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

Sisak Gergely

AUTOSAR CAN KOMMUNIKÁCIÓS MODULOK MEGVALÓSÍTÁSA

KONZULENS

Dr. Pintér Gergely
(ThyssenKrupp Presta Hungary Kft.)

Dr. Sujbert László, docens

BUDAPEST, 2012

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2012. 12. 07.

...……

Sisak Gergely

Kivonat

Az AUTOSAR a modern gépjárművekben alkalmazott elektronikus vezérlőegységek (ECU-k) számára egységes szoftverarchitektúrát definiáló szabvány. Az architektúra része az az alapvető szolgáltatásstruktúra (Basic Software réteg - BSW), amely eltakarja a hardver sajátosságait és ezzel támogatja az alkalmazási szoftver hordozhatóságát. A BSW réteg modulokból áll, amelyek egymástól világosan elkülöníthető funkcionalitást hordoznak. A dolgozat célja a kommunikációért felelős modulok alapvető funkcióinak bemutatása mellett a CAN protokollhoz kapcsolódó szoftverrétegbe tartozó két modul megvalósításának bemutatása.

Az ISO 15765 szabványon alapuló CAN Transport Layer (CanTp) biztosítja a keretformátumon túlmutató méretű adatok átvitelét a CAN hálózaton. Üzenetküldéskor az egy adatkeretben nem elküldhető üzenetek szegmentálását, fogadáskor a beérkező keretek újraegyesítését végzi. A CAN State Manager (CanSM) felelős a CAN hálózat hardver-egységeinek ki- és bekapcsolásáért. A kontrollerekre és transceiverekre jellemző működési állapotokat kezelő és az állapotok közt fellépő megfelelő tranzienseket biztosító állapotgépet valósítja meg.

Szakdolgozatom a CAN szabvány bemutatásával kezdődik, ezt az AUTOSAR szabvány alapvető jellemzőit összefoglaló és a kommunikációs stack működését ismertető részek követik. A modulok megvalósításával kapcsolatos leírások és megfontolások képezik a dolgozat további részeit. A CanTp modul esetén kitérek az ISO szabvány ismertetésére. Ezt követően bemutatom az általam tervezett adatfogadási és küldési algoritmusokat, valamint röviden megvizsgálom a modul erőforrásigényének alakulását különböző – résztevékenységeket ellátó – algoritmusok esetén. A modul tesztelésének bemutatása magába foglalja a tesztkörnyezet kialakításának és a tesztelés mérőszámainak leírását is. A CanSM modulról szóló rész ismerteti az állapottérképek megvalósításának legalapvetőbb módszereit, majd ezekből kiindulva igazolja az általam választott megoldást. Ezt követően röviden bemutatásra kerül a modul erőforrás használata, valamint a teszteléssel kapcsolatos legfontosabb megfontolások.

Abstract

AUTOSAR is an international standard defining a unified software architecture for Electronic Control Units (ECUs) of modern vehicles. The Basic Software Layer (BSW), one of its main layers provides hardware abstraction for the application software components and ensures their portability. The BSW consists of modules with different functionality. This BSc thesis offers a functional overview about the BSW communication stack and describes the implementation of two modules of the CAN stack: the CAN Transport Layer (CanTp) and the CAN State Manager (CanSM).

The main purpose of the CanTp module is to segment and reassemble CAN messages longer than 8 bytes. The module is based on the international standard ISO 15765. The CanSM module is responsible for the control flow abstraction of CAN networks. It realizes the state machine controlling the states representing different operation modes of the CAN controllers and transceivers and the transitions between them.

The first part of my thesis introduces the CAN standard, followed by a section about AUTOSAR. After the basic description of the standard, this part focuses on the BSW communication stack. The subsequent chapters summarize the development and testing of the modules. The ISO standard is shortly introduced at the beginning of the part about CAN Transport Layer followed by the description about the development of the transmission and reception algorithms considering the optimal use of resources. The next part gives a short theoretical summary about testing and shows how the CanTp module has been tested illustrated by an example. The chapter about CAN State Manager describes the basic methods for implementing state charts in general, and verifies my solution for the implementation. This part also deals with the resource use and testing of the CanSM module.

1 Bevezetés

1.1 Témaválasztás

A ThyssenKrupp Presta Hungary Kft-nél töltött gyakornoki munkám során kerültem kapcsolatba a cég előfejlesztési osztályával (Advanced Development), ahol egyes AUTOSAR modulok megvalósíthatóságának és testre szabhatóságának vizsgálata volt a feladatom. Ennek keretében nyílt lehetőségem BSW modulok írására (ld. 3.2.2), melyek közül kettő, a CAN stack részét képező modul (CAN Transport Layer és CAN State Manager) adja szakdolgozatom alapját.

1.2 A feladat értelmezése

A modulok megvalósítása mellett törekedtem az elméleti háttér ismeretanyagának elsajátítására is. Ennek megfelelően a dolgozat 2. fejezetében a CAN kommunikációs protokoll, 3. fejezetében az AUTOSAR szoftver-architektúra kerül bemutatásra. Utóbbiban külön kitérek a kommunikációs réteg, azon belül is a CAN stack működésére.

A dolgozat 4. és 5. fejezete a CAN Transport Layer és a CAN State Manager megvalósításáról szól. A modulok tervezésének fontos és sok időt felemésztő része a hozzájuk kapcsolódó szabványok elolvasása és értelmezése. A specifikációk pontos leírást adnak a megvalósítandó modul működését illető elvárásokról, de szabad kezet adnak az implementációs döntések meghozatala kapcsán. Ezért a vonatkozó AUTOSAR és ISO szabványok áttanulmányozása mellett fontosnak tartottam a megvalósításra kínálkozó lehetőségek (pl. a CAN State Manager modul esetében az állapottérképek megvalósítási módjainak) megismerését is. A modulok erőforrásigényének vizsgálata, és az ebből következő döntések meghozatala is hozzájárul ahhoz, hogy hatékonyabb kód szülessen. A tesztelés a szoftverfejlesztés megkerülhetetlen állomása. A modulok megfelelő működésének bizonyítása sokszor több időt vesz igénybe, mint maga a megvalósítás. A dolgozat röviden bemutatja a modulok hosszú tesztelési folyamatának első állomásaként általam megírt teszteket is.

2 A CAN kommunikációs protokoll

2.1 Bevezetés

Az autóiparban alkalmazott elektronikai berendezések gyors ütemű fejlődésével és széleskörű elterjedésével párhuzamosan szükségessé vált a járművekben található ECU-k (Electronic Control Unit) közötti szoros együttműködés. A kezdetben mindössze néhány vezetéket jelentő pont-pont alapú kommunikációs kapcsolat egy ideig még megfelelt a fejlődő iparág követelményinek, ám később igény támadt egy rendszerezett, lehetőleg busztopológiájú, egyszerű, olcsó és biztonságos kommunikációs rendszer kifejlesztésére. A felmerülő problémára kínált megoldást a Robert Bosch GmbH által megalkotott CAN (Controller Area Network) protokoll.

A CAN-re soros aszinkron adatkommunikáció jellemző. A hálózat általában busztopológiát követ, ahol az adatok küldése üzenetszórásos (broadcasting) jellegű, vagyis a hálózathoz csatlakozó egységek mindegyike veszi az éppen küldő egység üzenetét. A buszon egyszerre több master egység jelenléte is megengedett (multimaster), ezek a busz használatáért ütközésmentes arbitrációval versengenek (CSMA/CA). Az adatátvitel sebessége (125kbit/s-1Mbit/s) és a hálózat kiterjedése (30-500m) a felhasználás céljától függően változik.

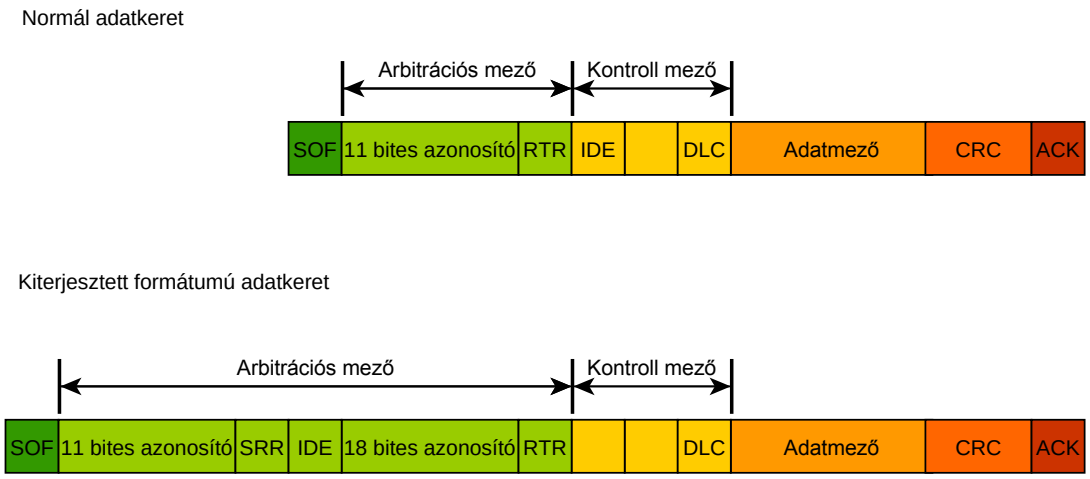
A CAN 2.0-ás szabvány (ISO 11898) 1991-es megjelenése óta a leggyakrabban alkalmazott autóipari kommunikációs technológiává vált. A CAN alkalmazásoknak mégis csak a harmada köthető az autóiparhoz. Előszeretettel használják orvosi műszerekben, ipari berendezésekben és más beágyazott alkalmazásokban is. A protokoll alapján további fejlesztések láttak napvilágot. Ilyen a CANopen, amely egy CAN alapokon nyugvó, szabványosított, beágyazott irányítórendszerek magasabb rétegeit leíró nemzetközi szabvány. A TTCAN (time-triggered CAN) protokoll idővezérelt kommunikációt tesz lehetővé úgy, hogy az eredeti CAN hálózat csak minimálisan változik.

2.2 Keretformátumok

Az adatok átvitele keretek (frame) küldésével, illetve fogadásával történik. A változó hosszúságú keretek 0-8 bájt adatot tartalmazhatnak [1].

2.2.1 Egy keret felépítése

A szabvány négy kerettípust definiál. Ennek megfelelően megkülönböztetünk *adatkeretet* (Data Frame), *hibakeretet* (Error Frame), *távoli keretet* (Remote Frame) *és túlcsordulás keretet* (Overload Frame). Az adatkeret lehet *normál*- (Standard) és *kiterjesztett* (Extended) *formátumú*. Utóbbit a szabvány 2.0-ás verziójában definiálták.



2.1. ábra: CAN keretek

Egy adatkeret felépítése:

- SOF (Start of frame): domináns bit, amely a keret kezdetét jelöli.
- azonosító mező (11 vagy 29 bit hosszú)
- RTR (Remote Transmission Request): ez a bit szolgál a kommunikáció kezdeményezésére (Remote Frame).
- SRR (Substitute Remote Request): csak kiterjesztett formátum esetén létezik, mindig recesszív értékű.
- IDE (ID Extension): domináns értéke standard, míg recesszív értéke kiterjesztett adatkeretet jelent.
- DLC (Data Length Code): az adatmező hosszát jelöli (0-8).
- adatmező: a ténylegesen továbbítandó adatok bájtjai (0-8), méretét a DLC írja le.
- CRC kód: 15 bites ellenőrző kód, ezt egy recesszív határoló (delimiter) bit követi.

- ACK (Acknowledge): nyugtázó bit. Az adó recesszíven hagyja, és várja, hogy valamelyik vevő domináns értéket adjon neki, ezzel jelezve, hogy az adott üzenet CRC-helyesen célba érkezett. Ezt is recesszív delimiter követi.
- A keret végét további 7 recesszív bit jelzi, amelyet 3 recesszív bites szünet (intermission) követ. Új adás a buszon csak ez után lehetséges.

2.2.2 Normál adatkeret (Standard Data Frame)

Az általános keretformátumnak megfelelően az adatkeret SOF bittel kezdődik, melyet egy 11 bites azonosító mező (Identifier) követ. Az ezután elhelyezkedő Remote Transfer Request (RTR) bit különbözteti meg, hogy adatkeretről vagy távoli keretről van-e éppen szó: domináns értéke adat-, míg recesszív értéke távoli keretet jelöl. Az eddigi bitek alkotják az úgynevezett *arbitrációs mezőt*.

A következő mező a *kontroll mező* (Control Field). Első, IDE (Identifier Extension) bitjének domináns értéke jelzi, hogy a keret standard formátumú. Recesszív IDE esetén kiterjesztett kerettel van dolgunk. Az ezt követő bit minden esetben domináns értéket vesz fel. A mező maradék négy bitje alkotja a DLC-t (Data Length Code), amely a keretben szereplő adatbájtok számát jelzi, ez az érték 0 és 8 között mozoghat.

A kontrol mezőt az *adatmező* követi, ahol tehát a DLC-ben definiált mennyiségű adatbájt helyezkedik el. Az ezt követő *CRC-mező* 15 bites CRC ellenőrző összegből, valamint egy recesszív értékű határoló (delimiter) bitből áll.

Az ezután következő nyugtázó ACK (Acknowledge) bitnél a küldő nem hajtja meg aktívan a buszt, a küldött üzenet sikeres vételére vonatkozó nyugtázást a fogadó egységek tudják jelezni egy domináns bit elküldésével. A keret végét 7 recesszív bitet tartalmazó End-of-Frame mező jelzi.

2.2.3 Kiterjesztett formátumú adatkeret (Extended Data Frame)

A kiterjesztett formátumú keret azonosító mezője hosszabb a normál kereténél. A keret a szokásos SOF bittel kezdődik, melyet – a normál kerethez hasonlóan – 11 bites azonosító mező követ. Az SRR bit (Substitute Remote Request) az RTR bit helyén található, értéke a szabvány szerint recesszív. Ezt követi az IDE bit, amelynek recesszív értéke jelöli a kiterjesztett adatkeretet. A kiterjesztett azonosító (Extended Identifier) az ez után

következő 18 bit. A kiterjesztett adatkeret tehát összesen 29 bit hosszúságú azonosítóval rendelkezik. A többi mező alapvetően megegyezik a standard keretnél megismertekkel.

2.2.4 Távoli keret (Remote Frame)

A CAN buszon az adatküldés üzenetszórásos jellegű, ezért az üzenetek eljutnak minden egyes vevő egységhez. Az üzenetváltást távoli keret küldésével lehet kezdeményezni. Ez felépítésében nagyban hasonlít az adatkeretre, itt is lehetséges mind standard, mind kiterjesztett azonosító használata. Az RTR bit recesszív állapota jelzi, hogy távoli keretről van szó. A keret nem tartalmaz adatmezőt, mivel csak a kommunikáció kezdeményezésére szolgál. Ha a címzett képes a kért adatot szolgáltatni, válaszként *megegyező azonosítóval* rendelkező adatkeretet küld.

2.2.5 Hibakeret (Error Frame)

Hibakeretet akkor küld egy egység, ha a buszon bármiféle hibát észlel. A hibakeret két részből áll: Error Flag és Error Delimiter alkotja.

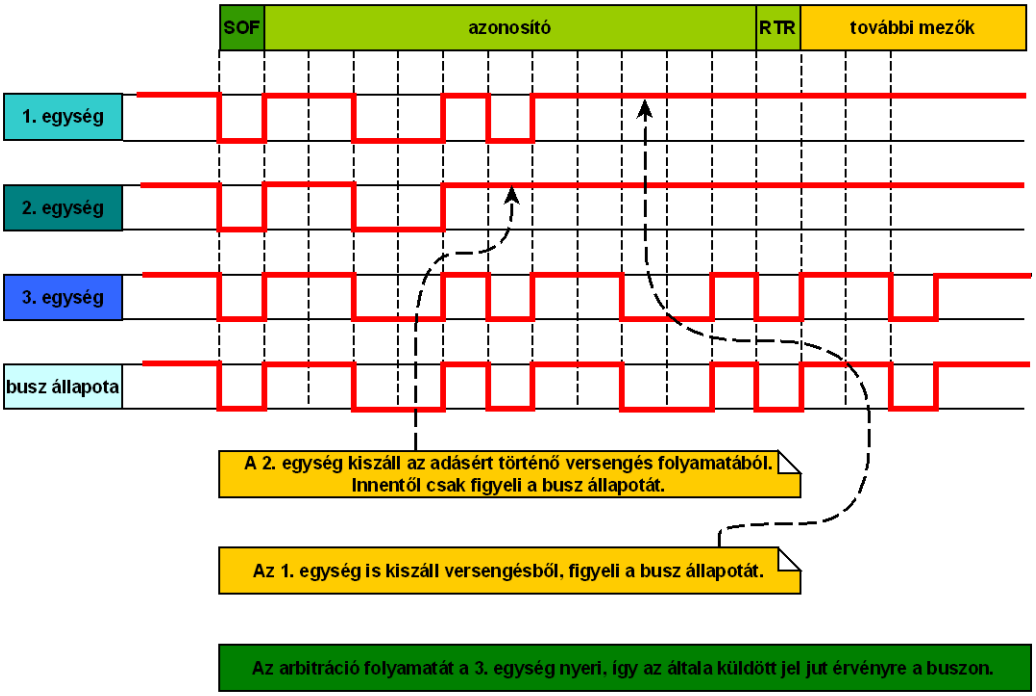
A hibát jelző Error Flag lehet aktív, illetve passzív. A *hiba-aktív* (Error Active) állapotban lévő egységek legalább hat darab domináns bitet küldenek egymás után (Active Error Flag), ha hibát észlelnek a buszon. Ezáltal megsértik a bitbeszúrás szabályát, így hozzák a buszhoz csatlakozó valamennyi egység tudomására a rendellenes működést.

A *hiba-passzív* (Error Passive) állapotú egységek ezzel szemben hat recesszív bitet küldenek a buszra. Amennyiben ezt más domináns bitek nem nyomják el, úgy a hiba-aktív egységek észlelik a bitbeszúrás megsértését, és emiatt hibakeretet generálnak. Az Error Delimiter nyolc recesszív bittel zárja a hibakeretet. Ezt követően indulhat újra a kommunikáció a buszon.

2.2.6 Túlcsordulás keret (Overload Frame)

A túlcsordulás keret felépítése megegyezik az aktív hibakeretével, de azzal szemben csak akkor küldhető, amikor nincs adás a buszon. Küldésével késleltethető a következő adatkeret érkezése.

2.3 Arbitráció és prioritás



2.2. ábra: az arbitráció folyamata

A CAN protokoll multimaster jellegéből adódóan előfordulhat, hogy egyszerre több egység is megkísérli az üzenetküldést a buszon. Egy adásra várakozó master egység adását csak akkor kezdheti meg, ha éppen egyetlen másik egység sem forgalmaz a buszon. Ha azonban több egység egyszerre (szűk időintervallumon belül) kezdi meg a küldést, *bitütközéses arbitráció* lép fel.

A CAN protokollra *CSMA/CA* (Carrier Sense Multiple Access with Collision Avoidance) átvitel jellemző. A busz állapotát valamennyi adó ellenőrzi vevő egysége segítségével. Azt figyelik, hogy az általuk küldött jel megegyezik-e a buszon lévő jellel. Amennyiben eltérést tapasztalnak az általuk várt és a tényleges buszállapot között, adásukat felfüggesztik, és későbbi időpontban próbálják meg az újbóli üzenetküldést. Az arbitráció jellegéből fakadóan a *nyertes adó egység jelsorozata sértetlen marad*, így nem kell megismételnie az üzenet küldését.

A logikai „1” szintet recesszív, a „0” szintet domináns jelszintnek nevezzük. A buszhoz csatlakozó egységek *huzalozott „ÉS” kapcsolatban* állnak egymással: a domináns jelszint érvényre jut, ha bármelyik adó domináns jelet ad a buszra, recesszív szint ellenben csak akkor lehetséges, ha az összes egység recesszív szinten tartja a buszt. A

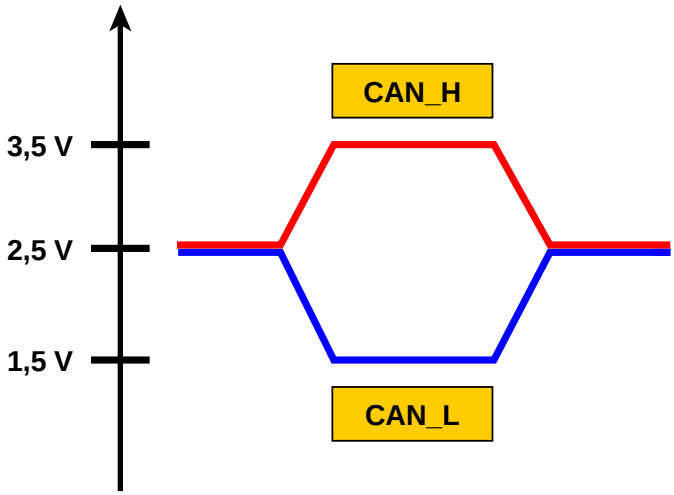
küldés kezdetét jelző SOF még megegyezik valamennyi egység esetén, de az azonosító mezőknél már eltérés tapasztalható. Az az egység nyeri az arbitrációt, amelyik tovább tudja biztosítani saját jelének a buszon történő megjelenését. Minél *kisebb azonosító-szám* szerepel az azonosító mezőben, annál *nagyobb az üzenet prioritása*, hiszen annál korábban kerül dominánsan érvényre jutó „0”-ás szint a buszra.

2.4 Fizikai réteg

2.4.1 Jelátvitel

A CAN hálózatok fizikai rétegének megvalósítására többféle lehetőség kínálkozik. Leggyakrabban *csavart érpárt* alkalmaznak. A hálózat aszinkron jellege miatt az órajel átvitele nem szükséges, így a vezetékezés költségei jelentősen csökkennek. A csavart érpár ellenálló a külső zavarokkal szemben. A jelátvitel zavarvédelmét emellett tovább erősíti a *differenciális jelátvitel*, amely bármely közös módusú zavar esetén az alkalmazott különbségképzéssel kiszűri a zavaró hatást.

A busz tehát mindössze két vezetéket igényel az átvitelhez, amelyek elnevezése CANH (CAN High) és CANL (CAN Low). A domináns („0”) logikai szintnek a vezetékek aktív meghajtással széthúzott állapota felel meg, míg a recesszív („1”) szintet passzív elengedett állapotuk jelenti.



2.3. ábra: differenciális jelátvitel a CAN buszon

Domináns állapotban tehát a buszmeghajtó széthúzza a jelvezetékek feszültségszintjét: a CAN_H vezetéken magasabb (kb. 3,5 V), míg a CAN_L vezetéken alacsonyabb (kb. 1,5 V) feszültség mérhető. Recesszív állapotban a jelvezetékek nincsenek meghajtva, így az alkalmazott lezáró ellenállások hatására azonos feszültségszintre kerülnek.

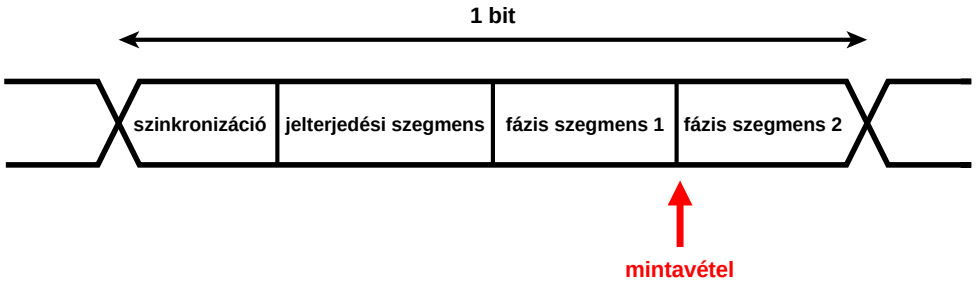
A busz kiterjedése függ az alkalmazott átviteli sebességtől. A maximális sebességű (1 Mbit/s) jelátvitel mellett a busz hossza maximálisan 30m. Alacsonyabb bitsebesség esetén hosszabb busz kiépítése is lehetséges (125 kbit/s esetén 500m).

2.4.2 Bitkódolás és bitidőzítés

A protokoll aszinkron mivoltának pozitívumait már láttuk a vezetékezés költségeinek csökkenése kapcsán, de nem szabad elfeledkezni a hátulütőjéről sem. Az adó és vevő egységek összehangolásának, szinkronizálásának összetett problémáját meg kellett oldani a megbízható kommunikáció fenntartásának érdekében.

Az üzenetenként átvitt legalább 44, de akár 128 bit meglehetősen hosszú jelsorozatot jelent. Az *időzítés összehangolása* tehát kiemelt jelentőségű, mert ilyen hosszú jelsorozatok esetén a küldő és fogadó oldal szinkronból történő kiesése fokozott kockázatot jelent. A CAN által használt bitkódolás, az NRZ-kódolás (Non Return to Zero) előnye, hogy egy bit elküldéséhez elegendő egy időosztás, hátránya viszont, hogy hosszú, azonos értékekből álló jelsorozatok esetén nehézkessé válik a szinkronizáció. Ennek kiküszöbölésére alkalmazza a CAN protokoll a *bitbeszúrás* (bit stuffing) nevű eljárást. Az üzenet küldője minden öt azonos értékű bit után egy ellenkező értékű bitet szúr be, amelyet a vevőegység a fogadó oldalon automatikusan eltávolít. A módszernek köszönhetően a buszon bizonyos időközönként garantáltan történik jelszint-váltás, így a probléma megoldódik.

Egy adott biten belül fontos a mintavétel időpontjának pontos meghatározása a minél megbízhatóbb jelátvitel megvalósítása érdekében. A névleges bitidő négy időszegmensre oszlik, melyek időtartamai az ún. *időkvantum* (Time Quantum) egész számú többszörösei. Az időkvantum a rendszerórából állítható elő leosztással.



2.4. ábra: a CAN bitidőzítése

A jelváltás üzemszerűen a *szinkronizációs szegmens* (Synchronization Segment) alatt történik meg, ennek hossza 1 időkvantum.

A *jelterjedési szegmens* (Propagation Segment) a fizikai jelterjedés miatti késleltetést modellezi és kompenzálja, hossza 1 és 8 időkvantum között programozható.

A *fázisszegmensek* (Phase Segment) között történik meg a mintavételezés. Értékük 1 és 8 időkvantum között mozoghat. A pontos mintavételi időpont meghatározása a fázisszegmensek hosszának állításával történik. *Újraszinkronizálás* akkor következik be, ha az *élváltás a szinkronizációs szegmensen kívül* következett be. Ilyenkor a vevő egység módosítja a fázisszegmensek hosszát. Az időeltolásra szolgáló paraméter a RJW (Resynchronisation Jump Width), melynek értéke 1 és 4 időkvantum között változhat.

2.4.3 A rendszer felépítése

A szabványban rögzített vezérlési feladatokat a CAN *kontroller* végzi el. Külső egységként is csatlakozhat egy adott mikrokontrollerhez, de gyakran integrált formában is megtalálható. Különböző paraméterek segítségével beállítható módon végzi az adatok küldését, vételét, részt vesz az arbitrációban. Rendszerint mind kimeneti, mind bemeneti adattárolót (FIFO) is tartalmaz, így várakozás vagy hiba esetén elkerülhető az adatvesztés. A CAN kontroller *logikai jelszintekkel* dolgozik. A buszhoz történő illesztésre speciális illesztőáramkör, a CAN *transceiver* szolgál, így valósulhat meg a szabványban rögzített *fizikai réteg*.

A CAN hálózat korábban ismertetett fizikai kialakítása miatt az átviteli sebesség függ az alkalmazott vezeték hosszától. A nagysebességű CAN hálózat (High-Speed CAN) 1Mb/s maximális sebességet enged meg, jellemző alkalmazási területe például a motorvezérlés. Az alacsony sebességű CAN (Low-Speed CAN) tipikusan 125kb/s se-

bességre képes és hibakezelő szolgáltatásai is vannak. Autóipari felhasználási területe jellemzően az utastér elektronikai berendezéseihez köthető.

2.5 Hibajelzés és hibakezelés

A CAN fejlett hibajelzéssel és hibakezeléssel rendelkezik. A hálózat sokféle hiba detektálására alkalmas.

Az adó egység – az arbitrációs folyamat ismertetésénél már bemutatott módon – folyamatosan összehasonlítja a küldött és a buszon található biteket. *Bithiba* (Bit Error) akkor keletkezik, ha az arbitrációs szakaszon kívül adódik eltérés az értékek között. A *beszúrási hiba* (Stuff Error) egyszerűen a bitbeszúrás szabályának megsértését jelenti. Az üzeneteket minden esetben 15 bites CRC (Cyclic Redundancy Check) kóddal védik. Amennyiben a vevő eltérést tapasztal a keret alapján számolt és a CRC mezőben szereplő értékek között, regisztrálja a *CRC hibát* (CRC Error). *Formai hiba* (Form Error) esetén a keret rögzített, fix értékű bitjeinek eltérése jelzi az adatok sérülését. *Nyugtázási hiba* (Acknowledgement Error) akkor keletkezik, ha egy adott üzenet küldője által recesszívként kiküldött ACK (Acknowledgement) bitje változatlan (recesszív) értékkel jelenik meg a buszon. Ilyen esetben egyik vevő sem fogadta sikeresen az üzenetet, hiszen az ACK bitet senki nem állította domináns szintre.

Ha egy egység bármilyen hibát észlel a kommunikációs folyamat során, hibakeret küldésével jelzi azt a buszhoz csatlakozó valamennyi egységnek. A hibák nyilvántartásba kerülnek, és adott hibaszint elérése után a hibásan működő egységet a CAN hálózat *szankciókkal sújtja*. Egy adott egység állapotát két számláló értéke befolyásolja. Az *REC* (Receive Error Count) a fogadásnál fellépő hibáknál, míg a *TEC* (Transmit Error Count) az adatküldésnél adódó hibák észlelése során növekszik. Mindkét számláló értéke csökken, ha sikeres vétel, vagy küldés történik.

A számlálók aktuális értékének megfelelően változik az egységek státusza három lehetséges állapot között. A *hiba-aktív* (Error Active) állapotú egységek teljes körűen részt vesznek a buszon zajló kommunikációban és hiba esetén Active Error Flaget küldenek. A *hiba-passzív* (Error Passive) állapotban lévők szintén kommunikálhatnak, azonban ezek csak Passive Error Flaggel tudják jelezni a nem megfelelő működést. *Lecsatlakozott* (Bus Off) állapotban az egységek semmiféle hatást nem gyakorolnak a busz állapotára.

3 Az AUTOSAR szabvány

3.1 Bevezetés

Az AUTOSAR mozaikszó az AUTomotive Open System ARchitecture szavakból tevődik össze. Egy mindenki által szabadon elérhető, autóipari teljes szoftverarchitektúrát definiáló szabvány. Számos neves autóipari cég együttműködése révén jött létre, és mind a mai napig folyamatos fejlesztés és frissítés alatt áll. 2002-ben kezdődött a szabvány fejlesztése, a jelenleg elérhető legfrissebb változat a 4.0-ás verzió. Jelen szakdolgozat alapjául is ez szolgál.

Az ezredfordulóra az autóelektronika oly mértékű térhódítása volt tapasztalható, hogy szükségessé vált az elektronikus egységeket kiszolgáló szoftverek szabványos architektúrájának kidolgozása. A több gyártó által, közös javaslatok felhasználásával megalkotott szabvány felhasználja az autóipar résztvevőinek sok évtizedes tapasztalatait és az *architektúrális elvek* megfogalmazása mellett *módszertani rendszert* is definiál a minél hatékonyabb szoftverek fejlesztéséhez.

Az AUTOSAR irányelveknek köszönhetően az összetett szoftverek könnyebben konfigurálhatóvá válnak, a modulokban és komponensekben történő fejlesztés és a köztük lévő határfelületek egységes kialakítása lehetővé teszi a könnyű illeszthetőséget és a különböző fejlesztők által megírt programok integrálhatóságát. Az alkalmazási réteg elválasztása az alapvető szolgáltatásokat nyújtó szoftverrétegtől lehetővé teszi, hogy egy alkalmazás fejlesztése biztos alapokon történjen, gyorsabbá és egyszerűbbé és ez által költséghatékonyabbá váljon. A villámgyorsan fejlődő autóelektronikai megoldások megkövetelik a szoftver architektúra *rugalmasságát és újrakonfigurálhatóságát*. Egy-egy elektronikai termék teljes élettartamán keresztül fejlődik, állandóan kisebb-nagyobb módosításokat, fejlesztéseket végeznek rajta. Egy jól strukturált kiszolgáló szoftver könnyedén megbirkózik az ebből adódó nehézségekkel, mindig optimálisan alakítható a megújult követelményrendszernek megfelelően.

Az autóiparban alkalmazott tipikus ECU az alábbi legfontosabb tulajdonságokkal bír:

- szoros kapcsolat a hardverrel (szenzorok és beavatkozók)

- járműiparban használatos kommunikációs hálózatokhoz való csatlakozás lehetősége (CAN, LIN, FlexRay, Ethernet)
- 16 vagy 32 bites mikrokontrollerek korlátozott memória-hozzáféréssel és számítási kapacitással
- valósidejű operációs rendszer
- flash memóriából történő programvégrehajtás

Az AUTOSAR autóipari szabvány megalkotásakor a fent említett tulajdonságokat vették alapul, erre optimalizálták a rendszer kiépítését.

