
Byte 4	User data 2							
Byte 3	User data 1							
Byte 2	User data 0							
Byte 1	Control Bit Vector							
Byte 0	Source Node Identifier							

3.1. ábra Általános CanNm PDU felépítése [12]

Ebben a konfigurációban a PDU hossza 8 byte, és a vezérlő byteokon kívül hat darab felhasználói adatbyte van beállítva. A CAN protokoll szerint a hálózaton időközönként kerülnek ki ezek hálózatmenedzsment üzenetek. Ha egy konfigurált ideig nem érkeznek ilyen üzenetek, az azt jelenti hogy a kapcsolódó ECU nem igényli tovább a hálózatot, és készen áll az alvó üzemmódba váltáshoz. A vezérlő byteok pozícióit konfiguráció során beállíthatjuk (Állhatnak a 0. és az 1. pozíciókban.). A Control Bit Vector beállítása nem kötelező, ha nincs ilyen byte a PDU-ban az azt jelenti hogy egyel több hely áll rendelkezésre a felhasználói adat számára. [12]

A Control Bit Vector felépítése a következő:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CBV	Reserved	Partial Network Information Bit	Reserved	Active Wakeup Bit	NM Coordinator Sleep Ready Bit	Reserved R3.2 NM Coordinator ID (High Bit)	Reserved R3.2 NM Coordinator ID (Low Bit)	Repeat Message Request

3.2. ábra CanNm CBV felépítése [12]

Az 1, 2, 5, 7 bitek foglalva vannak későbbi kiegészítések számára.

0. bit: Ismételt üzenet kérés (Repeat Message Request)

0: Az ismételt üzenet kérés állapot nincs igényleve

1: Az ismételt üzenet kérés állapotot igényelt

3. bit: NM Coordinator alvásra kész bit (NM Coordinator Sleep Ready Bit)

0: A fő koordinátor nem kéri a szinkronizált leállítás folyamatának az indítását.

1: A fő koordinátor kéri a szinkronizált leállítás folyamatának az indítását.

4. Aktív ébresztés bit (Active Wakeup Bit)

0: A node nem ébresztette fel a hálózatot (passzív felébresztés)

1: A node felébresztette a hálózatot (aktív felébresztés)

6. Részleges hálózat információ bit (Partial Network Information (PNI))

0: Az NM PDU nem tartalmaz a részhálózat kérést

1: Az NM PDU tartalmaz részhálózat kérést.

3.1.2 FlexRay NM-PDU

A FlexRay protokoll időosztásos elven alapul, ezért az NM-PDU-k periódusonként jelennek meg a hálózaton. A FlexRay hálózatmenedzsment képes arra, hogy az NM-Vote és NM-Data részeket egymástól függetlenül küldje el. Ennek a funkciónak a megvalósítására léteznek külön PDU formátumok a kettő továbbítására. Mivel az NM-Vote és NM-Data szállítása az üzenet statikus szegmensében is megtörténhet, az NM-Data PDU rendelkezik egy opcionális *szavazó bit*tel. A szavazóbitre azért van szükség, mert a CAN protokollal szemben itt a hálózatmenedzsment PDU-k periodikusan jelennek meg és nem tudják úgy jelezni a készenlétüket a leállásra, hogy nem küldenek NM üzenetet. Ehelyett a szavazóbit állapotát figyeli a rendszer. Ha az negatívan szavazott egy előre konfigurált ideg, az jelenti a lekapcsolásra kész állapotot. [15]

Az NM-Data PDU:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 7	User data 5							
Byte 6	User data 4							
Byte 5	User data 3							
Byte 4	User data 2							
Byte 3	User data 1							
Byte 2	User data 0							
Byte 1	Source Node Identifier							
Byte 0	Set to “0”	Control Bit Vector						

3.3. ábra FrNm PDU felépítése [15]

Ez a PDU formátum hasonlít a CAN hálózatmenedzsment PDU-ra. Tartalmaz Source Node Identifiert ami a küldő azonosítására szolgál, Control Bit Vectort ami a hálózat szabályozásához tartalmaz információt, valamint a felhasználói adatbyteokat. A CBV felépítése:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte	Set to "0"	Partial Network Information Bit	Res	Active Wakeup Bit	NM Coordinator Sleep Ready	Res	Res	RptMsg Request

3.4. ábra FrNm CBV felépítése [15]

Az 1, 2, és 5-ös bitek foglalva vannak későbbi kiegészítések számára.

0. bit: Ismételt üzenet kérés:

0: Ismételt üzenet állapotot nincs igényelve

1: Ismételt üzenet állapot igényelve van

3. bit: NM Coordinator alvásra kész bit (NM Coordinator Sleep Ready Bit)

0: A fő koordinátor nem kéri a szinkronizált leállítás folyamatának az indítását.

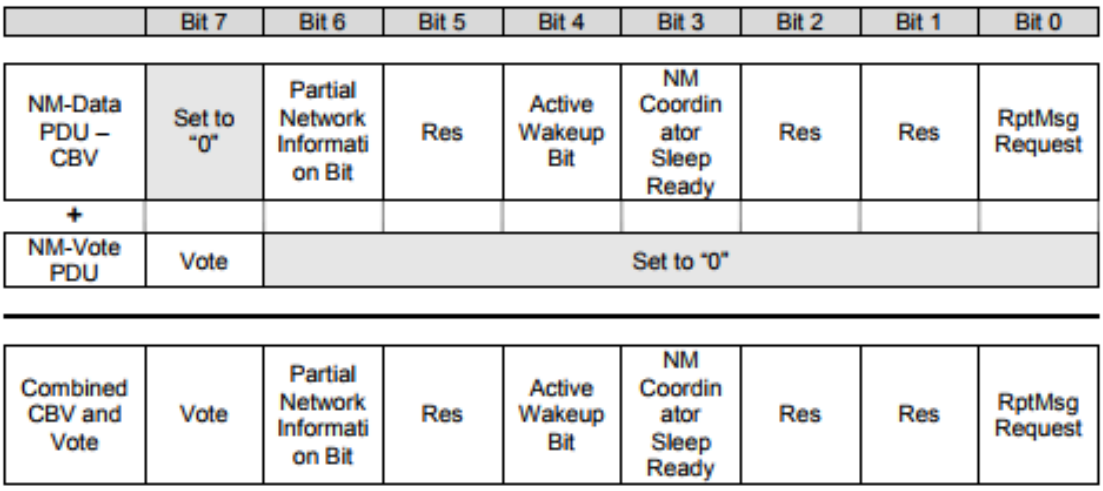
1: A fő koordinátor kéri a szinkronizált leállítás folyamatának az indítását.

6. Részleges hálózat információ bit (Partial Network Information (PNI))

0: Az NM PDU nem tartalmaz a részhálózat kérést

1: Az NM PDU tartalmaz részhálózat kérést.

A 7. bit 0-ra van állítva, mivel ha az NM-Vote és NM-Data egy PDU-ban küldődik ki, akkor a kettő PDU kombinálva van egy VAGY operátorral, és így a szavazóbit nincs befolyásolva:



3.5. ábra FrNm Data és Vote PDUk kombinálása [15]

3.2 A kommunikációs buszok állapotai.

A szakdolgozatomban a CAN és FlexRay hálózatmenedzsment modulokkal foglalkozom, ezért ennek a két modulnak mutatom be a lehetséges hálózatmenedzsment állapotait. [12][15]

3.2.1 A CAN buszok állapotai

A CAN protokoll szerint a buszoknak három fő állapota van:

- Network Mode (Hálózati mód)
- Prepare Bus-Sleep Mode (Felkészülés az alvó busz módra)
- Bus-Sleep Mode (Alvó busz mód)

Network Mode

A Network Mode három alállapotot tartalmaz:

- Repeat Message State (Üzenet ismétlő állapot)
- Normal Operation Mode (Normális működés állapot)
- Ready Sleep State (Készenlét az alvó állapotra)

Amikor a modul Bus-Sleep Mode, vagy Prepare Bus Sleep Mode -ból Network Mode-ba kerül, az mindig az üzenet ismétlő alállapotot veszi fel.

Repeat Message State

Ebben az állapotban ha a node nem passzív módban üzemel (NM üzenetek küldésére is képes), akkor biztosítja a hálózat többi tagja számára hogy azok észleljék annak alvó, vagy felkészülés az alvó busz módból hálózati módba kerülését. Ez úgy oldódik meg, hogy a Normal Operation Mode-hoz hasonlóan NM-üzeneteket küld a hálózatra ciklikusan egy rövidebb periódusidővel. Ez az állapot a node felébresztési folyamatának a kezdőállapota, és annak indításáért felelős.

Normal Operation Mode

A normális működés állapot biztosítja, hogy a hálózat bármely tagja ébren tudja tartani a saját hálózatmenedzsment klaszterét, ameddig a hálózat igényt tart rá. Ez úgy történik, hogy a node, amit igénybe vesz a hálózat, időközönként olyan hálózatmenedzsment üzeneteket küld a hálózatnak amiben értesíti a többi tagot az ébren maradási kérésről.

Ready Sleep State

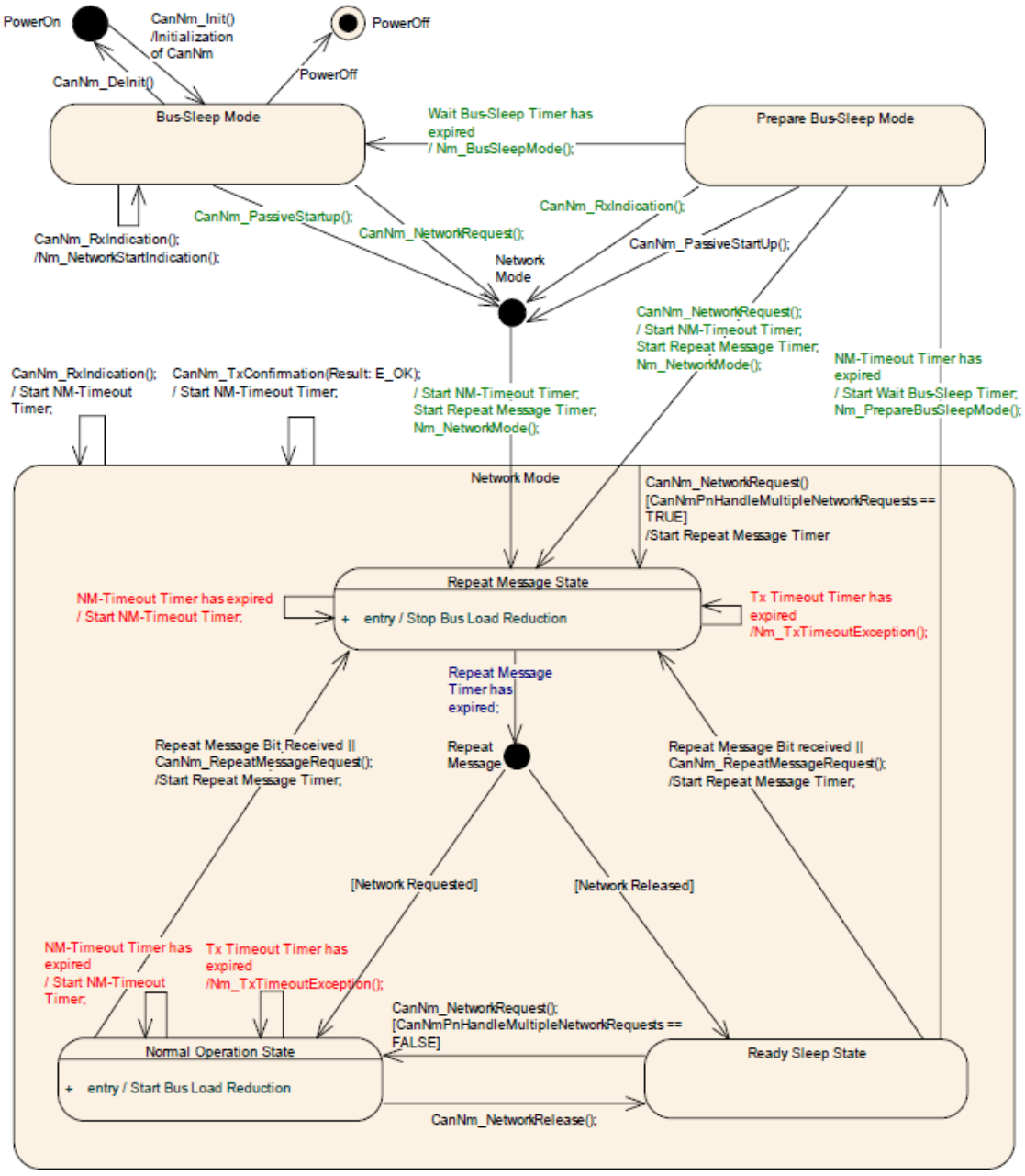
A célja ennek az állapotnak az, hogy minden node-nak legyen ideje a hálózati kommunikációk befejezésére, mielőtt alvó módba kerülnének. Ebben az állapotban nem kerülnek további NM-üzenetek küldésre.

Prepare Bus-Sleep Mode

Ebben a módban meg kell győződni, hogy a klaszter összes nodejának van elég ideje a kommunikációs aktivitásuk befejezésére, mielőtt mindannyian áttérnének alvó módba. Ilyenkor nem keletkezhet új NM-üzenet, tehát a küldött üzenetek mennyisége lecsökken a buszon, és csak azok kerülnek kiküldésre amik a Tx-pufferekben el vannak tárolva. Ennek a folyamatnak a végén megszűnik az aktivitás a buszon és az készen áll az alvó módba váltáshoz.

Bus-Sleep Mode

Akkor kerül egy node alvó módba, amikor már nem küld, és nem fogad üzeneteket. A fő célja ennek a módnak az energiafogyasztás minimalizálása. A alvó módba kacsolás folyamata a következő: a kommunikációs kontroller és transciever alvó módba kapcsolnak, a kommunikációs protokoll specifikus felébresztési mechanizmusok bekapcsolódnak, majd végül az energiafogyasztás az alvó módnak megfelelő szintet ér el.



3.6. ábra CAN hálózatmenedzsment állapottáblája[12]

3.2.2 A FlexRay buszok állapotai

A Flexray protokoll szerint működő buszoknak három fő állapota van:

- Bus-Sleep Mode (Alvó busz mód)
- Synchronize Mode (Szinkronizációs mód)
- Network Mode (Hálózati mód)

Bus-Sleep Mode

Inicializáció után a FlexRay hálózatmenedzsment állapotgépében a kezdeti állapot az alvó busz mód. A hálózat mindaddig ebben az állapotban marad, amíg nem érkezik kommunikációs kérés. Ez történhet passzív mód kéréssel vagy egy hálózati kéréssel. Alvó busz módban a kommunikációs kontroller alvó állapotot vesz fel, ahol az energiafogyasztás felveszi a lehető legalacsonyabb szintet.

Synchronize Mode

Szinkronizációs módban a FlexRay hálózatmenedzsment állapotgép megvárja, hogy az szinkronizálódjon a FlexRay klaszter ismétlődési ciklusához (Repetition Cycle), amiben a hálózatmenedzsment üzenetek küldődnek és érkeznek. Ez azért szükséges, mert a FlexRay protokoll időosztásos tulajdonsága miatt a hálózatmenedzsment klaszteren belül minden node-nak összehangoltan kell működnie.

Network Mode

A CAN protokollhoz hasonlóan, a FlexRay hálózati módjának is három alállapota van.

- Repeat Message State (Üzenet ismétlő állapot)
- Normal Operation Mode (Normális működés állapot)
- Ready Sleep State (Készenlét az alvó állapotra)

Repeat Message State

Az üzenet ismétlő állapotban azok a node-ok, amik alvó állapotból hálózati módba kerülnek, és nincsenek passzívan konfigurálva láthatóvá válnak a többi node számára. Ez úgy történik meg, hogy megkezdődik az NM-adatok küldése és a node megkezdi a szavazást a szavazó bit segítségével. Az üzenet ismétlő állapotot akkor hagyja el a modul, amikor egy kijelölt számláló letelik, tehát az összes többi node észlelete a hálózat módba kerülését.

