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

Bányai Tamás

CAN INTELLIGENS HÁLÓZATI KAPCSOLÓ MEGVALÓSÍTÁSA

Diplomaterv

KONZULENS

Dr. Balogh András
(ThyssenKrupp Presta Hungary Kft.)

Dr. Sujbert László
(BME-MIT)

BUDAPEST, 2016

DIPLOMATERV-FELADAT

Bányai Tamás (R44P4P)
szigorló villamosmérnök hallgató részére

CAN intelligens hálózati kapcsoló megvalósítása

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

A vezérlőegységek teszteléséhez szükség van speciális eszközökre, melyek a fenti hálózati kapcsolatokon történő kommunikációt szimulálják, illetve különböző felsőbb szintű protokollok segítségével a vezérlőegység egyes funkcióit (diagnosztikai kommunikáció, szoftverfrissítés, stb.) tesztelik. Annak érdekében, hogy egy-egy ilyen speciális eszközt több egység teszteléséhez lehessen automatikusan felhasználni, az eszköz és a teszt alatt álló rendszer összekötésének dinamikusan átkonfigurálhatónak kell lennie.

A jelölt feladata egy intelligens CAN kapcsoló (switch) elkészítése és illesztése a ThyssenKrupp Presta Hungary Kft.-nél alkalmazott tesztrendszerhez. Részletesebben, a következő részfeladatokat kell megvalósítani:

- Az eszközzel kapcsolatos követelmények összegyűjtése
- A rendszerarchitektúra megtervezése, hardver- és szoftverterv készítése
- A hardver részletes tesztelése, az elkészült eszköz illesztése
- Különböző switching algoritmusok vizsgálata és megvalósítása
- Forgalomnaplózás megvalósítása
- Hibainjektálási funkciók megvalósítása (üzenet eldobása, ismétlés, késleltetés stb.)
- Az elkészült eszköz tesztelése a célkörnyezetben.

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

Budapest, 2016. március 19.

Dr. Dabóczi Tamás
tanszékvezető

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

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057. Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

Tartalomjegyzék

Összefoglaló ... **6**
Abstract ... **7**
1. Hálózati szoftver általános felépítése ... **8**
1.1. Réteg alapú hálózati szoftverarchitektúrák ... 8
1.2. OSI hivatkozási modell ... 8
1.3. Adatkapcsolati réteg feladatai ... 10
1.4. Kapcsoló (Switch) ... 10
2. Kapcsolási algoritmusok alapjai ... **12**
2.1. Kapcsolási alapfogalmak ... 12
2.2. Kapcsolási módszerek csoportosítása ... 14
2.2.1. Vonalkapcsolás (Circuit switching) ... 15
2.2.2. Csomagkapcsolás (Packet switching) ... 16
2.3. Kommunikációs módszerek csoportosítása ... 17
3. Pont-pont hálózatok kapcsolási algoritmusai ... **18**
3.1. Mérőszámok és elemzési szempontok ... 18
3.1.1. Teljesítménymutatók ... 18
3.1.2. Elemzési szempontok és egyszerűsítések ... 20
3.2. Pont-pont kapcsolási algoritmusok elemzése ... 21
3.2.1. Vonalkapcsolás (Circuit Switching, CS) ... 22
3.2.2. „Tárol és továbbít” kapcsolás (Store-and-Forward, SF) ... 24
3.2.3. „Átvágó” kapcsolás (Cut-through, CT) ... 27
3.2.4. „Féreglyuk” kapcsolás (Wormhole Switching, WS) ... 30
3.3. A teljes blokkolásmentes hálózati késleltetés vizsgálata ... 34
3.4. Gyakorlatban használt kapcsolások összehasonlítása ... 35
4. Követelmények az elkészített eszközzel szemben ... **39**
4.1. Követelmények összegyűjtési módja ... 39
4.2. Célfunkció ... 40
4.3. Funkcionális követelmények ... 41
4.4. Vezérlési és csatlakozási lehetőségek ... 42
4.5. Fizikai kialakítás ... 43
4.6. Fejlesztés indoka ... 43

5. A hardvertervezés lépései 45
5.1. A hardverterv blokkvázlata 45
5.2. Nyomtatott áramköri lemez kialakítása 48
5.3. Tápellátás 49
5.4. Központi feldolgozóegység 50
5.5. Ethernet-interfész és -meghajtó 51
5.6. XPORT-interfész 51
5.7. CAN fizikai meghajtó 52
5.8. CAN-illesztő 53
5.9. Követelmények hardveres teljesítése 59
6. A szoftvertervezés lépései 61
6.1. A szoftverarchitektúra 61
6.2. A szoftverterv blokkvázlata 64
6.3. CAN vezérlőmodul 66
6.3.1. CAN ki- és bemenetkezelő (CanGpio) 67
6.3.2. CAN-üzenet RAM (CanMsgRam) 68
6.3.3. Alacsonyszintű CAN-vezérlő (CanModuleHandler) 69
6.3.4. CAN kapcsolótábla-kezelő (CanSwitchList) 70
6.3.5. CAN várakozásisor-kezelő (CanQueue) 72
6.3.6. CAN forgalomnaplózó (CanTrafficLogger) 74
6.3.7. Magasszintű CAN-funkciók (CanFunctions) 75
6.4. Követelmények szoftveres teljesítése 77
7. Megvalósított kapcsolási algoritmusok 78
8. Az elkészült eszköz tesztelése 79
8.1. Tesztelés fejlesztés közben 79
8.2. Az elkészült eszköz teljesítménytesztje 80
8.3. Beillesztés a tesztkörnyezetbe 83
9. Összefoglalás, továbbfejlesztési lehetőségek 84
Irodalomjegyzék 86
Ábrajegyzék 88
Függelék 89

HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2016. 12. 09.

...……………………………………………….

Bányai Tamás

Összefoglaló

A mai, modern személygépjárművekben számos fedélzeti kommunikációs sín köti össze a vezérlőegységeket, amelyek tesztelése legegyszerűbben e síneken keresztül lehetséges. A hatékony tesztfuttatásához szükség van speciális kapcsolóeszközökre, amelyek képesek dinamikusan átkonfigurálni a tesztelt rendszerek összeköttetéseit.

Az autók kommunikációs hálózatában általános esetben nincs szükség az összeköttetések megváltoztatására, így nincsenek piaci megoldások erre a feladatra. Ráadásul a tesztkörnyezet számos követelményt is támaszt egy kapcsolóval szemben. A diplomatervezés keretében egy CAN-hálózaton működő intelligens kapcsolóeszközt valósítottam meg, amely a speciális követelményeket és számos funkciót is teljesíti.

A dolgozat első felében ismertetem a megszerzett és a feladat megoldásához szükséges elméleti ismereteket. Először rövid áttekintést adok a hálózati szoftverek rétegzett felépítéséről, beleértve az adatkapcsolati réteg feladatait, amelyben a kapcsolóeszközök működnek. Ezután részletesen ismertetem a leggyakoribb kapcsolási algoritmusokat. Kitérek a kapcsolási alapfogalmakra, az algoritmusok jellemzéséhez használható mérőszámokra, és az elemzési szempontokra is. Végül mélyrehatóan elemzem az egyes algoritmusokat, hatékonyság és felhasznált erőforrások szempontjai alapján, és össze is hasonlítom őket egymással.

A dolgozat második felében a megvalósítás konkrét lépéseit és a döntéseim indoklását írom le. Először összegzem az összegyűjtött követelményeket a kapcsolóval szemben. Majd bemutatom az elkészült hardver blokkvázlatát, és kitérek az egyes részegységek tervezése során felmerült legfontosabb problémákra és az azokra adott megoldásokra. Ezután ismertetem a kifejlesztett vezérlőszoftver architektúráját, struktúráját, és részletesebben kitérek a legfontosabb szoftvermodul megvalósítására. A dolgozat végén szót ejtek az elkészült intelligens kapcsoló teszteléséről és a mérések során kapott eredményekről, illetve rávilágítok a jövőbeni fejlesztési lehetőségekre is.

A fejlesztés eredményeként elkészült az intelligens CAN hálózati kapcsolóeszköz, amely képes a CAN hálózati üzenetek kapcsolására az elvártaknak megfelelő sebességgel és késleltetéssel, valamint a követelményekben felsorolt összes funkciót is képes ellátni.

Abstract

Nowadays, a modern automobile contains several communication buses connecting the electronic control units (ECUs) providing an easy way to also test the ECUs. In order to achieve an efficient testing environment special switches are required to implement a run-time dynamic re-configurable connection between units under test.

Modifying connections within a car's communication network is generally never a demand, therefore the market offers no off-the-self solutions for this task. Furthermore, the test environment itself sets numerous technical requirements against such a switch. As a part of this diploma work a switch instrument for CAN network has been implemented which not only fulfills the mentioned special requirements but also provides a set of useful functions.

The first part of the thesis covers the theoretical knowledge required for the task. A short overview is initially given about the layered structure of the network software, discussing also the data link layer tasks where the switches are usually operating. Thereafter, the most frequently used switching algorithms are detailed, touching on their basic concepts, performance metrics and aspects of analysis. This part concludes in the in-depth analysis and even comparison of the algorithms based on aspects of efficiency and used resources.

The second part is dedicated to describing the particular steps of my solution for the task and the decisions behind them. After summarizing the requirements and demands against the switch the hardware block diagram is presented, elaborating on the problems and solutions related to the designing of each component. The architecture and structure of the developed software is then explained, focusing on the main software module. Eventually, the test methods of the CAN switch are described along with the results of the performed measurements and tests. The thesis is sealed with an outlook on possibilities for future further-development.

The result of the process is an actually manufactured instance of the intelligent CAN switch – capable of switching CAN network data packets with desired speed and delay while providing all required functions as well.

1. Hálózati szoftver általános felépítése

Manapság a hálózati eszközök szoftverei, és így a teljes hálózaton működő összes szoftver rengeteg funkcióval rendelkezik, így igen bonyolult felépítésű. Ebben a bevezető fejezetben bemutatom a hálózati szoftverek felépítését, és kiemelten részletezem a hálózati kapcsolókra vonatkozó elveket. A fejezetben előkerülő fogalmaknál, módszereknél az angol terminológiát is leírom zárójelben, hiszen sok forrás csak angolul áll rendelkezésre ebben a témakörben.

1.1. Réteg alapú hálózati szoftverarchitektúrák

A modern hálózatok bonyolult felépítése miatt leggyakrabban a hálózati szoftver egymásra épülő rétegekből (layers) áll. Minden egyes rétegnek egy-egy jól definiált célja és funkciója van, illetve ezen felül az egyes rétegek meghatározott szolgáltatásokat nyújtanak a felettük elhelyezkedő réteg számára, és elrejtik a felsőbbtől az adott réteg megvalósításának részleteit. Ez az elv erőteljesen hasonlít az objektumorientált programozásra. [1]

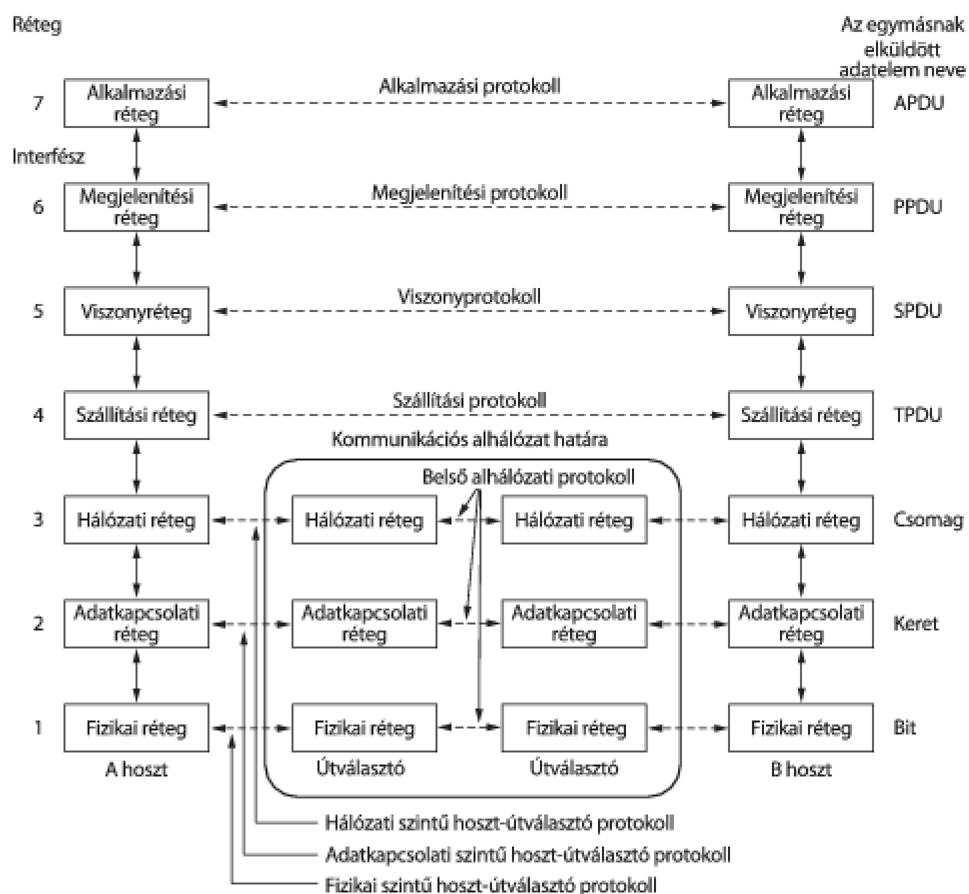
Az egy rétegben, két eszköz közötti kommunikáció szabályait (azaz a rétegbeli üzenetformátum, időzítések, stb. definiálását) az adott réteg protokolljának (protocol) nevezik. A két eszközben megvalósított azonos rétegben szereplő entitásokat, amelyek egymással az adott réteg protokollja szerint kommunikálnak egymással, társentitásnak (peer) nevezik. Az egy eszközön belüli, két szomszédos réteg között definiált kapcsolatot interfésznek (interface) nevezik. Ez előírja, hogy az alacsonyabb réteg milyen elemi műveleteket tesz elérhetővé a felsőbb réteg számára. Egy hálózat által meghatározott rétegek és protokollok halmazát összefoglalóan hálózati architektúrának (network architecture) nevezik. Ha egy adott hálózatban minden rétegben egyetlen protokoll definiált, akkor az összes rétegben megtalálható protokollok összességét protokollkészletnek (protocol stack) nevezik. [1]

1.2. OSI hivatkozási modell

Az OSI egy betűszó – az angol Open System Interconnection kifejezés alapján – amely a nyílt rendszerek összekapcsolásának modelljét írja le. A modell a Nemzetközi Szabványügyi Szervezet (International Standards Organization, ISO) ajánlásán alapul,

és ez tekinthető a réteges hálózati szoftverkialakítás során használt protokollok első elfogadott szabványosításának. Az OSI hivatkozási modellhez kapcsolt protokollokat manapság már nem használják, azonban maga a modell egy elég általános leírása a hálózati rétegek kialakítási módjainak, így igen jó alapnak számít a tématerületen. [1]

Az OSI hivatkozási modellben hét réteg definiált, amelyek az 1-1. ábrán láthatók. Az OSI hivatkozási modell szigorú értelemben nem hálózati architektúra, mert nem definiálja a rétegekben használt protokollokat (régen is különállóan kezelték a modellhez kapcsolt protokollokat), csak a rétegek feladatait adja meg. Éppen ez az erőssége, ami miatt máig is alapelv maradt.



1-1. ábra: Az OSI modell rétegei [1]

A réteges szoftverelrendezésben a hálózatra kapcsolódó eszközök különböző szintekig implementálhatják a hálózati szoftvert. Kommunikációkor azonban mindig az egyik eszköz adott rétege folytat kommunikációt egy másik eszköz ugyanazon rétegével. Két hálózati eszköz közötti kommunikáció során az információ a szomszédos rétegek között továbbítódik lefelé, esetleg kiegészítve saját vezérlési információkkal, egészen a legalsó rétegig, hiszen ez a réteg felelős a fizikai továbbításáért.

1.3. Adatkapcsolati réteg feladatai

Az ISO-OSI hivatkozási modell szerint az adatkapcsolati réteg alulról a második réteg. Közvetlenül a fizikai réteg felett, és a hálózati réteg alatt helyezkedik el. Ennek megfelelően a fizikai réteg szolgáltatásait fel tudja használni, így az információ fizikai jellé történő átalakítása és átküldésének lehetősége már használható a réteg számára.