Az AUTOSTAR tehát egy egységesített architektúrát épít ki, ahol egy szabványos központi szoftvermag köré rendeződnek az alkalmazásszintű egységek. Ezeket komponensként definiálja a szabvány. A *komponensek* önálló, újrafelhasználható zárt egységek, amelyek egy-egy jól körülhatárolható funkcionalitást valósítanak meg. Egymással jól definiált interfészeken keresztül kommunikálnak.

A fejlesztés modellalapú, a futásidejű környezet (RTE) és a konfigurációs fájlok generált elemként jönnek létre.

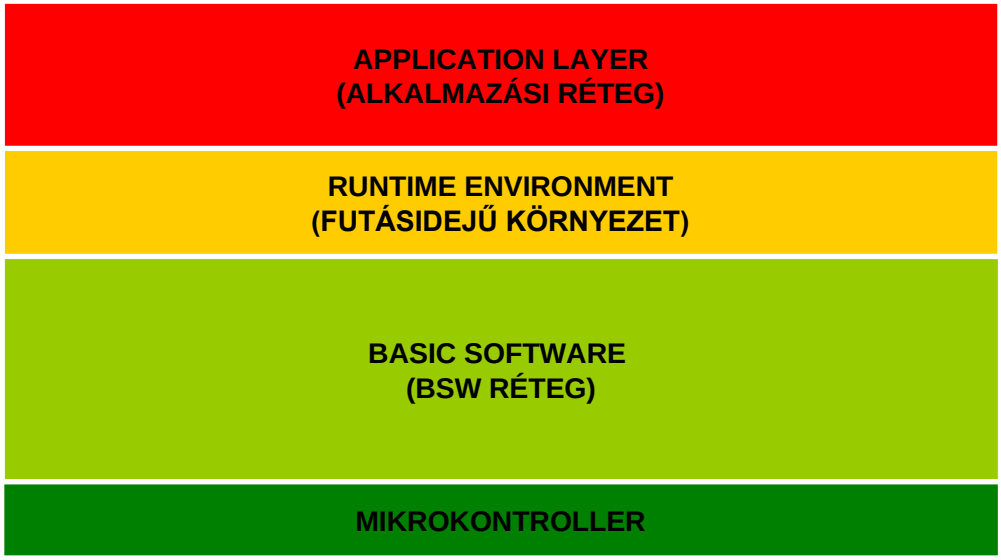
A szabvány csökkenti a szoftverfejlesztés költségeit. A szabványos modulok, komponensek, és az egységes architektúra jelenléte mellett ezt a komponensek egyszerű kicserélhetősége, kis változásokkal történő újrahasznosításának lehetősége is biztosítja. A rendszerezett felépítés lehetővé teszi nagy komplexitású integrált rendszerek megalkotását. A fejlesztés az összetettség miatt csapatmunkát követel, ez könnyen megvalósítható különböző csoportok, osztályok, sőt akár cégek között is.

3.2 Szoftver architektúra

3.2.1 Alapvető rétegek

Az AUTOSAR architektúra három fő rétegre bontható:

- Application Layer
- Runtime Environment (RTE)
- Basic Software (BSW)



3.1. ábra: az AUTOSAR szoftver architektúra rétegződése

Az AUTOSAR világosan elkülöníti az alkalmazásszintű és rendszerszintű szoftverelemeket. Előbbiek az alkalmazási rétegben elhelyezkedő szoftverkomponensek, míg utóbbiak gyűjtőneve a Basic Software.

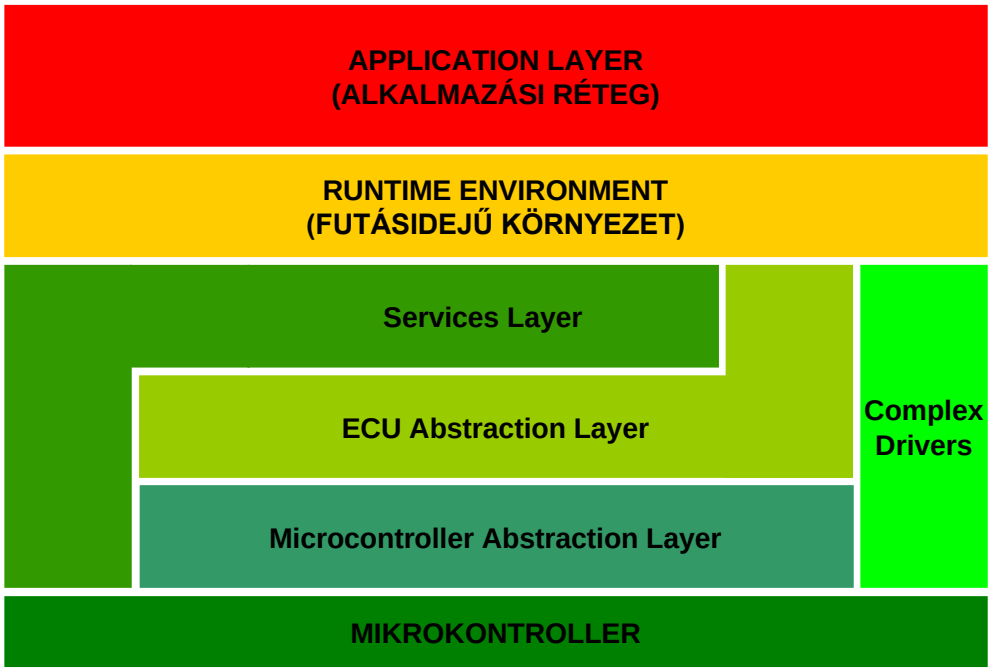
Az *alkalmazási réteg* (Application Layer) tartalmazza az ECU-n futó, az AUTOSAR elvek szerint felépülő alkalmazási szoftver komponenseket (SWC). A komponensek egymás közti és a környezet felé irányuló kommunikációját szabványosított portok biztosítják. Az AUTOSAR alapvető célja, hogy hardverfüggetlen keretrendszert biztosítson az alkalmazási komponensek fejlesztéséhez, ennek megfelelően a komponensek elől rejtve marad, hogy egy adott porton keresztül a kommunikáció milyen körülmények között zajlik le. A portok közötti összeköttetést egy virtuális hálózat, az ún. *Virtual Function Bus* (VFB) bonyolítja le. A VFB hardverfüggetlen kapcsolatot biztosít, így a megírt alkalmazások hordozhatóvá válnak, nem függnek az adott ECU tulajdonságaitól.

A *futásidejű környezet*, a Runtime Environment (RTE) a komponensek szemszögéből látott, generált VFB. Ez köti össze az absztrakt technológia és hardverfüggetlen alkalmazási réteg moduljait, megfelelő alapot nyújt ahhoz, hogy az alkalmazások elől a kommunikációs és hardveres rétegek rejtve maradjanak, ezáltal egységes kapcsolódási felületet képez. Az RTE-t minden ECU-ra egyénileg generálják. Az RTE dönt arról is, hogy egy adott adatcsomag az ECU-n belüli kommunikációs elemként jut célba, vagy egy másik ECU-val történő kommunikációs folyamatban vesz részt.

3.2.2 A BSW réteg

A BSW egy szabvány szerinti *alapvető szoftvercsomag*, amely a működési feltételeket biztosítja az alkalmazások futtatásához. Az alkalmazási réteghez hasonlóan önálló egységekből, ún. *modulokból* áll, melyek között jól definiált interfészeken zajlik a kommunikáció. A BSW réteg további részekre osztható.

3.2.2.1 A BSW réteg további felosztása



3.2. ábra: a BSW réteg felosztása

A *szolgáltatási réteg* (Services Layer), a BSW legfölső része a BSW moduljainak és az alkalmazási rétegnek nyújt alapvető szolgáltatásokat, utóbbi esetben az RTE-n keresztül. A Services Layer szolgáltatásai között szerepel a hálózati kommunikáció lebonyolítása és menedzselése (COM, ComM, Nm), memóriakezelés (NVRAM Manager), diagnosztikai szolgáltatás (DCM) és az ECU állapotának és futási módjának menedzsmentje (EcuM). Itt helyezkedik el az operációs rendszer (OS) is. A réteg javarészt mikrokontroller- és hardverfüggetlen funkciókat lát el, a fölsőbb rétegek számára teljes egészében hardverfüggetlen interfészt biztosít.

Az *ECU absztrakciós réteg* (ECU Abstraction Layer) jelenti az ECU független felületet a fölsőbb rétegek számára. Rajta keresztül a perifériák és külső eszközök elhe-

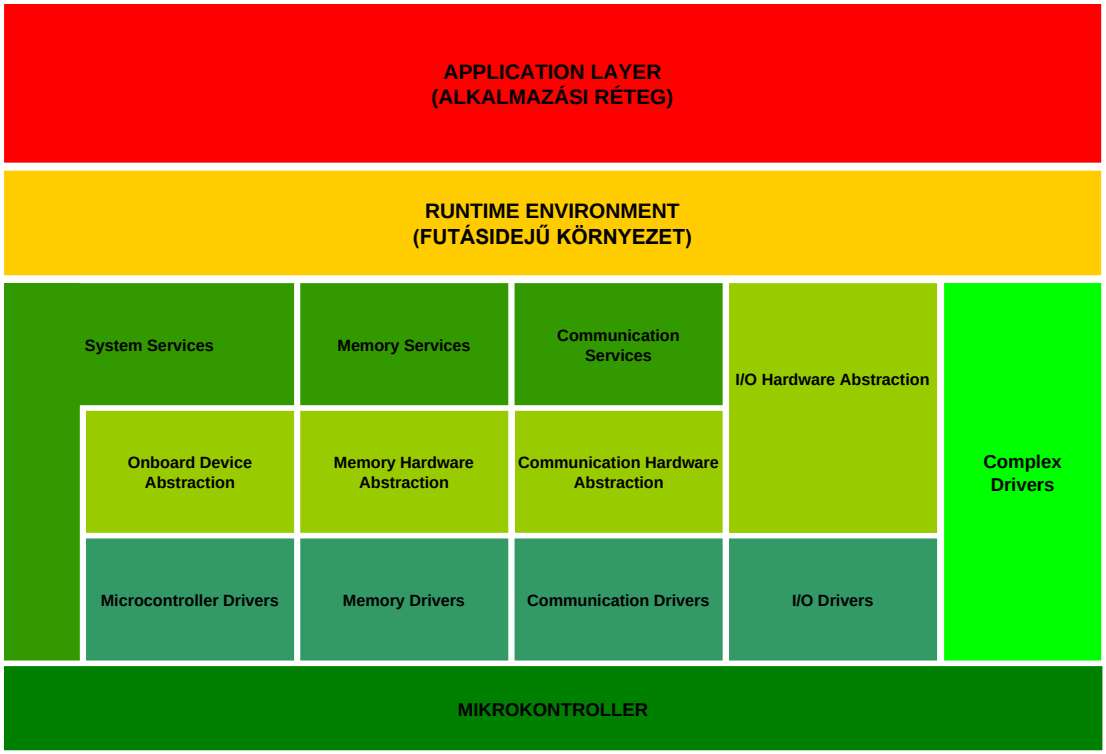
lyezkedésüktől és a mikrokontrollerhez való csatlakozási módjuktól függetlenül elérhetővé válnak.

A *mikrokontroller absztrakciós réteg* (Microcontroller Abstraction Layer) mikrokontroller-függő, a BSW legalsó rétege. A belső perifériákhoz és a mikrokontrollerhez közvetlen hozzáféréssel rendelkező szoftver modulokból áll. Célja, hogy a följebb elhelyezkedő modulokat függetlenné tegye a mikrokontroller tulajdonságaitól.

A *komplex meghajtók* (Complex Drivers) egy nem szabványosított, speciális funkciókat kiszolgáló rétegben találhatók. Az RTE és a hardver réteg között helyezkednek el, így közvetlen csatlakozási lehetőséget jelentenek egyes applikációk számára a szenzorokhoz, vagy beavatkozókhoz, így az AUTOSAR előtti időkben megszokott alacsonyszintű hardverkezelést is ellátó, rétegződés nélküli felépítést követik. A szabvány által közvetlenül nem támogatott speciális hardverelemek kezelésére, vagy extrém időzítési elvárások teljesítésére használnak komplex meghajtókat.

3.2.2.2 Funkcionális felosztás

A Basic Software Layer a modulok által nyújtott szolgáltatások jellege alapján is felosztható.



3.3. ábra: a BSW réteg funkcionális felosztása

Az (I/O) szolgáltatások a *szabványos ki- és bemeneteken* keresztül elérhető szenzorokkal és beavatkozókkal, illetve más perifériákkal kapcsolatosak. A *memória* (Memory) szolgáltatások elérhetővé teszik a belső és külső memóriablokkokat. A *rendszerrel kapcsolatos* (System) szolgáltatások szerteágazóak. Ide tartozik az operációs rendszer, az időzítők kezelése csakúgy, mint az ECU állapotainak, vagy a watchdog áramkörnek a menedzselése. A *kommunikációs* (Communication) szolgáltatások szabványosított hozzáférést nyújtanak a jármű kommunikációs hálózataihoz, valamint az ECU belső kommunikációs rendszeréhez. Az adott feladatok ellátására speciális modultípusok szolgálnak. Ilyenek a driverek, az interfész és a menedzser egységek.

A *driver* feladata egy külső vagy belső eszköz illesztése. A belső eszközök driver moduljai a mikrokontroller absztrakciós rétegben találhatók meg (pl. belső EEPROM, vagy a mikrokontrolleren elhelyezkedő CAN vezérlőegység). A külső eszközök (külső watchdog, külső EEPROM) driverei az ECU absztrakciós rétegben vannak, és a mikrokontroller absztrakciós rétegen keresztül érik el a külső eszközt (pl. az SPI-on elérhető külső EEPROM drivere az SPI busz driverén keresztül éri el a külső EEPROM-ot).

Az *interfész* modulok szintén az ECU absztrakciós rétegben találhatóak meg. Ezek elvonatkoztatnak az adott eszközök hardveres realizációjától (pl. a CAN Interface modul lehetővé teszi, hogy a CAN hálózat a kontrollerek számától, illetve hardveres realizációjuktól (on chip/ off chip) függetlenül elérhetővé váljon).

A *menedzser* modulok több kliens működését kötik össze, a szolgáltatási rétegben helyezkednek el [2].

3.3 A kommunikációs stack

3.3.1 A leggyakoribb kommunikációs protokollok

Manapság az autóiparban a legelterjedtebb kommunikációs protokollok a CAN, a LIN és a FlexRay. A különböző alkalmazási körök különböző fizikai kialakítást és eltérő adatátviteli sebességet eredményeznek.

A *LIN (Local Interconnect Network)* az 1990-es évek végén jelent meg és a CAN-nél egyszerűbb működés jellemzi. A CAN kiegészítő hálózataként feladata az autó egy-egy kisebb egységének vezérlése, illetve a kommunikáció biztosítása. Az egyes

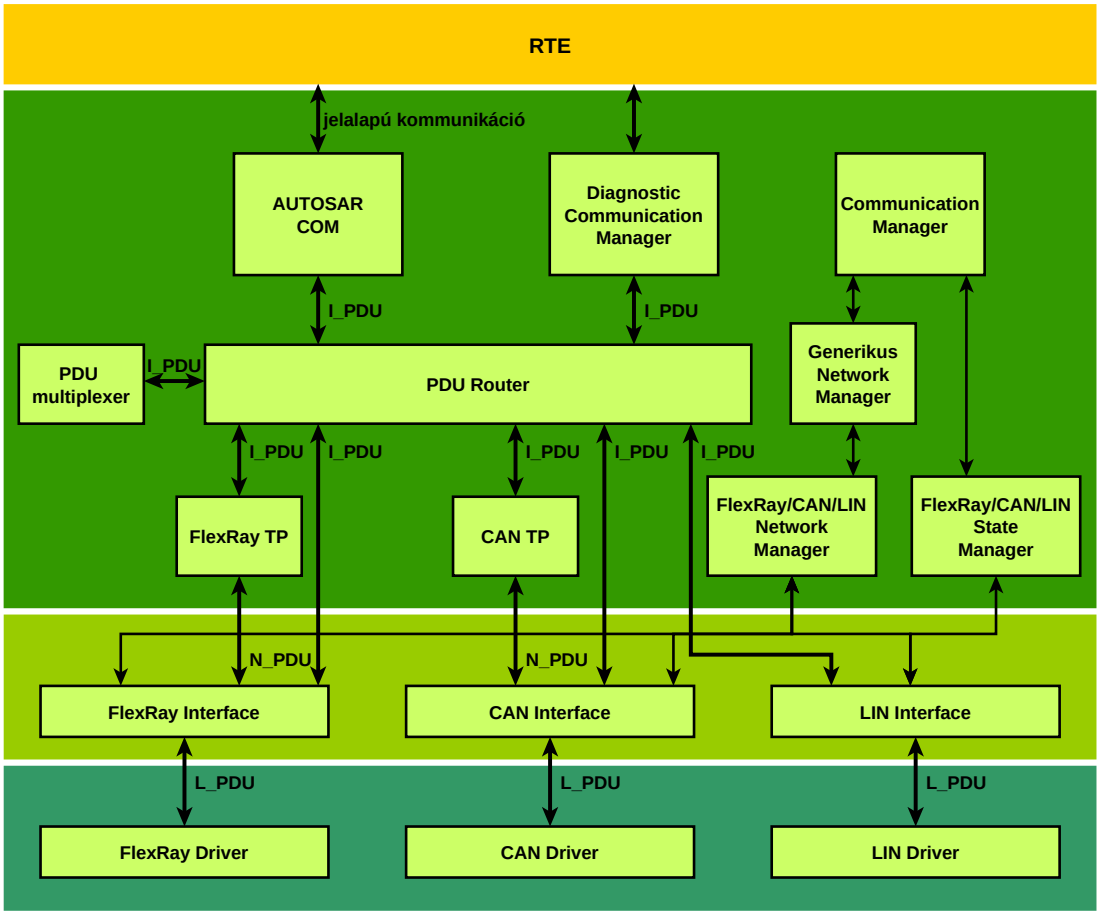
LIN alhálózatok egy másik kommunikációs protokoll (pl. CAN) szerint működő központi hálózatra csatlakozva kerülnek összeköttetésbe a központi vezérlőegységgel. A LIN fejlesztésekor tehát szem előtt tartották az egyszerűséget, az olcsóságot, valamint a CAN-nel történő együttműködés lehetőségét. A LIN aszinkron, soros átvitelű kommunikációs protokoll. Egy buszon egyetlen master és több slave egység helyezkedik el. Arbitráció nem fordul elő, mivel adatátvitelt csak a master egység kezdeményezhet. A kommunikáció üzenetszórásos alapú, a küldő egység üzenete a buszhoz kapcsolódó összes többi egységhez eljut. A fizikai réteg egyvezetékes, így az átvitel félduplex jellegű. (A buszon felváltva lehetséges küldés illetve fogadás.) A maximális adatátviteli sebesség 20 kbit/sec. Tehát egy egyszerű felépítésű, részegységek vezérlésére szolgáló, olcsón kialakítható és ennek megfelelően nem igazán gyors kommunikációs protokollról beszélünk [3].

A *FlexRay* az említett három protokoll közül a legújabb, legösszetettebb és a legnagyobb adatbiztonságot garantáló szabvány. Nagy sebesség (10 Mbit/s) jellemzi, hibatűrő, gyors és redundáns átvitelének köszönhetően idő- és biztonságkritikus alkalmazások esetén is használható (x-by-wire alkalmazások). A FlexRay hálózat két átviteli csatornával rendelkezik (A és B csatorna), mindkettő csavart érpárt használ a vezetékezéshez, amelyen differenciális jelátvitel zajlik. A hálózat topológiája rugalmasan alakítható ki, így a rendszer sokféle összeköttetést támogat. Az egyszerű pont-pont összeköttetés, a passzív busz és passzív csillag kialakítás mellett lehetséges az aktív csillag topológia is, vagy testre szabható lehetőséget kínálva rendelkezésre áll az eddig említettek ötvözete, a hibrid megoldás is. A kommunikációs protokoll idővezérelt. Az előre meghatározott időpontban lezajló, determinisztikus adattovábbítást épp úgy képes ellátni, mint a CAN hálózatnál megismert eseményvezérelt, változó időpontokban bekövetkező kommunikációs egységek kezelését. Az összetett tevékenység elvégzésére kommunikációs ciklusok állnak rendelkezésre. A statikus és a dinamikus adatkeretek külön meghatározott idősávot kapnak minden ciklus során. Az idősávok a konfiguráció során állíthatók be [4].

3.3.2 Az AUTOSAR kommunikációs szoftverrétege

Az autóipar leggyakrabban alkalmazott kommunikációs szabványait az AUTOSAR is támogatja. A *kommunikációs réteg* (Communication Stack) moduljai a BSW rétegben helyezkednek el. A szolgáltatási réteg kommunikációért felelős részében találhatók meg

a kommunikációs szolgáltatások (Communication Services) moduljai (3.4. ábra), melyek lehetővé teszik a CAN, LIN és FlexRay hálózatokon keresztül történő kommunikációt és egyben protokoll-független felületet képeznek a fölsőbb, alkalmazási rétegek felé. A funkcionális egységek tekintetében a három protokoll között igen nagy hasonlóság látható. A korábban ismertetett rétegződésnek megfelelően a legalsó réteget a kommunikációs hardver eszközök vezérléséért és felügyeletéért felelős driverek moduljai alkotják. A kapcsolatot köztük és a kommunikációs szolgáltatások moduljai közt a kommunikációs hardver absztrakciós rétegben (Communication Hardware Abstraction) elhelyezkedő interfészek jelentik.



3.4. ábra: a kommunikációs szoftverréteg

3.3.2.1 A protokoll-független modulok ismertetése

A BSW legfölső, szolgáltatási rétegében számos kommunikációval kapcsolatos funkció ellátását teljesítő, de még protokoll-független modul található. Az AUTOSAR COM, a Communication Manager, a Generic Network Management Interface, a PDU Router és

a Diagnostic Communication Manager modul kommunikációs hálózattól független, így *ECU-nként egy-egy példányban* léteznek.

Az AUTOSAR Communication modul

Az RTE és a PDU Router (PduR) között elhelyezkedő COM modul a kommunikáció lebonyolításának egyik központi egysége. A jelalapú és a PDU-alapú kommunikációs egységek között biztosít interfészt. Adatküldéskor az RTE felől érkező *jeleket* (signal) szűri, *I_PDU-kba* rendezi őket és továbbküldi a PDU Router felé, fogadáskor a beérkező PDU-k alapján generál signalokat. Elvégzi az esetlegesen szükséges endian-konverziókat is (LSB vagy MSB érkezik először). Felügyeli, ütemezi az üzenetek küldését és nyomon követi a küldött üzentek sikeres fogadására vonatkozó megerősítéseket (TxConfirmation) [5].

A Diagnostic Communication Manager

A Diagnostic Communication Manager (DCM) a kommunikációs hálózatokon keresztül érkező *diagnosztikai jellegű üzenetek* fogadásáért és továbbításáért, elosztásáért felelős. A DCM fő célja egy közös diagnosztikai API megvalósítása, amely külső diagnosztikai eszközök használatát és a fejlesztést teszi lehetővé.

A Communication Manager

A Communication Manager (ComM) modul feladata az egyes *kommunikációs hálózatok buszainak és hardveregységeinek* összehangolt és megbízható működtetése. A ComM az alatta elhelyezkedő kommunikációs modulok működését befolyásolja, irányítja. Fölfelé egy jól kezelhető felhasználói felületet biztosít. Egyszerűvé teszi a kommunikációs buszok használatát az azt igénybevevők számára. A hálózatok felhasználóinak nincs tudomása arról, hogy milyen hardveres egységeket vesznek igénybe, vagy melyik csatornán zajlik a kommunikáció, csak egyszerűen kommunikációs módot választanak (FULL_COMMUNICATION vagy NO_COMMUNICATION).

A ComM modul egyszerre több kommunikációs csatorna kezelésére képes. Minden csatornához tartozik egy belső állapotgép, amely a megfelelő protokoll State Manager moduljának segítségével állítja a hardveregységeket (kontrollereket és transceivereket), a Network Management modulon keresztül pedig a busz állapotát kezeli.

Network Management Interface

A generikus Nework Management Interface a ComM modulhoz kapcsolja a protokoll-specifikus Network Management modulokat (pl. CAN NM, FlexRay NM). Alkalmazása csak olyan kommunikációs rendszerben engedhető meg, amely támogatja a broadcast üzemmódot és a hálózat alvó állapotba (BUS SLEEP) küldését. A ComM és a NM modulok közötti közvetítésen kívül más alapvető szerepe, funkcionalitása nincs.

Gateway biztosítására szolgáló ECU-k esetén a Network Management Interface azonos vagy különböző típusú kommunikációs hálózatok közötti szinkronizációs teendőket is elláthat.(Egyszerre kapcsolja ki, vagy ébreszti őket.)

PDU Router

A PDU Router (PduR) egy úgynevezett *kapcsolási tábla* (routing table) alapján lát el *üzenetirányítási feladatokat*. Küldéskor a COM modul felől érkező PDU-kat a kapcsolási táblában nyilvántartott kapcsolási paraméterek alapján irányítja a megfelelő alsóbb rétegek felé, kommunikációs protokollt társít hozzájuk. Multiplexerként működik: kapcsolatban áll az összes Interface és Transport Layer modullal. Üzenetfogadáskor az alsóbb rétegek felől érkező adatokat továbbítja a fölsőbb rétegnek. A modul gateway funkciókat is támogat. Ez esetben a PduR különböző kommunikációs hálózatok összekapcsolására szolgál. Ezek lehetnek azonos és eltérő típusúak is (CAN1 – CAN2, CAN – FlexRay).

Fontos megjegyezni, hogy magának az I_PDU-nak a tartalmát a PDU Router nem módosítja. A PDU útja statikusan meghatározott, köszönhetően a statikus kapcsolási táblának és az üzenetekhez statikusan hozzárendelt PDU-Id-k nak [6].

I_PDU Multiplexer (IPduM)

Léteznek olyan I_PDU-k, amelyekhez több lehetséges SDU tartozik, mégis azonos PCI mezővel rendelkeznek (ld. 3.4.1.1). Az SDU azonosítására egy választó mező (selector field) szolgál. A választó mezőt az SDU tartalmazza. Küldéskor az IPduM a COM modultól kapott PDU-t átalakítja, és egy másik PDU-t küld a PduR modulnak.

A PDU multiplexelést egyelőre csak a CAN hálózatnál alkalmazzák, de nem kizárt, hogy a jövőben más protokollokra is kiterjesztik használatát.

3.3.2.2 Protokoll-specifikus kommunikációs modulok

A kommunikációs modulok egy része protokoll-specifikus, ezek minden egyes kommunikációs hálózat típusra külön-külön rendelkezésre állnak. Mivel alacsonyszintű, részben hardver-közeli tevékenységet ellátó modulokról van szó, mindegyikben felmerülnek olyan megoldandó problémák, amelyek általánosan nem fogalmazhatóak meg. Az AUTOSAR szabvány támogatja a kommunikációs protokoll mind adatkommunikációra vonatkozó, mind hardveres részének vezérléséért felelős moduljainak megvalósítását.

Az egyes funkciókért felelős egységek bemutatása a dolgozat hátralévő részében a CAN stack moduljainak példáján keresztül történik.

CAN Transport Layer

A CAN Transport Layer (CanTp) üzenetek szegmentálását vagy az ellenkező irányban történő újraegyesítését végzi. Küldéskor a fölsőbb rétegek felől érkező, a CAN protokoll keretformátumán túlmutató méretű adatblokkokat megfelelő CAN keretekbe rendezi, így megvalósítható továbbküldésük. Fogadáskor ellentétes folyamat játszódik le: a részletekben érkező adatok a fölsőbb rétegek számára egészben lesznek elérhetők.

CAN Interface

A CAN Interface (CanIf) modul absztrakciós felületként hozzáférést nyújt a CAN Driverek és CAN Transceiver Driverek szolgáltatásaihoz. Irányítja és felügyeli a CAN hálózatot. Részt vesz az üzenet küldésének és fogadásának folyamatában, és rajta keresztül érhetők el a hardveregységek [7].

CAN Driver

A CAN Driver elrejti egy adott típusú CAN kontroller hardver-specifikus tulajdonságait. A CanIf modul biztosít interfészt a CAN hálózat összes vezérlőjéhez. A CAN Driver hozzáfér a hardver memóriájához, a hardver által megkövetelt formátumba rendezi a küldeni kívánt adatokat, és irányítja az üzenetküldés és fogadás folyamatát [8].

CAN Transceiver Driver (CanTrcv)

A CAN hálózathoz tartozó transceiver üzemmódjainak menedzseléséért, az egység vezérléséért felelős modul. Implementációja erősen hardverfüggő. Kikapcsolt ECU esetén az ébresztés egy lehetséges módja a CAN hálózaton keresztül történik. Ez esetben a transceiver egyik lábán jelenik meg az aktivizáló jel, így a CanTrcv modul fontos sze-

rephez jut a wake-up folyamat kapcsán: a modul közreműködésével ébred az egész rendszer.

CAN Network Management

A CAN Network Management (CanNm) modul *kezeli a CAN busz állapotait*. A *NORMAL OPERATION* és *BUS SLEEP* mód közötti váltást felügyeli. (Utóbbi főként energiatakarékossági céllal, a használaton kívül lévő busz alacsony áramfelvételű működését jelenti.) Az AUTOSAR CanNm-algoritmus alapját periodikusan, üzenetszórásos (broadcast) jelleggel küldött Nm-üzenetek képezik. Amennyiben a hálózatra csatlakozó egységek közül bármelyiknek szüksége van az adott buszra, egy üzenet kiküldésével tudja elérni, hogy az normál állapotban maradjon. Az üzenetet minden, a hálózatra csatlakozó egység fogadja. Ha egy egység már nem tart igényt a buszra, azaz készen áll az alvó állapotra, akkor felfüggeszti az Nm-üzenetek küldését. Ha végül már egyik egységtől sem érkezik ilyen üzenet, a hálózat BUS SLEEP módba kerül (egy erre a célra szolgáló időzítő lejárta után). Kommunikációs igény esetén az ébresztést bármelyik egység képes előidézni, amennyiben újrakezdi az Nm-üzenetek küldését. A CanNm modul tevékenységét a ComM koordinálja, a generikus Network Management Interface-en keresztül.

CAN State Manager

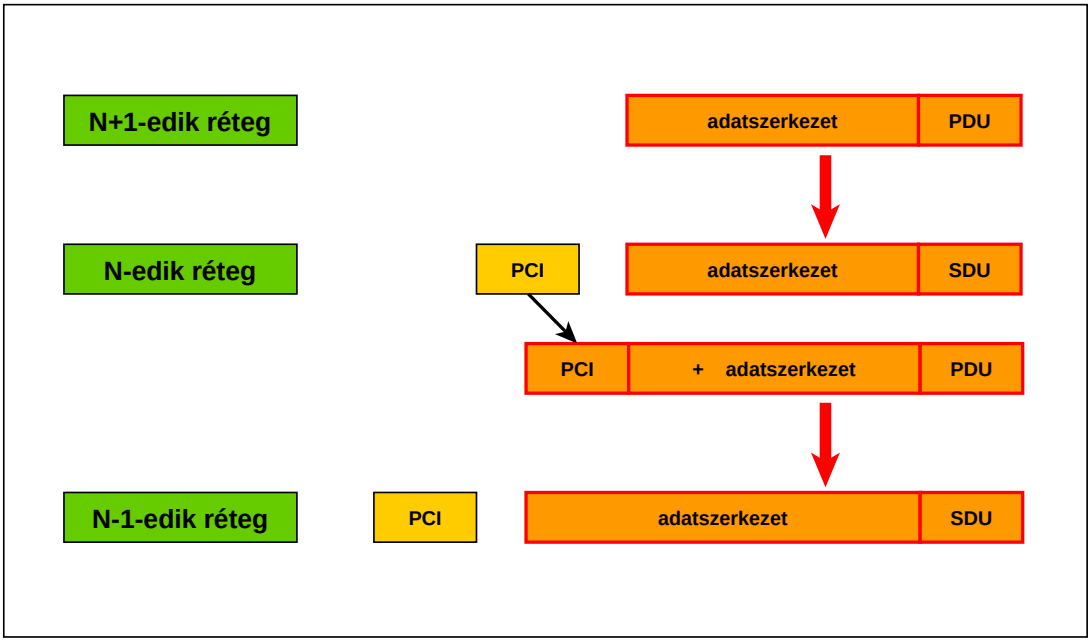
A CAN State Manager (CanSM) modul felelős a CAN hardveregységek ki- és bekapcsolásáért. A *kontrollerekre és transceiverekre jellemző* működési állapotokat kezelő és az állapotok közt fellépő megfelelő tranzienseket biztosító *állapotgépet* valósítja meg. A CanSM a CanIf felületén keresztül képes állítani, illetve lekérdezni a hálózathoz tartozó kontrollerek és transceiverek állapotát. Az erre vonatkozó vezérlést a ComM modultól kapja. A ComM modul az alkalmazási rétegtől az RTE-n keresztül egy azonosítót kap, valamint egy hozzá tartozó, kommunikációs módra vonatkozó kérést. A konfigurációjában rögzített adatok alapján megállapítja, hogy az adott azonosító például a CAN hálózathoz tartozik, így az állapotmenedzselésért felelős modulok közül a CanSM-et választja [9].

