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

Veres Gábor

AUTOSAR TCP/IP VEREM MEGVALÓSÍTÁSA

TANSZÉKI KONZULENS

Dr. Sujbert László

KÜLSŐ KONZULENS

Dr. Balogh András

(ThyssenKrupp Presta Hungary Kft.)

BUDAPEST, 2015

Tartalomjegyzék

Kivonat

A járműiparban megfigyelhető, hogy a gyártók egyre több biztonsági és kényelmi funkcióval látják el a gépjárműveiket. Ezen funkciók vezérlő egységei (ECU) együttműködve, mint elosztott rendszer hajtják végre feladataikat. Az egyes funkciók különböző fejlesztő csoportoknál történő tervezése szükségszerűvé tette, hogy az elektronikus vezérlőegységekben szabványos szoftverkomponenseket használjanak. Ezért a legnagyobb autóipari vállalatok létrehozták az AUTOSAR konzorciumot, ami az autók vezérlő egységein futó szoftverkomponensek szabványosításával foglalkozik.

Dolgozatomban két kommunikációval foglalkozó AUTOSAR szoftverkomponens megvalósítását mutatom be. Az egyik az Ethernet interfész modul a másik a TCP/IP modul. A dolgozat első felében rövid áttekintés nyújtok a felhasznált kommunikációs protokollokról, majd kiválasztok egy nyílt forrású TCP/IP vermet (LWIP), amit elhelyezek az AUTOSAR szoftver architektúrába.

Az így kialakított szoftver architektúra hiányzó rétegeinek a megvalósításával folytatom a munkát. Az Ethernet interfészen keresztül bemutatom az Ethernet alapú kommunikációt AUTOSAR szoftver komponensekkel. Az LWIP illesztését és felhasználását a 4,5,6 fejezeten keresztül mutatom be, ahol összehasonlítom a szabvány által elvárt működést a megvalósított működéssel. Majd kísérletet teszek az így kialakított szoftver rendszer egyfajta teljesítmény vizsgálatára.

A dolgozatom utolsó fejezeteiben egy saját TCP/IP stack megvalósításának kezdeti lépéseit mutatom be minimum követelmények mellet. Megvalósításra kerülnek az ARP, UDP protokollok valamint az IP protokoll egy része. Végül teszteléssel és az elkészült TCP/IP stack kipróbálásával zárom a munkámat.

Abstract

It can be observed that the manufacturers provide more and more safety and comfort features for their motor vehicles in the automotive industry. These electronic control units (ECU) of these features cooperate with each other and like distributed systems, they make their own tasks. Design of the individual functions are making at various developer groups. It has made it necessary that in electronic control unit use standardized software component. Therefore the largest automotive companies created the AUTOSAR consortium which deals with the standardizing of the software components, what running on control units of cars.

I present the implementing of two AUTOSAR software components – which deal with communications. One is the Ethernet interface module another is the TCP / IP module. In the first part of the thesis I provide a brief overview of the used communication protocols and after that I select an open source TCP/IP stack (LWIP), which I placed the AUTOSAR software architecture.

I continue the work with creating the missing layers of previously developed software architecture. I show Ethernet-based communication with AUTOSAR software components via the presentation of Ethernet interface. I illustrate the fitting and using of LWIP in the chapters of 4, 5 and 6, where I compare the operation of the required standard with the achieved operation. After that, I do a kind of evaluation of this software.

In the last chapters of my dissertation, I present the initial steps of my own TCP/IP stack, implementation what was done with minimum requirements.

The ARP, UDP protocols and a part of the IP protocol are implemented. Finally, I close my work with testing and trying the completed TCP / IP stack.

Bevezetés

A gépjárművekben a növekvő számú funkció és egyre nagyobb komplexitás növekvő adatforgalmat eredményez az ECU-k közötti kommunikációs hálózatokban. Ez indokolja, hogy a jövőbeli fejlesztések során az autókban jelenleg legtöbbet használt CAN és FlexRay buszok helyett, egy gyorsabb kommunikációt biztosító alternatívát keressünk. Egy új kommunikációs eljárás fejlesztése helyett logikus a meglévők közül választunk. Jó választás lehet az asztali számítógépek körében elterjedt Ethernet hálózat a 10Mb/s-1Gb/s terjedő sebességével és a rá épülő TCP/IP protokoll családdal. Ebbe az irányba mutat az is, hogy az AUTOSAR 4.0 már tartalmazza az Ethernetre vonatkozó saját szabványos modul definícióit a 4.1 kezdve pedig a TCP/IP-t is definiálja. Egyelőre nem cél a meglévő CAN és FlexRay hálózatok azonnali lecserélése Ethernet alapú hálózatra, hanem csak a diagnosztika területén történő bevezetése. Az ECU diagnosztikája során elterjedt protokoll az XCP protokoll, ami képes a CAN és FlexRay hálózatok mellet az Ethernet hálózaton is működni. Ethernet hálózat esetén viszont szükség van vagy a TCP vagy az UDP használatára0.

Az Ethernet

Az Ethernet eredetileg számítógép hálózatok kiépítésére kifejlesztett protokoll. Az Ethernet, mint szabvány alapvetően a fizikai megvalósítást írja le. A gyártók ezt a fizikai réteget általában két áramköri modullal valósítják meg. Az egyik egy fizikai illesztő áramkör (PHY) ami kapcsolatot teremt az Ethernet vezérlő és fizikai közeg között, a másik pedig maga az Ethernet vezérlő (MAC).

A TCP/IP

A TCP/IP általában egy teljes protokollcsaládot jelent, ami magában foglalja az összes IP-re (Internet Protocol) épülő protokollt. A TCP (Transmission Control Protocol) ennek a családnak egy tagja, de emellett rengeteg IP-re épülő protokoll van, mint például az UDP (User Datagram Protocol), vagy az ICMP (Internet Control Message Protocol).

Az internet rendkívül heterogén hálózatokat fogott össze, ezért elsődleges cél volt az IP számára, hogy a fizikai médiumtól függetlenül, bárhol képes legyen működni. Ehhez egy olyan absztrakciós rétegre van szükség, ami minden eszközön

megvalósítható és semmilyen eszközhöz nem köthető. Az IP-ben használható eszközöknek a következő követelményeknek kell megfelelniük:

- képes legyen egy meghatározott mennyiségű bájtot egyszerre átvinni (ezt hívjuk datagramoknak, vagy csomagoknak)
- képes legyen az eszköz alacsonyszintű (fizikai címét) leképezni egy logikai IP - címre

Az IP (internet protokoll) állapotmentes protokoll, nem garantálja a csomagok megérkezését a célhoz, sőt azt sem, hogy a csomagok ugyanolyan sorrendben érkeznek meg, mint ahogyan elküldték őket. Ez tulajdonképpen annyit jelent, hogy a fizikai médium megpróbál mindent megtenni a csomag célba juttatása érdekében, de nem garantál semmit.

Egy IP csomag két részből áll:

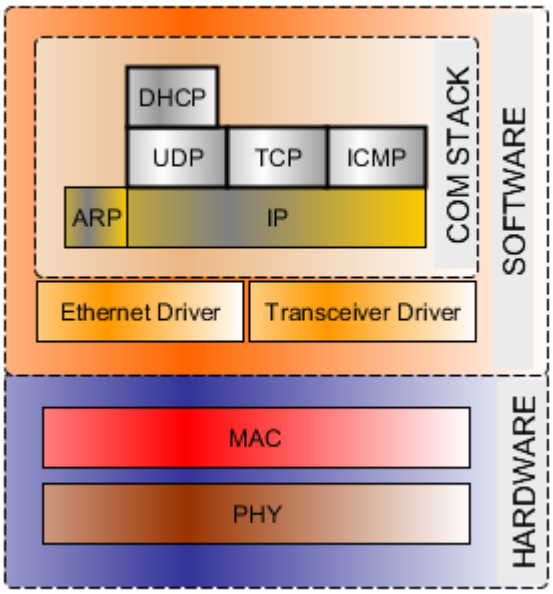
1) fejléc információk, amelyek tartalmazzák a forrás és célcímet, valamint

2) adatrész, ami alkalmazás-specifikus. Az IP-re épülő protokoll az IP adatrészét használhatja felsőbb szintű fejléc, valamint adat számára. Így egymásba skatulyázhatók a csomagok, a felsőbb szintű protokollok használhatják azt, amit az alsó szintű add, de kiegészíthetik további tulajdonságokkal.

Mint láthatjuk az IP használata elég nehézkes az alkalmazások szempontjából, hiszen gondoskodniuk kell a csomagok sorba állításáról, és az elvesző csomagok újraküldéséről. Azért hogy ne kelljen minden alkalmazásban ezeket a funkciókat implementálni, létrehozták a TCP protokollt, ami gondoskodik a csomagok sorrendbe állításáról, valamint az esetlegesen elvesző csomagok újraküldéséről. A TCP így már egy garantált csatornát biztosít a programunknak, hiszen gondoskodik arról, hogy ami a kapcsolat egyik végén "bement", az a másik végén ki is jön. Láthatjuk, hogy a TCP protokoll sokkal kényelmesebb az alkalmazások szempontjából. Ugyanakkor nem mindig van szükség a TCP-vel járó szolgáltatásokra, ezért kialakult más IP-re épült, de nem kapcsolatjellegű protokoll pl.: az UDP[2]. Az ARP protokoll nem IP alapú, de szerves részét képezi az IP működésének így erre a protokollra is a TCP/IP részeként tekintünk.

1 Ethernet alapú kommunikációs verem általános felépítése

Egy általános Ethernet alapú kommunikációs verem blokk vázlata az 1. ábrán látható. A legalsó szinten a hardver helyezkedik el.



1. Az Ethernet alapú kommunikációs verem

A hardverhez közvetlenül kapcsolódnak a driverek, amik a szoftverkomponensek legalsó rétegét képzik. A driverek a hozzájuk kapcsolódó hardver funkcióinak egyszerű elérését teszik lehetővé. Felette már a kommunikációs verem helyezkedik el, jelen esetben a TCP/IP stack.

1.1 A TCP/IP protokolljai

A TCP/IP elnevezés magába foglalja a következő alfejezetekben szereplő kommunikációs és segéd protokollokat

1.1.1 ARP: Address Resolution Protocol

Ez a protokoll, egy adott IP-címhez keresi meg, a MAC-címet. Mivel a TCP/IP 32bites (IPv4) (vagy 128bites (Ipv6) –ebben a megvalósításban nem használt–) IP-címek alapján címzik üzeneteiket az Ethernet vezérlők pedig 48bites MAC-címek

alapján küldik tovább üzeneteiket, szükség van egy IP-cím- MAC-cím összerendelésre, ezt az összerendelést az ARP protokoll fogja megtenni mielőtt TCP vagy UDP üzenetet küldenénk. Magát az ARP kommunikációt az RFC 826 írja le. Ez alapján kétféle üzenettípust különböztethetünk meg:

- ARP REQUEST: Egy node szeretné megtudni, hogy a cél IP-címhez milyen MAC-cím tartozik. Küld egy MAC szintű broadcast üzenetet, benne a kért IP-címmel.
- ARP REPLY: Az a node, aki magára vette a kérést - azaz az IP-címe megegyezett az ARP request broadcast üzenetben lévő IP-címmel – visszaküld egy ARP reply unicast üzenetet, azaz egy válasz üzenetet.

1.1.2 IP: Internet Protocol

Az OSI referencia modell hálózati réteg protokollja. „A protokoll összeköttetés-mentes. A szállított csomagok a datagramok, amely a forrás hoszt-tól a cél hosztig kerülnek továbbításra, esetleg több hálózaton is keresztül. A hálózati réteg megbízhatatlan összeköttetés-mentes csatornát biztosít, így az összes megbízhatósági mechanizmust a szállítási rétegben kell megvalósítani, ami biztosítja a két végállomás közötti megbízható összeköttetést. A szállítási réteg az alkalmazásoktól kapott üzeneteket maximum 64 kbájtos datagramokra tördeli,- ha szükséges-, amelyek az útjuk során esetleg még kisebb darabokra lesznek felvágva. Amikor az összes datagram elérte a célgépet, ott a szállítási réteg ismét összerakja üzenetté. A datagram két részből áll: egy fejrészből és egy adatrészből. A fejrészben 20 bájt rögzített és van egy változó hosszúságú opcionális rész is."[3] Az IP protokollt az RFC 791 definiálja.

1.1.3 TCP: Transmission Control Protocol

IP réteg feletti összeköttetéses szállítási réteg protokoll. A TCP üzenetküldés előtt, szükséges a kapcsolat kiépítése (3 utas kézfogás), majd az üzenetváltás befejezése után a kapcsolatot bontani kell. „Az üzenet elküldése után nyugtát vár, ha nem kapja, meg akkor újra küldi az üzenetet. A TCP fogadja a tetszőleges hosszúságú üzeneteket a felhasználói folyamattól és azokat maximum 64 kbájtos darabokra vágja szét. Ezeket a darabokat egymástól független datagramokként küldi el. A hálózati réteg sem azt nem garantálja, hogy a datagramokat helyesen kézbesíti, sem a megérkezett datagramok helyes sorrendjét. A TCP feladata az, hogy időzítéseket kezelve szükség szerint újraadja őket, illetve hogy helyes sorrendben rakja azokat össze az eredeti üzenetté. Minden TCP

által elküldött bájtnak saját sorszáma van. A sorszámtartomány 32 bit széles, vagyis elegendően nagy ahhoz, hogy egy adott bájt sorszáma egyedi legyen". A TCP működését az RFC 793 írja le.

1.1.4 UDP: User Datagram Protocol

IP réteg feletti összeköttetés-mentes szállítási réteg protokoll. Kapcsolatmentes. A feladó és a címzett nem beszélgetnek: a feladó elküldi a csomagot, a címzett vagy megkapja, vagy nem. De semmilyen formában sem jelez vissza. Megbízhatatlan. A csomagokban nincs semmi sorszámozás, nincs semmi elveszés elleni védelem. UDP esetében minden csomagvesztéssel kapcsolatos feladatot arra az alkalmazásra bízzák, amelyik küldi/fogadja a csomagokat. (CRC ellenőrzés ugyan van, de az csak csendes csomageldobáshoz vezet.) Nem tárol. Ahogy beérkezik egy UDP csomag az rögtön megy is tovább az alkalmazás réteg felé. RFC 768 tartalmazza az UDP meghatározását.

1.1.5 ICMP: Internet Control Message Protocol.

Az ICMP csomagok fognak visszajelzéseket adni mind a routolási hibákról, mind a csomagszállítási hibákról. Megközelítőleg egy tucat ICMP üzenettípus létezik. Minden üzenettípus IP-csomagba burkolva vándorol a hálózatban. A protokoll a hálózat tesztelésére is használható, ICMP Az ECHO REQUEST (visszhangkérés ('ping')) üzenet küldésével. Az ECHO üzenet kézhezvételét követően a címzettnek egy ECHO REPLY üzenettel kell válaszolnia. Az adott címzett elérhetőségét és működőképességét lehet megvizsgálni Az ICMP üzenet típusait és a protokoll működését az RFC 792 határozza meg.[3]

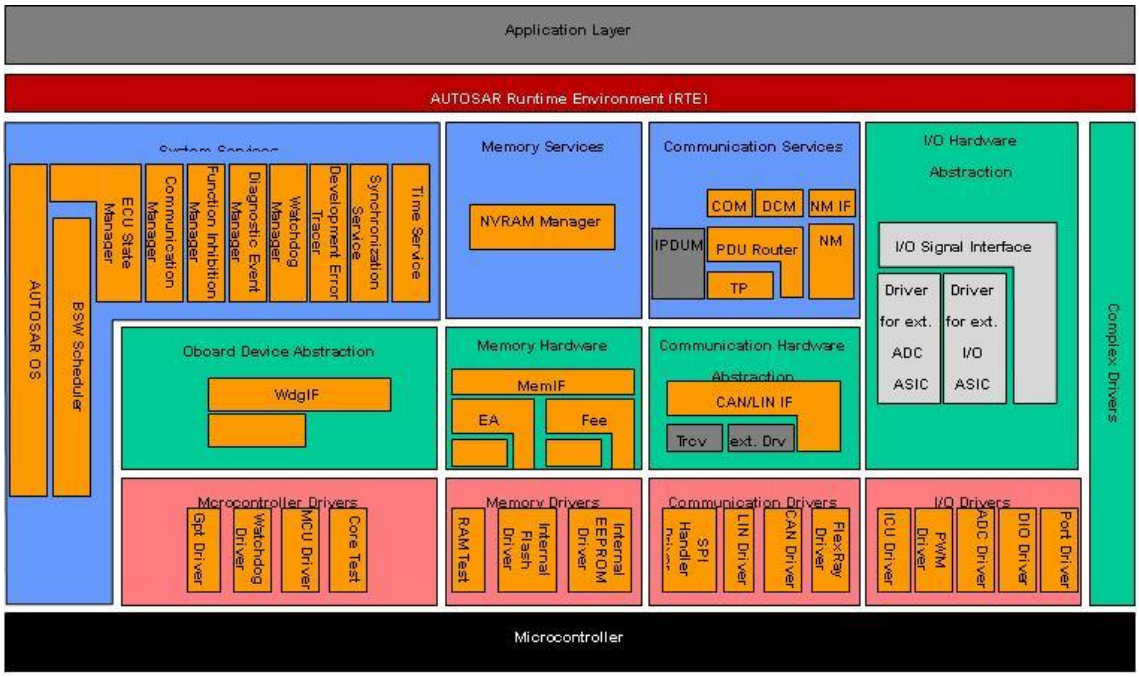
1.1.6 DHCP: Dynamic Host Configuration Protocol

A DHCP egy kliens-szerver UDP feletti protokoll. A protokoll lehetővé teszi a hálózaton levő számítógépek számára, hogy a DHCP szervertől kérjenek és kapjanak meg minden olyan szükséges hálózati beállítást, amely az adott hálózaton való biztonságos működéshez szükséges. A DHCP alkalmazásának több előnye is van. Az egyik, hogy nem szükséges a rendszergazdának minden egyes gépen egyenként kézzel elvégeznie a szükséges hálózati beállításokat. A kiosztott, de bizonyos beállított időn belül fel nem használt IP-címek bejegyzései törlésre kerülnek és azok másik gépek számára kioszthatóak.

2 AUTOSAR kommunikációs verem felépítése

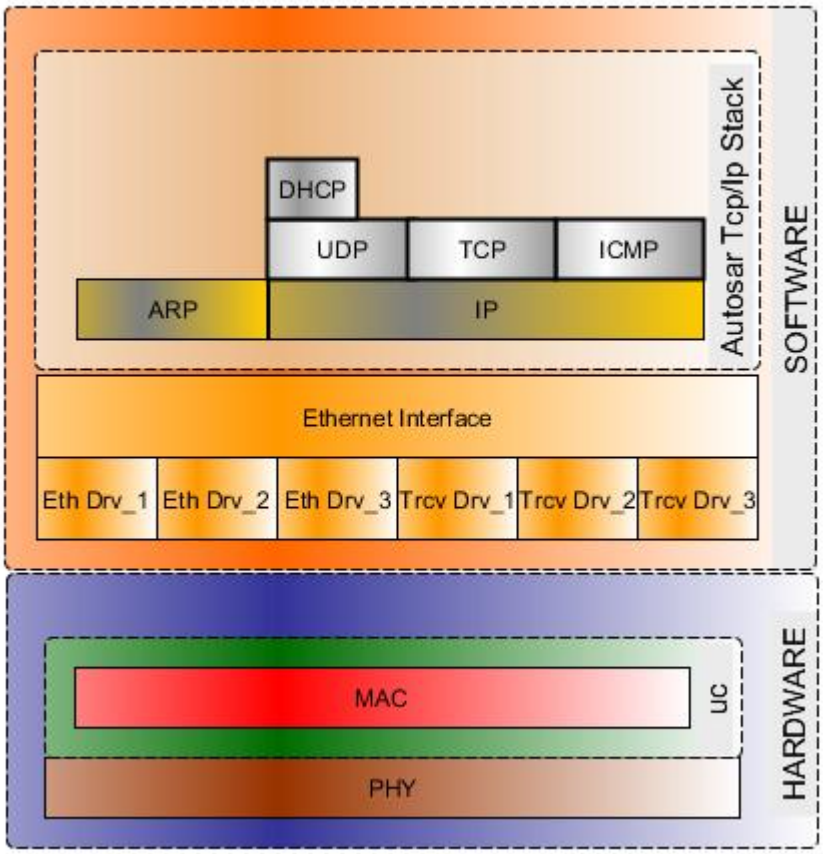
2.1 Az AUTOSAR

Az AUTOSAR autóipari nyílt rendszer architektúra Az architektúra réteges felépítésű szigorú megkötések vonatkoznak rétegek által megvalósított működésre. Az AUTOSAR alapja hogy az ECU funkciókat egymástól és a hardvertől független software modulok valósítják meg. A legalsó rétegben vannak megvalósítva a mikrokontroller függő modulok, driverek. Az alsó rétegre épülő rétegben a mikrokontroller belső perifériáihoz és hozzájuk kapcsolható külső eszközökhöz való hozzáférést biztosítja. Ez a réteg már független a kontrollertől, de még erősen függ az ECU felépítésétől. A szolgáltatás rétegben olyan funkciók vannak, mint például a kommunikáció és memória menedzsment, különböző diagnosztikai és watchdog szolgáltatások[4]. Ettől a rétegtől kezdve, a kontrollertől és az ECU-tól független modulok helyezkednek el. 2. Az AUTOSAR szoftver architektúrája 2 ábra mutatja az AUTOSAR szoftver architektúra általános blokkvázlatát.



2. Az AUTOSAR szoftver architektúrája

2.2 AUTOSAR TCP/IP kommunikációs verem

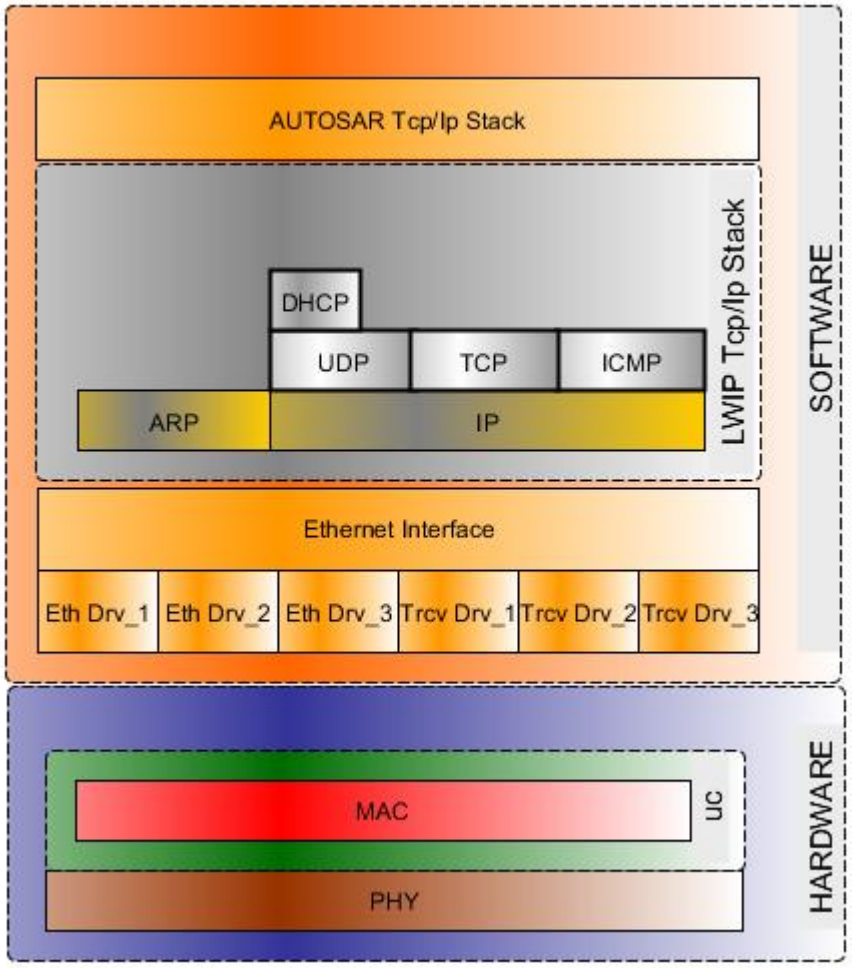


3. Az AUTOSAR kommunikációs verem szoftver architektúrája

Az AUTOSAR által definiált TCP/IP verem blokk vázlata a 3. ábrán látható. Lényeges eltérés az első ábrán látható általános felépítéstől, hogy az AUTOSAR több Ethernet és több fizikai illesztő drivert tud egyidejűleg kezelni. A több driver kezelésére az AUTOSAR bevezetett egy Ethernet interfész réteget, ami egységes felületet nyújt a hozzájuk kapcsolódó hardver funkciók elérésére. Továbbá a fenti ábrán jelöltem, hogy a tényleges hardver felépítése, amin a feladat elkészítése során használtam olyan, hogy a mikrokontroller tartalmazza az Ethernet vezérlőt (MAC) és a fizikai illesztő (PHY) külön áramkör. A két áramkör közötti kommunikáció az MII interfészen keresztül történik.

2.3 Nyílt forrású TCP/IP megvalósítás illesztése az AUTOSAR szoftver architektúrába

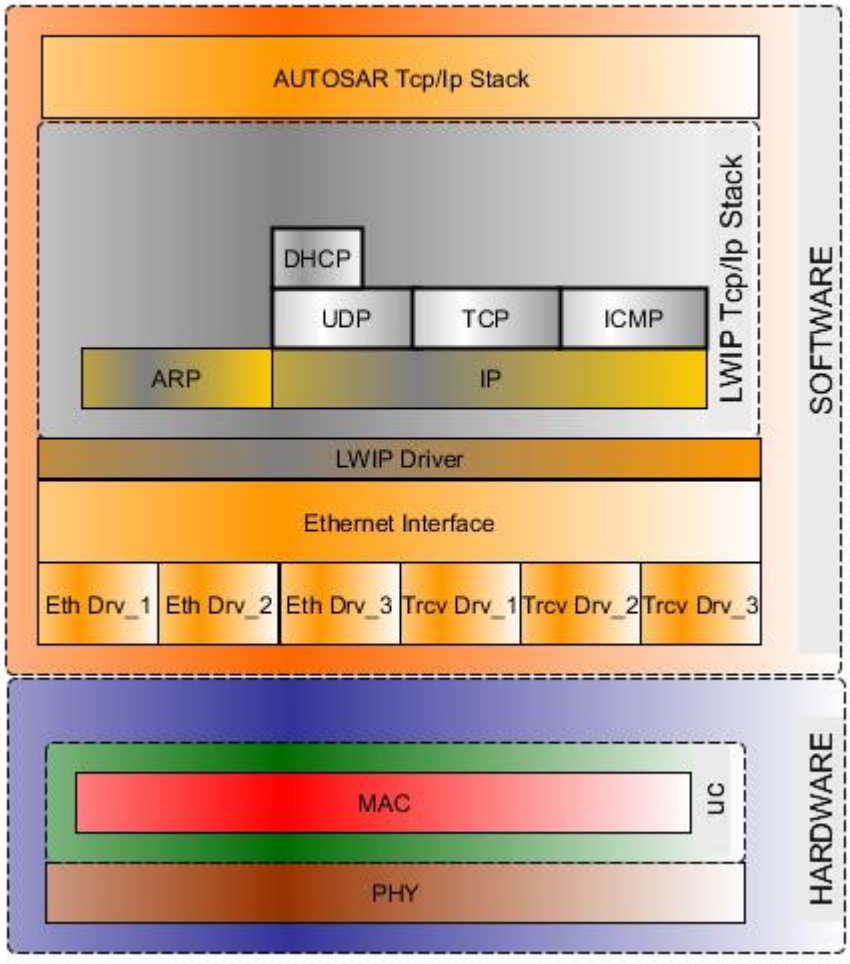
Feladatom az volt, hogy válasszak egy nyílt TCP/IP vermet, ami megvalósítja azokat a protokollokat, amelyeket az AUTOSAR megkíván. A Lightweight TCP/IP stack-et (továbbiakban csak LWIP) választottam, mivel széleskörűen elterjedt különböző beágyazott rendszerek területén (könnyű hozzá példaalkalmazást találni), megvalósítja a feladathoz szükséges összes protokollt és elég jól konfigurálható, testre szabható.



4. Az LWIP beillesztése az AUTOSAR szoftver architektúrába

A 4. Ábra mutatja, hogy az LWIP logikailag az AUTOSAR Ethernet interfész és az AUTOSAR TCP/IP rétegek közé került. Így „felülről” nézve az AUTOSAR TCP/IP modul a szabványos interfészt fogja kínálni a felsőbb szoftver moduloknak, amik majd használni akarják, de az LWIP és az AUTOSAR TCP/IP eltérő adatkezelése és a

függvények eltérő paraméterezése nem teszi lehetővé, hogy a megvalósítás csupán annyiból álljon, hogy az AUTOSAR TCP/IP meghívott függvénye meghívja a hozzátartozó LWIP függvényt. Továbbá az LWIP-t alulról is illeszteni kellett az AUTOSAR Ethernet interfész modulhoz. Itt a probléma megoldása egy kicsit egyszerűbb volt mivel az LWIP készítője (készítői) gondoltak arra, hogy majd különböző szoftverrendszerekbe lesz használva, így definiáltak egy úgynevezett „netif" struktúrát, ahova többek között azoknak a függvényeknek a kezdőcímei kerülnek, amik az Ethernet modul kezelését teszik lehetővé (Pl: Ethernet keret küldése fogadása). Ezek a függvények jól elkülöníthetőek ezért számukra egy további réteget definiáltam LwIP_driver néven. Így a tényleges így a tényleges szoftver architektúrát az 5. ábra mutatja be.



5. Az LWIP és az AUTOSAR szoftverarchitektúrája

3 AUTOSAR Ethernet Interfész modul

3.1 Ethernet Interfész modul konfigurációja

Az interfész modul feladata, hogy ha több Ethernet vezérlővel, vagy több fizikai illesztővel rendelkezünk, akkor ezek kezeléséhez egyszerű szabványos felületet nyújtson**.** Ahogy a legtöbb AUTOSAR modul esetében, itt is az egyik legfontosabb teendő a konfigurációs lehetőségek áttekintése. A konfiguráció két részt tartalmaz, az egyik, ami olyan konstansokat tartalmaz, ami alapvetően meghatározza a modult és a modul fordításakor fognak érvényesülni (Pl.: a modul egyes részei bele kerüljenek-e a kódba vagy ne). AUTOSAR terminológia szerint *Pre-compile time* konfigurációs osztályba tartozó paraméterek. Mivel ezek a paraméterek fordítás után már nem változhatnak ezért általában C makrók formájában kerülnek megvalósításra. A másik fajta konfigurációt a modul a futási időben fogja megkapni az init() függvény meghívásával és ezek a modul működését fogják meghatározni. Ezek a paraméterek pedig *Post-build time* konfigurációs osztályba tartoznak.