Az adatkapcsolati réteg (data link layer) legfontosabb feladata a fizikai átviteli rendszer hibázásainak elfedése, és egy megbízható, hatékony adategységek átvitelére alkalmas kommunikáció megvalósítása két szomszédos hálózati eszköz között, így egy hibamentesnek látszódó, stabil, szomszédos eszközök közötti kapcsolatot nyújt a felsőbb rétegek felé. [6]

A legfőbb megoldandó kérdés az átviendő információk olyan adatkeretekbe foglalása, amely struktúrában hatékony és biztos átvitelt valósít meg a kapcsolások (switching) segítségével. További fontos feladata a torlódások elkerülése forgalomszabályozással. Szigorúan véve a megosztott csatornákhoz való hozzáférés kezelése is ide tartozik, azonban ezt a feladatot megvalósító részét a rétegnek gyakran külön, a közeg-hozzáférési alrétegnek nevezik. [6]

1.4. Kapcsoló (Switch)

A kapcsoló az adatkapcsolati rétegben működik, és az ott elérhető információkat használja a feladatának elvégzéséhez, ennek következtében mindenképp digitális eszköz. Általában igen nagyszámú kapcsolattal rendelkezik, így a hálózati forgalom irányításának hatékony eszköze. Alapvető funkciója, hogy lehetővé tegye az összes kapcsolata között a szelektív kommunikációt, azaz biztosítja bármelyik irányban az üzenetek célzott továbbítását. [1] [7]

Fizikai kialakítását tekintve egy kapcsoló eszköz alapja a nagysebességű kapcsolómátrix, amely lehetővé teszi az összes csatlakozó összekapcsolását, illetve az összeköttetések bontását is. A felépítése részben hasonlít az alsóbb, fizikai rétegben működő elosztóra, így a legtöbb kapcsolóban van elosztó jellegű üzemmód is, amikor a bemeneti üzeneteket minden csatlakozójukra továbbítják elárasztásos jelleggel. A bontható kapcsolatok miatt azonban lehetséges olyan üzemmód is, amikor csak a vevő felé eső célcsatlakozójára továbbítja az adatot. Utóbbi esetben lehetséges az is, hogy egy időben több üzenetet is továbbítson a kapcsoló különböző célcsatlakozókra. Az első

működési módot válogatás nélküli üzemmódnak, a másodikat válogatásos üzemmódnak nevezik. Leggyakrabban a második, válogatásos üzemmódban működnek a kapcsolók, hiszen rendkívül előnyös tulajdonságokkal bír ez a mód. [10]

Magára a kapcsoló eszközre vonatkoznak megkötések is. A kapcsolóhoz csatlakozó sok eszköz között egy időben több kommunikáció is folyhat, és előfordulhat, hogy egyszerre többen is egy adott céleszköznek küldenének üzenetet. Emiatt az eszközben szükséges puffermemória alkalmazása, hogy ne vesszen el adat, azaz minden bejövő adatot tudjon fogadni, és tárolni addig, amíg a célcsatlakozó szabaddá nem válik. A csatlakozások kontrollálhatósága és flexibilitása érdekében a kapcsolónak képesnek kell lennie válogatásos és válogatás nélküli üzemmódra is, illetve a hibás üzenetek megakasztására, kiszűrésére is. [10]

A kapcsoló válogatás nélküli üzemmód esetén egyszerűsíti a végpontok bekapcsolódását a hálózatba, illetve segíti a kábelhibák lokalizálását. A válogatásos működési mód választása esetén további előnyöket biztosít a kapcsoló. Egyrészt megfelelő pufferméret esetén garantálható, hogy egy időben több keret továbbítására is képes legyen a kapcsoló, így több állomás között folyhat egyszerre a kommunikáció, azaz nőhet a helyi hálózat kapacitása. Másrészt ilyenkor a kapcsolóhoz csatlakozó eszközök között megfelelő pufferelés és duplex kapcsolat alkalmazásával elérhető, hogy ne alakuljon ki ütközés a kommunikációk során, azaz a kapcsolóhoz csatlakozó végpontok különböző ütközési tartományokba tartozzanak. Harmadrészt biztonságosabb lehet a hálózat, hiszen ilyenkor a kapcsoló az adatokat mindig csak a címzett felé továbbítja, így egy idegen eszközzel nem lehetséges az adatforgalom lehallgatása. A biztonságot tovább növeli, hogy egy meghibásodott vagy idegen eszköz érvénytelen üzeneteit képes lehet blokkolni, és így nem engedi a hiba továbbterjedését a hálózatban.

Összegezve a kapcsolók általában képesek adatszórással minden csatlakozójukra továbbítani az üzeneteket, hasonlóan, mint az elosztók. Emellett lehetséges célzott továbbítás is, amely több előnnyel is bír, amelyek közül a biztonságosabb működés a legfontosabb mind a titkosság, mind a hibaterjedés megakadályozása szempontjából.

2. Kapcsolási algoritmusok alapjai

Az adatkapcsolati rétegben működő hálózati eszköz a kapcsoló és a híd. A két eszköz alapvető működési viszonyait tekintve igen hasonlít egymásra, és mind a kettő kapcsolási algoritmussal valósítja meg az adatkapcsolati réteg feladatait.

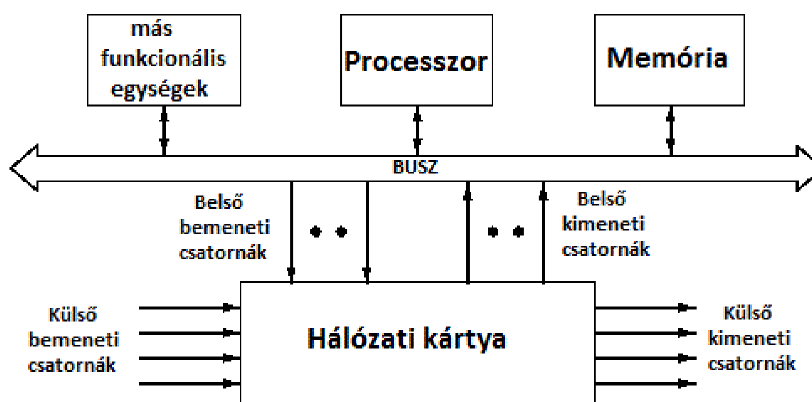
Ebben a fejezetben röviden ismertetem a kapcsolási algoritmusok alapvető vizsgálati szempontjait, illetve bemutatom a módszerek és az algoritmusok csoportosítását az alapvető működési szempontjuk alapján.

2.1. Kapcsolási alapfogalmak

A részletes vizsgálatok előtt szükséges néhány alapfogalom bevezetése és pontos definiálása, hogy a későbbiekben könnyebben megérthetőek, és pontosabban vizsgálhatóak legyenek a kapcsolási algoritmusok. [2]

A feldolgozó egység egy hálózati eszközben található szokásos busz architektúrájú, adatfeldolgozási célú számítógépes rendszer, amelynek busza kiegészül egy hálózati kártyával is. Egy ilyen hálózati eszköz blokkvázlata látható a 2-1. ábrán.

A külső csatorna az az adatút, amely a különböző hálózati eszközöket, azaz a kapcsolókat és a végpontokat köti össze, míg a belső csatorna a feldolgozó egység busza és a feldolgozó egység hálózati kártyája közötti fizikai interfész. Ezek a csatornák is fel lettek tüntetve az alábbi ábrán. A továbbiakban, ha nincs külön kiemelve, akkor csatorna alatt a külső csatornát kell érteni.



2-1. ábra: Hálózati feldolgozó egység általános felépítése [2]

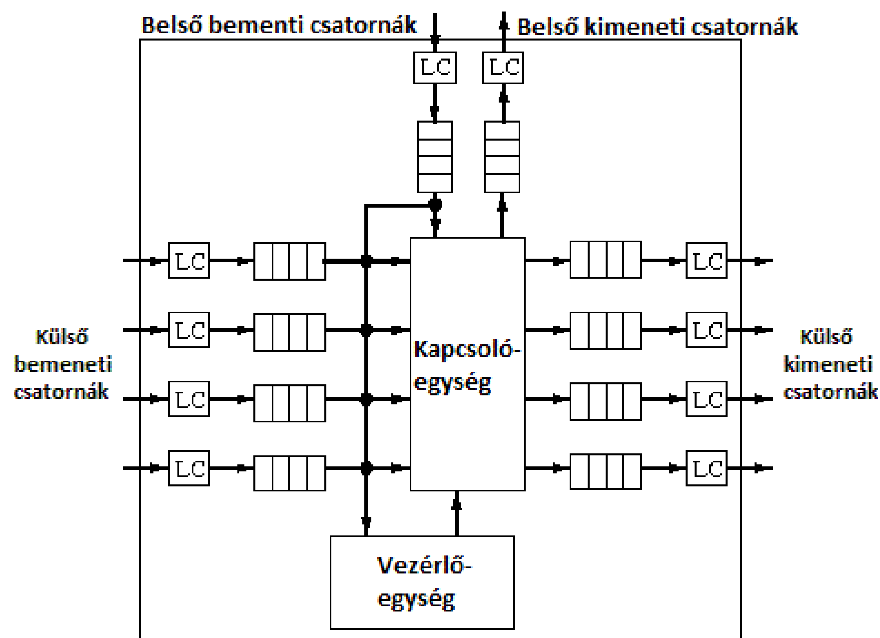
A csatornán keresztül folyhatnak át a kommunikációs adatok, amelynek része a ki-, illetve bemeneti puffermemóriák, a fizikai meghajtók, és a fizikai adatvonalak is.

A puffermemória egy FIFO memória, amely egy vagy több kommunikációs adategység tárolására szolgál. Természetesen bonyolultabb esetben az üzeneteknél lehetséges prioritás alkalmazása is, de a valóságban gyakran elegendő a FIFO viselkedés. Puffermemóriát alkalmazhatnak a hálózati feldolgozóegység bemeneti külső csatornáiban, kimeneti külső csatornáiban, vagy mind a két helyen is.

A fizikai meghajtó (Link Controller, LC) implementálja a kommunikációs protokoll előírásait két hálózati eszköz között. Lényegében ez dekódolja a vett fizika jeleket a feldolgozóegység számára értelmezhető információvá, illetve kódolja a kimeneti információt fizikai jellé.

A hálózati kapcsolóeszközön belül a kapcsolóegység biztosítja a külső ki- és bemenetek közötti kapcsolatok kialakításának és bontásának lehetőségét. Felépítésileg lehet egy teljes értékű keresztkapcsoló, amely az összes külső bemenetre biztosítja bármely külső kimenetre történő kapcsolásának lehetőségét, vagy néhány parciális kapcsoló, amelyek csak néhány külső be- és kimenetet kapcsolását teszi lehetővé, de megfelelő vezérléssel itt is elérhető az összes kapcsolási lehetőség.

A csatorna, a puffermemória, a fizikai meghajtó, a kapcsolóegység fogalmak megértését segíti a 2-2. ábra is.



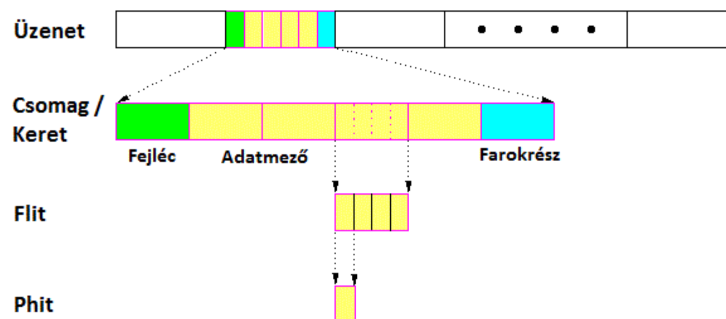
2-2. ábra: Hálózati kapcsoló vázlatos felépítése [2]

A kapcsolási módszer dönti el, hogy a hálózati erőforrások mikor és hogyan allokálódnak a kommunikációs adatátvitelek számára. Ez a mechanizmus definiálja, hogy hogyan történik az adatok elvétele a külső bemeneti csatornákról, és hogyan továbbítódnak a külső kimeneti csatornákra.

Az üzenet szigorúan véve az az információ, amelyet két különbönöző hálózati végponton futó alkalmazás ki akar cserélni egymással. A köznyelvi használatban valójában ez általánosabb fogalom, és egyúttal használatos a csomagok, illetve keretek helyett is. Lényegében az üzenet átvitelére jön létre a kommunikáció a hálózaton keresztül. A csomag vagy keret az a legkisebb információegység, amelyet a hálózat képes kezelni. Ennek méretét és felépítését általában a hálózati protokoll definiálja. Felépítésileg egy fejlécre, egy adatmezőre és egy farokrészre osztható.

A flit egy kisméretű információegység az adatkapcsolati rétegben, amely egy vagy több hálózati szó méretű lehet, és ennek megfelelően több hálózati ciklusba telik egy flit információ átvitele. A phit a legkisebb információegység a fizikai rétegben, amely egyetlen ciklusban átküldhető két hálózati eszköz fizikai rétege között. Fontos elválasztani a bittől, azaz információegység alapjától. Egyszerű soros kommunikáció esetén a phit és a bit egyezhet, azonban például egy egyszerű párhuzamos kommunikáció esetén egy phit megegyezik a párhuzamosan átvitt bitek számával.

Az információs egységek megértését segíti a 2-3. ábra.



2-3. ábra: Hálózati információs egységek [2]

2.2. Kapcsolási módszerek csoportosítása

A kapcsolási módszereket működésük alapelve szerint két nagyobb csoportba szokás osztani: a vonalkapcsolásra és a csomagkapcsolásra. A két csoport megkülönböztetésének alapja a kommunikációhoz szükséges hálózati erőforrások lefoglalása, illetve a kapcsolási irányról történő döntés időpontja.

2.2.1. Vonalkapcsolás (Circuit switching)

Vonalkapcsolás esetén a kommunikációs 3 fázisra osztható. Először a hálózati kapcsolóeszközök keresnek egy fizikai útvonalat, amely összeköti az adót a vevővel, közben a két végpont közötti áramkör kialakításához szükséges erőforrásokat is lefoglalják. Szükséges erőforrásoknak tekinthetők mind az útvonalon található kapcsolóelemek külső ki- és bemenetei, mind az egyes eszközöket összekötő fizikai adatvonalak is. Ezen erőforrások fontos tulajdonsága az elérhető kommunikációs sávszélesség, illetve sebesség. Másodszor, ha sikerült a megfelelő tulajdonsággal bíró áramkört kialakítani a két végpont között, akkor megindulhat a tényleges kommunikáció az adó és a vevő között. Amennyiben a hálózat terheltsége miatt nem lehetséges a szükséges erőforrások lefoglalása, nem áll rendelkezésre a kívánt sávszélesség, akkor nem jön létre az áramkör, és a kezdeményezés során az esetlegesen már lefoglalt erőforrások azonnal felszabadulnak a hálózaton más kommunikációk számára. Végül, a kapcsolók a kommunikáció befejeztével a lefoglalt erőforrásokat felszabadítják, így más kommunikációk számára is használhatóvá válik a hálózat. [2]

A vonalkapcsolás esetén elképzelhető lehetne olyan eset, hogy az áramkör lefoglalása után a kommunikáció egy folyamatos információáramlással, egy egyszerű bitfolyammal valósulna meg, azonban ez a hálózati adatvonalak hibái és a külső zavaró tényezők hatásai miatt nem lenne hatékony és megbízható. Az esetlegesen fellépő átviteli hibák kiszűrése és javítása érdekében érdemes megtartani az adatok különálló keretekben történő átküldésének mechanizmusát, bár a bitek átvitele folyamatos marad, többletráfordítást csak a fejléc és farokrész bitjei jelentenek. Vonalkapcsolás esetén azonban az egyes keretek mindig egyetlen útvonalon át haladhatnak keresztül a hálózaton, az előre lefoglalt áramkör útvonalán. Tehát az egyes hálózati eszközökben a kapcsolások irányáról már az áramkör lefoglalásának idejében döntés születik, és a kommunikáció alatt az egyes keretek egymás után ugyanazon az útvonalon terjednek.

Legjobb példa a vonalkapcsolásra a hagyományos telefonközpont. Telefonálás esetén először tárcsázni kell, azaz meg kell adni a hívott fél címét, ezután a telefonközpont megpróbál egy áramkört létesíteni a hívott fél felé, és ha ez sikerül, csak aztán ad vonalat, és csak ezután beszélhetünk a másik féllel.

Az előre felépített áramkör miatt a vonalkapcsolás mechanizmusa biztosítja, hogy ne fordulhasson elő torlódás a hálózatban. Ráadásul a vonalkapcsolás azt is

biztosítja, hogy a hálózat túlterhelése esetén is, a kapacitások alapján maximálisan kialakítható kommunikációs vonalak megvalósulhatnak, amelyeken az adatátvitel is problémamentesen megtörténhet, és csak az e felettiek blokkolódnak. Mivel a csomagok csak egyetlen útvonalon haladhatnak, emiatt garantált az is, hogy a keretek ugyanabban a sorrendben érkeznek meg a vevőhöz, mint ahogy az adó adta őket. [2]

A vonalkapcsolás nagy hátránya, hogy igen érzékeny a hálózati kapcsoló csomópontok hibáira. Hiszen ha egy kapcsoló hiba miatt működésképtelenné válik, akkor az összes rajta átmenő áramkör használhatatlanná válik, és lehetetlenné válik azokon az adattovábbítás.