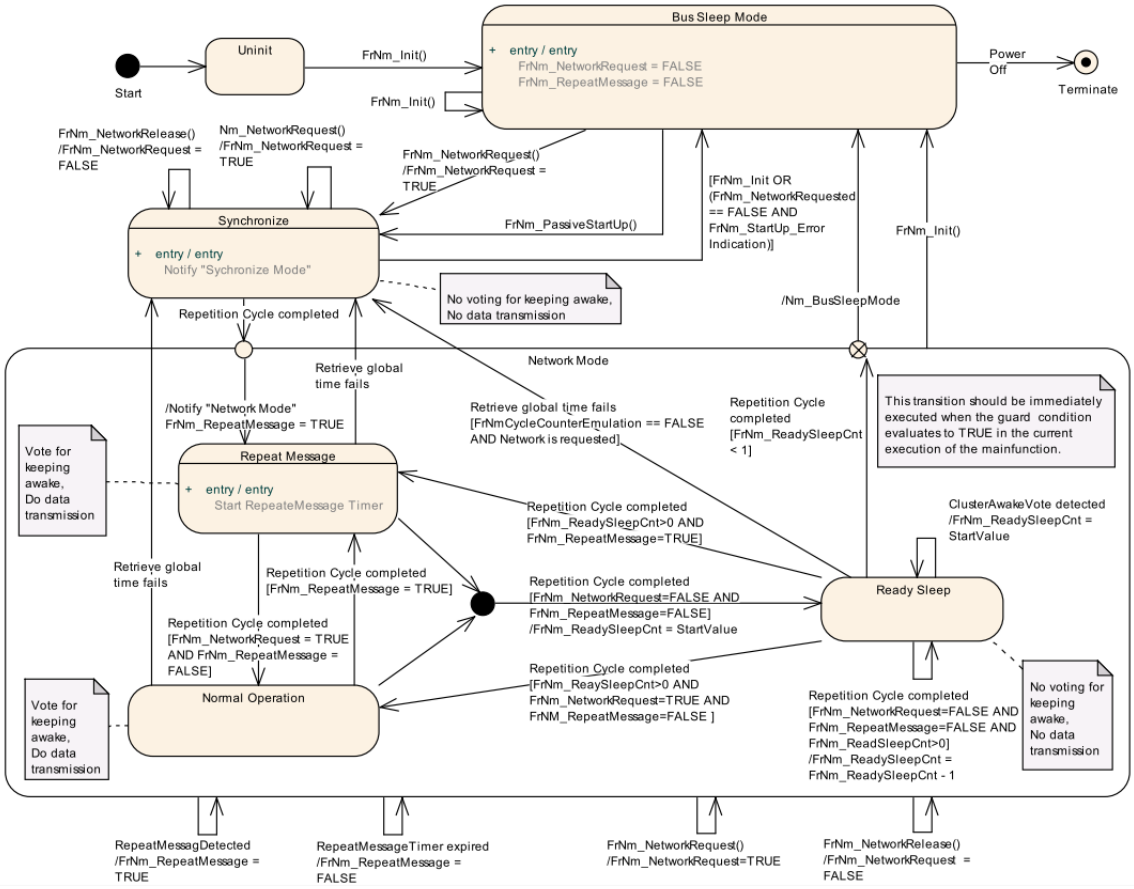
Normal Operation Mode

A célja ennek az állapotnak az, hogy mindaddig ébren tartsa a hálózatmenedzsment klasztert, amíg abból egy node a hálózati kommunikációra igényt tart. Ez úgy történik, hogy az a node, ami igényt tart a hálózatra, a periodikus NM-üzeneteiben a szavazóbitet pozitívra (1-re) állítja. A passzívan konfigurált node-ok,

amik nem képesek NM-üzenet küldésre, csak fogadásra, nem kerülhetnek ebbe az állapotba.

Ready Sleep State

Ebben az állapotban az NM klaszter vár az alvó állapotba kerüléssel addig, amíg bármely klaszterhez tartozó node igényli a hálózati kommunikációt. Amikor a modul Ready Sleep állapotba kerül, már nem küld NM üzenetet, és a kiküldött keretek szavazóbiteit negatívra (0-ra) állítja. Ezzel jelzi, hogy felkészült az alvó állapotba kerüléshez, már csak arra vár hogy a klaszter többi tagja is ugyanezt tegye.



3.7. ábra FlexRay hálózatmenedzsment állapottáblája [15]

4 Kommunikáció az AUTOSAR szabvány szerint

Ebben a fejezetben a hálózatmenedzsment modulok együttműködését mutatom be egy kommunikációs forgatókönyv segítségével, amiben a vezérlőegység elindít egy hálózati kommunikációt, a hálózat felébred és megindul rajta a hálózatmenedzselés folyamata.

Kommunikáció indítása felhasználói kérésre vagy egy BSW modul által történik. Üzenet küldéséről a ComM modul értesítést kap az RTE rétegen keresztül, és továbbítja a kommunikációs kérést az NmIf modulnak. Ez a modul megkapja megfelelő csatorna azonosítóját, a kiadott kérést és továbbítja a busz specifikus hálózatmenedzsment modulnak, ami a kérés hatására átváltja a CanNm állapotát hálózati módba. Ezen kívül a ComM a CanSM modul segítségével a buszt is a megfelelő állapotba állítja.

A felhasználó által indított kommunikáció egy virtuális függvénybuszon (Virtual Function Bus - VFB) keresztül zajlik, ami egy olyan absztrakt buszrendszer, melynek feladata hogy az ECU-k fizikai határait elrejtse a szoftverkomponensek elől, és így egy olyan rendszert alkosson, amiben kommunikáció során mindegy hogy melyik komponens melyik vezérlőegységen fut. [16] Az üzenet továbbításához a szoftverkomponensből érkező, az RTE-n keresztül haladó szignál a COM modulba jut. Itt a modul ezt I-PDU-vá alakítja, és kiegészíti vezérlőinformációkkal, amiben benne van a küldő és fogadó fél azonosítója. Az átalakított PDU ezután a PDURouter-be kerül, ami a beérkezett PDU azonosítója alapján egy előre konfigurált kapcsolási tábla (routing paths) segítségével továbbítja a busz specifikus Transport rétegnek, vagy akár egyből a busz specifikus Interface modulnak. A Transport rétegnek akkor kell küldeni a PDU-t az Interface modul előtt, amikor a PDU-t szegmentálni kell azért, hogy a fogadó félnek megfelelő legyen a hossza. [8][10][14]

4.1 Hálózatmenedzsment folyamata kommunikáció során

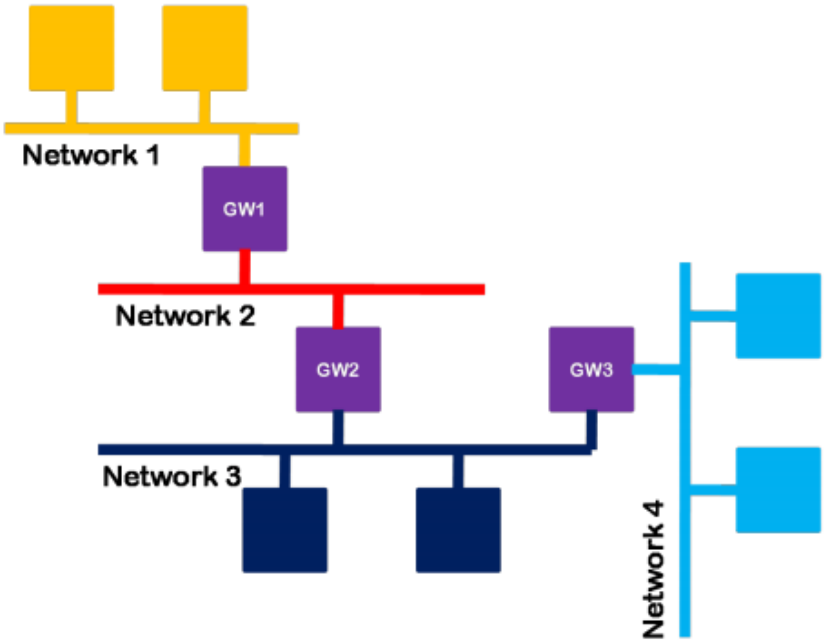
A hálózatmenedzsment folyamatát egy eseményvezérelt kommunikációs protokoll szerint működő hálózaton fogom bemutatni (CAN hálózat). Ez a hálózat úgy épül fel, hogy egy aktívan vezérelt node tartozik hozzá, és nincs további alhálózata. Egy node lehet aktívan vagy passzívan vezérelt.

Passzívan vezérelt node-ot csak egy aktívan vezérelt azonos hálózatba tartozó node kezelhet. Amikor egy kikapcsolt passzív node-ot szeretnénk felébreszteni, akkor a hálózat aktív node-jának kell elkezdenie periodikusan NM-üzeneteket küldeni. Ennek az üzenetnek a beérkezését a passzív node transciever drivere észleli és jelzést ad ki róla. Ezt a jelet a state manager kapja meg és jelenti a ComM modulnak. Amennyiben bekapcsol a csatorna, akkor az NmIf segítségével a node CanNm modullal passzív indítást hajt végre. Passzív indítás során a CanNm állapotgépe átvált Repeat Message módba, ahol nem kerül sor üzenet küldésre a passzív konfiguráció miatt. Adott idő után ez az állapotgép átvált a Ready Sleep State módba, mivel a node nem küldhet NM-üzenetet, tehát nincs szüksége a csatornára és átléphet a Prepare Bus-Sleep módba. Itt is eltölt egy előre megadott hosszúságú időt, hogy megvárja, amíg a befejeződik az üzenetküldés. Amikor ebben az állapotban érkezik egy NM-üzenet, akkor az állapotgép átvált Network Mode, ahol az állapotgép ismét az Repeat Message Mode-ba kerül. Amennyiben ez nem történik meg, az azt jelenti, hogy az Aktívan vezérelt node nem küld több üzenetet a hálózatnak és az átválthat Sleep Mode-ba. Az alvó üzemmódba lépésről a hálózatmenedzsment interfészen keresztül ad visszajelzést a node CanNm modulja a ComM modulnak, ami a State Managernek utasítást ad, hogy a Transcievert alvó állapotba állítsa.

Amikor aktív node-ot kell kezelni, az hasonlóan történik mint a passzív esetben. A különbség az, hogy a felébresztendő aktív node képes a klaszterének a felébresztésére, míg a passzív nem mivel az nem küldhet NM-üzeneteket. Egy szoftverkomponens által indított kommunikációs kérést az RTE rétegnek jelenti, ami a ComM modulnak továbbítja azt. A ComM a node állapot menedzserén keresztül kezdeményezheti annak a bekapcsolását. Mindeközben a hálózatmenedzsment interfész-en keresztül a busz specifikus hálózatmenedzsment modul is kérést kap a bekapcsoláshoz, és átállítja az állapotgépét Network Mode-ba, ezen belül is Repeat Message State-be. Ekkor elkezdődik a hálózatmenedzsment PDU-k küldése a hálózatra az aktív node által, jelezve, hogy a hálózat ébrentartására szükség van. A hálózatmenedzsment modul az interfész modulon keresztül küldi ki ezeket a PDU-kat a PDURouter elkerülésével. Üzenet ismétlő módban miután az összes szükséges üzenet ki lett küldve, a csatorna először Ready Sleep State-be majd Prepare Bus-Sleep módba és végül alvó állapotba kerül.

4.2 Hálózatmenedzsment egy összetett hálózaton.

Az összetett hálózatok felépítése hierarchikus, ami ebben az esetben azt jelenti, hogy egy fő csatornához az ahhoz tartozó átjáró node-okon keresztül beágyazott alhálózatok kapcsolódnak. A fő csatorna azzal a tulajdonsággal rendelkezik, hogy egy olyan koordinációs klaszterhez tartozik, amiben az összes csatorna aktívan koordinált. Ez a csatorna indíthatja el az irányított leállítás folyamatát az összes többi beágyazott csatornán, ami passzívan lett koordinálva a konfiguráció során. A beágyazott koordinátorok a passzívan koordinált csatornájukon keresztül kapják meg az NM üzeneteket, és továbbítják a következő koorindátornak az annak aktívan koordinált csatornáján keresztül.



4.1**. ábra Összetett hálózat [13]**

A fenti elrendezésben GW1 a Network 1-et és Network 2-t aktívan koordinált csatornaként konfigurálta. GW2 a Network 2-t passzívan és Network 3-at aktívan. Végül GW3 a Network 3-at passzívan konfigurálta fel, de Network 4-et aktívan. Ha például az első csatornáról szeretne üzenetet küldeni az egyik node a másiknak a harmadik csatornán, akkor az az előbb leírt ébresztési folyamatok sorozataként történik meg. Az első csatorna aktív koordinátora felébreszti az első csatornát, amivel együtt a második csatorna is felébred a passzívan konfigurált GW1-et beleértve. Így a GW1 felébreszti a második csatornát és vele együtt a harmadikat is a GW3 node segítségével. Ezek után megtörténhet az adatcsere. [13]

5 Hálózatmenedzsment modulok módosítása

A szakdolgozatom az AUTOSAR három hálózatmenedzseléssel foglalkozó moduljának módosítására épül, hogy azok megfeleljenek a 4.3-as szabványverzió követelményeinek. A feladatomhoz rendelkezésemre állnak az AUTOSAR 4.0-s és 4.3-as szabványdokumentumai és ezeknek a moduloknak a 4.0-s megvalósítása.

A modulok módosítása hasonló terv szerint történt mindhárom esetben. Első lépésben el kellett olvasnom a modul régi és új szabványdokumentumait, hogy megértsem annak működését, valamint megtaláljam milyen változások történtek és azok mekkora mértékűek. A modoulok működése mögötti elmélet mélyebb megértésére szükséges volt a velük kölcsönösen együttműködő modulok működésének a tanulmányozása is. Ezek után ki kell gyűjteni a követelmények közti változásokat, módosításokat, hogy átlátható legyen mely funkciók módosultak, és így előre meg lehet tervezni azok megvalósítását. Ugyanitt klasszifikálni kell az új és módosított követelményket tesztelhetőség és implementálhatóság szerint. Ha egy követelmény implementálható, akkor azt az implementálhatság definíciójának megfelelve mindenféleképpen kód szinten kell megvalósítani. Amikor egy követlemény tesztelhető, tesztelési eljárásokkal ellenőrizni lehet a működését.

Miután átfogóbb képet nyertem a modul működéséről és a módosult követleményeket is klasszifikáltam, meg kellett ismernem a 4.0-s implementáció forráskódját amit a következő lépésben módosítani kell. Részletesen meg kell figyelni hogy miként lett a kód elkészítve, különösen figyelve a statikus adatszerkezetére és segédfüggvényeire mivel azokat kell a legtöbbször megváltoztatni, vagy esetleg az új funkciók implementáláshoz felhasználni. Mindezek után lehet elkezdeni a módosítás folyamatát folyamatosan ügyelve hogy a szabvány modulra vonatkozó követelményeit szigorúan kövessem.

Ebben a fejezetben bemutatom a három hálózatmenedzsment modul funkcióit részletesebben, majd kitérek a szabványban történt módosításokra és azok megoldásának az implementálására.

5.1 Network Management Interface

Következőnek a 4.3-as Network Management Interface modul funkcionalitását részletezem, majd az abban történt változásokat és azok implementálását ismertetem. [12]

5.1.1 Funkcionalitás

A hálózatmenedzsment interfész modulnak két fő funkcionalitása van:

- Alap funkcionalitás (Base Functionality)
- NM Coordinator funkcionalitás

5.1.1.1 Alap funkcionalitás.

A modul alapfunkciói minden ECU-n megtalálhatóak. Ez az interfész funkcionalitás, ami adaptációs réteget nyújt a busz specifikus hálózatmenedzsment modulok (CAN NM, FlexRay NM, Lin NM, UDP NM és J1939 NM) és a ComM modul között. Ezt úgy valósítja meg, hogy a ComM modul felől érkező általános függvényhívásokat továbbítja a függvényben megadott azonosító alapján a megfelelő NM modulnak. Az NM modulok felől érkező *callback* függvényeket is hasonlóan kezeli ez a funkcionalitás, általános típusúvá alakítja őket, és továbbküldi a ComM modulnak.