A legfontosabb konfigurációs konstansok az interfész modulhoz:

- A transmit bufferek száma (ETHIF_MAX_TX_BUFFS_TOTAL)
- Maximális Fizikai illesztők száma (ETHIF_MAX_TRCVS_TOTAL)
- Megszakítások engedélyezése tiltása (ETHIF_ENABLE_RX_INTERRUPT/ETHIF_ENABLE_TX_INTERRUPT)
- Milyen időközönként ellenőrizze a kapcsolat állapotát (ETHIF_TRCV_LINK_STATE_CHG_MAIN_RELOAD)
- Várakozás határideje, amikor lekérdezéses módba várakozik egy csomag megérkezésére (ETHIF_MAIN_FUNCTION_RX_TIMEOUT)

A modul megvalósításakor a szabványban meghatározott konstansokon felül meghatároztam további 6 paramétert:

- Engedélyezzük-e a több Ethernet drivert (ETHIF_MULTIPLE_DRIVER)
- Engedélyezzük-e a több Fizikai illesztő drivert (ETHIF_MULTIPLE_TRANSCEIVER)
- Maximális Ethernet vezérlők száma (ETHIF_MAX_CTRL_SUPPORTED)

- Maximális függvények száma, amik akkor hívódnak meg, amikor Ethernet csomag érkezik (ETHIF_MAX_RX_IND_FUNC_SUPPORTED)
- Maximális függvények száma, amik akkor hívódnak, meg amikor Ethernet csomagot küldünk (ETHIF_MAX_TX_CONF_FUNC_SUPPORTED)
- Maximális függvények száma, amik akkor hívódnak, meg amikor megváltozik a kapcsolat állapota (ETHIF_MAX_TRCV_LINK_CHG_FUNC_SUPPORTED)

A másik konfigurációból a következő elemeket emelném ki:

Felsőbb réteg értesítésére szolgáló un. Callback függvény kezdőcímei. Ez alapján fogja az interfész modul meghívni a felsőbb réteg megadott függvényeit a következő esetekben:

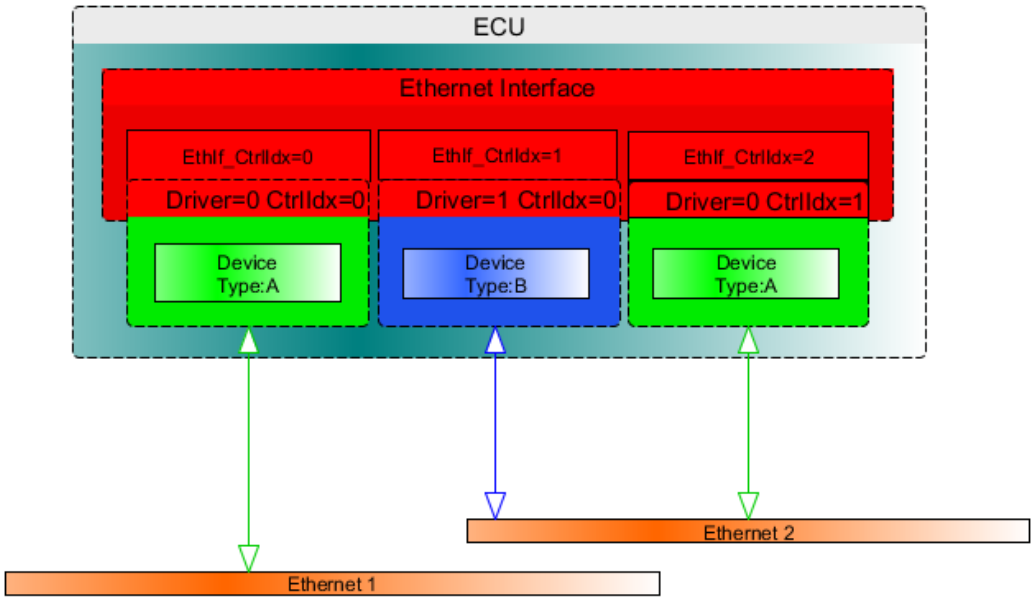
- Ethernet csomag érkezett (EthIfRxIndicationFunctions)
- Ethernet csomag el lett küldve (EthIfTxConfirmationFunctions)
- Megváltozott a kapcsolat státusza (TrcvLinkStateChgFunctions)

Az AUTOSAR a fentieknél több konfigurációs lehetőséget definiál. Nem említettem meg azokat a lehetőségeket, amivel a modul egyes részei ki és bekapcsolhatóak, mivel a feladatomhoz elegendőnek bizonyult a modul legminimálisabb beállítása is. Továbbá a szabvány nem követeli meg az összes konfigurációs paraméter felhasználását az adott modulban

3.2 Ethernet kontroller indexelése

A fentiekben említettem, hogy az interfész modul jelentősége akkor látszik igazán, ha több Ethernet vezérlővel, vagy több fizikai illesztővel rendelkezünk egy adott rendszerben. Ha csupán egy Ethernet vezérlőnk és egy fizikai illesztőnk van, akkor az interfész modul függvényei egyszerűen csak meghívják a hozzá tartozó driver függvényeit. Pl.: Az interfészmodul EthIf_ControllerInit(uint8 CtrlIdx, uint8 CfgIdx) függvénye meghívja az Ethernet driver: Eth_ControllerInit(uint8 CtrlIdx, uint8 CfgIdx) függvényét ugyan azokkal a paraméterekkel, amelyeket ő kapott. Viszont ha több modullal rendelkezünk, akkor paraméterben megadott kontroller indextől függően kell eldöntenünk, hogy melyik driver adott függvénye legyen meghívva és milyen index paraméterrel. Az AUTOSAR következő módon definiálja az indexelést[5]: A 6. ábrán az látszik, hogy ha 0-s kontroller indexel, hívjuk meg az Ethernet interfész függvényét, akkor az ugyan úgy 0-s indexel fogja meghívni az alatta lévő modul adott függvényét.

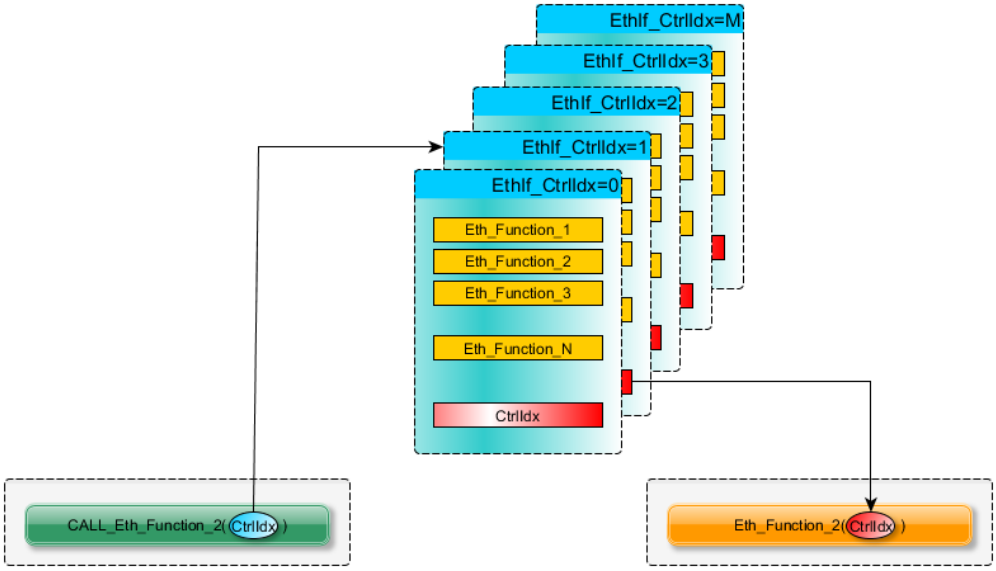
Viszont ha 1-es indexel, hívjuk meg az interfész modul függvényeit, akkor egy másik driver függvényét kell meghívnia, de 0-s kontroller indexel.



6. Az Ethernet interfész indexelése

Természetesen az interfész modulnak nem csak a fenti példa összeállítást kell tudni kezelni, hanem a legkülönfélébb konfigurációkat is. Így az indexelés problémájának a megoldására a következő módszert használtam.

Mivel az AUTOSAR pontosan definiálja, hogy egy Ethernet drivernek milyen függvényei lehetnek[6], ezért definiálni tudtam egy struktúrát, ami ezeknek a függvényeknek a kezdő címét tárolja (függvény pointer) és a végén még tárolja a hozzá tartozó lokális kontroller indexet. Ezután létre hoztam egy tömböt, aminek az elemeit a fent említett struktúrák alkotják, de a tömb indexelése azonos az interfész modul indexelésével.(7. ábra).



7. Az Ethernet interfész indexelésének megvalósítása

A konkrét megvalósítás egy függvényre:

```
#if(ETHIF_MULTIPLE_DRIVER == STD_OFF)

#define CALL_Eth_TxConfirmation(CTRLIDX) Eth_TxConfirmation((CTRLIDX))

#else

#define CALL_Eth_TxConfirmation(CTRLIDX) \
EthDrvS[CTRLIDX].Eth_TxConfirmation(EthDrvS[CTRLDX].CtrlIdx))

#endif
```

A fenti kódrészletben látszik, ha nincs engedélyezve több driver használata, akkor egyszerűen csak ugyanazzal a paraméterrel meghívja az adott driver függvényét. Ha engedélyezve van több driver, akkor a driverek függvényei az EthDrvS[]tömbben vannak eltárolva és a tömb indexelése fogja meghatározni,hogy melyik driver legyen meghívva (EthDrvS[CTRLIDX].Eth_TxConfirmation()). A paraméterként átadott kontroller index is ebből a tömbből lett meghatározva: EthDrvS[CTRLIDX].CtrlIdx. Ezzel a módszerrel áthidalható az a probléma, hogy összetett rendszer esetén az interfész modul által használt indexelés megfeleltethető legyen az egyes driverek által használt indexelésnek.

3.3 Ethernet Interfész modul működése

Az Interfész modul két „main” funkcióval rendelkezik. Az AUTOSAR szoftvermodulok „main” függvényei olyan függvények, amiket majd meghatározott

időnként (periodikusan) meghívnak majd. Így alkalmasak olyan feladatokra, mint például rendszeresen ellenőrizni a kapcsolat állapotát, vagy érkezett-e új csomag az Ethernet vezérlő felől. Az egyik „main” az EthIf_MainFunctionTx()a másik EthIf_MainFunctionRx().

3.3.1 EthIf_MainFunctionTx()

Ha a csomagküldés utáni megszakítás nincs engedélyezve, akkor ez a függvény értesíti a felsőbb szoftver rétegeket arról, hogy egy csomag küldése sikeres. Ha a megszakítás engedélyezve van, akkor ezt a feladatot a megszakítás kezelő rutin látja el. További feladata a kapcsolat állapotának ellenőrzése.

3.3.2 EthIf_MainFunctionRx()

Ha a csomagfogadás utáni megszakítás nem engedélyezett, akkor az EthIf_MainFunctoinRx() függvény a konfigurációban megadott ideig várakozva figyeli, hogy érkezett-e csomag az Ethernet vezérlő felől. Ha igen akkor a csomag kezdőcímét tovább adja a felsőbb rétegeknek, ha nem érkezett csomag, akkor a konfigurációban megadott „Timeout” idő lejárta után kilép a várakozásból. Ha a megszakítás engedélyezve van, akkor a megszakítás kezelő rutin akkor lesz meghívva, amikor csomag érkezik az Ethernet vezérlő felől és a megszakítás-kezelő fog gondoskodni arról, hogy a felsőbb szoftverrétegek meg kapják a beérkezett csomag kezdőcímét.

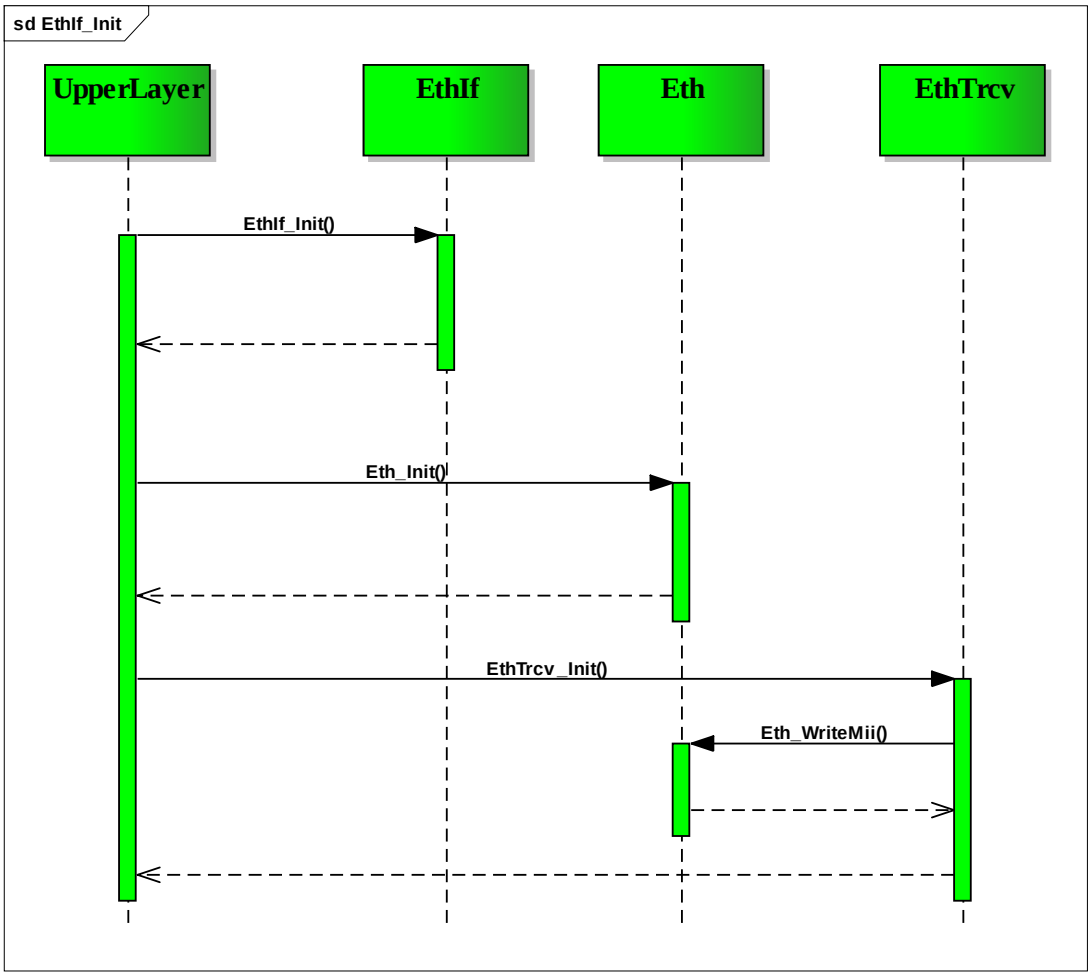
Az Ethernet interfész működését szekvencia diagramokon keresztül ábrázolom. Egy Ethernet alapú kapcsolat elindítása a hozzá kapcsolódó modulok inicializációjával kezdődik.

3.3.3 Inicializáció

Az AUTOSAR modulok inicializációjában olyan egyszer végrehajtandó feladatokat vannak, amelyek szükségesek a az adott szoftvermodul indulásához és működéséhez. A legtöbb AUTOSAR modul init() függvényének leírása tartalmazz azt a követelményt, hogy az előre megadott konfigurációs struktúrához való hozzáférés egy lokális változón keresztül történjen. Vagy elmentjük a konfigurációban szereplő paramétereket lokális változókban, vagy a kezdőcímét tároljuk el. Erre azért van szükség mivel többféle konfigurációval is rendelkezhetünk, de az init() függvénynek ezek közül csupán egy lesz átadva. Tehát itt dől el, hogy a modul melyik konfiguráció szerint fog működni. Ha a modul rendelkezik állapotokkal, akkor az init() függvényben

lesz beállítva a kezdőállapota. Az hogy a fentieken kívül milyen feladatokat kell még végrehajtani egy adott modul inicializációja során az a modul működésétől függ. Az Ethernet interfész esetében elegendő volt a konfigurációs struktúra címének egy lokális változóba történő mentése.

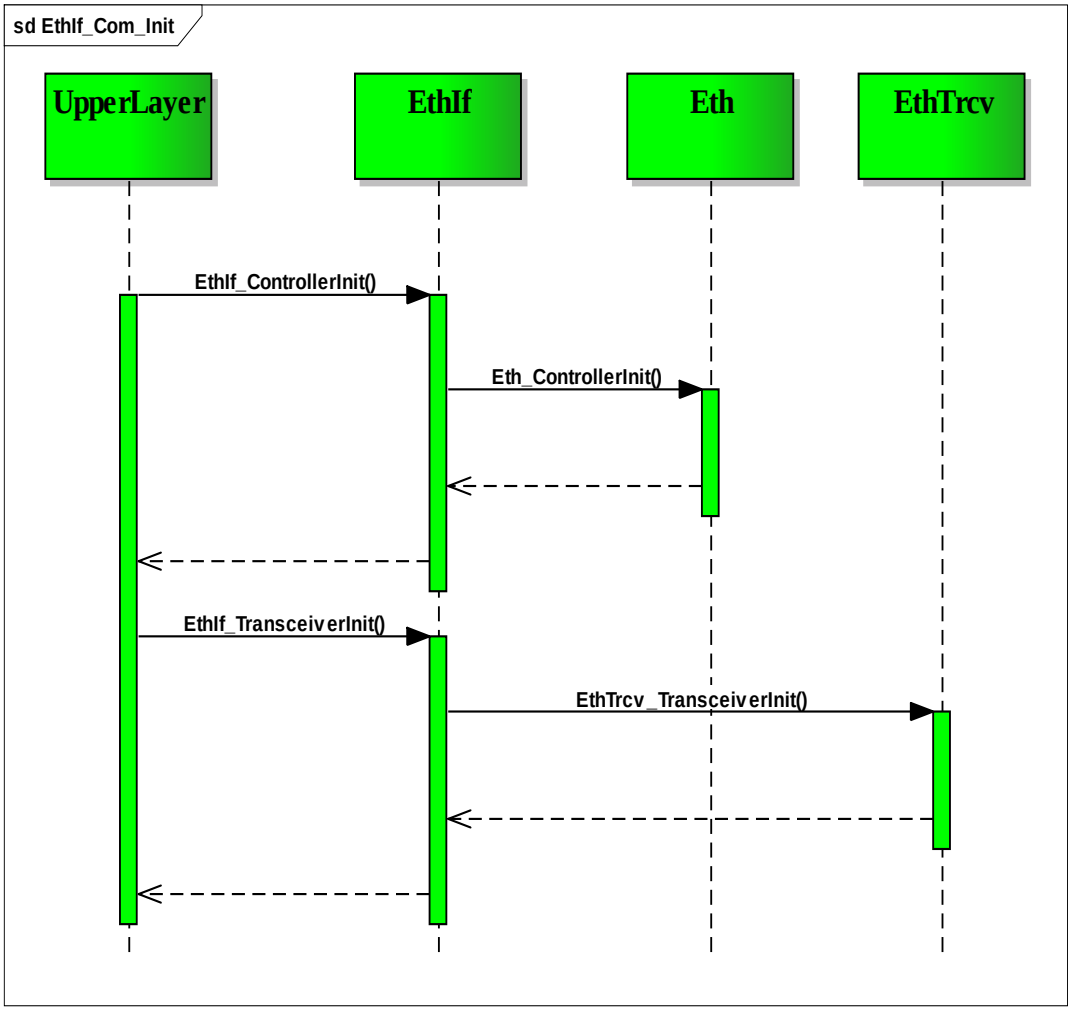
Két modul szükséges az Ethernet kapcsolathoz. Az egyik a fizikai illesztő driver (EthTrcv) és az Ethernet vezérlő driver (Eth). Ezekre épül az Ethernet interfész, ami egységes felületet nyújt akkor is, ha több Ethernet vezérlőnk vagy több fizikai illesztőnk van. Viszont a működéshez mindegyik modult inicializálni kell a 8. ábra szerint[5]. Egy felsőbb szoftverrétegből ahol elindítjuk, az Ethernet kapcsolatot meghívjuk mind három modul init() függvényét paraméterként pedig az adott modul konfigurációját, mint konfigurációs struktúrát adjuk meg.



8. Az Ethernet interfész inicializációja

3.3.4 A kommunikáció inicializálása

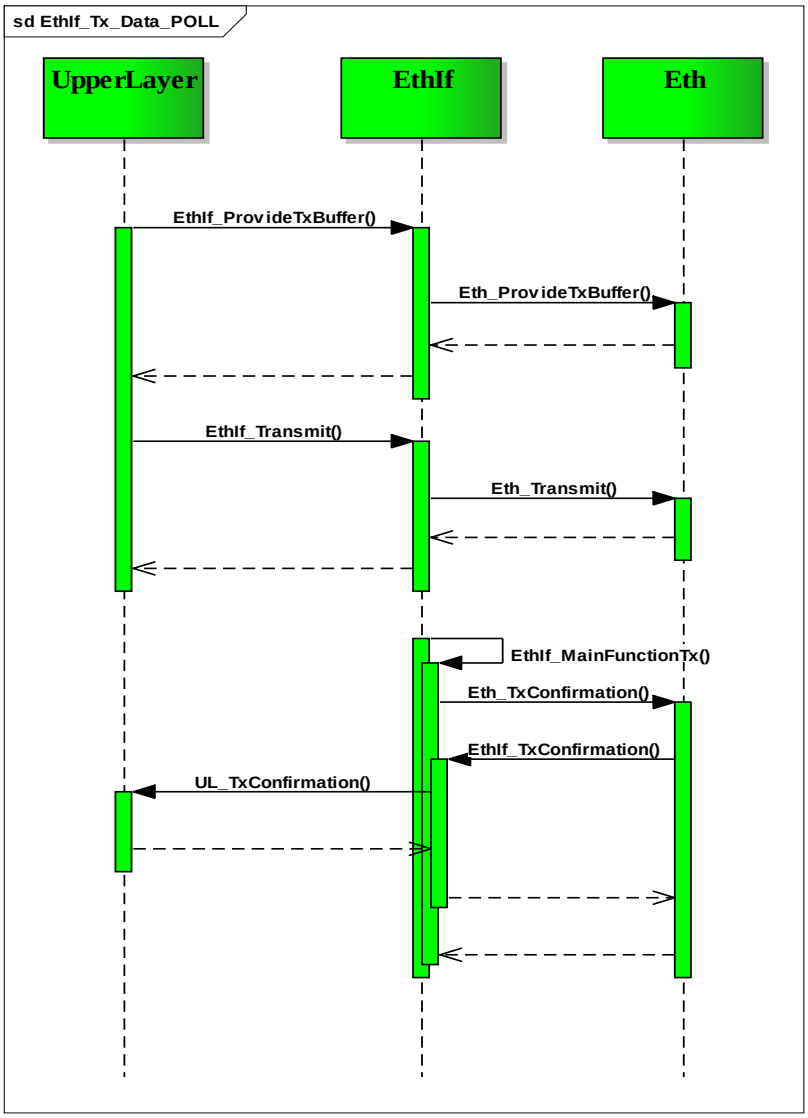
Itt az egyes Ethernet vezérlők és fizikai illesztők lesznek inicializálva. A 8. ábrán a szoftver modul inicializációja van ábrázolva. A konfigurációba van meg adva, hogy a modul hány Ethernet vezérlőt vagy fizikai illesztőt kezel. A 9. ábrán pedig már egy konkrét Ethernet vezérlő vagy fizikai illesztő incializálása történik. Felsőbb rétegből közvetlenül nem hívunk függvényeket az Eth, vagy EthTrcv modulokból, csak az Ethernet interfész modulon keresztül. Azt hogy melyik konkrét Ethernet vezérlő vagy illesztő lesz, kiválasztva az interfész függvénynek paraméterben megadott indexel határozzuk meg. Az indexelés kezelése pedig a 3.2 fejezetben ismertetett módon történik.



9. Az Ethernet kontroller inicializációja

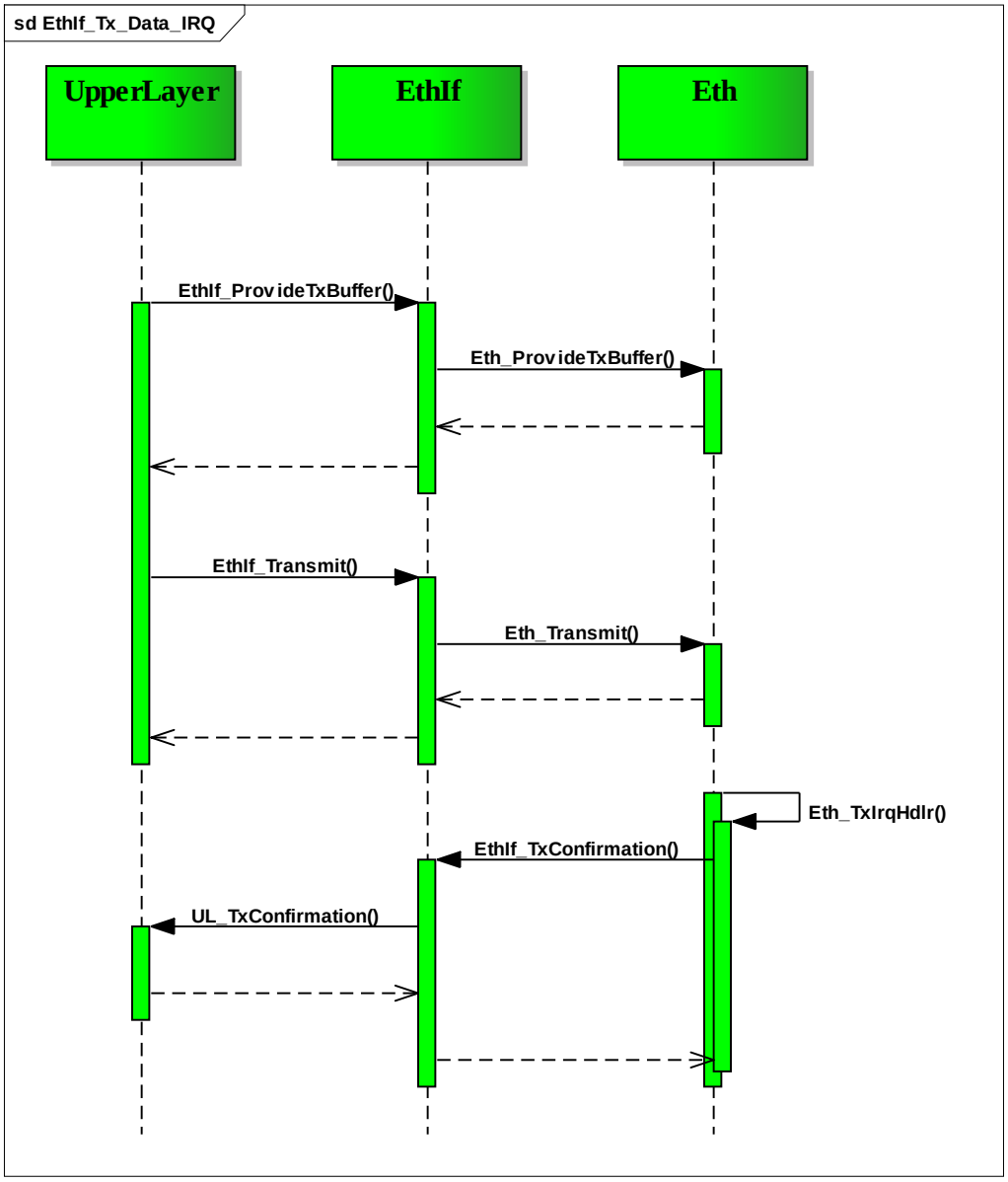
3.3.5 Egy Ethernet üzenet küldésének lépései

Először kérni kell egy memória területet, ahova majd a küldeni kívánt adta lesz bele másolva. (EthIf_ProvideTxBuffer()). A memória terület kérésekor megkapjuk a rendelkezésünkre bocsájtott memóriaterület kezdő címét és egy buffer indexet. Miután belemásoltuk az elküldeni kívánt adatot a kapott bufferbe, meghívjuk a küldést végző függvényt (EthIf_Transmit()). A transmit függvény paraméterének pedig a buffer indexet adjuk meg, így fogja tudni, hogy melyik bufferben találja a küldeni kívánt adatokat. A küldéshez tartozó main függvényben folyamatosan meghívja azt az Ethernet driver Eth_txConfirmation() függvényét, ami eldönti, hogy megtörtént-e az üzenet elküldése, ha igen akkor értesíti a felette lévő modult erről. A 10. ábrán látszik a main funkció működése. A látszólagos elbonyolítás azért szükséges, mert az AUTOSAR rétegzett szoftverarchitektúrája úgy van felépítve, hogy az ideális, ha egy felsőbb szoftver modul közvetlenül csak az alatta lévő modulokból hív függvényeket. Ha egy hardver rétegben egy esemény történik, (Pl.: Egy Ethernet üzenet érkezik vagy üzenet küldése megtörtént) azt csak az adott hardverhez kapcsolódó szoftver modul (driver) fogja észlelni. Így egy ilyen eseményről értesítést úgy lehet küldeni, hogy az alsóbb réteg, hívja meg a felsőbb réteg meghatározott függvényeit (Rx/Tx_Confirmation()). Azokat a függvényeket, amiket felsőbb rétegből az alatta lévő réteg hív meg, callback függvényeknek hívjuk. Itt is van egy olyan szempont, hogy egy alsóbb réteg csak a közvetlenül felette lévő szoftvermodul callback függvényeit hívhatja.



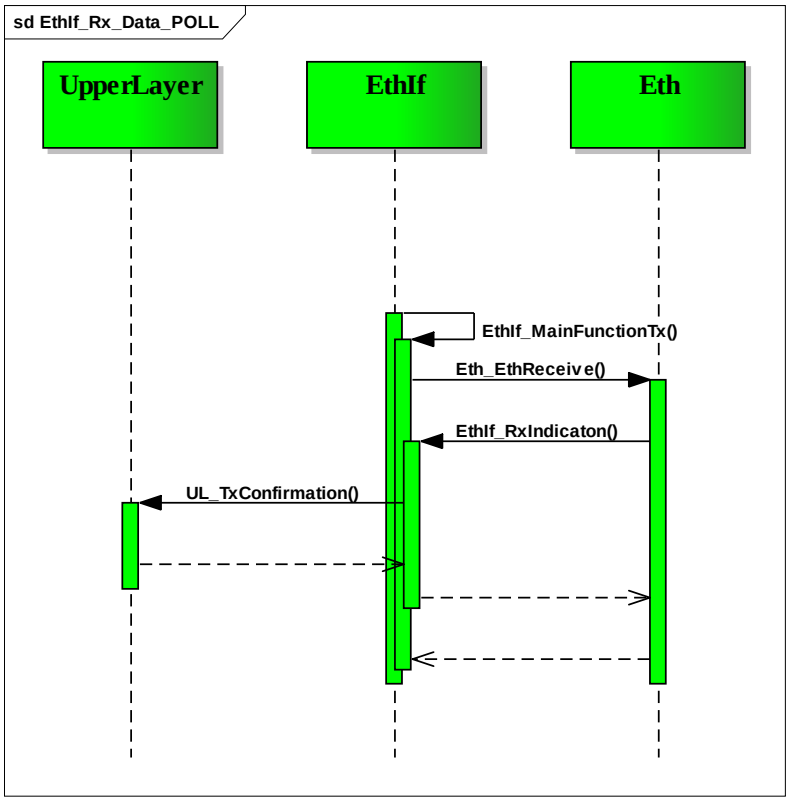
10. Ethrnet interfész Tx Polling

A 11. ábra is egy üzenetküldési folyamatot mutat be. Az üzenet küldés első két lépése ugyan az, mint a 10. ábrán (buffer kérés, és buffer elküldés), de a felsőbb réteg értesítése arról, hogy az üzenet el lett küldve más módon történik. Itt nem ez Ethernet interfész réteg hívja meg azt az Ethernet driver függvényt, ami eldönti, hogy történt-e üzenetküldés, hanem ha üzenetküldés történt, akkor kap egy megszakítást a processzor és a felsőbb rétegek értesítése a megszakítás kezelő rutinba fog megtörténni.