2.2.2. Csomagkapcsolás (Packet switching)

Csomagkapcsolás esetén nem szükséges előre keresni és kialakítani egy teljes kapcsolatot az adótól a vevőig a kommunikáció megkezdése előtt. Emiatt a kommunikációhoz szükséges ki- és bemenetek, adatvezetékek nem foglalódnak le a kapcsolat számára. Tehát a végpontok között nem jön létre dedikált vonal. Ehelyett a csomagokat folyamatosan továbbítják egymás között a hálózati eszközök, és a csomagok azonnal átküldésre kerülnek egy adatvonalon, amint rendelkezésre állnak a kapcsolóban. Ennek megfelelően az átvitelhez szükséges erőforrásokat dinamikusan allokálja a hálózat az átküldés idején belül is, és csak a szomszédos eszközhöz történő átjutás idejére allokálódik az éppen szükséges erőforrás a kommunikáció számára, utána pedig azonnal fel is szabadul. [2]

Fontos, hogy csomagkapcsolás esetén a kommunikációs adatok nem jellemezhetők folyamként, hanem mindenképpen keretekre, csomagokra tördelve reprezentálhatók. A csomagkapcsolás esetén nincs előre kialakított útvonal, helyette kommunikációs időben történik mindig a csomagok útvonalának kialakítása a hálózaton keresztül. Ennek megfelelően az adótól elinduló üzenet útvonaláról a küldési időben folyamatosan döntenek a hálózati kapcsolók. [2]

Egy szemléltető példa a csomagkapcsolásra az internethálózat. Az internetes böngészéskor a le- és feltöltött információk véges méretű adatcsomagok, és ezek átjuttatása a hálózat feladata.

Az erőforrások lefoglalásának módszeréből következik, hogy a hálózaton előfordulhatnak torlódások, mert előfordulhat olyan helyzet, hogy egy kapcsolóhoz túl sok csomag érkezik egyszerre. Ilyenkor ezeknek várakozniuk kell egymásra, sőt akár

holtpont jellegű helyzetek is előfordulhatnak. Ezek miatt a csomagkapcsolás esetén nem biztosított az, hogy a csomagok adott véges időn belül mindenképp eljutnak a céljukig, valamint lehetséges, hogy a hálózat túlterhelődik, azaz a kapcsolóelemek nem biztos, hogy képesek fogadni újabb csomagokat, akár adatvesztés is előfordulhat. Tehát a jelterjedésből fakadó késleltetésen felül a döntési késleltetés, illetve a várakozási késleltetés is megjelenik a kommunikáció során. [2]

A hálózati erőforrásokat, elérhető sebességet, kapacitást optimálisan kezeli az adatkapcsolási mechanizmus, hiszen nem szükséges a kommunikációhoz lefoglalni erőforrást, hanem a terjedéssel azonos időben csak az átjutás idejéig foglalódnak le. [10]

A csomagkapcsolás nagy előnye, hogy kevésbé érzékeny a hálózati kapcsoló csomópontok hibáira. Hiszen ha egy kapcsoló hiba miatt működésképtelenné válik, akkor az összes folyamatban lévő kommunikáció csomagjainak lehetséges más útvonalat keresni, amely elkerüli a hibás csomópontot.

2.3. Kommunikációs módszerek csoportosítása

A kommunikációs feladatokat legegyszerűbben a kommunikációban résztvevők száma alapján lehetséges csoportosítani. Ezek alapján megkülönböztetünk pont-pont típusú kommunikációt, illetve adatszóró típusú kommunikációt. [2]

A két csoport hardveres felépítése igen eltérő lehet. A pont-pont hálózatok esetén végpont párok vannak összekötve, és mindig csak két gép között történhet kommunikáció. De itt is megvalósítható az, hogy akár több különböző, vagy akár az összes többi végpontnak üzenetet küldhessen bármelyik eszköz, megfelelő kapcsolások és több üzenetküldési ciklus segítségével. Az adatszóró hálózatok esetén van egy központi csatorna, amelyre minden végpont felcsatlakozik, és így mindenki értesülhet az összes, a csatornán haladó üzenetről, és csak az üzenet címe alapján lehet eldönteni, hogy valójában kinek szól. De ebben a struktúrában is lehetséges kapcsolóelemek vagy szűrőeszközök beillesztésével és megfelelő vezérlésével elérni, hogy az üzenet valóban csak a céleszközhöz jusson el.

A kapcsolási algoritmusokat a pont-pont típusú hálózatok esetén a legszemléletesebb bemutatni. Az adatszóró hálózatokban történő kapcsolás is pont-pont hálózatokban kialakított kapcsolási módszerekre épül nagyvonalakban, és csak néhány kisebb módosítást alkalmaznak adatszórás esetén.

3. Pont-pont hálózatok kapcsolási algoritmusai

A legegyszerűbb kommunikációs feladat a két végpont közötti kommunikáció megvalósítása, emiatt ebben az esetben a legszemléletesebb az alapvető kapcsolási algoritmusok bemutatása és vizsgálata, hiszen itt lehet könnyen átlátni a hasonlóságokat és különbségeket.

Ebben a fejezetben először ismertetem az algoritmusok elemzésének szempontjait, és az algoritmusok tulajdonságait leíró mérőszámokat is bevezetem. Majd részletesen bemutatom a manapság alkalmazott kapcsolási technikákat. Végül pedig egy összehasonlítást is adok a legfontosabb tulajdonságok alapján.

3.1. Mérőszámok és elemzési szempontok

A vizsgálatok előtt fontos meghatározni azokat a szempontokat, amelyek mentén célszerű egy kapcsolási algoritmust vagy kapcsolási módszert elemezni. A szempontok meghatározásához pedig szükséges néhány olyan mutató megismerése, amelyek alapján lehetséges az egyes algoritmusok teljesítményének a mérése. A szempontok meghatározása mellett pedig néhány jelölést is bevezetek, amelyeket a későbbiekben használni fogok.

3.1.1. Teljesítménymutatók

Csatornaszélességnek nevezik azt a bitszámot, amelyet a külső csatornáin a hálózati eszköz fizikai rétege azonos időben, párhuzamosan képes kiküldeni a szomszédos eszköznek. Valójában ez a phit értéke bitekben mérve. A későbbiekben a jele w.

A csatornasebesség a hálózaton elérhető maximális sebességráta, amellyel egyetlen fizikai adatvonal képes az adatátvitelre. A mértékegysége bit/másodperc, és a jelölése a továbbiakban μ. A csatorna sávszélesség pedig egyetlen külső csatornán, azaz két szomszédos hálózati eszköz között elérhető maximális adatátviteli sebesség, amelynek jele B. A csatorna sávszélessége kiszámítható a csatornasebességből kétféle módon is. A csatorna sávszélesség értéke megegyezik a csatornasebességgel ($B = \mu$), ha phit/másodperc mértékegységben számoljuk ki. Vagy a csatornasebesség és a csatornaszélesség szorzata ($B = \mu * w$), ha bit/másodperc mértékegységben számolunk.

A csatornakésleltetés az az időtartam, amíg egységnyi phit mértékű információ terjed a szomszédos hálózati eszközök között. A jele t_m, mértékegysége pedig phit/másodperc. Természetesen több phit információ terjedésének az ideje arányos a phitszámmal, hiszen csak egymás utáni időpillanatokban képes a hálózati eszköz továbbítani a csatornán. [2]

A döntési késleltetés időtartama alatt a hálózati eszközbe beérkezett adatról eldönti a kapcsoló, hogy melyik kimenetére továbbítsa azt. A mértékegysége másodperc, és a jele t_r. Ez az időtartam szigorúan a döntési algoritmus futási ideje, maga a kapcsolás ez idő alatt nem hajtódik végre. [2]

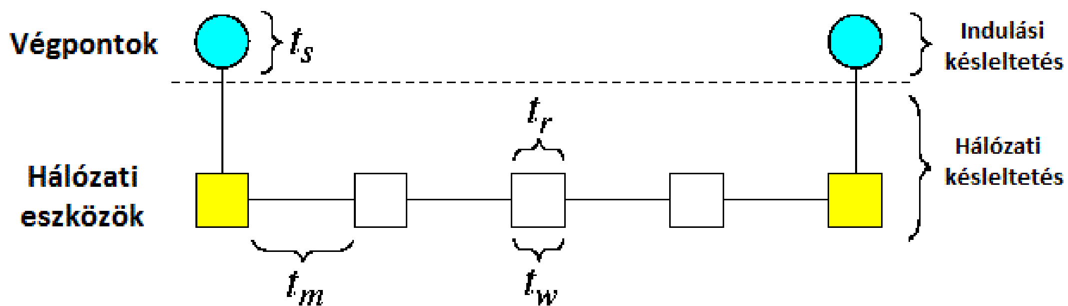
A kapcsolási késleltetés az az időtartam, amíg egységnyi phit hosszúságú információ terjed a hálózati eszközön belül. A jele t_w, és a mértékegysége pedig phit/másodperc. Fontos kiemelni, hogy ez szigorúan csak a terjedési időt foglalja magába, magát a döntés idejét nem, illetve a torlódások miatti késlekedést sem. Az eszközön belül ugyanarról a külső bemenetről ugyanarra a külső kimenetre csak egymás után terjeszthetők az adatok, így több phit információ terjedésének ideje arányos a phitszámmal. [2]

Az indulási késleltetés az egyes adatcsomagok adóban történő kialakításához, illetve vevőben történő lebontásához szükséges időtartam, amelynek jele t_s, és másodperc a mértékegysége. Tartalmazza a keretezés kialakításának, illetve lebontásának idejét, az adatok esetleges másolását a puffer és a feldolgozóegység memóriája közötti, illetve azok érvényességének vizsgálatát is. Az indulási késleltetés statikus időtartam, amely nem függ a hálózati adatforgalomtól, illetve a hálózati eszközök távolságától, hanem kizárólag a keretek hosszától függ. Emiatt habár késleltetésként megjelenik a hálózati kommunikációban, mégsem egy jól használható mérőszám a hálózati eszközök vizsgálatához. [2]

Egy hálózat jellemzésére a legmegfelelőbb mérőszám a hálózati késleltetés. Ez az időtartam az üzenet vagy csomag első fejléc phitjének hálózatba történő belépésétől az utolsó farokrész phitjének a hálózatból történő kilépéséig tartó idő. Fontos megjegyezni, hogy ez az időtartam erőteljesen függ attól, hogy a hálózatban az adó és a forrás között milyen hosszú útvonalon, azaz hány hálózati eszközön át kell az üzenetnek terjednie. Későbbiekben ennek a számnak a jele d. A hálózati késleltetés két komponensből tevődik össze. Az első a blokkolásmentes csomagátviteli idő, amely az ideális esetben történő csomagterjedési idő, azaz amikor minden hálózati erőforrás

azonnal a rendelkezésre áll, és nem alakul ki torlódás, illetve emiatt várakozás. A második a blokkolási idő, ami tartalmazza az összes üzenet vagy csomag által elszenvedett késleltetési időt, ami a torlódás és a nem azonnal rendelkezésre álló erőforrások miatt keletkezik. Mind a két összetevő fontos része az időzítések vizsgálatának, de szigorúan véve az algoritmus jellemzésére a blokkolásmentes csomagátviteli idő a legalkalmasabb. [2]

A különböző késleltetési időket összefoglalja a 3-1. ábra.



3-1. ábra: Terjedési késleltetések [2]

3.1.2. Elemzési szempontok és egyszerűsítések

A különböző kapcsolási algoritmusok működésének leírása mellett, a vizsgálatok alapszempontja a hálózati késleltetési idő, hiszen ez az a paraméter, amely a leginkább függ a hálózati eszközökben futtatott algoritmustól. Ennek megfelelően minden algoritmus esetén e paraméter elméleti értékét kiszámolva lehetséges egymással összehasonlítani az egyes algoritmusok teljesítményét. [2]

A vizsgálódások nem veszik figyelembe az indulási késleltetést, illetve zérus értékként jelenik meg a számításokban. Ez nem okoz torzítást a jellemzések és az összehasonlítások során, hiszen statikus paraméter, azaz nem függ az alkalmazott kapcsolási eljárástól, emiatt minden algoritmus esetén azonos értékű lenne. Ennek megfelelően szigorúan az üzenetek és csomagok hálózaton történő viselkedések alapján történik az algoritmusok jellemzése. Az algoritmusok bemutatása után röviden ismertetésre kerül a teljes blokkolásmentes hálózati késleltetés és közelítése is, amelynél szükséges figyelembe venni az indulási késleltetést is.

A hálózati késleltetés értékét alapvetően egyetlen, a hálózat által kezelt egységre célszerű meghatározni, emiatt a csomagkapcsolási módszerek esetén egy csomagra, a vonalkapcsolás esetén pedig egy üzenetre lehetséges megvizsgálni. Valójában a

csomagkapcsolás esetén a csomagok késleltetése megegyezik az egész üzenet késleltetésével is, mert az üzenet első és utolsó csomagjának késleltetése is azonos értékű, és nem akkumulálódik a csomagok késleltetéseinek hatása az üzenet terjedéskor, így az egész üzenet késleltetése is ugyanekkora lesz.

A hálózati késleltetés értékét paraméteresen a hálózati eszközök száma (d), és azok részeinek időigénye (csatornakésleltetés (t_m), döntési késleltetés (t_r), kapcsolási késleltetés (t_w)) alapján lehetséges meghatározni.

Összefoglalva: a vizsgálódás célja, hogy a hálózati késleltetés értékét egy zárt formula segítségével meg lehessen adni minden kapcsolási algoritmus esetében egyetlen hálózat által kezelt egységre, ugyanazokat a paramétereket felhasználva.

Fontos kiemelni, hogy a kapcsolási algoritmusok elméleti elemzése a hálózati késleltetésnek csak a blokkolásmentes csomagátviteli idejét vizsgálja. Az ismertetések során mindig azzal a feltételezéssel lehet élni, hogy az üzenet vagy csomag azonnal megkapja a továbbjutáshoz szükséges erőforrást. Tehát a többi csomag által generált forgalomból, illetve az esetleges túlterhelésekből származó torlódásokból származó hatásokra nem tér ki a vizsgálat. Úgy is lehet fogalmazni, hogy a legjobb hálózati késleltetési idők alapján történik a vizsgálódás.

Egy további egyszerűsítés még a vizsgálódások során, hogy az egyes csomagok fejlécét és a farokrészét is zérusértékűnek vesszük, azaz elhanyagoljuk a keretezés hatását. Ez egy kényelmi egyszerűsítés az elméleti számolások során, azonban nem okoz torzítást, hiszen a fejléc vagy a farokrész hossza a kommunikációs protokollban definiált, és nem a kapcsolási algoritmustól függ.

3.2. Pont-pont kapcsolási algoritmusok elemzése

Az előbb felsorolt szempontok szerint, a megemlített egyszerűsítéseket kihasználva már lehetséges bemutatni és elemezni a pont-pont típusú hálózatokban kifejlesztett kapcsolási algoritmusokat.

A gyakorlatban a vonalkapcsolási módszer megvalósítására egyetlen kapcsolási algoritmust alkalmaznak, amelyet ugyanúgy vonalkapcsolásnak neveztek el. Csomagkapcsolási módszerekre három algoritmus is használatos, ezek a „tárol és továbbít”, az „átvágó” és a „féreglyuk” kapcsolások.

3.2.1. Vonalkapcsolás (Circuit Switching, CS)

A vonalkapcsolás, vagy más néven az áramkörkapcsolás, a korábban ismertetett kapcsolási módszernek megfelelően három fázisra osztható. Az első fázis az áramkör kialakítása, a második a tényleges adatcsere, a harmadik pedig az áramkör lebontása. Ezek a fázisok lényegében megegyeznek az összeköttetés alapú kommunikáció fázisaival, és emiatt a vonalkapcsolás mindig összeköttetés alapú hálózatot eredményez.