5.1.1.2 NM Coordinator funkcionalitás

Ennek a funkciónak a feladata az, hogy egy algoritmus segítségével irányítsa a modul a hálózatmenedzselő folyamatának a leállítását egy, vagy több a vezérlőegységhez tartozó független buszon. A leállítási folyamathoz nem szükséges a koordinációs klaszter összes elemének ébren lennie, viszont a leállításhoz a modul csak az éber hálózatokat koordinálhatja. Amik már alvó módban vannak azokat csak figyelemmel tartja.

Az algoritmus képes arra hogy olyan topológiát kezeljen, ahol egy NM koordinátor kezel egyszerre több buszt.

5.1.1.3 Koordinált buszok ébren tartása

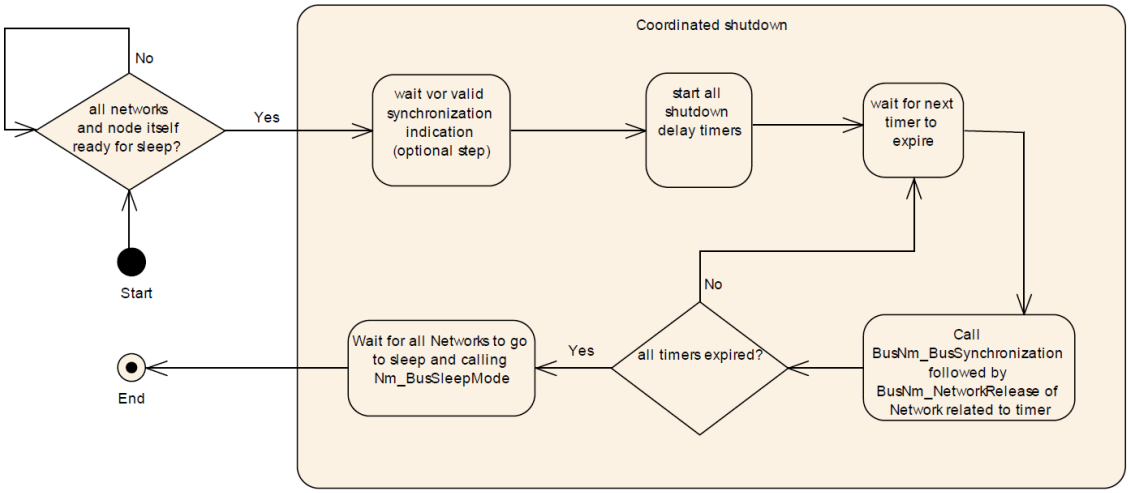
Az NM Coordinátor fő feladata az, hogy ameddig legalább a koordinációs klaszterben egy node ébren tartja a csatornát, addig az egész klasztert ébren tartsa. Ez úgy történik, hogy amíg kommunikál a node a hálózaton, addig a busz specifikus

hálózatmenedzsment modul adott időközönként NM üzenetekkel jelzi a többi node-nak hogy igényt tart a buszra és nincs kész az alvó állapotra. Amennyiben készen áll a node az alvó módra, azt egy *callback* függvényhívással jelzi az NM interfésznek. Ha az NM klaszter *NmChannelSleepMaster* konfigurációs paramétere IGAZ-ra van állítva, nem szükséges jeleznie a készenlétet az alvó állapotra, mielőtt elkezdődne a hálózat leállítása. Ez a funkció akkor van használatban például, amikor LIN kommunikációs protokoll szerint történik a kommunikáció és a helyi ECU a LIN bus master. Ebben az esetben ez az ECU egyedül dönthet a hálózat leállításáról.

5.1.1.4 Koordinált buszok leállítása.

Amikor a koordinációs klaszter összes eleme készen áll az alvó módra vagy már abban a módban van, elindul a leállítás folyamata az éber hálózatokon. Az NM koordinátor folyamatosan figyelemmel tartja, hogy a feltételek teljesültek-e. A folyamat a következő:

- Készen áll-e a hálózat összes node-ja az alvó üzemmódra?
- Szükség (ha egy hálózat igényli a szinkronizációt) esetén meg várni kell egy szinkronizációs pontot a hálózatban.
- A leállítási időzítők elindulnak. Ők felelnek a leállítás szinkronizálásáért.
- Amikor egy időzítő lejár, az NM elengedi a hálózatot egy <*BusNm*>*_RequestBusSynchronization* függvényhívással, amit <*BusNm*>*_NetworkRelease*. Itt a <*BusNm*> a csatorna kommunikációs protokolljának a jelölése. Pl.: CanNm, FrNm stb. .
- Amikor az összes timer lejárt, a hálózatok alvó módba kerülnek, amit *Nm_BusSleepMode* függvényhívással jeleznek az NM interfésznek.



5.1. ábra Leállítási algoritmus folyamatábrája [13]

A leállítási időzítők hossza kommunikációs protkoll függő. A kiszámítása *NmGlobalCoordinatorTime* – TSHUTDOWN_CHANNEL. Ennek az első tagja egy konfigurációs paraméter, aminek a megadása a rendszertervező feladata. A második egy csatorna specifikus változó amit ki kell számolni a csatorna paramétereknek megfelelően. Például egy CAN hálózat esetén a TSHUTDOWN_CHANNEL = *Ready Sleep Time* + *Prepare BusSleep Time*. Általános busz esetén, ami támogatja a hálózatmenedzselést ez a változó az NM interfész *NmGenericBusNmShutdownTime* konfigurációs paramétere.

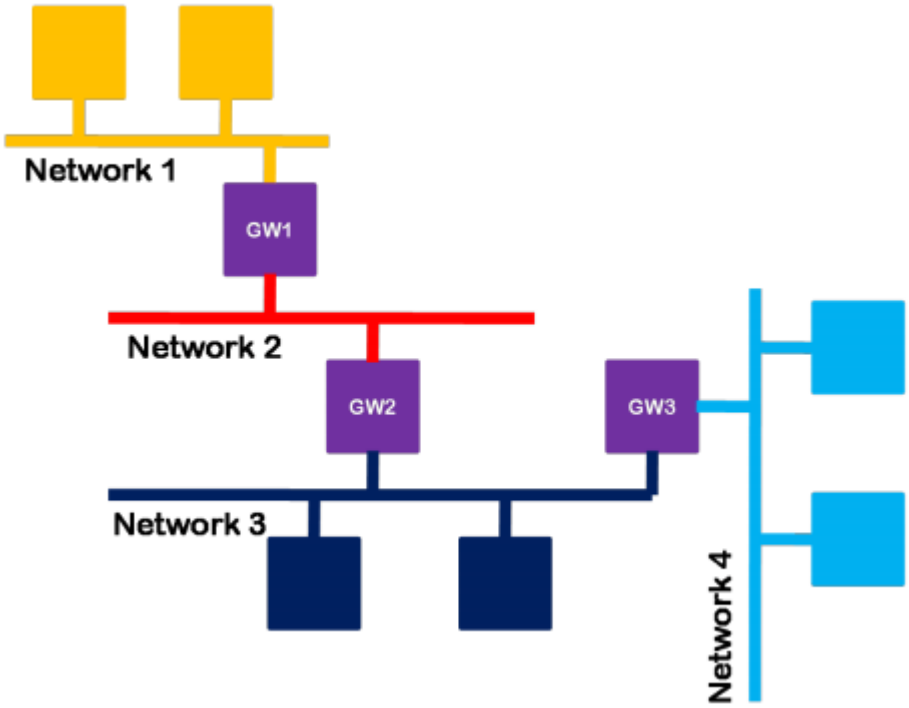
5.1.1.5 Beágyazott buszok leállításának az irányítása.

Amikor befejeződik az NM-PDUk küldése, akkor a hálózat elkezdheti a leállítást, amit mindig az a csatorna vezérel, ami a koordinációs hierarchia szerint a legfelsőbb elem. A kérés egy Sleep-Ready Bit segítségével történik az NM-PDU üzenet CBV részében. Ezt a bitet akkor állítja 1-be az NM koordinátor, amikor:

- a klaszter összes node-ja kész az alvó állapotra,
- szinkronizációs hálózat esetén egy szinkronizációs pontban van a rendszer, ahol az idővezérelt hálózatok is készen állnak a leállásra.
- az összes csatornája aktívan koordinált (hierarchia legfelsőbb eleme).

Amikor egy ECU passzívan koordinált csatornáján érkezik egy jelzés a leállításról egy Nm_*CoordReadyToSleepIndication* függvényhívással, akkor az aktívan koordinált csatornákon bebillenti a Sleep Ready bitet.

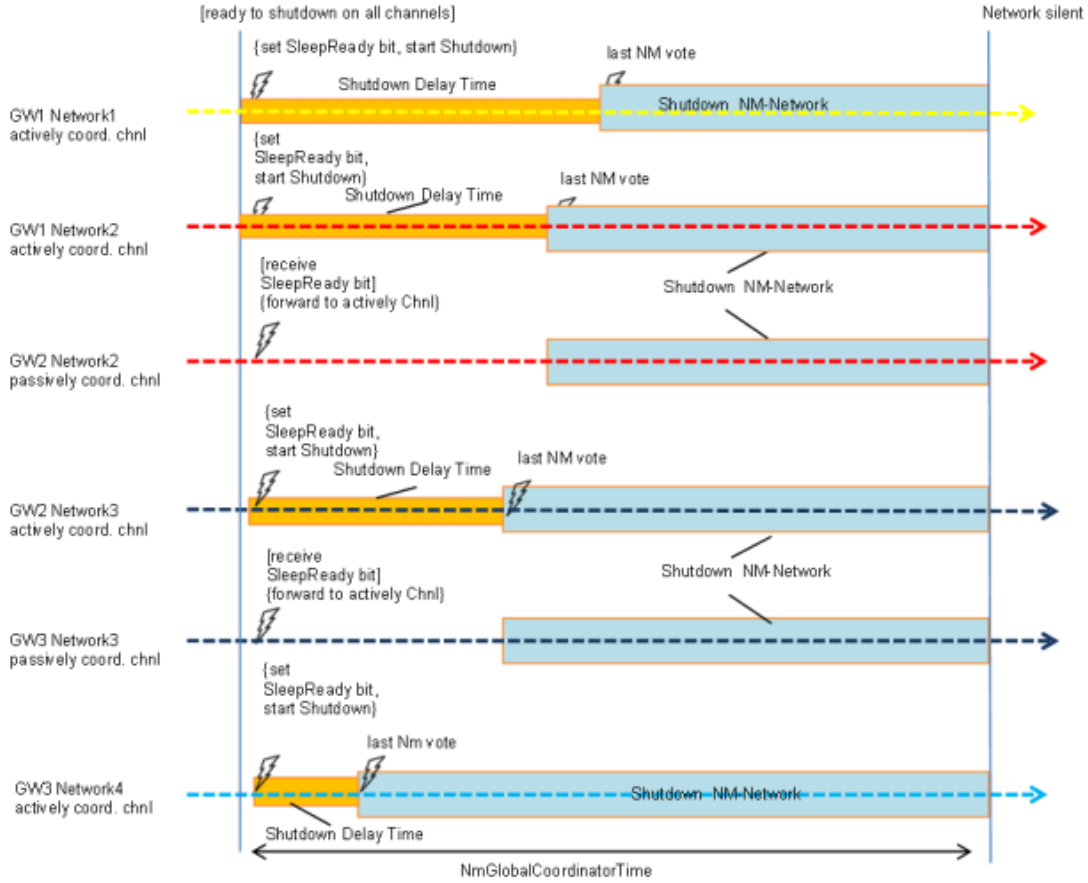
Nm_CoordReadyToSleepCancellation érkezésekor, ami azt jelenti hogy a leállítás folyamatát meg kell szakítani, ez a bit 0 állapotba kerül.



5.2. ábra Beágyazott hálózat [13]

A leállítás folyamata a 3.2-es fejezetben bemutatottak szerint a következő: a legfelsőbb koordinátor csatorna észleli, hogy az összes node készen áll a leállításra, ami a *Sleep Ready* bitet 1-es állapotba helyezi. Ez után a hierarchiában alább lévő koordinátorok passzívan konfigurált csatornájukon keresztül megkapják a leállítási kérést, amit továbbítanak az aktívan koordinált csatornák segítségével a *Sleep Ready Bit* állításával. A példában a Network 1 node-jai kapják meg először a leállítási kérést, és billentik be a Sleep Ready Bitet amibe beletartozik GW1 is. GW1 ezután továbbítja a kérést a Network 2-nek GW2 segítségével, majd ehhez hasonlóan Network 3 is leáll.

A folyamatot a következő ábra szemlélteti:



5.3. ábra Leállítás NmGlobalCoordinatorTime segítségével [13]

5.1.1.6 Felébresztés és a koordinált kikapcsolás megszakítása.

A node és hálózatok általános felébresztéséért az NM modul nem felel. Ezt a funkciót ComM valósítja meg.

Amikor a leállítás feltételei teljesülnek és a koordinálási funkcionalitás engedélyezve van, elindul a koordinált kikapcsolás folyamata. Algoritmustól függően időbe telik amíg az összes busz kikapcsolt állapotba kerül. Ha a klaszter egyik eleme kommunikálni kezdene teljes kikapcsolás előtt, versenyhelyzet alakulhat ki négy féle módon:

- A klaszter hálózatán az egyik node, ami még nem kapcsolt ki, ismét igényt tart a kommunikációra. Ezt a busz specifikus NM modul észleli, és jelenti az NM interfésznek.

- Egyik node a klaszter olyan hálózatán ami már nem igényli a kommunikációt és Prepare Bus Sleep Mode ban van ismét igényt nyújt a hálózati kommunikációra.
- A kommunikációs menedzser igényt nyújt a koordinációs klaszter egyik hálózatára.
- A hálózat aktív koordinátora egy olyan üzenetet küld, amiben a Ready-Sleep bit törölve van. Ezt a passzívan koordinált csatornák busz specifikus Nm moduljai észlelik, majd továbbítják a NM interfésznek.

5.1.2 Változások

Az NmIf a hálózatmenedzsment funkciók csatornákon kerülnek végrehajtásra, amik csatornaazonosítókkal vannak ellátva. A megvalósításban minden egyes csatornához tartozó információ egy adatstruktúrába van felsorolva, hogy egyszerűen elérhető legyen az összes információ a csatornákról. Ez két struktúrából készült el. Egyik tárolja a csatorna statikus, konfigurációs változóit, míg a másik a futásidőben változó információkat.[17]

A modul statikus adatstruktúráját három helyen kellett változtatnom. A globális paraméterekhez egy új változó került: *NmCoordinatorSyncSupport*. Ha ennek az értéke TRUE, a modul elérhetővé teszi a beágyazott alhálózatok koordinálásának a funkcióját. A csatornákhoz tartozó konfigurációs paraméterlistából kikerült a csatorna azonosító. Ez azt jelenti hogy ezt a változót nem lehet konfigurálásnál kézzel beállítani, hanem azt a konfiguráció generátor a ComM modultól kérdezi le. Ezen kívül az eddig globális *NmComUserDataSupport* paraméter csatornára vonatkozó konfigurációs paraméter lett. Ez azt jelenti hogy csatorna specifikusan lehet beállítani hogy egy csatorna a felhasználói adatát a COM modultól kérje le.

A 4.0-s specifikáció szerint a leállítási időket kizárólag konfigurálás közben lehetett megadnia a rendszertervezőnek. Az új szabvány szerint ez a protokoll-függő információ adott egyenlet szerint kerül kiszámításra az eddigi kézi beállítás helyett. Például a CAN leállítási idejét *Ready Sleep Time + Prepare BusSleep Time* szerint lehet kiszámolni. A két tagját az összegnek a CanNm modul konfigurációja tartalmazza. Az új verzióban ez a statikus struktúrában helyet foglaló változó, a konfiguráció generálás során kap értéket a megfelelő kommunikációs protokoll szerinti beállítások szerint.