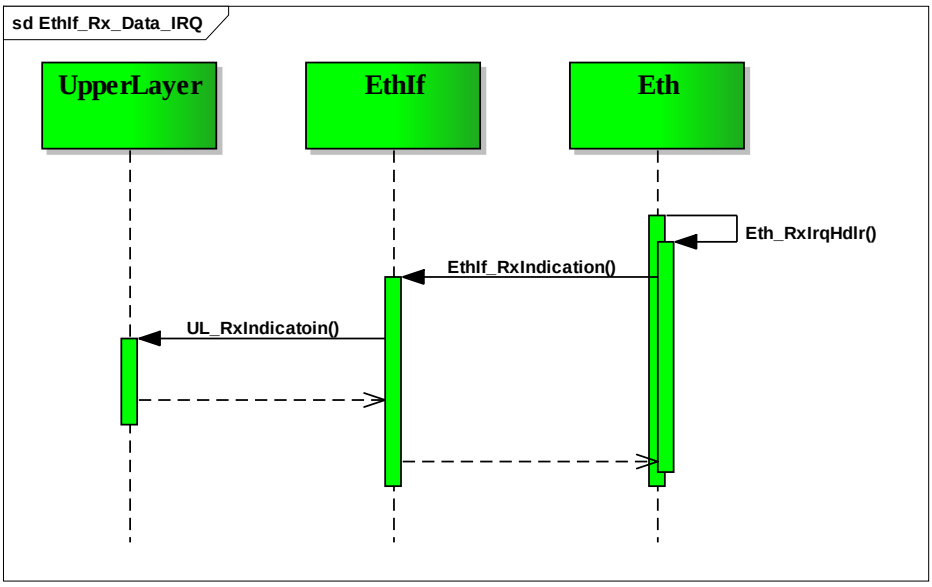
11. Ethrnet interfész Tx IRQ

3.3.6 Adat fogadása



12. Ethrnet interfész Rx Polling

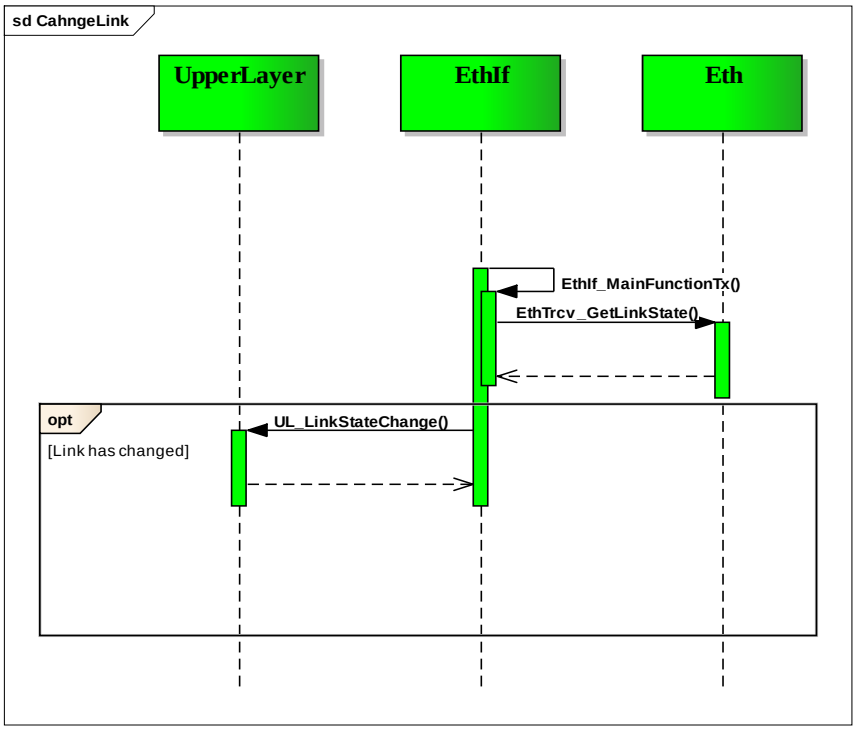
Az Ethernet interfész modul adat fogadáshoz tartozó main funkciójába folyamatosan meghívjuk az Ethernet modul Eth_Receive() függvényét. Ez a függvény csak akkor tér vissza, ha üzenet érkezik. Ezért az interfész modulba van egy maximális idő amíg várakozik az Eth_Receive() függvényre. Az idő letelte után tovább lép. Ha üzenet érkezik, akkor az Ethernet driver modul értesíti alsóbb rétegeket arról, hogy üzenet érkezet. Amikor értesíti az Ethernet interfész réteget akkor az üzenet Ethernet keretformátumnak megfelelő formában van. Ha az upper layer a TCP/IP stack lenne és ő is értesítené a felette lévő réteget, akkor ő már az adatot, UDP vagy TCP csomag formájába adná tovább. A felette lévő rétegek pedig addig továbbítják a kapott adatcsomagokat, amíg el nem érik a azt a modult ahol az adat felhasználásra kerül. A 13. ábrán az üzenet fogadás megszakításos üzemmódban zajlik. Amikor üzenet érkezik a processzor a megszakítás kezelő rutinra ugrik, ahol pedig meghívja az Ethernet interfész üzenet fogadáshoz tartozó callback függvényét, ami a felsőbb réteg callaback függvényét hívja meg.



13. Ethrnet interfész Rx IRQ

3.3.7 A kapcsolat állapotának megváltozása

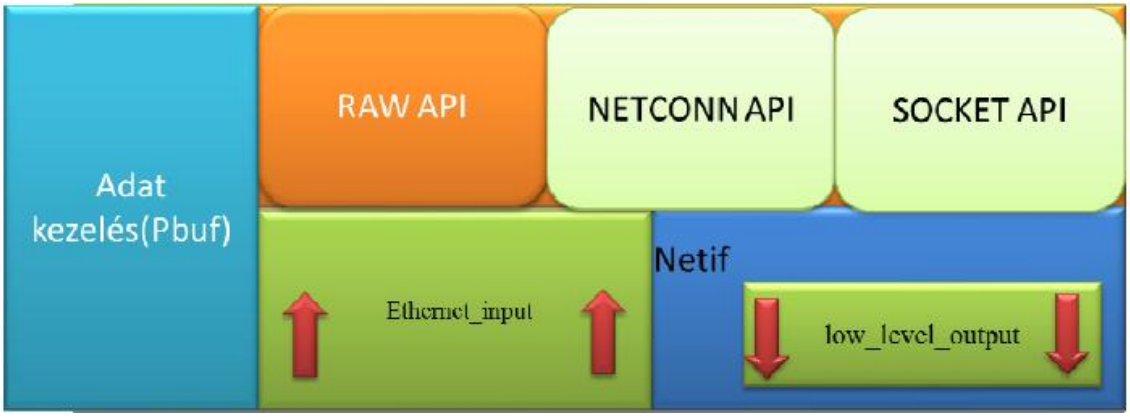
Az Ethernet interfész modul küldéshez tartozó main funkciója folyamatosan lekérdezi a fizikai illesztő driver modulját, hogy a kapcsolat állapotába történt-e változás. Ha igen akkor, meghívja a felette levő modul kapcsolat állapotváltozáshoz tartozó callback függvényt.(14. ábra)



14. Az Ethrnet kapcsolat állapotának megváltozása

4 LWIP TCP/IP stack

Az LWIP driver réteg ismertetése előtt érdemes átnézni magának az LWIP-nek a működését. Az LWIP-t az alatta és fölötte lévő szoftvermodulok kapcsolódási pontjain keresztül ábrázoltam a következő ábrán:



15. Az LWIP interfészei

Az alsóbb rétegek felől az LWIP megad, számunkra egy un. „netif” (network inerface) struktúrát, ahova olyan paraméterek kerülnek, mint IP-cím és az „output” függvény címe. Amikor az LWIP egy csomagot akar küldeni akkor a netif struktúrába megadott output függvényt fogja meghívni és átadni a küldeni kívánt csomag címét. Az Ethernet interfész modul az érkező csomagok címét az ethernet_input() LWIP függvényen keresztül fogja továbbítani, ami a csomag típusától függően(TCP/UDP/ICMP/ARP) fogja azt továbbadni a megfelelő protokoll rétegnek. A csomagok kezelésére az LWIP egy saját adatstruktúrát használ (Pbuf). Ezeket láncolt lista formájába tárolja. Amikor üzenetet kapunk akkor egy ilyen „Pbuf” formájában fog az üzenet rendelkezésre állni. Amikor pedig üzenetet küldünk, akkor létre kell hoznunk egy ilyen Pbuf struktúrát és megadni neki az elküldeni kívánt üzenet kezdőcímét és hosszát. Ekkor a protokoll rétegeken áthaladva az LWIP elkészíti a kívánt csomagot, amit a low_level_output függvényben fogunk bemásolni abba a bufferba, amit végül az Ethernet vezérlő el fog küldeni. Az LWIP-n belül nem történik memória tartalom másolás. Az egyes rétegek csupán a Pbuf-ban szereplő üzenet kezdőcímét adják át egymásnak. Az üzenet elküldése után gondoskodni kell a Pbuf felszabadításáról. Itt nem valódi dinamikus memória kezelésről van szó. Az LWIP konfigurációjában megadhatunk egy maximális Pbuf számot, amit nem léphetünk túl, amikor újabb

üzeneteket küldünk. Ez a konfigurációs paraméter adja meg azt a maximális méretet, amit végül az LWIP statikusan fog lefoglalni a Pbuf-ok számára.

4.1 Pbuf kezelése

Egy Pbuf a következő elemeket tartalmazza:

- A következő Pbuf címe (ha több van, akkor láncolt listaként van tárolva)(next)
- A tárolt adat kezdőcíme (payload)
- A teljes Pbuf listában tárolt adat hossza (tot_len)
- Az adott Pbuf-ban található adat hossza (len)
- Az adat tárolási helyét (RAM/ROM) meghatározó típus (type)

Pbuf kezelése:

- Lefoglaljuk a Pbuf-hoz szükséges területet (pbuf_alloc()): Ahol paraméterbe meg kell adni, hogy melyik protokoll réteg számára lesz lefoglalva:

  1. Szállítási rétegnek (PBUF_TRANSPORT) lehet UDP vagy TCP
  2. IP rétegnek (PBUF_IP)
  3. Kapcsolati rétegnek ebben az esetben Ethernet (PBUF_LINK)
  4. Vagy használhatjuk „nyers” módon, amikor nem adjuk meg, hogy melyik réteg számára lesz lefoglalva (PBUF_RAW)

- Majd meg kell adni a tárolni kívánt adat maximális hosszát
- Utána pedig az elküldeni kívánt adat tárolási módját:

  1. RAM-ban van tárolva (PBUF_RAM)
  2. ROM-ban van tárolva (PBUF_ROM)
  3. Egy RAM-ban tárolt érték referenciáját adtuk-e meg (PBUF_REF)
  4. Egy másik Pbuf-ból származó adatról van-e szó (PBUF_POOL)

- A Pbuf felhasználási helyénél gondoskodni kell a Pbuf felszabadításáról, vagy ellenőriznünk kell, hogy „át halad-e” olyan függvényen, amiben végül is megtörténik a Pbuf felszabadítása (pbuf_free())[7].

4.2 A PCB-k

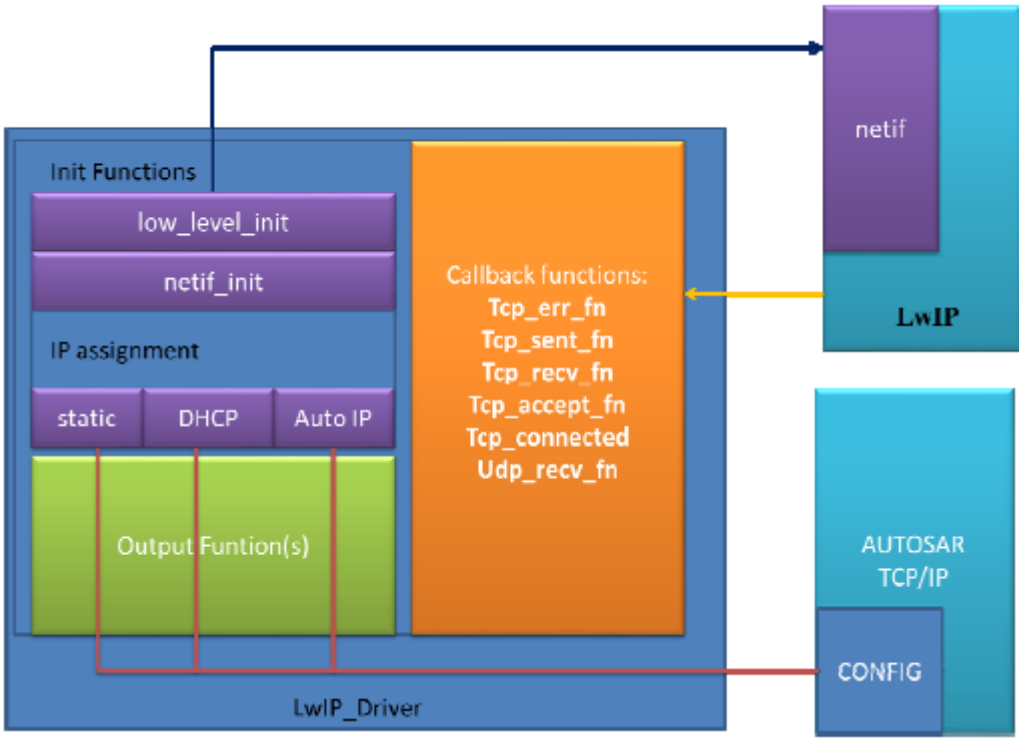
Egy konkrét kapcsolat kezelésére az LWIP un. protocol control blokk-okat (*_pcb) kínál. Külön van az UDP és külön a TCP kapcsolat számára. Egy ilyen pcb tartalmaz minden olyan információt, ami egy adott kapcsolat állapotára vonatkozik és szükséges lehet a kapcsolat fenntartásához. Az UDP kapcsolat leírója (udp_pcb) többek között a következő számunkra fontos elemeket tartalmazza:

- Helyi IP-cím
- Helyi port
- A következő udp_pcb címe (-ha több van -)
- Az udp_recv_fn() callback funkció címe (- akkor hívja meg ha UDP csomag érkezett)
- Az IP-cím, ahova küldeni akarjuk a csomagokat
- Amelyik portra küldeni akarjuk a csomagokat

A TCP kapcsolat számára egy a fentiekhez hasonló struktúrát definiál az LWIP, de a TCP kapcsolat összetettsége miatt jóval több adattagja van a tcp_pcb struktúrának

5 LWIP driver réteg

Az LWIP driver rétegben 3 fontos dolgot valósítottam meg. Az első az „output” függvény, amit a „netif” struktúra kap meg. A második, többfajta IP-cím hozzárendelő függvény a különbözőféleképpen konfigurált AUTOSAR TCP/IP modul számára. Végül az UDP és TCP kapcsolathoz szükséges callback függvények.



16. Az LWIP driver blokkvázlata

Az init() függvények tulajdonképpen a netif struktúrát tölti fel paraméterekkel. Majd a kapcsolat indításakor jutnak érvényre a benne szereplő beállítások.

low_level_init:

- Az Ethernet interfésztől elkéri a fizikai címet (MAC) és eltárolja a netif struktúrába.
- Beállítja az MTU (maximum trasfer unit) értékét ez a maximális egyszerre elküldhető csomag méret (jelenleg 1500 byte)

netif_init:

- Megad egy két karakterből álló azonosítót az adott netif struktúrának (- mivel több network interface is lehet -)
- Beállítja az LWIP által előre definiált kimeneti függvényt, (- használhatunk sajátot is-) ez fogja összeállítani az IP csomagot és továbbítani annak a függvénynek, ami majd Ethernet csomagot készít belőle.
- Végül meghívja a low_level_init függvényt

Az output függvény:

- Az LWIP felől Pbuf –okba kapja meg az elküldendő adatokat, először kér egy buffert az Ethernet interfész felől.
- Majd bemásolja az elküldendő csomagot ebbe a bufferba a Pbuf-ból.
- Majd az Ethernet interfész transmit függvényét használva elküldi az csomagot a célállomásnak.

5.1 IP-címhozzárendelés

Egy IP alapú kapcsolat felépítésénél gondoskodni kell a kapcsolat számára IP-címről. Az AUTOSAR 3 fajta IP-cím hozzárendelést határoz meg[8], amit konfigurációban adunk meg:

- Statikus IP-cím hozzárendelés: amikor „kézzel ” adjuk meg a következő 3 paramétert:

1. IP-cím
2. Alhálózati maszk
3. Alapértelmezett átjáró címe (Ezeket szintén a modul konfigurációba adjuk meg.)

- DHCP alapján történő cím hozzárendelés: itt azon hálózaton ahova csatlakoztatjuk, az eszközt futtatnunk kell egy DHCP szervert, ami gondoskodni fog a csatlakoztatni kívánt eszköz felkonfigurálásáról
- AutoIP esetén a konfigurálandó eszköz saját maga fogja megoldani az IP-cím hozzárendelést egy előre meghatározott tartományból választ magának IP-címet.

Mind három hozzárendelési mód számára külön LWIP IP-címhozzárendelő függvényeket határoztam meg. Az IP-cím hozzárendelő függvények a következő lépéseket hajtják végre:

- A statikus IP-cím hozzárendelés esetén az AUTOSAR TCP/IP modul konfigurációjában meghatározott IP-címeket hozzárendeli a „netif” struktúrához
- A másik két esetben először nullákat rendel hozzá és utána elindítja az LWIP AutoIP vagy DHCP modulját
- A netif struktúrának megadja az Output függvény címét

- Az így kitöltött network interface struktúrát érvényre jutatja. A továbbiakban ez alapján fog működni az LWIP.

5.2 Az LWIP_Driver függvényei

A callback függvényeket az LWIP hívja meg az egyes események bekövetkezésekor. Nekünk kell ezeket definiálni, hogy mi történjen, ha hiba van, vagy ha a csatlakozás bekövetkezett stb. Itt az LWIP jól követi az AUTOSAR azon elgondolását, hogy ezen eseményekről értesíteni kell az AUTOSAR TCP/IP feletti rétegét (SocketAdaptor réteg). Mivel a TCP kapcsolat összeköttetéses és az egyes elküldött üzenetekről visszaigazolást vár, ezért sokkal, több adminisztrációval jár egy TCP kapcsolat fenntartása, mint egy UDP kapcsolaté ebből kifolyólag sokkal több callback függvény is tartozik egy TCP kapcsolathoz.

- Tcp_connected(): Ezt a függvényt akkor hívja meg az LWIP, amikor a TCP kapcsolat felépült olyan módon , hogy sikerült csatlakozni egy szerverhez. Itt regisztrálja a TCP protocol control blockba a következő függvényeket: Tcp_recv_fn(), Tcp_sent_fn(), Tcp_err_fn(). Továbbá értesíti a felsőbb AUTOSAR réteget a csatlakozásról (SoAd_TcpConnected()).
- Tcp_accept_fn(): Értesíti a felsőbb AUTOSAR réteget a csatlakozásról (SoAd_Accepted()).Akkor hívódik, meg amikor a kapcsolat szervernek van beállítva és elfogad egy kapcsolódási kérelmet, így felépítve a TCP kapcsolatot. Itt regisztrálja a TCP protocol control blokkba a következő függvényeket: Tcp_recv_fn(), Tcp_sent_fn(), Tcp_err_fn().

5.2.1 A TCP protocol control blokk callback függvényei

- Tcp_sent_fn(): Akkor hívódik meg, amikor egy TCP üzenet el lett küldve és értesíti a felsőbb réteget erről (SoAd_TxConfirmation()).

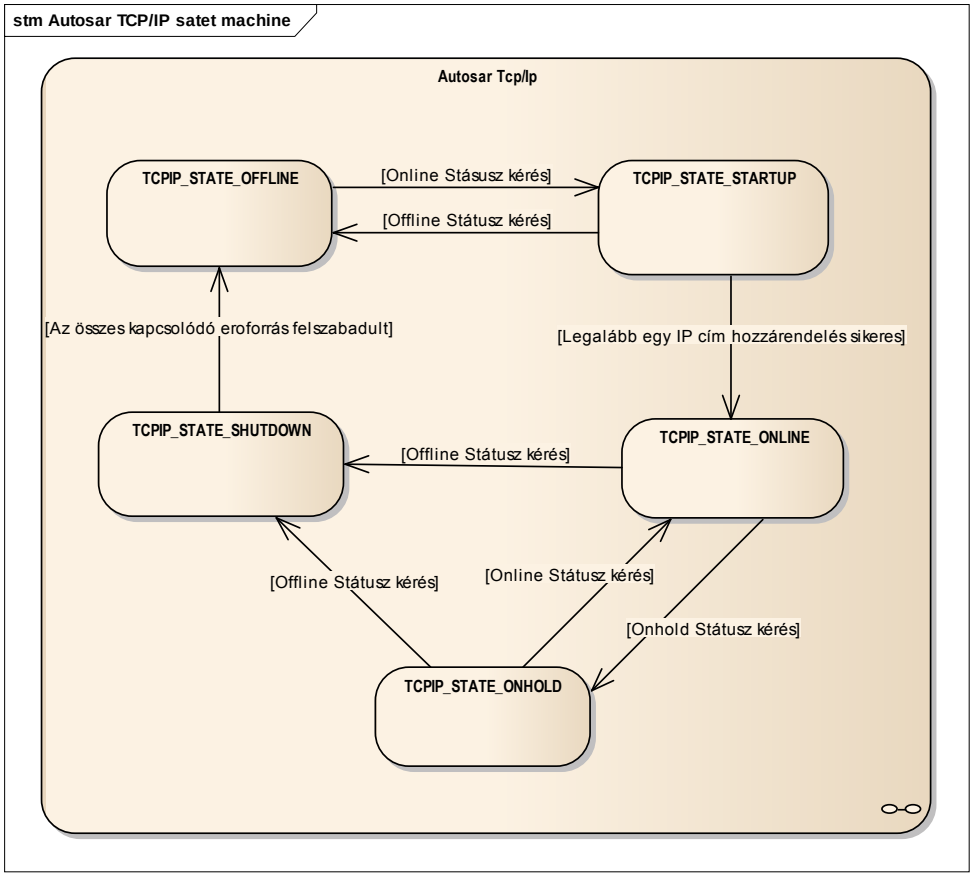
- Tcp_recv_fn(): Akkor lesz meghívva amikor TCP üzenet érkezik és erről értesíti a felsőbb réteget(SoAd_RxIndication()). Az üzenet Pbuf-ba érkezik, amiből a hasznos tartalmat adjuk csak tovább a felsőbb szoftver rétegnek. Utána fel kell szabadítani a Pbuf-ot.
- Tcp_err_fn(): Az LWIP akkor hívja, meg amikor a TCP kapcsolat során valamilyen hibát észlel. Itt 3 hibát emelnék ki, amiről értesíteni kell az AUTOSAR felsőbb rétegét:
  - TCP_IP_RESET: amikor a kapcsolat újra indul
  - TCPIP_TCP_CLOSED: amikor a kapcsolat lezárul
  - TCPIP_TCP_FIN_RECEIVED: amikor a kapcsolat lezárását kérő jelet fogad úgy, hogy a kapcsolat még aktív.

Ezek a hibajelzéseket a függvény meghívásakor paraméterként ad át az LWIP. Itt megjegyezném, hogy ezek a jelzések normál működés során is fellépnek, nem feltétlenül jelentenek hibát a TCP kapcsolatban, de ezekhez csak a hibakezelő függvényen(Tcp_err_fn()) keresztül férünk hozzá és mindezektől függetlenül értesíteni kell a felsőbb réteket ezen események bekövetkezéséről az AUTOSAR szerint (SoAd_TcpIpEvent() függvény meghívása).

6 AUTOSAR TCP/IP verem LWIP-vel

Az AUTOSAR TCP/IP stack megvalósítását is a konfigurációs struktúra maghatározásával kezdtem. Itt ez egy jóval nagyobb feladat volt, mint az interfész modul esetében, sokkal több konfigurációs beállítást és adat típust igényelt. Nincs is mindegyik AUTOSAR által definiált konfigurációs paraméter felhasználva, mivel egy részük megegyezik az LWIP konfigurációjában található paraméterekkel, így azok ott lettek definiálva és felhasználva.

6.1 Az megvalósított TCP/IP modul állapotgépes modellje:



17. Az AUTOSAR TCP/IP Állatgépes modellje

Az AUTOSAR TCP/IP modul használata előtt itt is incializálásal kell kezdeni, ahol a konfigurációjában megadott paraméterek érvényre jutnak. Ezután a modul OFFLINE állapotba kerül. Az állapotgépet megvalósító függvény a modul main

funkciójába van meg hívva. Az állapot váltásokat TcpIp_RequestComMode() függvényen keresztül kérhetjük.

OFFLINE állapotba a következőket figyeli:

- Érkezett–e állapot változás kérés
- Ha ONHOLD állapot kérés érkezet, akkor feltétel nélkül ONHOLD állapotba kerül a modul.
- Ha ONLINE kérés érkezett, akkor pedig START_UP állapotba kerül a modul.
- OFFLINE állapotból csak ONHOLD és ONLINE állapotokba kerülhet.

START_UP állapotban:

- Elindítja az IP-cím hozzárendelést a konfiguráció alapján. (lehet statikus, DHCP, vagy AUTO_IP).
- Figyeli, hogy érkezett-e állapotváltozás kérés:
  - Ha OFFLINE kérés érkezet leállítja a folyamatban levő IP-cím hozzárendelést és OFFLINE állapotba lép.
  - Ha legalább egy IP-cím hozzárendelés sikerült, akkor ONLINE állapotba lép
  - START_UP állapot egy átmeneti állapot, direkt módon nem is kérheti felsőbb modul, hogy lépjen START_UP állapotba, csak az által léphet ebbe az állapotba, ha ONLINE kérés érkezett előtte.

ONHOLD állapotban:

- Felfüggeszti a modul működését olyan módon, hogy az eddig kapott beállítások megmaradnak (pl.: IP-cím)
- Ebből az állapotból kérésre OFFLINE vagy ONLINE állapotba kerülhet.
- Ha ONHOLD állapotban egyetlen IP-cím sem áll a modul rendelkezésére, akkor
- automatikusan OFFLINE állapotba kerül.

SHUTDOWN állapotban:

- Ellenőrzi, hogy az összes IP-cím fel van-e szabadítva (nem használja semmi), ha igen akkor OFFLINE módba lép
- Ez az ONHOLD állapothoz hasonlóan átmenti, állapot direkt módon nem lehet kérni, hogy a modul kerüljön ebbe az állapotba.

ONLINE állapot:

- Legtöbb feladatot ellátó állapot, a modul tulajdonképpen itt végzi el a normál működéshez szükséges feladatokat adatok küldése és fogadása csak ebben az állapotban lehetséges.
- Ebben az állapotban 2 fajta állapotváltás kérés érkezhet. Az egyik ONHOLD: itt felfüggeszti a modul működését olyan módon , hogy a hozzákapcsolódó Ethernet vezérlőt is leállítja. A másik az OFFLINE kérés ekkor felszabadítja a socketeket, lezárja a folyamatban levő kapcsolatokat, megszünteti az IP-cím hozzárendeléseket és belép a SHUTDOWN állapotba.

A modul úgy működik, hogy amikor egy felsőbb modul elindít egy kérést a TCP modul felé (Pl.: TcpTransmit()), akkor meghívja az adott feladathoz szükséges TCP/IP modul függvényét, de a feladat nem hajtódik végre azonnal, hanem a kérést eltárolja és a legközelebbi main funkció végrehajtásakor lesz teljesítve. Ezek alapján a következő kéréseket figyeli ONLINE állapotban:

- IP-cím hozzárendelés kérés
- ARP kérés (IP-cím feloldás)
- TCP csatlakozási kérés
- TCP küldés kérés

A fenti kérések végrehajtása az LWIP megfelelő függvényének meghívásával történik.

- Figyeli, hogy van-e rendelkezésre álló IP-cím, ha nincs, akkor elindít egy OFFLINE állapotváltás kérést.

6.2 A TCP/IP socket

A socket lényegében egy absztrakció, amit a Berkeley BSD unix vezetett be. Ennek segítségével egy alkalmazói program egyszerűen hozzáférhet a TCP/IP protokollokhoz. A socket lényegében egy IP-címből, ami egy gazdagépet, és egy úgynevezett port címből áll, ami egy alkalmazást azonosít az adott gazdagépen. Ezzel a két adattal az Interneten egyértelműen azonosíthatóak az alkalmazások[9]. A megvalósított socket viszont az IP-cím és port párosnál azonban több információt tartalmaz. Az LWIP külön használ az UDP-nek és külön a TCP kapcsolatnak már

említett protocoll controll block-ot (pcb-t). Az AUTOSAR TCP/IP is meghatároz hasonló kapcsolt leírót socket néven.

A megvalósított TCP/IP socket a következő elemeket tartalmazza:

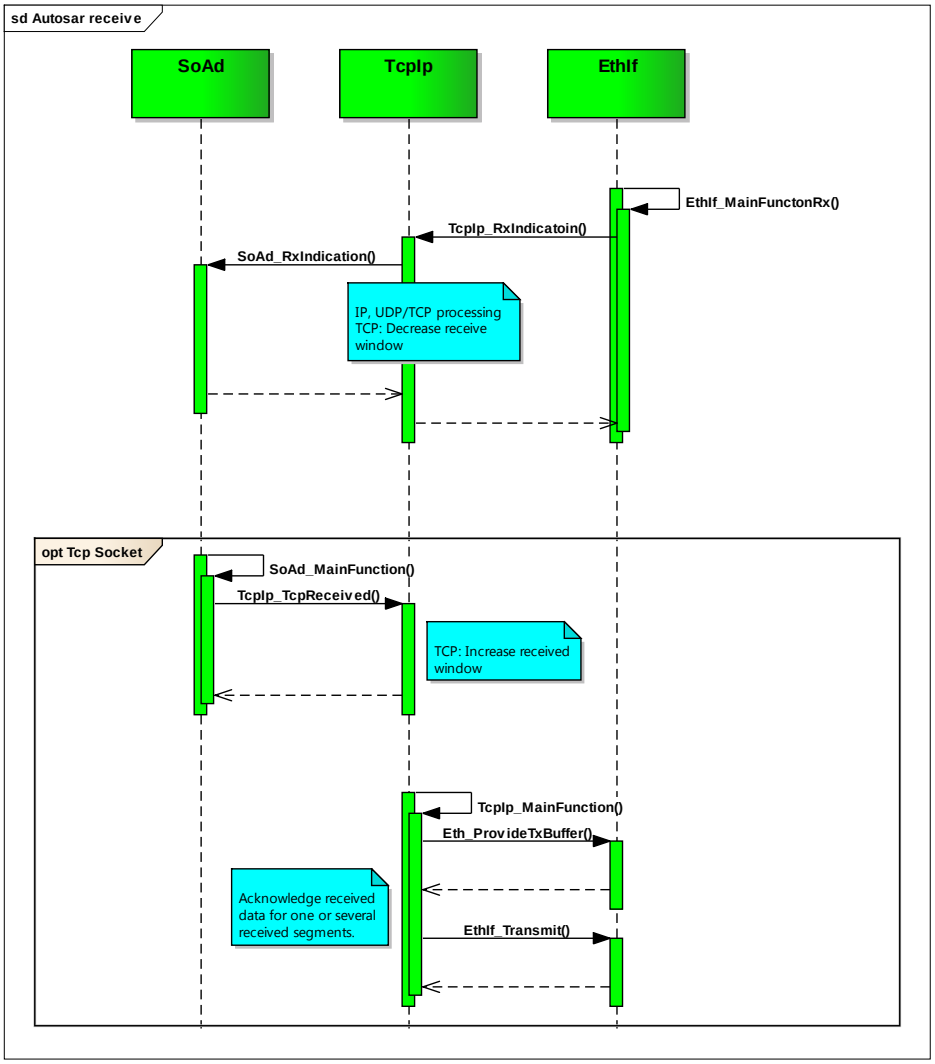
- Milyen típusú a socket (UDP/TCP)
- LwIP TCP protcoll controll block (a socket típusától függően használja)
- LwIP UDP protcoll controll block (a socket típusától függően használja)
- Socket azonosítója
- Egy referenciát oda ahol az UDP és TCP kapcsolatok számát tároljuk
- Megadjuk, hogy server socket-e, vagy nem
- Megadjuk, hogy le van-e foglalva a socket vagy nem
- Hozzá van-e rendelve egy kapcsolathoz vagy nem
- Egy pointert, ami a használt IP-címre mutat

Látszik, hogy a socket típusától függően felhasználtam az LWIP kapcsolat leíróját, és kiegészítettem azokkal az elemekkel, amik szükségesek az AUTOSAR TCP/IP modul működéséhez. A modul inicializálása és ONLINE állapotba lépése után egy TCP vagy UDP kapcsolat létesítésének első lépése egy socket kérése a modultól.