Az áramkör kialakításának fázisában az adó először egy kezdeményezőjelet küld a hálózatba, ami általában egy kisméretű üzenet. Ez mindenképp tartalmazza a vevő címét, és mellette esetleg vezérlési információk is lehetnek benne. Ez a kezdeményezőjel a hálózaton belül eljut a vevőig különböző kapcsolóeszközökön keresztül, és terjedés közben lefoglalódnak az átjutáshoz felhasznált erőforrások (fizikai adatvonalak, kapcsolók ki- és bemenetei) a kialakítandó vonal számára, amelyen később majd az üzenetek terjedhetnek. Ha a jel eljutott a vevőig, akkor kialakítható volt a vonal, és már a szükséges erőforrások is lefoglalásra kerültek. Tehát a vonalkapcsolás módszerének megfelelően a tényleges kommunikáció elindítása előtt lefoglalódik az összes szükséges erőforrás. A kialakítás fázisának végén egy visszaigazoló jel is visszaküldésre kerül a vevőtől az adó felé. Így az adó tudomására jut, hogy lehetséges volt kiépíteni az áramkört, és megkezdheti a tényleges kommunikációt. [2]

A tényleges kommunikáció fázisában az adó már – a lefoglalt erőforrások alapján kialakítható – teljes sebességgel, illetve sávszélességgel kezdi küldeni az üzenetét. Az üzenet minden adata könnyen és kis késleltetéssel eljut a vevőig, hiszen a kommunikáció teljes ideje alatt lefoglalva maradnak a vonal erőforrásai, így nem alakulhat ki torlódás, és nem lehetséges az, hogy az adatoknak várakozniuk kell. [2]

Miután az üzenet utolsó részlete is átért a vevőhöz, akkor a lebontási fázisban a vonalhoz rendelt erőforrások felszabadulhatnak. A felszabadítás igényét jelezheti egy utolsóként átküldött speciális karakter is, vagy kezdeményezheti külön üzenetben a vevő is.

A legjobb hálózati késleltetés képlete [2]:

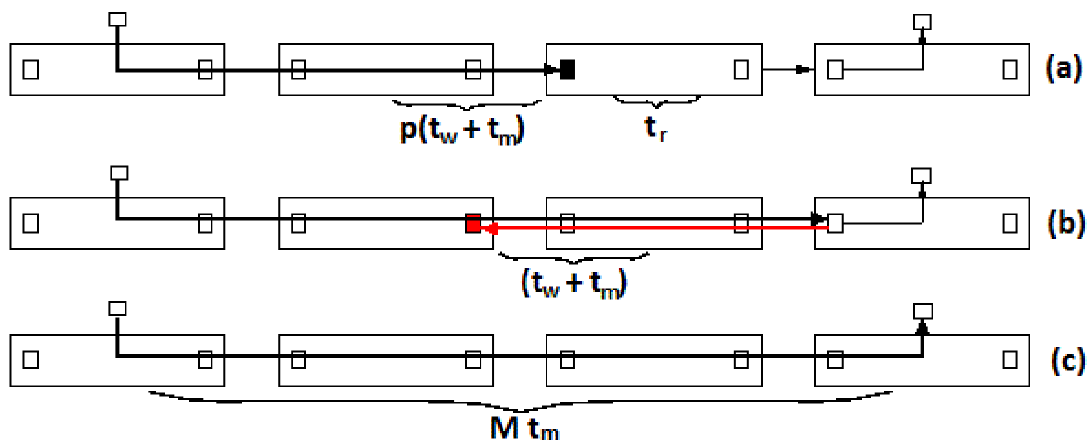
$$t_{CS} = d(t_r + (p+1)(t_w + t_m)) + Mt_m$$

Az egyenlet egy M phit hosszúságú üzenet d kapcsolón keresztüli távolságra küldendő terjedési idejét írja le, ahol t_m a csatornakésleltetés, t_r a döntési késleltetés és t_w

a kapcsolási késleltetés. Ezen felül az áramkör kialakítása *p* phit méretű kezdeményezőjellel történik.

Az egyenlet alapján is jól láthatók az egyes fázisok időigényei. Az áramkör kialakításakor a kezdeményező jel átterjesztéséhez $d(t_r+p(t_w+t_d))$ idő szükséges. A visszaigazoló jel 1 phit méretű, és $d(t_w+t_m)$ idő alatt érkezik meg. A kapcsolat felépítése után az adatáramlás maximális sebességgel történhet, így az *M* phit hosszúságú üzenetnek Mt_m átviteli idő szükséges. A vonal lebontása pedig automatikusan történik, külön időjárulék nélkül, az utolsó információ átvitele után.

A vonalkapcsolás működésének és időzítéseinek szemléltetése látható az alábbi, 3-2. ábrán is.



3-2. ábra: Vonalkapcsolás működése [2]

Az ábrán a téglalapok jelölik az egyes kapcsolóeszközöket, amelyeken keresztülhalad az üzenet. A kapcsolókon belüli kis négyzetek pedig a be- és kimeneti puffermemóriát szimbolizálják. Természetesen egy kapcsolóeszköz több ki- és bemenettel rendelkezik, azonban az ábrákon ezek nem lettek külön feltüntetve. Az összes ábrán a bal oldali kapcsolóhoz csatlakozik az adó, a jobb oldalihoz pedig a vevő. Az ábra (a) része a kezdeményezőjel terjedését, a (b) része a visszaigazolást, a (c) része pedig az üzenet továbbítását szemlélteti úgy, hogy közben az egyes fázisokat meghatározó késleltetések jelölve vannak. [2]

Fontos kiemelni, hogy amíg a kezdeményezőjelet a közbenső hálózati eszközök pufferelik, addig a tényleges kommunikációs adatokat már nem. A kezdeményezőjelet szükséges pufferelni, mert lehetséges, hogy a hálózat terheltsége miatt nem lehet kialakítani teljes hosszában a vonalat, és ilyenkor a kezdeményezőjel várakozásra

kényszerülhet. Viszont a vonal kialakítása után nem fordulhat elő ütközés vagy várakozási kényszer. A bejövő üzenetrészeket azonnal lehetséges továbbítani az eszköz kimenetén is, emiatt nem szükséges a pufferelés, amely csak lassítaná az átvitelt. Átvitelkor a hálózat egésze egy egyszerű vezetéknek látszódik az adó és a vevő között.

Az előzetesen lefoglalt erőforrások miatt elméletben szükségtelen az üzenet csomagokká tördelése, és egyszerű bitfolyamként átmehet az egész üzenet a hálózaton. A valóságban azonban mégis használják a keretezést az átviteli hibák ellen. Ilyenkor természetesen a hálózati késleltetés képlete módosulna, de egyszerűsítésként minden kapcsolás esetén elhanyagoljuk a keretezés hatását.

A legjobb hálózati késleltetési idő képlete alapján belátható, hogy a hálózati késleltetés arányos az üzenet méretének és a hálózaton megtett útvonal távolságának összegével.

A vonalkapcsolást a működéséből adódóan akkor érdemes használni, ha viszonylag ritkán nagyobb mennyiségű adatátvitel szükséges, vagy ha az üzenetek küldésekor nagyon pontos és rövid időzítéseket kell betartani. Rövid üzenetek esetén nem célszerű a vonal felépítése és lebontás két üzenet között, mert ez számottevő holtidőt eredményezne, viszont ilyenkor pazarló módon a küldési szünetekben is folyamatosan foglaltak lesznek a vonalhoz szükséges erőforrások.

3.2.2. „Tárol és továbbít” kapcsolás (Store-and-Forward, SF)

Ezt a kapcsolási algoritmust gyakran csomagkapcsolásnak is nevezik, azonban fontos leszögezni, hogy a későbbiekben bemutatott másik két kapcsolási technika is csomagkapcsolás-elvű.

Az üzenetet a hálózatba küldése előtt az adónak fel kell darabolnia a hálózat számára kezelhető, adott méretű adategységekre, és ki kell alakítania a hálózati protokollnak megfelelő keretezést is, mert csomagkapcsolás-elvű. Magán a hálózaton már csak a protokollnak megfelelő csomagok terjedhetnek.

A „tárol és továbbít” algoritmus alapelve, hogy egy-egy csomagot egész egységként kezel, azaz a köztes csomópontbeli kapcsoló eszközök mindig teljes hosszukban fogadják az egész csomag összes adatát, mielőtt továbbküldenék. Ennek megfelelően az adatvezetékeken, ha megkezdődött egy csomag adatainak továbbítása, akkor az mindig teljes egészében megtörténik a csomag utolsó phit információjának

megérkezéséig. Természetesen több csomag esetén kialakulhatnak torlódások a hálózaton, azonban ilyenkor a később érkező csomag küldése addig nem kezdődhet meg, amíg az előző átvitele nem fejeződött be, és a kapcsolóeszköz nem képes fogadni a bemenetén a következőt. Ugrásnak nevezik a „tárol és továbbít" algoritmus alaplépését, amely során egy teljes csomag összes adata az előző kapcsolóeszköz kimeneti pufferéből a következő eszköz bemeneti pufferébe továbbítódik. [2] [5]

Fontos megkötés, hogy az összes hálózati eszköznek képesnek kell lennie egy teljes csomag összes phit információjának tárolására mind a bemeneti, mind pedig a kimeneti puffermemóriájában is. Ennek oka, hogy csomagkapcsolási módszer esetén előfordulhatnak torlódások, és emiatt lehetséges az, hogy a csomag összes adatának várakoznia kell, de ilyen esetben is biztosítania kell az eszközöknek azt, hogy nem veszik el adat a hálózatban, és ez csak ideiglenes tárolással oldható meg. [8] [9]

A kapcsolóeszközök minden egyes csomag irányáról különálló módon, egymástól függetlenül döntenek. Az egyes csomagok útvonalának kiválasztása csak azután kezdődik el, miután a csomag teljes egészében beérkezett a bemeneti puffermemóriába. Tehát a kapcsolóeszközök először mindig fogadják az új csomag összes adatát, utána minden új csomag esetén újra lefuttatják a döntési algoritmust, amelynek eredményeként születik meg a döntés, hogy melyik kimeneti csatlakozóra kerüljön majd az adott csomag, végül pedig átmásolják a csomag teljes tartalmát a választott kimeneti puffermemóriába. [8] [9]

A legjobb hálózati késleltetés képlete [2]:

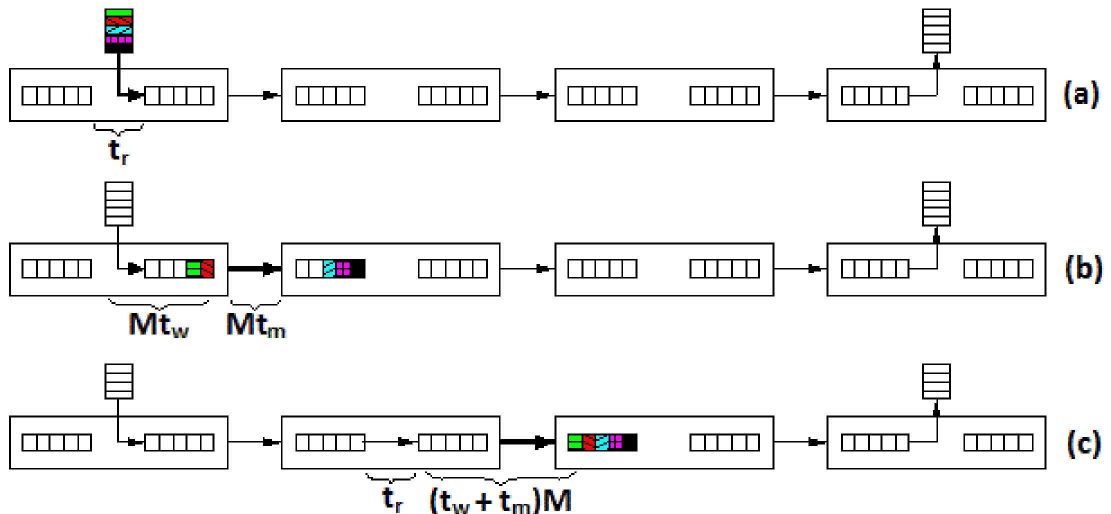
$$t_{SF} = d(t_r + (t_w + t_m)M)$$

Az egyenlet egy összesen M phit hosszúságú csomag, d kapcsolón keresztüli távolságra küldendő üzenet egy csomagjának terjedési idejét írja le, ahol t_m a csatornakésleltetés, t_r a döntési késleltetés és t_w a kapcsolási késleltetés.

Az egyenlet részei is jól mutatják, hogy mi történik az egyes ugrások során. A kapcsolóeszköz mindig a teljes csomag beérkezése után dönt az egész csomag útvonaláról, ez a különálló t_r rész az egyenletben. De a csomag összes phitjét külön tudja csak átmásolni a kapcsolóeszköz a bemenetéről a kimenetére, ehhez t_wM idő szükséges, illetve az egyes phiteket külön lehetséges továbbítani az adatvonalakon a következő kapcsolóhoz, amely t_mM idő. Észre lehet venni, hogy hallgatólagosan a fenti egyenletben nem jelenik meg a csomagok kialakításából származó fejlécek és

farokrészek átviteleinek időjárulékai, de egyszerűsítésként minden kapcsolás esetén elhanyagoljuk a keretezés hatását.

A „tárol és továbbít" kapcsolás működésének és időzítéseinek szemléltetése látható az alábbi, 3-3. ábrán is.



3-3. ábra: „Tárol és továbbít" algoritmus működése [2]

Az ábrán a téglalapok jelölik az egyes kapcsolóeszközöket, amelyeken keresztülhalad az üzenet csomagja. A kapcsolókon belüli kis négyzetek pedig a be- és kimeneti puffermemóriát szimbolizálják. Természetesen egy kapcsolóeszköz több ki- és bemenettel rendelkezik, azonban az ábrákon ezek nem lettek külön feltüntetve. Az összes ábrán a bal oldali kapcsolóhoz csatlakozik az adó és a jobb oldalihoz pedig a vevő. Az ábra (a) része a csomag elindítását és az első irányválasztást, a (b) része az első ugrás közbeni adatmásolás állapotát, a (c) része pedig a második ugrás utáni állapotot szemlélteti úgy, hogy közben az ugrásokat meghatározó késleltetések is jelölve vannak. [2]

Egyes kommunikációs protokollok esetén nagy puffermemóriára lehet szükség minden eszközben, hiszen ebbe a teljes csomagnak bele kell férnie. Ez drágíthatja az eszközöket, vagy lassíthatja a kommunikációt is. De mindenképp azt eredményezi, hogy korlátozott lehet a csomagok mérete. [10]

A legjobb hálózati késleltetési idő képlete alapján belátható, hogy a hálózati késleltetés arányos a csomag méretének és a hálózaton megtett útvonal távolságának szorzatával. Emiatt erőteljesen figyelni kell arra, hogy a kapcsolások során a lehető

legrövidebb útvonalon jusson el a csomag a célhoz, illetve törekedni kell mindemellett arra is, hogy lehetőleg kisméretű csomagok legyenek a hálózaton.

A „tárol és továbbít" algoritmus előnyös akkor, hogyha kisméretű és gyakori üzenetek csomagjai terjednek a hálózaton. Hiszen egy csomag továbbítása során egy időben mindig csak két szomszédos eszköz közötti egyetlen adatátviteli csatornát foglal le az algoritmus a teljes útvonalból.

3.2.3. „Átvágó" kapcsolás (Cut-through, CT)

Ez az egyik legkifinomultabb és legtöbb erőforrást igénylő kapcsolási technika, amelyet a gyakorlatban is használnak. Szokás „átvágó", vagy „átfuttató" kapcsolásnak is nevezni.

Az üzenetet a hálózatba küldése előtt az adónak ugyanúgy fel kell darabolnia a hálózat számára kezelhető adott méretű adategységekre, és ki kell alakítania a hálózati protokollnak megfelelő keretezést is, mint a „tárol és továbbít" algoritmus esetében, hiszen az „átvágó" algoritmus is csomagkapcsolás-elvű.

Az „átvágó" kapcsolás algoritmus alapelve szerint nem szükséges megvárni a teljes csomag összes phitjének megérkezését, hanem a csomag továbbküldése már azelőtt megkezdődhet, mielőtt beérkezne az utolsó adata is a csomópontba. Ezt nevezik átvágásnak. Természetesen az épp beérkező csomag adatainak továbbküldése, átvágása csak akkor történhet meg, ha a csomag irányát már eldöntötte a kapcsolóeszköz, illetve ha éppen szabad a választott kimeneti csatorna. Valóságban az irány eldöntése kevesebb időt vesz igénybe, mint a csomag teljes beérkezése, emiatt megtörténhet az átvágás a kapcsolóeszközön beül, ha szabad a választott kimeneti csatorna. Sőt, ideális esetben, ha több egymás utáni kapcsolóeszközben szabad a választott kimeneti csatorna, akkor ténylegesen kialakulhat olyan helyzet, hogy a csomag egyes részei egy időben, kettőnél több kapcsolóban is tartózkodnak az átvágások miatt. [5]

Az „átvágó" algoritmus esetén a kapcsoló a csomag első néhány phit információját először a bemeneti pufferébe gyűjti, de csak addig, amíg nem áll annyi információ a rendelkezésére, ami alapján a továbbítás irányáról képes dönteni. Miután meghozta a döntését, hogy az épp beérkező csomagot melyik kimeneti csatornára kell küldenie, akkor megvizsgálja, hogy a választott kimeneti csatorna épp szabad-e. Ha szabad, akkor végrehajtja az átvágást, vagyis a csomag már beérkezett adatait átmásolja a választott csatorna kimeneti pufferébe, és elkezdi a csatornán továbbküldeni azokat. A

csomag később megérkező adatait pedig a beérkezésük után azonnal átmásolja a bemeneti pufferből a kimeneti pufferbe, vagyis folytatja az átvágást a csomag minden további részére. Ha azonban a választott kimeneti csatorna épp foglalt, akkor a csomagot teljes egészében megvárja és bemásolja a bemeneti puffermemóriájába. És csak miután beérkezett a csomag összes adata, akkor vizsgálja meg újra, hogy a választott kimeneti csatorna szabad-e, és amint szabaddá válik, akkor másolja át a kimeneti puffermemóriába a csomagot, és kezdi el kiküldeni azt. Tehát, ha nem lehetséges az átvágás, akkor a „tárol és továbbít" algoritmus szerint működik tovább. [2]

Ezek alapján látható, hogy minden hálózati eszközre igaz az „átvágó" algoritmus esetén is az, hogy képesnek kell lennie egy teljes csomag összes adatának tárolására mind a bemeneti, mind pedig a kimeneti puffermemóriájában is az adatvesztések elkerülése érdekében. [8] [9]

Az „átvágó" algoritmus erőssége a blokkolásmentes esetben mutatkozik meg, vagyis ha nincsen torlódás, és minden hálózati kapcsoló esetén szabad a választott kimeneti csatorna. Ugyanis ilyenkor csővezeték (pipeline) módon haladnak keresztül az egymást követő hálózati eszközökön a csomag részei, mintha egy láncot alkotnának. Természetesen ilyen esetben egy másik kommunikáció számára az összes a csővezetékben résztvevő hálózati eszköz blokkolást okozna.