3.4 Két ECU közötti, CAN alapú kommunikáció

3.4.1.1 A PDU szerkezet bemutatása

Az alkalmazási réteg komponensei egymással jelek (signalok) formájában kommunikálnak. Ezzel szemben a BSW rétegben – így a CAN stacken belül is – az adatkommunikáció ún. PDU-k küldésével és fogadásával történik.

A rétegek közti interakció alapegysége, a *PDU (Protocol Data Unit)* két részre oszlik: magába foglalja a PCI-t és az SDU-t. Az *SDU (Service Data Unit)* üzenetküldéskor egy adott réteg szempontjából nézve egy fölsőbb rétegtől érkező adatcsomag. Az adott réteg PCI-vel kiegészítve, saját PDU-jaként küldi tovább az alsóbb réteg számára. Üzenetek fogadásakor a PDU-ból kinyert SDU továbbítandó a fölsőbb rétegek irányába. A *PCI (Protocol Controll Information)* az SDU továbbításához szükséges többletinformációt tartalmazó mező. A küldő oldalon a protokoll réteg adja a küldendő adathoz és fogadáskor eltávolítja a fogadott PDU-ról.



3.5. ábra: a PDU-SDU szerkezet

Egy adott réteg PDU-ja tehát megegyezik az alatta elhelyezkedő réteg SDU-jával. A kezelhetőség és a PDU-SDU elnevezés relatív mivolta miatt a PDU-k és SDU-k rétegfüggő előtagot kapnak. Az I_ előtag az interakciós, az N_ a hálózati (network), az L_ az adatkapcsolati (data link) rétegre utal. A COM és DCM modul, valamint az adattarta-

mon nem változtató PduR az interakciós, a CanTp a hálózati, míg a CanIf és a CanDriver az adatkapcsolati ISO rétegben helyezkedik el [2].

ISO RÉTEG	ELŐTAG	AUTOSAR MODUL	PDU NÉV	CAN	LIN	FlexRay
INTERAKCIÓS	I	COM, DCM	I_PDU			
		PDU ROUTER				
HÁLÓZATI	N	TRANSPORT LAYER	N_PDU	CAN SF/FF/CF/FC	LIN SF/FF/CF/FC	FR SF/FF/CF/FC
ADATKAPCSOLATI	L	INTERFACE, DRIVER	L_PDU	CAN	LIN	FR

3.6. ábra: összefoglaló táblázat a PDU-król

3.4.1.2 A kommunikációhoz szükséges bufferek

A küldeni kívánt vagy a fogadott adatok a kommunikációs folyamat során számos modult érintenek. A modulok között tehát *minimálisra* kell korlátozni az *adatmásolást és tárolást*. Ennek megfelelően csak a legfelső és legalsó réteg – a COM és a CAN Driver – rendelkezik buffer-kapacitással. A közbenső rétegek közvetlenül ezeken a memóriaterületeken dolgoznak. Az adatok konzisztenciájának érdekében a küldés vagy fogadás idejére az adott folyamatban részt vevő memóriaterületet le kell zárni, és csak akkor szabad ismét elérhetővé tenni, amikor véget ért a folyamat. Küldéskor ez az időszak a küldés kérésétől (PduR_ComTransmit) a sikeres küldést jelző nyugtázásig (Com_TxConfirmation), míg beérkező üzenet esetén az adatfogadás jelzésétől (RxIndication) az adat feldolgozásáig tart. (Megjegyzés: a leírtakkal némiképp ellentmondásosan – a feltorlódott, küldeni kívánt adatok eltárolására – a CanIf modulnak is lehet saját buffere (CanIfTxBuffer).)

3.4.2 Adatküldés

3.4.2.1 Alkalmazási réteg

Az alkalmazási rétegben a kommunikáció komponensek között zajlik. Egy komponens kizárólag szabványosított portjain keresztül képes az üzenetek küldésére és fogadására. Az adott komponens elől rejtve marad az, hogy a másik komponens, amelyikkel kommunikál az azonos, vagy másik ECU-ban található. A kommunikációs stack működése akkor vizsgálható annak teljességében, ha két különböző ECU-n futó alkalmazáskom-

ponens közötti üzenetváltást feltételezünk. A továbbiakban tehát erre az esetre koncentrálunk. A komponens egyszerűen kiírja a rendelkezésre álló portra (<PORT>) a küldeni kívánt adatkomponenst (<DATA>).

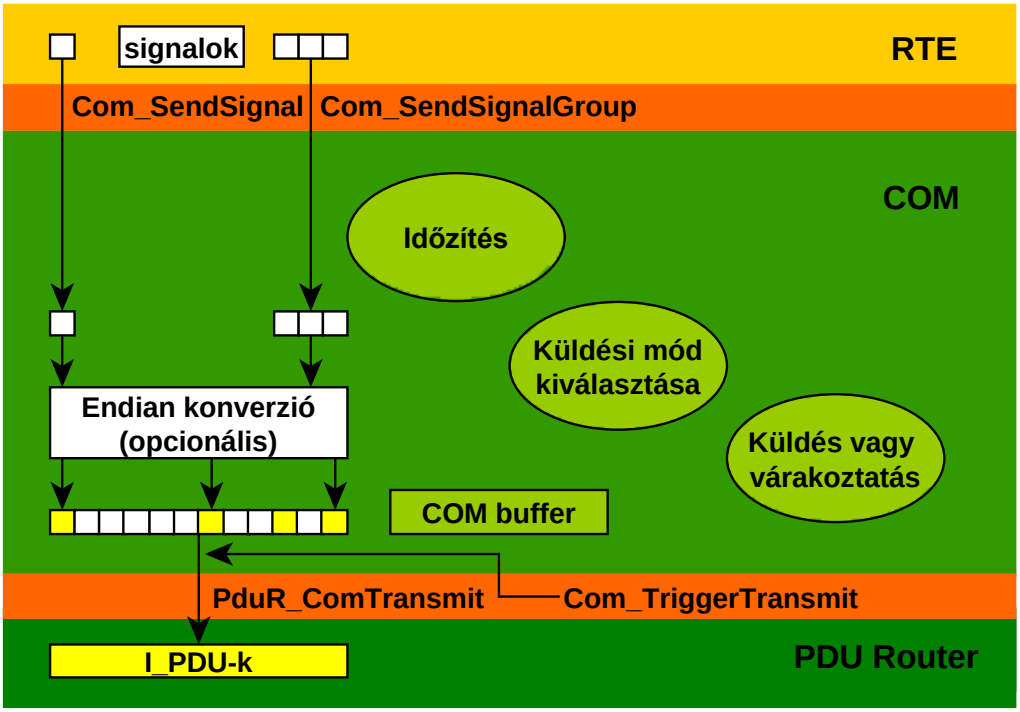
```
Rte_Write_PORT_DATA()
```

A megfelelő kommunikációs porton keresztül történő üzenetküldés bonyolult belső mechanizmusa rejtve marad a fölső alkalmazási réteg elől. Számára a BSW modulok szintjén történő kommunikáció összetettségéből nem látszik semmi.

3.4.2.2 RTE

Az RTE dönt az adat további útjáról, jelen forgatókönyv szerint tehát a BSW réteg felé küldi azt, a COM modulon keresztül. Ehhez a *Com_SendSignal()* vagy *Com_SendSignalGroup()* függvényeket használja.

3.4.2.3 COM



3.7. ábra: adatküldés a COM modulban

A COM modul felelős a kapott signalok, vagy signal-csoportok PDU-ba rendezéséért, valamint a küldés ütemezéséért. Amennyiben az alsóbb modulok készen állnak a küldé-

si folyamatra (a *Com_TriggerTransmit()* függvényt a PduR-en keresztül meghívja a CAN stack), a COM az elkészített I_PDU-kat továbbküldi a PduR felé.

```
PduR_ComTransmit(PduIdType, PduInfoType)
```

A *PduR_ComTransmit()* adatküldést támogató függvény bemeneti paraméterei a BSW-n belüli adatkommunikáció alapvető típusai. A *PduIdType* szolgál az üzenetek azonosítására (unsigned integer). A *PduInfoType* egy olyan struktúra, amelynek két elme van: az *SduLength* (unsigned integer) és az *SduDataPtr* (unsiged integer *). Előbbi a struktúra által tárolt bájtok számát jelenti, míg utóbbi az adatbuffer kezdőcímére mutató pointer. A PduR_ComTransmit() és még sok más függvény visszatérési értéke is *StdReturnType* típusú. Az ilyen típussal visszatérő függvények a sikeres végrehajtást *E_OK*, a sikertelent *E_NOT_OK* értékkel jelzik [5].

3.4.2.4 PDU Router

A PduR a COM modultól kapott I_PDU-t jelen forgatókönyv szerint a CAN hálózatba továbbítja. Amennyiben a küldéshez az adat szegmentálása szükséges, a CAN Transport Layeren keresztül kell küldeni. (A CanTp részletes működését a 4.1 fejezet tárgyalja.) Ellenkező esetben – ha egyetlen CAN frame-ben elküldhető adatról van szó – a PduR közvetlenül a CanIf modulnak küldi az adatot.

```
CanTp_Transmit(PduIdType, PduInfoType*)
CanIf_Transmit(PduIdType, PduInfoType*)
```

3.4.2.5 CAN Interface

Bevezetés

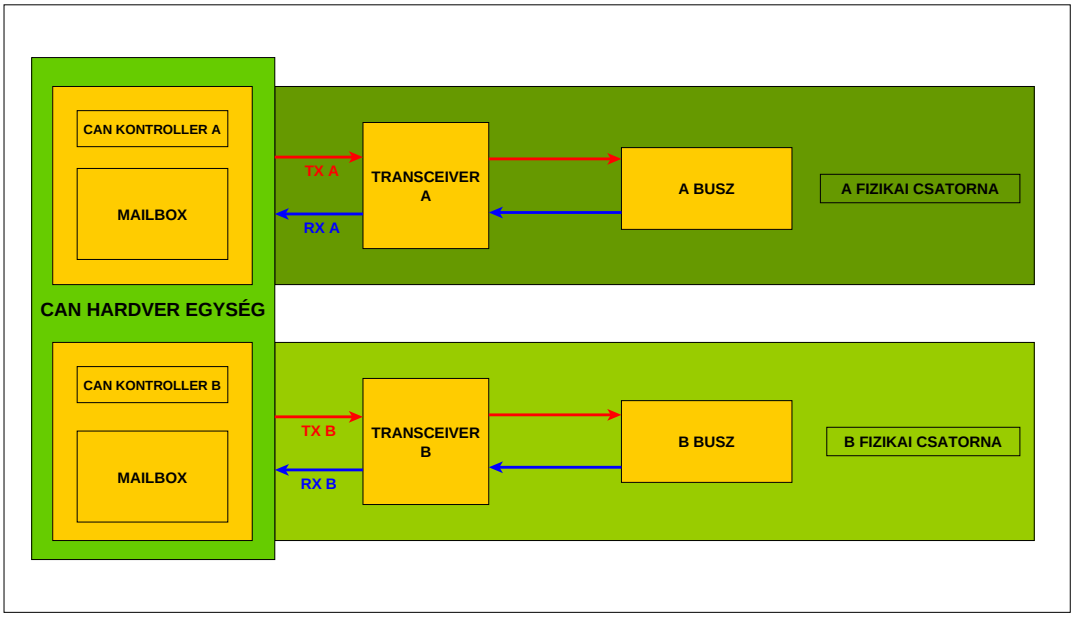
A küldés hardveres oldalának menedzselésére szolgáló modulok (CanIf, CAN Driver) működésének megértéséhez fontos a következő fogalmak tisztázása:

- **CAN hardware egység (HW UNIT)**

Egy hardver egység egy vagy több azonos típusú CAN kontrollerből áll. A hardver egységet a hozzá tartozó CAN Driver modulon keresztül lehet elérni. Különböző típusú kontrollerek esetén tehát mindegyikhez külön driver modul tartozik. A CanIf modul konfigurálásakor eltárolja a kontrollerek számát és típusát egy a kapcsolatokat nyilvántartó táblázatban (mapping table).

- **Fizikai csatorna (Physical Channel)**

Egy CAN kontroller és a CAN hálózat közötti kapcsolódási felületet testesíti meg. Egy fizikai csatornához *egy CAN kontroller és egy CAN transceiver* tartozik. A lenti ábrán egy hardver egység látható, amely egy-egy fizikai csatornához csatlakozó azonos típusú kontrollereket tartalmaz.



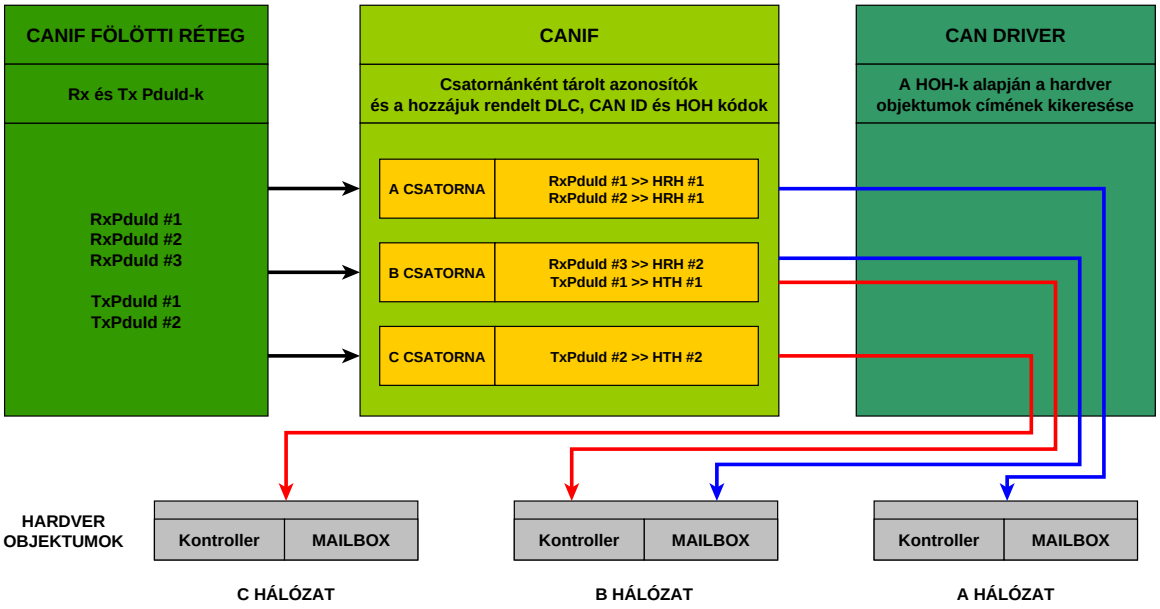
3.8. ábra: CAN hardver egység

- **HOH (Hardware Object Handle)**

Absztrakt referencia egy CAN mailbox struktúrára, amely olyan CAN-nel kapcsolatos paramétereket tartalmaz, mint a CAN ID (azonosító), az SDU hosszát leíró DLC (Data Length Code) és persze maga az adat.

- **HTH (Hardware Receive Handle):** a küldéshez tartozó HOH.
- **HRH (Hardware Receive Handle):** a fogadáshoz tartozó HOH.

A HOH-kat a CAN Driver modul függvényeinek hívásánál használjuk, és annak konfigurációja tartalmazza őket. Így a CanIf csak felhasználója ezeknek az azonosítóknak, ezáltal független marad a hardvertől. Az azonosítón keresztül éri el az egyes hardver-specifikus kommunikációs buffereket.



3.9. ábra: a CAN Interface hardver-elérése

A küldés folyamata a CanIf modulon belül

PDU-k küldése kizárólag a CanIf modulon keresztül lehetséges, direkt hozzáférése a CAN Driver modulhoz csak a CanIf modulnak van. A küldés akkor sikeres, ha a driver modul a megfelelő hardver egység bufferébe másolta a küldeni kívánt L_PDU-t.

A CAN Interface a PDU azonosítója alapján kiválasztja a CAN kontrollert, amely kapcsolódik az üzenet címzettjéhez vezető buszhoz. A mapping table segítségével megkeresi a megfelelő HTH-t. Az L_PDU-t a *Can_Write(HTH, DATA*)* függvény segítségével írja a CAN Driver bufferébe. A függvény visszatérési értékének típusa *Can_ReturnType:* ez a művelet kimenetelétől függően a *CAN_OK*, *CAN_NOT_OK* vagy a *CAN_BUSY* értékeket veheti fel.

A bufferekkel kapcsolatosan korábban már említett megfontolás miatt üzenetküldéskor követelmény, hogy egy L_PDU csak egy helyen legyen eltárolva. Ez a memória alapesetben a vonatkozó hardveregység buffere. Másik megoldásként, ha számára bufferelés engedélyezett (konfigurációfüggő), a CanIf képes tárolni a PDU-t, amennyiben azt küldéskor a CAN Driver modul elutasította (A CanWrite() CAN_BUSY-val tér vissza). Ha a CanIf belső bufferelése nincs engedélyezve, akkor a CAN mailbox foglaltsága esetén a küldési folyamat elutasításra kerül.

A CanIf_Transmit() függvény nem csak akkor tér vissza E_OK értékkel, ha a küldés ténylegesen megtörtént, hanem akkor is, amikor a CanIf modul saját bufferében (CanIfTxBuffer) eltárolta és a későbbiekben tervezi elküldeni az adott L_PDU-t. Utóbbi esetben a tényleges küldésről a garanciát a *CanIf_TxConfirmation()* callback függvény CAN Driver által történő meghívása jelenti.

A CanIfTxBuffer-ben tárolt, küldésre váró PDU-k közül felszabaduló hardver erőforrás esetén először mindig a legnagyobb prioritásút kell elküldeni, majd törölni a bufferből. Az L_PDU-kat tartalmazó bufferekhez való hozzáférést az adat konzisztencia érdekében kritikus szakaszok használatával kell védeni [7]. (A CanIf küldési funkciójával kapcsolatos kiegészítések a Függelék F1, F2 és F4 fejezetében találhatók.)

3.4.2.6 CAN Driver

A CAN Driver a megfelelő kontroller megfelelő bufferébe másolja az adatot, és vezérli a küldés folyamatát. A buszra kerülő adatkerettel a küldés folyamata a küldő ECU számára befejeződik.

3.4.2.7 Megerősítés

A sikeres küldés nyugtázására a *CanIf_TxConfirmation(PduIdType)* callback függvény szolgál. A CAN Driver küldi a CanIf-nek, amely továbbítja az azonosítóhoz tartozó PDU eredeti küldőjének. A CAN stack valamennyi rétegén keresztül jut el a nyugtázás a COM modulig. Ehhez minden réteg meghívja a fölötte található réteg TxConfirmation() függvényét.

3.4.3 Adatfogadás

3.4.3.1 CAN Driver

Az adatfogadás folyamata meglehetősen hasonlít a küldéshez, a kommunikációs modulok gyakorlatilag a küldés leírásánál bemutatott feladataik inverz műveleteit végzik el. A legfigyelemreméltóbb talán a CanIf modul szerepköre, így annak részletes bemutatása indokolt.

A CAN hálózaton keresztül az ECU-ba beérkező üzenetek hardverfüggő tulajdonságokat mutatnak, a CAN Driver és a CanIf együttműködése teszi őket a fölsőbb rétegek számára hardvertől függetlenül elérhetővé. A megfelelő formátumra hozás, a *normalizálás* alapvetően a CAN Driver feladata. Amennyiben szükséges (ezt az adott

kontroller által használt adatszerkezet szabja meg), a beérkező adatot a CAN Driver eltárolja saját átmeneti tárolójában. A CanIf modul a számára eljuttatott, fogadott adatról már nem látja, hogy a CAN Driver bufferéből vagy egyenesen a hardvertől érkezik felé. A lényeg, hogy a CanIf_RxIndication() függvény meghívásakor normalizált L_PDU-t vár [8].

3.4.3.2 CAN Interface

A *CanIf_RxIndication(HRH, CAN ID, DLC, DATA*)* callback függvény jelzi a CanIf modul számára egy új adatcsomag érkezését. A függvény bemeneti paraméterei azonosítók *(HRH, CAN ID)*, az adat hosszúságot jelölő *DLC* és az adatot tartalmazó *bufferre mutató pointer* (ld. a Függelék F3 és F5 fejezetében).

Az adatok konzisztenciájának biztosítása érdekében a CAN hardver egység mindaddig zárva marad, amíg a CanIf_RxIndication() függvény vissza nem tér. Amennyiben az adatok átmeneti tárolása szükséges (normalizáció céljából), a hardver csak addig lesz zárva, amíg a másolás az átmeneti bufferbe meg nem történik, ez után készen áll újabb PDU fogadására.

Szoftveres szűrés, DLC ellenőrzés

A hardveren keresztül sikeresen fogadott PDU-k nem mindegyike kerül feldolgozásra, hiszen köztük akadhatnak olyanok, amelyek nincsenek definiálva egy adott ECU konfigurációjában. A szoftveres szűrés eredményeképpen a HRH és CAN ID paraméterekből jutunk el az L_PDU-khoz.

Ha a CanIf megtalálja a CanIf_RxIndication() függvényben paraméterként kapott azonosítót az adott L_PDU HRH-jához a konfiguráció során hozzárendelt CAN ID-k között, akkor az L_PDU elfogadásra kerül. Az erre szolgáló keresési algoritmus implementációfüggő. Ezután történik meg az adathosszúság (DLC) ellenőrzése, amennyiben a funkció engedélyezve van [7].

A CanIf miután előkészítette a PDU-t (szoftveres szűrés, DLC ellenőrzés), továbbküldi azt, a fölsőbb réteg RxIndication() függvényének meghívása által:

```
CanTp_RxIndication(PduIdType, PduInfoType*)
PduR_CanIf_RxIndication(PduIdType, PduInfoType*)
```

3.4.3.3 COM

A PDU Router modulba az információ tehát közvetlenül, vagy a CanTp közbeiktatásával jut el. Ez utóbbi akkor szükséges, amikor több keretben érkezik egy nagyobb, összetartozó adategység, így annak kontrollált újraegyesítése szükséges. A PduR a *Com_RxIndication(PduIdType, PduInfoType*)* callback hívásával adja át a COM modulnak a hozzáférést a fogadott adat bufferéhez. A COM modul saját bufferébe másolja az adatot, majd saját ütemezése szerint signal, vagy signal-csoport formájában továbbítja az RTE-n keresztül az alkalmazási réteg felé.

```
Com_ReceiveSignal() vagy Com_ReceiveSignalGroup()
```

Ezzel az adatfogadás BSW réteget érintő része befejeződik. Az adat a cél komponens megfelelő portján keresztül jut el rendeltetési helyére.

4 A CAN Transport Layer modul tervezésének bemutatása

4.1 Bevezetés

Ha egyetlen mondatban kellene megfogalmazni a PduR és CanIf között elhelyezkedő CanTp feladatát, akkor a 8 bájtnál hosszabb PDU-k szegmentálását és a másik oldalon történő újraegyesítését említenénk. A PDU Router szerepe a protokollválasztás mellett annak eldöntésére is kiterjed, hogy adott I_PDU küldésekor szükséges-e a CanTp modul használata. A CAN Interface (CanIf) fizikai megvalósításuktól és elhelyezkedésüktől függetlenül elérhetővé teszi a CAN hardver eszközöket és a CAN buszt. Fogadáskor az N_PDU azonosító alapján dönt a PduR-nek történő közvetlen küldésről, vagy ellenkező esetben a CanTp közbeiktatásáról.

Az AUTOSAR szoftverarchitektúrában betöltött szerepe szerint a következő szolgáltatások köthetők a CAN Transport Layerhez ([10]):

- küldéskor az adat részekre bontása, szegmentálása
- fogadáskor az adatelemek egyesítése
- az adatáramlás irányítása
- szegmentáláskor keletkező hibák detektálása
- folyamatban lévő küldési vagy fogadási folyamat felfüggesztése, visszavonása

Az AUTOSAR szabvány létrehozásakor törekedtek arra, hogy az egyes BSW modulok specifikációi már meglévő szabványok követelményrendszerét is felhasználva készüljenek el. Ennek megfelelően a CAN Transport Layer alapjául az ISO 15765 nemzetközi szabvány szolgál (ld. 4.1.1 fejezet).

Az ISO szabvány rövid ismertetése után a modul általam megvalósított implementációjának bemutatása következik. A leírás tehát nem feltétlenül a szabvány logikai felépítését követi, hanem a modul funkcionális egységeinek belső működése szerint tárgyalja az alapvető tevékenységeket.

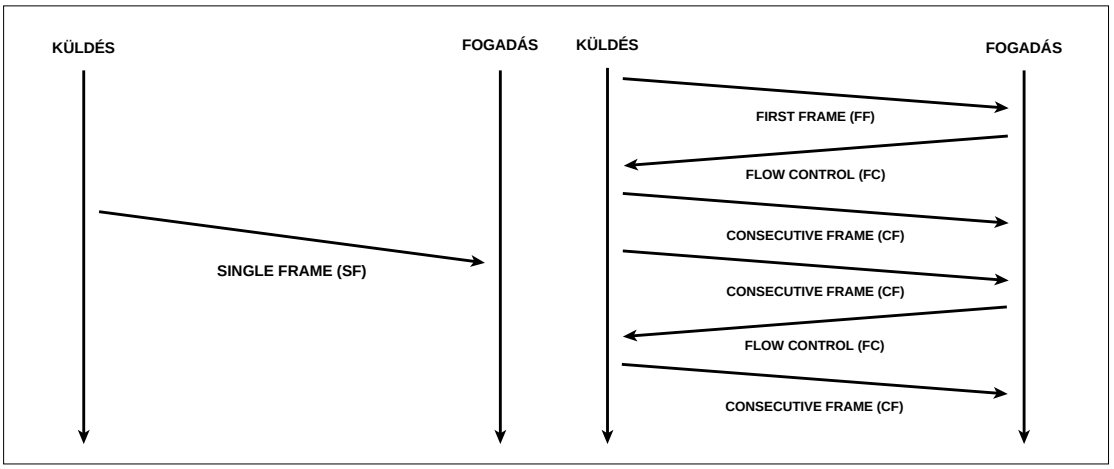
4.1.1 Az ISO 15765-2 szabvány

4.1.1.1 A szabvány felépítése

A szabványt azért hozták létre, hogy a CAN protokoll alapján működő diagnosztikai rendszerekre vonatkozó követelményeket összefoglalják (Road vehicles – Diagnostic communication over Controller Area Network (DoCAN)). Emellett természetesen minden olyan CAN-alapú rendszerre vonatkozhat, amelyben szükséges a transport layer protokoll használata. A szabvány négy nagy részből áll:

1. General information and use case definition (Alapvető információ és felhasználási területek)
2. Transport protocol and network layer services (OSI rétegek leírása)
3. Implementation of unified diagnostic services (UDS on CAN) (Egységesített diagnosztikai szolgáltatások megvalósítása CAN-en keresztül)
4. Requirements for emissions-related systems (Károsanyag kibocsátással kapcsolatos követelmények)

A CAN Transport Layer kapcsán felmerülő kérdéseket a 2. rész tisztázza [11]. A Network Layer alapvetően két fajta szolgáltatást nyújt a fölsőbb rétegek számára: a kommunikációs szolgáltatásokat és a protokoll paramétereinek beállítására vonatkozó szolgáltatásokat. A belső működéséhez köthető fő tevékenységek az üzenetek szegmentálása, a folyamvezérléssel támogatott adatküldés, és a fogadott adatkeretek újraegyesítése. Üzenetek küldésekor a keretformátumnál nagyobb méretű adatblokkok részekre osztva, több keret formájában jutnak át a CAN hálózaton. Fogadáskor az érkező framekből állítható össze egy szintén nagyobb méretű adatcsomag.



4.1. ábra: szegmentálás nélküli és szegmentált kommunikáció

A kommunikáció során felhasznált szegmentálási és időzítési paraméterek egy része állítható (*BS, STmin*), ehhez nyújt lekérdezési (read) és állítási (set) függvényeket a network layer. A transport layer protokoll funkciói az üzenetek küldése vagy fogadása (4095 adatbájt méretig) és a sikeresen vagy sikertelenül befejezett kommunikáció észlelése, jelentése más rétegek számára.

4.1.1.2 N_PDU

A CanTp modulon belül a kommunikáció alapegysége az N_PDU. Egy N_PDU három részre osztható:

- N_AI: Address Information (címzéssel kapcsolatos információs mező)
- N_PCI Protocol Control Information (a protokoll működéséhez szükséges információk)
- N_DATA (maga az átküldendő adat)

4.1.1.3 Az adatkeretek típusai

Az adat mennyiségétől és a PDU funkciójától függően megkülönböztetünk *SF (Single Frame), FF (First Frame), CF (Consecutive Frame)* és *FC (Flow Controll)* PDU-kat.

A Single Frame – címzési módtól függően – 6-7 adatbájtnál nem hosszabb üzenet. Az ezt meghaladó adatmennyiség esetén több adatkeret formájában történik a küldés (multiple frame transmission). Ez utóbbi esetben az első átküldött frame (FF) tartalmazza az adathosszúságot, és a küldendő üzenet első részletét, az ezt követő keretekben (CF) pedig a további adatok átvitele történik. A CF-ek számozva vannak. A fo-

gadási oldalon a számok ellenőrzésével és eltérés esetén hibajelenéssel biztosítható az összetartozó adatelemek helyes sorrendben történő átvitele és újraegyesítése.

Fogadáskor a befogadó buffer kapacitása szerint vezérelhető a túloldali küldés folyamata, lehetőség van az adatfolyam irányítására (Flow Controll). A küldő figyelembe veszi a fogadó bufferelési lehetőségeit és a FC-nak megfelelően ütemezi a további adatkeretek küldését.

4.1.1.4 PDU-kitöltés (padding)

Egy N_PDU maximális mérete 8 bájt. Amennyiben ennél kevesebb tényleges információt hordozó bájtot tartalmaz, a PDU lehet rövidebb is, de konfigurációtól függően a hasznos információ ki is egészülhet ún. paddingbájtokkal. Ez esetben minden PDU 8 bájt hosszúságú lesz.

4.1.1.5 Címzés

A CAN Transport Layer által alkalmazott címzési módok:

- normál
- kiterjesztett (extended)
- vegyes (mixed)

A címzéssel kapcsolatos információk a CAN frame N_AI mezőjében találhatók. Normál címzés esetén a teljes címzéshez elegendő a mező maximálisan 29 bites mérete. A másik két címzési mód esetén az adatmező első bájtja is hordoz címzéssel kapcsolatos információkat: ez a terület kiterjesztett címzési mód esetén az N_TA (Target Address), míg vegyes címzés esetén az N_AE (Address Extension) értékét veszi fel.

4.1.1.6 Az N_PCI mező

Az alábbi táblázat foglalja össze az egyes frametípusokra jellemző N_PCI mezőket.

N_PDU típus	N_PCI byte-ok			
	byte #1		byte #2	byte #3
	7-4. bit	3-0. bit		
Single Frame (SF)	típus: 0	SF_DL		
First Frame (FF)	típus: 1	FF_DL		
Consecutive Frame (CF)	típus: 2	SN		
Flow Control (FC)	típus: 3	FS	BS	STmin

4.2. ábra: PCI mezők az egyes kerettípusok esetén

Az első bájt fölső 4 bitje minden esetben a keret típusát jelzi (SF 0, FF 1, CF 2, FC 3). A SF egyetlen bájtból álló PCI mezője a típust jelző fölső 4 bitből és a Single Frame adathosszát jelző, az alsó 4 biten elhelyezkedő *SF_DL (Single Frame Data Length)* kódból áll. Az SF_DL értékkészlete 1-7. A FF-hez 2 bájtos PCI mező tartozik. Az első bájtból az alsó 4 bit és a teljes második bájt hordozza a *FF_DL (First Frame Data Length)* információt, melynek értéke 0x7 és 0xFFF között változik. Ez az érték jelenti az adott kommunikációs folyamat adatbájtjainak teljes hosszát (FF és az azt követő CF-k). A CF egyetlen PCI bájtjának alsó 4 bitje az *SN (sequence number)* értéket tartalmazza. Ennek segítségével ellenőrizhető, hogy az egymást követő CF-k megfelelő sorrendben érkeztek-e: az SN mező a küldő oldalon tulajdonképpen a kiküldött CF-k sorszámozására szolgál. A fogadó oldal ellenőrzi, hogy mindig a soron következő CF érkezik-e, és eltérés esetén hibát jelez. (Az SN érték 15 után túlcsordul, és 0-val újrakezdődik a sorszámozás.) A FC PDU adatmezőt nem tartalmaz, csak 3 bájt PCI-t. Az első bájt alsó 4 bitje az *FS (Flow Status)* érték, amely az FC típusát jelzi. A második és harmadik bájton a *BS (Block Size)* és *Stmin (minimal Separation Time)* értékek találhatók.