6.3 Kommunikáció a megvalósított TCP/IP stack alatt

6.3.1 AUTOSAR Receive:

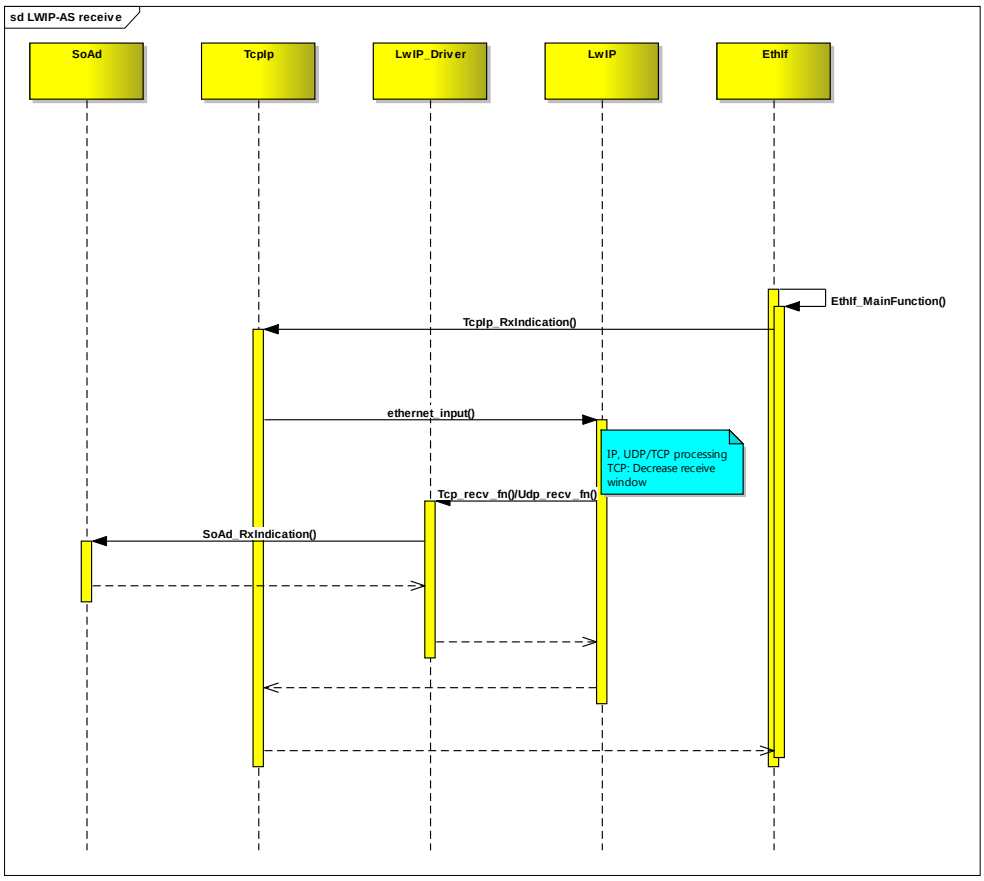
Amikor adat érkezik Ethernet hálózatról akkor 3.3.6 pontban meghatározott módon történik az adat fogadása, azaz a receive folyamat. Az Ethernet interfész modul, vagy EthIf_MainFunctionRx() függvényben egy Ethernet driver függvényhíváson keresztül meghívja az EthIf_RxIndication()-t, vagy az Ethernet driver megszakítás kezelője fogja meghívni az EthIf_RxIndication() függvényt. Ami a felette lévő réteg RxIndicatoin() függvényét fogja tovább hívni, ami jelenleg a TcpIp_RxIndication() függvénye lesz. A függvények egymásnak a paraméterként adják át a fogadott adattal kapcsolatos információkat (fogadott adat kezdőcíme, hossza, melyik Ethernet kontrollerről érkezett). A TcpIp_RxIndication() feldolgozza a kapott adatot(protokoll fejléceket eltávolítja, ha az adat több darabban érkezik, akkor összeilleszti a fogadott darabokat…) majd az így feldolgozott adatot a felette lévő réteg RxIndication() függvény hívásán keresztül továbbítja. TCP kapcsolat esetén viszont a megérkezet, adatról nyugtát kell küldeni a küldő félnek.



18. Az AUTOSAR receive folyamat

6.3.2 A TCP/IP receive folyamata LWIP-vel:

A tényleges megvalósítás abban tér el a 6.3.1 részben ismertetett adat fogadási folyamattól, hogy a TcpIp_RxIndication() nem közvetlenül felette elhelyezkedő AUTOSAR réteg függvényét hívja, hanem a fogadott adatokat az LWIP-nek adja át az ethernet_input() LWIP függvényhíváson keresztül, ahol megtörténik a TCP/IP protokolljainak megfelelő feldolgozás. Ezután az LWIP fogja meghívni az AUTOSAR felsőbb rétegének RxIndication() függvényét az LWIP driver Tcp_recv(), vagy Udp_recv() függvényhívásán keresztül(5.2.1 fejezet)



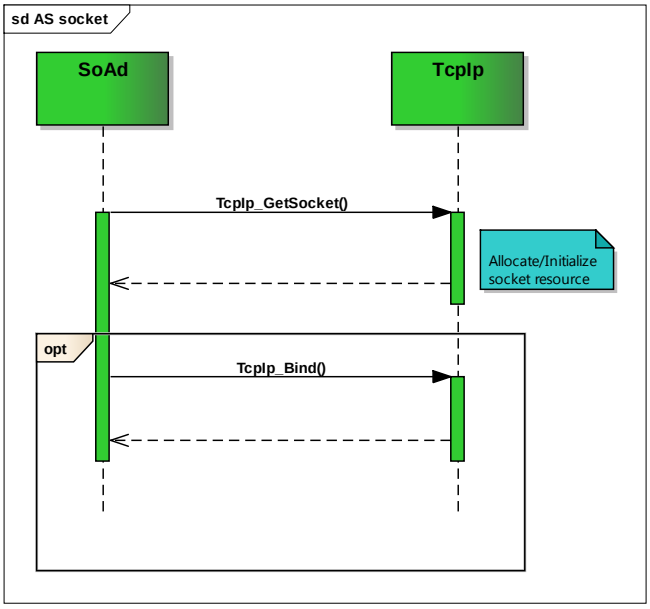
19. A megvalósított TCP/IP receive folyamat

Az AUTOSAR TCP/IP és a LWIP-vel megvalósított stack is adat fogadása esetén feltételezi, hogy van egy létező socket, ami azonosítja az alkalmazást, akinek a fogadott üzenet szól (6.2 fejezet). Ha nincs ilyen socket, vagyis az érkező üzenetben szereplő cél IP-cím és porthoz nem létezik socket, akkor a függvényhívási folyamat nem jut el SoAd_RxIndication() hívásig és az érkezett csomagot eldobja a TCP/IP stack.

6.3.3 AUTOSAR UDP Transmit

Egy IP alapú üzenet küldése úgy kezdődik, hogy a cél IP-címhez az ARP táblázatóból meghatározza ahhoz az IP-címhez tartozó MAC-címet. A táblázatot, vagy ARP cache-t előzőleg az ARP protokoll kezeli. Az UDP küldés, vagy transmit is így kezdődik. Ha nincs a cél IP-cím az ARP cache-be, akkor a TcpIp_UdpTransmit() hibával tér vissza(ARP_CACHE_MISS) és az üzenet nem lesz elküldve, viszont elindítja az ARP cím feloldási folyamatát. Ha a cím szerepel az ARP cache-be, akkor el tudja küldeni az üzenetet az Ethernet interfészen keresztül a 3.3.5 pont szerint. Itt az AUTOSAR TCP/IP stack feladata, hogy az TcpIp_UdpTransmit() paraméterében

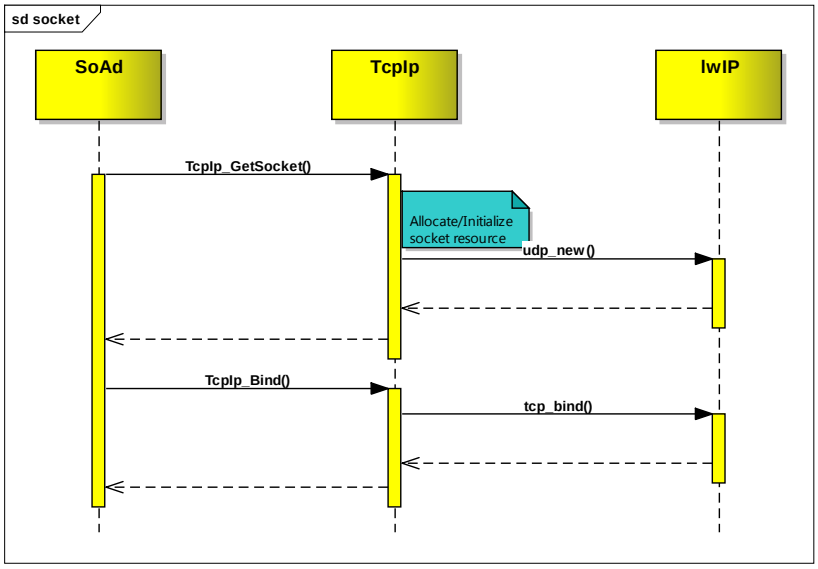
megadott adatot először egy UDP keretbe helyezi, majd az így kapott UDP csomagot IP keretbe teszi, és végül az Ethernet interfész fogja az egymásba ágyazott csomagokat Ethernet keretbe helyezni. A transmit folyamat szekvencia diagramja a függelékben látható. Bár a függelékben szereplő szekvencia diagram nem tartalmazza a socket létrehozását, de a TcpIp_UdpTransmit() hívása előtt létre kell hozni egyet (6.2 fejezet), amihez egy IP-címet és egy portot rendelünk hozzá a TcpIp_Bind() függvénnyel.



20. Az AUTOSAR Socket kérés

6.3.4 UDP Transmit folymat LWIP-vel:

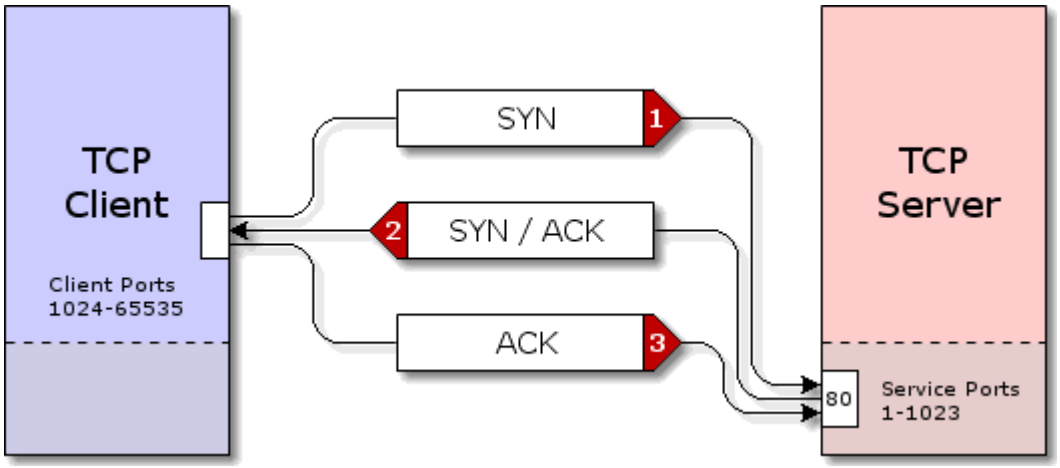
Mivel az LWIP Pbuf struktúrába tárolja és adja át belső protokoll rétegeinek a kapott adatot, így a TcpIp_UdpTrasmit()-ban először Pbuf-ba kell helyezni a paraméterben kapott adatot.(4.1 fejezet).Itt valójában nem történik tényleges adat másolás, mivel a Pbuf struktúra csupán a felhasznált adat kezdőcímét tárolja. Miután elkészült a Pbuf, ezt már el tudjuk küldeni az udp_sendto() LWIP függvényhívással. Az LWIP az LWIP driver low_level_output() függvényhívásán keresztül küldi el az UDP csomagot(5 fejezet). A szekvencia diagram F2 függelékben van. A socketet ebben az esetben is létre kell hozni, itt azonban a TcpIp_GetSocket() meghívja az LWIP udp_new() függvényét, ami egy UDP típusú protcol control blokkal tér vissza(6.2 fejezet).



21. A megvalósított TCP/IP socket kérés

6.3.5 AUTOSAR TCP Csatlakozási folyamat, mint szerver

A TCP az UDP-vel ellentétben kapcsolat centrikus. Még a tényleges adatküldés előtt a felek kiépítenek egy kapcsolatot és azt ápolják is[10]. A kapcsolat kiépítését a kliens kezdeményezi, amihez a szerver oldalon lennie kell egy létező socketnek, ami vár a csatlakozási kérelemre. Tehát a kapcsolat kiépítéséhez először kérni kell egy socketet a TCP/IP stack-től. A bind() után pedig a socketet un. Listen állapotba tesszük, amiben a socket várakozik a csatlakozási kérelemre. A csatlakozási kérelem az első lépése a stabil TCP kapcsolat felépítéséhez. A teljes csatlakozási folyamatot a 22. ábra mutatja.



22. A három utas kézfogás[11]

Egy TCP csomag továbbiakban TCP szegmens a fejlécének flags mezőjében jelzi, hogy milyen típusú TCP szegmensről van szó[12]. A kapcsolódási kérelmet a kliens adja ki egy olyan TCP szegmens küldésével, aminek fejlécében a SYN flag

magasra van állítva. Majd a szerver válasz üzenetében a SYN flag mellett az ACK flag is magasra van állítva. Végül a kliens ezt a válaszüzenetet nyugtázza egy ACK típusú üzenettel. Ekkor befejeződik a 3 utas kézfogás és a kapcsolat stabil állapotba kerül. Ekkor a TCP/IP stack értesíti a magasabb szoftver réteget egy SoAd_TcpAccepted() függvényhívással. Szekvencia diagram az F3 függelékben van.

6.3.6 TCP Csatlakozási folyamat, mint szerver LWIP-vel:

A tényleges megvalósításban a socket létrehozása és bind() után a 3 utas kézfogást az LWIP fogja megtenni. Miután befejeződött a csatlakozási folyamat meghívja az LWIP driver TcpAccepted() függvényét(5.2 fejezet,F4 függelék).

6.3.7 AUTOSAR TCP kapcsolódás, mint kliens

Az első lépésként itt is socketet kérünk, majd TcpIpBind() függvényhívással IP-címet és port-számot rendeljük a sockethez. Viszont ebben az esetben nem Listen állapotba tesszük a socketet, hanem el kell indítani a csatlakozási folyamatot a TcpIp_Connect() függvényhívással. Itt nem azonnal fogja elküldeni a TCP SYN szegmenset, hanem majd a modul main függvénye fogja végig vinni a 3 utas kézfogást. Itt csupán regisztráljuk a csatlakozási kérelmet. A csatlakozási folyamat befejezése után a SoAd_TcpConnected()-el értesíti a felsőbb AUTOSAR modult. (F5 függelék)

6.3.8 TCP Csatlakozási folyamat, mint kliens LWIP-vel

A TcpIp_Connect() hívása után, a main függvényben a tcp_connect() LWIP függvény hívásával kezdődik a TCP kapcsolat kiépítése. Az LWIP fogja megcsinálni a 3 utas kézfogást és a kapcsolódási folyamat befejezése után, meghívja Tcp_connected() LWIP driver függvényt (5.2 fejezet F6 függelék).

6.3.9 AUTOSAR TCP küldési folyamat

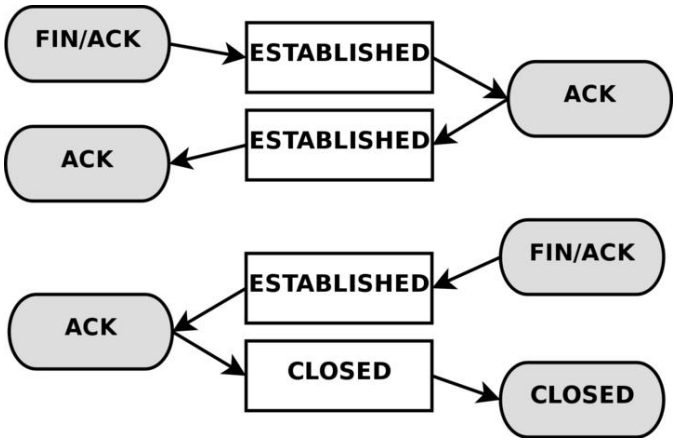
TCP küldés csak akkor lehetséges, ha a csatlakozási folyamat befejeződött és létrejött a stabil kapcsolat. A TcpIp_Transmit() ebben az esetben se azonnal hajtódik végre, hanem a kérés elmentésre kerül, majd a main függvénybe fog megtörténni a tényleges TCP szegmens küldés. A TCP szegmens elküldése után és stabil kapcsolat esetén, a fogadó fél nyugtát küld, hogy megérkezett az elküldött TCP szegmens (F7 függelék).

6.3.10 TCP küldési folyamat LWIP-vel

A TcpIp_Transmit() hívása után, a main függvényben az LWIP tcp_output() függvény lesz meghívva. Az LWIP készíti el a paraméterben megadott adatokból a TCP szegmenst. Végül az LWIP driver low_level_otput() függvényen keresztül küldi el az üzenetét(5.2 fejezet). A nyugta érkezésének jelzésére viszont az LWIP nem kínál callback függvényt, így annak figyelése a TcpIp_RxIndication() függvényben történik. Ha nyugta érkezett egy elküldött TCP szegmensről, akkor a SoAd_TxConfirmation()-nal értesíti a felette lévő AUTOSAR réteget(F8)

6.3.11 Socket lezárása

A socketet a TcpIp_Close() függvénnyel tudjuk lezárni. Ha UDP socket volt, akkor a függvényhívás azonnal felszabadítja a socketet. Ezt az LWIP udp_remove() függvényhíváson keresztül teszi meg. Ha a socket TCP kapcsolathoz lett létrehozva, akkor választhatunk, hogy a TCP kapcsolatot azonnal lezárja egy reset TCP szegmens küldésével, vagy kivárjuk a TCP lezárási folyamatát. A reset szegmens egy olyan TCP üzenet, aminek fejlécébe a flgas mező RST bitje magasra van állítva, ez jelzi a fogadó félnek, hogy itt azonnal befejeződik a kapcsolat. Ez az LWIP tcp_abort() függvényén keresztül történik. A TCP kapcsolt normális lezárását a következő ábra mutatja.



23. A TCP kapcsolat lezárása

A lezárást kezdeményező fél küld egy TCP szegmenst, amiben a FIN és ACK flagek magasra vannak állítva. Majd a fogadó fél nyugtát küld róla. Utána a fogadó fél is kezdeményezi a kapcsolat lezárását. Ezt a kétirányú kapcsolatot két irányból kell lezárni is[10]. Majd erre az üzenetre kapott nyugta után záródik le a kapcsolat. Ezt a folyamatot a tcp_close() LWIP függvénnyel indítja TcpIp_Close().

7 Az LWIP-vel elkészített AUTOSAR TCP/IP stack vizsgálata

Az elkészült TCP/IP stack vizsgálatánál alapvetően egyfajta teljesítménymérés volt a cél. Nem valódi átviteli sebességet akartunk meghatározni, csupán annyi volt a cél, hogy egy olyan maximális időt határozzunk meg ami két csomag érkezése között eltelik és csomagvesztés nem történik. Ez a fajta mérés csak UDP csomagok esetén működhet, mivel TCP esetén újraküldi a csomagot, ami elveszett, amit abból határoz meg a küldő fél, hogy nem kap nyugtát (ACK TCP szegmenst) a fogadó féltől. A fent vázolt mérés az a TCP/IP stacket vételi azaz RX irányba vizsgálja. Adási azaz TX irányú vizsgálatnál egy ciklusba a TcpIp_UdpTransmit() függvényhívással folyamatosan csomagokat küldtem késleltetés nélkül és mértem a két csomag között eltelt időt a fogadó oldalon. A csomagok a fejléceken kívül csupán egy sorszámot tartalmaztak. Az IP csomagtördelés nem volt engedélyezve. RX irányba a feldolgozás sebessége nem függ a csomag mérettől, mivel nem történik adatmásolás a TCP/IP stacken belül, csupán az Ethernet driver által meghatározott RX buffer memória címtől kezdi a beérkezett csomag feldolgozását. Végül a TCP/IP stack is egy pointert fog tovább adni a felsőbb szoftverrétegnek, ami már a hasznos adat címét tartalmazza. TX irányba viszont a protokoll fejlécekkel ellátott csomagot be kell másolni az Ethernet bufferbe, nagyobb csomagméret esetén ez már befolyásolja a küldési sebességet. A vizsgálatot elvégeztem olyan konfigurációba, ahol a megszakítás le volt tiltva és a csomag érkezését „lekérdezéses” (polling) módszerrel detektálta, és olyan beállítások mellett, ahol a megszakítások engedélyezve voltak az Ethernet kontroller felől.

7.1 Felhasznált hardver és szoftver eszközök

A hardware, amin teszteltem a TCP/IP stacket az a Freescale Power Architektúrájú processzora (Freescale: MPC5744P). A kontroller tartalmazza az Ethernet kommunikációs interfészt, ami egy fizikai illesztőn keresztül csatlakozik a hálózatra. A fizikai illesztő típusa: Texas Instruments DP83848C (100Mb/s Full duplex beállítással). A fizikai illesztő és a kontroller között a kommunikáció az MII, vagyis a Medium Independent Interface-en keresztül történik. A teszteléshez két fejlesztőpanelt használtam:

Az első: (MPC5744PEVB257UG) A BGA tokozású processzor fogadására alkalmas foglalattal rendelkező fejlesztő panel, a második (MPC5746MMB mother board) ami az előző fejlesztőpanel csatlakoztatásához szükséges konnektorokkal rendelkezik, és ezen helyezkedik el az Ethernet fizikai illesztője.

A mikrokontroller felprogramozásához és hibakereséshez a Lauterbach trace32 nevű eszközét használtam.

A szoftverek elkészítéséhez az Eclipse integrált fejlesztő környezet állt rendelkezésre. A fordító pedig a Green Hills Software beágyazott rendszerkehez készült C fordítója volt.

A PC és a kontroller közötti kommunikációt Wireshark nevű programmal figyeltem. Egy szerű szöveges üzenetek küldésére soros portot, fogadására pedig a Putty programot használtam.

A csomagok küldéséhez egy saját Java-ban fejlesztett grafikus kezelőfelülettel rendelkező mini alkalmazást használtam. Képes volt beállítani, hogy egy mérés során, hány UDP csomagot küldjön és bizonyos mértékig beállítható rajta két csomag között eltelt idő.

A Wiresharktól kapott időinformációk (timestamp) feldolgozására ugyan csak egy saját Java kisalkalmazást használtam, ami az egymás után érkező csomagok sorozatából kinyerte a minimum/maximum időket és meghatározott egy átlagidőt, ami két csomag között eltelt átlagos idő. Végül az eredményeket egy txt fájlba tárolja.

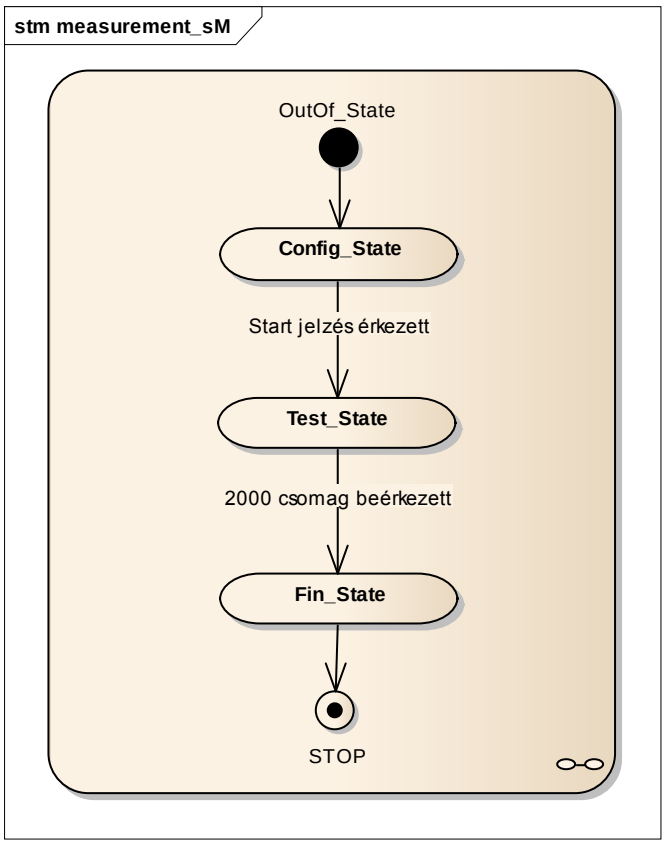
7.2 A mérés menete

Egy mérésnek azt tekintettem, amikor 2000 UDP csomagot küldtem a kontrollernek és utána a kontroller oldalon egy függvény kiértékelte, hogy mindegyik csomag megérkezett-e. Így a kontroller felől a soros porton minden mérés után egy OK vagy NOK (sikertelen) jelzést kaptam eredményül. A 2000 csomagból, ha csak egy is hiányzott, akkor az eredmény sikertelen. Két csomag között eltelt időt Wiresharkon keresztülnéztem. Egy mérés során a Wireshark regisztrálta mind a 2000 elküldött csomagot, a nézet pedig úgy volt beállítva, hogy az időbélyegek a csomagoknál az előtte érkezett csomag idejétől eltelt időt mutatta mikrosecundumban. Ezt a 2000 csomagból álló regisztrátumot dolgozza fel az előző pontban említett Java kisalkalmazás. Tehát egy mérés során rendelkezésre állt egy minimum, egy maximum

érték, a 2000 UDP csomagra vonatkozó átlag idő érték és a kontrollertől érkező értékelés, hogy veszített-e csomagot. A méréseket ismételten végrehajtottam úgy, hogy közbe csökkentettem a két csomag közötti időt egészen addig, amíg a kontroller felől NOK jelzést nem kaptam.

7.2.1 A csomagok kiértékelése a kontrolleren

A csomagok kiértékelését SoAd_RxIndication() függvényben valósítottam meg, mivel ez az a függvény, amit a TCP/IP stack minden csomag érkezése után meghív, ezzel átadva a felsőbb rétegnek a kapott adatot.



24. A csomagok kiértékelésének állapot diagramja

A kiértékelés folyamatát a 24. ábra mutatja. A mérés kezdetekor a csomagokat küldő program folyamatosan startjelzést küld. A startjelzés egy UDP csomag, amiben egy olyan 32 bites értéket küld, aminek minden bitje magas (0xFFFFFFFF). A kiértékelés addig van Config_State állapotban, amíg ez a startjelzés nem érkezik meg. A startjelzés megérkezése után visszaküldi ezt a start csomagot a küldő programnak és Test_State állapotba lép. A start jelzés visszaküldésével jelezi a küldő programnak, hogy megkapta a start jelzés és készen áll a sorszámozott csomagok fogadására. A

csomagokat küldő program ekkor kezdi el küldeni a 2000 db sorszámozott UDP csomagot. Test_State állapotban fogadja a 2000db UDP csomagot, és egy tömbben tárolja a sorszámokat, majd Fin_State állapotba lép. Ebben az állapotban történik annak megállapítása, hogy veszített-e csomagot a mérés során. Először sorba rendezi a sorszámokat a tömbben, utána ellenőrzi, hogy két egymáskövető elem különbsége egy lesz-e. Ha nem 1, akkor NOK választ ad a soros porton keresztül, egyébként OK választ.

7.3 A mérés eredménye

Egy nem sikerült mérés esetén feltételeztem azt, hogy a csomagvesztés akkor következett be, amikor két UDP csomag között a legrövidebb idő telt el. Tehát a mérés sorozat értékelésénél mindig a legrövidebb minimum időket vetem figyelembe. Ennek ellenére sem letett meghatározni egy éles határvonalat két csomag közötti eltelt időre nézve, így az eredményeket 3 kategóriára osztottam.

A mérési eredmények lekérdezéses koponfigurációval

TIME [ms]	POLL
-0,148 -tól	OK
0.148-0.080	OK/NOK
0.080-ig	NOK

A mérési eredmények megszakításos konfigurációval:

TIME [ms]	IRQ
-0,054 -tól	OK
0,054 -0,051	OK/NOK
0,051-ig	NOK

Az első kategória, ahol a csomagok folyamatosan megérkeztek hiba nélkül (OK). A második kategória, ahol a csomagok megérkezésének a sikeressége ingadozik néha megérkezett mind a 2000 néha csomagvesztés történt (OK/NOK). Végül az utolsó kategória, ahol minden elküldött 2000db-os sorozatba történt csomagvesztés (NOK). A teljes mérési sorozat az F9,F10,F11 függelékekben található.

8 Saját TCP/IP stack megvalósítása

Az előző pont eredményei alapján az LWIP-vel felépített TCP/IP stack teljesítménye sok feladathoz elegendő lenne, viszont felépítése nem illeszkedik az AUTOSAR szoftver architektúrába. Például egy kisebb szoftverréteg kellett az Ethernet Interfészhez való illesztésre, továbbá az LWIP-n belül az adat Pbuf struktúrában „utazik” és bár nem történik adat másolás, a Pbuf-ok adminisztrációját akkor is el kell végezni.(lefoglalás, felszabadítás). Az LWIP sokkal több szolgáltatást kínál, mint ami az AUTOSAR megkíván, ezért bonyolultabb a belső felépítése és viszonylag sok függvényhíváson keresztül valósítja meg a szolgáltatásait (nagy overhead). Ezért a feladatom további részében megpróbálom kiváltani az LWIP funkcióit saját fejlesztésű szoftver modulokkal.

Elsődleges cél volt, hogy egy modulárisabb az AUTOSAR szoftverarchitektúrába jobban illeszkedő almodulok valósítsák meg az AUTOSAR TCP/IP szolgáltatásait. Nem volt cél a teljes TCP/IP stack implementálása, de legalább az UDP csomagok küldését és fogadását meg kellet valósítani, továbbá a PC-n kiadott ping parancsra tudnia kellet válaszolni. A megvalósítás során figyelembe kellet venni a tovább fejlesztés lehetőségét.