A legjobb hálózati késleltetés képlete [2]:

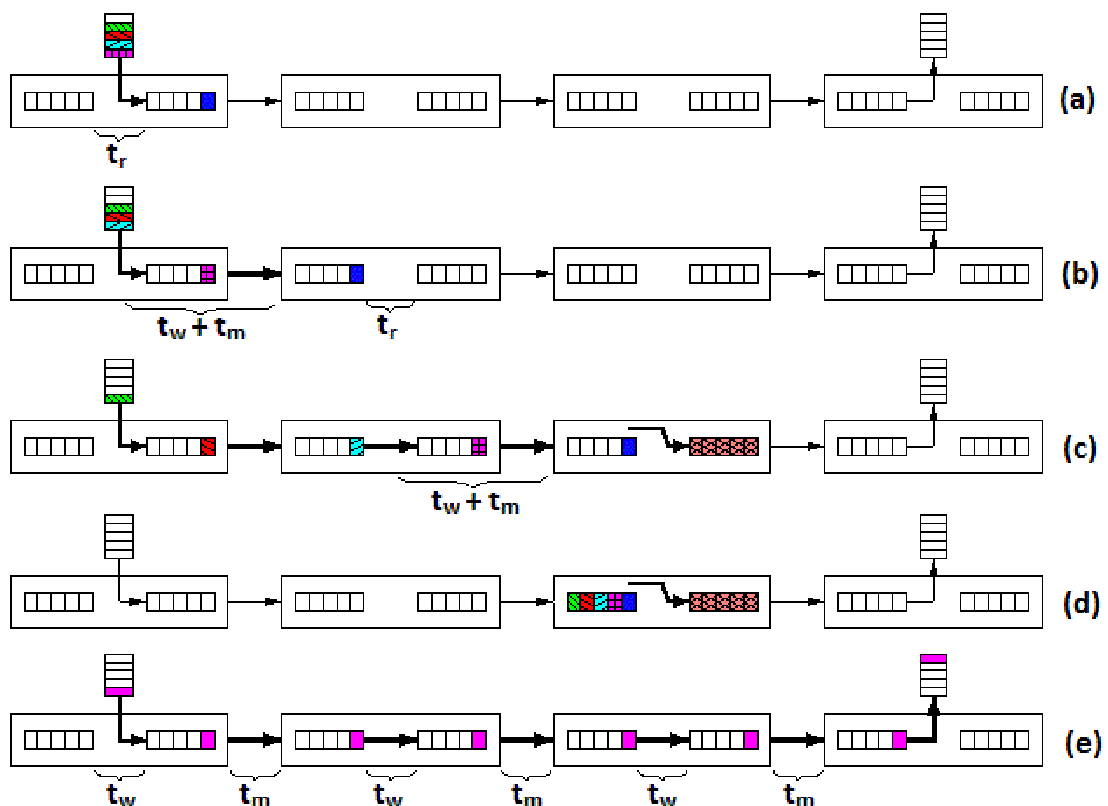
$$t_{CT} = d(t_r + t_w + t_m) + max(t_w, t_m)M$$

Az egyenlet egy összesen M phit hosszúságú csomag, d kapcsolón keresztüli távolságra küldendő üzenet egy csomagjának terjedési idejét írja le, ahol t_m a csatornakésleltetés, t_r a döntési késleltetés és t_w a kapcsolási késleltetés.

Az egyenlet részei is jól mutatják a csővezeték kialakulását és az átvágás erősségét ideális esetben. A cél eléréséig minden egyes kapcsoló esetén csak az első phit adat szenved el döntési késleltetést, mert ennek ismeretében a kapcsoló már ki tudja választani a szükséges kimeneti csatornát. Továbbá ezt az első adategységet szokásosan mindig a kimentre kell kapcsolni, illetve ez szenvedi el a terjedési késleltetést is. Ezt szemlélteti az egyenletben az első $d(t_r+t_w+t_m)$ tag. Amint az első phit adat elérte a célt, onnantól kezdve a többi adatnál már késleltetés nélkül lehetséges az átvágás, és emiatt a terjedési és a kapcsolási késleltetés közül a nagyobbik időtartam okozza csak a késleltetését az áthaladáskor, azaz csővezetékszerűen haladhatnak. Ezt mutatja az

egyenlet második $max(t_w,t_m)M$ tagja is. Fontos megjegyezni, hogy a képlet azzal az egyszerűsítéssel is él, hogy az átvágás megvalósításának nincsen további időjáruléka. A képlet feltételezi azt is, hogy a hálózati eszközök bemeneti és kimeneti puffermemóriával is rendelkeznek. Ha csak az egyik van, akkor nem a terjedési és a kapcsolási késleltetés közül a nagyobbik időtartammal, hanem a kettő összegének idejével ($max(t_w,t_m)$ helyett t_w+t_m értékkel) kell számolni az átvágó phitek esetén.

Az „átvágó" kapcsolás működésének és időzítéseinek szemléltetése látható az alábbi, 3-4. ábrán is.



3-4. ábra: „Átvágó" algoritmus működése [2]

Az ábrán a téglalapok jelölik az egyes kapcsolóeszközöket, amelyeken keresztülhalad az üzenet csomagja. A kapcsolókon belüli kis négyzetek pedig a be- és kimeneti puffermemóriát szimbolizálják. Természetesen egy kapcsolóeszköz több ki- és bemenettel rendelkezik, azonban az ábrákon ezek nem lettek külön feltüntetve. Az összes ábrán a bal oldali kapcsolóhoz csatlakozik az adó és a jobb oldalihoz pedig a vevő. Az ábra (a) része a csomag elindítását és az első irányválasztást, a (b) része a vezető adategység továbbugrását és az első átvágást, a (c) része a szabad kimeneti

csatornák esetén történő átvágásokat, a (d) része a torlódás esetén történő bevárást, az (e) része pedig az ideális csővezetéket szemlélteti úgy, hogy közben az ugrásokat meghatározó késleltetések is jelölve vannak. [2]

A „tárol és továbbít” algoritmushoz hasonlóan itt is problémát jelent a szükséges nagy puffermemória, emiatt az „átvágó” algoritmus esetében is korlátozott méretű csomagok lehetnek csak a hálózaton. [10]

A legjobb hálózati késleltetési idő képlete alapján belátható, hogy a hálózati késleltetés arányos a csomagok méretének és a hálózaton megtett útvonal távolságának összegével. Jelentős előny a „tárol és továbbít” algoritmushoz képest, hogy csak két paraméter összegével és nem a szorzatával arányos. Továbbra is célszerű törekedni arra, hogy a kapcsolások során a lehető legrövidebb útvonalon jusson el a csomag a célhoz, de fontosabb ennél, hogy lehetőleg kisméretű csomagok legyenek a hálózaton.

Az „átvágó” algoritmus mindig csak az első néhány információs egység alapján dönt az egész csomag irányáról, azaz csak a csomag eleje tartalmazza a vevő címét, a további adatok az első adatot követhetik csak. Emiatt egy csomag adatainak küldését nem lehet összefésülni vagy multiplexelni más csomag adataival. [10]

Az „átvágó” algoritmust is általában akkor használják, hogyha kisméretű és gyakori üzenetek csomagváltásai történnek a hálózaton, hasonlóan a „tárol és továbbít” algoritmushoz, mert igen hasonlítanak egymásra. De az átvágással ideális körülmények között kisebb hálózati késleltetés érhető el. Ráadásul a legrosszabb esetben is csak a „tárol és továbbít” algoritmussal egyező késleltetést érhet el az „átvágó” algoritmus.

3.2.4. „Féreglyuk” kapcsolás (Wormhole Switching, WS)

A „féreglyuk” kapcsolás az „átvágó” kapcsolás egy továbbfejlesztése, de a működési alapelvük megegyezik, emiatt a köznyelvben keveredhet a két algoritmus.

Az üzenetet a hálózatba küldése előtt az adónak továbbra is fel kell darabolnia a hálózat számára kezelhető adott méretű adategységekre, és ki kell alakítania a hálózati protokollnak megfelelő keretezést is, mint a „tárol és továbbít” és az „átvágó” algoritmusok esetében, hiszen a „féreglyuk” algoritmus is csomagkapcsolás-elvű.

A „féreglyuk” kapcsolás alapelve megegyezik az „átvágó” kapcsoláséval, vagyis a csomag továbbküldése már azelőtt megkezdődhet, sőt meg is kell, hogy kezdődjön, mielőtt beérkezne az utolsó adategysége is a csomópontba.

A legfontosabb különbség, hogy a „féreglyuk” kapcsolás esetén a hálózati kapcsolóknak nem szükséges olyan nagyméretű puffermemóriákkal rendelkeznie, hogy az egész csomag összes adata beleférjen, hanem elegendőek kisméretű pufferek is. Emiatt a gyakorlatban az ilyen kapcsolók csak igen kisméretű, néhány phit méretű információ befogadására alkalmas ki- és bemeneti puffermemóriákkal rendelkeznek.

A kis pufferek miatt a „féreglyuk” kapcsolási technika teljes egészében az átvágásokra épül. Az első adategység tartalmazza csak a vevő címét, és csak az abban lévő információk alapján minden kapcsolóeszköz el tudja dönteni, hogy melyik kimeneti csatornáján kell továbbítani. A csomag továbbra is kisebb részekre bontható, és ezek az adategységek az első útvonalát követik egymás után, az átvágást kihasználva, a kapcsolókban. Mivel minden kapcsolóeszköz puffere kicsi, és nem fér bele a teljes csomag összes adategysége, emiatt mindenképp szükséges, hogy az összes kapcsolón átvágás történjen, és emiatt lényegében egy csővezetéket kezdenek el kiépíteni az adótól a célállomás felé. A csomag átviteléhez lefoglalt egymást követő pufferek és adatkapcsolati vonalak halmazát nevezik féreglyuknak. Ideális esetben a féreglyuk teljes hosszában létre tud jönni az adótól a vevőig, és lehetővé válik a csomag azonnali átvitele. Ha azonban a terjedés közben az egyik kapcsolóeszköznél a kimeneti csatorna foglalt, akkor a kis pufferek miatt nincsen lehetőség a csomag egyes adategységeinek továbbterjesztéséhez a féreglyukban, és mivel egy kapcsoló pufferében történő bevárása sem lehetséges a teljes csomagnak, emiatt az egész lánc beragad, leáll az aktuális állapotában. Ráadásul a leállás alatt az eddig kialakított teljes útvonalon a lefoglalt erőforrásokat fogva tartja, és ezzel blokkolja más kommunikációk terjedését. Ez az állapot egészen addig így marad, amíg fel nem szabadul a kimeneti csatorna, és a féreglyuk nem tud továbbterjedni. [2]

Rendkívül fontos megállapítás, hogy a féreglyuk hossza arányos a csomag hosszával. Ez azt jelenti, hogy éppen annyi adatkapcsolati vonalat és annyi kapcsoló pufferét tartja fogva a kommunikáció, ahány adategységből áll a csomag. És a féreglyuk terjedése során előrelépés esetén mindig lefoglalja az újabb erőforrást a csomag eleje számára, míg a végén lévő erőforrások felszabadulnak. Előfordulhat olyan helyzet is, hogy a csomag több adategységből áll, mint az adó és a vevő közötti távolság. Ilyenkor a féreglyuk az adótól a vevőig tart, és ezen át haladnak az adategységek. Összességében elmondható, hogy kritikus információ az, hogy hány adategységből áll össze egy

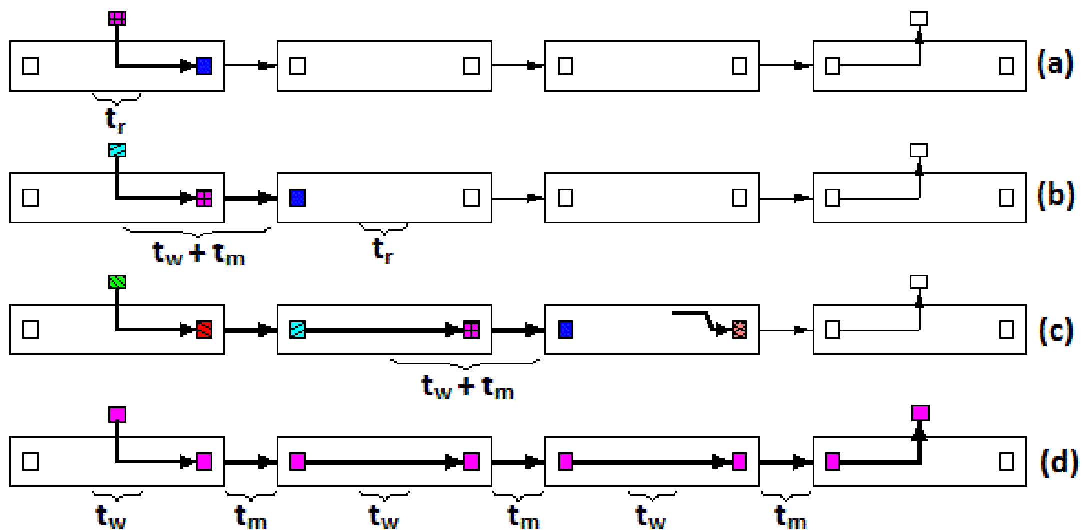
csomag, ugyanis ilyen hosszú lesz a féreglyuk, és ezzel arányos mennyiségű erőforrást foglal a csomag továbbításához a kommunikáció minden időpillanatban.

A „féreglyuk” algoritmus is blokkolásmentes esetben hatékony módszer. Abban az esetben, ha nincs torlódás és minden hálózati kapcsoló esetén szabad a választott kimeneti csatorna, akkor kialakulhat a féreglyuk csővezetéke, és ezen keresztül kis késleltetéssel haladnak keresztül az egymást követő hálózati eszközökön a csomag adategységei, mintha egy láncot alkotnának.

A legjobb hálózati késleltetés képlete megegyezik az „átvágó” algoritmus esetén felírt képlettel, hiszen ideális esetben ugyanúgy egy csővezeték képződik. Az egyenlet jelölései is megegyeznek az „átvágó” kapcsolásnál alkalmazottakkal. [2]

$$t_{WH} = t_{CT} = d(t_r + t_w + t_m) + max(t_w, t_m)M$$

A „féreglyuk” kapcsolás működésének és időzítéseinek szemléltetése látható az alábbi, 3-5. ábrán is.



3-5. ábra: „Féreglyuk” algoritmus működése [2]

Az ábrán a téglalapok jelölik az egyes kapcsolóeszközöket, amelyeken keresztülhalad az üzenet csomagja. A kapcsolókon belüli kis négyzetek pedig a be- és kimeneti puffermemóriát szimbolizálják. Természetesen egy kapcsolóeszköz több ki- és bemenettel rendelkezik, azonban az ábrákon ezek nem lettek külön feltüntetve. Az összes ábrán a bal oldali kapcsolóhoz csatlakozik az adó és a jobb oldalihoz pedig a vevő. Az ábra (a) része a csomag elindítását és az első irányválasztást, a (b) része vezető adategység továbbugrását és az első átvágást, a (c) része a torlódás esetén történő

beragadást, a (d) része pedig az ideális csővezetéket szemlélteti úgy, hogy közben az ugrásokat meghatározó késleltetések is jelölve vannak. [2]

A „tárol és továbbít”, illetve az „átvágó” algoritmusokkal ellentétben a „féreglyuk” kapcsolás esetén már lehetséges, hogy a kapcsolók kis pufferekkel rendelkezzenek, emiatt gyorsabb működésű és olcsóbb eszközök készíthetők.