A 4.0-s verzióban a beágyazott alhálózatok koordinálásának a szabálya szerint bármilyen hálózati topológia kialakítható volt, de egy hálózatban csak egyetlen aktív NM koordinátor lehetett. Ha minden hálózat és node kész az alvó állapotra a klaszterben, elkezdődhetett a leállítás folyamata, amit a hálózat aktív koordinátora indított el a *NmCoordSleepReady* bit 1-re állításával. Ezt a leállítási folyamatot bármely NM klaszterbe tartozó aktív koordinátor elindíthatta.[18] A 4.3-as követelmények szerint olyan hierarchikus topológiának kell fennállnia a csatornák között, amiben a legfelsőbb csatorna olyan koordinációs klaszterhez tartozik, amiben minden elem aktívan koordinált. Leállítási folyamatot csak ezek a csatornák indíthatnak úgy, hogy a hozzájuk tartozó aktív koordinátorok bebillenthetik az *NmCoordSleepReady* bitet az NM-PDU-ban ha a klaszter összes node-ja készen áll a leállításra. A folyamat ezt leszámítva megegyezik a 4.0-s verzióban definiálttal: a passzív NM-koordinátorok fogadják az aktív NM koordinátor jelzését a leállításhoz, és továbbítják azt a node-on belül a klaszter aktív koordinátorának. A változást a koordinációt kezelő segédfüggvények módosításával volt a legegyszerűbb megoldani. A feladat az, hogy kizárólag akkor indulhasson leállítási folyamat a *Sleep-Ready* bit 1-re állításával, amikor az NM klaszter összes eleme aktív koordinátor. Az *Nm_StartClusterShutdown*() függvényt módosítottam, aminek feladata hogy a *Sleep-Ready* bitet 1-re állítsa és elindítsa a csatornák leállítási időzítőit. Ez a függvény úgy változott meg, hogy megvizsgálja a függvényparaméterként megadott csatorna klaszterét és csak akkor állítja *Sleep-Ready* bitet 1-re, amikor a vizsgált klaszter összes csatornája aktív koordinátor.

A kikapcsolás folyamatát a 4.0-s szabvány szerint meg kell szakítani, amikor egy busz már alvó állapotba került és kommunikációs kérés érkezik. A busz specifikus NM modul ebben az esetben nem válthat állapotot, hanem az NmIf modultól ébresztést kér *Nm_NetworkStartIndication*() függvényhívással. Az új megvalósításban ez a feltétel kikerült, helyette a kikapcsolás megszakítása akkor történik meg, amikor olyan üzenet érkezik a busz specifikus NM modulnak, amiben a Ready-Sleep bit törölve lett. Az új szabvány szerint a kikapcsolás folyamata megszűnik, amikor *Nm_CoordReadyToSleepCancellation*() függvény hívódik. A feladat ennek az új *Nm_CoordReadyToSleepCancellation*() API-nak az implementálása, aminek a feladata az, hogy a Sleep-Ready bitet törli az NM-PDU CBV mezőben és meghívja azt a koordinációs segédfüggvényt, ami megszakítja a leállítás folyamatát a szabvány szerint.

Az API-elkészítése során ügyelni kell arra, hogy megfeleljen a szabvány hibadetektálási feltételeinek: ellenőriznie kell hogy híváskor a modul inicializálva van és a függvényparaméterként átadott csatorna azonosító nem hibás. Ez az API-t akkor kerül meghívásra, amikor a passzívan koordinált csatornán olyan NM PDU érkezett, amiben *Sleep Ready* bit 0. Ilyenkor a klaszter összes aktívan koordinált csatornáján a *Sleep Ready* bitet 0-ra állítja.

A hibadetektálásra vonatkozó követelménykeben is történtek változások, azon belül is kiegészült hogy milyen esetekben kell a Det modulnak jelenteni az NM_E_INVALID_CHANNEL hibákat. Ha egy API, amit a modul helyes viselkedés során aktív módban használ, passzív beállítású csatornán hívódott meg annak E_NOT_OK értékkel kellett visszatérnie a régi megvalósításban. Az új módosítások szerint az E_NOT_OK visszatérési érték mellett NM_E_INVALID_CHANNEL hibát kell jelentenie a Development Error Tracernek. Ilyen API például a *Nm_EnableCommunication*(), ami a kommunikációs engedélyezési utasítást továbbítja egy busz specifikus Network Management modul számára.

5.2 CAN Network Management

A következő fejezet a 4.3-as CAN Network Management modul funkcionalitásáról szól, valamint a szabványban történt 4.0-s verzió után történt változásokról és azok implementációjáról. [12]

5.2.1 Funkcionalitás

A CAN Network Management modul működése egy decentralizált direkt hálózatmenedzselési stratégián alapszik, miszerint a rendszer összes node-ja önállóan végzi a az energiamenedzselési funkciókat.

A CAN hálózatmenedzselési algoritmus periodikusan kiküldött NM-üzeneteken alapszik, amiket a klaszter összes node-ja fogad. NM üzenet fogadása azt jelenti, hogy a küldő node-nak szüksége van még a hálózatra, tehát ébren kell tartani azt. Ha bármelyik node készen áll a csökkentett üzemmódú alvó állapotra, abbahagyja az NM üzenetek kiküldését. Minden üzenet érkezésekor elkezd egy konfigurált értékű számláló visszaszámolni, és amikor ez lejár az azt jelenti, hogy a klaszter összes node-ja készen áll az alvó üzemmódra.

Ezt az algoritmust két fő követelménnyel lehet jellemezni:

- A CanNm klaszter összes node-ja periodikusan NM üzeneteket küld, amíg annak szüksége van a buszon való kommunikációra. Ellenkező esetben az NM üzenetküldés megszűnik.
- Ha a CanNm klaszterben megszűnik a kommunikáció a buszon, és nem érkezik NM üzenet egy előre konfigurált ideig, akkor alvó módba kerül a rendszer.

5.2.1.1 Operációs módok

A 3.2.1 fejezetben bemutattam, hogy a CAN hálózatmenedzsment állapotgépe milyen állapotokba kerülhet. Minden elektromos vezérlőegység CanNm modulja egy-egy állapotot vesz fel a csatornáihoz, amik egy fizikai kommunikációs csatornának felelnek meg. Ezek az állapotok megfelelnek a hozzájuk tartozó fizikai buszok állapotának. Ebben az alfejezetben bemutatom ezeknek az állapotoknak a funkcióit, valamint hogy adott eseményre melyik másikat módba kellkerülniük.

Kezdetben az elektromos vezérlőegységeken a modulok egy inicializálatlan állapotban vannak. A CanNm esetén ebből az állapotból egy *CanNm_Init*() függvényhívással lehet, miután az állapotgép Bus-Sleep módba kerül.

Amikor az egyik node NM üzenetet kap **Bus Sleep Mode**-ban, azt jelzi azt a Network Management interfésznek egy *Nm_NetworkStartIndication*() függvénnyel. Ez a bekapcsolási kérést továbbítja a ComM modulnak. Amennyiben a csatorna bekapcsol, az ébresztés történhet passzív vagy aktív módon a kérésnek megfelelően. Passzív esetben *CanNm_PassiveMode*() függvényhívással, *CanNm_NetworkRequest*() hívással pedig az aktív ébresztésnél. Ez után az állapotgép átvált Network Mode-ba.

A **Network Mode**-ba kerülést a CanNm modul jelenti az NmIf modulnak *Nm_NetworkMode*() függvényhívással. Ezen kívül elindul egy NM-Timeout számláló, ami újraindul minden egyes sikeresen küldött vagy fogadott PDU esetén. Amikor Network Mode-ba kerül az állapotgép, az azon belül is a Repeat Message State-et veszi fel.

Repeat Message állapotban újraindul az NM PDU-k kiküldése amennyiben nem passzív módban működik a modul. Az állapotgép mindaddig tartja a Repeat Message állapotot és küld ki NM üzeneteket, amíg le nem jár egy másik számláló, amit egy előre

konfigurált hosszra lett beállítva *CanNmRepeatMessageTime* szerint. Miután ez a számláló lejárt, átvált a modul Normal Operation módba, ha még szüksége van a buszra, vagy ha nem, Ready Sleep állapotba.

Normal Operation State-ben is az NM PDU-k kiküldése a feladat, ha a node nincs passzív módban. Innen a modul Ready Sleep állapotba kerülhet, ha nincs többet szüksége a buszra egy *CanNm_NetworkRelease*() függvényhívással. Ezen kívül még Repeat Message állapotba is juthat, ha *CanNm_RepeatMessageRequest*() hívást kap a vezérlőegységtől, vagy amikor egy olyan NM üzenetet kap, amiben a *Repeat Message Request* bit 1-be lett állítva.

Amikor Normal Operation módból került a node **Ready Sleep** állapotba, akkor megszűnteti a modul az NM üzenetek kiküldését. Ready Sleep Mode-ból állapotot vált Normal Operation módba, amikor kommunikáció igénye mellett egy *CanNm_NetworkRequest*() függvényhívás érkezik. Repeat Message állapotba akkor kerülhet a modul, ha *CanNm_RepeatMessageRequest*() függvényhívás történik, vagy a beérkezett NM-üzenetben a Repeat Message Request bit 1-be lett állítva.

A **Prepeare Bus Sleep** módot jeleznie kell az általános hálózatmenedzsment modulnak egy *Nm_PrepareBusSleepMode*() függvényhívással. Ebben a módban ha sikeresen fogad a modul egy NM üzenetet, az állapotgépnek át kell váltania Network Mode-ba, azon belül is Repeat Message State-be. *CanNm_NetworkRequest*() hívás esetén a hálózat szintén Network Mode-ba kerül, és ha a konfigurációban azonnali újraindítást állítottak be (*CanNmImmediateRestartEnabled*), akkor minél előbb NM-üzenetet kell a node-nak küldenie. Ha egy előre meghatározott ideig (konfigurációs paraméter: *CanNmWaitBusSleepTime*) nem érkezik NM-üzenet és igény a hálózatra, akkor a node Bus Sleep módba kerül.

5.2.1.2 Kommunikáció ütemezése

Küldés

Küldeni a node NM-üzenetet akkor tud, amikor aktív üzemmódban működik. Normal Operation módban és Repeat Message állapotban periodikusan történik a PDU-k kiküldése. Ha Bus Sleep-ből került Network Mode-ba a modul aktív felébresztéssel, és azonnali NM üzenetküldés lett beállítva előre megadott számú üzenettel (*CanNmImmediateTransmission* darab), akkor a legelső PDU-t minél hamarabb kell

kiküldenie. Ezeket az azonnal kiküldendő üzeneteket *NmImmediateNmCycleTime* időnként küldi ki.

Fogadás

Ha egy NM-PDU-t sikeresen fogadtak, akkor a CanIf modul azt egy *CanNm_RxIndication*() függvényhívással jelzi. Ilyenkor a CanNm modul a függvényben megadott információt elmenti egy belső pufferbe.

5.2.1.3 Busz terheltség csökkentése

A klaszterben az összes node közös (*CanNmMsgCycleTime)* időközönként küldi ki az NM üzeneteiket. Emiatt a busz terheltsége függ a node-ok számától. A terheltség csökkentésére a következő két feltétel teljesülése szükséges:

- NM üzenet érkezésekor a következő üzenet egy egyedi, node-ra definiált idő után kerül kiküldésre (*CanNmMsgReducedTime*) ami kevesebb mint a közös paraméter (*CanNmMsgCycleTime*), de annak a felénél nagyobb.
- NM üzenet kiküldése után a következő üzenetet *CanNmMsgCycleTime* után kerül kiküldésre.

Ennek a hatására csak a klaszter két legkisebb *CanNmMsgReducedTime* paraméterű node-ja fog felváltva üzenetet küldeni a hálózaton. Ha ezek közül az egyik megszűnteti a küldést, akkor a klaszter következő legkisebb *CanNmMsgReducedTime* paraméterű node-ja kezd el üzenetet küldeni. Az algoritmus így bztosítja, hogy a busz terheltsége korlátozva van maximum kettő NM üzenet / *CanNmMsgCycleTime*-ra.

5.2.1.4 Távoli alvó mód detektálás

Egy Normal Operation módban lévő node figyeli, hogy a klaszterén belül az összes többi node készen áll-e az alvó módra, amit onnan tud, hogy *CanNmRemoteSleepIndTime* időn belül nem érkezett NM üzenet a hálózaton. Ilyenkor a modul jelez a felsőbb NmIf modulnak *Nm_RemoteSleepIndication*() függvényhívással. A távoli alvó mód detektálást vissza kell vonnia a modulnak, ha Normal Operation Mode-ban vagy Ready Sleep állapotban üzenetet fogadott egy Remote Sleep Indication jelzés után. Ezen kívül Repeat Message állapotba kerüléskor is meg kell a távoli alvó módot szűntetni, és ezt jelenteni az NmIf modulnak *Nm_RemoteSleepCancellation*() függvényhívással.

5.2.1.5 Felhasználói adat

A CanNm NM üzeneteiben lehetőség van felhasználói üzenetek tárolására, amik a koordináció irányításával kapcsolatosak. Ezeknek az adatoknak a küldésére és fogadásra a modulnak erre megfelelő API-kat kell nyújtania. Egy másik alternatíva szerint ezeket a felhasználói adatokat a COM modul segítségével is lehet fogadni.

5.2.1.6 Koordináció szinkronizálás támogatása

Ha egy buszra egyszerre legalább kettő koordinátor node van csatlakozva, akkor az NM üzenet *NmCoordinatorSleepReady* bitje határozza meg, hogy a fő koordinátor kéri-e a leállítás elindítását. A szinkronizált leállítást az általános hálózatmenedzsment modul valósítja meg. A CanNm modul felelős azért hogy jelezze az NmIf-nek mikor kész a leállításra, vagy mikor kell a leállítást megszakítani.

5.2.1.7 Részhálózat kezelés

A részleges hálózatkezelés azt jelenti, hogy egy algoritmus segítségével kiszűrjük azokat az NM üzeneteket, amik a kijelölt ECU számára nem relevánsak, így ha nem érkezik releváns üzenet, akkor az NM-Timeout számláló lejár, és a modul átkerülhet alvó állapotba.

Ez úgy történik, hogy a klaszteren belül a nodeok egy kisebb csoportba sorolódnak a funkciójuk szerint. A CanNm üzenetekben a CBV mezőben a PNI bit segítségével, ha az engedélyezve van, lehet megadni hogy az üzenet melyik alhálózat számára tart vezérlő információt. Ez az információ a felhasználói adat mezőben érkezik. A külső részhálózat kéréseket az ERA (External Request Array) tárolja, míg a belső kéréseket a külsővel közösen az EIRA (External Internal Request Array). A részhálózatot megvalósító logikaát a ComM modul valósítja meg, a CanNm modul csak azt az információt adja át a ComM-nek hogy történt-e részhálózat igény.

5.2.1.8 Küldési hibakezelés

Konfigurációtól függően, a CanNm értékeli hogy a NM üzenetek küldése sikeres volt-e vagy sem. Ha egy megadott időn belül a sikeres fogadás visszajelzése nem történik meg, akkor a modul ezt jelzi a felsőbb rétegeknek.

5.2.1.9 Hibakezelés

A CanNm modulnak a következő hibákat kell észlelnie és továbbítani a Development Error Tracer modulnak.

- Inicializálatlan modul. API hívás történt amikor a modul még inicializálatlan állapotban volt.
- Rossz csatorna azonosító. Érvénytelen azonosítóval történt API hívás.
- Rossz PDU azonosító. Érvénytelen PDU azonosítóval történt API hívás.
- NM PDU érkezett Bus-Sleep Mode-ban.
- Sikertelen inicializáció.
- NM-Timeout Timer rendszertelenül, Ready Sleep állapoton kívül állt le.
- Null pointer-t adtak át argumentumként.
- *CanNm_DeInit*() API hívás történt, amikor nem minden CAN hálózat volt Bus-Sleep Mode-ban.

5.2.2 Változások

A Network Management Interface-hez hasonlóan a CanNm modul is több csatornát kezel. Ennek a modulnak az adatszerkezetében is minden csatornához hozzá van rendelve két struktúra: az egyik tárolja a konfigurációs információkat, míg a másik a futásidejű adatokat.