8.1 Tervezési megfontolások

Mivel az AUTOSAR szabványban meghatározott TCP/IP szoftver modul túl komplex, hogy egy modulként legyen implementálva, így szükséges volt almodulokra bontani. Az AUTOSAR szoftvermodulok felbontásához először a konfigurációs adatmodellt érdemes átnézni, mivel ez azonnal ad egy támpontot arra nézve, hogy hogyan érdemes a modul követelményit megvalósítani. Az AUTOSAR szabvány a TCP/IP funkciókat nem írja, le csak hivatkozik a megfelelő RFC szabványra. Az RFC szabványok viszont jól körül határolják azokat a feladatokat, amiket érdemes egy almodulba helyezni. Ezek alapján nyilvánvaló volt, hogy miként bontsam fel almodluokra. A következő lépés , hogy megvizsgáljuk, milyen szoftvermodulok helyezkednek el a megvalósítani kíván modul „alatt” és „felett”, és milyen függvényhívásokon keresztül kapcsolódik hozzájuk. Ebben az esetben először a kétféle adatáramlási irány alapján néztem a függvényhívási láncolatot. Adat érkezésekor az

Ethernet interfész modul hívja az TcpIp_RxIndication() függvényt, ha TCP vagy UDP csomag érkezet, akkor végül a hasznos adatnak SoAd_RxIndication() függvényhíváson keresztül kell a felette levő réteghez kerülnie. Itt meg kellet vizsgálni, hogy a TcpIp_RxIndication() milyen paramétereket kap és ezekből végül, hogyan állítható elő a SoAd_RxIndication() paraméter listája. TX irányba viszont bármely TCP/IP-beli transmit függvény hívása, végül az Ethernet Interfész transmit függvényét fogja hívni. Az AUTOSAR kommunikációval kapcsolatos szoftvermodulok többsége adat fogadása esetén a modulok (a fent említett módon) az RxIndication() függvényein kapják meg az érkező adatokat, így az almoduloknál én is ezt az elvet követtem, így minden a közvetlen kommunikációval foglalkozó almodul rendelkezik RxIndication() függvénnyel. Kivétel azok a szoftvermodulok, amelyek a kommunikáció adminisztrálására szolgálnak (Pl.: Socket kezelés és IP-cím hozzárendelés). A következő amit meg kellet vizsgálni, hogy az almodulokba van-e olyan feladat, vagy követelmény ami igényeli, hogy a adat érkezéstől fogadástól függetlenül végre legyen hajtva rendszeresen. Például az ARP cache egy bejegyzését, ha nincs használva egy bizonyos idő után „elfelejti”, azaz itt egy számláló értékét folyamatosan csökkentettem, ha elérte a 0-t, akkor töröltem az ARP cache-ből a bejegyzést. A számláló csökkentése máris egy olyan feladat, amit ciklikusan kell végrehajtani és az AUTOSAR szoftvermodulok mintájára ez a modul main függvényébe került. Az AUTOSAR modulok tervezéséhez az is sokat segít, hogy a szabvány általában meghatározza a modul megvalósításához szükséges adattípusokat, adatszerkezeteket, de ebben az esetben ez csak részben igaz, mivel a követelmények megfogalmazásánál nem adja meg részletesen, hogy egy TCP/IP protokoll milyen típusokat vagy adatszerkezeteket használjon, csupán annyit mond, hogy valósítsa meg valamely RFC szabványt. Az RFC szabványok által definiált protokoll keretformátumok adták a megvalósított adat típusok egy részét így a protokollt megvalósító almodulok leírásánál ismertetve lesznek ezek az adatszerkezetek, amelyek mindegyike C struktúraként vannak felépítve. Mivel a cél legalább UDP csomagok küldése/fogadása volt sorba vettem, hogy milyen almodulok szükségesek hozzá, és ezek implementációjával kezdtem a feladatot. Az UDP csomagok IP keretekbe vannak ágyazva. Így szükség volt az IP almodulra és természetesen az UDP almodulra. Minden IP alapú kapcsolat esetén szükség van az IP-cím MAC-cím párosításra,ehhez, vagy minden esetben hálózaton keresztül lekérdezzük, hogy egy adott IP-címnek ki a gazdája és az milyen fizikai címmel rendelkezik (ez nem a legjobb megoldás: feleslegesen terhelni a hálózatot), vagy egyszer lekérdezzük és azt eltároljuk.

Erre szolgál az ARP protokoll. A kommunikációs folyamatok adminisztrációjához és a TCP/IP modul működéséhez pedig a socketeket kezelő almodulra és az IP-cím hozzárendelést végző almodulra volt még szükség.

8.2 ARP almodul

ARP protokoll 1.1.1 fejezetben leírtak szerint működik. A külön függvény gondoskodik az ARP REQUEST és ARP REPLY üzenetek küldésére, aminek feladata az ARP keretformátum szerinti üzenete összeállítása és elküldése az EthIf_Transmit() függvényen keresztül. A két függvényt az ArpMsg_Send() függvényhíváson keresztül hívható meg, aminek paraméterben adjuk meg, hogy ez most egy ARP REQUEST vagy ARP REPLY üzenet.

Egy ARP cache bejegyzés a következő információkat tartalmazza:

- A cél IP-cím
- A cél IP-címhez tartozó MAC-cím
- A saját IP-cím
- Meddig él a bejegyzés (ttl: time to live)
- Hányszor próbálkozott egy bejegyzéshez meghatározni a cél MAC-címet (retry_counter)
- A bejegyzés állapota (state)
- A használt Ethernet kontroller index

A fentiek egy C struktúrába vannak összefogva, amiből egy tömböt képezve alakul ki az ARP cache.

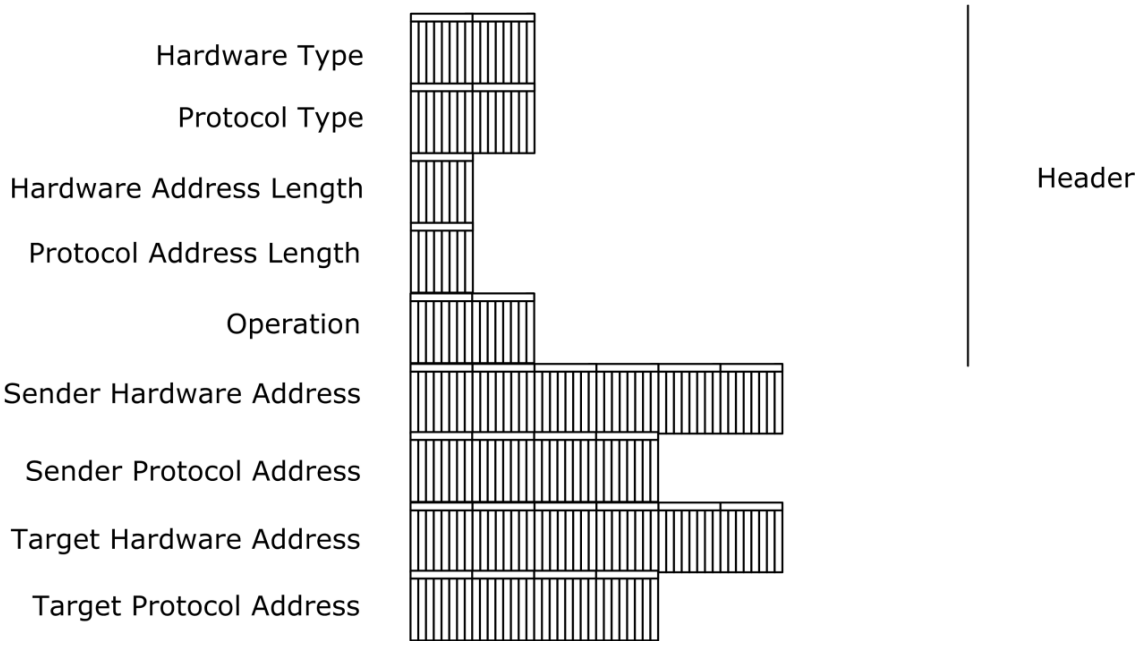
8.2.1 ARP AUTOSAR konfiguráció

Az AUTOSAR a következő paramétereket határozza meg az ARP almodul konfigurálására:

- Az ARP táblázat mérete: maximum hány bejegyzés lehet a táblázatba (TCPIP_ARP_TABLE_SIZE_MAX), mivel nem használunk dinamikus memóriakezelést, ezt a területet előre le kell foglalni az ARP cache számára, így ez a paraméter *Pre-compile time* konfigurációs osztályba van besorolva.
- TimeOut vagy ttl paraméter: meghatározza, hogy egy nem használt ARP bejegyzés meddig maradjon a cache-be.

- Gratuitous ARP üzenetek száma: Gratuitous ARP üzenet egy olyan ARP REQUEST, ami a saját IP-címét teszi a cél IP-cím helyére, vagyis a saját IP-címét kérdezi le, hogy van-e olyan node a hálózaton, akinek ugyan az IP-címe. Ezt akkor kell megtenni, amikor elkezdünk egy IP-címet használni, ezzel elkerülhetjük az IP-címütközést. Helyesen választott IP-cím esetén Gratuitous ARP üzenetre nem érkezhet válasz, ha mégis akkor a választott IP-cím foglalt.

8.2.2 ARP keretformátum



25. Az ARP keretformátum[10]

- HARDWARE TYPE: Két bájt, arra vonatkozik, hogy a Network Interface rétegben milyen hálózati technikát használunk. Az Ethernet kódja az egyes (0x0001).
- PROTOCOL TYPE: Két bájt, arra utal, hogy az ARP milyen protokoll számára biztosítja a névfeloldást. Elméletileg ugyanazok lehetnek az értékei, mint az Ethernet keret Type mezőnek - gyakorlatilag, ha nem 0x0800 (IP), akkor a node eldobja a csomagot.
- HARDWARE ADDRESS LENGTH: A hardware address jelen esetben a 6 bájtos MAC-cím.
- PROTOCOL ADDRESS LENGTH: Ez pedig az IP-cím hossza. Értelemszerűen az értéke fixen 4.
- OPERATION: Két bájt, az ARP keret típusát adja meg. Az ARP REQUEST típusa 0x0001, az ARP REPLY típusa pedig 0x0002

- SENDER HARDWARE ADDRESS, SHA: A feladó MAC-címe.
- SENDER PROTOCOL ADDRESS, SPA: A feladó IP-címe.
- TARGET HARDWARE ADDRESS, THA: A címzett MAC-címe.
- TARGET PROTOCOL ADDRESS, TPA: A címzett IP-címe. [10]

8.2.3 ARP modul működése

A bejegyzések a következő állapottokkal rendelkeznek:

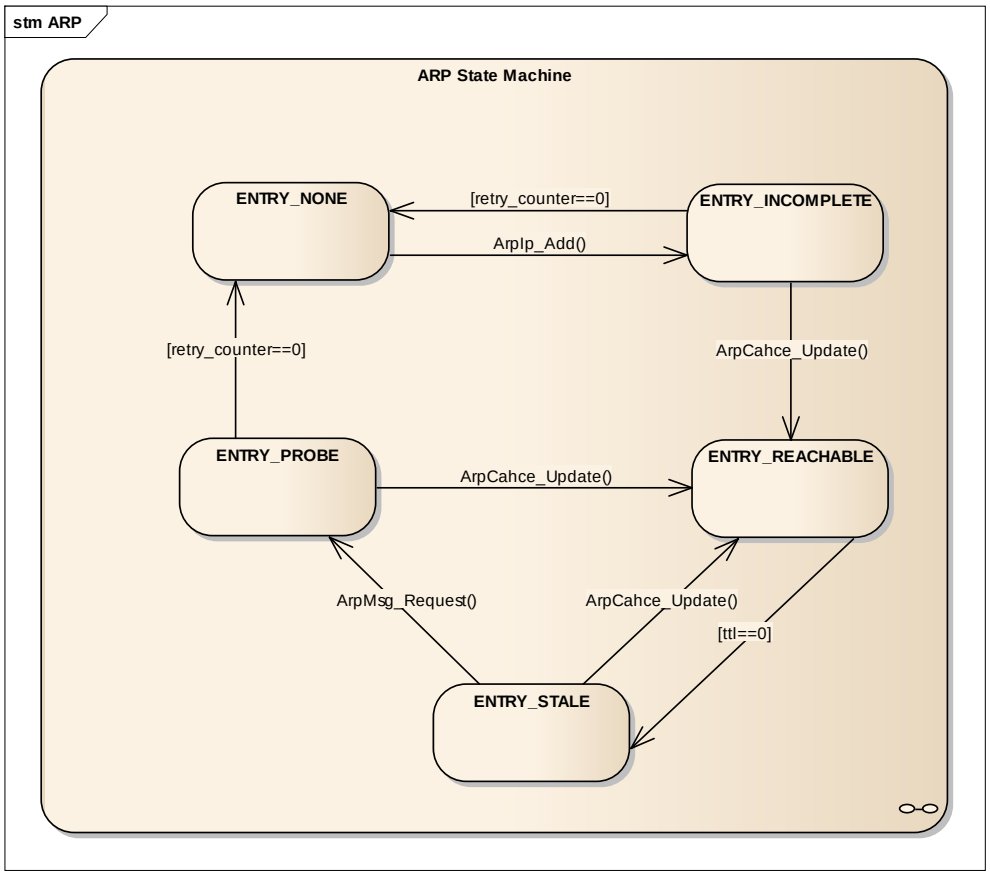
ENTRY_INCOMPLETE: ez a bejegyzés kezdeti állapota. Az ArpIp_Add() függvényhívás hozzáad egy új bejegyzést az ARP táblázathoz. Az ArpIp_Add() függvény 3 esetben van meghívva:

- Amikor olyan ARP REQUEST érkezik, aminek az IP-címe még nincs benne az ARP cache-be
- Amikor IP alapú üzenetet akarunk küldeni és a cél IP-cím nincs benne az ARP cache-be
- Amikor IP alapú üzenet érkezett nekünk címezve és a küldő fél IP-címe nincs benne az ARP cache-be. Ha a küldő fél ARP protokollon keresztül kapta meg az Ethernet vezérlőnk címét, akkor ez az eset nem fordulhat elő.

Ebben az állapotban a bejegyzés nem használható. Itt a bejegyzés alapján az almodul main függvényéből meghívott állapotgép fogja minden egyes bejegyzés aktuális állapotához tartozó feladatokat elvégezni. Ebben az esetben egy ARP REQUEST kérést fog küldeni. Ha válasz érkezik, meghívja az ArpCache_Update() függvényt, aminek hatására ENTRY_REACHABLE állapotba kerül. Ha az ARP REQUEST üzenetre nem érkezik válasz, akkor újra próbálkozik, amíg a retry_counter értéke el nem éri a 0-t. Minden újrapróbálkozás esetén csökkenti eggyel az értékét. A kezdeti értéket ARP_MAXRETRY paraméter adja meg, ami nem része az AUTOSAR szabvány által meghatározott konfigurációs paramétereknek. Ha retry_counter elérte a 0-t akkor ENTRY_NONE állapotba lép.

ENTRY_REACHABLE állapotba a bejegyzés elérhető és használható. Itt tartalmaz minden olyan információt, ami szükséges egy IP alapú üzenet elküldéséhez. Ebbe az állapotba csak ArpCache_Update() függvény hívásával kerülhet. Ez a függvény két esetbe lesz meghívva. Az első, amikor ARP REQUEST üzenetre ARP REPLY

válasz üzenet érkezik a másik, amikor olyan IP alapú üzenet érkezik, aminek a feladó IP-címe szerepel már az ARP cache-be. ENTRY_REACHABLE állapotba csökkenti a ttl értékét és addig marad az állapotban, amíg a ttl értéke el nem éri a 0-t. Kezdeti értéket a ttl a konfigurációs paraméter alapján kapja, továbbá ebben az állapotban kezdeti értékre állítja a retry_counter értékét.



26. Az ARP cache bejegyzés állapotgépes modellje

ENTRY_STALE állapot egy átmenetei állapot, akkor kerül ide a bejegyzés, amikor az ENTRY_REACHABLE állapotban a ttl értéke elérte a 0-t. Ekkor küld egy ARP REQUEST üzenetet és csökkenti a retry_counter értékét eggyel, majd ENTRY_PROBE állapotba lép. Ha közbe ArpCache_Update() hívás történik, akkor azonnal visszalép az elérhető állapotba.

ENTRY_PROBE állapotban folyamatosan ARP REQUEST üzeneteket küld és csökkenti a retry_counter érétkét. Ha érkezik rá válasz, akkor meghívja az ArpCache_Update() függvényt és elérhető állapotba kerül, ha nem érkezik válasz, akkor és a retry_counter elérte a 0-át akkor a bejegyzés ENTRY_NONE állapotú lesz.

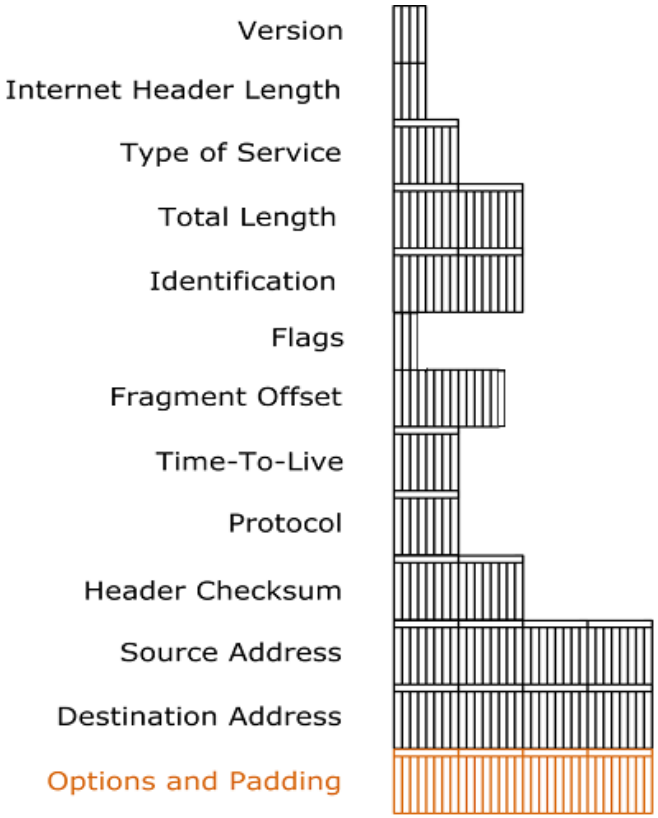
ENTRY_NONE egy átmeneti állapot, itt csupán a bejegyzés törlése történik meg.

Az IP alapú csomag érkezésénél az ArpIp_RxHander() függvény gondoskodik() az ARP cache kezeléséről, ARP csomag érkezése esetén pedig Arp_RxIndicatoin() függvény dolgozza fel az érkezett csomagokat. Ha IP üzenetet akarunk küldeni, akkor ArpGet_RmtPhyAddr() függvényhívással kérdezhetjük le az ARP táblázatot.

8.3 IP almodul

Az IP almodul feladata érkező üzenetek esetén ellenőrzi, hogy az üzenet valóban neki van címezve, ha igen akkor az IP keretbe milyen protokollt használó csomag van. Ez alapján fogja továbbítani a következő protokoll rétegnek. Üzenet küldése esetén pedig felépíti az IP fejlécet és bemásolja az így kapott csomagot az Ethernet TX bufferba. Az almodul nem rendelkezik állapotokkal és nincs olyan feladata, amit ciklikusan végre kell hajtania, így main függvénnyel se rendelkezik. Az IP protokoll egyik fő feladata az útvonalválasztás, amihez úgynevezett routing táblázatot használ. Ez az implementáció egyelőre nem valósítja meg az IP protokoll e funkcióját, így nem is rendelkezik routing táblázattal. Ez majd a további fejlesztések során fog bekerülni az IP almodulba. Az AUTOSAR IP almodulra vonatkozó konfiguráció csak a tördeléssel, vagy fragmentációval kapcsolatos paramétereket tartalmazza, viszont ez eddig még nem lett implementálva így a konfigurációs paramétereket ebben a fejezetben nem ismertetem. A rendelkezésre álló Ethernet kontroller képes az IP csomagok fragmentációjának kezelésére, de ennek a lehetőségnek a kihasználása azt eredményezné, hogy a megvalósított TCP/IP stack nem lenne hardver független.

8.3.1 IP keretformátum



27. Az IP fejléc[10]

VERSION: Az IP verziószáma. Jelenleg csak a 4-es, illetve a 6-os érték értelmezett.

INTERNET HEADER LENGTH: Mint az ábrán is látható, ez egy négybites mező. Itt tárolódik, hogy mekkora is az IP fejléc.

TYPE OF SERVICE (TOS): Eredeti értelmezésben nem nagyon használják, ebben a megvalósításban egyáltalán nincs felhasználva

TOTAL LENGTH: A teljes IP datagram méretét jelzi, bájtban. Mivel a mező kétbájtos, így az IP datagram maximális mérete 65535 byte lesz. Ha ennél nagyobb a csomag mérete, akkor tördelni kell.

IDENTIFICATION: Ha tördelni kell, akkor valahogy jelezni kell, mely töredékek tartoznak ugyanahhoz az IP datagramhoz. Ezért ebben a mezőben azonosító számot kapnak a csomagokat.

FLAGS: Tördeléshez kapcsolódó jelző bitek:

- Az első bit használaton kívüli.

- A második bit azt jelzi, hogy tördelhető-e a csomag? Ha az értéke egy, akkor nem.
- A harmadik bit azt mutatja, hogy a mostani IP csomag az utolsó-e, vagy jön még töredék. Ha az értéke 0, akkor ez az utolsó.

FRAGMENT OFFSET: Tördelés esetén ez az érték mutatja meg, hogy a konkrét csomag hányadik töredék éppen.

TIME-TO-LIVE: A mezőben lévő érték azt mutatja, hogy meddig él a csomag. Ha lejárt, akkor az a hálózati eszköz, amelyiknél az eset bekövetkezett, eldobja a csomagot. Valamikor idő dimenziójú volt, de az RFC 1812 óta inkább számláló, melynek értéke minden hopnál - azaz hálózati eszközön történő áthaladásnál - csökken egyet. Logikusan, ha küldünk egy csomagot, amelynek a TTL értéke 0, akkor az nem fog kimenni a subnetről. Ha a TTL értéke 1, akkor a subnetről kimegy ugyan, de maximum csak a szomszéd alhálózatra.

PROTOCOL: Azt mondja meg, hogy az IP Payload-ban milyen protokollhoz tartozó csomag utazik

HEADER CHECKSUM: Tulajdonképpen egy CRC, de csak a fejléc integritását biztosítja.

SOURCE ADDRESS: A feladó IP-címe.

DESTINATION ADDRESS: A célállomás IP-címe.

OPTIONS AND PADDING: Opcionális és ebben az implementációban nem használt mezők[10]

A VERSION és INTERNET HEADER LENGTH mezők 4 bitesek, de ezek együtt vannak tárolva egy 8 bites elemi adattípusban (uint8). INTERNET HEADER LENGTH-ben tárolt 4 bites érték 0-15-ig terjedhet, viszont egy IP fejléc hossza 20-60 bájt lehet, így az INTERNET HEADER LENGTH-ben tárolt értéket meg kell szorozni 4-el, akkor kapjuk meg a fejléc tényleges hosszát bájtban. A FLAGS és FRAGMENT OFFSET mezők együtt egy 16bites előjelnélküli adattípusba lettek eltárolva (uint16).

8.3.2 IP modul működése

Az IP almodul függvényei csak adat érkezésekor és adat küldésekor vannak meghívva. Adat érkezésekor csak az elfogadási kritériumok teljesülése esetén hívódik meg az Ip_RxIndication(). Ezek a kritériumok a következők:

Létezni kell egy socketnek amire előzőleg meg volt hívva a TcpIp_Bind() függvény. A bind() függvénnyel tulajdonképpen meghatározzuk, hogy a sockethez melyik Lokális IP-címet használjuk. Az IP-cím egyértelműen azonosítja az Ethernet kontrollert, viszont egy Ethernet vezérlőnek több IP-címe is lehet. Ezek alapján az az eset is elfogadott, ha a sockehez nem egy konkrét IP-címet rendelünk, hanem azt mondjuk, hogy egy meghatározott Ethernet vezérlőhöz tartozó összes IP-cím hozzá van rendelve a sockethez. Vagyis, ha egy Ethernet vezérlő felől érkező bármely üzenet elfogadott lesz, ha az IP fejlécben szereplő IP-cím egyezik bármely IP-címmel, ami ahhoz az Ethernet vezérlőhöz tartozik. Végül, ha a sockethez nem adjuk meg, hogy melyik kontroller IP-címei legyenek, hozzá rendelve akkor bármely kontroller felől érkező IP csomag, amelynek cél IP-címe egyezik a TCP/IP modul bármely létező IP-címével elfogadásra kerül. Az Ip_RxIndication() az IP fejléc protokoll mező alapján továbbítja a beérkezett üzenetet a megfelelő almodulnak.

Az üzenet küldés folyamata úgy kezdődik, hogy érkezik egy transmit függvényhívás, valamely IP alapú protokollt megvalósító almodulból. Ebben a transmit hívásban az adott protokollnak megfelelő fejléc összeállítása történik, majd az adat és az elkészült fejléc külön paraméterként lesz átadva az IpSend() függvények. Ezzel elkerülhető a memória tartalom másolás a köztes protokoll rétegekben, mivel az IpSend() még összeállítja a saját fejlécét, kér egy Ethernet TX buffert az Ethernet interfész modultól, és a tényleges adatmásolás csak itt fog történni a keretformátumoknak megfelelően. Először bemásolja az IP fejlécet majd UDP/TCP/ICMP fejléceket és végül a hasznos adatot. Az IP almodulhoz tartozó AUTOSAR konfiguráció nem határozza meg a fejlécben szereplő TIME-TO-LIVE érétket. Ez minden esetben az IP csomagban szereplő protokollt megvalósító konfigurációban van megadva.

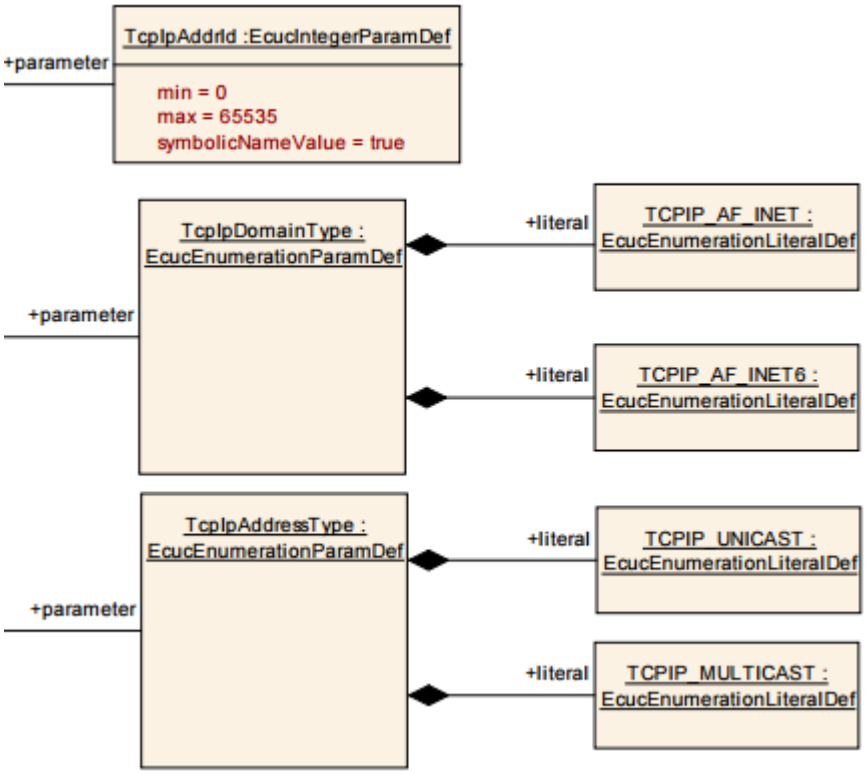
8.4 IPAssignment almodul

Ez az almodul felel azért, hogy a konfigurációban meghatározott IP-címet megkapja a TCP/IP modul. A TCP/IP modul. A TCP/IP modul IP-cím nélkül nem léphet ONLINE állapotba (17. ábra), azaz nem tud hálózati kommunikációt bonyolítani. Az AUTOSAR egy viszonylag nagy és bonyolult konfigurációs struktúrát határoz meg az IP-címek konfigurálására, ami figyelembe veszi, hogy IPv4 vagy IPv6 címről van-e szó és, hogy az IP-címet statikusan „kézzel” adtuk meg, vagy valamilyen IP-cím

kiosztási eljáráson keresztül kapja meg (DHCP/AUTOIP). A teljes konfigurációs struktúra leírásával a következő fejezet foglalkozik

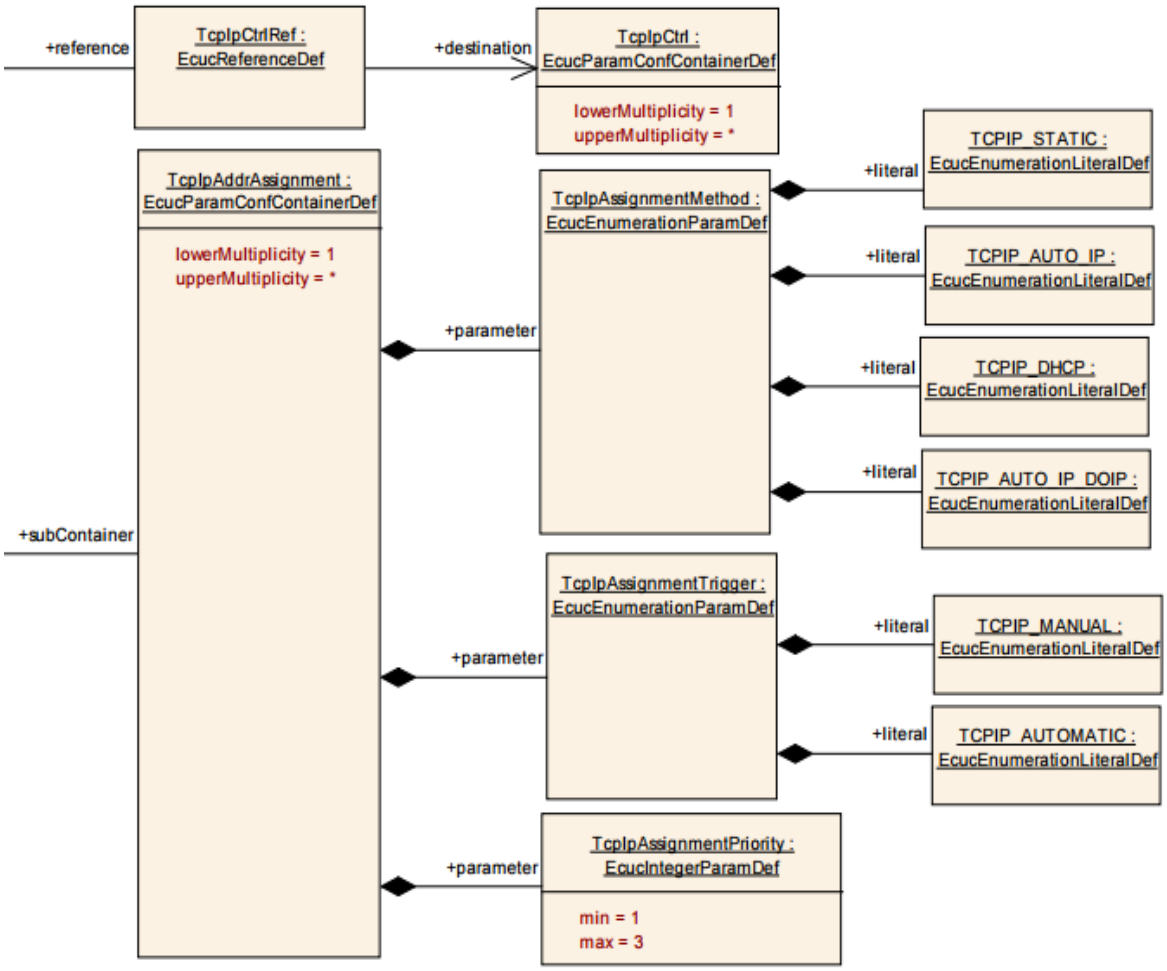
8.4.1 IPAssignment AUTOSAR konfiguráció

A következő 3 ábra az AUTOSAR IP hozzárendelésre vonatkozó konfigurációját mutatja.