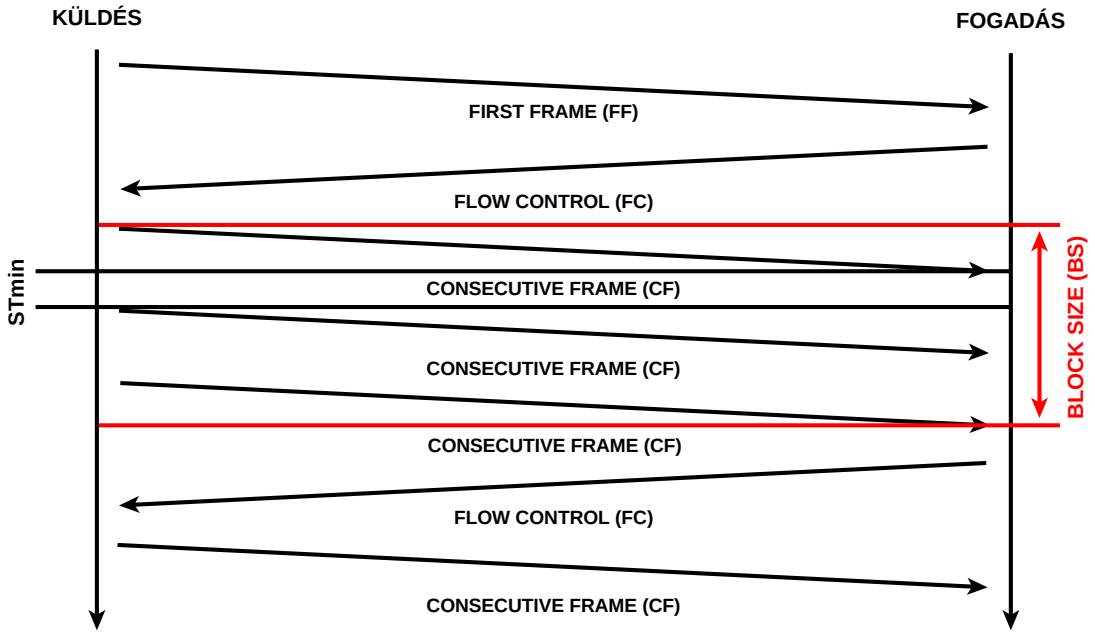
Az FS lehetséges három értéke:

- 0: CTS (ContinueToSend) – a vevőegység készen áll további keretek fogadására
- 1: WAIT – a vevő a küldés átmeneti szüneteltetését kéri, mivel nem áll rendelkezésére megfelelő bufferkapacitás
- 2: OVFLW (OverFlow) – túlcsordulás jelzése

Az FS mező más értéket nem vehet fel.

A BS érték a küldő számára jelzi, hogy a vevő oldal – bufferkapacitásának függvényében – mekkora egységekben képes fogadni az érkező adatkereteket. Egy blokk egy vagy több keretből állhat. A BS érték az egy blokkba tartozó keretek számát jelöli. A küldő folyamatosan, FC nélkül ennyi adatkeretet küld el, majd vár, amíg a vevő újabb CTS FC-t nem küld számára. Speciális esetként, ha a vevő képes az összes adat azonnali tárolására, a küldő fél blokkokra bontás nélkül, a kereteket folyamatosan küldheti. (Ezt jelzi a BS 0-ás értéke.) Az STmin (Separation Time minimum) a fogadó egység által mért, két CF fogadása közötti időintervallum. Értéke 0x00 és 0x7F között ms-ban értendő (0-127ms), míg a 0xF1-0xF9 közötti sáv a 100 és 900 µs közötti értékeknek felel meg. Más értékeket az STmin nem vehet fel.

A CanTp specifikáció definiálja, hogy egy kommunikációs folyamat során a BS és STmin érték nem változtatható. (Kivétel ez alól az utolsó blokk mérete, amikor már nem feltétlenül áll rendelkezésre annyi átküldendő adat, hogy kitöltsön egy teljes blokkot.)



4.3. ábra: a BS és STmin értékek

4.1.1.7 Konfiguráció

Az AUTOSAR modulok testreszabhatósága, konfigurálhatósága alapvető implementációs követelmény. A konfigurációs paraméterek az ECU szoftverének különböző élet-

szakaszaiban változtathatók. Ennek megfelelően léteznek fordítási idő (*compile time*) előtt változtatható, a lefordított állományok összeszerkesztése (*link time*) előtt változtatható és futási időben változtatható (after *build time*) konfigurációs paraméterek. A konfigurációs osztály (configuration class) egy adott paraméter esetén azt az időszakot határozza meg, amelyben az adott paraméter még változtatható *(Pre-compile time, Link time, Post-build time).*

A sokszor meglehetősen nagy mennyiségű és bonyolult, egymásba ágyazott struktúrákban elhelyezkedő paraméterek tárolására használt logikai egység a *container*. A containerek tartalmazhatnak al- azaz ún. *subcontainereket*, melyek akár többszörös multiplicitással is létrehozhatók. A C nyelvben írt modulok esetén a containerek praktikusan struktúráknak feleltethetők meg, ezek alcontainerei a struktúra egyik adattagjaként definiált, egy másik struktúrára (az alcontainerre) mutató pointeren keresztül érhetők el. Amennyiben egy containerhez több azonos típusú alcontainer tartozik (tehát az alcontainer többszörös multiplicitású), az eredeti containerből a subcontainerekre mutató pointerek tömbjének megfelelő indexű tagján keresztül lesz elérhető egy adott subcontainer. A szabvány minden egyes modul specifikációjában külön fejezetet szentel a konfigurációs paraméterek leírásának. Az egyes modulok működését, felépítését jól jellemzi a hozzájuk tartozó konfigurációs struktúra, ezért a működés bemutatását ennek ismertetésével kezdem.

4.2 A CAN Transport Layer modul konfigurációs struktúrája

4.2.1 Konstans paraméterek

A fordítási időben már rendelkezésre álló, egyszerű paramétereket preprocesszor direktívák (#define-ok) segítségével konstansként definiálhatjuk. Ezek között az egész modulra jellemző azonosítók, flagek és globális információt hordozó paraméterek találhatók meg:

- diagnosztikai célú azonosítók, verziószámok:

  CANTP_MODULE_ID

  CANTP_INSTANCE_ID

  CANTP_VENDOR_ID

  CANTP_SW_MINOR_VERSION

CANTP_SW_MAJOR_VERSION

CANTP_SW_PATCH_VERSION

- egyes funkciók feltételes fordítását állító flagek:

  CANTP_DEV_ERROR_DETECT

  CANTP_CHANGE_PARAMETER_API

  CANTP_READ_PARAMETER_API

  CANTP_VERSION_INFO_API

- a padding során használt, az adatmezőt 8 bájtossá kiegészítő bájtok értékét rögzítő konstans (CANTP_PADDING_BYTE)

A CAN Transport Layer modul egyszerre több kommunikációs kapcsolat menedzselésére képes. A kommunikáció ún. csatornákon (*channel*) keresztül zajlik, amelyek csak a CanTp modulon belül elérhető, virtuális kommunikációs közeget jelentenek. Minden egyes N_SDU statikusan kapcsolódik valamelyik ilyen csatornához és csakis egyetlen csatornához. A csatorna tehát egy belső útvonalat testesít meg, amelyen az N_SDU küldése, vagy fogadása zajlik. Egy csatornához több N_SDU is kapcsolódhat.

Egy csatorna lehet *FULL DUPLEX* és *HALF DUPLEX*: előbbin egyszerre lehetséges fogadás és adatküldés, míg az utóbbin csak felváltva.

A konstansként definiált konfigurációs paraméterek között kaphat még helyet a kommunikációs csatornák számát megadó *CANTP_CHANNEL_NO* és az egyes csatornákhoz kapcsolódó N_SDU-k maximális számát megadó *CANTP_MAXID* paraméter.

4.2.2 A modul inicializálása

A CanTp modul működéséhez inicializáció szükséges. A konfigurációs paraméterek átadása egy konfigurációs struktúrára mutató pointeren keresztül történik az inicializáló függvény segítségével.

```
CanTp_Init(CanTp_ConfigType* CfgPtr)
```

4.2.3 A konfigurációs típus (CanTp_ConfigType)

A konfigurációs struktúra típusdefinícióját a modul készítője hozza létre a szabványban leírtak alapján, kiegészítve az egyéni implementációs döntései következményeiként szükségessé váló paraméterekkel. A rendszer konfigurálásakor a konfigurátor generálja

a struktúra egy példányát, amelyre a *CfgPtr* pointerrel mutat. Ezt a pointert bocsátja a modul rendelkezésére az ECU State Manager (EcuM) által meghívott *CanTp_Init()* függvény.

A konfigurációs struktúra tartalmazza az egyes csatornákhoz rendelt struktúrákra mutató pointerek tömbjét, valamint a CanTp main függvényének periódusidejét:

```
typedef struct
{
    float32 CanTpMainFunctionPeriod;
    CanTp_ChannelType* CanTp_Channel[CANTP_CHANNEL_NO];

} CanTp_ConfigType;
```

A csatornákhoz rendelt, fogadott és küldött N_SDU-k paramétereit a *CanTp_TxNSduType* és *CanTp_RxNSduType* típusú konténerek tartalmazzák, minden egyes N_SDU-ra külön-külön. A CanTp_ChannelType az N_SDU-k konfigurációs struktúráira mutató pointerek tömbjeit tartalmazza, valamint az adott csatorna Tx és RxNSDU-inak számát meghatározó *CanTp_TxNSduLength* és *CanTp_RxNSduLength* paramétereket. Emellett definiálja az adott csatorna kommunikációs módját is (HALF vagy FULL DUPLEX) a *CanTpChannelMode* paraméter segítségével.

```
typedef struct
{
    CanTpChannelModeType CanTpChannelMode;

    uint8 CanTp_TxNSduLength;
    uint8 CanTp_RxNSduLength;
    CanTp_TxNSduType* CanTp_TxNsdu[CANTP_MAXID];
    CanTp_RxNSduType* CanTp_RxNsdu[CANTP_MAXID];

} CanTp_ChannelType;
```

A TxNSduType a küldéskor, az RxNsduType a fogadás folyamán használt legfontosabb paramétereket tartalmazza az összes csatorna minden egyes definiált N_SDU-jához.

A legfontosabb TxNSdu paraméterek:

- azonosítók (TxNSduId, TxNPduId, RxFcNPduId): az adott üzenet azonosítója a fölsőbb (NSdu), és alsóbb (NPdu) réteg irányába, valamint az üzenethez tartozó FC azonosító
- időzítéshez szükséges paraméterek: CanTpNas, CanTpNbs, CanTpNcs (ld. 4.4.2)
- CanTpTxAddressingFormat: az adott N_SDU-nál alkalmazott címzési mód

- CanTpTxPaddingActivation: az adott N_SDU-nál alkalmaznak-e PDU-kitöltést (ld. 4.1.1.4)
- címzéshez kapcsolódó kiterjesztések (N_TA, N_AE)

A legfontosabb RxNSdu paraméterek:

- azonosítók (RxNSduId, RxNPduId, TxFcNPduId) (az adott üzenet azonosítója a fölsőbb (NSdu), és alsóbb (NPdu) réteg irányába, valamint az üzenethez tartozó FC azonosító)
- időzítéshez szükséges paraméterek: CanTpNar, CanTpNbr, CanTpNcr és CanTpRxWftMax (ld. 4.4.3)
- CanTpRxAddressingFormat: az adott N_SDU-nál alkalmazott címzési mód
- CanTpRxPaddingActivation: az adott N_SDU-nál alkalmaznak-e PDU-kitöltést

Az egymásba ágyazott konfigurációs struktúrának köszönhetően, könnyen hozzáférhető bármilyen paraméter. Ha például az i-edik csatorna j-edik Rx N_SDU-jának címzési módjára van szükségünk, az alábbi módon tudjuk lekérdezni:

```
CfgPtr-> CanTp_Channel[i]->CanTp_RxNsdu[j]-> CanTpRxAddressingFormat
```

4.3 A CanTp modul működése

4.3.1 A csatornák állapotának kezelése

Az adatküldés során megvalósuló szegmentálás és a fogadáskor történő adat-újraegyesítés folyamata a korábban már említett virtuális csatornákon történik. A működés során a csatornák aktuális küldési és fogadási aktivitását belső állapotváltozó-tömbök tárolják:

```
CanTpRxState[CANTP_CHANNEL_NO]

CanTpTxState[CANTP_CHANNEL_NO]
```

Az állapotváltozók a *CANTP_RX_PROCESSING* és *CANTP_RX_WAIT* valamint a *CANTP_TX_PROCESSING* és *CANTP_TX_WAIT* értéket vehetik fel, attól függően, hogy van-e aktív küldés, illetve fogadás az adott csatornán. Az azonos indexhez tartozó Rx és Tx állapotok egy bizonyos csatornára vonatkoznak. HALF DUPLEX csatorna

esetén a *CanTpRxState* és *CanTpTxState* állapotváltozók közül csak az egyik, míg FULL DUPLEX csatornánál mindkettő lehet PROCESSING állapotú.

Inicializáláskor a modulhoz definiált összes csatorna mind fogadási, mind küldési állapota WAIT értéket vesz fel. Küldés vagy fogadás esetén az aktuális csatorna PROCESSING állapotúra vált, majd a kommunikációs folyamat végeztével ismét WAIT értékkel jelzi, hogy felszabadult. A folyamat elindítása előtt mindig vizsgálni kell, hogy az adott csatorna szabad-e. (FULL DUPLEX esetben elég, ha az aktuális irányban szabad (pl. küldés).) Az állapotváltozók írását és olvasását kritikus szakaszok alkalmazásával védeni kell.

4.3.2 Adatfogadás

4.3.2.1 Bevezetés

Az adatfogadás teljes folyamatának átlátásához szükséges a szomszédos modulok egy-egy függvényének megismerése, valamint a CanTp legfontosabb belső számlálóinak definiálása.

- **PduR_CanTpStartOfReception(id, TpSduLength, bufferSizePtr)**

A fogadási folyamat elején a PduR-en keresztül történő buffer-igénylésre szolgál. Az *id* azonosítóval rendelkező N_SDU-knak *TpSduLength* méretű bufferre van szükségük. Amennyiben a buffer rendelkezésre áll, a függvény *BUFREQ_OK* értékkel tér vissza és a *bufferSizePtr* által mutatott memóriaterületre az elérhető bufferkapacitás értékét írja bájtban.

- **PduR_CanTpCopyRxData(id, infoptr, retry, bufferSizePtr)**

Adatmásolásra szolgáló függvény. A PduR az *id* azonosítóval rendelkező N_SDU-knak lefoglalt helyre másolja az *infoptr* struktúrán keresztül átadott adatot. A *bufferSizePtr*-en keresztül jelzi, hogy mekkora a rendelkezésre álló buffer a további másolásokhoz. (A retry paramétert a CanTp modul nem használja.)

- **PduR_CanTpRxIndication(id, result)**

A függvény segítségével értesíthető a fölsőbb réteg a sikeres vagy sikertelen fogadási folyamatról.

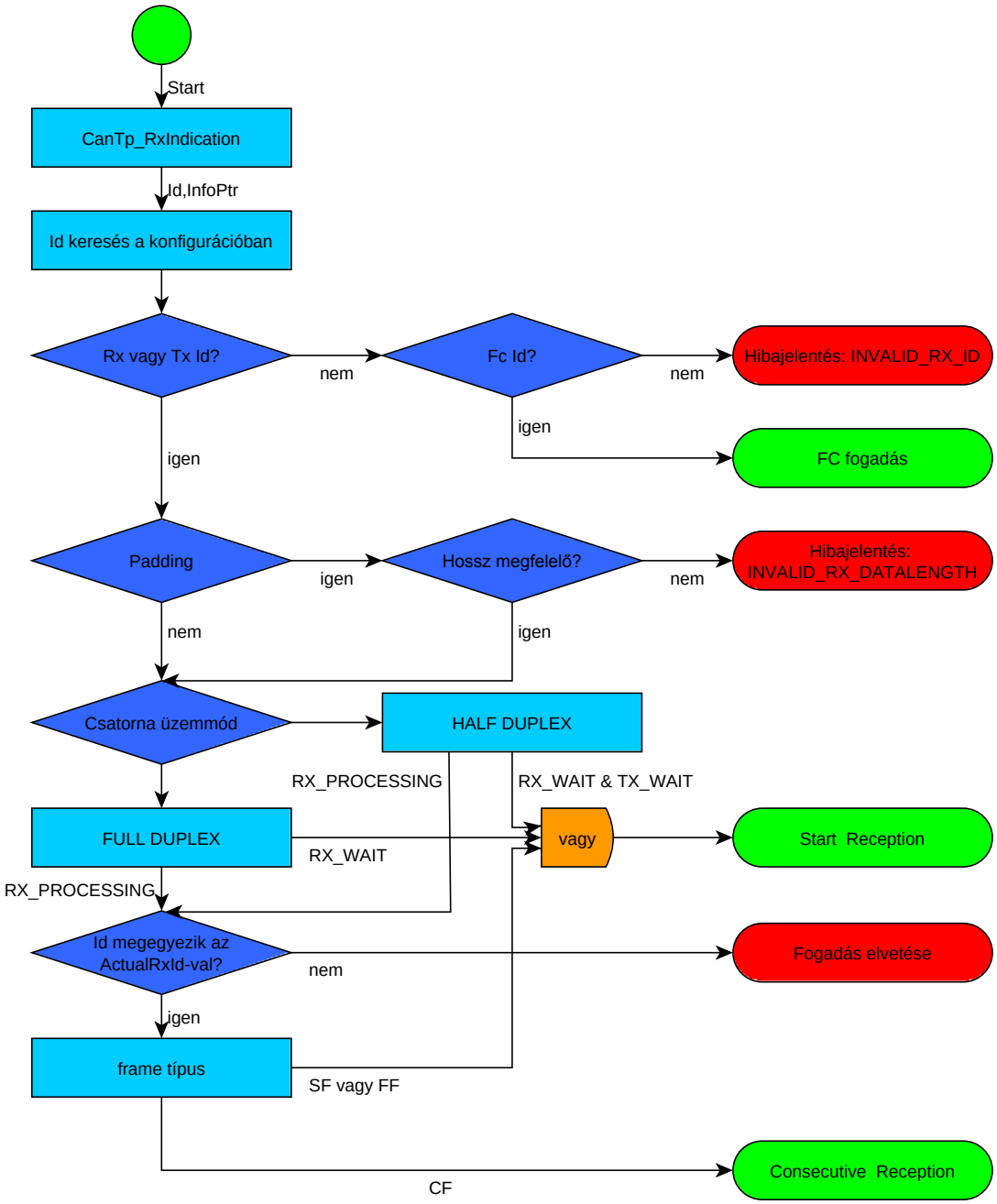
- **CanIf_Transmit(id, infoptr)**

A függvény az alsóbb rétegek felé irányuló adatküldésért felelős. Az *id* azonosítójú N_PDU-kat az *infoptr* mutatón keresztül teszi elérhetővé. Az adatfogadás során a függvényre a Flow Controll framek küldéséhez van szükség.

Az alábbi változók és számlálók nem szerepelnek a modul specifikációjában, implementációs döntés eredményeképpen kerültek bevezetésre:

- **CanTp_ActualRxPduId:** egy adott csatornán aktuálisan zajló kommunikációhoz tartozó azonosító.
- **CanTp_ActualBS:** az adott folyamatra egyénileg számolt blokkméret.
- **RDL (Remaining Data Length):** a fogadási folyamat elején a teljes adathossz rendelkezésre áll (SF_DL, FF_DL). A további fogadás során a CanTp nyilvántartja, hogy a teljes folyamatra vonatkoztatva mennyi a hátralevő feldolgozandó adatbájtok száma. Ennek aktuális értékét jelzi a RDL.
- **BSC (Block Size Counter):** A fogadás blokkokban történik. Az aktuális blokkból hátralevő CF-ek számát a BSC szolgáltatja. Értéke minden érkező CF után csökkentendő, 0 esetén új blokk kezdődik. Ekkor a BSC a CanTp_ActualBS értékkel frissül.
- **SNC (Sequence Number Counter):** A küldő oldalon a CF-ek számozására szolgáló SN értékeket a fogadó félnek ellenőriznie kell. A SNC aktuális értéke azt adja meg, hogy milyen számmal kellene érkeznie a soron következő CF-nek.

4.3.2.2 CanTp_RxIndication (CanTpRxPduId, CanTpRxPduPtr)



4.4. ábra: CanTp_RxIndication

A CanIf modultól érkező kommunikáció egyetlen interfésze a *CanTp_RxIndication(CanTpRxPduId, CanTpRxPduPtr)* függvény. Az ezen keresztül átadott N_PDU lehet SF, FF, CF, és FC. Az RxIndication függvény mindezek fogadását kell, hogy támogassa, így meglehetősen szerteágazó tevékenység jellemzi.

A *CanTpRxPduId* a feldolgozásra váró üzenet azonosítója, a *CanTpRxPduPtr* paraméter a beérkező adatot tartalmazó struktúrára mutató pointer.

Az RxIndication függvény első lépésként megkeresi a konfigurációs struktúrában a beérkezett azonosítót.

```
CfgPtr-> CanTp_Channel[i]->CanTp_RxNsdu[j]-> CanTpRxNPduId
```

Találat esetén az indexeket (i és j) a CanTp eltárolja, és az adatfogadás folyamata során ezek segítségével nyeri ki a szükséges egyéb paramétereket a konfigurációs struktúrából. Ha a fogadandó üzenetek azonosítói (Rx) között találja meg, az adatfogadás ágán, amennyiben a beérkező FC-k azonosítói közt, a FC feldolgozás ágán halad tovább a függvény. Ha az azonosítót nem találja, hibajelzéssel visszatér (*CANTP_E_INVALID_RX_ID*).

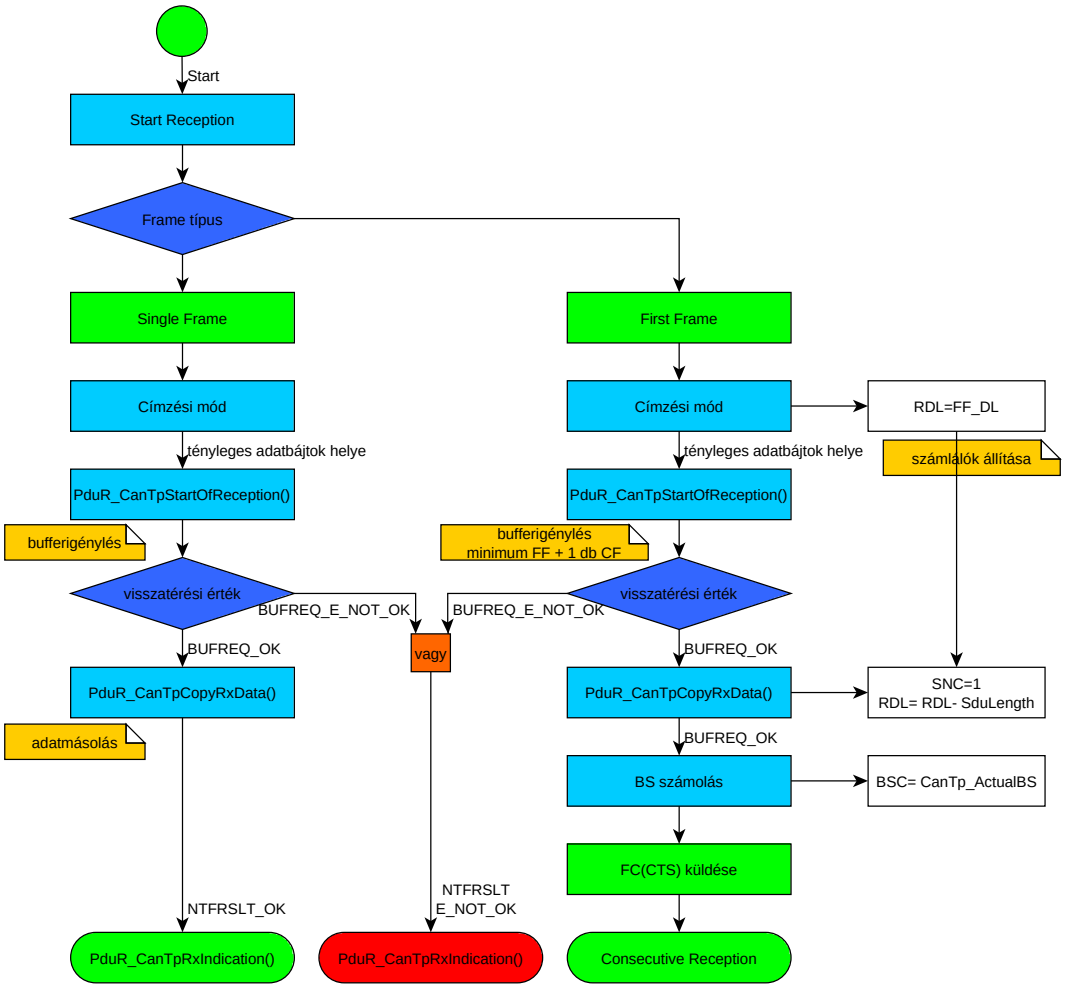
Ha az üzenetfogadás ágán haladunk tovább, a következő lépés a PDU hosszának ellenőrzése. Bekapcsolt PDU-kitöltés (padding) mellett az adathosszúság fixen 8 bájt, kikapcsolt padding esetén a címzési módnak (normál, extended vagy mixed) és a keret típusának (SF vagy CF) megfelelően kerül ellenőrzésre, hogy az érkezett PDU rendelkezik-e a minimális adathosszúsággal. Ellenkező esetben a hibajelzés *CANTP_E_INVALID_RX_LENGTH*.

Ha tehát a konfigurációban szereplő azonosítóval van dolgunk, amelynek adathosszúsága is az értelmezhető tartományba esik, nincs más dolgunk, mint eldönteni, hogy az adott PDU-hoz tartozó csatornán lehetséges-e a kommunikáció.

FULL DUPLEX csatorna esetén, ha a csatorna fogadási állapotát jelző CanTpRxState állapotváltozó RX_WAIT állapotú, új kommunikációs folyamat kezdődhet (*StartReception*). Ha a beérkező PDU SF vagy FF, az CanTpRxNPduId lesz a csatorna aktuális azonosítója (*CanTp_ActualRxPduId*). HALF DUPLEX csatorna esetén ehhez a CanTpTxState TX_WAIT állapota is szükséges. Amennyiben a csatornán adatfogadás zajlik (RX_PROCESSING) a kommunikáció csak akkor lehetséges, ha a függvény bemeneti CanTpRxNPduId paramétere megegyezik a tárolt aktuális értékkel, egyéb esetben a folyamat elutasításra kerül. Egyezőség esetén tehát lehetséges a kommunikáció: SF és FF esetén új folyamat kezdődik azonos azonosítóval és az előző elutasításra kerül (StartReception). CF esetén folytatódik a már zajló adatfogadás (*ConsecutiveReception*).

4.3.2.3 StartReception

Az adatfogadás eltérő módon zajlik SF és FF érkezésekor.



4.5. ábra: fogadási folyamat kezdete

Single Frame fogadás

SF érkezése esetén (ld. F6) a címzési mód megállapítása után az érkezett PDU-ból kinyerhető az adathosszúság (*SF_DL*), valamint a fölsőbb rétegnek továbbküldendő hasznos adat (*SF_DataPtr*). A címzési mód annyiban érdekes számunkra, hogy ennek ismerete biztosítja, hogy megtaláljuk a PCI mező első bájtjának helyét. Ebből már beazonosítható az adatkeret típusa, valamint a szükséges hosszinformáció. Ez után történik a buffer igénylése a fölsőbb rétegtől.

```
PduR_CanTpStartOfReception(CanTpRxNPduId, SF_DL, bufferSizePtr)
```

Ha rendelkezésre áll a kért buffer, a StartOfReception függvény *BUFREQ_OK*-val tér vissza. Ekkor megtörténhet az adatmásolás.

```
PduR_CanTpCopyRxData(CanTpRxNPduId, SF_DataPtr, retry, bufferSizePtr)
```

Amennyiben ez sikeresen megtörténik, a *PduR_CanTpRxIndication()* függvénnyel jelezzük a fölsőbb rétegek felé hogy a fogadási folyamat véget ért. (Sikeres fogadás esetén NTFRSLT_OK, sikertelen fogadás esetén *NTFRSLT_E_NOT_OK* az eredmény.)

```
PduR_CanTpRxIndication(CanTpRxNPduId, NTFRSLT_OK)
```

First Frame fogadás

Több adatkeret küldése esetén változik a helyzet (ld. F7): a bonyolultabb folyamat menedzseléséhez FC framek küldése is szükségessé válik, hogy a küldő tudja, mikor és mennyi keretet küldhet számunkra.

A címzési mód megállapítása után kinyert FF_DL érték a teljes szegmentált üzenet hosszát jelzi, ezért a FF-t követő CF framek fogadásakor is szükség van rá. A *FF_DL* értéket tehát eltároljuk az *RDL* számlálóban.

```
PduR_CanTpStartOfReception(CanTpRxNPduId, FF_DL, bufferSizePtr)
```

Amennyiben a *StartOfReception()* függvény *BUFREQ_OK*-val tér vissza, nem lényeges számunkra, hogy a bufferSizePtr által mutatott, a rendelkezésre álló bufferkapacitást jelző érték megegyezzen FF_DL-el. Minimális követelményként első körben a FF-hez tartozó adatbájtokat és még egy CF adatbájtjait kell tudnia eltárolni a fölsőbb réteg bufferének. Amennyiben legalább ekkora méretű buffer rendelkezésre áll, megkezdődhet az adatmásolás.

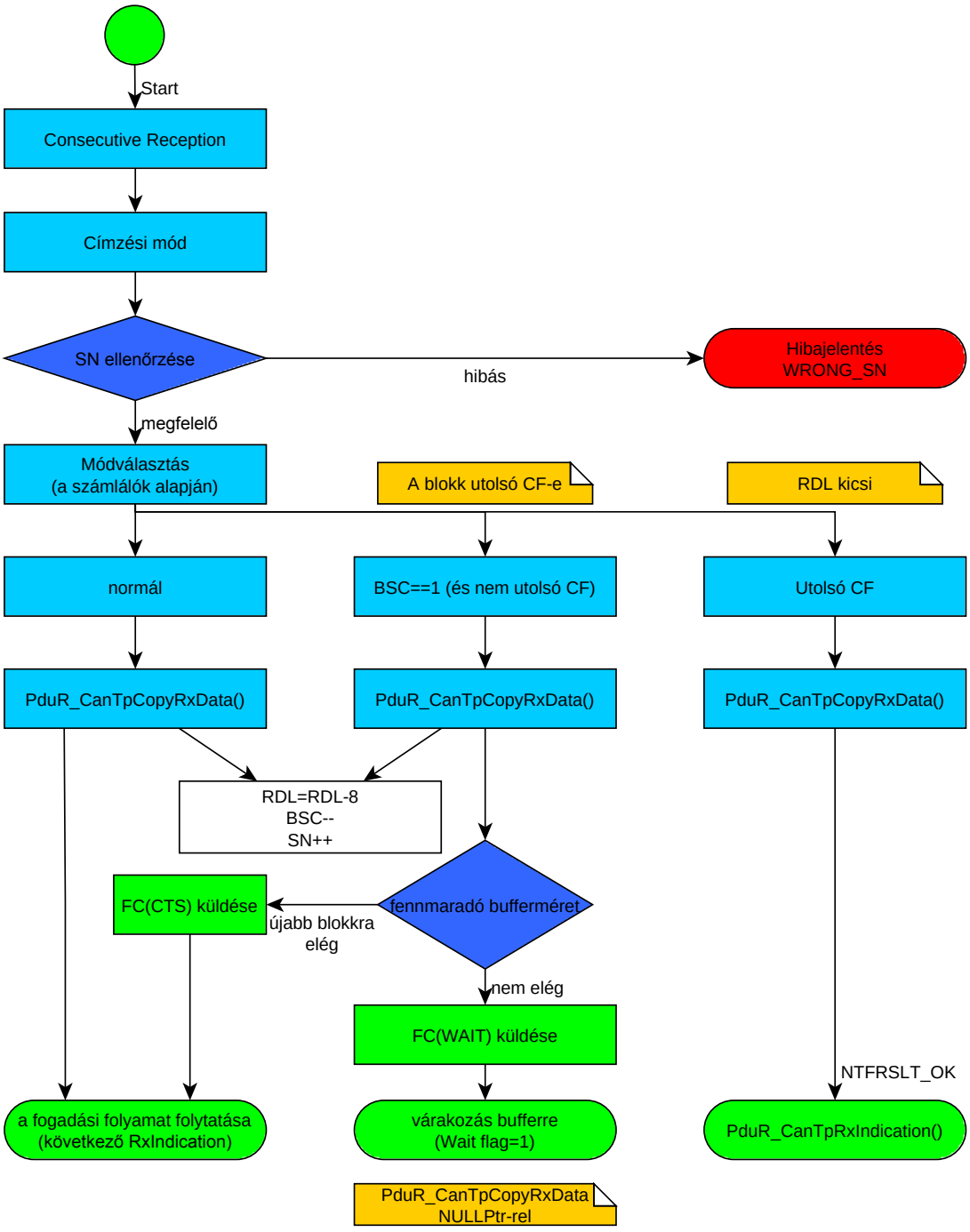
```
PduR_CanTpCopyRxData(CanTpRxNPduId, FF_DataPtr, retry, bufferSizePtr)
```

Ez után történik a fogadást támogató számlálók beállítása. Az *RDL számlálót csökkentjük* a már továbbküldött adat hosszával (RDL = RDL – FF_DataPtr->SduLength). Az *SNC-t elindítjuk*, a következő CF-t már az 1-es SN értékkel várjuk majd.