A 4.3-as szabványban a régi megvalósítással szeben egyszerre több RxPdu-t is kezelhet a modul. Eddig az RxPdu-kat a konfigurációs source fájlban úgy valósították meg, hogy azt egy adatmezővel (integereket tároló tömb) és egy azonosítóval definiálták. Az azonosító a csatornák konfigurációs adatait tartalmazó struktúrához tartozik. A módosítás megvalósításához ezt a struktúrát, mivel több RxPdu is lehet egyszerre, ki kellett egészíteni egy *numberOfRxPdus* változóval ami ezeknek az azonosítóknak a darabszámát tartalmazza. Ha egyszerre több ilyen RxPdu-t definiál a felhasználó, akkor a konfigurációs source fájlban annyi RxPdu adatmezőt kell létrehozni valamint egy tömböt, ami tárolja az RxPdu-k azonosítóit. Így a csatornák konfigurációs struktúrájában az eddigi egy darab RxPdu azonosító helyett egy pointert adunk át, ami a csatorna RxPdu azonosítóinak a tömbjére mutat. Ezt a megoldást le lehet egyszerűsíteni úgy, és így erőforrást spórolni, hogy egyetlen egy RxPdu adatmezőt

definiálunk. Ezt ennél a modulnál megtehetjük, mivel mindig a legutoljára fogadott üzenetet kerül feldolgozásra, így elég ha csak azt az egy PDU-t tárolja el modul.

A megvalósításhoz az adatstruktúrák mellett módosítanom kellett egy segédfüggvényt is, mivel az RxPdu-k mennyiségének a változása befolyásolta annak működését. Ennek a függvénynek a feladata hogy az átadott PDU azonosító szerint visszatérjen a PDU-hoz tartozó csatorna azonosítójával. A működése ennek a változás előtt úgy valósult meg, hogy megvizsgálta az összes csatornát és összehasonlította a csatronaazonosítót az argumentumban átadott azonosítóval. Ha ezek megegyeztek akkor visszatért azzal az azonosítóval. Elméletben a működés nem változott, csak csatornánként több azonosító is lehet, ezért a csatornák adatszerkezetében nem egy változót kell vizsgálni, hanem többet, melyek egy tömbben foglalnak helyet.

A *CanNm_Deinit()* egy új API, aminek a feladata hogy a modult Bus-Sleep Mode-ból egy uninicializált állapotba vigye. A megvalósításban ellenőrizni kell azt a feltételt, hogy a klaszter összes node-ja alvó busz módban van. Ha nem minden busz van abban az állapotban, akkor hibát jelent a Development Error Tracer modulnak. Az implementálásnál ellenőrizni kell, hogy a hívás megfelel-e az előbb leírt feltételeknek. Ha megfelelt, akkor uninicializált állapotba válthatja az összes csatornát.

Egy másik új API, ami bekerült a 4.3-as verzióba a *CanNm_TriggerTransmit()*.

Ennek a feladata, hogy ismételt PDU elkérés esetén a CanIf modul számára a függvényparaméterként megadott TxPdu azonosítójú PDU-ba átmásolja a másik paraméterként átadott PDU adatát. Ha a COM modul felelős a felhasználói adatok lekéréséért, akkor a felhasználó adatot is átmásolja a TxPdu-ba. Ezt a függvényt az alsóbb rétegek moduljai hívják meg, amikor azoknak a felsőbb rétegektől ismételten el kell kérniük az adott PDU-t.

Az NM-PDU-k kiküldése a CanIf modulon keresztül történik meg. A PDU kiküldése után a CanIf modul ennek a *CanNm_TxConfirmation()* függvénynek a segítségével közli a CanNm modullal, hogy, a PDU kiküldése sikeres volt vagy sem. A 4.0-s megvalósításban ha egy PDU kiküldése sikertelen volt, akkor ez a függvény egyátalán nem került meghívásra. Az új szabvány szerint ennek a függvénynek az argumentumlistája kiegészült egy Boolean típusú *result* paraméterrel, amiben a CanIf modul közli hogy a PDU kiküldése sikeres volt-e vagy sem. Ennek a segítségével könnyebben lehet kezelni a sikertelen küldés esetét, aminek a során értesíteni kell az NmIf modult.

A 4.0-s szabványban a felhasználói adat hosszát a PDU-ban a rendszertervező állíthatta be. A 4.3-as modulban ez a lehetőség kikerült, helyette ezt az információt a TxPdu hosszából lehet kiszámítani. A PDU Router modulban lehet konfigurálni a PDU-k hosszát, a CBV és NID byteokat pedig a CanNm modulban. Ezeknek a változóknak a felhasználásával a felhasználói adat hosszaát megkapjuk, ha a TxPdu hosszából kivonjuk a CBV és NID adatmezők hosszát. A frissített implementációban a konfiguráció generálás után kap értéket ez a változó automatikusan.

Szinkronizált leállítási folyamatnál a leállítási kérést a PDU-kban fogadott *NmCoordinatorSleepReady* bit jelzi a CBV-ben. A modul feladata hogy a bit értékének megfelelően kezelje a leállítási kérést. Az új szabványban ez a funkcionalitás kiegészült a leállítás megszakításának a kezelésével. Ha a beérkezett PDU-ban az *NmCoordinatorSleepReadyBit* 1, és a modul állapotgépe Network Mode-ban van vagy a leállítás folyamata megszakításra került ezelőtt, jelzi az NmIf modulnak az újabb leállítási kérést *Nm_CoordReadyToSleepIndication*() függvényhívással. Amikor ez a bit a PDU-ban 0 értékű, valamint a leállítás folyamata elindult, a megszakításról a modul értesíti az NmIf modult *Nm_CoordReadyToSleepCancellation*() függvényhívással.

Ennek az új funkciónak a megvalósítását az RxPdu-k feldolgozását kezelő függvénybe illesztettem be. Az eseményeket, amik jelzik, hogy korábban történt-e már leállítási kérés vagy megszakítás egy globális flag-ként deklaráltam a csatornák futásidejű adatstruktúrájában, és használtam fel a *Sleep-Ready* bit kezelésnél.

5.3 FlexRay Network Management

Ez a fejezet a 4.3-as FlexRay Network Management modulról szól. Itt bemutatom a modul funkcionalitásait és a változásokat amik az új szabványban történtek, valamint azok implementálását. [15]

5.3.1 Funkcionalitás

Az előző fejezetben bemutatott CAN Network Management-hez hasonlóan a FlexRay Network Management modul is egy decentralizált hálózatmenedzselési stratégián alapul. Ugyan úgy ez itt is azt jelenti, hogy minden node önállóan végzi el az energiamenedzselési feladatait a hálózatmenedzselési üzenetek segítségével.

A FlexRay koordinációs algoritmusa periodikus NM-Vote üzeneteken alapul. Ezeket az üzeneteket a klaszter összes tagja fogadja. Üzenet fogadása esestén a küldő

félnek szüksége van a hálózatra ezért ébren kell tartani azt. Ha bármely node készen áll a csökkentett energiafelhasználású Bus-Sleep módra, akkor abbahagyja a pozitív NM-vote üzenetek kiküldését, helyettük negatívat küld, és ha egy előre meghatározott ideig nem is fogad olyan üzenetet amiben a szavazó bit pozitív,,,,,, akkor megtörténhet az állapotváltás az alvó üzemmódba.

Egy node fel is ébresztheti a hálózatot, ha annak szüksége van a buszon történő kommunikációra. Ez úgy történik meg, hogy NM-Vote üzeneteket kezd el küldeni.

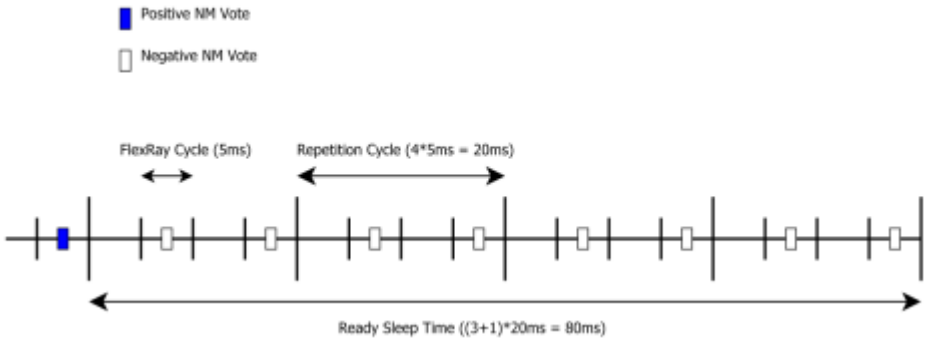
A FlexRay NM modulnak a következő funkciókat kell ellátnia:

- Periodikusan frissítenie kell a FlexRay NM-PDU-kat.
- Kódolnia és dekódolnia kell a FlexRay NM-PDU-kat.
- Küldési hibakezelés megvalósítása.
- Az általános hálózatmenedzselési interfész értesítése állapotgép-beli váltások esetén.

Az algoritmus alapja, hogy az összes FlexRay hálózatmenedzselésben résztvevő ECU egy globális idővel szinkronizáltan működik, ami a kommunikációs séma ismétlődésén alapszik. Ezt a szabvány Repetition Cycle-nak (ismétlési ciklusnak) nevezi. Az aszimmetrikus viselkedés megelőzésére az FrNm az állapotváltásait illeszti az NM Repetition Cycle-hoz, ami pedig a FlexRay kommunikációs ciklushoz illeszkedik. Így garantált, hogy az összes ECU állapotváltozása szinkron módon történik az NM klaszteren.

Ha a buszon történő kommunikáció megszűnik, akkor a modul állapotgépe Bus-Sleep Mode-ba kerül az *FrNmReadySleepCnt*+1-edik ismétlési ciklus (Repetition Cycle) végén, ahol *FrNmReadySleepCnt* egy konfigurációs paraméter. Az időt, amit az *FrNmReadySleepCnt*+1 képvisel kiszámolható a következő képpen: (*FrNmReadySleepCnt*+1) * *FrNmRepetitionCylce* * ’Egy FlexRay ciklus hossza’. Példa erre:

Ha FrNmReadySleepCnt = 3; Repetition Cycle = 4; FlexRay Cycle = 5msec, akkor ez az idő (3 + 1) * 4 * 5msec = 80msec.



5.4. ábra Ready Sleep idő számítása [15]

5.3.1.1 Operációs módok

A 3.2.2 fejezetben bemutattam a FlexRay hálózatmenedzsment modul állapotgépének az állapotait. Minden egyes FlexRay csatorna egy állapotgépet rendel magához, aminek az állapota megegyezik a busz állapotával. Az állapotok közötti váltásokat a modul jelzi az általános hálózatmenedzsment modulnak az *Nm_StateChangeNotification*() függvényhívással, ami pedig a ComM modult értesíti. Mint a CanNm modul fejezetében itt is bemutatom az egyes állapotok funkcióit és az állapotváltásokat előidéző eseményeket.

Inicializáció után a node állapotgépe Bus-Sleep Mode-ba kerül, ahonnan a kommunikációs kontroller alvó állapotba válthat, ezzel az energiafogyasztást minimalizálva.

Bus-Sleep Mode-ot akkor hagyhatja el a modul, amikor pozitív Nm szavazatot kap, vagy a felsőbb rétegektől kap aktív vagy passzív felébresztési kérést. Pozitív szavazat érkezése esetén erről értesül az NmIf modul *Nm_NetworkStartIndication*() függvényhívással, és ComM modulnak továbbítja azt. A ComM modul a beérkezett kérések hatására elindítja a hálózati kommunikációt. A felébresztési folyamatban a ComM modul értesíti az NmIf (valamint az FrSM) modult, ami továbbítja a kérést az FrNm modulnak: ha az ébresztés passzív, *FrNm_PassiveStartUp*(), ha pedig aktív akkor *FrNm_NetworkRequest*() függvényhívással. Ébresztés után az állapotgép Synchronize Mode-ba jut.

Synchronize Mode-ban a módban az állapotgép megvárja, amíg megtörténik a szinkronizálódás a csatornához. Ilyenkor a modul nem küld se NM adatot se NM

szavazatot. Ezt az állapotot a modul elhagyja és Network Mode-ba áll, amikor legelőször két ismétlési ciklus közé kerül. Hiba esetén, amit az *FrNm_StartupError*() jelez két esemény történhet: ha a csatornára igényt tartanak az állapotgép marad Synchronize Mode-ban, ha pedig nincs igény a csatornára akkor alvó állapotba kerül.

Network Mode-ba kerüléskor a modul a Repeat Message alállapotot veszi fel. Ha *FrNm_Init*() függvényhívás történik, akkor ezt az állapotot az állapotgép azonnal elhagyja.

Repeat Message State-ben indul el el az NM adatok kiküldése, amennyiben azt a konfiguráció során beállították. Ezzel együtt a node a klaszterének az ébrentartását is megszavazza. Belépéskor egy számláló indul el, ami ha lejár, az állapotgép elhagyja a Repeat Message állapotot. Az állapotváltás itt is két ismétlési ciklus határa között történik, mivel a számláló kezdeti értéke egy ciklus hosszának egész számú többszöröse. Ilyenkor az állapotgép Ready Sleep State-be kerül, ha a csatornára nem érkezett kommunikációs kérés. Ellenkező esetben az új állapot a Normal Operation State. Olyan hiba esetén, amikor a globális időt nem éri el a modul, Synchronize State-be kell kerülnie.

Normal Operation State-ben kiküldésre kerülnek az NM-adatok amik egyben a klaszter ébrentartását szavazzák meg pozitív NM-szavazatok kiküldésével. Repeat Message State-be vált az állapotgép, ha az NmIf modultól *FrNm_RepeatMessageRequest*() függvényhívás érkezik. Ha a hálózatra nincs igény és egy ismétlési ciklus végén jár a klaszter, akkor a modul a Ready Sleep állapotot veszi fel. A Repeat Message State-hez hasonlóan ha olyan hiba adódik, amikor a globális időt nem éri el a modul, Synchronize állapotba kell kerülnie.

A **Ready Sleep** állapot megvárja, amíg a klaszter többi node-ja is készen áll az alvó üzemmódra. Ilyenkor megszűnteti a modul az Nm-adatok kiküldését, és negatív NM-szavazattal az alvó módba lépést szavazza meg. Bus-Sleep módba akkor kerül a modul, amikor *FrNmReadySleepCnt*+1 ismétlési ciklus telt el pozitív szavazat fogadása nélkül. Ebből az állapotból is kerülhet a Normal Operation State-hez hasonlóan az állapotgép Repeat Message State-be *FrNm_RepeatMessageRequest()* hívás esetén. Ha az egyik ismétlési ciklus végén a hálózatra igény érkezett, az azt jelenti hogy nem áll készen a node az alvó üzemmódra, így Normal Operation State-be kell kerülnie.

A modul konfigurációjától függően rendelkezhet egy olyan funkcióval, ami az ismétlési ciklust emulálja egy számláló segítségével. Ennek a neve FRNM_CYCLE_COUNTER_EMULATION. Ha ez a funkció engedélyezve van, és a globális idő lekérése során hiba keletkezett, az emulált számlálót veszi figyelembe a modul. Amikor ez a számláló az ismétlési ciklusa végére ért, akkor az egy olyan NM Repetition Cycle-nak számít amiben nem érkezett pozitív szavazat. Abban az esetben amikor ez a funkció nincs engedélyezve és globális idő elérési hiba keletkezett, a modul a csatorna igényeltségének megfelelően átkerül Synchronize vagy Bus-Sleep módba.

5.3.1.2 Kommunikáció

A FlexRay NM hálózatmenedzseléssel kapcsolatos funkcióit a beérkezett NM-szavazatok befolyásolják. A 3.1.2 fejezetben leírtak szerint a modul képes NM-adat és NM-szavat üzenetek kiküldésére. A FlexRay szavazó algoritmusa független az NM-adatoktól, mivel a FlexRay protokoll harveresen támogatja az NM-szavazatok küldését és fogadását az NM-adattól függetlenül. Ennek köszönhetően felgyorsul az NM-szavazatok periodikus kiküldése az NM-adatokhoz képest, és így gyorsulnak az egyes hálózatmenedzslési döntések kiértékelése. A szabvány ezt a funkciót úgy támogatja, hogy konfigurációsan (FRNM_PDU_SCHEDULE_VARIANT paraméter segítségével) állítható hogy az NM-szavazat és NM-adat statikus, dinamikus vagy esetleg külön-külön szegmensben kerüljenek kiküldésre. Hét ilyen variánst definiál a szabvány erre:

- NM-szavazat és NM-adat egy PDU statikus szegmensében foglalnak helyet.
- NM-szavazat és NM-adat egy PDU dinamikus szegmensében foglalnak helyet.
- NM-szavazat és NM-adat különböző PDU-k statikus szegmensében foglalnak helyet.
- NM-szavazat a statikus, míg az NM-adat a dinamikus szegmensben foglal helyet.
- NM-szavazat a dinamikus, míg az NM-adat a statikus szegmensben foglal helyet.
- NM-szavazat és NM-adat különböző PDU-k dinamikus szegmensében foglalnak helyet.