28. Az IPAssignment konfiguráció 1[8]

- Minden használható IP-cím egyedi azonosítóval rendelkezik, mivel ezeket az IP-címeket egy tömbben tároltam, így ez az azonosító egyben a tömb indexet is jelenti.(TcpIpAddrId)
- A domain type paraméter azt adja meg, hogy IPv4 (TCPIP_AF_INET), vagy IPv6 (TCPIP_AF_INET6) címről van-e szó.
- Az Address type meghatározza, hogy a tárolt IP-cím unicast, vagy muticast. A multicast címzést az Ethernet drivernek is támogatnia kell, mivel a multicast IP-címekhez külön MAC-cím tartozik, amit az Ethernet vezérlőnek a saját egyedi MAC-címén kívül ismernie kell.



29. Az IPAssignment konfiguráció 2[8]

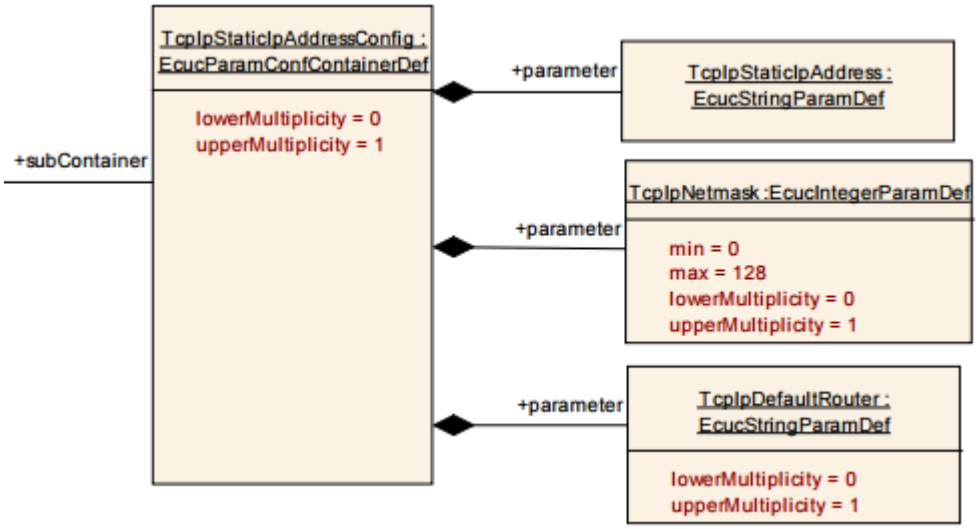
- Az IP-címek egy meghatározott Ethernet kontrollerhez lesznek, hozzárendelve, ennek az indexét adja meg a TcpIpCtrlRef.

A TcpIpAddrAssignment konfigurációs konténer 3 paramétert tartalmaz:

- AssignmentMethod: 4 féle eljárás lehetséges, amik szerint egy ATOSAR TCP/IP modul IP-címet kaphat:
  - Statikus úton „kézzel” adjuk meg (TCPIP_STATIC) az IP-címet, ezt a 30 ábrán látható konfigurációs konténerbe kell eltárolni.
  - DHCP-n keresztül kapja meg, ilyenkor a hálózaton léteznie kell egy DHCP szervernek, ami a DHCP protokollt használva adja meg a hálózati beállításokat.(TCPIP_DHCP).
  - TCPIP_AUTOIP esetén egy előre meghatározott címtartományból választ magának IP-címet

    - A TCPIP_AUTOIP_DOIP-nél az ISO 13400 szabvány szerinti IP-cím hozzárendelést kell megvalósítani. A szabvány a járművek diagnosztikai kommunikációjával foglalkozik IP alapú hálózaton.(DoIP: Diagnostic communication over Internet Protocol)

- AssignementTrigger: Manuális trigger esetén (TCPIP_MANUAL) az adott IP-cím csak egy TcpIp_RequestIpAddrAssignment() függvényhívás után válik használhatóvá. Automatikus trigger esetén nem kell meghívni a TcpIp_RequestIpAddrAssignment() függvényt, hanem amit lesz érvényes IP-cím az automatikusan, hozzárendelődik a konfigurációban meghatározott Ethernet kontrollerhez és használható lesz.
- AssignemntPriority: Egy TcpIpAddrId által azonosított IP-címet tartalmazó struktúra igazából több Assignment method konfigurációt tartalmaz, de ezek közül csak egy juthat érvényre. A prioritás azt mondja, meg hogy melyik legyen az. A prioritás értéke 1,2,3 lehet, ahol az egy a legerősebb. Ha egy erősebb prioritású IP beállítás érvényes IP-címet kap, azaz használhatóvá válik, akkor lecseréli az éppen érvényben lévő gyengébb prioritású IP-címet. Pl.: Ha hozzárendelünk egy 3-as prioritású IP-címet, akkor az azonnal érvényes beállítás lesz és TcpIpAddrId a statikusan beállított IP-címet fogja azonosítani. Ha mellette van egy magasabb prioritású AssignmentMethod DHCP beállítással és a DHCP szervertől időközbe kapott egy érvényes hálózati konfigurációt, akkor a TcpIpAddrId innentől kezdve a DHCP által adott IP-címet fogja azonosítani.
- A staitikus IP-címek tárolását a 30 ábra mutatja. Az IP-címet szöveges formába tárolja a TcpIpStaticIpAddress paraméterben. Az átalakításról még a modul init() függvényében gondoskodni kell, így a további felhasználásnál már szám formájában áll rendelkezésre.
- A netmask értéke a CIDR –nek megfelelő formában van egy 8 bites típusba tárolva. Ez a szám azt mondja meg, hogy IPv4 esetén a 32 bites netmask hány bitje magas értékű. A számolást a legmagasabb helyértékű bittel kezdjük pl.:/24 CIDR formájú netmask az 255.255.255.0 értéket jelenti, azaz a felső 24 bit az egyes. Ennek a konverziója is a modul init() függvényében történik.



30. Az IPAssignment konfiguráció 3[8]

A default router értéke a SataticIp paraméterhez hasonlóan string formában van megadva. Ha olyan IP csomag érkezik, ami nem ahhoz az alhálózathoz tartozik, akkor a defult router által megadott IP-címre továbbítja a kapott csomagot.

8.4.2 IPAssignement almodul működése

Az almodul az IP-címeket a következő struktúra tárolja:

```
typedef struct{
    uint16                          Id;
    TcpIpAddr_AssignmentType        TcpIpAddrAssignment;
    uint8                           CtrlIdx;
    uint32                          TcpIpLocalAddr;
    uint32                          TcpIpcDefaultRouter;
    uint32                          TcpIpNetmask;
    TcpIp_IpAddrStateType           State;
    boolean                         BoundToSocket;
    boolean                         RequestedFlag;
}IPv4AddrSettingType;
```

- Az Id: Egyedi azonosító egyben az őt tároló tömb indexe is.
- TcpIpAddrAssignment: A konfigurációs struktúrában található összes paramétert eltárolja.
- CtrlIdx: melyik Ethernet vezérlőhöz akarjuk majd hozzá rendelni
- TcpIpLocalAddr: Az IP-cím.
- TcpIpcDefaultRouter: Az alapértelmezett átjáró IP-címe.
- TcpIpNetmask Alhálózati maszk

- State: Az IP-cím állapota. Az AUTOSAR az IP-címek számára 3 állapotot határoz meg:
  - TCPIP_IPADDR_STATE_ASSIGNED: Az IP-cím hozzá van rendelve egy Ethernet vezérlőhöz, használható
  - TCPIP_IPADDR_STATE_ONHOLD: Egy már ASSIGNED állapotú IP-cím kerülhet ebbe az állapotba. Itt átmenetileg nem használható az IP-cím, de bármikor kérhető, hogy ASSIGNED állapotú legyen.
  - TCPIP_IPADDR_STATE_UNASSIGNED: Az IP-cím nem használható
- BoundToSocket: Megadja, hogy az adott IP-cím bind()-al hozzá lett-e rendelve egy sockethez
- RequestedFlag: Megadja, hogy az adott Id-ra volt-e meghívva TcpIp_RequestIpAddrAssignment() függvény.

Az almodul a fenti struktúrából felépített kettő tömböt/táblázatot használ. Az elsőbe azok a konfiguráció által meghatározott paraméterek kerülnek, amelyek prioritásuk miatt elsőbbséget élveznek az IP-cím hozzárendelési folyamatba. A második táblázatba kerül a konfiguráció által megadott összes többi IP-cím konfiguráció. A táblázatok feltöltését a modul init() függvénye végzi.

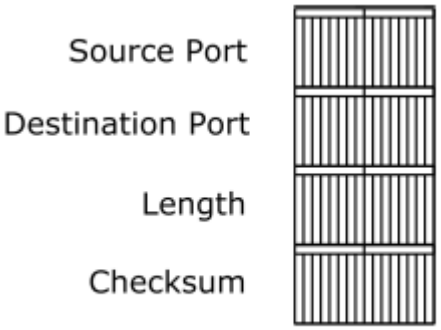
Az almodul main függvényében először csak az első táblázatban szereplő beállításokat vizsgálja és automatikus trigger esetén azonnal ASSIGNED állapotú lesz az IP-cím. Manuális trigger esetén, akkor lesz ASSIGNED állapotú, ha a requested flag magas. Itt indítaná el a különböző IP-cím kérési eljárásokat(DHCP, AutoIp, AutoIp DoIp almodulok egyenlőre nincsennek megvalósítva) amennyiben nem statikus IP-cím van konfigurálva. Ez a lépés minkét táblázatra érvényes. Ha az első táblázatba egy TcpIpAddrId-hoz olyan hálózati beállítás van, ami nem érvényes, de ugyan ahhoz az Id-hoz a második táblázatba van érvényes hálózati beállítás, akkor az átkerül az első táblázatba és a következő lefutási ciklusba a fent említett első táblázatra érvényes lépéseket végrehajtja rajta a main függvény. Ha az első táblázatba létezik érvényes hálózati beállítás, de a második táblázatban lévőnek nagyobb a prioritása, akkor is kicseréli a kettőt.

Az IP-cím, ha nincs sockethez rendelve, akkor az IpAssign_Release() híváson keresztül kerülhet UNASSIGNED állapotba.

8.5 UDP almodul

Az almodul feladata az UDP fejléc összeállítása és továbbítása az IP almodulnak az IpSend() függvényenkereszül, amikor UDP üzenetet akarunk küldeni. UDP csomag fogadásakor az IP almodul hívja meg az Udp_RxIndication() függvényt, ezen keresztül kapja mega fogadott csomagokat. Az UDP almodul konfigurációja egyetlen paramétert tartalmaz a Time to Live (ttl) paramétert. Ezt nem az UDP fejléc összeállításához fogja felhasználni, hanem az IP fejléc ttl paramétere fogja tartalmazni. Az almodul függvényei csak UDP csomag érkezésekor, vagy UDP üzenet küldésekor vannak, meghívva. Nem rendelkezik main függvénnyel.

8.5.1 UDP keretformátum



31. Az UDP fejléc

SOURCE PORT: Azt a portot azonosítja, amelyiken az alkalmazás elküldte a csomagot.

DESTINATION PORT: Azt a portot tartalmazza, amelyiken a fogadó oldali alkalmazás fogadja a csomagot.

LENGTH: Az UDP csomag mérete. Minimum 8 bájt (ekkor csak a header van), maximum maximum 65535 bájt lehet.

CHECKSUM: Ellenőrző összeg.

8.5.2 UDP modul működése

UDP csomag érkezésekor az Udp_RxIndication() függvényben történik a checksum ellenőrzése. Ha nem megfelelő, akkor eldobja a csomagot, ha 0, akkor az azt

jelenti, hogy az UDP csomag nem használ ellenőrző összeget. Ha checksum megfelelő volt, akkor a SoAd_RxIndication() függvényen keresztül adja tovább a kapott UDP csomagot a felsőbb rétegnek. UDP csomag küldésénél az almodul csak a fejléc összeállítását végzi és a checksum értékét számolja ki. A checksum számításához nem csak az UDP fejlécet és a küldeni/fogadni kívánt adatokat veszi figyelembe, hanem az IP fejléc küldő és fogadó IP-címet tartalmazó mezőit is és az IP prtokoll mezőjét. Az UDP fejlécet és az IP fejléc 3 mezőjét együttesen UDP pszeudo headernek nevezzük. Így UDP csomag összeállításához, vagy fogadásához is szükség van az IP fejlécre. Az almodulnak nincs ismétlődő feladata, így main függvénnyel sem rendelkezik.

8.6 Socket almodul

A socketet megvalósító struktúra abban tér el a 6.2 fejezetben ismertetett socket felépítéstől, hogy az LWIP protokoll kontrol blokkokat saját protokollokra való információs struktúrára cseréltem. Ebben az esetben csak az UDP-re vonatkozóan lett megvalósítva és csupán az UDP ttl értékét tartalmazza, ezen keresztül fogja megkapni a ttl értékét az IP fejléc UDP csomag küldésekor.

8.6.1 A Socket modul működése

Az almodul külön táblátazban tárolja a TCP és UDP socketeket, aminek a méreteit TCPIP_UDP_SOCKET_MAX, TCPIP_TCP_SOCKET_MAX pre-compile konfigurációs paraméterek tartalmazzák. A socketeknek 6 állapota van:

```
typedef enum{
    SOCKET_STATE_FREE
    SOCKET_STATE_ALLOCATED,
    SOCKET_STATE_BOUND,
    SOCKET_STATE_LISTEN,
    SOCKET_STATE_ACCEPTED,
    SOCKET_STATE_CONNECTED
}Socket_State_type;
```

Állapot váltás csak függvényhíváson keresztül lehetséges.

SOCKET_STATE_FREE: Kezdeti állapot. A socket szabad és használható.

SOCKET_STATE_ALLOCATED: TcpIp_SocketGet() függvényhívás hatására, ha van szabad socket, akkor egyet lefoglal. A TcpIp_SocketGet() paraméterbe kapja meg, hogy TCP , vagy UDP socketről van szó és szabad socket Id-t add vissza egy pointeren keresztül.

SOCKET_STATE_BOUND: Állapotba TcpIp_SocketBind() függvényhívás által kerülhet. Ellenőrzi, hogy a socket SOCKET_STATE_ALLOCATED állapotban van-e és feltölti a socket adattagjait

A maradék 3 állapot a TCP socketek működéséhez szükséges. A socket felszabadítását a TcpIp_SocketRelease() függvény végzi. Ebben a függvényben nem ellenőrzi, hogy a socket éppen milyen állapotban van és, hogy használva van-e. Ezt a függvényhívás helyén kell megtenni.

9 Almodulok Tesztelése

Az AUTOSAR modulok tesztelését általában a black-box tesztelési modell szerint végzik. A fekete doboz tesztelésnél a tesztelőnek nem kell ismerni a szoftverkomponens belső felépítését. Kizárólag a teszt során előre megtervezett bemenetekre adott válasz alapján döntik el, hogy a szoftverkomponens jól működik-e.

A TCP/IP stack almoduljainak tesztelésénél viszont a white-box megközelítést alkalmaztam. Ebben az esetben a tesztelő ismeri a modul belső felépítését és hozzáfér a modul belső változóihoz. Erre azért volt szükség, mivel a TCP/IP stack almoduljai közül valamelyik táblázatokat tartalmaz, valamelyik pedig belső állapotokkal rendelkezik, így ezek vizsgálatához közvetlen hozzáférésre volt szükség a belső állapotváltozókhoz és táblázatokhoz.

A tesztelést a CUnit-al végeztem. A CUnit tartalmazza a teszteléshez szükséges előre definiált függvényeket. A modult izolációs teszteléssel vizsgáltam. Modulok izolációs tesztelésénél a modulok egyenként, elszigetelten teszteltek[14]. A tesztelés során a következőket figyeltem:

- Amelyik modul rendelkezett állapotokkal ott külön teszteseteket definiáltam minden egyes állapot átmenethez és vizsgáltam, hogy megtörténik-e az állapot váltás.
- Minden esetben, amikor egy függvény az almodul belső táblázatát írja vagy olvassa, akkor ellenőriztem a táblázatba az eredményt.
- Ha egy függvény paraméterként pointert vár, le volt tesztelve null pointerrel is.
- Ha egy almodul egy másik modul függvényét hívta, akkor az a függvény egy függvénycsonkkal (stub) volt helyettesítve.

A függvény csonkok alkalmazásával vizsgálható volt, hogy megtörténik-e a függvényhívás és a függvény paraméterek globális változóba történő mentésével ellenőrizhető volt a paraméter átadás.

9.1 Teszt csonk alkalmazása

Például az ARP almodul tesztelésénél amikor ArpSend_Msg() függvényhívás történt, akkor végül az Ethrent Interfész EthIf_Transmit() függvényét hívja. Ekkor nem a valódi EthIf_Transmit() függvényt használja, hanem egy függvénycsonkot, ami annyit

csinál, hogy az EthIf_Transmit()-nek átadott paramétereket elmenti a paraméter listának megfelelő struktúrába.

A függvény tényleges deklarációja:

```
Std_ReturnType EthIf_Transmit( uint8 CtrlIdx,
                               uint8 BufIdx,
                               Eth_FrameType FrameType,
                               boolean TxConfirmation,
                               uint16 LenByte,
                               uint8*PhysAddrPtr);
```

A globális adat struktúra definíciója:

```
typedef struct{
        uint8 CtrlIdx;
        uint8 BufIdx;
        Eth_FrameType FrameType;
        boolean TxConfirmation;
        uint16 LenByte;
        uint8* PhysAddrPtr;
        BufReq_ReturnType return_value;
    }Transmit_type;
```

Látható, hogy a függvény összes paramétere szerepel a fenti struktúrában. Az összes Ethernet interfész függvényre, amit használ az ARP almodul, elkészült a hozzá tartozó struktúra és végül az összes struktúrát egy közös struktúrába helyeztem el.

```
typedef struct{
    GetPhysAddr_type     GetPhysAddr;
    ProvideTxBuffer_type ProvideTxBuffer;
    Transmit_type        Transmit;
}EthIfStubLogType;
```

Ebből az adattípusból lett létrehozva az a globális struktúra, amit az adott függvény hívása után vizsgálni lehet.

```
EthIfStubLogType EthIfStubLog;
```

Ezek alapján a függvény törzs:

```
    {
    EthIfStubLog.Transmit.CtrlIdx=CtrlIdx;
    EthIfStubLog.Transmit.BufIdx=BufIdx;
    EthIfStubLog.Transmit.FrameType=FrameType;
    EthIfStubLog.Transmit.TxConfirmation=TxConfirmation;
    EthIfStubLog.Transmit.LenByte=LenByte;
    EthIfStubLog.Transmit.PhysAddrPtr=PhysAddrPtr;
    DataLogPtr=Eth_buffer_stub
```

```
return EthIfStubLog.Transmit.return_value;
}
```

A fent leírtak szerint annyi történik, hogy a függvény paramétereit elmenti a megfelelő struktúra hozzá tartozó mezőibe.

9.2 Almodulok felhasználása, kipróbálása mikrovezérlőn

9.2.1 A megvalósított TCP/IP függvények

Az almodulok felhasználásával a következő AUTOSAR TCP/IP függvények készültek el:

```
void TcpIp_Init( const TcpIp_ConfigType* ConfigPtr )
```

A modul ezen a függvényen keresztül kapja meg a konfigurációt. A ConfigPtr a használni kívánt konfigurációs struktúra kezdőcímét tartalmazza. Ebben a függvényben meghívásra kerül az összes almodul init(…) függvénye.

```
Std_ReturnType TcpIp_GetSocket(TcpIp_DomainType Domain, TcpIp_ProtocolType
Protocol, TcpIp_SocketIdType* SocketIdPtr)
```

A socket almodul TcpIp_SocketGet(…) függvényét hívja. A SocketIdPtr-n keresztül adja vissza a Protocol által meghatározott (UDP/TCP) socket azonosítóját. A Domain paraméterben azt adjuk meg, hogy a socket IPv4 vagy IPv6 IP-címet tartalmaz-e. A modul egyelőre nem támogatja az IPv6 címeket. Ha nincs elég socket, akkor a függvény az Std_ReturnType által meghatározott E_NOT_OK értékkel tér vissza

```
Std_ReturnType TcpIp_Close(TcpIp_SocketIdType SocketId, boolean Abort)
```

A függvény lezárja SocketId által meghatározott socketet. A TcpIp_SocketGetbyId(…) függvényhívással megkeresi az adott azonosítójú socketet a Socket almodulban. Ha nem találja, akkor TcpIp_Close(…) E_NOT_OK értékkel tér vissza, egyébként UDP socket esetén lezárja a socketet. TCP socket esetén kétféle lezárási mód van. Ezt az Abort paraméterrel tudjuk beállítani.

```
Std_ReturnType          TcpIp_Bind(TcpIp_SocketIdType          SocketId,
TcpIp_LocalAddrIdType LocalAddrId, uint16* PortPtr)
```

Ez a függvény a Socket és az IPAssignemnt almodulokat is felhasználja. A TcpIp_SocketBind(…) függvénnyel fogja a SocketId által megaadott sockethez hozzárendelni a LocalAddrId által meghatározott IP-címet. Az IP-címeket az IP Id-ból az IpAssign_GetIpData(…) függvény határozza meg. A socketnek itt már allokálva kell

lennie. A PortPtr a port memória címére mutat, automatikus port kiválasztás esetén pedig ezen keresztül adja vissza a kiválasztott port címét.

```
Std_ReturnType TcpIp_RequestComMode( uint8 CtrlIdx, TcpIp_StateType State
)
```

A 6.1 fejezetben bemutatott állapotoknak megfelelő állapot váltásokat kérhetjük Ethernet kontrollerenként külön-külön. A kontrollert a CtrlIdx paraméterrel választjuk ki a kívánt állapotot pedig azt State paraméterben. A 6.1 fejezetben szereplő állapot gépes modell az nem változott, érvényes erre a TCP/IP megvalósításra is.

```
Std_ReturnType     TcpIp_RequestIpAddrAssignment(     TcpIp_LocalAddrIdType
LocalAddrId,    TcpIp_IpAddrAssignmentType    Type,    TcpIp_SockAddrType*
LocalIpAddrPtr )
```

A függvény a LocalAddrId által azonosított IP-címet hozzárendeli az Ethernet kontrollerhez. A kontroller indexet az IP almodul konfigurációja tartalmazza. Ez a függvényhívás csak manuálisra konfigurált IP-címekre használható. Az IPAssignment IpAssign_Request(…) függvényét használja. A type paraméterben az IP-cím kiosztás módját adjuk meg DHCP/AutoIp/AutoIp DoIp vagy Static. Ha olyan LocalAddrId-ra hívjuk meg, amihez nincsen konfigurálva statikus IP-cím, akkor LocalIpAddrPtr címen lévő IP-címet fogja használni.

```
Std_ReturnType          TcpIp_ReleaseIpAddrAssignment(TcpIp_LocalAddrIdType
LocalAddrId )
```

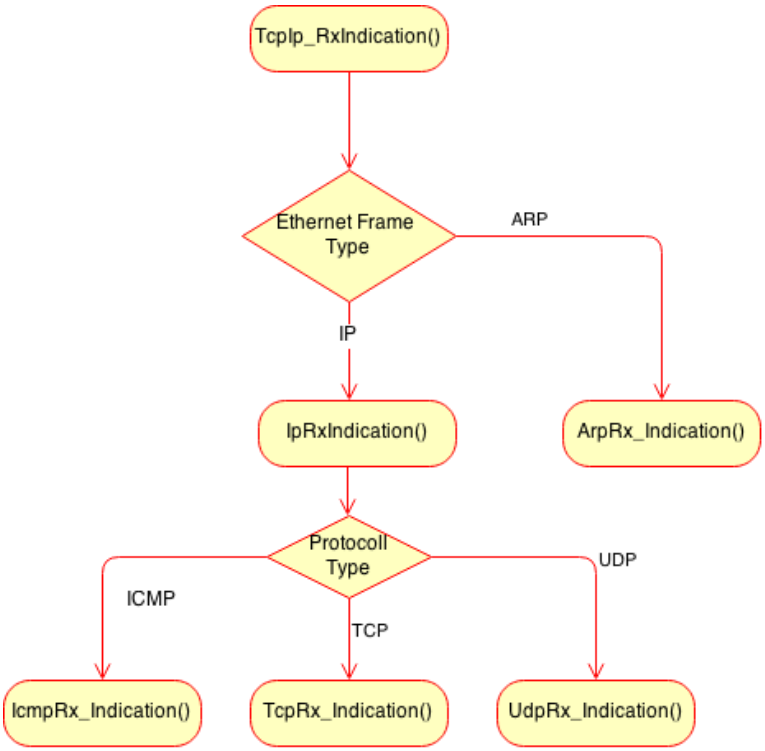
Felszabadítja a LocalAddrId által meghatározott IP-címet. Az IPAssignment IpAssign_Release(…) függvényét hívja.

```
TcpIp_ReturnType  TcpIp_UdpTransmit(TcpIp_SocketIdType  SocketId,  uint8*
DataPtr, TcpIp_SockAddrType* RemoteAddrPtr,uint16 TotalLength)
```

A függvény SocketId által meghatározott Ethernet interfészen keresztül elküld egy UDP csomagot. A TcpIp_SocketGetbyId(…) megkeresi az Id-hoz tartozó socketet. Ha nincs ilyen Id-val socket akkor TCPIP_E_NOT_OK hiba értékkel tér vissza a függvény. Ha a socket létezik, akkor F16 függelékben szereplő szekvencia diagram szerint folytatja az üzenet küldését. Ebben a függvényben az UDP fejléc összeállításához UdpCreate_Header(…) UDP almodul függvényét használja. Az UDP fejlécben viszont az ellenőrző összeghez szükség van az UDP pszeudo header-re, ami pedig az IP fejléc bizonyos mezőit használja, így itt az IP fejléc szükséges mezőit meg kell határozni.

```
TcpIp_RxIndication(uint8 CtrlIdx, Eth_FrameType FrameType, boolean
IsBroadcast, uint8* PhysAddrPtr, uint8* DataPtr, uint16 LenByte )
```

Az FrameType paraméter alapján továbbítja a megfelelő protokoll rétegnek a beérkező csomagot.



32. A beérkező üzenetek továbbítása

```
void TcpIp_MainFunction( void )
```

A modul main függvényébe az ARP almodul és az IPAssignement almodul main függvényei vannak meghívva, továbbá az AUTOSAR TCP/IP modul állapotgépét megvalósító függvény.

Az IcmpRx_Indication() függvénynek csak az a része lett elkészítve, ami válaszol az ICMP echo request típusú üzenetre, azaz képes válaszolni a ping-re.

9.2.2 Tesztelés mikrovezérlőn

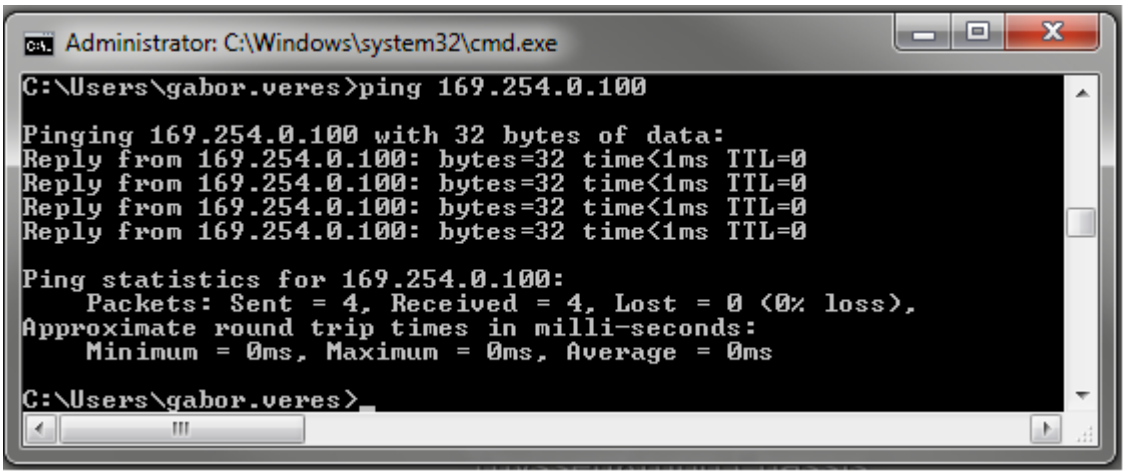
Az eddig elkészült TCP/IP modul kipróbálását a 7.1 fejezet szerinti összeállításon végeztem. A mikrovezérlőre a következő AUTOSAR szoftvermodulok lettek letöltve:

- Ethernet Transceiver Driver
- Ethernet Driver

- Ethernet Interface
- TcpIp

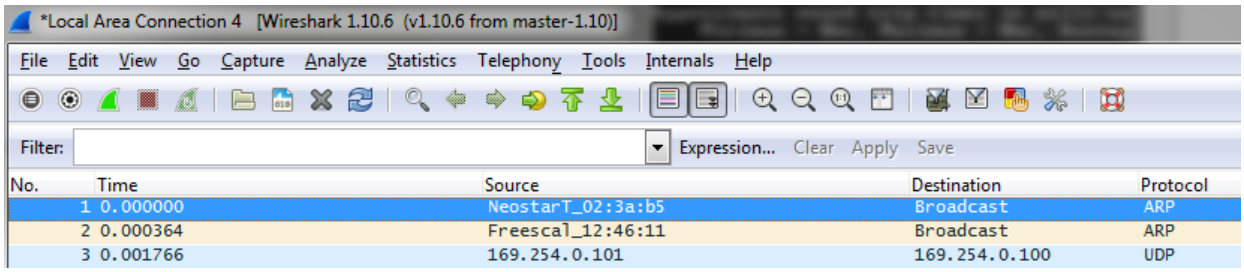
A kipróbáláshoz mindegyik szoftvermodulhoz el kellet készíteni egy konfigurációt. Azaz a szabványban meghatározott konfigurációs adatszerkezetet realizáló struktúra adattagjait fel kell tölteni értékekkel. A következő lépésben a létrehozott konfigurációkat érvényesíteni kell a modulok init() függvényein keresztül. Amelyik modul rendelkezik main függvénnyel azokat a main függvényeket egy végtelen ciklusba meghívjuk. Egy valódi alkalmazásban operációs rendszer mellett, mint taskok(folyamatok) lesznek alkalmazva ezek a main függvények, aminek ütemezésért az operációs rendszer a felelős.

A mikrovezérlőre letöltött szoftvermodulok működését a következő két ábrával szemléltetem. Az első a ping-re adott válaszát mutatja, a másodikon pedig egy UDP csomag fogadását láthatjuk.