„Féreglyuk” kapcsolás esetén a legjobb hálózati késleltetési idő képlete megegyezik az „átvágó” kapcsoláséval, azaz a hálózati késleltetés arányos a csomag méretének és a hálózaton megtett útvonal távolságának összegével. Ez továbbra is jelentős előnyt jelent a „tárol és továbbít” algoritmushoz képest. Továbbra is célszerű törekedni arra, hogy a kapcsolások során a lehető legrövidebb útvonalon jusson el a csomag a célhoz, de fontosabb ennél, hogy lehetőleg kisméretű csomagok legyenek a hálózaton.

A „féreglyuk” kapcsolás esetén azonban fokozottan oda kell figyelni csomagok méretére, mert a féreglyuk hossza arányos az adategységek számával, amely pedig az erőforrásigénnyel arányos, emiatt a kisméretű csomagok az előnyösek.

Az „átvágó” algoritmushoz hasonlóan itt is igaz, hogy csak a csomag eleje tartalmazza a vevő címét, amely alapján az útválasztás lehetséges, így a többi adategység az elsőt követi mindenképpen. Emiatt egy csomag adatainak küldését nem lehet összefésülni vagy multiplexelni más csomag adataival.

A „féreglyuk” kapcsolás lehetővé teszi a kis pufferű, gyors és olcsó kapcsolóeszközök kialakítását, ráadásul elegendő csak a bemeneti puffer megléte a működéséhez.

A legnagyobb hátránya azonban a féreglyuk beragadásának lehetősége, amely során az eddig lefoglalt erőforrásokat is foglalva tartja. Ráadásul egyetlen kommunikációs féreglyuk beragadása a hálózaton lavinaszerűen blokkolhatja a többi kommunikációt is, mert azok sem fognak tudni erőforráshoz jutni. Emiatt ez a technika gyakran holtpontok kialakulásához vezet.

A holtpont kialakulása ellen gyakran alkalmaznak virtuális csatornákat a „féreglyuk” algoritmus esetén. Már szó volt róla, hogy ennél a kapcsolásnál nem lehetséges a csomag adatait összefésülni, vagy multiplexelni más csomag adataival, amelynek oka az, hogy nem az összes adategység hordoz információt a csomag célcíméről. Kis továbbfejlesztéssel azonban elérhető, hogy fel lehessen osztani a fizikai

adatkapcsolatot virtuális csatornákká, amelyek segítségével már viszont lehetséges az összefésülés és a multiplexelés. A virtuális csatornák kialakításához egyrészt szükséges a fizikai adatkapcsolat megosztása multiplexeléssel a virtuális csatornák között, másrészt pedig az is fontos, hogy mindegyik kialakított virtuális csatornához tartozzon egy-egy különálló puffermemória mind a két eszközben. További feltétel, hogy a csatorna protokollja képes legyen megkülönböztetni a különböző virtuális csatornákon küldött adatokat egymástól. Ezek megvalósításával teljesíthető a fizikai adatkapcsolat látszólagosan egyidejű megosztása több csomag adategységei között.

A virtuális csatornákat eredetileg a holtpont elkerülése érdekében alkalmazták, azonban javítják a hálózati késleltetést és az áteresztőképességet is. Ha két csomag ugyanazt az adatkapcsolatot szeretné használni, akkor virtuális csatornák nélkül az előbb érkező csomag nyerne, és a később érkező csomag egészen addig blokkolódna, amíg az első összes adategysége át nem ment a kapcsolón. Virtuális csatornák használatával lehetséges a két csomag között a fizikai csatorna megosztása. A korábban érkező csomag eleinte a teljes csatornasebességet kihasználva továbbítódik az adatkapcsolaton, majd mikor a második csomag is ugyanazt a csatornát igényelné, akkor megosztódik a fizikai csatorna a két csomag között, és mind a kettő a fizikai csatornán elérhető sebesség felével továbbítódik. Ez igen előnyös lehet, például ha az előbb érkező csomag igen hosszú, a később érkező pedig igen rövid. Virtuális csatornák nélkül a későbbi csomag sokáig várakozna, pedig az csak rövid ideig használná az adatkapcsolatot. A virtuális csatornák megengedik, hogy a rövid csomag is átjuthasson az adatkapcsolaton, miközben a hosszú csomag átvitele csak egy kis ideig lassul le. Így mind a két kommunikáció együttes átlagos késleltetése javul. További előny, hogy ha a két csomag közül, amelyek megosztottan használnak egy fizikai csatornát, az egyik blokkolódik máshol, akkor a nem blokkolt csomagot a megosztott csatornán a fizikailag elérhető teljes sebességgel lehetséges továbbítani minden egyéb változtatás nélkül.

3.3. A teljes blokkolásmentes hálózati késleltetés vizsgálata

Egy hálózat megtervezése során a csatornák megtervezésekor és a kapcsolási algoritmusok kiválasztásakor az indulási késleltetés mértékét nem szabad elhanyagolni, mert a valóságban igen meghatározó késleltetést visz bele a kommunikációba. A modern eszközökben elérhető magas átviteli sebességek és kis késleltetések miatt nem

elegendő csupán az üzenet vagy csomag terjedésének idejét figyelembe venni, szükséges az üzenetek csomagjainak kialakításának idejével is számolni.

Általánosságban elmondható, hogy az indulási késleltetés (t_s) sokkal nagyobb értékű, mint akár a csatornakésleltetés (t_m), akár a döntési késleltetés (t_r), vagy akár a kapcsolási késleltetés (t_w). Ennek megfelelően az alábbi egyszerűsítéssel lehet élni. [2]

$$t_s \gg t_m \rightarrow \; t_m \approx t_w \approx t_r$$

Az egyszerűsítések következtében az egyes algoritmusoknál kiszámított elméleti legjobb hálózati késleltetések képletei is egyszerűsíthetők. [2]

$$t_{SF} \cong t_s + dMt_m$$

$$t_{CS} = t_{CT} = t_{WH} \cong t_s + dt_d + Mt_m$$

Az egyenletekben megjelenő t_d a döntési késleltetés és a kapcsolási késleltetés összege ($t_d=t_r+t_w$), t_s az indulási késleltetés és t_m pedig a csatornakésleltetés.

Az egyszerűsítések figyelembevételével azonnal láthatóvá válik, hogy a vonalkapcsolás, az „átvágó" kapcsolás és a „féreglyuk" kapcsolás közelítőleg egyformán gyors kapcsolási módszerek, míg a „tárol és továbbít" kapcsolás különbözik ezektől sebességben. Itt igen erőteljesen látható a különbség a hálózati késleltetések között. A „tárol és továbbít" algoritmus esetében a hálózati késleltetés arányos a csomagméret és a távolság szorzatával, míg a másik három esetén csak a két paraméter összegével.

Továbbá valódi hálózatok esetén viszonylag nagyobb méretű csomagok is előfordulhatnak a hálózaton, illetve a modern hálózati eszközök kis késleltetéssel rendelkeznek. Emiatt általánosságban igaz az $Mt_m >> dt_d$ egyenlőtlenség. Ennek következtében a hálózati késleltetés csak igen kis mértékben függ az adó és a vevő távolságától, sőt blokkolásmentes esetben nem is függ tőle. Emiatt a vonalkapcsolás, az „átvágó" kapcsolás és a „féreglyuk" kapcsolás távolságfüggetlen, míg a „tárol és továbbít" kapcsolás távolságfüggő kapcsolás.

3.4. Gyakorlatban használt kapcsolások összehasonlítása

Manapság az Ethernet technológia (IEEE 802.3-as szabvány) a legelterjedtebb és leggyakrabban használt a helyi hálózatok esetén, emiatt az ilyen típusú hálózatokban alkalmazott kapcsolási algoritmusok a legkiforrottabbak. Az Ethernet protokoll által

előírt keretben a fejlécben található a vevő címe, míg a farokrészben található egy ellenőrző összeg, amely alapján a vett keret hibamentességét lehetséges megállapítani.

Ezekben a hálózatokban már az 1990-es évek óta a „tárol és továbbít”, valamint az „átvágó” kapcsolási algoritmust használják. A két algoritmust működését ismerve érdemes egymáshoz képest összehasonlítani azok tulajdonságait az Ethernet technológia által előírt protokoll keretei között is.

A „tárol és továbbít” algoritmus esetén a kapcsolóeszköz mindig bevárja a teljes csomag minden adatát, mielőtt egyáltalán a továbbhaladási irányáról döntene. Ennek következtében lehetősége van a csomag tartalma alapján az ellenőrző összeg kiszámítására, és a farokrészben találtható értékkel történő összevetésére, és eltérés esetén a hibás keret eldobására, mert a teljes csomag a kapcsolóeszközön belül tartózkodik, és mert csak a teljes beérkezése után kezdi meg a csomag továbbítását. Így a hibás csomagot észlelni és eldobni is képes, azaz megakadályozhatja a hibás keretek terjedését. Az „átvágó” algoritmus esetében az átvágások miatt ugyan van lehetőség az ellenőrző összeg kiszámítására az áthaladás során, és a keret végi értékkel is össze tudja hasonlítani a kapcsoló, de nem tudja eldobni a csomagot eltérés, hiba esetén, mert mire rendelkezésére állna a keret helyességének ténye, addigra már továbbította átvágással a csomag elejét. Így a hibás csomagot csak észlelni tudja, de eldobni nem, legfeljebb megjelölheti kerethibásként a csomag végén, de a hibás kereteket mindenképp továbbítja. [3] [4]

A teljes csomag adatainak bepufferelése miatt a „tárol és továbbít” algoritmus egyszerűbb, átláthatóbb és kevesebb erőforrást igénylő továbbítási mechanizmussal rendelkezik, mint az „átvágó” algoritmus. A betárolás hiánya miatt az „átvágó” algoritmus esetében szükség lehet további műveletek végrehajtása a biztonságos adatkezelés érdekében. [3]

A „tárol és továbbít” algoritmus csak késleltetve, a teljes csomag beérkezési után hoz döntést a csomag irányáról, emiatt egy hosszabb, de viszonylag egységes időtartamot igényel a továbbítás végrehajtása minden csomag esetén. Ezzel szemben az „átvágó” algoritmus ideális körülmények között sokkal hamarabb, a vevő címét tartalmazó adategységek beérkezése után már képes a továbbítás megkezdésére. De ez csak akkor igaz, ha a választott kimeneti csatorna épp szabad, így a valóságban erőteljesen változhat a továbbítás időigénye, de legrosszabb esetben sem lépi túl a „tárol és továbbít” algoritmus időigényét. [3] [4]

Valóságos körülmények között is kimutatható az átvágás miatti időnyereség az „átvágó” algoritmus esetén a „tárol és továbbít” algoritmushoz képest, de ez a manapság használt kommunikációs sebességek esetén, a legnagyobb hálózati eszközöket gyártók teszteredményei alapján is csupán néhány 10 μs ideális esetben. [3] [10]

A gyakorlati használatban ráadásul az „átvágó” algoritmus nem feltétlen csak a csomag első néhány adategységének fogadása után dönt a csomag továbbításának irányáról. Gyakran több információt is begyűjt a csomagról, akár a magasabb rétegek fejléc-információit is bevárja, hogy kifinomultabb és pontosabb útválasztási döntést hozhasson az eszköz, és ne csak az adatkapcsolati réteg csekély információi alapján döntsön. [3]

Fontos kiemelni, hogy az „átvágó” algoritmus néhány felhasználási esetben elveszítheti ezt az előnyét a „tárol és továbbít” algoritmushoz képest. Legszembetűnőbb eset, ha két különböző sebességű hálózat között működik a kapcsolóeszköz. Ha a gyorsabból a lassabba szükséges küldeni, akkor a sebességkülönbség miatt úgyis szükséges az üzenetek pufferelése, és ilyenkor a „tárol és továbbít” elvén működik a kapcsoló. Ha a lassabból a gyorsabba szükséges küldeni, akkor pedig az átvágás esetén aláfutás alakulna ki, és nem állna rendelkezésére az adat, amely átvághatná magát a kapcsolón, azaz itt is pufferelés és a „tárol és továbbít” elve szükséges. Egy egységes sebességű erősen terhelt hálózat esetén is elveszítheti az előnyét, ahol a nagy forgalom miatt sok blokkolás alakul ki a kommunikációk között. Ütközés esetén pedig mindenképp fogadja a csomag összes adategységét, így hiába döntene hamarabb a kimenet irányáról, csak az egész csomag beérkezése után továbbítja azt. Azaz ilyen helyzetben lényegében az „átvágó” kapcsolás „tárol és továbbít” elven működik. Ablakozást alkalmazó kommunikációs protokollok esetén sem érvényesül teljes mértékben az átvágás előnye, hiszen ilyenkor néhány átküldött csomag után az adó egy visszajelzésre vár, amely már önmagában sokkal jobban növeli a válaszidőt, mint a kapcsolások miatti késleltetés. Amennyiben a hálózattal szemben felállított igény bizonyos csomagok kiszűrése, ahol a szűrési feltétel nem feltétlenül a csomag fejlécére vonatkozik, akkor pedig nem is lehetséges az átvágást használni, hiszen mindenképp szükséges bevárni a csomag tartalmát is. [3]

Legtöbb felhasználási területen ráadásul nem is a kapcsolások miatti késleltetés a meghatározó két egymással kommunikáló alkalmazás üzenetcseréi között, mert leggyakrabban az alkalmazások között már milliszekundum nagyságrendű késleltetések

alakulhatnak ki az alkalmazások működéséből adódóan. Ilyenkor a kapcsolásokból adódó néhány mikroszekundum késleltetés elhanyagolható, és alig érzékelhető különbség a két kapcsolási algoritmus között. [10]

A nagyteljesítményű számítástechnika (High Performance Computing, HPC) és szuperszámítógépek esetén válik fontossá a két algoritmus közötti különbség. Az ilyen alkalmazások közötti megengedett késleltetés néhány 10 μs vagy kevesebb, és ilyenkor már célszerű az „átvágó" algoritmust használni. [3]

Régebben szinte kizárólag a „tárol és továbbít" algoritmus szerint működtek a kapcsolóeszközök, mert a korlátozott sebesség és a nagyobb hibaarány miatt csak kis komplexitású áramköröket lehetett megvalósítani. A technikai fejlődés következtében az eszközök megbízhatóbbak, emiatt kevesebbet hibáznak, így nem feltétlenül jelent problémát az esetleges hibák továbbterjedése, illetve a bonyolultabb felépítésű eszközök nem lassúak, így már az összes funkcionalitással együtt megvalósítható azonos sebesség mellett az „átvágó" kapcsolás is. Manapság az „átvágó" algoritmust alkalmazó eszközökben beállítható, hogy mekkora mennyiségű információ alapján döntsön a továbbítás irányáról, így lényegében kezd összemosódni a két algoritmus működési elve. [3]

4. Követelmények az elkészített eszközzel szemben

Az előző fejezetekben részleteztem a diplomaterv elkészítéséhez szükséges elméleti ismereteket. Ebben a fejezetben pedig részletezem azokat a legfontosabb követelményeket, amelyekre figyelni kellett a tervezés során, és amelyeket az elkészült eszköznek maradéktalanul teljesítenie kell.

Elsőként ismertetem az elvárások összegyűjtésének módját, utána pedig sorban részletezem az intelligens kapcsolóval szemben állított követelményeket.

4.1. Követelmények összegyűjtési módja

A diplomatervezés során megtervezett intelligens kapcsolóeszközzel szemben még a tervezési fázis előtt, a munka kezdetén gyűjtöttem össze az alapvető elvárásokat, amelyeknek meg kell felelni a tervezés során. Természetesen az eleinte felmerült követelménypontok kissé módosultak a tervezés során, de radikális változás nem történt.

A követelmények összeírása egy hosszabb megbeszélés keretében történt, ahol a konzulensem, Balogh András, és a leendő felhasználók közösen elmagyarázták az eszköz céljait és ismertették igényeiket, kéréseiket, amelyeket ezzel az eszközzel kívánnak megvalósítani. A megbeszélés végén a kapcsoló felhasználási környezetét és annak fizikai elrendezését is megmutatták.

A felhasználási környezetről kiderült, hogy az intelligens hálózati kapcsolóeszköz a folyamatos integráció (continuous integration) [11] folyamata során lesz használva. A folyamatos integráció folyamata során, ha egy fejlesztő változtat egy kódrészleten, és azt egy verziókövető rendszerben véglegesíti (commit) azt, akkor egy program megpróbálja lefordítani a megváltoztatott kódot, és ha sikerült, akkor néhány előre megírt, automatizált tesztet is lefuttat egy tesztkörnyezetben. A fordítás és a futtatás eredményéről pedig azonnal értesítheti is a fejlesztőt, aki így rögtön megbizonyosodhat arról, hogy a megváltoztatott kód is működőképes maradt-e vagy sem. Nagyobb cégeknél, mint amilyen a ThyssenKrupp Presta Hungary Kft. is, többféle termékeket fejlesztenek párhuzamosan, sőt a különböző programok egymásra is épülnek, így még fontosabb és hatékonyabb eszköz a folyamatos fejlesztés.