- Kombinált NM-szavazat és CBV a statikus, és az NM-adat a dinamikus szegmensben foglal helyet. (3.1.2-ik fejezet)

5.3.1.3 Távoli alvó mód detektálás

A távoli alvó mód a CAN NM modulban leírtakhoz hasonlóan (5.2.1.4 fejezet) azt jelenti, hogy egy node észreveszi azt, amikor a klaszterén belül az összes többi node készen áll az alvó módra, és csak ő az egyetlen aki a hálózatot ébren tartja. Ha ez bekövetkezik, és egy megadott ideig nem érkezik pozitív NM-szavazat, erről a modul értesíti az NmIf modult.

Az FrNm modulnak is vissza kell utasítania a távoli alvó busz mód jelzését, ha az nem Network Mode-ban van. Abban az esetben, amikor távoli alvó busz jelzés után kérés érkezik Normal Operation vagy Ready Sleep módban a kommunikációra a buszon vissza kell vonni a jelzést és az esetet jelenteni kell az NmIf modulnak *Nm_ReadySleepCancellation*() függvényhívással. Ugyan ez a teendő akkor is, amikor jelzés történt és a modul Repeat Message állapotba kerül Normal Operation vagy Ready Sleep módokból.

5.3.1.4 Felhasználói adat

A FlexRay NM-adat üzeneteiben koordináció irányításával kapcsolatos felhasználói adatok lehetnek. Ezeknek az adatoknak a küldésére és fogadásra a modulnak nyújtani kell ehhez szükséges API-kat. Ezeket a felhasználói adatokat, amennyiben a modult megfelelően konfigurálták, a COM modul segítségével is lehet fogadni.

5.3.1.5 Duális FlexRay csatornák támogatása

Az előző fejezetekben leírt PDU felépítési variánsokon kívül a FlexRay protokoll lehetőséget ad két fizikai csatorna használatára, melyeken adatküldés valósítható meg. Ilyenkor a FlexRay szegmensek a két csatorna egyikét vagy egyszerre mindkettőt igénybe vehetik a kommunikációra.

5.3.1.6 Koordinált buszleállítás

Amikor több mint egy koordinátor ECU csatlakozik egy buszhoz, a CBV *NmCoordinatorSleepReady* bitje jelzi, amikor a fő koordinátor elindítja a leállítás folyamatát. Amikor az FlexRay modul egy olyan NM üzenetet fogad, amiben ez a bit 1-

re van állítva, és a modul állapotgépe Network Mode ban van jelzi az NmIf-nek a koordinált leállítási kérést *Nm_CoordReadyToSleepCancellation*() függvényhívással. Ha viszont az üzenetben a bit 0, és már jeleztek az NmIf-nek a koordinált leállításról, akkor azt vissza kell vonni a leállítást *Nm_CoordReadySleepCancellation*() függvényhívással.

5.3.1.7 Részhálózatok kezelése

A részleges hálózatkezelés, mint a CanNm modulnál is azt jelenti, hogy egy algoritmus segítségével kiszűrjük azokat az NM üzeneteket, amik a kijelölt ECU számára nem tartanak releváns üzenetet. Így ha nem fogad adott ideig olyan üzenetet a node, amiben a szavazóbit 1-re van állítva, átállítja az állapotát Ready Sleep mode-ba.

Az FrNm üzenetekben a CBV mezőben a PNI bit segítségével, ha az engedélyezve van, lehet megadni hogy az üzenet melyik alhálózat számára tart vezérlő információt az adott üzenet. Ez az információ a felhasználói adat mezőben érkezik. A külső részhálózat kéréseket az ERA (External Request Array) tárolja, míg a belső kéréseket a külsővel közösen az EIRA (External Internal Request Array). A részhálózatot megvalósító logikaát a ComM modul valósítja meg, az FrNm modul csak azt az információt továbbítja a ComM-nek hogy történt-e részhálózat igény.

5.3.1.8 Hibakezelés

Az FrNM a következő fejlesztés időben keletkezett hibákat észleli, és erről értesíti a Development Error Tracer modult:

- Inicializálatlan modul. API hívás inicializálatlan modul esetén.
- Rossz csatornaazonosító. API hívás hibás csatorna azonosítóval
- Rossz PDU azonosító. API hívás hibás PDU azonosítóval
- Sikertelen inicializáció.

5.3.2 Változások

Az FrNm modul adatszerkezete is követi az előző két modul példáját. Csatornánként itt is két fő struktúrával van megvalósítva: egyik a konfiguráció során generált statikus adatokat tárolja, míg a másik a futás időben változókat.

A CanNm modulhoz hasonlóan változott a szinkronizált leállás kezelése az FrNm modulban is. A leállítási kérést a beérkezett PDU *NmCoordinatorSleepReady* bitje jelzi. Ha ez a bit 1 és Network Mode-ban működik a modul, akkor *Nm_CoordReadySleepIndication*() függvényhívással jelez az NmIf modulnak. Másik esetben, amikor ez a bit 0, valamint már elindult a leállítási folyamat akkor meg kell szakítani azt. Ezt is jelezni kell az NmIf-nek ami *Nm_CoordReadyToSleepCancellation*() függvényhívás segítségével történik meg. A 4.0-s szabványban nem volt a második eset kezelése implementálva, ezért ezt a beérkező PDU-kat kezelő függvényekhez illesztettem. A csatorna specifikus változókat a futás közben változó struktúrában kiegészítettem egy-egy flaggel, amik jelzik hogy a modulban történt-e leállítási kérés vagy megszakítás.

Az *FrNm_TriggerTransmit()* API-t az alsóbb rétegek moduljai hívják meg, amikor azoknak a felsőbb rétegektől ismételten el kell kérni az adott PDU-t. A 4.3-s megvalósítás kiegészült azzal, hogy amikor a COM modul felelős a felelős a felhasználói adat lekéréséért akkor a PDU Routertől kell azt az adatot lekérni *PduR_FrNmTriggerTransmit*() függvényhívással és kombinálni a további NM byteokkal.

Amikor a FlexRay State Machine modulban a klaszter szinkronizációja sikertelen, az *FrNm_StartupError()* függvény hívódik meg. Ilyenkor a modulban állapotváltás történik aszerint, hogy a hálózatra a modul igényt tart-e vagy sem. Ha igény van rá, a régi működés szerint Normal Operation (vagy Repeat Message ciklus számlálási emuláció hiányában) módból Synchronize állapotba kell váltania. Az új működésben ezek az állapotváltások akkor történnek meg, amikor a globális idő lekérése sikertelen és *FrNm_StartUpError*() híváskor. Ha ilyenkor Synchronize állapotban működik a modul, akkor ott is marad. Abban az esetben, amikor nincs igény a hálózatra, a modul Synchronize módból, Repeat Message módból vagy Ready Sleep módból Bus-Sleep Mode-ba kerül, valamint Normal Operation esetén Synchronize módba vált. Az új, 4.3 szabványspecifikációk szerint Bus-Sleep Mode-ba kell váltania az NM állapotgépnek Synchronize vagy Ready-Sleep Mode-ból. A módosult állapotváltásokat úgy tudtam a függvényben megoldani, hogy ellenőriztem a modul állapotát a megfelelő futás idejú adatstruktúrában található adatokkal (jelenlegi mód, állapot és csatorna igényeltsége), és amikor ez teljesült a megfelelő, már rendelkezésemre álló állapotváltást megvalósító függvényt hívtam meg.

Ugyanúgy, mint a *CanNm_TxConfirmation*() API argumentumlistája, a *FlexRay_TxConfirmation*() függvénye is kiegészült egy *result* argumentummal. Ez itt is azt jelenti, hogy a függvényt hívó FlexRay Interface ennek a változónak a segítségével tudja továbbítani az FrNm modulnak hogy egy PDU kiküldése sikeres volt-e vagy sem. A 4.0 szabványban ez az API csak akkor hívódott meg, amikor sikeres volt a PDU küldés. Ellenkező esetben egyátalán nem hívta meg a függvényt az interfész modul. Az FrNm ebben az esetben onnan tudta, hogy sikertelen volt a PDU küldés hogy nem érkezett visszaigazolás a küldésről egy előre megadott időn belül.

6 Hálózatmenedzsment modulok konfigurációja

Ebben a fejezetben az AUTOSAR szabvány szerinti konfigurációk felépítését fogom bemutatni, ami tartalmazza konfigurációs adatszerkezet felépítését, a konfigurációs paraméterek típusait majd ezeknek a konfigurációknak a generálását. [2]

6.1 Elméleti áttekintés

Az AUTOSAR szabvány a modulok megvalósítását egy kézzel írt (statikus) és egy konfigurációtól függő (dinamikus) részre bontással javasolja. A dinamikus rész konfigurációs paramétereit három csoportra lehet bontani aszerint, hogy a fejlesztési folyamat melyik pontjában kapnak értéket.

Pre-compile time

A konfigurációs paramétereket az előfordító dolgozza fel, így ezeknek még a fordítás előtt mindenképpen értéket kell kapniuk, hogy lehetőség nyíljon a segítségükkel a modulok forráskódjának a módosítására. Ebbe a csoportba sorolhatók a kódoptimalizálásért felelős paraméterek, mint például egy funkcionalitást engedélyező vagy tiltó makró, vagy a maximális tömb méretek meghatározása.

Link time

Ebben a csoportban a paraméter értéke még a fordítás után megváltoztatható, de a linkelési fázis előtt mindenképpen rögzülnie kell, mivel a linker helyettesíti be a megfelelő objektumokra való hivatkozást. A segítségükkel adatszerkezeteket generálhatunk, melyekre később a statikus forrásokból extern deklarációval hivatkozhatunk.

Post-build time

A paraméter értéke a build folyamat után is szabadon megváltoztatható, a konfigurációt a modul futásidőben kapja meg. Ezeknek a paramétereknek a memóriában küldön rész van lefoglalva, amikbe feltölik őket, és használat során erre a memóriaterületre mutató pointerekkel éri el a modul. Előnyük, hogy változás esetén nem szükséges a modul újrafordítása.

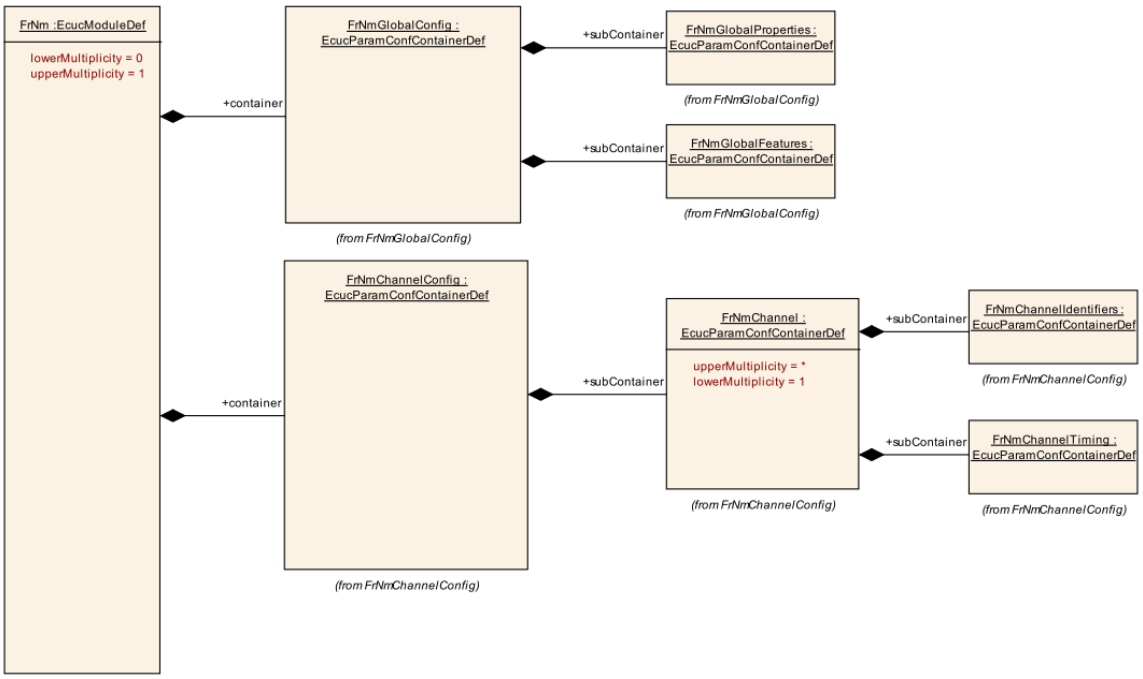
Konfigurációs osztályok

A konfigurációs paraméterek definiáláskor három konfigurációs osztályba kerülhetnek aszerint hogy azokat hogyan konfigurálják:

- VARIANT-PRE-COMPILE: csak *Pre-compile time* tulajdonságú paraméterek kerühetnek ebbe a csoportba
- VARIANT-LINK-TIME: csak *Pre-compile time* és *Link time* tulajdonságú paraméterek kerülhetnek ebbe a csoportba.
- VARIANT-POST-BUILD: a paraméterek mindhárom konfigurációs csoportba kerülhetnek

6.2 Adatszerkezet

A BSW modulok szabványdokumentumaiban a 10. fejezet tartalmazza a konfigurációs adatokat. Ezeket az adatokat konfigurációs paramétereknek nevezi a szabvány, amiket csoportosítva konténerekbe sorol. Például a FlexRay Network Management modul konfigurációs struktúrája a következő:



6.1. ábra FrNm konfigurációs konténerei és azok kapcsolata [15]

A konfigurációs paraméterek tulajdonságait egy táblázat reprezentálja.

SWS Item	**ECUC_FrNm_00030 :**		
Name	FrNmRepeatMessageTime		
Description	Timeout for Repeat Message State. Defines the time in seconds how long the NM shall stay in the Repeat Message State. The value "0" denotes that no Repeat Message State is configured, which means that Repeat Message State is transient and implies that it is left immediately after entry and consequently no startup stability is guaranteed and no node detection procedure is possible.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. 65.535]		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Timing value shall be an integer multiple of the time for one repetition cycle.		