A blokkméret számítása a FC framek küldése miatt fontos. A FF fogadása után még rendelkezésre álló bufferkapacitás, a még elküldésre váró adatmennyiség és az adott címzési mód által befolyásolt CF adathosszúság (normál esetben 7, kiterjesztett és vegyes címzésnél 6 bájt) határozza meg. A kiszámolt blokkméretet (*CanTp_ActualBS*) adjuk értékül a *BSC* számlálónak.

A CanTp CTS Flow Controll küldésével jelez az alsóbb rétegeknek hogy készen áll a további adatfogadásra.

4.3.2.4 Consecutive Reception



4.6. ábra: az adatfogadás folyamatának folytatása

CF fogadásakor az első lépés mindig az *SN-érték ellenőrzése*. Amennyiben a CanTp eltérést tapasztal a várt (SNC által számított) és tényleges (A CF PCI mezőjéből kiolvasható) SN között, hibát jelez (*NTFRSLT_E_WRONG_SN*).

Ha az SN érték megfelelőnek bizonyult, három ágon folytatódhat a CF feldolgozása:

1. Normál esetben az adatok fogadására szolgáló *buffer nincs megtelve, és az aktuális CF nem az utolsó* a fogadás során (RDL mutatja). A BSC 0-tól eltérő értéke jelzi, hogy az aktuális blokkból még van hátra fogadandó CF.

```
PduR_CanTpCopyRxData(CanTpRxNPduId, CF_DataPtr, retry, bufferSizePtr)
```

A CF adatbájtjainak másolása után a számlálók állítása következik:

- a megmaradt adathosszúság csökkentése
- (RDL = RDL – CF_DataPtr->SduLength)
- a blokkból hátralevő CF-k számának aktualizálása (BSC--)
- az SN számláló frissítése (SNC++), illetve az SN=15 értéknél a túlcsordulás biztosítása (SNC++ helyett SNC=0)

2. Ha egy blokk utolsó CF PDU-ja érkezik, a normál esetnél leírtak elvégzése után mindenképpen meg kell vizsgálni a *buffer kapacitását* (bufferSizePtr). Annak megfelelően, hogy a rendelkezésre álló memória elegendő-e még egy további teljes blokk fogadására, vagy nem, a FC framet CTS vagy WAIT értékkel kell elküldeni.

Ha a buffer megtelt, további buffer igénylése szükséges a fölső rétegtől. Ez esetben egy *Waitflaget* billentünk be, az adott csatornára vonatkozóan, így a CanTp_RxIndication() függvény visszatérhet, a további teendőket a CanTp modul main függvénye végzi el.

```
PduR_CanTpCopyRxData(CanTp_ActualRxPduId, NULLPtr, retry, bufferSizePtr)
```

A *PduR_CanTpCopyRxData* speciális felhasználása *bufferigénylésre* szolgál (a szokásos adatstruktúra helyett *NULL-pointer* a második paraméter). A main függvény addig hívogatja így a függvényt, amíg sikerül egy, a következő blokk méretének megfelelő buffer foglalása. Ez idő alatt a fogadó félnek WAIT FC-k küldésével jelzi, hogy még nem áll készen a fogadásra. Amikor végül elérhető a kívánt buffer, a main függvény törli a Waitflaget és CTS Flow Controll framet küld az alsó rétegnek.

3. Ha az aktuális CF az utolsó a teljes folyamatban (RDL értéke alapján), a PduR_CanTpCopyRxData() függvény sikeres meghívása után jelezni kell a fölsőbb rétegek felé, hogy a folyamat véget ért.

```
PduR_CanTpRxIndication(CanTpRxSduId, NTFRSLT_OK)
```

4.3.2.5 FC küldés

A FC framek küldése a CanIf_Transmit() függvénnyel valósul meg. Az adatküldés résznél leírtakhoz hasonlóan történik a PDU bájtjainak összeállítása, struktúrába rendezése és elküldése.

4.3.3 Adatküldés

4.3.3.1 Bevezetés

Küldés során a következő más modulokhoz tartozó függvények kerülnek felhasználásra:

- **PduR_CanTpCopyTxData(id, infoptr, retry, bufferSizePtr)**

Az *id* azonosítójú, küldeni kívánt N_SDU-t másolja a CanTp által átadott *infoptr* struktúrába.

- **CanIf_Transmit(id, infoptr)**

A CanIf modul felé történő küldéshez használt függvény.

- **PduR_CanTpTxConfirmation(id, result)**

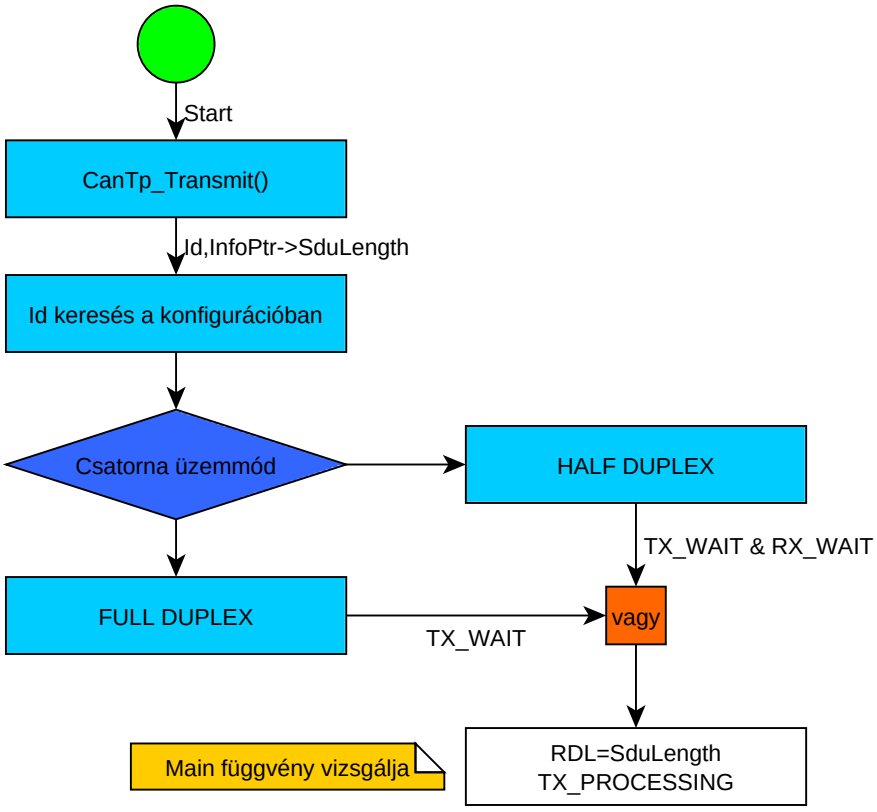
Az *id* azonosítóval elküldött N_SDU sikeres (*NTFRSLT_OK*) vagy sikertelen (*NTFRSLT_E_NOT_OK*) küldéséről szóló visszajelzés a PduR-en keresztül a fölsőbb rétegek felé.

Az adatfogadásnál bevezetett változók és számlálók megfelelője a küldéskor is szükséges: **CanTp_ActualTxPduId, CanTp_ActualBS, RDL (Remaining Data Length), BSC (Block Size Counter), SNC (Sequence Number Counter).**

A küldés és fogadás közötti fő különbség az, hogy szemben a CanTp_RxIndication() függvénnyel, amelyet az alsóbb réteg egy fogadási folyamat során minden egyes PDU-nál meghív, a küldés esetében a CanTp_Transmit() függvény csak elindítja a folyamatot. A továbbiakban a küldés folyamatáért a modul main függvénye a felelős. A *CanTp_MainFunction()* lefutásakor egy for ciklus segítségével megvizsgálja a *CanTpTxState[CANTP_CHANNEL_NO]* tömb állapotát minden egyes csatornára. Ha bármelyik csatorna *CANTP_TX PROCESSING* állapotú, meghívódik a TX taszkok menedzseléséért felelős *CanTp_MainTransmit()* függvény.

4.3.3.2 CanTp_Transmit(CanTpTxSduId, CanTpTxInfoPtr)

Amennyiben a küldés során szükséges a szegmentálás, a PduR a CanTp modul felé irányítja az I_PDU-kat. A szegmentálás folyamata a CanTp_Transmit() függvény meghívásával indul (ld. F8).

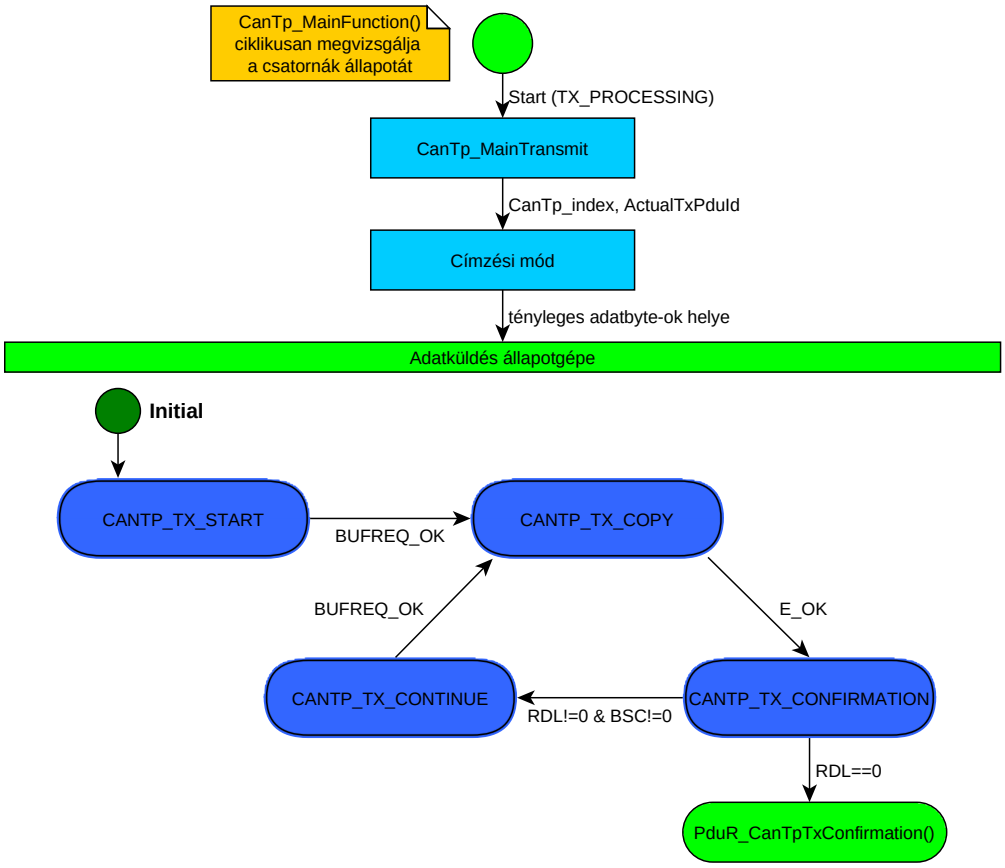


4.7. ábra: CanTp_Transmit()

A beérkező CanTpTxInfoPtr struktúrából csak a teljes küldendő adat hosszát jelölő adathosszúság paraméter *(SduLength)* kerül feldolgozásra. A függvény a beérkező *CanTpTxSduId* azonosítót megkeresi a konfigurációs struktúrában, majd megvizsgálja a hozzá tartozó csatorna foglaltsági állapotát. Amennyiben az adott csatornán lehetséges új adatküldés indítása, a *CanTpTxState* állapotváltozót *CANTP_TX_PROCESSING* állapotúra változtatja, az adott azonosítót rögzíti az *ActualTxPduId* változóban és elmenti a *CanTpTxInfoPtr->SduLength* adathosszúság értéket a *RDL* számlálóba. Innentől kezdve szerepel a modul nyilvántartásában, hogy az adott csatornán küldés zajlik, valamint elérhető a szegmentálásra váró üzenet azonosítója, és hossza.

4.3.3.3 CanTp_MainTransmit

A CanTp_MainTransmit függvény nem szerepel a szabványban. Létrehozása implementációs döntés, amely lehetővé teszi a küldéssel kapcsolatos tevékenységek jól elkülöníthető funkcionális egységbe való foglalását. A függvény a CanTp_MainFunction() *CanTp_index* ciklusváltozóját használja a konfigurációs struktúra és más felhasznált állapotváltozó- vagy számlálótömbök indexeléséhez. Így éri el az aktuálisan feldolgozott csatornán zajló küldés menedzseléséhez szükséges összes szükséges információt.



4.8. ábra: a küldési folyamatok állapotgépe

Amikor a main függvény meghívja a CanTp_MainTransmit() függvényt, csak a konfigurációs struktúra csatornákat tartalmazó tömbjének indexeléséhez szükséges CanTp_index áll rendelkezésre. A CanTp_TxNsdu containerek tömbjének indexe (*TxNsdu_index*) a korábban mentett *ActualTxPduId keresésével* állítható be.

```
CfgPtr->CanTp_Channel[CanTp_index]->CanTp_TxNsdu[TxNsdu_index]->
```

Ezzel elérhetővé válik az összes szükséges konfigurációs paraméter. Ezután a CanTp_MainTransmit() függvény egy átmeneti bufferben összeállítja a küldeni kívánt

N_PDU-t. Első lépésként a címzési mód beazonosítása után a PDU *első adatbájtjába* másolja a *címzési kiegészítéseket* (N_TA, N_AE).

Mivel a küldés teljes folyamata általában több main ciklusra is kiterjed, mindig fontos tudni, hogy éppen hol tart, amikor a ciklikusan meghívásra kerülő main függvény átmenetileg felfüggeszti a folyamatot. Ennek megfelelően a legegyszerűbb és egyben legszemléletesebb megoldásként állapotgépet használunk a küldési teendők menedzselésére. Ha a folyamat bármelyik állapotban félbemarad (pl. várakozás megerősítésre), a main függvény következő lefutásakor ugyanabból az állapotból kell, hogy folytatódjon a futás. Ehhez az állapotokat minden csatornára külön-külön el kell tárolni egy globális változó tömbben *(CanTp_Tx_StateVar[CANTP_CHANNEL_NO])*.

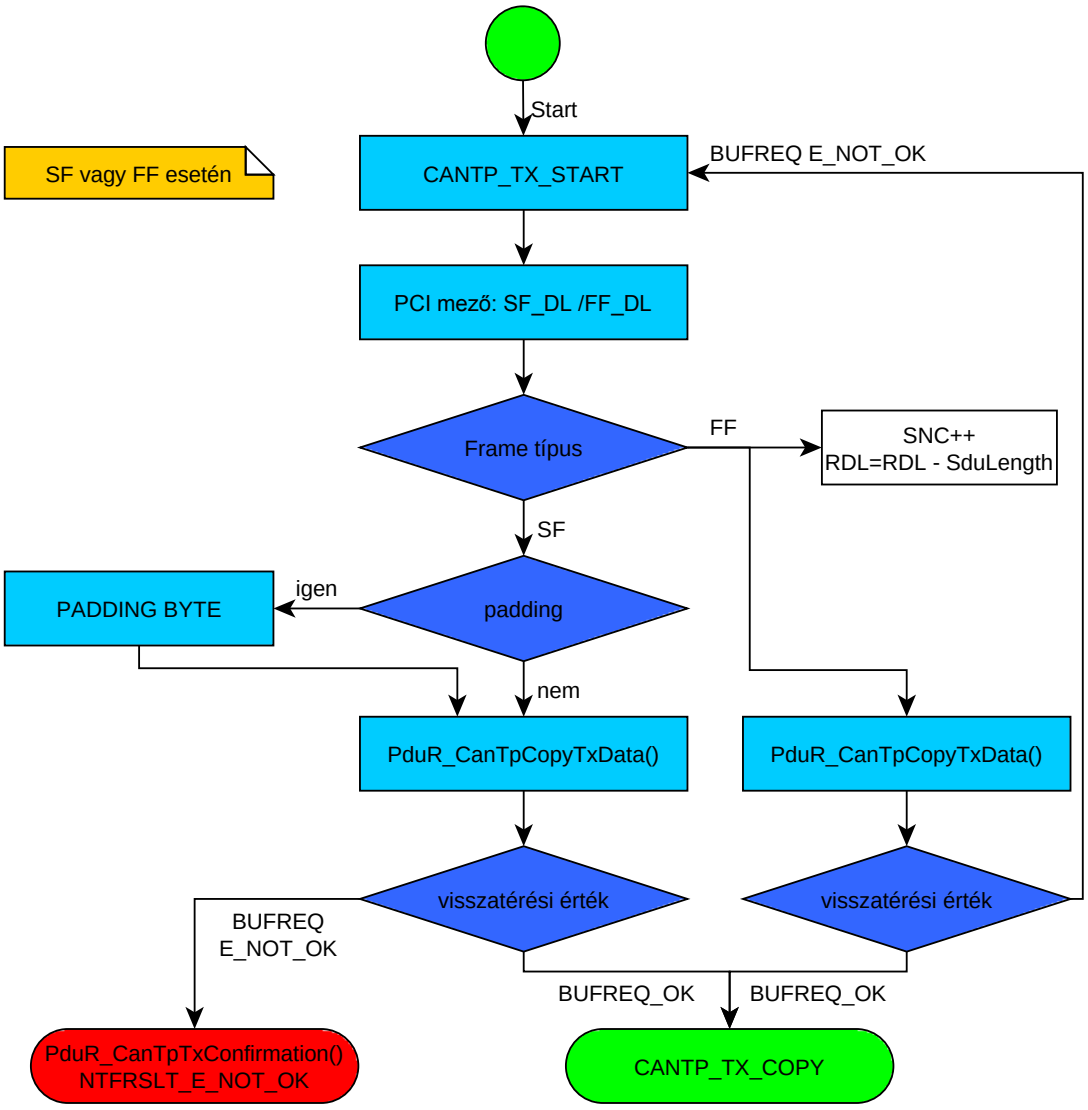
A CanTp_MainTransmit() állapotgépének állapotai és az állapotváltás feltételei

CANTP_TX_START

Az állapot a küldés folyamatának kezdetekor, így kizárólag SF és FF összeállításakor használatos. A *PCI mezők kitöltése* az első lépés, ehhez minden szükséges információt a *RDL számláló* hordoz. A még küldésre váró adatok hossza szabja meg a keret típusát. A PCI mezőben szereplő SF_DL vagy FF_DL érték pontosan megegyezik az RDL aktuális értékével (ld. 4.3.2.1).

A keret típusától és a címzési módtól függően a *PduR_CanTpCopyTxData()* függvény segítségével az átmeneti buffer valamelyik részébe másoljuk a hasznos adatot. (pl. SF esetén, ha az RDL=3 és a címzés normál módban történik, a CopyTxData-nak átadott PduInfoType típusú struktúra SduDataPtr tagja az átmeneti buffer második helyen álló bájtjának címe, míg az SduLength paraméter az RDL aktális értéke, azaz 3 lesz. FF esetében, ha a címzés kiterjesztett, a címzési kiterjesztés és a 2 bájtos PCI mező miatt az SduDataPtr-nek átadott cím az átmeneti buffer 4. bájtjának címe lesz, a hossz értéket rögzítő SduLength pedig az 5 értéket veszi fel, mivel ennyi marad a keret teljes 8 bájtos hosszából.)

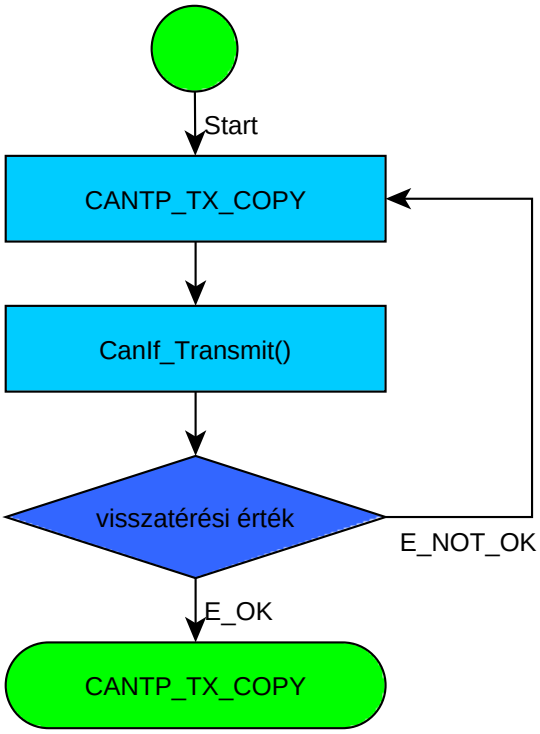
Ha az aktuális PDU First Frame, *frissíteni* kell a *számlálók értékét*. Csökken a másolandó adathossz (RDL) és nő a sorszámozásért felelős SNC értéke. SF esetében, ha a padding engedélyezett az adott N_PDU-hoz, az adattal fel nem töltött bájtok helyére a konfigurációban rögzített paddingbájt kerül.



4.9. ábra: CANTP_TX_START

A *PduR_CanTpCopyTxData()* visszatérési értéke *BufReq_ReturnType* típusú. Amennyiben a függvény *BUFREQ_OK* értékkel tér vissza, az állapotgép továbblép a *CANTP_TX_COPY* állapotra. Ezzel szemben a *BUFREQ_E_NOT_OK* visszatérési érték SF esetén a küldési folyamat végét eredményezi. A sikertelen végrehajtásról a CanTp a *PduR_CanTpTxConfirmation* függvénnyel értesíti a fölsőbb réteget (*NTFRSLT_E_NOT_OK*). FF küldésekor a modul újabb lehetőséget biztosít az adatmásolásra, tehát az állapotgép egyszerűen a CANTP_TX_START állapotban marad. A CanTp fejlett időzítési rendszere (ld. 4.4) biztosítja, hogy a modul hosszabb ideig ne rekedhessen meg egy adott küldési állapotban (foglalva ezzel az adott csatornát). Egy bizonyos időzítő lejárta után a küldési folyamat elutasításra kerül.

CANTP_TX_COPY

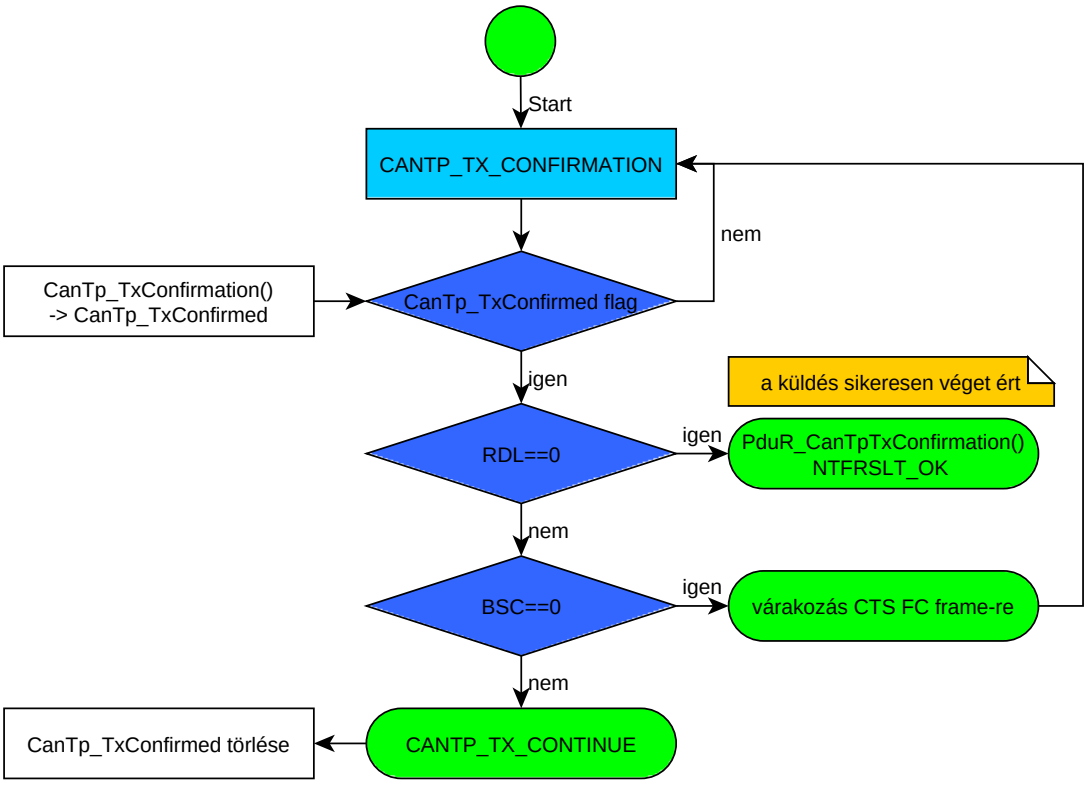


4.10. ábra: CANTP_TX_COPY

A CANTP_TX_COPY állapotban történik meg az aktuális N_PDU elküldése a CAN Interface modul számára. Erre a célra szolgál a *CanIf_Transmit(id, infoptr)* függvény. Az *id* azonosítóval rendelkező üzenet *infoptr* által mutatott PduInfoType struktúrája a korábban összeállított PDU-t tartalmazó átmeneti buffer kezdőcíméből és teljes hosszból áll. A buffer tartalmazza az esetleges címkiterjesztés mellett a PCI- és adatmezőt.

A függvény E_OK visszatérési értéke lépteti az állapotgépet a *CANTP_TX_CONFIRMATION* állapotba.

CANTP_TX_CONFIRMATION



4.11. ábra: CANTP_TX_CONFIRMATION

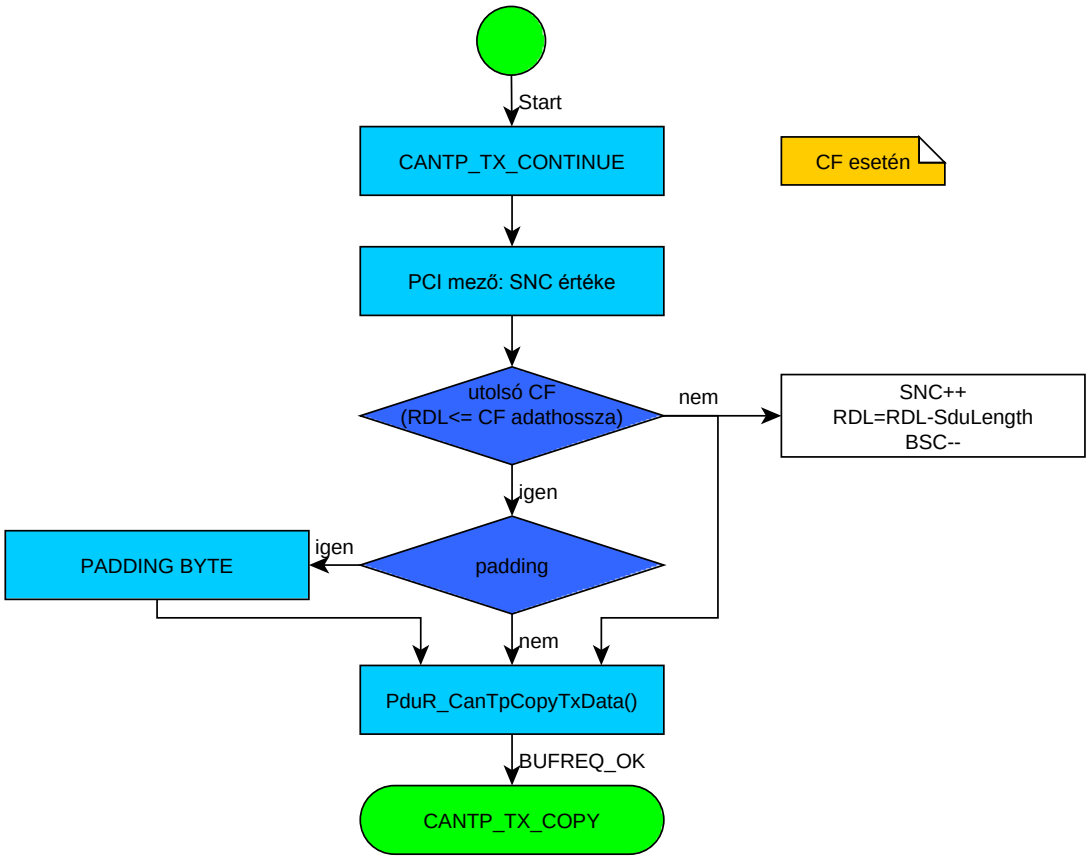
A sikeres küldésről szóló visszaigazolást a CanIf a *CanTp_TxConfirmation(id)* callback függvény meghívásával jelzi a CanTp modulnak. Ezt a CanTp nem képes azonnal feldolgozni, hiszen a küldési folyamat a main lefutásához kötött. Ezért a TxConfirmation() függvény bebillent egy, az adott csatornára vonatkozó, ún. *CanTp_TxConfirmed flaget*, amely a legközelebbi main híváskor kerül feldolgozásra. A flag vizsgálata és kezelése történik a CANTP_TX_CONFIRMATION állapotban.

Ha az *RDL értéke 0,* akkor az utolsó adatcsomag is kiküldésre került, így a megerősítés egyben a teljes folyamat sikeres végét is jelenti. Ez esetben a CanTp meghívja a *PduR_CanTpTxConfirmation()* függvényt a *NTFRSLT_OK* értékkel.

Ha az *RDL 0-tól különböző* értékű, akkor még szükség van további adatszegmensek küldésére. Ehhez az állapotgép a *CANTP_TX_CONTINUE* állapotba lép, amennyiben teljesül a feltétel, mely szerint a *BSC nem 0* értékű. Ellenkező esetben egy blokk küldése befejeződött, így a további küldéshez a fogadási oldalról származó CTS FC érkezése szükséges.

CANTP_TX_CONTINUE

Ebben az állapotban történik a Consecutive Framek előállítása.



4.12. ábra: CANTP_TX_CONTINUE

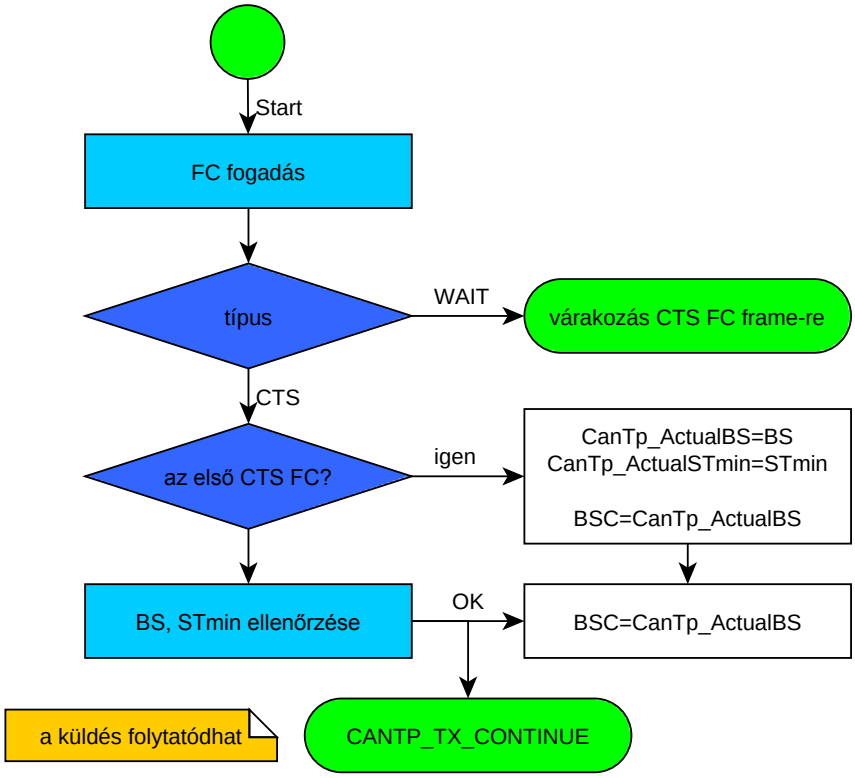
CF küldése esetén a PCI mező kitöltéséhez az SNC számláló aktuális értéke szükséges. Ezt követően – a SF és a FF esetéhez hasonlóan – a *PduR_CanTpCopyTxData()* függvény segítségével az adatbájtok az átmeneti buffer megfelelő memóriaterületére kerülnek. Ha az aktuális az *utolsó CF* (ez az RDL érték alapján megállapítható), meg kell vizsgálni, hogy engedélyezett-e a *padding*, és amennyiben igen, az esetlegesen üresen maradó bájtok helyét fel kell tölteni a paddingbájt értékével.