33. Egy ping-re adott válasz

A 34. ábrán a Wireshark hálózat analizátorról készült kép van. A PC kezdeményezi az UDP csomag küldését, de először ARP request üzenetet küld, amire a mikrovezérlő válaszol egy ARP reply üzenettel (2. sor). Ezzel megtörténik az IP-cím MAC-cím párosítás és csak utána tudja küldeni az UDP csomagot. Ha a kontroller felől akarunk UDP csomagot küldeni, akkor az TcpIp_UdpTransmit() függvény először hibával tér vissza mivel a cél IP-cím nincs benne az ARP cache-be, de ennek hatására az ARP almodul elküldi az ARP request kérést és ha érkezik válasz akkor már a második TcpIp_UdpTransmit() sikeres.



34. Egy UDP csomag fogadása

Összefoglalás

A szakdolgozatom elkészítse során alaposan meg kellett ismernem a TCP/IP-t alkotó protokollokkal, továbbá két AUTOSAR szoftvermodullal az Ethernet Interface és a TCP/IP modulokkal. Mivel a TCP/IP széleskörűen elterjedt és a protokollok működését leíró RFC szabványok már több mint 30 évesek, ezért viszonylag sok leírás és magyarázat állt rendelkezésemre a nyers szabvány értelmezésén túl, viszont az AUTOSAR szoftvermodulok megvalósításához csak az AUTOSAR konzorcium által kiadott szabványok álltak rendelkezésre.

Dolgozatom első részében egy nagyon rövid áttekintés nyújtottam a TCP/IP protokolljairól, majd összefoglaltam az AUTOSAR szoftverarchitektúrát. Ez adta az alapot a további szoftver rétegek elhelyezésére ebben az architektúrában.

Az AUTOSAR tulajdonképpen megadja az Ethernet interfész helyét a szoftverarchitektúrában, de egy nem AUTOSAR szoftvermodul helyét nem definiálja. A dolgozatom következő pontja az volt, hogy egy nyílt forrású TCP/IP megalótást illesszek az AUTOSAR szoftver architektúrába, így bemutattam, hogy hova és hogyan illesztve lehet felhasználni a választott TCP/IP stacket. Az így kialakított szoftver architektúra rétegeinek megvalósításával folytattam munkámat.

Az AUTOSAR modulok megvalósítását minden esetben a konfigurációs lehetőségek áttekintésével kezdtem, majd a konfigurációhoz és a modul működéséhez szükséges adatszerkezetek implementációjával folytattam. A szabványban meghatározott függvények írása a modul megvalósításának utolsó lépése volt.

Az Ethernet interfésznél a szabványban megadott indexelés betartása volt a kihívás. Itt bemutattam az AUTOSAR szerinti Ethernet alapú kommunikációt.

A választott nyílt forrású TCP/IP verem az LWIP volt, így a dolgozatomat az LWIP azon részeinek bemutatásával kezdem, ami feltétlenül szükségesek az LWIP működéséhez, viszont a dolgozat további részeihez nem kapcsolódnak szorosan. A dolgozat középső részében mutatom be az LWIP-vel elkészült TCP/IP megvalósítás működését és összehasonlítását az AUTOSAR által elvárt működéssel. Itt olyan lényegesebb funkciókra fektettem a hangsúlyt, mint a TCP kapcsolat felépítése TCP és UDP csomagok küldése. Sikerült elérnem azt, hogy az eddig elkészült TCP/IP

megvalósítást, mint fekete dobozt tekintjük és csak az AUTOSAR által definiált függvényeket használjuk, akkor nem látszik az, hogy a tényleges működést egy nyílt forrású TCP/IP stack valósítja meg.

Bár már munka e szakaszán látszottak azok a problémák, amik végül azt eredményezték, hogy egy teljesen új TCP/IP stack megvalósításába kezdek, ennek ellenére megpróbálkoztam az elkészült TCP/IP egyfajta teljesítmény mérésével. Ehhez a feladathoz, kidolgoztam egy mérési eljárást és saját szoftver eszközöket hoztam létre a méréshez, de a mérési elv és a rendelkezésre álló eszközök korlátozott eredményt adtak.

Az Ethernet interfész és az LWIP működésének megismerése és az eddig elvégzett munka olyan tapasztalatot nyújtott, ami feltétlenül szükséges volt a további munkám során, ahol egy saját TCP/IP stack megvalósítását kezdem el. A TCP/IP komplexitása és mérete miatt szükség volt kisebb almodulokra bontani. Az almodulok bemutatásánál, igyekeztem függvényhívásokon és az almodulok állapotain keresztül bemutatni a modulok működését. Nem akartam részletesen bemutatni minden egyes függvény minden egyes paraméterét. Az almoduok működésének megértéséhez segítséget nyújtanak a függelékben található szekvencia diagramok. A kitűzött cél az UDP csomagok küldése, fogadása és a ping parancsra válaszolnia kell tudnia. Sikerült elérnem ezeket a célokat, de még így is elmondható, hogy több AUTOSAR szabvány által megfogalmazott követelmény hiányzik, mint ami meg van valósítva.

Az AUTOSAR TCP/IP stack tovább fejlesztése, vagyis hiányzó követelmények megvalósítása az IP almodul RFC által is megfogalmazott működésének kiegészítése lehet. Ezek a csomag tördelés (fragmentáció) RX/TX irányban, routolás lehetősége, vagyis egy routing táblázat és a táblázat kezelő függvényeinek hozzá adása az elkészült IP almodulhoz. A további lépésekhez felhasználástól függő sorrendet lehet felállítani, de az IP-címek kiosztásával kapcsolatos almodulok, mint a DHCP, vagy AutoIp megkönnyítik a konfigurációt. Ha megbízható IP alapú kapcsolatot szeretnénk, akkor további fejlesztési cél lehet a TCP protokoll megvalósítása.

A dolgozatomat az almodulok tesztelésével zárom. Valójában minden almodul elkészítése után rögtön az almodul tesztelésével folytattam a munkát. Utolsó lépésként az almodulokat felhasználva megvalósítottam azokat az AUTOSAR TCP/IP függvényeket, amik szükségesek voltak egy UDP csomag küldéséhez és fogadásához, majd mikrovezérlőre letöltve próbáltam ki a programot.

Irodalomjegyzék

[1] Scherer Balázs: *Autóipari Beágyazott Rendszerek-Diagnosztika c. jegyzet(BME_MIT)*

[2] Charles L. Hedrick: *Introduction to the Internet Protocols 1987*(Original Document: http://www.szabilinux.hu/tcpip/tcptut.zip)- Hungarian translation Vincze Tamás.: *Bevezetés az Internet Protokollokba 1996,* (http://www.szabilinux.hu/tcpip/index.html)

[3] Dr Kónya László: *Számítógép-hálózatok 2006*

[4] AUTOSAR Consortium. *AUTOSAR Technical Overview, 2011.* http://www.autosar.org/fileadmin/files/releases/3-2/main/auxiliary/AUTOSAR_TechnicalOverview.pdf

[5] AUTOSAR Consortium. *Specification of Ethernet Interface 2014* http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/communication-stack/standard/AUTOSAR_SWS_EthernetInterface.pdf

[6] AUTOSAR Consortium. *Specification of Ethernet Driver 2014* http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/communication-stack/standard/AUTOSAR_SWS_EthernetDriver.pdf

[7] Adam Dunkels: *lwIP Documentation 2008* http://www.nongnu.org/lwip/main.html

[8] AUTOSAR Consortium. *Specification of TCP/IP Stack 2014* http://www.autosar.org/fileadmin/files/releases/4-1/software-architecture/communication-stack/standard/AUTOSAR_SWS_TcpIp.pdf

[9] Vincze Tamás és Vajda János *Hálózati kislexikon* http://www.kfki.hu/~cheminfo/hun/olvaso/lexikon/s.html

[10] Petrényi József*: TCP/IP Alapok I. kötet V2.0 2009* http://mek.oszk.hu/08300/08374/pdf/tcpip1.pdf

[11] Mahmoud Alshinhab:*TCP 3 way handshake* 2012 https://tuxawy.wordpress.com/page/2/

[12] RFC: 793 *TRANSMISSION CONTROL PROTOCOL* (DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION)1981 https://www.ietf.org/rfc/rfc793.txt

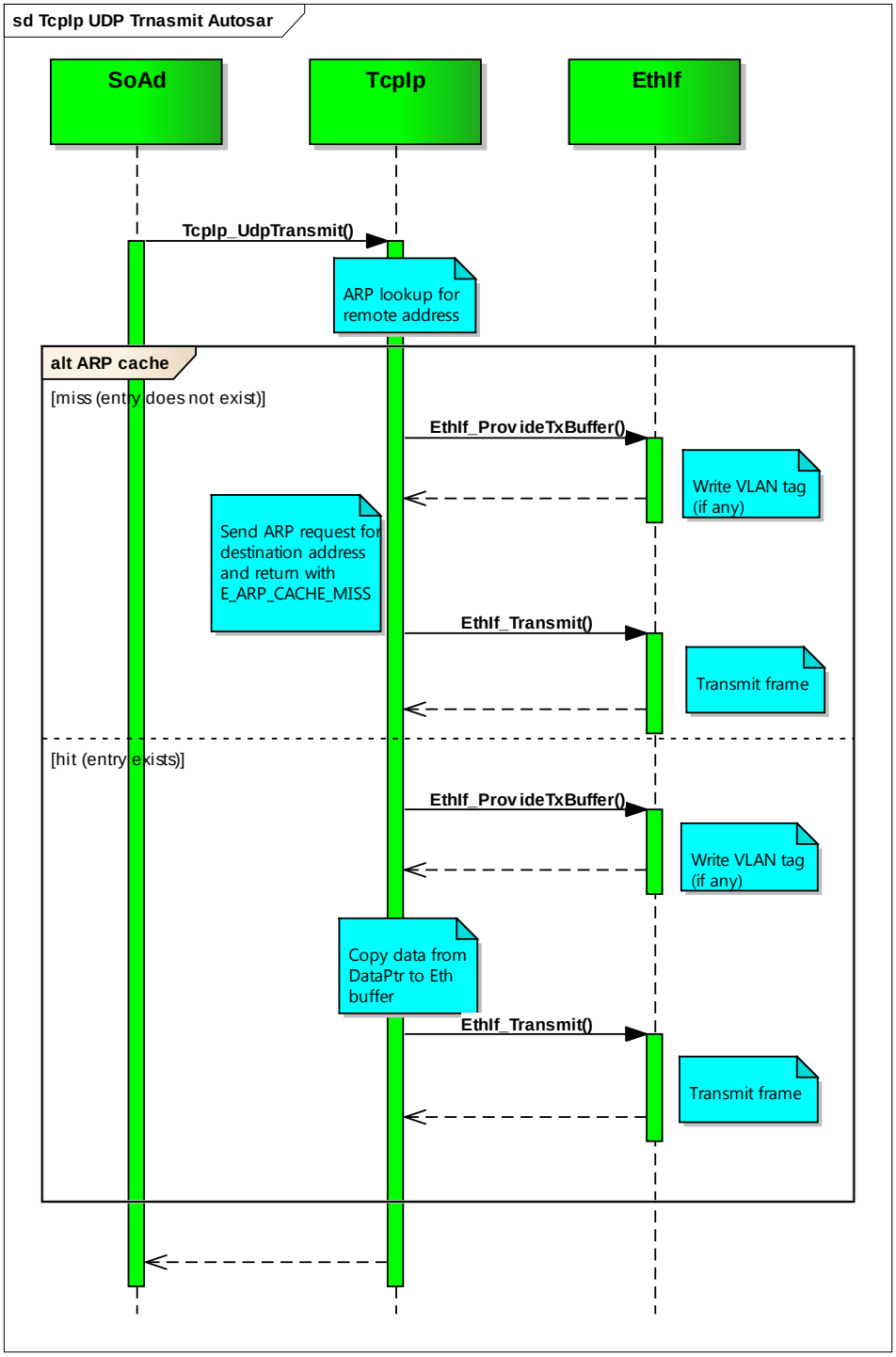
[13] iptables.info: *TCP connections* http://www.iptables.info/en/connection-state.html

[14] Majzik István: *Előadásvázlat a „Valósidejű és biztonságkritikus rendszerek” tárgyhoz (BME-MIT 2010)*

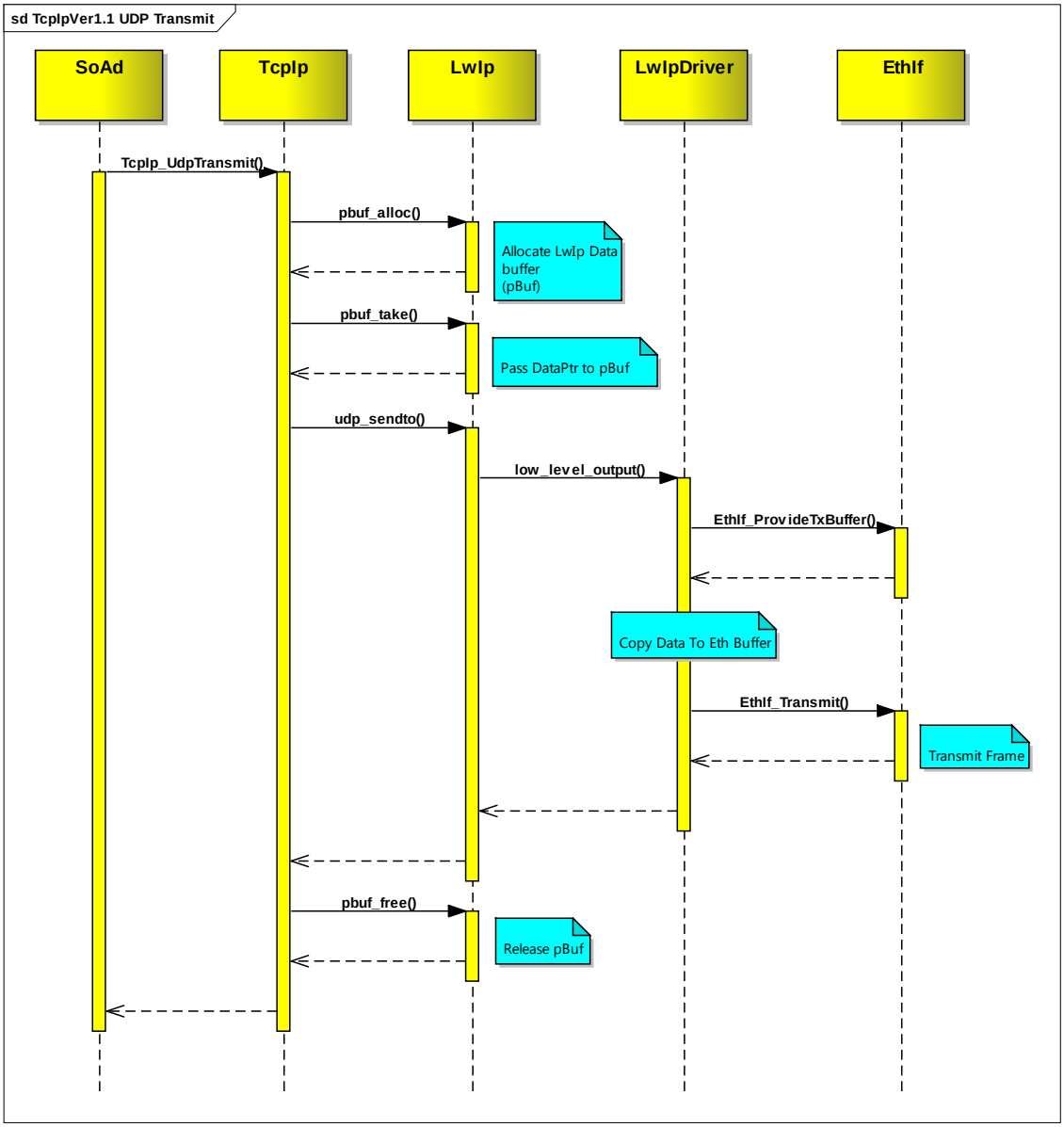
Függelék

Szekvencia diagramok

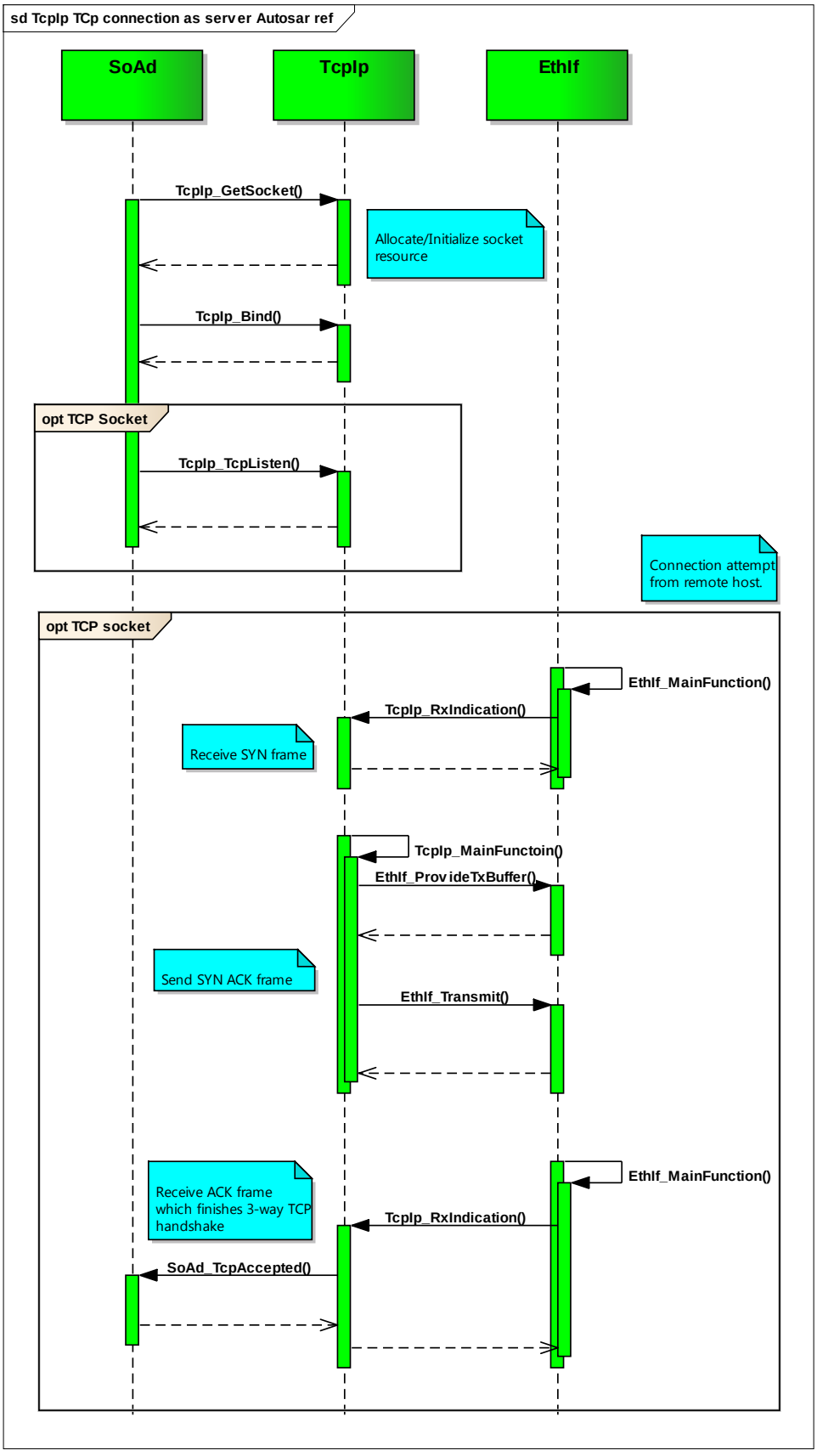
F1 AUTOSAR UDP Transmit



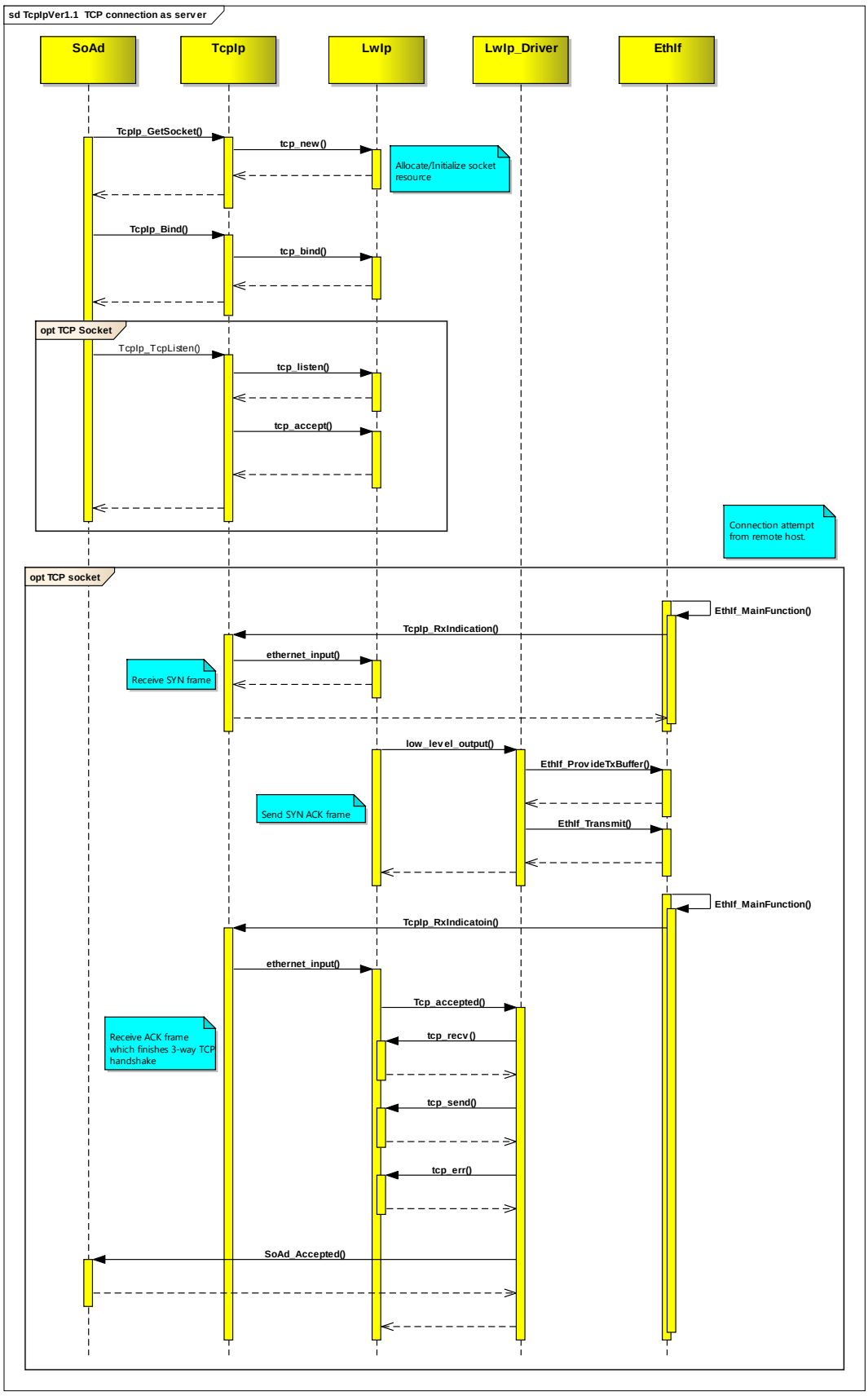
F2 UDP Transmit folymat LWIP-vel



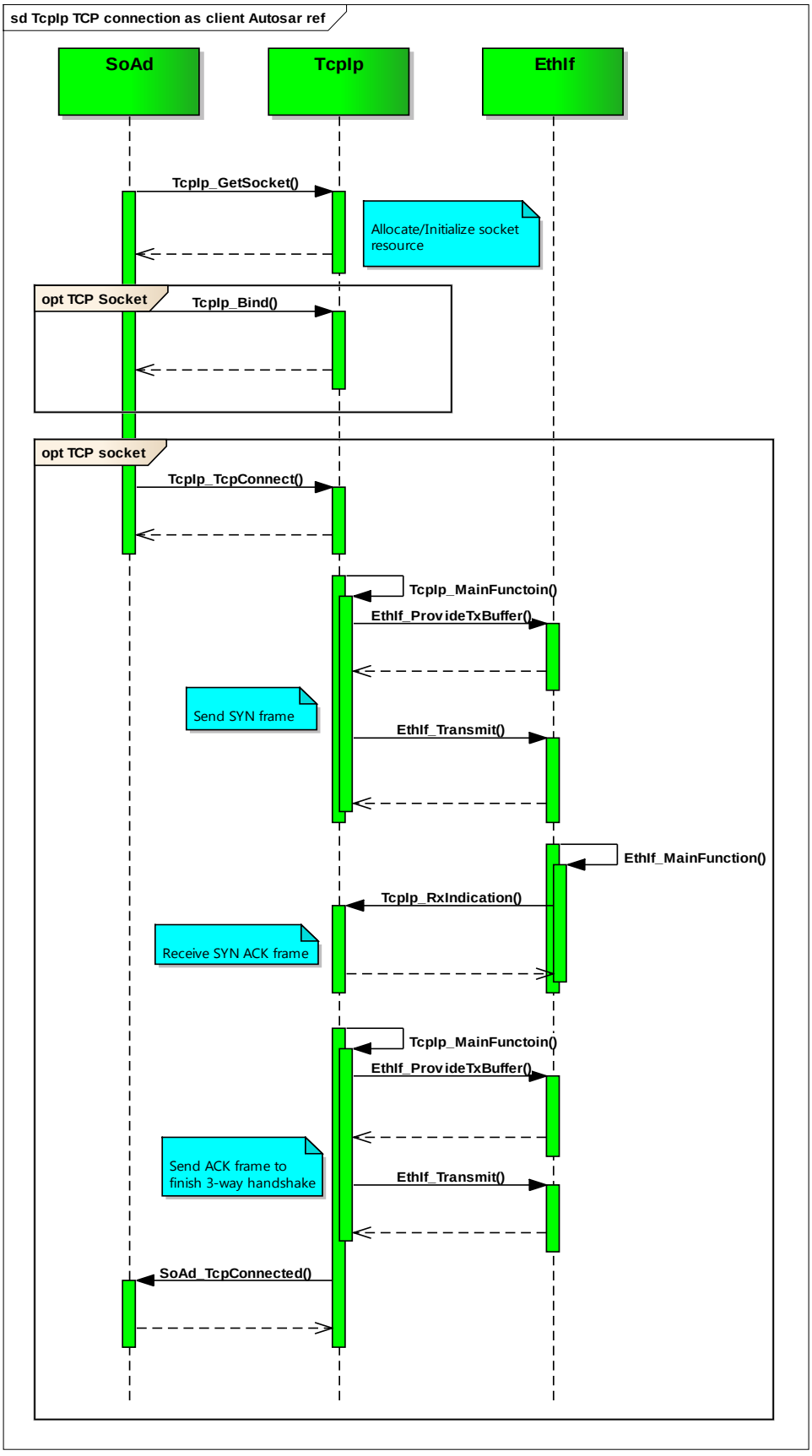
F3 AUTOSAR TCP Csatlakozási folyamat, mint szerver



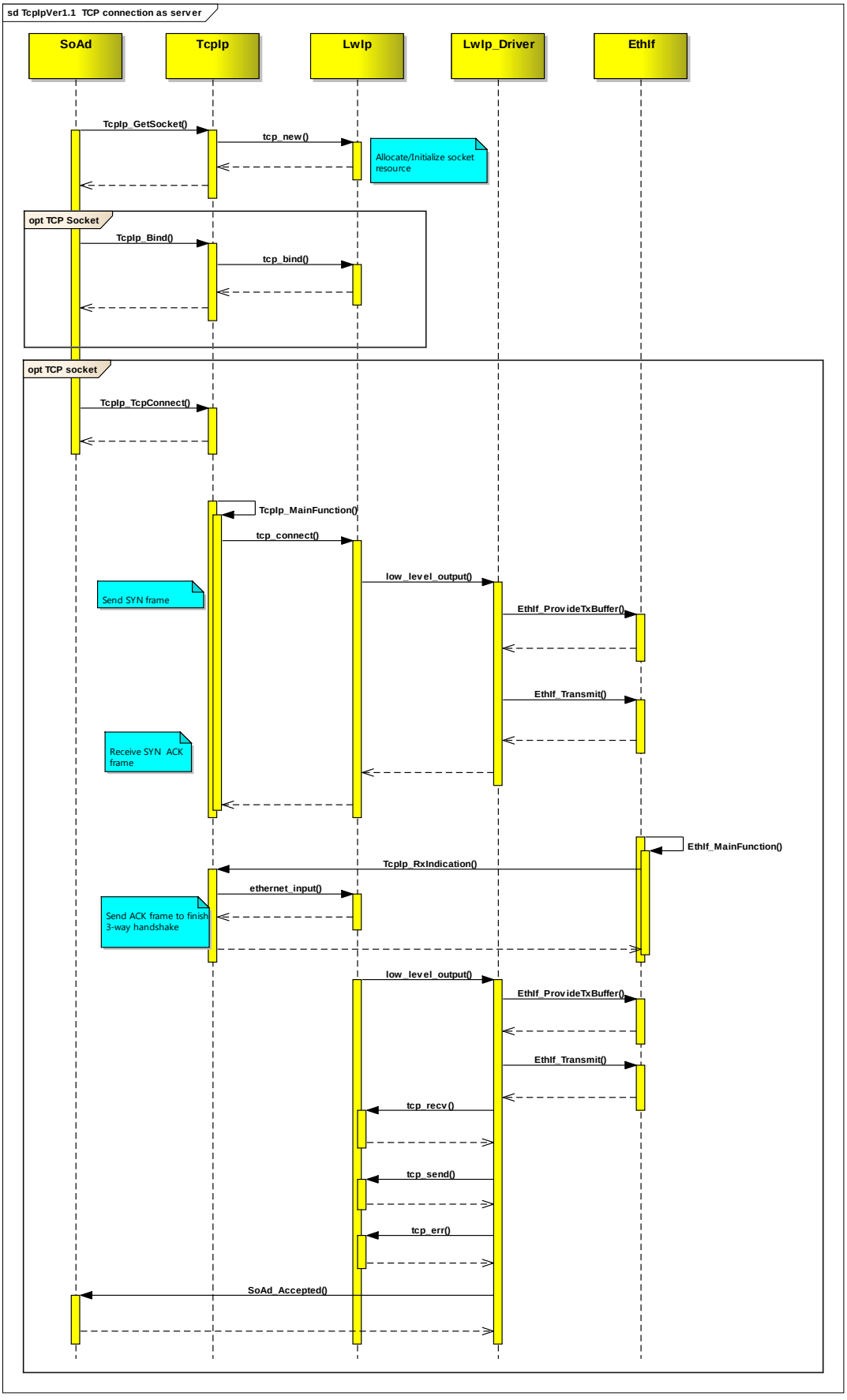
F4 TCP csatlakozási folyamat, mint szerver LWIP-vel