6.2. ábra FrNmRepeatMessageTime konfigurációs paraméterének ének a definíciója [15]

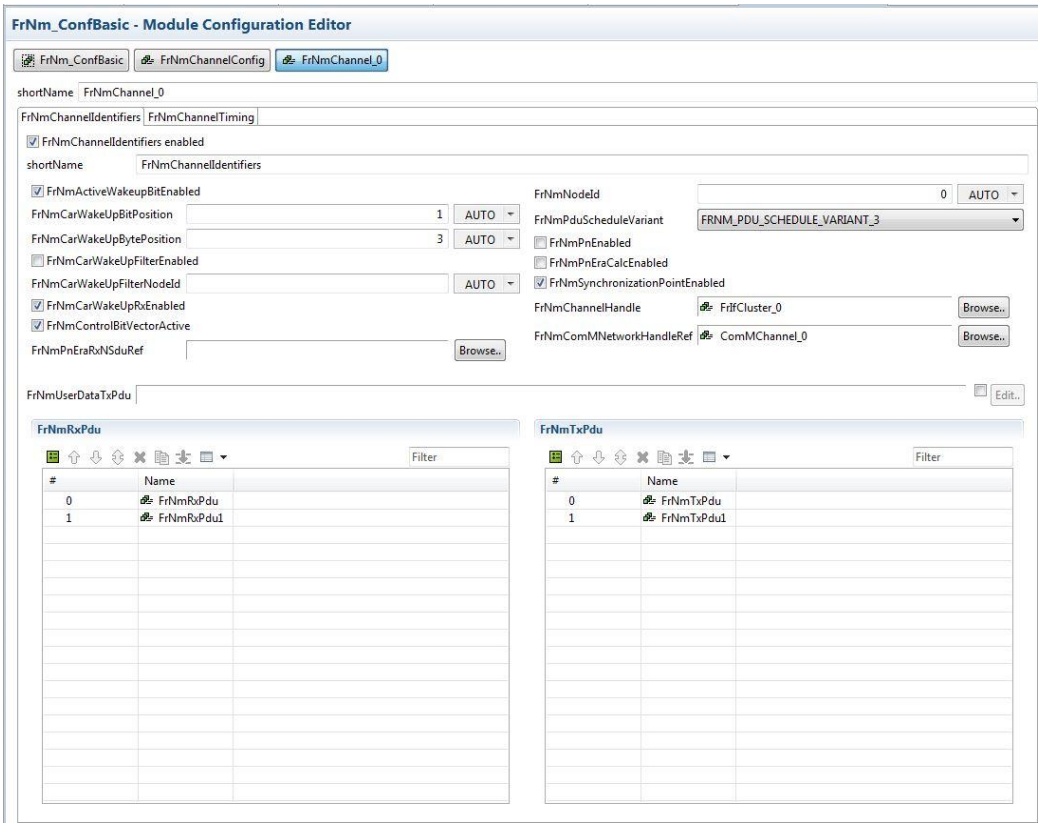
Egy ilyel táblázat magában foglalja:

- Paraméter leírását, definícióját.
- Multiplicitását, ami konfigurációs paraméterek estén általában azt írja elő hogy szükséges-e deklarálni a paramétert.
- Típusát, ami a megszokott változó típusok (integer, boolean, float) mellett lehet referencia is egy másik modul konfigurációs paraméterére.
- Paraméter határértékei.
- Alapmérezetett értékét.
- Konfigurációs osztályát.
- Más paraméterektől való függőségét (például ha egy adott paraméter értéke FALSE, akkor kiértékelendő paraméternek TRUE-nak kell lennie)

6.3 Konfiguráció generátor

Rendszertervezésnél a szabvány szerint megadott konfigurációs paramétereknek értéket kell kapniuk amiben a rendszertervezőnek a konfiguráció generátor segít. Ez a generátor a thyssenkrupp Presta Hungary Kft. által fejlesztett eclipse alapú AUTOSAR Architect programot egészíti ki. Tervezéskor a programban rendelkezésre áll egy

grafikus felület, ami a paramétereknek való értékadást segíti. Ez az AUTOSAR grafikus modellezési tulajdonsága miatt valósulhat meg. Generáláskor a feladat az implementált adatmodellnek és a konfigurációs beállításoknak megfelelően létrehozni a konfigurációs fileokat.



6.3. ábra FrNm konfiguráció generátor grafikus környezete

A modulok eddigi kódgenerátora Apache Velocity kódszintetizáló technológián alapul. Ez a generálás sablonosan történik: egy előre megírt kódkörnyezetbe helyettesíti be a megfelelő konfigurációs változókat a generátor az AUTOSAR Architect-ben beállítottak alapján. A forráskód egy ilyen generátorban hasonló ha például egy előfordítási macro elkészítése esetén:

```
#define FRNM_COM_USER_DATA_SUPPORT
#if($assertionHelper.assertBoolean($ecuc4,$gcfg,
”FrNmGlobalFeatures/FrNmPassiveModeEnabled”))TRUE#{else}FALSE#end
```

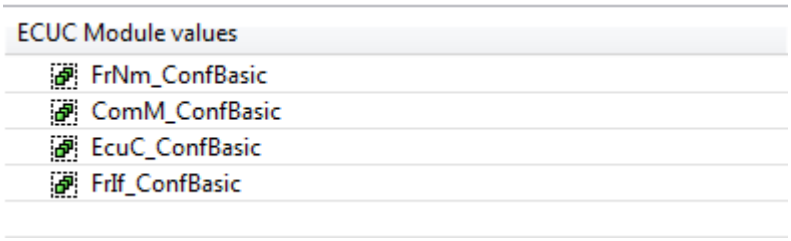
Generálás után, ha a modul passzív módban működik a következő sor jelenik meg a forrásfileban:

```
#define FRNM_COM_USER_DATA_SUPPORT FALSE;
```

Ez a módszer azért tekinthető kényelmetlennek a konfiguráció generáláshoz, mivel a konfigurációs beállítások konzisztenciájának az ellenőrzése nehezebben oldható meg és kevésbé összefüggő mint például egy JAVA nyelven alapuló kódgenerátor. A kódgeneráláshoz először az AUTSAR Architectben megalkotott modell-t kell bejárni, kigyűjteni az ott definiált konfigurációs változókat. Ezeket miután kigyűjtöttük fel kell dolgozni, ellenőrizni kell hogy a szabványnak megfelelően lettek-e beállítva a rendszertervező által. Legvégül miután bebizonyosodott hogy az adatszerkezet megfelel lehet legenerálni a kódot. A kódgenerálást ezek szerint három fő lépésre lehet bontani, amiket a következőkben részletezek.

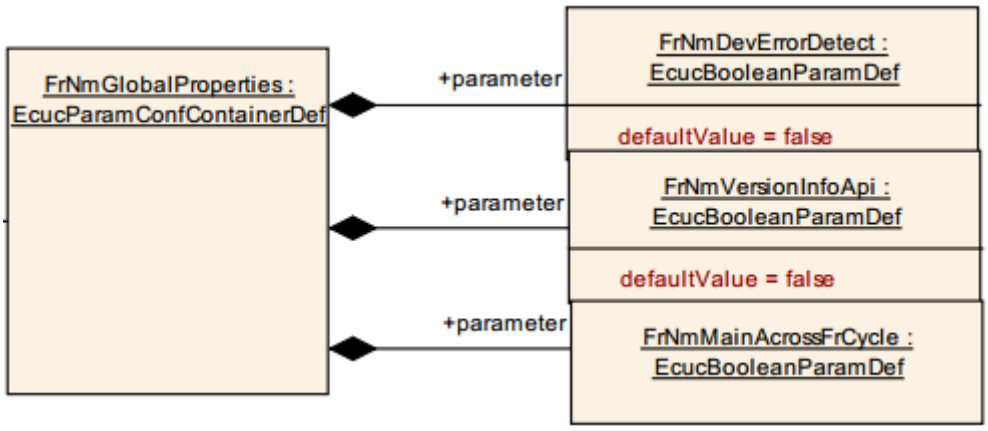
Adatok importálása a szabványnak megfelelően osztálystruktúrákba.

Első feladat, hogy a modulnak megfelelően létrehozzuk a konfigurációs konténereknek megfelelő osztálystruktúrákat. Ezekbe az osztályokba olyan változókat kell deklarálni, amibe beolvashatjuk a konténer konfigurációs paramétereit. A konfiguráció generálásához szükséges modulok paraméterei egy EcucVauleCollection-ben foglalnak helyet, innen kell őket kiolvasni kódgenerálás során.

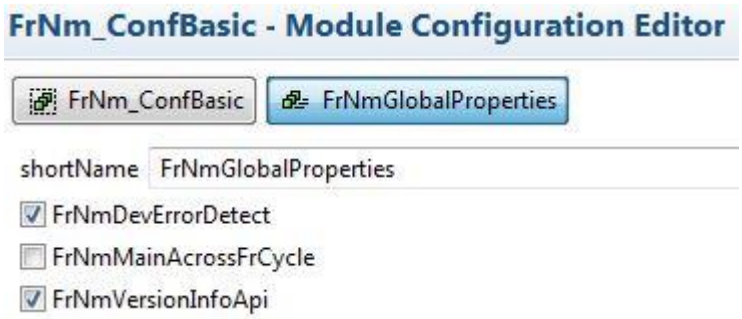


6.4. ábra Példa a konfigrációs EcucValueCollection-re

A FlexRay Network Management modul Global Properties konténerének a megvalósítása eszerint:



6.5. ábra FrNm GlobalProperties konténere és annak paraméterei [15]



6.6. ábra FrNmGlobalProperties paraméterei beállítását segítő grafikus felület

```
public class FrNmGlobalProperties{

    private String name;
    private Boolean frNmDevErrorDetect;
    private Boolean frNmMainAcrossFrCycle;
    private Boolean frNmVersionInfoApi;

    public FrNmGlobalProperties(EcucContainerValueElement container){

        name = container.name();
        frNmDevErrorDetect = container.bool("FrNmDevErrorDetect");
        frNmMainAcrossFrCycle =
container.bool("FrNmMainAcrossFrCycle");
        frNmVersionInfoApi = container.bool("FrNmVersionInfoApi");
    }

    public String getName() {
        return name;
    }

    public Boolean getFrNmDevErrorDetect(){
        return frNmDevErrorDetect;
    }

    public Boolean getFrNmMainAcrossFrCycle(){
        return frNmMainAcrossFrCycle;
    }

    public Boolean getFrNmVersionInfoApi(){
        return frNmVersionInfoApi;
    }
}
```

A container osztály bool() metódusának segítségével lehet megkeresni a konfigurációs paraméterek közül azt a változót, amit a függvényben átadott string argumentum definiál. Ez a metódus a talált értéket egy Boolean értékként adja vissza.

Egyes moduloknál megeshet, hogy vannak olyan paraméterei, melyek egy másik modul konfigurációs paraméterét veszik át referenciaként. Az ilyen esetekben ezeket a kiegészítő modulokat definiálni kell az *EcucValueCollection*-ben, beállítani a megfelelő konfigurációt, és generálás során kiemelni a paramétert a megfelelő helyre. Az FrNm

esetén ilyen modul a PDU Router, ahonnan a megfelelő PDU-k hosszára van a kódgenerálás során szükség.

Az importált adatok konzisztenciájának az ellenőrzése: *model checking*

A generált paraméterket beolvasás után és a kódfileok generálása előtt ellenőrizni kell, hogy megfelelnek-e a szabvány előírásainak. Ezeket az előírásokat részben a paraméter definícióknál azok függőségei tartalmazzák. Példa ilyenre az *FrNmCoordinatorSyncSupport* konfigurációs paraméter, aminek FALSE értéket kell kapnia ha az *FrNmPassiveModeEnabled* paraméter TRUE.

SWS Item	**ECUC_FrNm_00081 :**		
Name	FrNmCoordinatorSyncSupport		
Description	Enables/disables the coordinator synchronization support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: FrNmCoordinatorSyncSupport has to be set to FALSE if FrNmPassiveModeEnabled is set to TRUE.		

6.7. ábra FrNmCoordinatorSyncSupport konfigurációs paraméter definíciója [15]

Itt kell elennőrizni azokat a paramétereket és konténereket amik rendelkeznek multiplicitási vagy határértéki megkötésekkell. Ilyenre példa az FrNmTxPdu konténer, amiből a felhasználó a specikifikáció szerint maximum 4-et hozhat létre, vagy az *FrNmRepeatMessageTime* paraméter, aminek az értéke csak 0 és 65.535 közé eshet.

Ezt a funkciót egy külön *FrNmModelCheck* osztályba implementáltam, ami meghívásra kerül a generáció során, és csak akkor folytódik, ha minden konzisztencia ellenőrzési metódus hiba nélkül teljesül. Hiba esetén a generálás félbeszakad, és a hibának megfelelően értesül a felhasználó a hiba okáról, valamint a paraméterekről. Az FrNmTxPdu konténerek számának az ellenőrzésére szolgáló metódus a következő képpen néz ki: minden csatornát végigjár a függvény és leellenőrzi a konténerekben található FrNmTx bufferek számát. Ha ez nagyobb mint 4, akkor jelenti a hibát az átadott multiStatus argumentumnak.

```
private static void checkTxBufferMultiplicity(FrNmConfig parsedConfig, MultiStatus
multiStatus){

    for(FrNmChannelConfig channelConfig : parsedConfig.getFrNmChannelList()){
        for(FrNmChannel channel : channelConfig.getFrNmChannelList){

            if (4 < channel.getFrNmChannelIdentifiers().getFrNmRxPduList().size()){
                multiStatus.add(new Status(IStatus.ERROR, Activator.PLUGIN_ID),"/n
The multiplicity of FrNmTxPdu container shall be 0...4 per channel. Check: "+
channel.getName()));
            }
        }
    }
}
```

Szöveges fileok generálása.

Az utolsó lépés a szöveges fileok elkészítése. Ehhez a generátor létrehozza a szükséges header és source fileokat. Miután elkészültek, egy string-be kerülnek bele a modul konfigurációs paraméterei a statikus adatszerkezetének megfelelően. Ehhez cég által elkészített kódgenerációs segédfüggvényeket használtam fel amikkel gyorsabban történt a file generátor megírása. Ez a formázott string kerül bele a generálás végén a source és header fileokba.

7 Modulok tesztelése

Az implementációm tesztelésére rendelkezésemre álltak tesztkörnyezetek és tesztkészletek, amikkel ellenőrizhettem hogy a moduljaim elvárásoknak megfelelően működik. Tesztelés történhet rendszer szinten (System Test), egyes modulok összekapcsolásával (Integration Test), egy-egy modulra vonatkoztatva (Component Test) és a felhasználó/megrendelő által a végterméken végzett fekete doboz teszt, amivel azt vizsgáljuk hogy a megrendelő számára megfelel-e a működés (Acceptance Test).

A szoftver tesztelése során kódfedettségi mérőszámok segítségítenek a tesztelés mérésével [19]:

Utasítási lefedettség (Statement Coverage): a tesztkészlet által kipróbált futtatható utasítások százaléka

Elágazás lefedettség (Branch Coverage): a tesztkészlet által meghívott elágazások sázalékos aránya.

Feltétel lefedettség (Condition Coverage): A 100%-os elágazási feltétel lefedettség azt jelenti, hogy minden döntési utasításban minden egyes feltétel IGAZ, illetve HAMIS ága tesztelve van.

Döntési feltétel lefedettség (Decision Condition Coverage): a tesztkészlet végrehajtása során az összes feltétel eredmény és döntési eredmény meghívásának százalékos aránya.

Módosított döntési lefedettség (Modified Condition Decision Coverage): olyan arányszám, amely azt mutatja, hogy a tesztkészlet hány százalékban hívta meg az egyes feltétel kimeneteket, amelyek egymástól függetlenül befolyásolják a döntés eredményét.

7.1 Komponens tesztelés

Egy AUTOSAR alapszoftver modul tesztelését először mint önálló egységet kell letesztelni. Ennek a célja nem az hogy a megfelelő működést bizonyítsuk, hanem az esetleges hibás működés észlelése a rendszerszintű tesztelés előtt. A megvalósítását ennek a tesztelésnek fekete doboz (black box) tesztelésnek hívják, mivel a modul belső

viselkedését nem, csak a kívülről látható viselkedését vizsgálja, ami a modul API függvényein keresztül valósul meg.

A modul teszteléséhez szükséges egy driver egység, ami a komponens felett elhelyezkedő modulok vezérlését és/vagy felhívását végzi. Ők adják meg a teszteléshez szükséges paramétereket amikkel futtatjuk azokat. Ezen kívül csonkokra (stub-ok) is szükség van: ezek valósítják meg a modul által hívott függvényeket. Segítségükkel tudjuk a tesztek helyességét megvizsgálni.

A komponenstesztelés, amivel foglalkoztam a CUnit függvénykönyvtár segítségével lett megvalósítva, ami tesztelést segítő adminisztráló és összehasonlító (egyenlőségvizsgálatot végző) funkcionalitással rendelkezik. Egy CUnit teszt több tesztkészlettel, úgynevezett *Test Suit*-tal rendelkezik, melyek külön teszteseteket tartalmaznak. Tesztelés során ez a tesztkörnyezet képes a kódlefedettségi mérőszámok kiszámítására.

A tesztelést megvalósító 4.0-s verziójú tesztkönyvtárak a rendelkezésemre álltak, segítségükkel ellenőrizhettem az általam módosított modulokat. Ezek a modul-beli változások miatt csak részben futottak le sikeresen. Az elbukó teszteket ki tudtam javítani a szabványban történt módosítások implementálásával, hogy lássam jó-e a modul megvalósítása.

Először a konfigurációkat készítettem el a modulokhoz. Minden modulhoz több konfigurációs beállítás tartozik, melyek segítségével ellenőrizni lehet annak egy-egy funkcióját. Ilyen például a CAN Network Management passzív és a teljes kommunikációs üzemmódja. Miután ezek elkészültek, összefordítottam a modulokat a hozzájuk tartozó tesztkönyvtárral és lefutattam a teszteket. Ezt követően megvizsgáltam azok közül melyik esetek buktak el, és milyen hibával.