Ha *nem az utolsó Consecutive Frame*-mel van dolgunk, *állítani* kell a *számlálók* értékét (RDL = RDL – SduLength, SNC++, BSC--). A CopyTxData *E_OK* visszatérési értéke vezérli az állapotgépet a *CANTP_TX_COPY* állapotba.

Innentől az állapotgép működése a már ismertetett állapotokat bejárva folytatódik.

4.3.3.4 FC érkezése

A CanTp_RxIndication() függvény tárgyalásánál láttuk, hogy az N_PDU-k egységesen ezen az interfészen érkeznek a CanTp modulba. A SF-ek, FF-ek és CF-ek esetét már részletesen elemeztük az adatfogadás folyamatánál, a FC érkezése az adatküldéshez kötődik, így most kerítünk rá sort.



4.13. ábra: FC fogadása

A Flow Controll framek az adatküldés folyamatát vezérlik. A CanTp_MainTransmit() függvény állapotgépe a *CANTP_TX_CONFIRMATION* állapotban várja a FC-k érkezését, abban az esetben, ha a BSC számláló értéke 0. Ez ugyanis egy blokk végét jelzi, amely után a további küldéshez CTS FC frame fogadása szükséges. Az állapotgép mindaddig a CANTP_TX_CONFIRMATION állapotban marad, amíg a BSC számláló értéke 0. CTS FC érkezésekor a modul a *BSC* számlálót az adott küldési folyamatra számított BS értékre állítja, így megkezdődhet egy újabb blokk küldése (CANTP_TX_CONTINUE). A WAIT értékű FC (további) várakozásra utasítja a CanTp modult, mivel a BSC értékét 0-n hagyja, nem módosítja.

Az első CTS érkezésekor a CanTp elmenti az adott csatornára a kapott BS és az időzítéshez szükséges STmin értékeket (CanTp_ActualBS, CanTp_ActualSTmin). Mivel egy adott küldési folyamat során a BS és STmin nem változhat, a később érkező összes CTS feldolgozása azzal kezdődik, hogy a CanTp ellenőrzi, hogy a mindenkori BS és Stmin megegyezik-e a mentett értékekkel. Hiba esetén a küldési folyamat elutasításra kerül.

4.3.4 A CanTp modul további szolgáltatásai

A CAN Transport Layer modul az eddig bemutatott szolgáltatások támogatására rendelkezik még egy-két további, az eddigieknél lényegesen egyszerűbb függvénnyel.

- **CanTp_Shutdown(void)**

A kikapcsoló függvény hatására az inicializálás előtti állapotba kerül a CanTp. Minden folyamatban lévő kommunikációs folyamat elutasításra kerül, az erőforrások felszabadulnak. A CanTp *újraélesztésére* a *CanTp_Init()* függvény meghívása szolgál.

- **CanTp_CancelReceive(id), CanTp_CancelTransmit(id)**

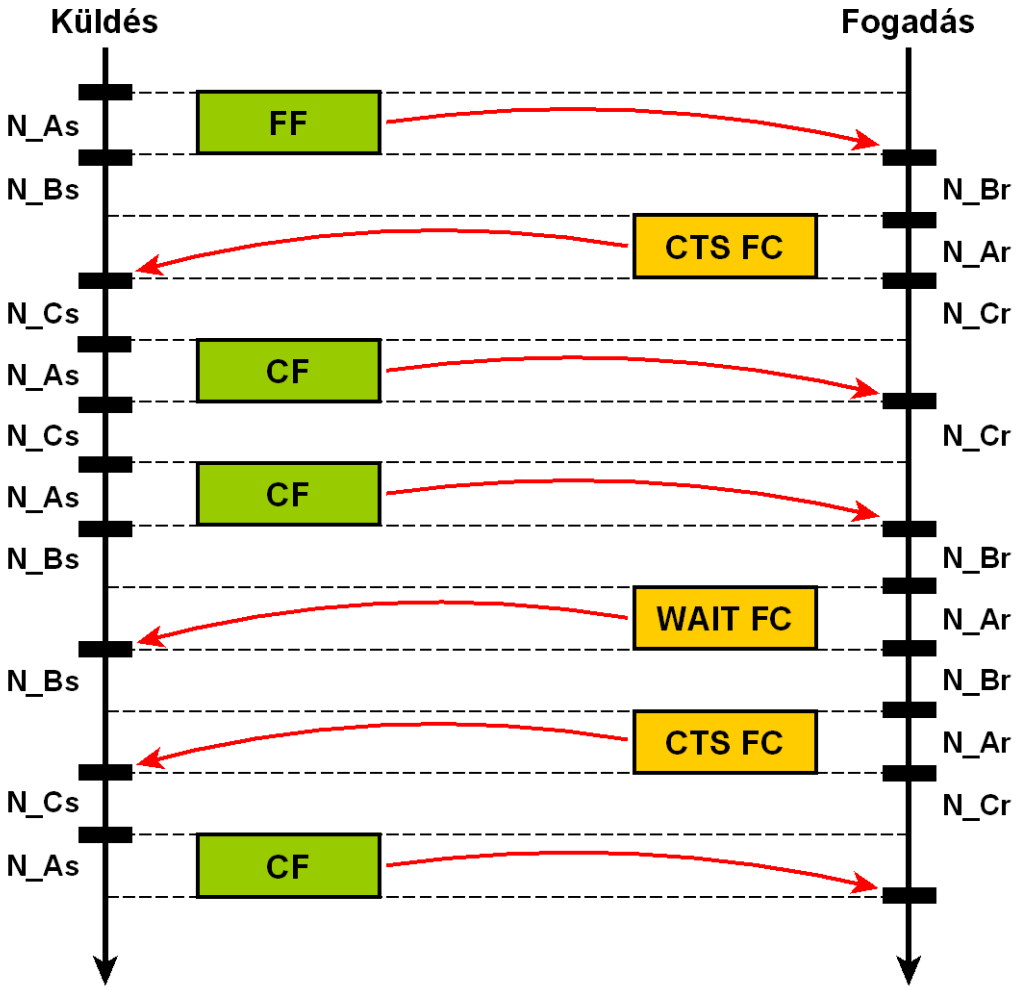
A függvények az adott *id*-hoz tartozó, folyamatban lévő küldést vagy fogadást szüntetik meg.

- **CanTp_ChangeParameter(id, parameter, value)**
- **CanTp_ReadParameter(id, parameter, value)**

Az *id* azonosítójú N_SDU-hoz rendelt *BS* és *STmin* (parameter) konfigurációs paraméterek *kérdezhetők le*, illetve *változtathatók* más értékűre (value).

A függvények implementációja függ a *CANTP_CHANGE_PARAMETER_API* és *CANTP_READ_PARAMETER_API* konfigurációs paraméterektől: kikapcsolt állapotú makrók esetén nem fordulnak bele a modul kódjába.

4.4 Időzítési kérdések



4.14. ábra: a CanTp időzítése

4.4.1 Az időzítés alapja

A szegmentálás és újraegyesítés folyamán előforduló különböző részfolyamatok megfelelő időzítése fontos követelménye az ISO 15765-2 szabványnak. Mind küldő, mind fogadó oldalon folyamatos az időzítés: ha egy mért időtartomány lejár, rögtön követi azt egy másik. Ez egyszerűvé teszi az időmérésre szolgáló számlálók kezelését. Az *időzítés alapja* a CanTp_MainFunction() periódusideje, amely konfigurációs paraméterként áll rendelkezésre (*CanTpMainFunctionPeriod*). Az összes többi, a konfigurációban a Tx és Rx N_SDU-khoz rendelt *időzítési paraméter* (N_As, N_Bs, N_Cs, N_Ar, N_Br, N_Cr) a CanTpMainFunctionPeriod *egész számú többszöröse*.

Minden csatornához létezik egy Rx és egy Tx számláló (*CanTp_Receiver_TMR, CanTp_Sender_TMR*), amelyek aktív fogadás vagy küldés esetén folyamatosan futnak: minden egyes main lefutásra értékük eggyel csökken. Az egyes időzítési paraméterekhez köthető események bekövetkezésekor a számláló a soron következő időzítési paramétert kapja értékül, majd ha időben bekövetkezik a következő esemény, akkor az azutáni paramétert és így tovább. A számláló *normál működés* esetén tehát soha *nem nullázódik*, amíg egy Rx vagy Tx taszk fut. Ha azonban bármely számláló *1-0 átmenetet* produkál (ezt a CanTp main függvénye észleli), az azt jelenti, hogy a CanTp egy esemény bekövetkeztére a konfigurációban rögzített, még tolerálható értéknél többet kényszerült várni. Ilyenkor az a CanTp időzítési hibára hivatkozva (*time-out*) felfüggeszti az aktuális küldési vagy fogadási folyamatot, és ezt jelenti a fölsőbb rétegeknek a *PduR_CanTpTxConfirmation()* függvény segítségével.

4.4.2 Az időzítéshez kapcsolódó konfigurációs paraméterek

A küldő oldalon:

- N_As: a PDU elküldésére szolgáló idő a küldés kezdetétől az érkezésről szóló megerősítésig (TxConfirmation)
- N_Bs: a FC előtti utolsó érkezett adatkeret megerősítésétől (TxConfirmation) a következő FC érkezésének jelzéséig eltelt idő
- N_Cs: a CTS FC érkezésétől a következő CF küldéséig eltelt, vagy két CF küldése között eltelt idő

A fogadó oldalon:

- N_Ar: a fogadó oldalon a FC átküldésének ideje
- N_Br: egy adat fogadása és a FC kiküldése közötti, vagy két FC küldése közötti idő
- N_Cr: a CTS FC küldésétől a következő CF érkezéséig eltelt idő

Hibás működés esetén, ha az időzítő lejár, a CanTp küldésnél a PduR_CanTpTxConfirmation(), fogadáskor a PduR_CanTpRxIndication() függvény segítségével jelenti a timeout-jelenséget (*NTFRSLT_E_TIMEOUT_A, NTFRSLT_E_TIMEOUT_BS, NTFRSLT_E_TIMEOUT_CR*).

4.4.3 A WAIT framek maximális száma

Bufferre való várakozás esetén a fogadó fél WAIT FC framek küldésével kéri a másik oldalt a küldés átmeneti felfüggesztésére. A maximálisan kiküldhető WAIT-ek száma korlátozott, nem lépheti túl a konfigurációban rögzített *CanTpRxWftMax* értéket. Amennyiben ez mégis megtörténik, a modul véget vet a fogadás folyatnának és jelzi a hiba okát a fölsőbb rétegeknek.

```
PduR_CanTpRxIndication(id, NTFRSLT_E_WFT_OVRN)
```

4.5 Hibakezelés

A BSW modulok alapvető szolgáltatásai közé tartozik a *Development Error Tracer* (Det) segítségével történő hibajelentés. A nem várt, rosszul időzített, vagy hibás paraméterek, függvényhívások, állapotváltások kiszűrése megkönnyíti az ECU szoftverelemeinek megfelelő módon történő integrálását.

A CAN Transport Layer modulban a Det által jelzett hibák vonatkozhatnak:

- adott függvény hívásakor a modul inicializálatlan állapotára (CANTP_E_UNINIT)
- bizonyos körülmények között nem támogatott műveletre (CANTP_E_OPER_NOT_SUPPORTED)
- hibás azonosítóra (CANTP_E_PARAM_ID)
- adott szolgáltatásnak NULL-pointerként átadott paraméterre (CANTP_E_PARAM_POINTER)

Emellett diagnosztizálják a küldés (CANTP_E_TX_COM) és fogadás (CANTP_E_RX_COM) során fellépő időtúllépésből adódó hibákat, valamint a nem megfelelő adathosszúsággal rendelkező adatkeretek érkezését (CANTP_E_INVALID_RX_LENGTH).

A CAN Transport Layer modulban a Det szolgáltatásai egy makró segítségével ki- és bekapcsolhatók.

4.6 Erőforrásigény

Egy modul tervezésekor mindig felmerül, hogy hogyan és milyen mértékben vagyunk képesek *futási időt vagy memóriát spórolni*. A két törekvés általában egymás ellenében hat. Az optimalizálás kérdése igen összetett, és a modulfejlesztés jelen fázisában még mérések sem támasztják alá az e téren elért eredményeket. A fejlesztés során mégis érdemes feltérképezni, hogy milyen törekvések milyen irányban befolyásolják a modul erőforrásigényét.

Bizonyos kérdések a modul konfigurációjakor, illetve az egyes modulok integrációja kapcsán dőlnek el. A CanTp esetében az egyik legalapvetőbb kérdés, hogy adott N_PDU fogadásakor vagy N_SDU küldésekor *szükség van-e* a modul használatára. Amennyiben a rendszer adott konfigurációja nem igényli az üzenetek szegmentálását és újraegyesítését, a CanTp akár egy triviális megvalósítással is helyettesíthető. Ha mégis használatban van a CAN Transport Layer, az egyik legfontosabb, a futási időt jelentősen érintő feladat az *azonosítók keresésének megoldása*. Ez szintén elég erőteljesen konfigurációfüggő.

Az alapprobléma az, hogy akár a fölsőbb, akár az alsóbb rétegek felől érkezik kérés, mindig egy azonosítót kap a CanTp, amelyhez aztán a további teendők végrehajtásához meg kell keresnie az adott azonosítót tartalmazó csatornát. Ha extrémen fontos a *gyors futás*, a konfigurációkor kérhetjük egy olyan *táblázat* összeállítását, amely valamennyi azonosítóhoz eltárolja a csatorna számát, így mindig egyetlen lépésből tudjuk, hogy milyen indexet kell használni a konfigurációs struktúra és a korábban bemutatott globális változókat tartalmazó tömbök indexeléséhez. Ez a táblázat azonban extra memóriahelyet foglal. Ha *memóriára optimalizálunk*, kizárólag a konfigurációs struktúra felhasználásával, de adott esetben meglehetősen *hosszas lineáris keresés* során is megtalálhatjuk a keresett indexet.

A fönti két esetben semmifajta rendszerezettséget nem feltételeztünk az egyes csatornákhoz rendelt azonosítók között. Ha azonban az egyes csatornákhoz valamilyen *szabályos rendben* vannak rendelve az azonosítók, a keresési algoritmusok lecserélhetők. Erre szélsőséges példa lehet az, amikor minden csatornához csak egyetlen azonosító tartozik, és azok *szorosan indexelve* követik egymást: ez esetben maga az azonosító szolgálhat az indexelésre. Kevésbé egyszerű eset, amikor az azonosítók *növekvő sor-*

rendben követik egymást, de több azonosító tartozik egy csatornához, de így is *gyorsabb* keresési algoritmus alkalmazható, mint a legegyszerűbb lineáris keresés.

A futási időt szintén csökkentheti például, ha nem alkalmazunk a kommunikáció során PDU-kitöltést, így lassú másolási folyamatok spórolhatók meg. Összegzésképpen elmondható, hogy egy mindenre felkészített modulnál sokkal gyorsabban képes futni és kisebb memóriaigényű is az a változat, amely valamilyen szempontból speciális konfigurációra van tervezve. A kettő ötvözhető is, preprocesszor direktívák segítségével *feltételesen fordíthatóvá* tehető a kód egy-egy része: így elkészíthető egy összetettebb és egy egyszerűbb változat, amelyek közül az adott konfiguráció dönti el, hogy melyik változat forduljon le a modul integrálásánál.

Esetemben, mivel az integráció csak később történik meg, a lehető legegyszerűbb és mindenre felkészített változat megírását választottam, lineáris kereséssel (Így az azonosítók csatornához rendelésekor nem feltétel azok sorrendben történő elhelyezése.). Ha azonban birtokomba jut egy tipikus konfigurációról szóló terv, reményeim szerint sikerül egy feltételesen fordítható, gyorsabb futást eredményező keresési algoritmus implementálása is.

4.7 A modul tesztelése

4.7.1 A tesztelés célja és a tesztkörnyezet kialakítása

Egy program tesztelésének legalapvetőbb célja azt bizonyítani, hogy az teljesíti a specifikációban foglalt követelményeket. Ez történhet rendszer szinten (*rendszertesztelés*), az egyes modulok összekapcsolásakor (*integrációs teszt*) és egy-egy modulra vonatkoztatva (*komponenstesztelés*). Egy AUTOSAR BSW modul fejlesztése során az első lépés természetesen az utóbbi: a modult mint önálló egységet kell letesztelni. Ez először nem is feltétlen a megfelelő működést bizonyítja, hanem biztosítja, hogy a modul *hibás* működése *észlelhető* legyen.

Azért fontos, hogy a modul írója is végezzen legalább alapvető funkciókat érintő tesztelést, mert ezáltal kiszűrhetővé és maga a fejlesztő által rögtön orvosolhatóvá válnak az adódó – olykor banális – hibák. Az AUTOSAR a BSW modulok korábbi verzióihoz rendelkezésre bocsátott *TTCN (Testing and Test Control Notation)* nyelven írott megfelelőségi (*compliance*) teszteket. Ez a fajta tesztelés azt vizsgálja, hogy az adott

tesztelt modul mennyire tartja be a szabvány előírásait. A teszteknek mind leírása, mind implementációja mindenki számára ingyenesen elérhető volt. Az AUTOSAR jelenleg nem nyújt frissítéseket az említett tesztesetekhez, a jövőben több modult érintő nagyobb egységek (pl. a teljes CAN stack) tesztelését biztosító elfogadási (*acceptance*) tesztek közreadását tervezi.

A teszteléshez szükséges környezet tartalmaz *drivereket* és *stub-*, azaz helyettesítő *függvényeket*. A driver, vagyis meghajtó az a programrész, amely adott forgatókönyv alapján meghívja a tesztelendő modul egyes függvényeit. (Tehát maga a teszteseteket leíró kód.) A stub függvények a tesztelendő modullal kapcsolatban álló, más modulokhoz tartozó függvényeket helyettesítik. Egyszerű működéssel ruházzák fel őket, ezáltal biztosítják, hogy amikor a tesztelt modul meghívja valamelyik stubként implementált függvényt, kiszámítható lesz az adott függvényhívás eredménye, így a modul működése a tesztelés során könnyebben követhetővé válik.

4.7.2 A teszt hatékonyságának mérőszámai

Ahhoz, hogy egy modul teljes egészében le legyen tesztelve, minden egyes bemeneti kombinációra tesztelni kell működését. A kimerítő tesztelés irreális időigénye miatt természetesen nem kivitelezhető, célunk mégis, hogy jól behatárolható idő alatt, mégis kellő alapossággal teszteljük a modult. Ehhez az összes releváns forgatókönyv alapján megvizsgáljuk a működést. Eközben a kód nagy része lefedettségbe kerül.

A kódlefedettség vizsgálata egy olyan elemző módszer, amely megvizsgálja, hogy a szoftvernek mely részei lettek végrehajtva a teszt futtatása során. A következő lefedettség-kategóriák a kód különböző hatékonyságú tesztelésének mérőszámai [12].

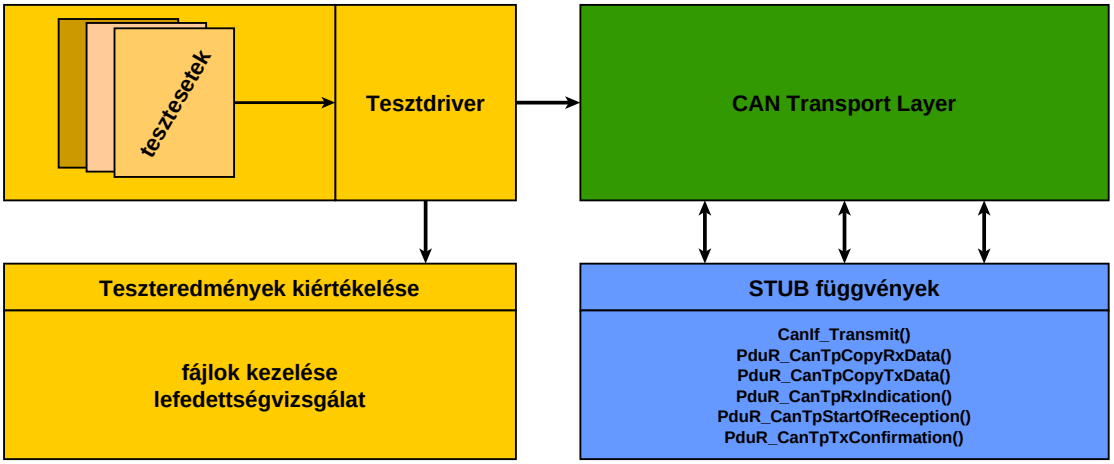
- Utasítás lefedettség (Statement Coverage): a tesztelés során kipróbált futtatható utasítások százaléka.
- Elágazás lefedettség (Branch Coverage): a tesztesetek által meghívott elágazások százalékos aránya.
- Döntési feltétel lefedettség (Decision Condition Coverage): a teszt futtatásakor az összes feltétel eredmény és döntési eredmény meghívásának százalékos aránya. (Feltétel eredmény alatt valamely feltétel igaz vagy hamis értékre történő kiértékelését, míg döntési eredmény alatt egy adott futtatási ágat kiválasztó döntés eredményét értjük.)

- Módosított döntési feltétel lefedettség **(**Modified Condition Decision Coverage): annak mérőszáma, hogy a teszt végrehajtása során milyen arányban kerülnek meghívásra a döntési eredményeket függetlenül befolyásoló egyes feltételek lehetséges eredményei.

4.7.3 A CanTp modul tesztelése

4.7.3.1 Speciális tesztkörnyezet kialakítása

A CAN Transport Layer modul általam végzett tesztelése alapvetően arra irányult, hogy a helyes működést bizonyítsam, illetve kiszűrjem a kódban található hibákat. Ehhez a *CUnit* teszteszközt használtam, a lefedettség egyszerű vizsgálatához pedig a GCC (GNU Compiler Collection) beépített *GCOV* eszköze volt segítségemre.



4.15. ábra: a tesztkörnyezet kialakítása

A létrehozott tesztkörnyezet tartalmazza a CUnit függvénykönyvtárára támaszkodó *tesztdrivert* és az összes, a CanTp által meghívott – más modulokhoz tartozó – függvény degradált működésű helyettesítő *(stub)* változatát. A CUnit a teszteléshez C programnyelven írt, egyszerű egyenlőségvizsgálatot *(ASSERTION)* megvalósító függvényeket bocsát rendelkezésre. A működés vizsgálatához segéd-buffereket alkalmaztam: küldéskor a PduR felől érkező és a CanIf felé távozó PDU-kat eltárolva, és a bufferek tartalmát összehasonlítva ellenőrizhető, hogy megfelelő módon történt-e meg a szegmentálás. Fogadáskor hasonlóképpen lehet elemezni az ellenkező irányú folyamat sikeres végrehajtását. A stub függvények a segéd-buffereken dolgoznak, így modellezik a CanTp fölötti és alatti modulok működését.

A könnyebben megvalósítható tesztelés érdekében a tesztkörnyezet egyszerű fájlkezelő függvényeket is tartalmaz:

```
void LoadFromFile(PduInfoType* pduin, const char * filename)

void SaveToFile(PduInfoType* pduout, const char * filename, boolean frameformat)

boolean CheckWithFile(PduInfoType* pdutocheck, const char * filename)
```

A fájlból betöltő (Load) függvény a tesztelés folyamatának elején feltölti a küldéshez és fogadáshoz használt segéd-buffereket (TxIn és RxIn). A fájlba író (Save) függvény a teszt végeztével a CanTp kimeneti PDU-it tároló (TxOut és RxOut) bufferek tartalmát menti. Ez utóbbinál beállítható, hogy szegmentált üzenetek esetén az adatkereteknek megfelelően tördelje a fájl sorait (a frameformat paraméter TRUE értéke esetén). A harmadik függvény a kimeneti fájl ellenőrzésére szolgál: a bemenet alapján *elvárt* kimenetet veti össze a *tényleges* kimenettel. Egyezőség esetén *TRUE* értékkel tér vissza.

4.7.3.2 A tesztelés bemutatása egy példán keresztül

Nézzünk egy egyszerű példát a szegmentált adatküldés tesztelésére!

```
LoadFromFile(&TxIn, "test_CanTp4in.txt");

01-02-03-04-05-06-07-08-09-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-
```

A teszteset elején a driver a TxIn buffert feltölti fájlból kiolvasott értékekkel. (Az egyszerűség kedvéért legyenek ezek most a 01 és 24 közötti pozitív egész számok.) A küldés a CanTp_Transmit() függvény meghívásával kezdődik. Ez után a CanTp_MainFunction() többszöri meghívásával és a megfelelő FC framek és megerősítések jól ütemezett elküldésével a küldés folyamata megvalósul:

```
CanTp_MainFunction();

CanTp_RxIndication(Id, &FlowControll);

CanTp_TxConfirmation(Id);
```

A driver természetesen nem hívhatja meg a stub függvényeket, azokat a háttérben a CanTp hívja meg, belső működésének megfelelően:

```
PduR_CanTpCopyTxData()

CanIf_Transmit()
```

A CopyTxData stub függvény a TxIn buffer tartalmának minden egyes meghíváskor egy-egy újabb részletét adja át a CanTp hívására. A CanIf_Transmit stub a kapott PDU-kat eltárolja a TxOut bufferben. A sikeres küldés eredményét a teszteset végén a driver a kimeneti fájlba írja:

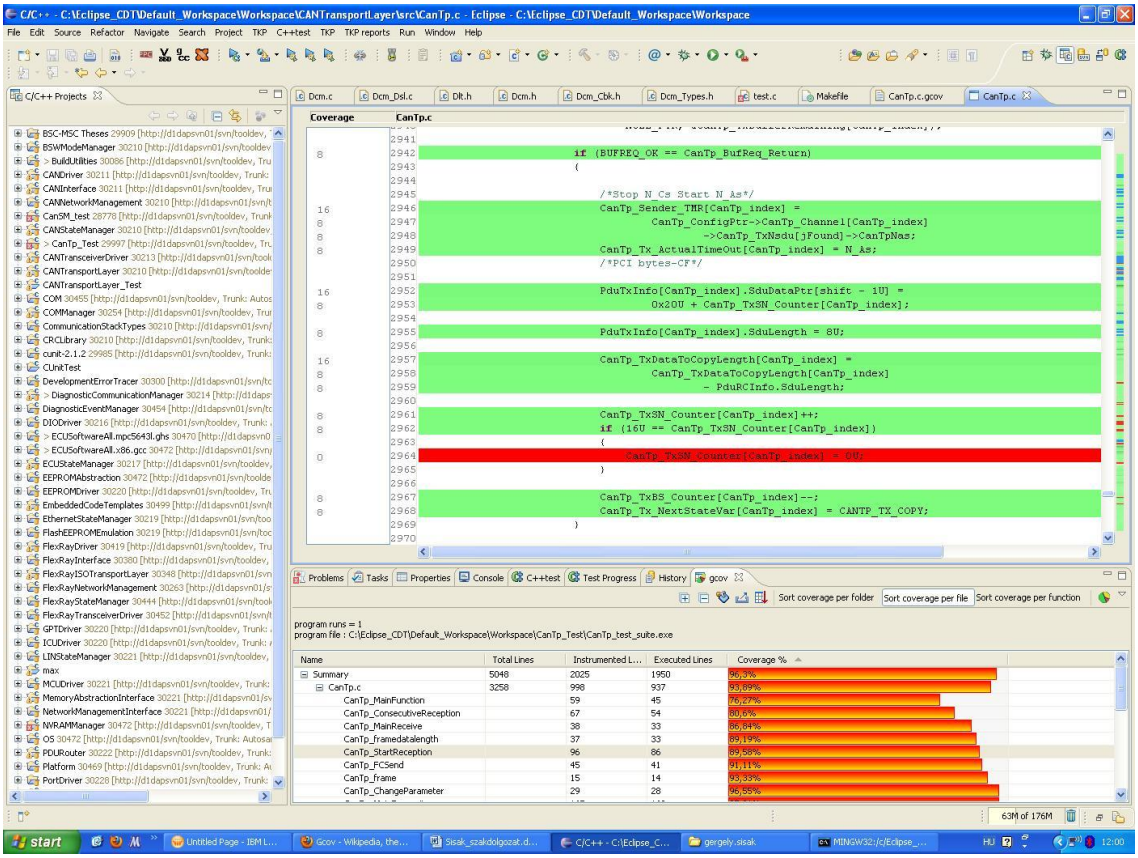
```
SaveToFile(&TxOut, "test_CanTp4out.txt", TRUE);

10-18-01-02-03-04-05-06-
21-07-08-09-10-11-12-13-
22-14-15-16-17-18-19-20-
23-21-22-23-24-ff-ff-ff-
```

A kapott kimenetről látszik, hogy a szegmentálás sikeresen megtörtént: a tesztkonfigurációnak megfelelően normál címzéssel és aktív paddinggel. Az első sor egy FF, amely tartalmazza az 1 értékű típusazonosítót, a 0x18 (=24) értékű FF_DL kódot és az első hat adatbájtot. A további sorok CF-eket tartalmaznak: 2-es típusazonosítóval, a sorszámuknak megfelelő SN értékekkel (1, 2, 3) és 7-7 adatbájttal. Az utolsó frame végén 0xFF értékű paddingbájtok találhatók.

Mindeközben a driver természetesen megvizsgálja a kulcsfontosságú változók várt és a teszt futtatása során adódó értékeit a CUnit által biztosított CU_ASSERT_EQUAL(tényleges érték, várt érték) függvény-makró segítségével. A GCOV eszköz utasítás- és elágazás lefedettséget mér. Minden egyes kódsorról naplózza, hogy lefutott-e, vagy sem, és ha igen, akkor hányszor. A hozzá tartozó Eclipse-es plugin ennek megfelelően a lefutott sorokat zölddel, a le nem futottakat pedig pirossal színezi, és külön oszlopban jelzi minden sorra a lefutások számát (4.16. ábra). A modul jelenlegi utasítás- és elágazás lefedettsége 90% körüli érték. Ez kellő mennyiségű információt nyújt a modul működéséről, de az integráció előtt mindenképpen szükséges lesz egy átfogóbb, nagyobb lefedettséget biztosító tesztelés megvalósítása is.

A modul készítője által írt tesztek hátulütője a specifikációban rögzített kérdéses pontok egyoldalú értelmezése. Sokkal nagyobb biztonságot jelent, ha független tesztelést is végeznek a modulon. Amennyiben így is megfelelőnek bizonyul a működés, nagyobb a valószínűsége annak, hogy az implementációban valóban az szerepel, amit az AUTOSAR a specifikáció elkészítésekor rögzíteni akart. Az átfogóbb tesztelés kapcsán emellett mindenképpen szükséges a módosított döntési feltétel lefedettség vizsgálata és lehetőleg minél nagyobb lefedettség elérése.



4.16. ábra: a CanTp modul kódlefedettségének megjelenítése Eclipse-es gcov pluginnel

5 A CAN State Manager modul tervezésének bemutatása

A CAN State Manager (CanSM) feladata a COM Manager által kért *kommunikációs módok beállítása* a hálózatokon. A CanSM a kontrollereket és transceivereket a CAN Interface modulon keresztül éri el, és a hardver üzemmódokat érintő változásokról is azon keresztül értesül. A CanSM a hálózatok kezelésére állapotgépeket valósít meg.

5.1 Állapottérképek megvalósítási módjai

Összetett állapottérképek megvalósítása nem triviális kérdés, az optimális megoldás a feladat jellegétől függ. A két legismertebb megoldás az egymásba ágyazott switch-case szerkezeteken (nested switch) illetve állapottáblákon (state table) alapul [13].