Egy tesztelő eszköz azonban több célhardver tesztelése során is használható, így nem célszerű minden tesztkörnyezetbe egy-egy különálló tesztelő eszközt is beépíteni, hiszen ilyenkor az egyes tesztelő eszközök a működési idő igen alacsony hányadában lennének kihasználva. Ehelyett sokkal költséghatékonyabb ezeknek a közösen használt egységeknek a megosztása az egyes tesztkörnyezetek között, mert a különböző tesztek megfelelő időzítésével elkerülhető, hogy a különböző tesztrendszerek a közös erőforrásokra várjanak, ellenben a közös erőforrások kihasználtsága növekszik. Ilyen tesztkörnyezetben a folyamatos integrációt alkalmazva pedig elengedhetetlen, hogy a közös erőforrásnak számító tesztelő eszközöket távolról programozható módon bármelyik tesztkörnyezetbe bekapcsolhassuk, hiszen csak így lesz képes az összes tesztkörnyezet vezérlésére. Ezek alapján belátható, hogy egy igen fontos alkotóegysége lesz a diplomatervezés során elkészült intelligens kapcsoló a tesztkörnyezetnek.

Fizikai elhelyezését tekintve a kapcsolóeszköz egy szerver-szekrényben (rack cabinet) fog működni. Emiatt különösebb mechanikai terheléseknek, vagy egyéb veszélyforrásoknak nem lesz kitéve az eszköz, így kereskedelmi célú olcsóbb alkatrészek használata is megengedett. A zárt környezet miatt viszont fontos, hogy távolról is vezérelhető legyen az eszköz.

4.2. Célfunkció

Az eszköz alapvető és elsődleges célfunkciója, hogy bármelyik hálózati csatlakozóján beérkező kommunikációt minden egyéb csatlakozóra képes legyen továbbítani.

A legáltalánosabb felhasználási mód esetén mindig csak két csatlakozó között kell összeköttetést létesíteni, mert mindig a közös erőforrásnak tekinthető tesztelő eszközt kell annak a tesztrendszernek a hálózatára rácsatlakoztatni, ahol épp szükséges, Olyan használati eset a folyamatos tesztelés folyamata során nem fordulhat elő, amikor két tesztkörnyezetet kommunikálna egymással, így egy időben mindig elegendő két csatlakozási végpont között pont-pont kapcsolatot létrehozni.

Az általános felhasználási módon felül célszerű egyéb alkalmazási lehetőségekre is felkészíteni az eszközt a továbbfejleszthetőség és későbbi széleskörű alkalmazás érdekében. Ezek alapján lehetővé kell tenni egy időben nem csak pont-pont kapcsolatok, hanem akár több végpontú, adatszórásos hálózatok kialakítását is. Továbbá annak a lehetőségét is érdemes kialakítani, hogy egyszerre akár több hálózat

adatforgalmát is kezelni tudja a kapcsolóeszköz, és így több hálózatra csatlakozó eszközök között is kapcsolatot tudjon teremteni úgy, hogy a különböző hálózatokon lévő eszközök adatforgalmai nem zavarják, és nem akadályozzák egymást.

4.3. Funkcionális követelmények

A célfunkció és a legvalószínűbb használati módon túl számos egyéb peremfeltétel teljesítése is szükséges, a megfelelő és a felhasználók számára elfogadható megoldás érdekében.

Az egyik legfontosabb funkcionális követelmény, hogy az intelligens kapcsolóeszköznek legalább nyolc CAN hálózati csatlakozási lehetőséggel kell rendelkeznie. Ennek megfelelően lehetővé kell tennie, hogy a tesztelő eszközt akár hét tesztkörnyezetbe is be lehessen kapcsolni, vagy akár négy pont-pont kapcsolat kialakítására is képes legyen egy időben.

Manapság több gépjárműben is alkalmaznak FlexRay-hálózatot, emiatt további fontos követelmény, hogy ne csak CAN, hanem FlexRay-hálózatok kezelésére is fel kell készíteni az eszközt. Bár a diplomaterv kiírásának nem része a FlexRay-hálózatok kezelése, azonban az elterjedtsége miatt, konzulensem kérésére hardveresen biztosítani kell ezt a lehetőséget is. Peremfeltételként hasonlóan a CAN-hez, nyolc FlexRay kapcsolat kialakítását kell legalább megvalósítania az eszköznek.

Fontos megállapítás, hogy az intelligens kapcsolónak nem tartozik a feladatai közé az egyes hálózatokon egyéb adatok generálása, csak valamelyik aktív csatlakozójára érkező üzeneteket kell továbbítania. Sőt az érkező üzenetek összefésülése egy vagy kevesebb üzenetté, illetve a beérkező üzenetek kisebb egységekre darabolása sem tartozik a feladatai közé. Összességében tehát elmondható, hogy valóban csak az adatkapcsolati rétegben kell tevékenykednie az intelligens kapcsolónak.

Az elkészült eszköznek hardveresen elegendő egy alapvető elosztói funkciót biztosítania, azaz bonyolultabb programozás nélkül, egy inicializálás után bármelyik csatlakozóján beérkező üzenetet továbbítani tudja az összes azonos típusú hálózati csatlakozójára. Ez a követelmény biztosítja, hogy a lehető leghamarabb felhasználhatóvá váljon egyszerűbb feladatok esetén.

A megtervezett eszköznek elegendő szoftveresen teljesítenie a célfunkciót, vagyis lehetővé tenni, hogy egy csatlakozón beérkező adatcsomagot csak adott

célcsatlakozóra, vagy csatlakozókra továbbítsa irányítottan. Azonban az elvárás, hogy a lehető legkisebb késleltetést szenvedjék el a kapcsolón áthaladó csomagok.

A követelmények két magasabb szintű funkció teljesítésére is kötelezik az intelligens kapcsolót: szoftveresen képesnek kell lennie hibainjektálásra, valamint a rajta áthaladó forgalmat naplóznia kell tudni. Ezek a diplomaterv feladatkiírásában is szerepelnek. A hibainjektálás során a rajta áthaladó üzenetek egy részén különböző módosításokat kell alkalmaznia, amelynek célja, hogy szimulálni lehessen a külső zavarokat, és tesztelni lehessen, hogy különböző tesztelt hardverek hogyan viselkednek a hibás átvitelből következő adatok megváltozásának hatására. A forgalomnaplózás során az eszköznek egy időbélyeggel együtt tárolnia kell, hogy milyen adatok haladtak át rajta, mert így az esetlegesen hibás tesztek esetén könnyebben megtalálható hiba oka.

4.4. Vezérlési és csatlakozási lehetőségek

Az elkészített eszköz egy jól definiált tesztrendszerbe fog beépülni, emiatt már követelményként adott volt, hogy pontosan milyen csatlakozási lehetőségekkel rendelkeznie, hogy könnyedén fel lehessen használni.

A funkcionális követelményeknél már megemlítésre került, hogy legalább nyolc CAN és legalább nyolc FlexRay csatlakozóval kell rendelkeznie. Ezen kívül a távolról, Ethernet-hálózaton keresztül történő vezérelhetőség érdekében egy RJ45-ös csatlakozót is fel kell szerelni. A tápellátásnak pedig egy power jack csatlakozón keresztül kell biztosíthatónak lennie.

A távolról vezérelhetőségen túl, rendelkeznie kell közvetlen felhasználói beavatkozási és kijelzési lehetőségekkel is. A kijelzésre négy, az elülső oldalra kivezetett LED-et, míg a beavatkozásra egy alaphelyzetbe állító- és két általános nyomógombot kértek a felhasználók. Ezekkel a fejlesztést lehet segíteni, mert könnyebbé válik a hibakeresés.

Egy később felvetett, de a követelmények közé beépülő pont, hogy legyen egy LCD-kijelző is, amelyre az intelligens kapcsolóeszköz állapotát ki lehet íratni.

4.5. Fizikai kialakítás

A csatlakozási lehetőségeken túl az elkészített eszköz fizikai méreteire is adottak voltak a követelmények, hiszen a követelmények összegyűjtésekor kiderült, hogy a kész eszköz egy általános szerver-szekrénybe kerül.

Az adott fizikai elhelyezés miatt a nyomtatott áramköri panelnek a szabványos 19 colos illesztőhelyre kell beférnie. [12] A szabványnak megfelelő illesztődobozok kívülről 482.6 mm szélességűek és 220.0 mm mélységűek, míg magasságban többféle, de a 44.5 mm-es fix egységméret (U méret) egész számú többszörösének megfelelő magasságú dobozméretek közül lehet választani. Ezek alapján a bennük elhelyezhető NYÁK-ok méreteit is megszabják az egyes illesztődobozok. A könnyű felszerelhetőség miatt a szélességet pontosan be kell tartani, míg a mélységnél és a magasságnál az előírás csak egy maximális érték a NYÁK és a beültetett alkatrészek méretére.

Az egyszerűbb illesztődoboz-kialakítás, és a könnyű hozzáférhetőség érdekében az összes adatkapcsolati (tehát a tápcsatlakozó kivételével az összes) csatlakozónak az elülső oldalon kell elhelyezkednie. Ez azért fontos, mert így az eszköz végleges helyére történő telepítésekor könnyen lehetséges a vezérlési és adatvonalait összekötni a szerver-szekrényben lévő többi tesztrendszerrel.

Az utolsó, de fontos fizikai követelmény, hogy minden kezelt kommunikációs rendszer (Ethernet, CAN, FlexRay) a szabványok szerinti maximális sebességgel is működőképes legyen, hiszen a különböző tesztrendszerek bármelyike megkövetelheti a szabványban előírt maximális sebességértékeket.

4.6. Fejlesztés indoka

A követelmények alapján látható, hogy az intelligens kapcsolóeszköz eleinte egy adott célfunkcióra lesz felhasználva, ennek megfelelően egyedi igényeket kell hogy kielégítsen.

A követelmények összegyűjtése után, de a fejlesztés megkezdése előtt természetesen egy felmérés, keresés is történt, amelyek célja az volt, hogy kiderüljön, hogy megéri-e elkezdeni a fejlesztést, vagy esetleg az igények kielégíthetőek egy a piacon már kapható eszközzel. Ennek eredményként kiderült, hogy a piacon egyáltalán nem létezik olyan hálózati eszköz, amely kimondottan csak CAN, vagy CAN- és FlexRay-hálózatok bármelyik OSI rétegében operálnának.

A hálózati eszközök helyett, csak hálózati analizátorok, és naplózó eszközök, vagy különféle hálózati átalakítók találhatók a piacon. A hálózati átalakítók a CAN hálózati forgalmát, valamilyen egyéb, általában a PC-s világban közismert hálózatra (USB, Ethernet) továbbítják. Ezeket akár fel is lehetne használni, hiszen elképzelhető lenne olyan megoldás is, hogy a tesztelő eszköz CAN adatforgalmát átalakítjuk ethernetes adatcsomaggá, majd a kapcsolás a megfelelő tesztrendszer irányába ezen a hálózaton történik meg, végül pedig a tesztrendszerek előtt egy újabb hálózati átalakító visszaalakítja az adatokat a CAN-hálózatra. A megoldás előnye, hogy az ethernetes közegben jól kiforrott technológiák léteznek a kapcsolás elvégzésére, és emiatt olcsó, egyszerű, és többféle üzemmódú eszközök is léteznek. De a megoldásnak több hátránya is van. A legfontosabb, hogy a sok hálózati átalakítás miatt elviselhetetlen késleltetést vinne bele a tesztrendszerbe a kapcsolóeszköz, pedig a cél az lenne, hogy a lehető leghamarabb jusson el a megfelelő irányba az üzenet a tesztrendszerek és a tesztelő eszköz között, mert a tesztelt eszközöknél az időzítések is kritikus tényezők. További hátránya, hogy sok különálló eszközt igényelne, hiszen minden csatlakozni kívánó CAN-hálózathoz hozzá kell illeszteni egy-egy átalakítót is, így valószínűleg drága lenne ez a megoldás.

Érdemes megemlíteni, hogy a diplomatervezés végén ismételten elvégezve a felmérést, a fejlesztés alatt egyetlen kész piaci megoldás született. A Hotraco Agri cég készített egy CAN hálózati kapcsolóeszközt, amelyet forgalmaz is. Azonban ez csak négy hálózatot tud kezelni, illetve csak alacsony sebességű CAN-hálózatokat képes kezelni, valamint különálló erős és masszív IP54-es tokozást kapott. A cég egyébként sertés- és baromfifeldolgozó cégek számára kínál kisegítő rendszereket. Emiatt érthető, hogy a nagy távolságok miatt elegendő a lassú hálózati kapcsolat kezelése, viszont a robusztus tokozás kritikus ezekben az alkalmazásokban. [13]

Összességében tehát elmondható, hogy nincsen olyan piacon is kapható kész termék, vagy egyéb már meglévő eszközök felhasználásával megvalósítható megoldás, amelyekkel ki lehetne elégíteni az összes követelményt, így érdemes megvalósítani ezt a fejlesztést.

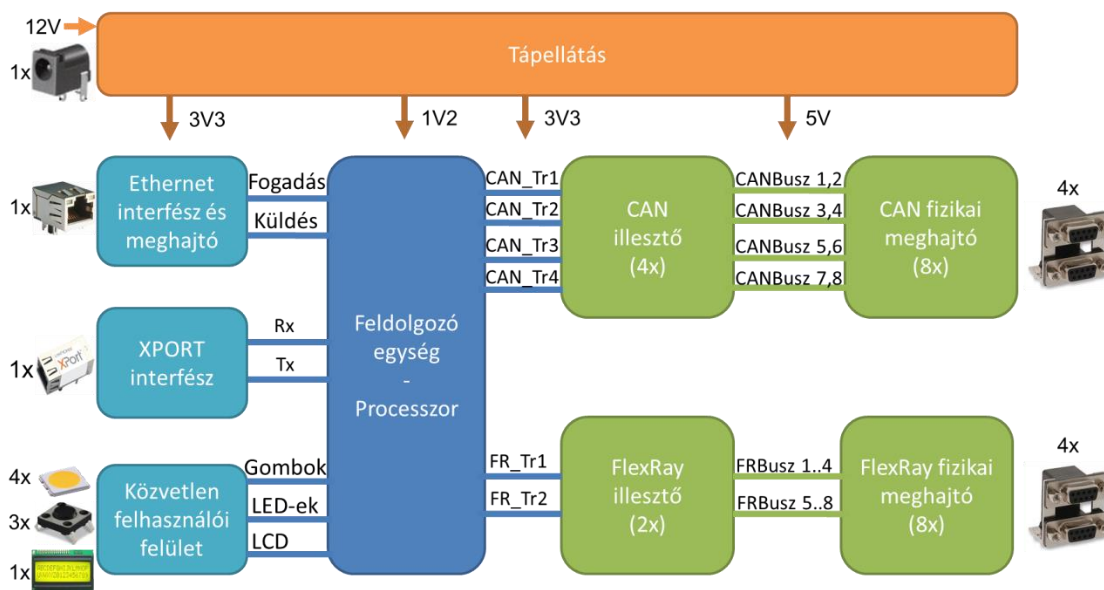
5. A hardvertervezés lépései

Az előző fejezetben ismertettem az intelligens kapcsolóeszközzel szemben támasztott követelményeket, és ezek alapján ebben a fejezetben pedig már rátérek a feladat megvalósítása során felmerült konkrét kérdések és problémák ismertetésére, valamint az azok megoldásaira is. Első lépésként a diplomatervezés során elkészült hardvert, és a tervezése során alkalmazott megoldásokat fogom bemutatni.

Az elkészített hardver minden részletére azonban nem áll módomban kitérni, így a későbbi alfejezetekben is csupán a megvalósítás legfontosabb lépéseire térek ki, amelyek nehezen érthetőek, vagy nem egyértelműek a megoldás szempontjából. Emiatt is fontos, hogy az egyes egységek részletezése előtt bemutassam a követelmények alapján kialakított hardvertervet.

5.1. A hardverterv blokkvázlata

A hardverterv, és az ebből elkészített hardver blokkvázlata látható az 5-1. ábrán:



5-1. ábra: Hardverterv blokkvázlata

Az ábrán láthatók az elkészült eszköz legfontosabb részegységei, illetve az ezek közötti kapcsolatok is, az alegységek színei pedig az egyes egységek fő funkcionalitásaira utalnak.