Tesztelés során a stub fileokat ki kellett egészítenem a 4.3-as verzióba került új API függvényekkel. Ezeknek a felépítése a többi függvénycsonk felépítéséhez hasonló: a tesztelés adatszerkezetében létrehoztam egy számlálót, amit a függvénycsonk meghívása esetén egyel léptetek felfelé, és jelzem a tesztkörnyezetnek a függvény hívását. Már meglévő függvénycsonkokat is módosítanom kellett, például amikor egy meglévő API argumentumlistája kiegészült egy új változóval. Itt is a tesztkörnyezet adatszerkezetét és a megfelelő függvénycsonkot kellett módosítanom az előző példához hasonlóan.

A drivereket akkor kellett módosítanom, amikor a modul működésében történt változás. Ilyen például amikor egy API hívása során megváltozott hogy melyik állapotot kell felvennie, vagy amikor egy olyan függvény hívódik meg, ami a 4.0-ban nem. Ilyenkor a módosításhoz szükséges segédfüggvények a rendelkezésemre álltak, ezért csak meg kellett hívnom őket a megfelelő helyeken, vagy a már meglévő segédfüggvények argumentumlistáját kellett megváltoztatnom.

Ezzel az általam módosított 4.0-s verziójú komponensteszteléssel csak saját magamnak tudtam leellenőrizni hogy jól implementáltam a módosításokat. A cég követelményeinek az ilyen tesztelés csak akkor felel meg, amikor azt egy független tesztelő készíti el.

7.2 Interoperability testing

Miután meggyőződtem hogy a moduljaim sikeresen átmentek a módosított komponens teszteken, áttértem egy rendszer szintű, gyártótól kapott tesztkörnyezetet felhasználó tesztelési módszerre amit interoperability (együttműködő képességi) tesztnek hívunk. Ezzel azt ellenőrizzük, hogy a modulok megvalósítása kompatibilis-e a BSW többi moduljával és a funkcionalitása is megfelel a gyártó előírásainak.

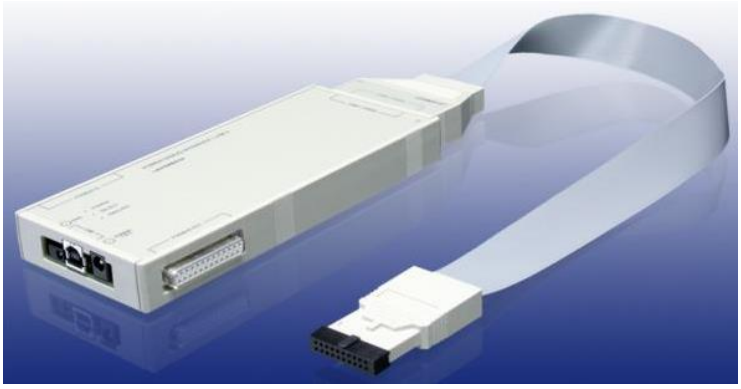
7.2.1 Az Interoperability tesztelés összeállítása

Az Interoperability tesztelés első lépése a teszteléshez rendelkezésre álló szoftverkomponensek, ECU kivonatok valamint egyéb elemek integrálása a BSW modulokkal, generált RTE réteggel, operációs rendszerrel és konfigurációs fileokkal. Az eredménye ennek egy integrált kódkészlet, amin el lehet végezni a tesztelést.

A tesztelést egy cég által fejlesztett AUTOSAR evaluation board-on végeztem, amin egy power pc architektúrájú 32-bites mikrokontroller futtatja a tesztelendő kódot. Ehhez a panelhez kommunikációs protokolloknak megfelelő jelszintillesztő kiegészítő egységek is tartoznak, amik segítségével kommunikációs hálózatokhoz kapcsolható a panel.

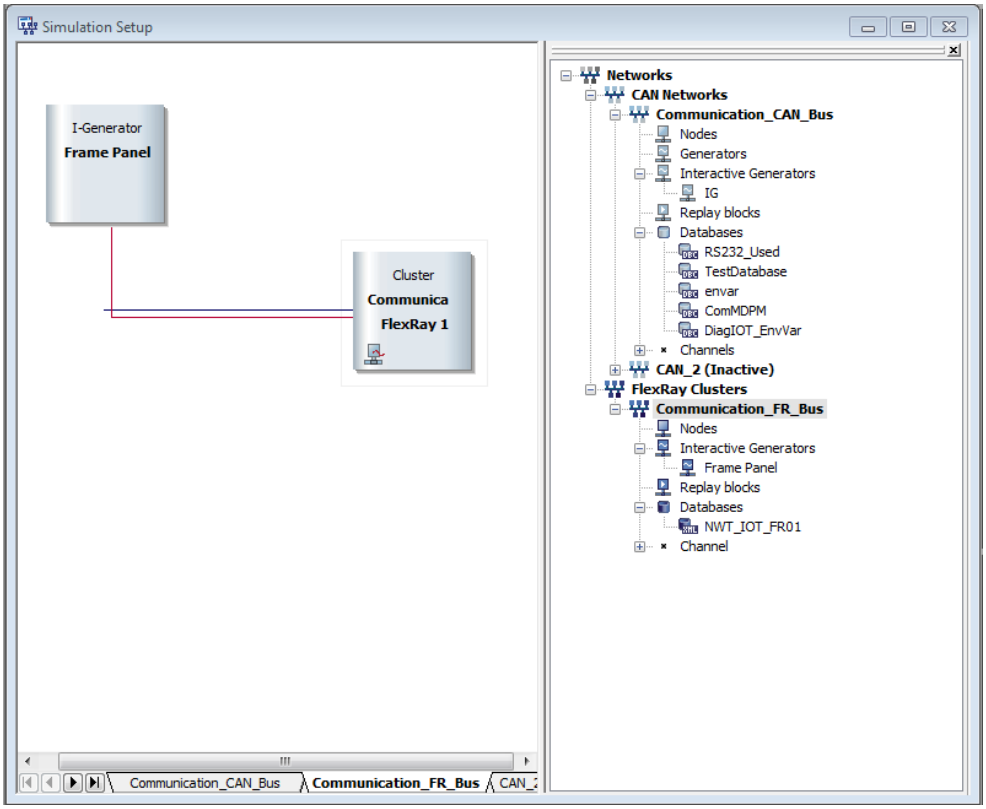
A tesztelés során a hibakeresés egy Lauterbach Power Debug Interface egység segítségével történik, amit a PC oldalon a TRACE32 szoftver segítségével használtam. Ez a hardver a fejlesztőkártyán a mikrokontroller debug interfészéhez kapcsolódik egy JTAG kapcsolaton keresztül, aminek a segítségével tesztelés során megfigyelhető és írható a mikrokontroller memóriája, regiszterei. Ezen kívül utasításról utasításra

végigjárható a kód. A teszt futása során töréspontok is elhelyezhetők a forráskódban, amit elérve a futás megszakad. [20]



7.1 ábra Lauterbach Power Debug Interface [20]

A teszt scriptek futásáért, valamint az ECU-k szimulációjáért egy másik szoftver, a CANoe felel. Ennek a PC oldali programnak a segítségével futtathatóak a rendelkezésemre álló tesztek, itt kell konfigurálni a CAN hálózatokat és a FlexRay klasztereket, valamint ugyanitt lehet elindítani a tesztelés folyamatát.



7.2. ábra CANoe szimulációs beállítási felülete

7.2.2 Mérés menete

A tesztelést azzal kezdtem, hogy a szakdolgozat során elkészített moduljaim összefordítottam az interoperability tesztkönyvtár többi elemével. Amikor sikeresen összefordult minden, a mérési elrendezés összeállításával folytattam a mérést. Illesztettem a fejlesztőpanelhez a szükséges hardverelemeket, valamint a Lauterbach debuggert. Mindezek után felkonfiguráltam a TRACE32 szoftvert a kiadott batch file segítségével, és feltöltöttem a megfelelő tesztet a CANoe programba.

A tesztek egy .dat kiterjesztésű file-ban találhatóak és tesztesetekre vannak bontva. Modul funkcionalitásában történő változás esetén ezeket kell módosítani, például ha egy állapotváltásnál változik hogy melyik állapotot kell felvennie a modulnak. Egy teszteset több *Test Record*–ból épül fel. A *Test Record*-ok egy teszteset futásának a lépései, meghívják a teszteléshez szükséges funkciókat (például egy API-t) és ellenőrzik hogy a rendszer az elvártnak megfelelően működik (megvizsgálja a rendszertől kapott választ és összehasonlítja a várt eredménnyel). Egy ilyen Record tartalmazza az utasítást rendszer számára, és a várt választ.

A tesztelés eredményét, ami alapján látni lehet hogy hiba esetén hol bukott el, egy .xml kiterjesztésű fileba menti a tesztkörnyezet. Ez tartalmazza, a Recordok, valamint a kiküldött és beérkezett adatok időzítését. Segítségükkel ki lehet szűrni, hogy hol, és milyen hiba esett a tesztben. Ilyen hiba lehet, amikor a rendszer rossz adatokkal tér vissza, vagy ezek időzítése nem megfelelő.

[-] **4 Test Case Init4: Init4: Passed**

Test case begin: 2017-11-27 09:55:43 (logging timestamp 44.041023)
Test case end: 2017-11-27 09:55:45 (logging timestamp 46.065523)

Main Part of Test Case

Timestamp	Test Step	Description	Result
44.041023	RECORD4	Enable all NM Indications	-
44.041023	1.	TARGET_TEST	-
44.041023	1.1.	CLEAR_TARGET_TEST_RESULT	-
44.041023	1.2.	SEND_VALUE	-
44.041023		tx cmd 0x62 0x01 (t= 44.0410, rs232, ecu 0) data=00 00 00 00	-
44.042000		RS232 transmission finished (t= 44.0420, dt=0.980ms, ecu 0)	-
44.042000	1.3.	WAIT_TARGET_TEST_END	-
44.065523		RS232 rx (t= 44.0655, ECU=0): DLC: 5, cmd: 0x62, sub cmd 0x01 data=00 00 00	-
44.065523	1.3.	target test step passed	pass

7.3. ábra Egy teszteset futásának az eredménye

8 Összefoglalás

8.1 Értékelés

A dolgozatomban a feladatom három AUTOSAR hálózatmenedzselési modul, Network Management Interface, CAN Network Management, FlexRay Network Management 4.0 szabványverziójának a módosítása volt úgy, hogy azok megfeleljenek az új 4.3-as szabványnak. A munkám az összes olyan modul szabványdokumentumának a tanulmányozásával kezdődött, melyek elengedhetetlenek voltak az AUTOSAR kommunikációs és hálózatmenedzselési működések megértésére. Ezek alatt az AUTOSAR kommunikációs blokkjában szereplő modulokra gondolok, mint például a PDU Router és a kommunikációs protokoll specifikus interfészek. Ezt követően a hálózatmenedzsment modulok 4.0-s és 4.3-as verzióit összehasonlítottam, kigyűjtöttem a változásokat és klasszifikáltam őket.

Ez után bemutattam az AUTOSAR szoftverarchitektúra elméleti hátterét, kiemelve a kommunikációs réteget és annak a hálózatmenedzselési funkcióit. Részletesen kiemeltem a három hálózatmenedzselési modul működését és funkcióját, majd megvalósítottam azokat a változásokat amiket a klasszifikálás során implementálhatónak soroltam be.

Az implementáció után a modulok konfigurációjának szerepét és felépítését mutattam be az AUTOSAR szabványban, majd a konfiguráció elkészítését segítő konfiguráció generátorok megvalósítását szemléltettem. Itt kitértem az eddig használt sablonos kódgenerálási megoldásra, és bevezettem a cég számára előnyösebben felhasználható JAVA nyelven alapuló generátor elkészítésének a lépéseit.

Az implementáció és a megfelelő konfigurációs fileok elkészítése után elkezdtem a modulok tesztelését. Ehhez bemutattam a teszteléshez szükséges alapfogalmakat és mérőszámokat, majd két tesztelési módszerről írtam, amiket felhasználtam az általam megvalósított modulok működésének az ellenőrzéséhez. Az egyik ilyen tesztelés a komponens alapú *black box* tesztelés, amiben csak a modul kívülről látszó működését lehet ellenőrizni. Ehhez a modulok 4.0-s verziójához készült tesztkörnyezeteket használtam fel úgy, hogy módosítottam azokat az elemeket, amik megváltoztak a modul működésében. Ezzel tudtam magamnak igazolni a modulok

helyes működését. A másik módszer az Interoperability tesztelés, amihez egy gyártó által készített tesztkörnyezet és teszt scriptek álltak a rendelkezésemre, Ezzel azt lehetett ellenőrizni, hogy a moduljaim helyesen működnek-e együtt a BSW többi moduljával. Az itt előkerülő hibákat, amik a verziók közti eltérések miatt adódtak javítanom kellett a forráskódokban.

8.2 Továbbfejlesztési lehetőségek

Az elkészült, immár a 4.3-as verziónak megfelelő működésű moduljaim működését komponens teszteléssel ellenőriztem. Ehhez a már rendelkezésemre álló 4.0-s verziójú teszkörnyezetetet használhattam fel, amit módosítottam hogy megbizonyosodjak a moduljaim helyes viselkedéséről. A működés hivatalos teszteléséhez egy független tesztkönyvtárat kell létrehozni, hogy az megfeleljen a cég által adott teszteléssel kapcsolatos előírásoknak. Ebben az esetben nem elég a régi verzió módosítása.

Az interoperability tesztelést az FrNm modulon végeztem el, mivel annak a tesztkörnyezetében történt a legtöbb módosítás a szabványverziók között. A jövőben a másik kettő modulra is szeretném elvégezni ezt a tesztelést, és kijavítani a verziók közti eltérésekből adódó hibákat.

Irodalomjegyzék

[1] AUTOSAR Consortium, Motivation and goals of AUTOSAR, https://www.autosar.org/about, 2017

[2] AUTOSAR Consortium, Layered Software Architecture 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf, 2016

[3] Faragó Dániel, AUTOSAR alapú autoipari szoftverrendszerek - Basic Software, communication, előadásjegyzet, 2017

[4] AUTOSAR Consortium, CAN Driver 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_CANDriver.pdf, 2016

[5] AUTOSAR Consortium, Specification of CAN Transceiver Driver 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_CANTransceiverDriver.pdf, 2016

[6] AUTOSAR Consortium, Specification of CAN Transport Layer 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_CANTransportLayer.pdf, 2016

[7] AUTOSAR Consortium, Specification of CAN State Manager 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_CANStateManager.pdf, 2016

[8] AUTOSAR Consortium, Specification of PDU Router 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_PDURouter.pdf, 2016

[9] AUTOSAR Consortium, Specification of IPDU Multiplexer 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_IPDUMultiplexer.pdf, 2016

[10] AUTOSAR Consortium, Specification of Communication 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_COM.pdf, 2016

[11] AUTOSAR Consortium, Specification of Diagnostic Communication Manager 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_DiagnosticCommunicationManager.pdf, 2016

[12] AUTOSAR Consortium, CAN Network Management 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_CANNetworkManagement.pdf, 2016

[13] AUTOSAR Consortium, Network Management Interface 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_NetworkManagementInterface.pdf, 2016

[14] AUTOSAR Consortium, Specification of COM Manager 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_COMManager.pdf, 2016

[15] AUTOSAR Consortium, FlexRay Network Management 4.3, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-3/AUTOSAR_SWS_FlexRayNetworkManagement.pdf, 2016

[16] Faragó Dániel, AUTOSAR FlexRay kommunikációs modulok megvalósítása, szakdolgozat, BME-VIK, 2012

[17] Hegyi Balázs, AUTOSAR hálózatmenedzsment modulok megvalósítása, szakdolgozat, BME-VIK, 2012

[18] AUTOSAR Consortium, Network Management Interface 4.0, szabványdokumentum, https://www.autosar.org/fileadmin/user_upload/STANDARDS/classic/4-0/AUTOSAR_SWS_FlexRayNetworkManagement.pdf, 2011

[19] Magyar Szoftvertesztelői Tanács Egyesület, Hungarian standard glossary of terms used in Software Testing, kifejezések gyűjteménye, http://www.masterfield.hu/docs/HTB_Glossary_3_13.pdf, 2014

[20] Lauterbach development tools, Debugger Basics – Training, használati útmutató, http://www.lauterbach.com/pdf/training_debugger.pdf, 2017