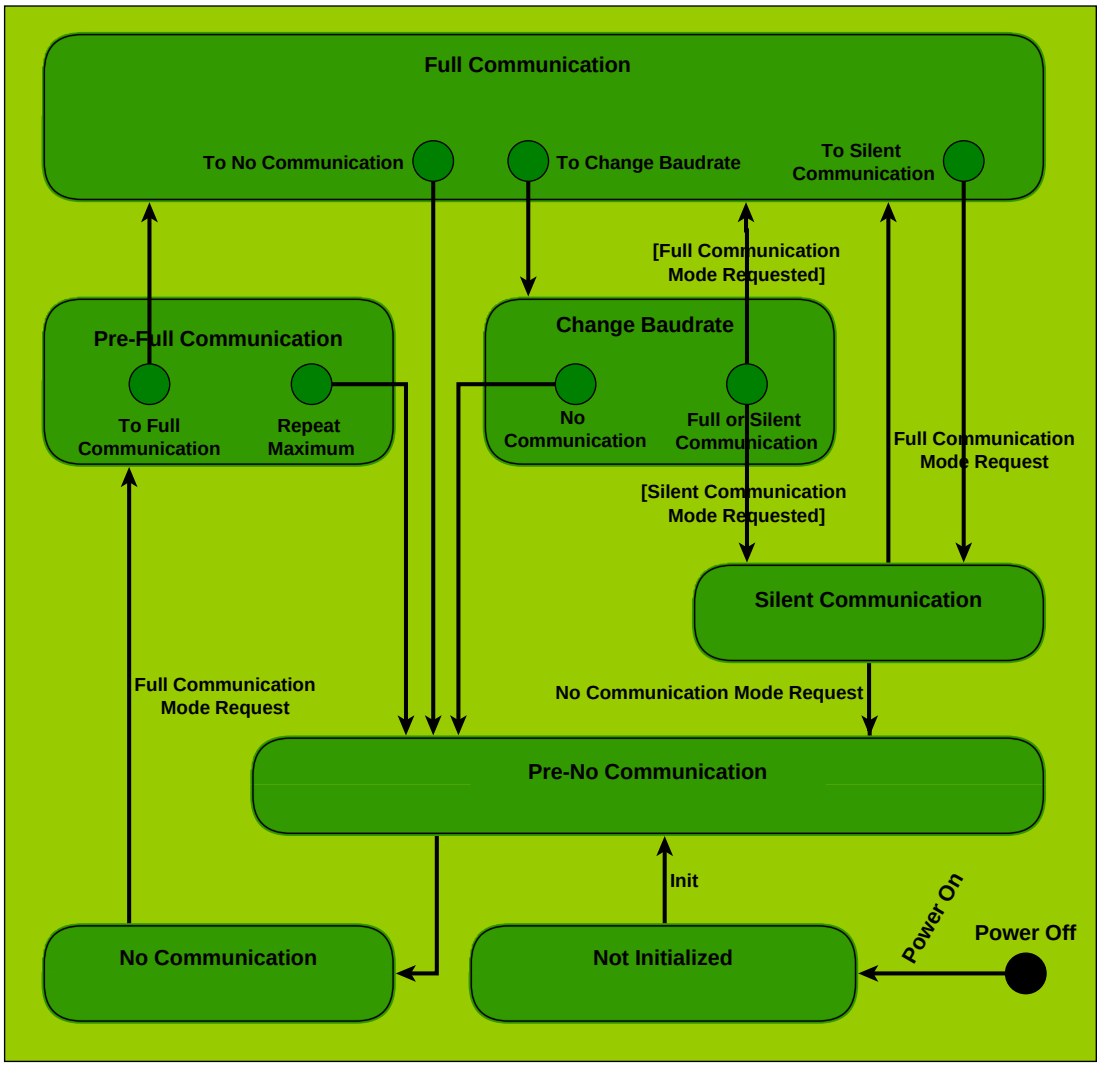
A *nested switch* minta az aktuális állapotoknak és érkező eseménynek megfelelően két egymásba ágyazott elágazásból áll. Az állapotokat és eseményeket célszerű szimbolikusan enumerációkkal definiálni. Ez a megoldás nem nyújt közvetlen eszközöket állapotok közti hierarchikus viszonyok tükrözésére, pl. összetett állapotok elhagyásakor ugyanazt az aktivitást (a szülő állapot kilépési aktivitását) több helyen is szerepeltetni kell a forráskódban. Ezáltal jelentősen nőhet a kód mérete és több hibalehetőség is adódik.

Az *állapottáblás* megvalósításnál egy kétdimenziós pointertömb sorai az *állapotoknak,* oszlopai az állapotváltásokat eredményező *eseményeknek* felelnek meg. Az i-edik sor j-edik mezőjében található pointer olyan függvényre mutat, amely az „i” állapotban a „j” eseményre adott választ valósítja meg: az aktivitásokat és az esetleges állapotváltást. Ez a megoldás sem támogatja közvetlenül a hierarchikus állapotok kezelését.

Látható, hogy a két megoldás közel azonos kifejezőerővel bír, erőforrásigényükben azonban adódnak eltérések. Viszonylag sok állapot és kevés átmenet esetén a nested switch struktúra *kisebb tárigényű,* mint a foghíjasan kitöltött állapottábla, ugyanakkor az állapottábla előnye, hogy az elágazások kiválasztása *gyors és konstans idejű,* szemben a nested switch struktúrával, ahol függhet az állapotok és átmenetek számától. A CanSM viszonylag ritkán fut, így a *sebesség nem kritikus,* emellett a *megoldás tárigénye* beágyazott platformon mindig releváns. E megfontolások vezettek arra, hogy a modul megvalósítására a nested switch mintát válasszam.

5.2 A CanSM modul megvalósítása

5.2.1 A szabvány által előírt követelmények



5.1. ábra: a CAN State Manager fő állapotgépe

A CAN State Manager a képen látható állapotgépéhez teljes és pontos leírást ad az AUTOSAR szabvány. Ezért a fő- és alállapotgépek részletezése helyett az implementáció kapcsán felmerülő kérdésekről érdemes inkább beszélni. A CAN State Managert a ComM modul vezérli a CAN hálózat állapotai szerint (*Full Communication, No Communication, Silent Communication*). Az ennek megfelelő állapotok, valamint az ezeket megelőző és előkészítő (Pre-előtaggal rendelkező) átmeneti állapotok mellett az átvitel során alkalmazott sebesség megváltoztatására szolgáló (*Change Baudrate*) állapot is szerepel a CanSM fő állapotgépének lehetséges állapotai között. Az ECU bekap-

csolásakor az állapotgép a *Not Initialized* állapotba lép. Itt még semmilyen funkcionalitás nem érhető el. Az *Init* függvény meghívása után az állapotgép a *Pre-No Communication* állapotot veszi fel, ahonnan további működése során juthat el az említett állapotokba.

Az állapottérképen látható *őrfeltételek* egyszerűen kiértékelhető kifejezések, a *triggerek* közvetlenül értelmezhetők (függvényhívások a modul interfészén), és az *állapotváltáshoz rendelt aktivitások* is pár sorban implementálhatók. Az adott állapotokban ezzel szemben olykor sok lépésen keresztül zajló, összetett tevékenység történik. Ennek megfelelően a *Pre-No Communication, Pre-Full Communication* és a *Change Baudrate* állapotok belső működését további *alállapotgépek* írják le.

5.2.2 Az állapotgépek megvalósítása a CanSM modul esetén

5.2.2.1 A modul felsőszintű állapotgépe

A CAN State Manager modul *valamennyi konfigurált CAN hálózatra* képes kezelni a kontroller és transceiver egységek állapotváltozásait, ennek megfelelően (mivel az egyes hálózatokhoz azonos szerkezetű, de egymástól függetlenül működő állapotgép tartozik) a modul *main* függvényének lefutásakor elvégezzük az események feldolgozását mindegyik állapotgép kontextusában.

A CanSM_MainFunction() folyamatosan figyeli, hogy történt-e állapotváltás az adott csatornára vonatkozóan. Ehhez az aktuális állapotot (*CanSM_BSM_State*) a következő állapotot tároló *CanSM_BSM_NextState* változóval hasonlítja össze. Amennyiben a két érték megegyezik, az állapotgép nem szándékozik állapotot váltani. Ekkor az aktuális állapotra vonatkozó állapotfüggvény hívódik meg.

```
switch (CanSM_BSM_State[CanSM_index]){
     case CANSM_BSM_S_PRE_NOCOM:
          CanSM_Bsm_S_Pre_Nocom_SF();
          break;
     case CANSM_BSM_S_NOCOM:
          CanSM_Bsm_S_Nocom_SF();
          break;
...
     case CANSM_BSM_S_SILENTCOM:
          CanSM_Bsm_S_Silentcom_SF();
          break;
     default:
          break;
}
```

Az állapotfüggvények tartalmazzák az egyes állapotokhoz tartozó alállapotgépek implementációját (amennyiben az adott állapothoz létezik ilyen). Az állapotléptetéshez szükséges őrfeltételek teljesülésének vizsgálata és a trigger-események kezelése is itt történik. Amennyiben az állapotgépnek állapotot kell váltania, a CanSM_BSM_NextState állapotváltozónak az *új állapotot* adja értékül.

A CanSM_MainFunction() következő meghívásakor tapasztalja, hogy a CanSM_BSM_State és CanSM_BSM_NextState eltérő értéket mutat. Ezért meghívja az új állapot belépési függvényét (Enter Function).

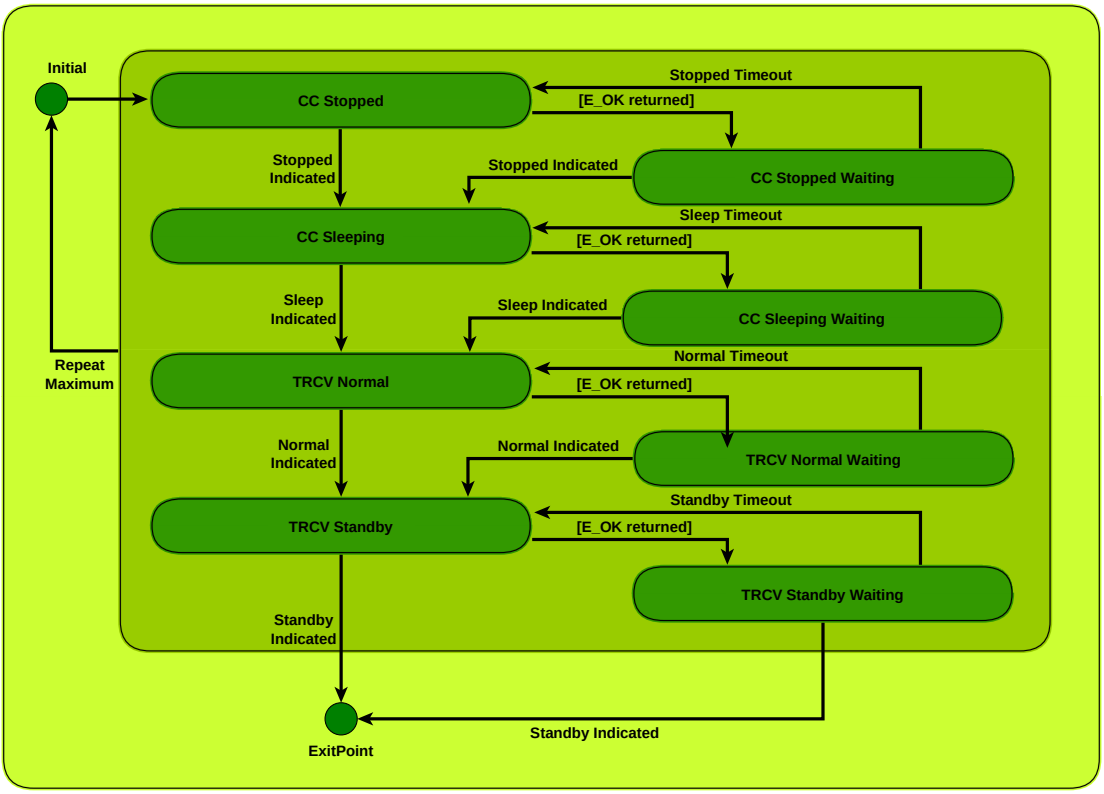
```
switch (CanSM_BSM_NextState[CanSM_index]){
     case CANSM_BSM_S_PRE_NOCOM:
          CanSM_Bsm_S_Pre_Nocom_E();
          break;
     case CANSM_BSM_S_NOCOM:
          CanSM_Bsm_S_Nocom_E();
          break;
...
}
```

A különböző állapotokhoz tartozó belépési függvények hajtják végre az adott állapotváltáshoz rendelt *akciókat*. Azért választottam ezt a megoldást, mert egy bizonyos állapotba történő belépéskor szinte mindig egyazon történések zajlanak le, függetlenül attól, hogy mi volt az azt megelőző állapot. Ezáltal a CanSM_BSM_NextState érték (feltéve hogy nem egyezik meg az aktuális állapottal), jól definiál egy bizonyos akciósorozatot. Egy belépési függvény meghívása után mindig aktualizálni kell az állapotgép állapotváltozóját.

```
CanSM_BSM_State[CanSM_index] = CanSM_BSM_NextState[CanSM_index]
```

Az állapotgép mindaddig az új állapotban marad, amíg – külső események hatására, vagy az állapotot jellemző belső állapotgép működése kapcsán – a NextState állapotváltozó értéke meg nem változik.

5.2.2.2 Az alállapotgépek megvalósítása



5.2. ábra: a CanSM modul Pre-No Communication alállapotgépének részlete

A modul alállapotgépeinek közös jellemzője, hogy viszonylag sok belső állapottal rendelkeznek, az állapotok közti léptetés azonban rendkívül egyszerű, láncszerű. Az állapotok általában a hardveregységek (kontroller vagy transceiver) működési állapotainak felelnek meg (pl. CC_SLEEP vagy TRCV_STANDBY), a hozzájuk köthető tevékenység legnagyobbrészt a CAN Interface modul valamely függvényének meghívása:

- CanIf_SetControllerMode(id, ControllerMode)
- CanIf_SetTrcvMode(id, TransceiverMode)
- CanIf_SetPduMode(id, Mode)
- CanIf_CheckTrcvWakeFlag(id)
- CanIf_ClearTrcvWufFlag(id)

A függvények értelemszerűen az *id* azonosítóhoz rendelt hardver *működési módját* állítják be, illetve a transceiver ébresztési (wake) funkcióját menedzselik. Visszatérési érté-

kük StdReturnType típusú. Egy adott állapotból a következőbe történő lépéshez két feltételnek kell teljesülnie:

- a meghívott függvénynek E_OK-val történő visszatérése
- az adott függvényhez tartozó callback függvénynek azonos azonosítóval történő meghívása a CanIf által

A callback függvények jelzik egy-egy művelet sikeres végrehajtását a hívó fél számára. (pl. a SetControllerMode meghívása után a CanIf a CanSM_ControllerModeIndication(id, ControllerMode) callback függvény segítségével jelzi, hogy az *id* azonosítójú kontroller felvette a *ControllerMode* üzemmódot.)

Egy adott függvény E_OK-kal történő visszatérése *(őrfeltétel)* és a hozzá tartozó callback érkezése *(trigger)* után léphet az állapotgép a következő állapotba. Amennyiben csak az őrfeltétel áll fenn, az állapotgép az adott állapothoz tartozó *WAIT állapotba* kerül, és ott várja a trigger-esemény bekövetkeztét. Sikertelen várakozás esetén, egy bizonyos időzítő lejárta után az állapotgép visszakerül a WAIT-et megelőző állapotba.

Az egyszerű működés és az állapotokhoz tartozó speciális őrfeltételek, illetve trigger-események miatt a célszerű implementáció az alállapotgépek esetében is a switch-case szerkezetes megoldás volt. Ehhez az alállapotok aktuális értékét csatornánként el kell tárolni, így két main hívás között nem vész el az állapot. Az állapotok közti léptetés szükségessége egyszerű feltételvizsgálattal kiértékelhető.

5.3 A CAN State Manager hibakezelése

A CAN State Manager a fejlesztés ideje alatt fellépő hibákról értesíti a Det modult *(Det_ReportError)*. Az inicializálatlan állapotban történő függvényhívásokat, a hibás vagy NULL-pointerként átadott bemeneti paramétereket éppúgy jelenti, mint az állapotmenedzseléssel kapcsolatos hibákat. Ez utóbbi kategóriába tartozik, amikor a ComM modul függő módváltás alatt újabb módváltást kér, vagy a módváltás egy konfigurációban megszabott értéknél többször nem sikerül.

Az átviteli sebesség megváltoztatásakor követelmény, hogy egy adott hálózatba tartozó valamennyi kontroller támogassa az új értéket. Ennek megfelelően a *CheckBaudrate(network, baudrate)* függvény meghívásakor a CanSM modul ellenőrzi az adott *baudrate* értéket a *network* hálózat összes kontrollerére, és ha mindannyiuk ál-

tal elfogadásra kerül, elmenti azt. A *CanSM_ChangeBaudrate(network, baudrate)* függvényhívás csak akkor eredményezhet tényleges változást, ha a függvény *baudrate* bemeneti paramétere megegyezik a korábban vizsgált és mentett értékkel. Ellenkező esetben a váltás nem történik meg, és a modul CANSM_E_PARAM_INVALID_BAUDRATE kóddal hibajelentést küld a Det-nek.

5.4 Erőforrás használat

A CAN State Manager modul támogatja a *partial networking* (PN) szolgáltatást: az egy funkcionális egységbe tartozó részhálózatok együttes hálózati menedzsmentjét. A CANSM_BSM_S_PRE_NOCOM állapotban megvalósított alállapotgépnek két változata van: az egyik támogatja a PN-t, a másik nem. Ennek megfelelően a modul konfigurációjában csatornánként rögzítve van, hogy melyik ágon fusson le a program. Futási időben, feltételvizsgálattal dől el, hogy az adott csatorna esetén melyik állapotgép lép működésbe.

A CAN Interface specifikációjában előírtak szerint a CanIf modul esetén globális kapcsolóval is állítható, hogy támogatja-e a PN-t. A CanIf és a CanSM közti együttműködés megkönnyítése érdekében létrehoztam egy azonos rendeltetésű kapcsolót a CanSM modul konfigurációs paraméterei között is. A kódban elhelyezett preprocesszor direktívák segítségével feltételesen fordíthatók azok a részek, amelyeket érint a PN. Így kikapcsolt PN esetén jelentősen *csökkenhet a kódméret*, illetve az elmaradó feltételvizsgálatoknak köszönhetően valamelyest *csökken a futási idő is*.

A *keresési algoritmusok* a CanTp modul esetéhez hasonlóan jelentősen befolyásolják a futási időt. Kevés konfigurált kontroller, illetve transceiver esetén gyorsabb a modul működése. Hasonló a helyzet akkor is, ha a hardver egységekhez rendelt azonosítók indexelésre is felhasználhatók, egy gyors elérésre optimalizált konfiguráció esetén.

5.5 A modul tesztelése és további élete

A CanSM modul teszteléséhez – a CanTp esetéhez hasonlóan – szükség van tesztdriverre és a megfelelő stub-függvények megvalósítására. Az állapotgép helyes működésének bizonyítása meglehetősen hosszú folyamat, hiszen meg kell vizsgálni az összes lehetséges állapotátmenetet. A szisztematikus tesztelés állapotról állapotra halad és minden lehetséges átmenetet bejár. Ehhez a tesztdriver az adott állapotváltozást előidé-

ző CanSM-függvényeket (pl. *CanSM_ControllerModeIndication*), valamint a modul main függvényét hívja meg adott forgatókönyv szerint. A beérkezett kérést a modul regisztrálja és a main meghívásakor az állapotgép a megfelelő állapotba kerül.

Az elágazások teszteléséhez a stub-függvényeket a *driver által manipulálhatóvá* kell tenni. Gyakori eset, hogy a CanIf függvényeinek hívásakor az adott függvény E_OK visszatérési értéke biztosítja a továbblépést egy következő állapotba. A driver által előre meghatározott visszatérési érték teszi kiszámíthatóvá a függvények működését (pl. a driver által beállított E_NOT_OK visszatérési érték esetén a belső függvényhívás nem eredményez állapotváltást). A különböző stub-függvények különböző visszatérési értékkel történő használata lehetővé teszi a tesztelési forgatókönyvek rugalmas alakíthatóságát és ezáltal a modul átfogó tesztelését.

A tesztek során egyrészről az állapotváltozókat felhasználva ellenőriztem, hogy egy adott kiindulási állapotból megfelelő gerjesztés esetén az állapotgép a várt következő állapotba kerül-e. Másrészről a modul belső életével nem törődve megvizsgáltam, hogy adott – például kontroller és transceiver üzemmódra vonatkozó – kérések teljesülnek-e, pusztán a main függvény és néhány callback függvény meghívásával.

A CanSM modul tesztelése során a helyes működést a specifikáció által támasztott követelményeknek megfelelően pontról-pontra ellenőriztem. Ezzel párhuzamosan sikerült megfelelő kódlefedettséget elérni, így a fejlesztési folyamat rám eső részét a későbbi visszajelzésekig átmenetileg befejezettnek tekinthettem.

Azóta a modul integrációja szempontjából fontos lépésként, az általam definiált konfigurációs struktúra alapján elkészült a CAN State Manager *konfiguráció generátora*, melynek segítségével, a konfigurációs adatok bevitele után automatikusan generálható C-kód. Tőlem függetlenül jelenleg is készülnek *kézzel írott tesztek*, melyek a szabvány egyes pontjait más ember szemszögéből elemezve hivatottak az elkészült modul helyes működésének igazolására. Ezzel párhuzamosan megkezdődött a CAN stack moduljainak *integrációs tesztelése a célplatformon*. Ennek keretein belül a CanSM modul működőképesnek bizonyult.

6 Az elvégzett feladatok összegzése, tanulságai

A szakdolgozat elkészítéséhez az AUTOSAR szabvány alapos megismerésére volt szükség, mivel a feladat a modulok megvalósítása mellett a kommunikációs réteg áttekintésére is vonatkozott. Ennek megfelelően az alapvető architektúrális elvek feltérképezése után a kommunikációban résztvevő modulok specifikációit tanulmányoztam. A dolgozatban található két ECU közötti kommunikációt bemutató leírás ilyen formában nem szerepel a szabványban. Hosszas utánaolvasás és a megszerzett ismeretek rendszerezésének, összefoglalásának eredményeképpen jött létre. A CAN-ről szóló rövid összefoglalót a 2.0-ás szabvány alapján készítettem. Mindkét elméleti leírás illusztrálásához – valamint a szakdolgozat további fejezeteihez is – saját magam készítettem ábrákat.

Az AUTOSAR szabvány specifikálja a modulok alapvető felépítését és az elvárásokat, de a tervezőre bízza a működést biztosító algoritmus megtervezését. A BSW stacknek jelenleg nincs elérhető nyílt forrású megvalósítása, és a fejlesztés erőforrásigénye, illetve a kapcsolódó költségek (pl. kötelező konzorciumi tagság) miatt a jövőben sem valószínű, hogy ilyen készüljön. Ennek megfelelően, és hogy implementációs döntéseim valóban önállóak legyenek, más megvalósításoktól teljesen függetlenül, pusztán a modulok specifikációja alapján hoztam létre őket, korábbi megvalósításokat nem vizsgáltam.

A CAN Transport Layer modul tervezéséhez az AUTOSAR specifikáció alapos, minden részletre kiterjedő megismerése mellett az ISO 15765-2-es szabvány áttanulmányozására is szükség volt. Ezt követően valósítottam meg az adatküldést és fogadást kiszolgáló, két jelentősen eltérő algoritmust. A modul tesztelése sikeresen megtörtént, az általam kialakított tesztkörnyezet a modul specifikációjában rögzített elvárások megvalósulásának igazolásán túl lehetővé teszi a küldés és fogadás során alkalmazott szegmentálás és újraegyesítés fájlkezeléssel támogatott tesztelését is.

A CAN State Manager tervezését a CanTp modulhoz hasonlóan megelőzte a szabvány feldolgozásával töltött munka. Emellett a megvalósítás előtt áttekintettem az állapottérképek megvalósításának két legalapvetőbb módszerét, majd ezek alapján választottam ki az alkalmazott nested switch módszert. A döntés során figyelembe vettem a modul erőforrásigényének alakulását az alkalmazott megoldás függvényében. A meg-

valósítás – az imént említett szempontokat leszámítva – kevesebb önállóságot követelt, mint a CanTp modul esetében, mivel a szabvány az állapotgépet teljes egészében részletesen specifikálja. Az elkészített modul helyes működését tesztek futtatásával igazoltam.

A feladatkiírásban megfogalmazott elvárásokat sikerült teljesítenem, leszámítva a TTCN nyelven írott megfelelőségi tesztek futtatását, amelyek helyett kézzel írott teszteket hoztam létre. A megvalósított modulok jelenleg további tesztelésre és integrációra várnak. Így elmondható, hogy a kitűzött célt elértem, de a modulok működőképessége csak akkor lesz teljes biztonsággal megállapítható, ha sikeresen beépülnek az esetlegesen megvalósuló teljes ECU szoftverbe. A szakdolgozat írása közben is rádöbbentem egy-két olyan hibára, vagy hiányosságra, amelynek következtében módosítottam a kódon és a folyamat várhatóan még folytatódik: habár a szoftver készen van és tesztekkel tudom bizonyítani helyes működését, feltehetően mégis történni fognak kisebb nagyobb módosítások a közeljövőben.

A modulok megvalósításán túl is rendkívül tanulságos volt számomra a dolgozat megírása. Az AUTOSAR szabvánnyal való megismerkedés részeként a CAN stack-hez tartozó modulok specifikációinak áttanulmányozása által átfogó képet nyertem a BSW rétegen belüli kommunikációs folyamatokról, amely a későbbiekben megkönnyítheti esetleges további modulok megvalósítását. A tesztelés kapcsán szerzett elemi ismeretek az eddigi ismereteim e téren jelentkező hiányosságait képesek valamilyen szinten pótolni. A C-nyelven történő beágyazott szoftverfejlesztés a szakirányos tárgyak hallgatása óta nem volt számomra teljesen ismeretlen, de csak minimális alkalmam volt a gyakorlatban is kipróbálni. A ThyssenKrupp Presta Hungary Kft. kiváló környezetet biztosított, ahol segítőkész konzulensek segítségével tudtam bekapcsolódni a szoftverfejlesztés folyamatába, megszerezve ezzel azt az alapszintű tudást, amely további tanulmányaimhoz és munkámhoz nélkülözhetetlen.

Irodalomjegyzék

[1] Robert Bosch GmbH: *CAN 2.0 specifikáció, ISO 11898-2* (1991)

[2] AUTOSAR Consortium: *Layered Software Architecture, 3.2.0* (2011) http://www.autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] LIN Consortium: *Overview* http://www.lin-subbus.org/

[4] National Instruments: *FlexRay Automotive Communication Bus Overview* (2009) http://www.ni.com/white-paper/3352/en

[5] AUTOSAR Consortium: *Specification of Communication, 4.2.0* (2011) http://www.autosar.org/download/R4.0/AUTOSAR_SWS_COM.pdf

[6] AUTOSAR Consortium: *Specification of PDU Router, 3.2.0* (2011) http://www.autosar.org/download/R4.0/AUTOSAR_SWS_PDURouter.pdf

[7] AUTOSAR Consortium: *Specification of CAN Interface, 5.0.0* (2011) http://www.autosar.org/download/R4.0/AUTOSAR_SWS_CANInterface.pdf

[8] AUTOSAR Consortium: *Specification of CAN Driver, 4.0.0* (2011) http://www.autosar.org/download/R4.0/AUTOSAR_SWS_CANDriver.pdf

[9] AUTOSAR Consortium: *Specification of CAN State Manager, 2.2.0* (2011) http://www.autosar.org/download/R4.0/AUTOSAR_SWS_CANStateManager.pdf

[10] AUTOSAR Consortium: *Specification of CAN Transport Layer, 4.0.0* (2011) http://www.autosar.org/download/R4.0/AUTOSAR_SWS_CANTransportLayer.pdf

[11] International Organization for Standardization: *INTERNATIONAL STANDARD Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services ISO 15765-2* (Second Edition 2011)

[12] Hungarian Testing Board: *Szoftvertesztelés egységesített kifejezéseinek gyűjteménye* (2012)

[13] Miro Samek, Paul Y. Montgomery: *State Oriented Programming. Embedded System Programming* (2000)

Függelék

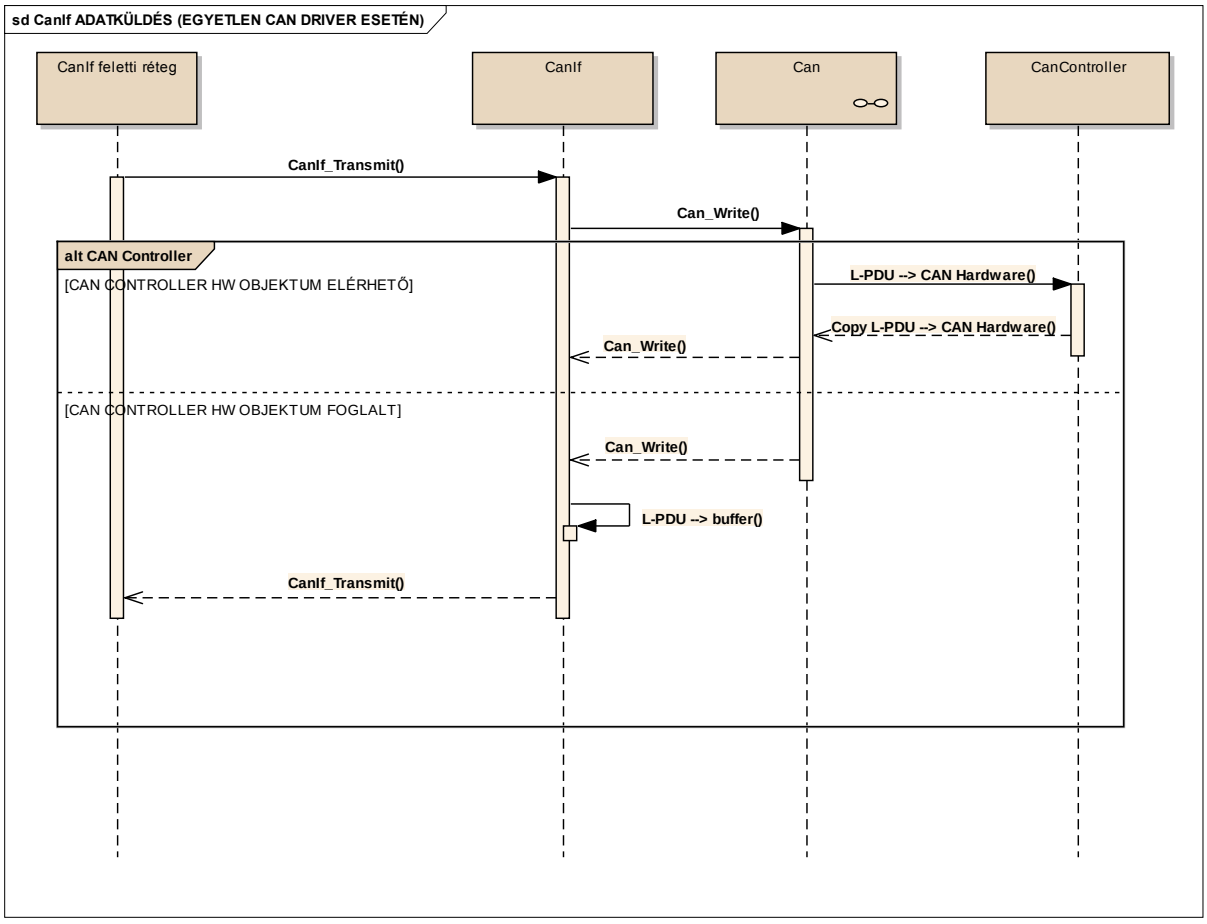
Szekvencia diagramok és folyamatábrák

Az alábbi ábrák az AUTOSAR specifikációkban ([7][10]) megtalálható szekvencia diagramok és folyamatábrák magyar nyelvű változatai. Tanulmányozásuk nagyban megkönnyíti a CanIf és CanTp modulok működésének megértését.

CAN Interface

A CanIf modul működésének áttekintése egy-egy egyszerű példán.

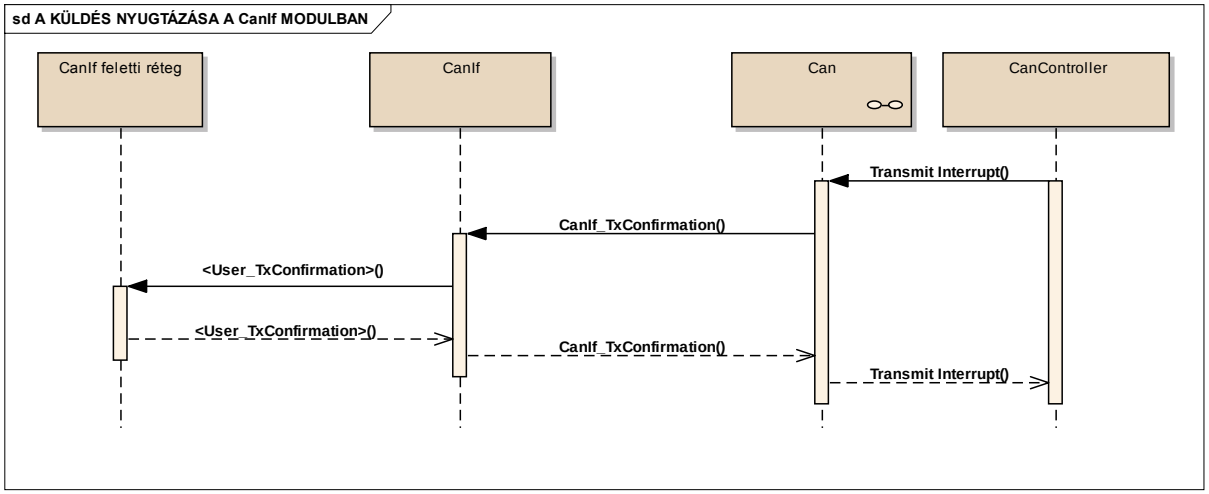
F1 Adatküldés a CAN Interface modulban



- A fölső réteg meghívja a CanIf_Transmit(CanTxPduId, PduInfoPtr) függvényt. A bemeneti értékek validálása után a kapott paraméterek alapján azonosítható a szükséges CAN kontroller, és az adat továbbküldhető.