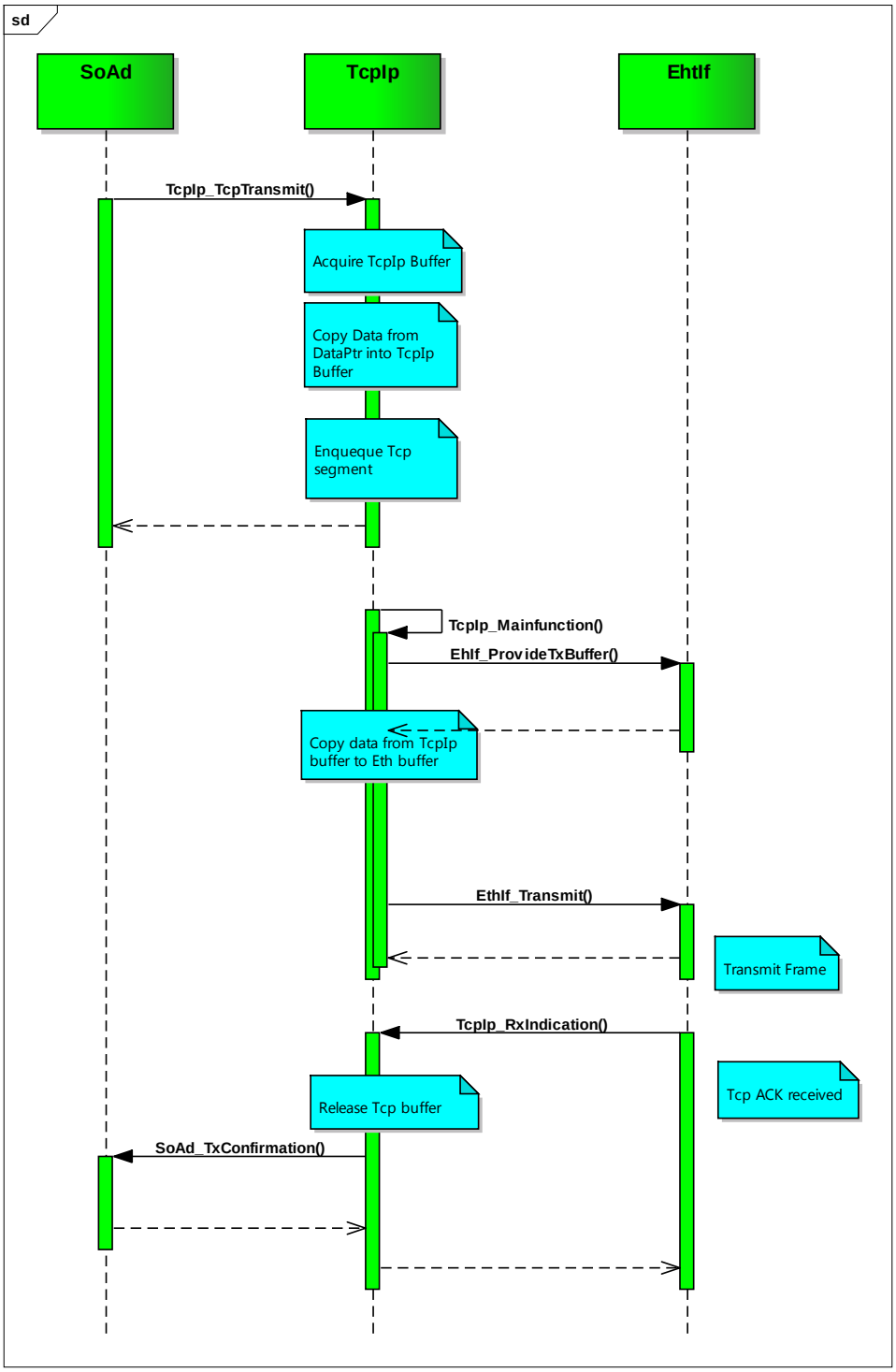
F5 AUTOSAR TCP kapcsolódás, mint kliens



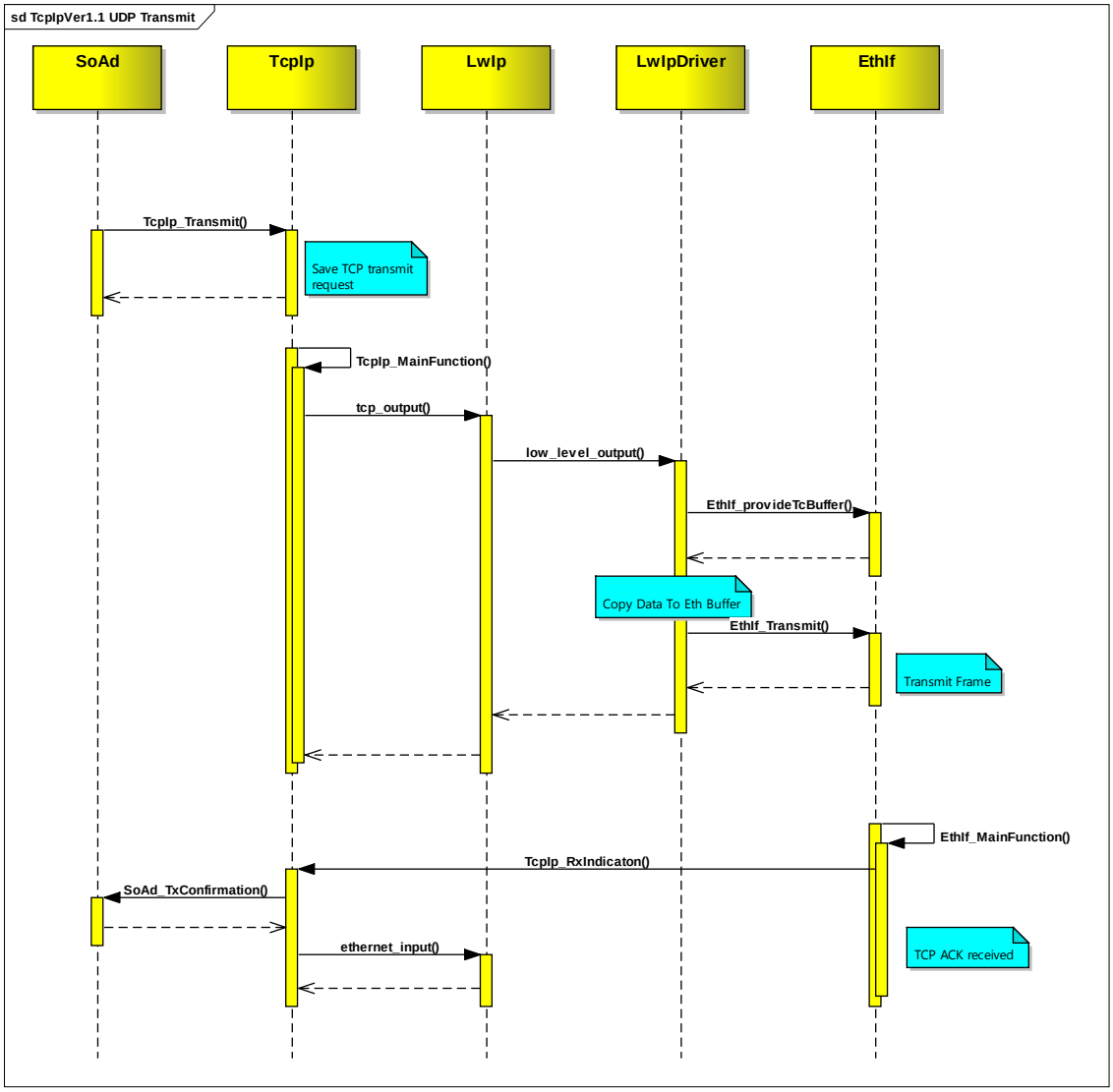
F6 TCP Csatlakozási folyamat, mint kliens LWIP-vel



F7 AUTOSAR TCP küldési folyamat



F8 TCP küldési folyamat LWIP-vel



Mérési eredmények

F9 RX irányú mérési eredmények a megszakítás tiltásával

Rx Test Result in Polling mode			
Min[ms]	Avg[ms]	Max[ms]	Result
0.063	0.605	499.527	NOK
0.077	2.884	576.316	NOK
0.077	2.496	500.087	NOK
0.077	2.969	1040.226	NOK
0.077	2.603	500.153	NOK
0.078	2.557	500.069	NOK
0.078	2.988	1143.737	NOK
0.078	2.504	500.086	NOK
0.079	3.028	963.389	NOK
0.079	3.998	3136.467	NOK
0.079	2.562	500.153	NOK
0.079	2.562	500.130	NOK
0.079	2.528	500.044	NOK
0.080	2.724	521.518	NOK
0.080	3.076	1067.778	OK
0.080	3.030	997.465	NOK
0.080	0.609	497.576	NOK
0.081	2.604	500.058	NOK
0.081	3.076	1319.538	OK
0.081	2.606	500.062	NOK
0.081	2.954	908.349	NOK
0.081	2.695	500.080	OK
0.081	2.147	500.070	OK
0.081	2.593	500.140	NOK
0.081	2.480	500.203	NOK
0.081	5.637	6156.527	OK
0.081	4.286	3360.155	OK
0.081	2.720	500.164	OK
0.081	2.472	500.126	OK
0.081	2.397	500.079	NOK
0.082	3.281	1276.862	NOK
0.083	0.644	498.284	OK
0.083	2.616	500.103	OK
0.086	2.816	500.085	OK
0.086	4.043	2990.494	OK
0.091	2.579	500.155	OK
0.091	2.833	500.143	OK
0.095	2.977	984.983	OK
0.096	2.433	500.114	OK
0.099	3.065	1048.278	OK
0.103	3.207	1292.259	OK
0.103	3.184	1481.363	OK
0.105	2.524	500.096	OK
0.105	2.875	1018.549	OK

0.107	2.584	500.085	OK
0.107	2.877	683.353	OK
0.108	3.250	1151.441	NOK
0.108	2.972	504.812	NOK
0.109	3.087	851.624	NOK
0.113	3.652	2146.815	OK
0.113	3.116	995.485	OK
0.115	3.137	1030.173	OK
0.115	3.009	986.932	OK
0.117	3.425	1486.652	NOK
0.117	3.501	1889.572	OK
0.117	3.014	798.130	OK
0.119	3.090	1092.502	OK
0.110	3.034	1023.052	OK
0.110	2.719	500.253	NOK
0.110	2.486	500.155	OK
0.120	3.063	847.322	OK
0.120	2.475	500.079	OK
0.122	2.988	500.054	OK
0.122	2.737	500.255	NOK
0.122	4.050	2714.098	NOK
0.122	3.491	1899.299	OK
0.122	4.384	3400.459	NOK
0.124	3.323	1319.780	NOK
0.124	2.485	500.044	OK
0.124	2.969	824.153	OK
0.127	3.074	863.660	OK
0.127	3.178	1209.530	OK
0.130	2.432	500.103	OK
0.129	2.923	751.961	OK
0.131	0.679	498.309	OK
0.132	3.438	1498.624	NOK
0.132	2.543	500.190	OK
0.135	2.619	500.102	OK
0.135	2.705	507.350	NOK
0.135	3.370	1776.921	OK
0.135	3.161	1095.854	OK
0.136	2.849	500.101	OK
0.140	2.603	500.120	OK
0.141	2.617	500.120	OK
0.143	2.534	500.131	OK
0.145	1.456	1885.599	OK
0.146	0.867	500.110	OK
0.146	0.686	498.308	OK
0.148	1.195	587.896	NOK
0.148	0.894	500.147	OK
0.148	1.104	500.104	OK
0.148	1.142	500.139	OK
0.148	0.976	1209.056	OK
0.149	1.015	500.062	OK
0.151	1.171	1329.804	OK
0.152	1.502	1203.539	OK

0.154	0.333	496.918	OK
0.154	1.137	2136.556	OK
0.164	1.667	1953.315	OK
0.169	0.363	497.520	OK
0.169	0.891	500.161	OK
0.321	52.253	863.439	OK

F10 RX irányú mérési eredmények a megszakítás engedélyezésével

Rx Test Result in IRQ mode			
Min[ms]	Avg[ms]	Max[ms]	Result
0.035	0.570	499.916	NOK
0.037	0.600	499.479	OK
0.047	0.570	498.344	NOK
0.047	0.570	497.045	NOK
0.048	0.743	500.094	NOK
0.048	0.579	498.020	NOK
0.048	1.696	1913.812	NOK
0.048	1.655	2172.916	NOK
0.049	0.575	497.293	NOK
0.049	0.564	498.066	NOK
0.049	0.574	497.664	NOK
0.049	1.847	2552.010	NOK
0.050	0.939	754.649	NOK
0.050	0.572	499.652	NOK
0.050	1.143	1153.627	NOK
0.050	0.981	830.910	NOK
0.050	0.574	500.086	NOK
0.050	1.048	918.084	NOK
0.050	1.399	1642.910	NOK
0.051	0.519	497.701	NOK
0.051	0.581	498.806	OK
0.051	0.580	497.031	NOK
0.052	0.794	722.575	OK
0.052	0.794	722.575	NOK
0.052	0.568	497.934	NOK
0.052	1.127	1124.324	NOK
0.052	1.577	2021.922	OK
0.052	1.049	964.871	NOK
0.052	0.571	499.173	OK
0.052	1.901	2668.341	OK
0.052	0.574	498.127	NOK
0.052	1.468	1784.679	OK
0.052	0.576	497.862	NOK
0.052	0.583	498.045	NOK
0.052	0.569	497.723	NOK
0.052	0.572	498.300	OK
0.052	0.582	497.179	OK
0.052	1.136	1135.282	OK
0.052	0.584	497.288	NOK
0.052	4.892	8649.983	OK
0.052	0.606	497.336	OK
0.052	0.628	516.216	NOK

0.053	1.056	960.691	NOK
0.053	1.810	2467.306	OK
0.053	0.583	499.321	OK
0.053	1.195	1260.758	NOK
0.053	0.568	499.630	NOK
0.053	1.599	2049.600	OK
0.053	0.568	498.648	NOK
0.053	1.108	1060.588	OK
0.053	0.579	497.512	OK
0.053	1.074	1015.010	OK
0.053	0.582	499.850	OK
0.053	0.577	506.363	OK
0.053	0.585	497.447	OK
0.053	1.068	976.396	OK
0.053	0.580	497.290	OK
0.053	0.579	497.389	OK
0.053	1.138	1130.165	OK
0.053	1.848	2568.182	NOK
0.053	0.578	498.000	NOK
0.053	1.229	1306.817	OK
0.053	0.582	497.621	OK
0.054	2.040	2942.713	NOK
0.054	0.569	498.007	OK
0.054	1.453	1770.725	OK
0.054	0.570	498.514	OK
0.054	0.579	497.405	NOK
0.054	1.342	1530.139	OK
0.054	0.570	497.716	OK
0.054	3.507	5858.263	OK
0.055	0.748	498.780	OK
0.055	0.587	498.247	OK
0.055	0.581	497.566	OK
0.055	0.590	498.104	OK
0.056	0.576	497.584	OK
0.057	0.593	498.521	OK
0.058	0.575	497.586	OK
0.058	0.972	773.600	OK
0.058	0.572	497.668	OK
0.060	0.801	498.178	OK
0.060	0.580	499.505	OK
0.062	0.591	498.058	OK
0.062	1.282	1373.457	OK
0.062	1.022	889.822	OK
0.065	2.359	3559.290	OK
0.064	0.586	498.407	OK
0.067	0.589	496.829	OK
0.067	0.996	814.043	OK
0.069	1.523	1838.333	OK
0.072	0.606	497.340	OK

0.074	1.082	946.204	OK
0.074	0.605	497.470	OK
0.074	0.611	497.640	OK
0.074	1.227	1264.587	OK
0.076	1.130	1031.988	OK
0.077	0.604	498.507	OK
0.079	1.284	1366.416	OK
0.081	0.608	497.305	OK
0.080	0.617	498.762	OK

F11 TX mérési eredmények

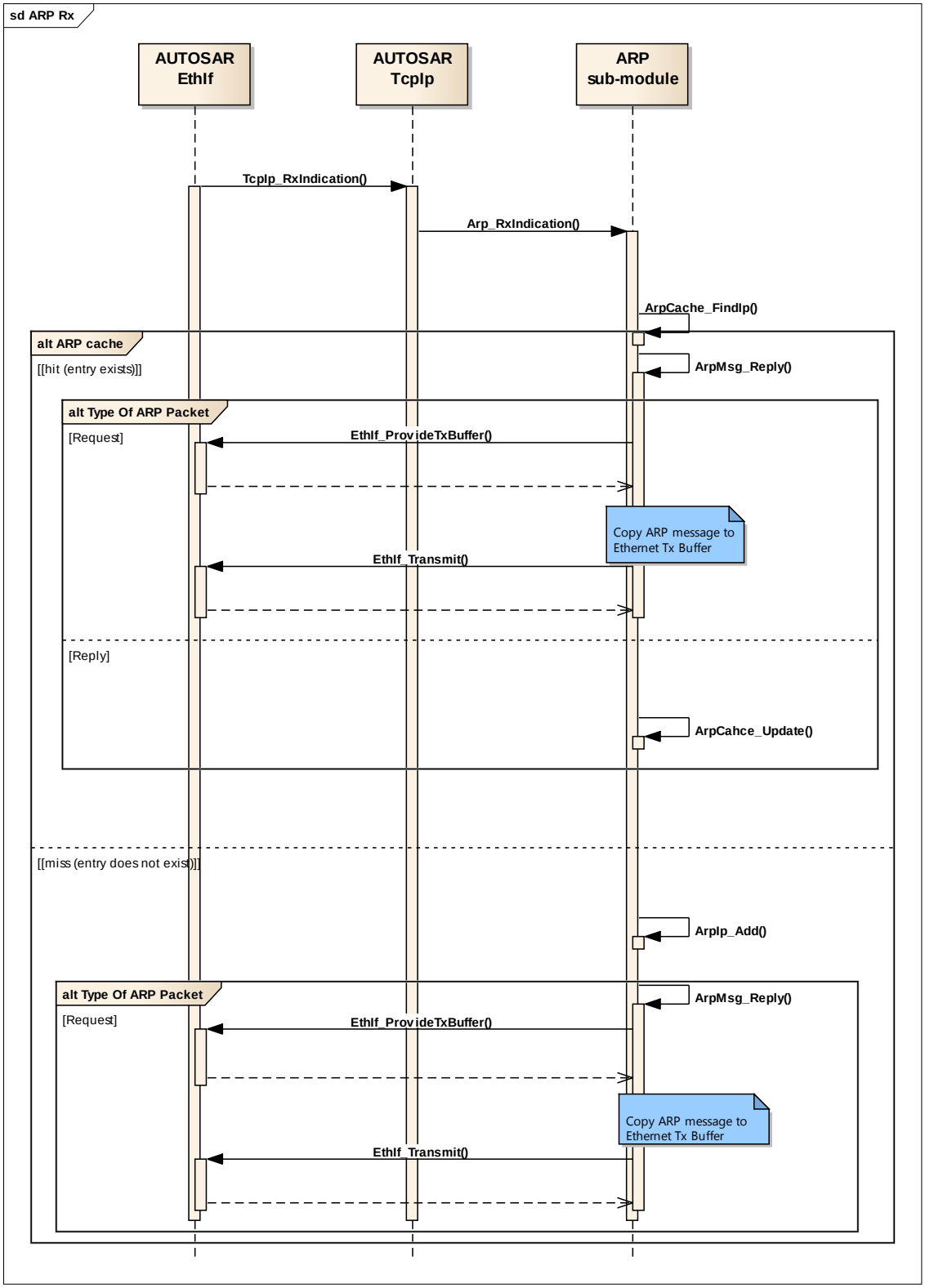
Tx Test Result [64byte]		
Min[µs]	Avg[µs]	Max[µs]
25.000	143.471	33808.000
25.000	144.098	33792.000
1.000	126.226	33896.000
1.000	146.540	34068.000
1.000	149.283	34392.000
1.000	123.331	34013.000
1.000	127.411	34094.000
1.000	136.581	34725.000
1.000	140.966	33836.000
1.000	131.218	34196.000

Tx Test Result [128byte]		
Min[µs]	Avg[µs]	Max[µs]
1.000	179.719	33947.000
1.000	182.704	34047.000
1.000	211.191	34528.000
1.000	199.071	33849.000
1.000	189.416	33912.000
1.000	208.717	34044.000
1.000	205.686	33880.000
1.000	226.073	34268.000
1.000	207.231	34236.000
1.000	215.847	34013.000

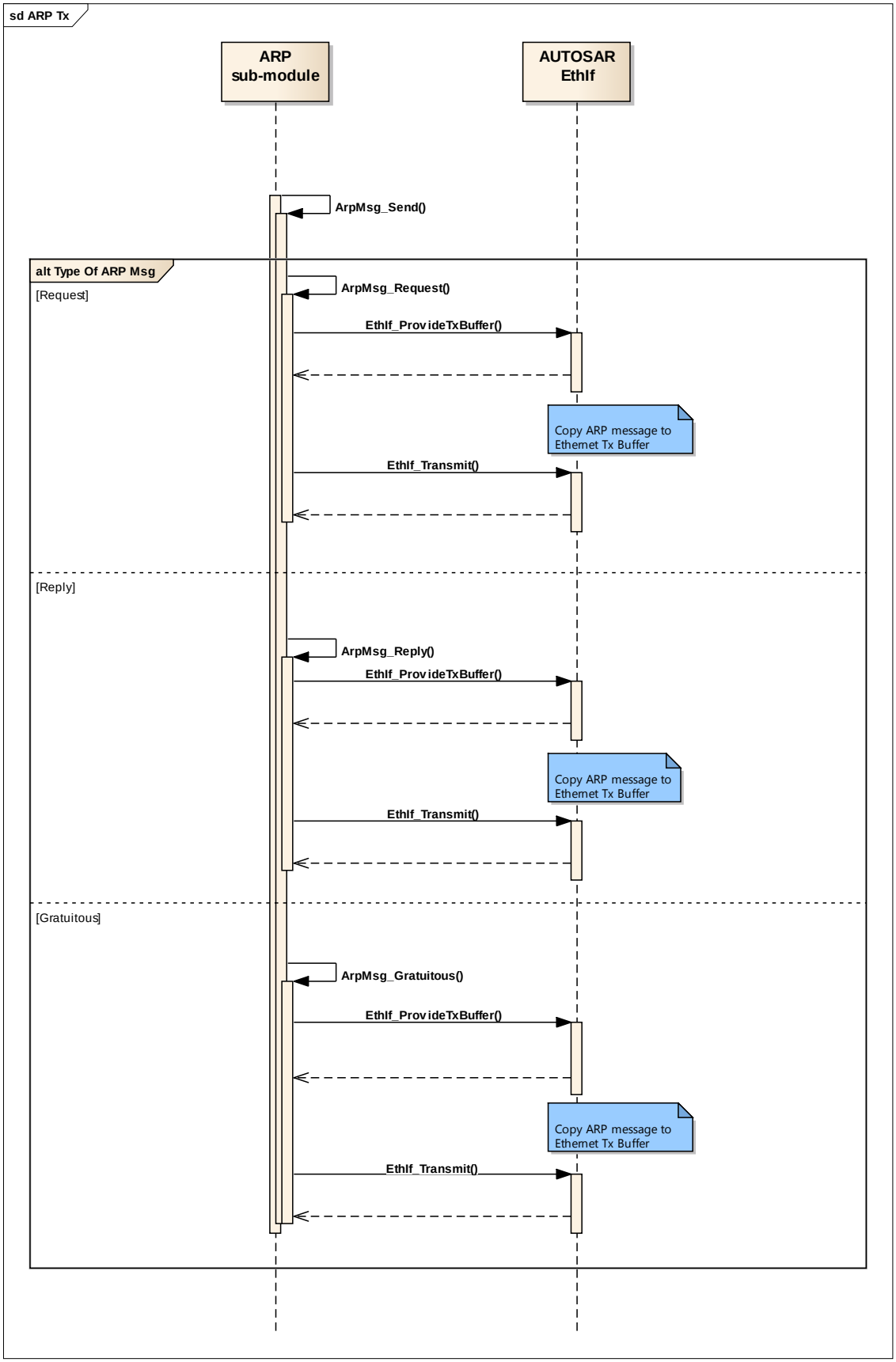
Tx Test Result [256byte]		
Min[µs]	Avg[µs]	Max[µs]
1.000	267.203	34407.000
1.000	325.238	34002.000
1.000	463.120	34206.000
1.000	273.869	34287.000
1.000	296.777	34154.000
1.000	320.154	34292.000
1.000	276.333	34308.000
1.000	282.873	34486.000
1.000	290.471	33903.000
1.000	314.744	34153.000

Szekvencia diagramok 2

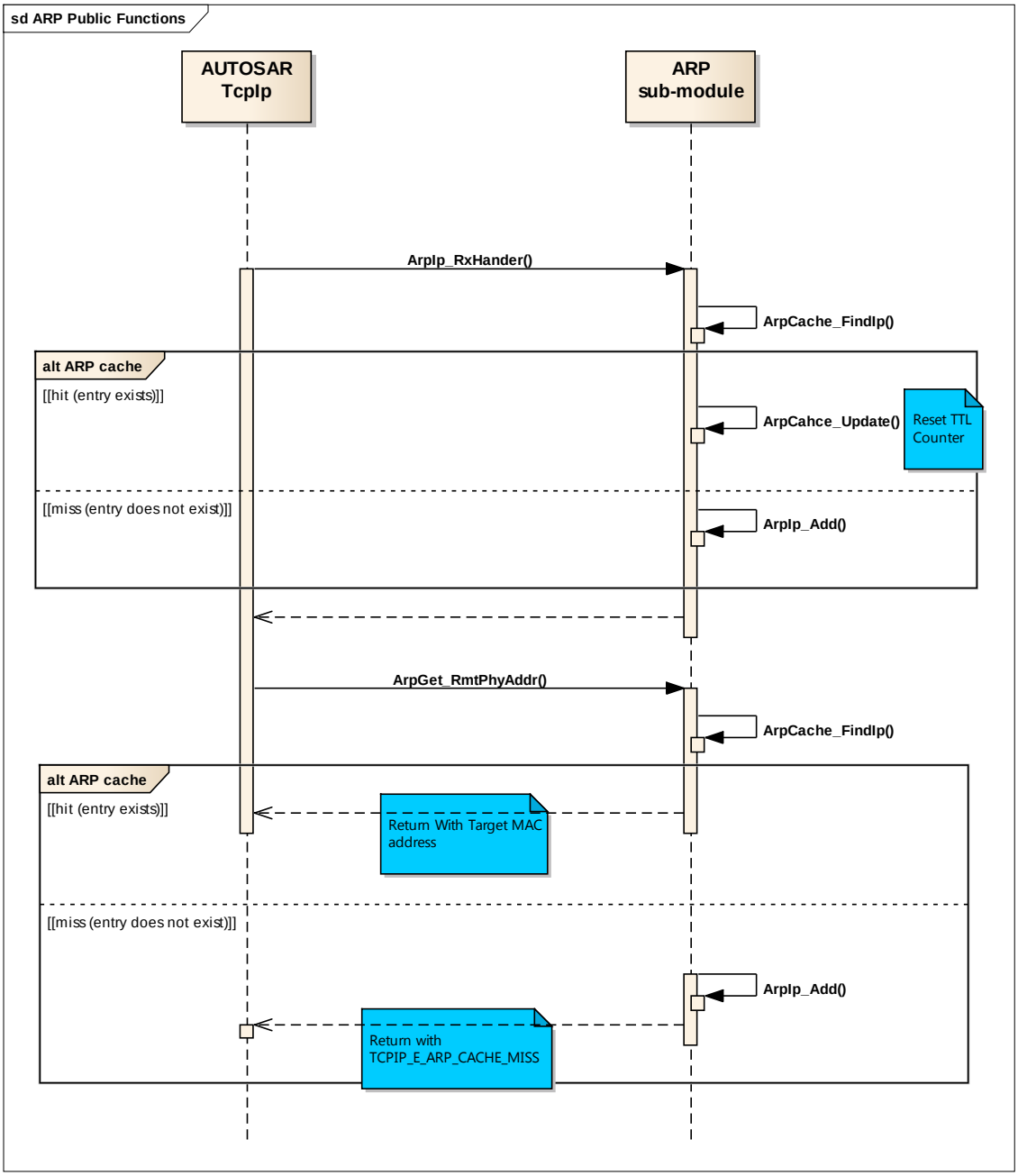
F12 ARP csomag fogadása



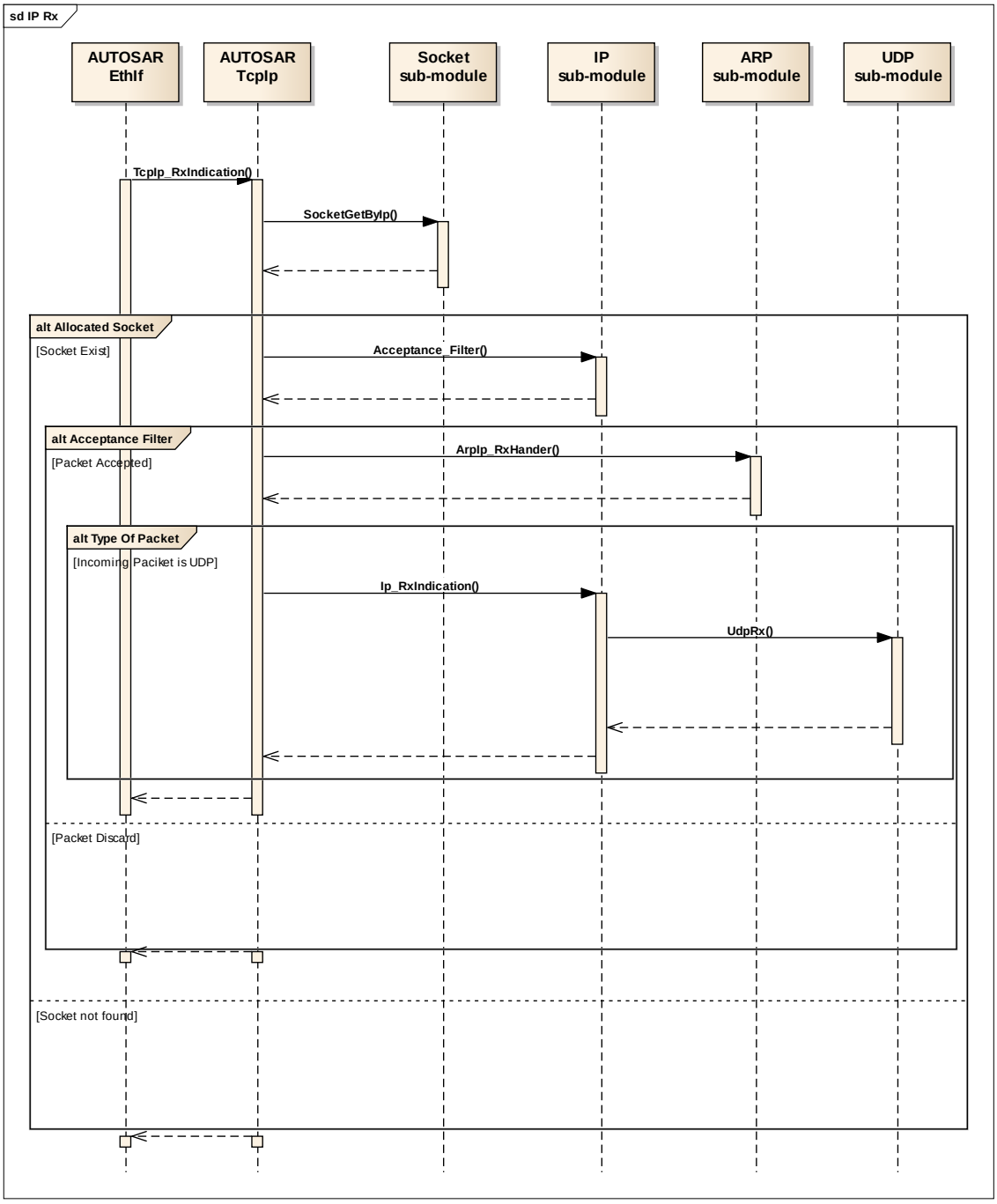
F13 ARP csomagok küldése



F14 IP csomagok kezelése az ARP almodulban



F15 IP csomag fogadása



F16 IP csomag küldése