- A CanIf modul meghívja a CanWrite(CanHWId, PduInfoPtr) függvényt a megfelelő HTH azonosítóval. Amennyiben szabad a CAN hardware, a CanWrite függvény átmásolja az L_PDU adatot, majd E_OK-val visszatér. Ha a HW nem elérhető, a visszatérési érték E_BUSY, ekkor a CanIf belső bufferében tárolja az L_PDU-t. Ezután a CanIf_Transmit() visszatér E_OK-val.

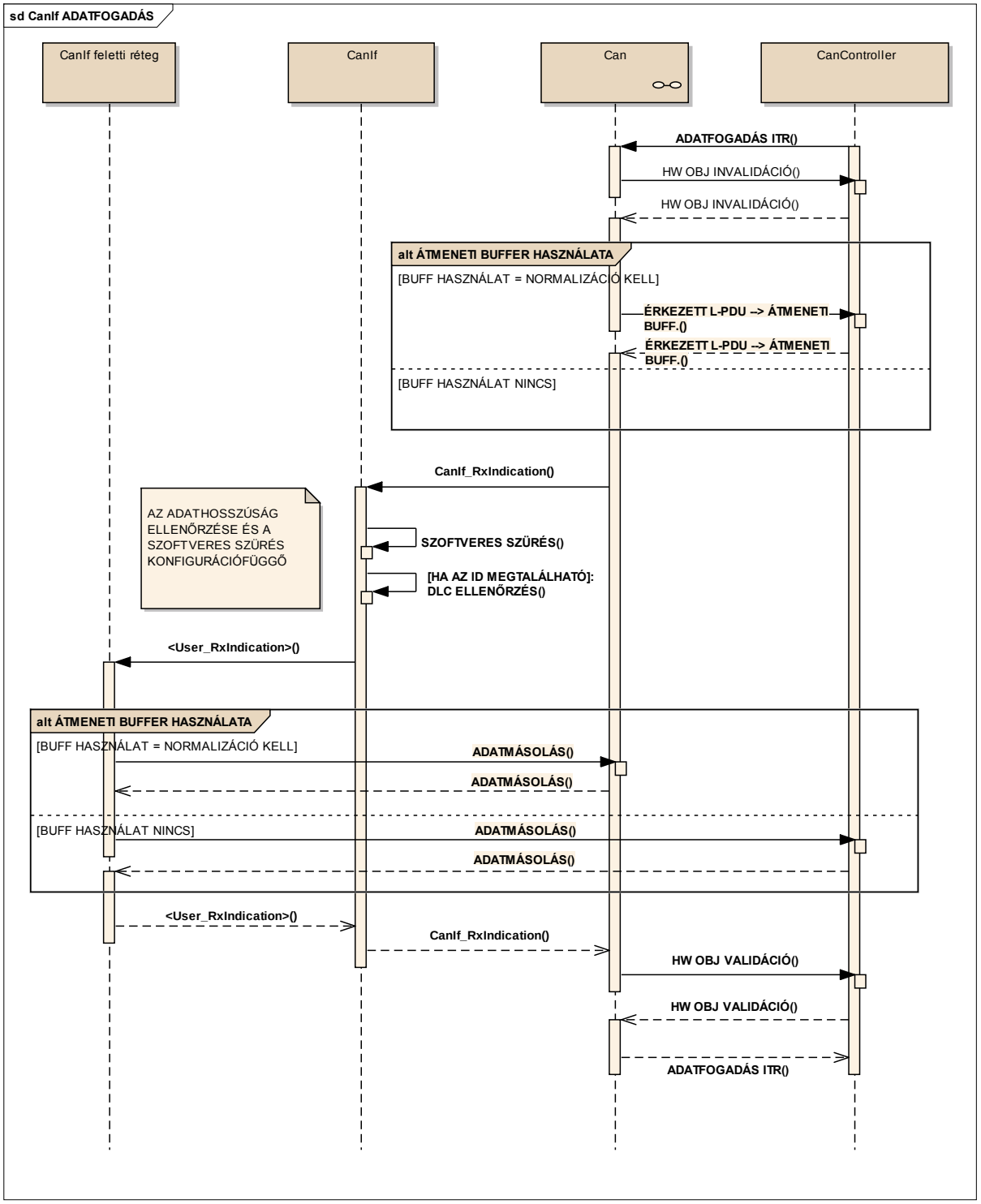
F2 A küldés nyugtázása a CAN Interface modulban (megszakításos eset)



- Az küldött CAN frame a fogadási oldalon Interruptot vált ki, így értesíti a küldő oldalt.
- A CanDriver meghívja a CanIf_TxConfirmation(CanTxPduId) callback függvényt. A CanTxPduId paraméter egyértelművé teszi, melyik küldött PDU-ra vonatkozik a megerősítés. A CanDriver az összes függőben levő üzenet azonosítóit eltárolja, így a nyugtázáskor szükséges keresés csak ezek között zajlik.
- A CanIf modul meghívja a küldött üzenethez tartozó fölső réteg (pl. CanTp) TxConfirmation callback függvényét.

F3 Megszakítási módban történő adatfogadás a CAN Interface modulban

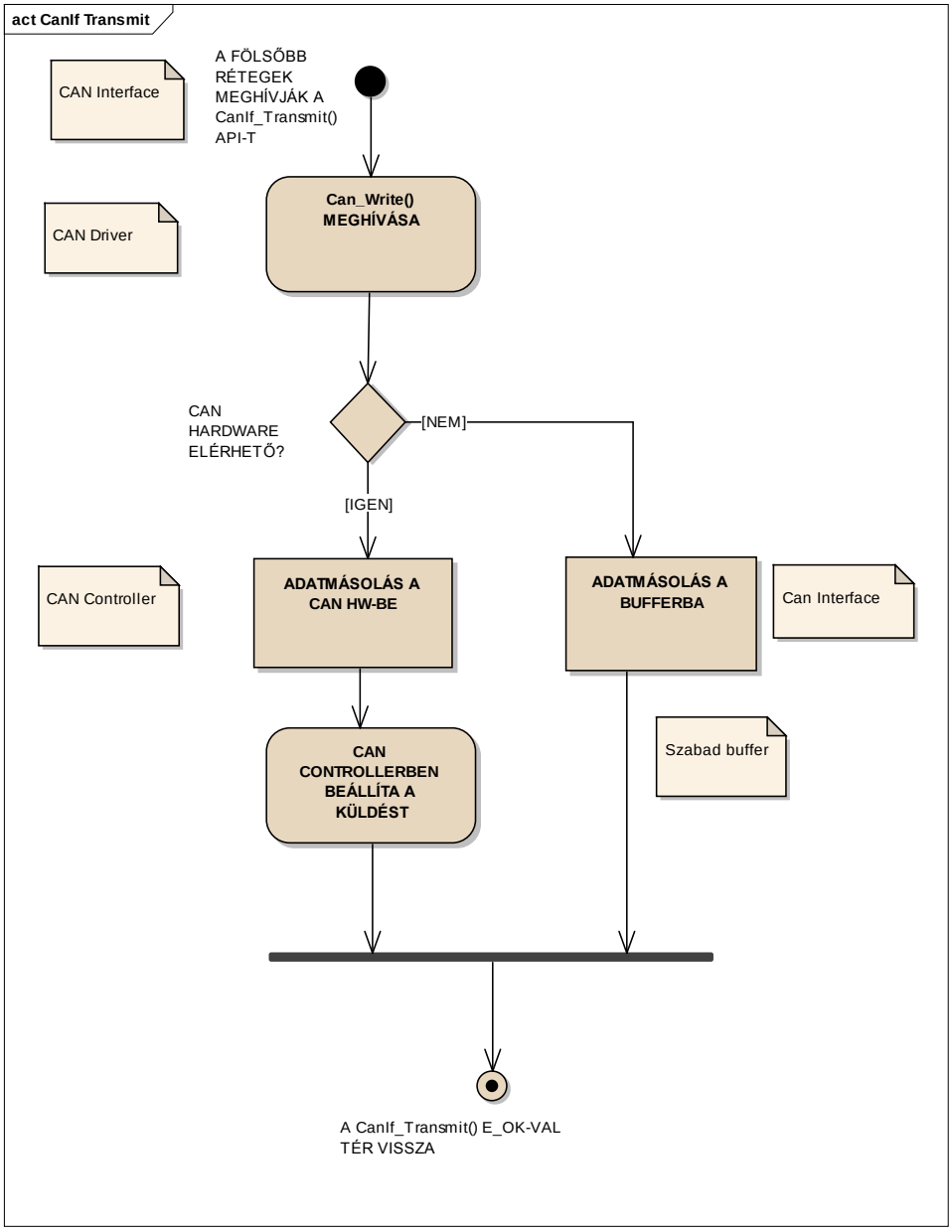
- A CAN controller sikeres fogadást jelez és megszakítást vált ki. A CAN Driver kizárólagos hozzáférést kap a CAN mailboxhoz, vagy legalábbis a fogadott adatot tartalmazó hardver objektumhoz (Hardware Objektum invalidáció).
- Normalizáció, tárolás az átmeneti bufferben (amennyiben szükséges).



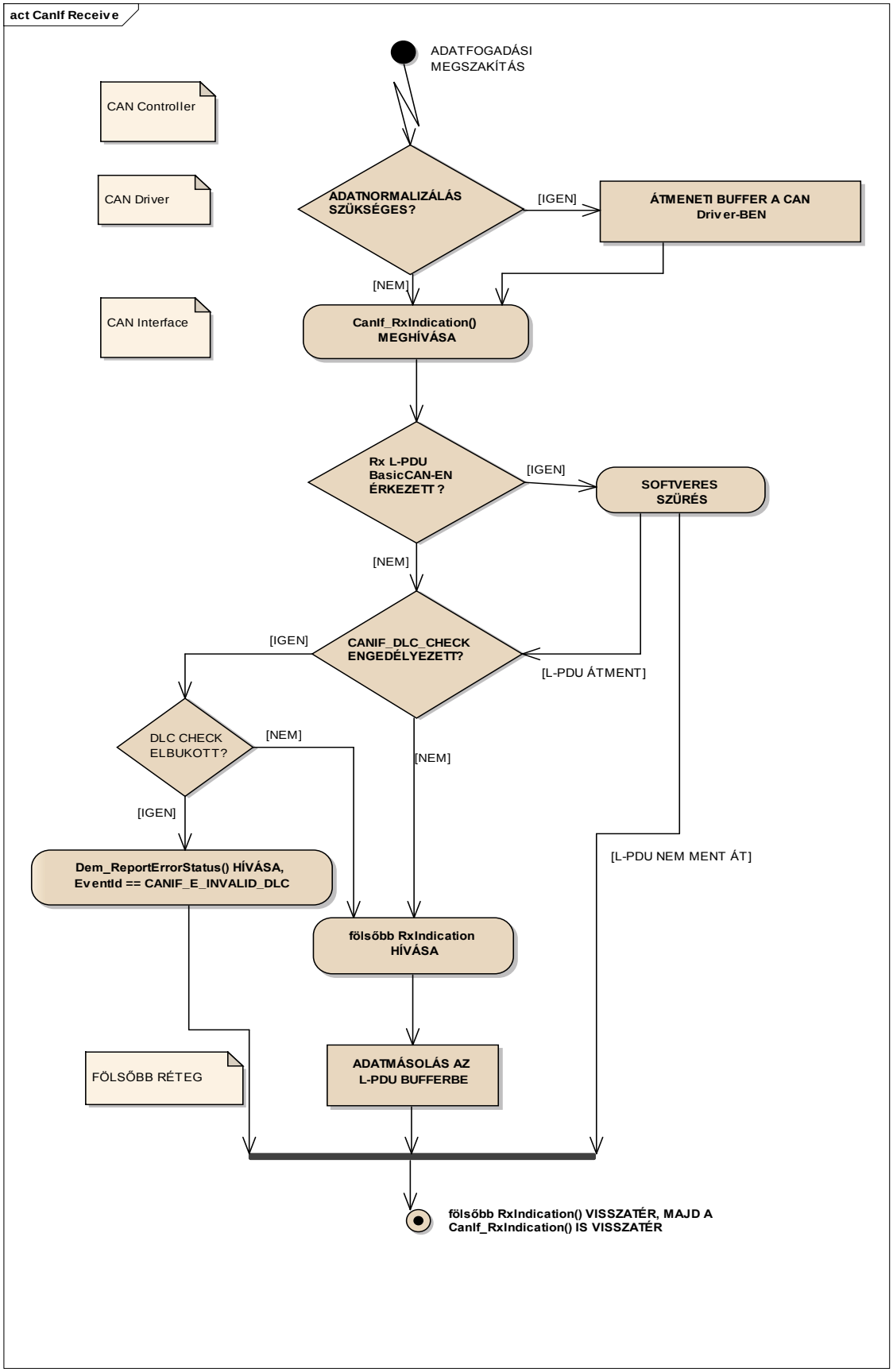
- A CanIf_RxIndication(HRH, CAN ID, DLC, CanSduPtr) függvény meghívása a HRH érték azonosítja a megfelelő kontrollert és a hozzá tartozó RAM területet. Az átmeneti buffer elérhető a CanSduPtr-en keresztül. Szoftveres szűrés ellenőrzi, hogy a fogadott L_PDU azonosítója az adott ECU-ban regisztrálva van-e vagy nem. Amennyiben nem, az L_PDU nem jut el a fölsőbb rétegekhez.

- DLC ellenőrzése: amennyiben az L_PDU-t megtalálja a szűrő algoritmus, az adathosszúság is ellenőrzésre kerül. Az aktuális értéket a konfigurációban megadott elvárthoz hasonlítja a CanIf.
- Küldés a fölsőbb rétegek felé: a megfelelő RxIndication() függvény meghívása. A függvény visszatérése után az adott CAN hardverhez való kizárólagos hozzáférési jog megszűnik.

F4 A CAN Interface küldési folyamatábrája



F5 A CAN Interface adatfogadási folyamatábrája



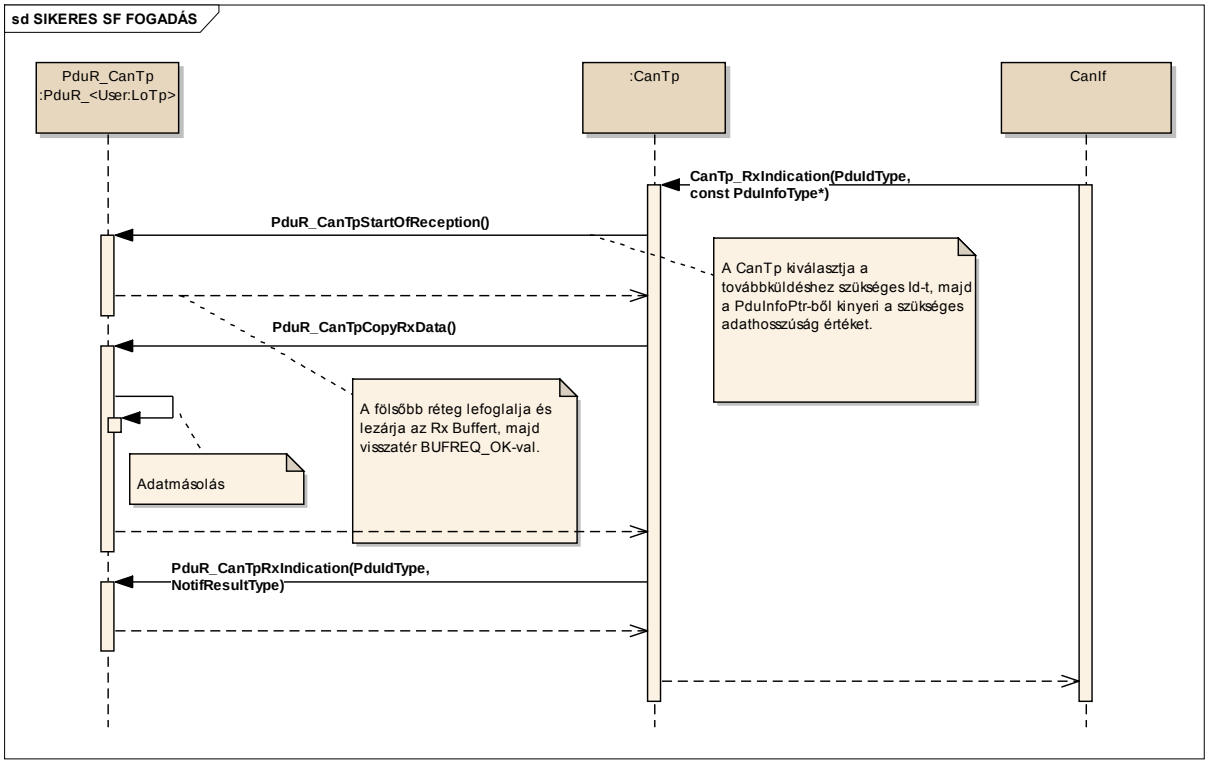
CAN Transport Layer

A CAN Transport Layer modul működésének áttekintése egy-egy egyszerű példa alapján.

F6 Single Frame sikeres fogadása

Előzetes feltevések:

- a bemeneti paraméterek szerepelnek a CanTp modul konfigurációjában
- a fogadott adat hossza nem lépi túl a SINGLE FRAME-ben küldhető adatbájtok számát.
- a fogadás sikeresen lezajlik



A fogadás folyamatának lépései

- Az alsóbb réteg (CanIf) egy frame fogadásakor értesíti a CanTp modult. A CanTp_RxIndication(CanTpRxPduId, CanTpRxPduPtr) függvényt a megfelelő azonosítóval meghívja. A CanTpRxPduPtr-en keresztül átadott struktúra tartalmazza a fogadott L_PDU adatot, valamint az adathosszúságot.

- PduR_CanTpStartOfReception(CanTpRxSduId, TpSduLength, bufferSizePtr): a CanTp a fölsőbb réteghez (PduR) tartozó azonosítót (CanTpRxSduId) kikeresi a konfigurációs struktúrából, majd TpSduLength méretű buffert kér az adatküldéshez. Az adathosszúságot jelen esetben a SF PCI mezőjének SF_DL értéke tartalmazza. Ha rendelkezésre áll a kért buffer, a StartOfReception() függvény BUFREQ_OK-val tér vissza. A rendelkezésre álló buffer mérete a bufferSizePtr-en keresztül elérhető.
- PduR_CanTpCopyRxData(CanTpRxSduId, CanTpInfoPtr, bufferSizePtr): adatmásoláskor a CanTp modul a frame-szerkezetből kinyert hasznos adatot struktúrába rendezi, amelyet egy rámutató pointeren keresztül ad át a fölső rétegnek. Itt megtörténik az adatmásolás a lefoglalt bufferbe.
- Ha véget ért az adatfogadás teljes folyamata, a CanTp a PduR_CanTpRxIndication(CanTpRxSduId, Result) callback függvény hívásával jelzi a PduR modulnak. Az eredmény (Result) sikeres küldés esetén NTFRSLT_OK.

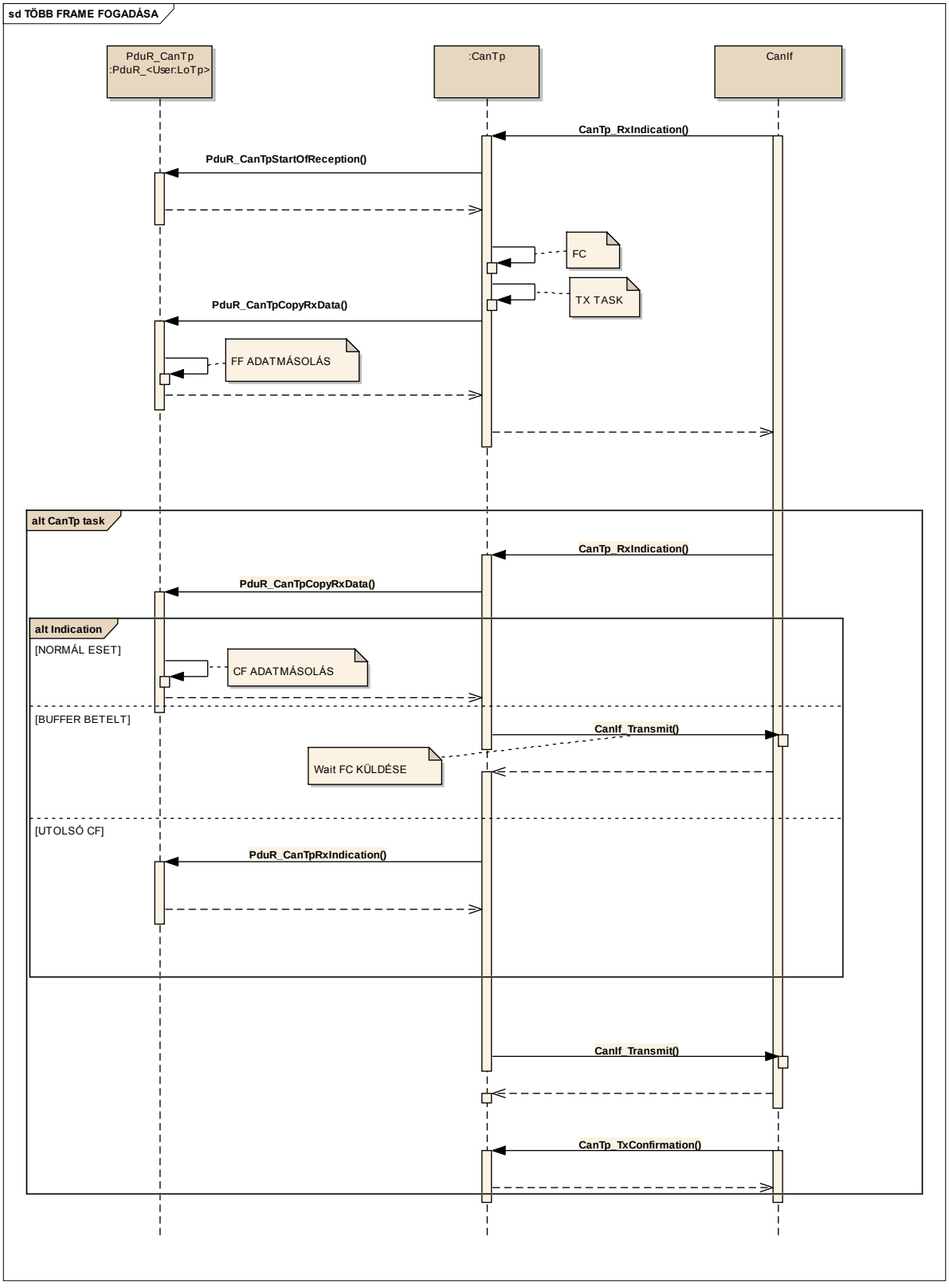
F7 Több frame-ből álló adat sikeres fogadása

Előzetes feltevések:

- a bemeneti paraméterek szerepelnek a CanTp modul konfigurációjában
- a fogadott adat hossza túllépi a SINGLE FRAME-ben küldhető adatbájok számát, így a folyamat a FIRST FRAME és az azt követő CONSECUTIVE FRAME(K) fogadásával történik
- a fogadás sikeresen lezajlik

A fogadás folyamatának lépései

A fogadás blokkokban történik. A rendelkezésre álló bufferkapacitás függvényében a CanTp modul kiszámítja a blokkméretet (BS). Ez az érték a túloldali küldő számára a FlowControl érkezése nélkül, egy blokkban küldhető framek számát jelenti. A fogadó oldalon a tárkapacitás vizsgálata minden blokk után megtörténik, és amennyiben nincs elegendő memória a következő teljes blokk másolásához, várakozásra kéri a küldőt, egészen addig, amíg az igényelt memóriaterület rendelkezésre nem áll. A folyamat menedzseléséhez tehát FC framek küldése is szükségessé válik.



- Az alsóbb réteg (CanIf) egy frame fogadásakor értesíti a CanTp modult. A CanTp_RxIndication (CanTpRxPduId, CanTpRxPduPtr) függvényt a megfelelő

azonosítóval meghívja. A CanTpRxPduPtr-en keresztül átadott struktúra tartalmazza a fogadott L_PDU adatot, valamint az adathosszúságot.

- PduR_CanTpStartOfReception(CanTpRxSduId, TpSduLength, bufferSizePtr): a CanTp a fölsőbb réteghez tartozó azonosítót kikeresi a konfigurációs struktúrából, majd TpSduLength méretű buffert kér az adatküldéshez. Az adathosszúságot jelen esetben a fogadott FF PCI mezőjének FF_DL értéke tartalmazza. Ha rendelkezésre áll a kért buffer, a StartOfReception függvény BUFREQ_OK-val tér vissza. A rendelkezésre álló buffer mérete a bufferSizePtr-en keresztül elérhető.
- Ezt követően történik meg a FF másolása a fölsőbb réteg által foglalt bufferbe a PduR_CanTpCopyRxData(CanTpRxSduId, CanTpInfoPtr, bufferSizePtr) függvénnyel.
- Consecutive Frame (CF) fogadásakor három eset lehetséges:
    1. PduR_CanTpCopyRxData(CanTpRxSduId,CanTpInfoPtr,bufferSizePtr): normál esetben az adatokat fogadó buffer nincs telve, és a CF nem az utolsó frame a fogadási folyamat során.
    2. PduR_CanTpCopyRxData(CanTpRxSduId, NULL, bufferSizePtr): a buffer megtelt. Ez esetben további buffer igénylése szükséges a fölső rétegtől. Erre a PduR_CanTpCopyRxData() függvény speciális meghívása szolgál. A normális esetben adatstruktúra átadására szolgáló CanTpInfoPtr paraméter helyett NULL-pointert adunk a függvény bemeneti értékének.
    3. Ha véget ért az adatfogadás teljes folyamata, a CanTp a PduR_CanTpRxIndication(CanTpRxSduId, Result) callback függvény hívásával jelzi a PduR modulnak. Az eredmény (Result) sikeres küldés esetén NTFRSLT_OK.
- A FC framek küldését a CanIf_Transmit(CanTxFcPduId, PduInfoPtr) függvény látja el. A konfigurációs struktúrában rögzített, az aktuális PduId-hoz tartozó FlowControl azonosító (CanTxFcPduId) és a CTS, illetve WAIT információt hordozó PduInfoPtr a függvény bemeneti paraméterei.

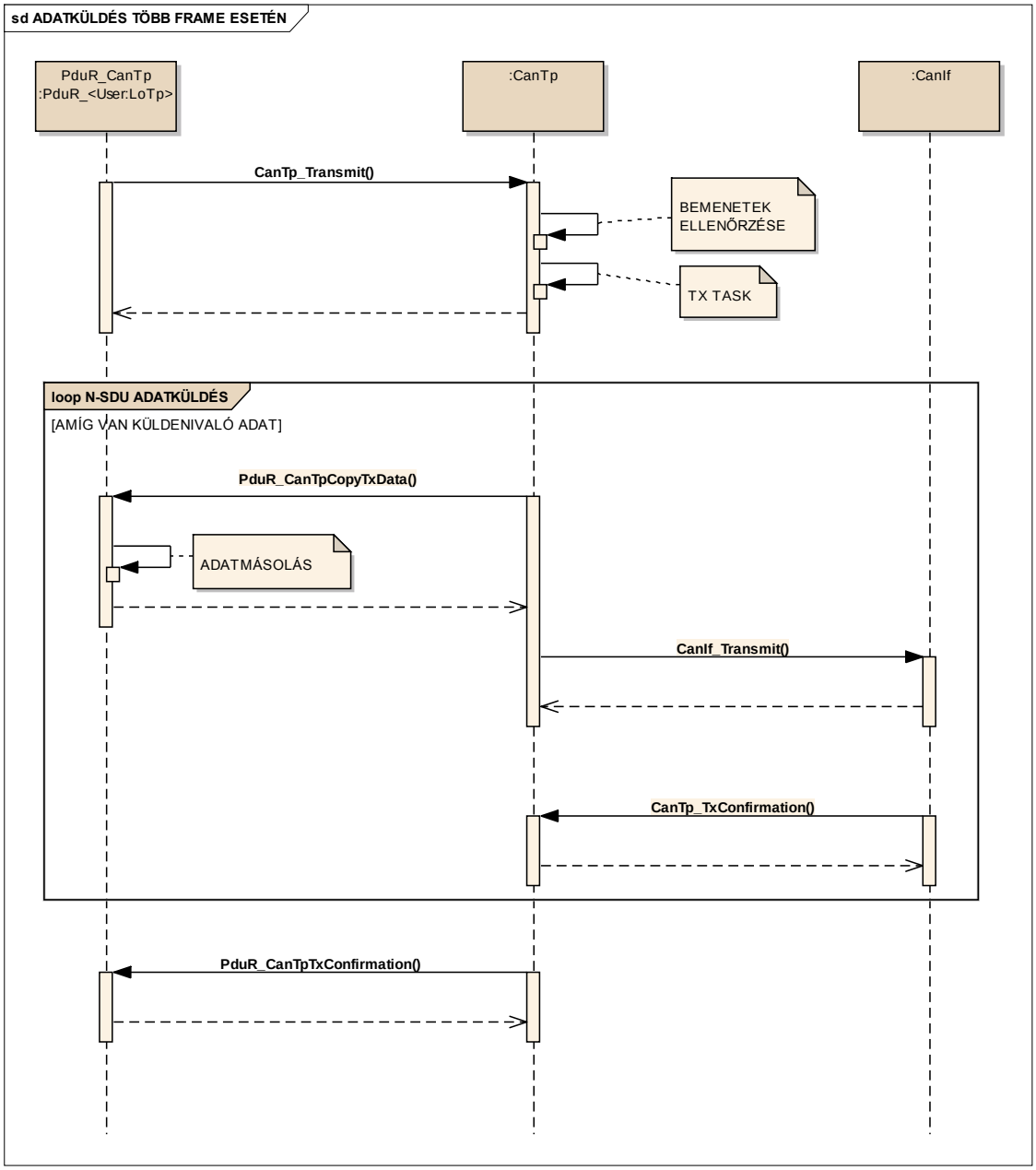
F8 Több frame-ből álló adat sikeres küldése

Előzetes feltevések:

- a bemeneti paraméterek szerepelnek a CanTp modul konfigurációjában
- a küldött adat hossza túllépi a SINGLE FRAME-ben küldhető adatbájtok számát, így a folyamat a FIRST FRAME és az azt követő CONSECUTIVE FRAME(K) küldésével történik
- a küldés sikeresen lezajlik

A küldés folyamatának lépései

- A küldés folyamata a CanTp_Transmit(CanTpTxSduId, CanTpTxInfoPtr) függvény meghívásával indul. A CanTp ellenőrzi a CanTpTxSduId-t, hogy egy, a modul konfigurációjában szereplő azonosítóról van-e szó. A küldés során használt többi paramétert is kikeresi, az adott PDU-hoz (címzési kiterjesztések, a fogadó oldaltól származó FC frameknél használt FC azonosító, illetve az alsóbb (CanIf) réteggel történő kommunikáció során használt PduId). A CanTpTransmit() függvény hívásakor a CanTpTxInfoPtr által mutatott struktúrából csak az adathosszúság paraméter kerül feldolgozásra (a hossz itt a teljes küldési folyamat során átküldeni kívánt adat hosszát jelöli), az adatbájtok később, a PduR_CanTpCopyTxData() függvény meghívásával várnak elérhetővé.
- PduR_CanTpCopyTxData(CanTpTxSduId, PduTxInfoPtr, availableDataPtr): a CanTp modul függvényhívására a felsőbb réteg (PduR) bemásolja az adatszegmenst a célbufferbe (PduTxInfoPtr). Sikeres adatmásolás esetén a visszatérési érték BUFREQ_OK.
- A CanTp modul a CanIf_Transmit(CanTxPduId, PduInfoPtr) függvény meghívásával továbbítja az adatszegmenst. Ehhez az azonosító átkonvertálása szükséges. (A CanTp által használt N_SDU Id megfelelője a CAN Interface modul esetén az L_SDU Id.) A szokásos PduInfoPtr struktúra is módosul, mivel a frameszerkezethez szükséges címkiegészítéseket, valamint az esetleges paddingbájokat a CanTp modul hozzátoldja a fentről kapott adatbájtokhoz.



- A sikeres küldést a CanIf modul a CanTp_TxConfirmation(CanTpTxPduId) callback függvénnyel jelzi.
- A további framek (CF-k) esetén hasonló folyamat játszódik le:

  PduR_CanTpCopyTxData(): adatmásolás

  CanIf_Transmit(): adattovábbítás

  CanTp_TxConfirmation(): nyugtázás
- A küldést a fogadáshoz hasonlóan a kommunikációban résztvevő másik fél FlowControl frame-jei szabályozzák. Küldő oldalon is nyilván kell tartani a

blokkméretet, hiszen egy CTS FC frame fogadása után a küldés folyamatosan történik egy blokknyi kiküldött frame erejéig. Ezt követően az újbóli küldés csak újabb CTS FC érkezése után lehetséges.

- Ha az egész frame sikeresen kiküldésre került, a CanTp modul a PduR_CanTpTxConfirmation(CanTpTxSduId, Result) callback függvénnyel jelzi a fölsőbb modul felé az eredményt.