A legmagasabb szintű funkcionalitás alapján négy csoportba sorolhatók az egyes alegységek. Felül, narancssárga színnel, a többi egységtől erőteljesen elkülönítve látható a nyomtatott áramkör teljes tápellátásáért felelős modul. Középen, a sötétkék színű rész a központi feldolgozóegység, amelynek segítségével lehetséges majd az összetettebb feladatok elvégzésére is beprogramozni az eszközt. A bal oldalon, világoskék színnel találhatók az egyes vezérlési és kommunikációs alegységek, amelyeken keresztül tarthatja a kapcsolatot az eszköz a felhasználóval. Végül jobb oldalon, zöld színnel láthatók a hálózati illesztők és meghajtók azokhoz a hálózatokhoz, amelyeket a követelmények alapján kezelni, illetve esetlegesen manipulálni szükséges.

A blokkvázlat színekkel jelzett legmagasabb szintű funkcionalitásain túl érdemes röviden összefoglalni az egyes alegységek feladatköreit is, mielőtt az egyes egységek részletes ismertetésébe belemerülnénk.

A tápellátás feladata a többi alegység, és így összességében az egész nyomtatott áramköri panel számára a megfelelő feszültségszintek biztosítása. A panel 12 V-os bemeneti feszültségről működik, és ezt alakítja át az egyes egységek számára szükséges, 1.2 V-os, 3.3 V-os és 5 V-os feszültségszintekre úgy, hogy az összes szinten minden alkatrész a szükséges maximális teljesítményt is fel tudja venni a táphálózatból. Kapcsolatait tekintve csak minden további modulnak biztosítja a tápellátást, de vezérlési bemenete nincsen, azaz digitális szempontból izolált a többi modultól.

A feldolgozóegység felelős az eszköz összetettebb funkcióinak elvégzéséért. Feladatát alapvetően 3 lépésre lehet lebontani. Először fogadja a vezérlési és kommunikációs egységek felől érkező parancsokat. Utána értelmezi a kapott parancsokat. Végül pedig a kapott parancsnak megfelelően beállítja a hálózati illesztők állapotát, miközben vissza is jelez a felhasználónak, ha az szükséges. Ezek alapján fontos kiemelni, hogy ez az alegység az, amely minden vezérelhető modullal kapcsolatban áll, hiszen minden vezérlési funkciót a központi feldolgozóegységben lehetséges implementálni. Ennek megfelelően fontos, hogy viszonylag változatos módon és rugalmasan konfigurálhatóan kapcsolódjon a többi alegységhez.

Az Ethernet-interfész és -meghajtóegység az egyik legfontosabb külső vezérlési és kommunikációs modul. Ennek feladata, hogy a felhasználótól fogadja az Ethernet alapú hálózaton keresztül érkező parancsokat, és továbbítsa a parancsok üzeneteit a központi feldolgozóegység számára. Továbbá lehetővé kell tennie, hogy ha szükséges, akkor a központi egység ki is küldhessen üzenetet a hálózaton, amelyben értesítheti a

felhasználót az állapotáról. Így központi feldolgozóegység felé egy fogadási és egy küldési lehetőséget nyújt, míg a külvilág felé egy az ethernetes hálózatokon legelterjedtebb RJ45 csatlakozót biztosít.

Az XPORT-interfész az Ethernet-interfésznél egyszerűbb és kompaktabb kommunikációs lehetőséget biztosít a külvilág és a feldolgozóegység között. Feladata lényegében ugyanaz, vagyis fogadja a felhasználótól érkező parancsokat és küldési lehetőséget biztosít a feldolgozóegység számára, de kapcsolatait tekintve egyszerűbb, hiszen a feldolgozóegység felé egy UART kommunikációs protokollt biztosít, míg a külvilág felé pedig ugyanúgy egy ethernetes RJ45 csatlakozót. Ez az egyszerűsített kommunikációs lehetőség egyszerűsíti és gyorsítja a vezérlési program fejlesztését a panelen.

A közvetlen felhasználói felület négy LED-ből, három nyomógombból, valamint egy karakteres LCD-kijelzőből áll. Ezek segítségével a felhasználó közvetlenül is utasíthatja a feldolgozóegységet, illetve a feldolgozóegység is közvetlenül értesítheti a felhasználót az állapotáról, ha az épp futó vezérlőprogramban valamilyen módon implementált ezek kezelése. Ezek a közvetlen be- és kimenetek leginkább a fejlesztés során a hibakeresést és -javítást segíthetik elő.

A CAN-illesztő a legfontosabb modul a feldolgozóegység és a kezelt hálózatok között. Ennek megfelelően a legfőbb feladata, hogy elérhetővé tegye a központi feldolgozóegység számára az összes CAN-vonal elérését mind olvasási, mind pedig írási lehetőséget is biztosítva. Továbbá lehetővé teszi az egyes CAN-hálózatok hardveres összekötését, és így az üzenetek gyors átjátszását közöttük. A követelmények alapján összesen nyolc vonal kezelését és manipulálását teszi lehetővé az eszköz, és ehhez elegendő négy illesztőegység. Ennek megfelelően a központi egység kapcsolatban áll mind a négy CAN-illesztő részegységgel, illetve egy-egy CAN-illesztő pedig két-két CAN-vonal jeleit képes kezelni fogadási és küldési irányban is. A központi egység és a CAN-illesztő között egyszerű logikai jelek segítségével lehetséges a vezérlés és az adatcsere, míg a CAN-illesztő és a CAN-meghajtó között már a CAN-szabványnak megfelelő jelek biztosítják az összeköttetést. [14]

A CAN fizikai meghajtó alapvető feladata a jelek továbbítása az illesztő modul és a csatlakozók között. A továbbítás során megfelelően formálja is a digitális jeleket, illetve védelmet is nyújt az esetleges csatlakozó felőli túlfeszültségek ellen. A továbbítás mellett a CAN-szabvány [14] által előírt 120 Ω-os lezárást is biztosítja

opcionálisan, valamint gondoskodik arról is, hogy a nem meghajtott differenciális jelvezetékek esetén a tápfeszültség felére húzza mind a két vezetéket, azaz differenciálisan zérus jelértéket biztosít a szabvány alapján. Lényegében ez a modul kezeli a CAN hálózati kapcsolatot a külvilág felé. Így az illesztőmodul felé a szabvány által előírt protokoll szerinti kommunikáció segítségével kapcsolódik, míg kifelé egyszerűen egy DSUB9-es csatlakozót biztosít a CAN-es hálózatok csatlakoztatására.

A FlexRay-illesztő feladatát tekintve erőteljesen hasonlít a CAN-illesztő modulra, csak nem a CAN-es, hanem a FlexRay-es hálózatot kezeli. A legfontosabb feladata ugyanaz, hogy elérhetővé tegye a központi feldolgozóegység számára az összes FlexRay-vonal elérését olvasási és írási irányban egyaránt. Szintén lehetővé teszi a különböző FlexRay-hálózatok közötti gyors, hardveres üzenetátjátszási lehetőséget. A követelmények alapján összesen nyolc vonal kezelését és manipulálását teszi lehetővé az eszköz, és ehhez elegendő csupán két illesztőegység. Kapcsolatait tekintve is hasonlít a CAN-es megoldáshoz, azaz a központi egység egyszerű logikai jelek segítségével kapcsolatban áll mind a két FlexRay-illesztő részegységgel, illetve egy-egy illesztő pedig két-két FlexRay-vonal jeleit képes kezelni, ahol már a FlexRay-szabványnak megfelelő jelek biztosítják az összeköttetést. [15]

A FlexRay fizikai meghajtó alapvető feladata a jelek továbbítása az illesztő modul és a csatlakozók között. Hasonlóan a CAN-meghajtóhoz, ez a modul is védelmet nyújt a csatlakozó felőli túlfeszültségek ellen, illetve a digitális jelek kondicionálását is elvégzi. A továbbítás mellett a FlexRay-szabványnak megfelelő lezárást [15] is biztosítja, illetve gondoskodik a nem meghajtott jelvezetékeken a szabványnak megfelelő jelszint kialakításáról is. Lényegében ez a modul kezeli a FlexRay hálózati kapcsolatot a külvilág felé. Így az illesztőmodul felé a szabvány által előírt protokoll szerinti kommunikáció segítségével kapcsolódik, míg kifelé egyszerűen egy DSUB9-es csatlakozót biztosít a FlexRay-es hálózatok csatlakoztatására.

5.2. Nyomtatott áramköri lemez kialakítása

Az intelligens kapcsoló nyomtatott áramköri lemezének fizikai méretére szigorú követelményeket szab az adott fizikai elhelyezés. A szerverszekrénybe történő rögzítési a szabványának megfelelő [12] illesztődoboznak a Schroff Multipac PRO dobozcsalád 20860-605-ös tagja lett választva. [16], amelynek magassága két egységnyi. A benne elhelyezhető NYÁK pedig 403- 413 mm szélességű, maximum 216 mm mélységű lehet.

Az elkészített panel szélessége 408 mm, mélysége 107 mm, magassága pedig 89 mm lett. Ezzel a szélességgel könnyedén lehet majd ki- és becsúsztatni az illesztőbozba, amely megfelelő stabilitással képes azt megtartani, A panel mélysége alapján belátható, hogy szükség esetén akár két panel is betehető egyetlen dobozba úgy, hogy az egyik csatlakozói a panel elején, a másiké pedig a hátulján vannak kivezetve, így a későbbiekben könnyedén lehetséges egyetlen szerverszekrényben több tesztkörnyezet kialakítása is, anélkül, hogy a kapcsolóelemek több szekrényhelyet igényelnének. A panel magasságát a választott csatlakozók adják, továbbá, így könnyedén lehetséges az LCD-kijelző ráillesztése az elülső oldalra, azaz a szerverszekrényből is könnyedén leolvasható lesz a kijelzett állapot.

A követelmények között szerepelt, hogy minden kommunikációs rendszer a maximális sebességgel működhessen. Ennek alapján a fizikai elrendezésben a központi feldolgozóegységhez legközelebb az Ethernet-interfész és meghajtó található, utána a FlexRay-interfész, legmesszebb pedig a CAN-interfész található a panelen. Ennek oka, hogy a gyorsabb adatvonalak kisebb vezetékhosszt engednek meg, a terjedési késleltetés miatt, illetve általában érzékenyebbek a külső zavarokkal szemben is. Így a leggyorsabb kommunikációs hálózatoknál kritikus szempont, hogy a lehető legrövidebb jelvezetékeken keresztül legyenek elvezetve, míg a lassabb kommunikációk esetében nem okoz problémát a hosszabb jelvezeték. Azaz a kommunikációs rendszerek működési sebessége meghatározta a panelen történő elhelyezésüket is.

5.3. Tápellátás

A tesztrendszerbe történő könnyű beilleszthetőség miatt a bemeneti feszültség 12 V, kimenetként pedig 1.2 V-os, 3.3 V-os és 5 V-os feszültségszinteket kell biztosítania. Az összes tesztelt eszközhöz hasonlóan power jack típusú csatlakozón keresztül lehetséges a betáplálás, az egyszerű illeszthetőség érdekében.

A be- és a kimeneti feszültségszintek közötti nagy feszültségkülönbség, és a fizikai meghajtók, illetve a központi feldolgozóegység nagy teljesítményigényei miatt mindeneképp kapcsolóüzemű tápegység alkalmazása szükséges. Az ilyen tápegységek által okozott zavarok jelen alkalmazásnál nem okoznak problémát, hiszen az eszköz digitális felépítésű, azaz elég robusztus a tápegység által okozott zajokkal szemben.

A tápfeszültségek kialakítását a Texas Instruments TPS65251 integrált áramköre végzi el. A chipben három szinkron buck (feszültségcsökkentő) kapcsolás található,

amelyek közül kettő 2 A, egy pedig 3 A-rel is folyamatosan terhelhető a kimenetén. A chip képes 5, 9, 12, és 15 V-os feszültségű hálózatokról is működni, míg 0.8 V és a bemeneti tápfeszültség közötti tartományban képes kimenetet szolgáltatni. A választás fő indoka, hogy képes a három szükséges feszültségszintet kialakítására úgy, hogy közben a nagy teljesítményigényt is kielégítse. További indok, hogy megfelelő ellenállások és kondenzátorok csatlakoztatásával minden paramétere beállítható, tehát a feldolgozó egység szempontjából lényegében passzív és izolált elem. Ezenkívül a kapcsolási frekvenciája is széles tartományon belül állítható, így az általa generált zaj frekvenciatartománybeli komponensei beállíthatók úgy, hogy egyik kommunikációs hálózatot se zavarhassa meg jelentősen. [17]

A ThyssenKrupp egy másik eszközben már korábban is alkalmazta ezt a tápegységet, amely így bizonyítottan működőképes, illetve a kapcsolási rajzot és az paneli elrendezését át tudtam venni. Fontosnak tartom megemlíteni, hogy a kapcsolás átvételekor ellenőriztem a kapcsolás méretezését is, amelynek során kiderült, hogy a kimeneti pufferkondenzátorok méretezése nem megfelelő a beállított teljesítményigényre, az 5 V-os tápellátás esetén, így a megtervezett panelnél, és a korábbi eszköznél is javítottuk a tápegység kapcsolását.

5.4. Központi feldolgozóegység

A központi feldolgozóegységként használt mikrokontrollert a ThyssenKrupp Presta Hungary Kft. biztosította, és a pontos típusa nem publikus, azonban a diplomaterv szempontjából lényeges adatait bemutatom.

Az alkalmazott mikrokontroller 32 bites 180 MHz-es órajelű, kifejezetten autóipari környezetbe szánt, sőt biztonságkritikus alkalmazásokban is bevethető. A mikrokontrollerben összesen hat processzormag található, amelyek összesen két számítási egységet képeznek, illetve egy mag hardveres védelmi modulként működik. Az összes processzormag Power PC architektúrájú, amely igen közkedvelten alkalmazott beágyazott rendszerekben, valamint hatékony és nagy számítási teljesítményt tesz elérhetővé. A mikrokontroller felépítése alapján elmondható, hogy egyrészről igen erőteljes védelmi mechanizmusokkal rendelkezik, másrészről igen nagy teljesítményt és számítási kapacitást lehet vele elérni.

A két számítási egységnek köszönhetően könnyedén szétválaszthatók a központi feldolgozóegység feladatai. Ugyanis a második számítási egységet kifejezetten a

perifériák kezelésére szánta a gyártó, és az összes periférián történő adatfogadást és adatküldést ezen a magon meg lehet valósítani. Míg ezzel párhuzamosan az első számítási egység az egyéb szükséges számításokat végezheti. Igen erőteljes támogatást nyújt perifériák terén is az alkalmazott mikrokontroller, amelynek igazi erőssége a változatos kommunikációs perifériakészlete. Az alkalmazás szempontjából a legfontosabb, hogy tartalmaz 18 db LIN/UART-, 8 db CAN-, 1db FlexRay-, 2 db Ethernet-, valamint 1 db I2C modult is kommunikációs perifériaként.

A főbb tulajdonságok alapján jól látható, hogy az igen fejlett, nagyszámú és változatos perifériakészlet, valamint a kiváló architektúrája miatt volt célszerű ezt a mikrokontrollert választani az intelligens kapcsolóeszköz központi feldolgozó egységeként.

5.5. Ethernet-interfész és -meghajtó

Az intelligens kapcsolóeszköz az Ethernet-interfészen és -meghajtón keresztül fogadhatja a parancsokat a felhasználótól, illetve értesítheti aktuális állapotáról a felhasználót.

Ez a modul lényegében egy adó-vevő chipből és egy csatlakozóból áll. A felhasznált adó-vevő a Texas Instruments DP83848JSQ/NOPB típusú egycsatornás 10/100 MB/s sebességű Ethernet adó-vevője. Ezt kifejezetten beágyazott rendszerekbe ajánlják a kis mérete, egyszerű kezelhetősége, és erős meghajtási képessége miatt. A csatlakozó pedig a követelményekben előírt az ethernetes hálózatokon legelterjedtebb RJ45 csatlakozó. [18]

Az Ethernet-modult készként vettem át, hiszen a ThyssenKrupp Presta Hungary Kft. több korábbi projektjében is használtak már ezt a működőképes és bevált kapcsolási rajzot és huzalozást. Az adó-vevő vezérlési jelei kompatibilisek a központi feldolgozóegységként választott mikrokontroller perifériájával is. Tehát a kapcsolási rajzot teljes egészében átvehettem, a nyomtatott áramköri rajzolatnál pedig csak a mikrokontroller irányában történő vezetékezést kellett átrajzolni.

5.6. XPORT-interfész

Az XPORT egy kompakt, kis tokba integrált beágyazott eszköz, amelynek segítségével könnyen távolról is elérhetővé lehet tenni egy fejlesztett panelt. A

használatával könnyedén elérhetővé válik akár az internetről is az eszköz, illetve segítheti a fejlesztett panel kódjának hibakeresését is. [19]

Fizikai mérete igen kicsi, lényegében egy RJ45-ös csatlakozóba integrálták bele az összes, a hálózati interfészek kezeléséhez szükséges hardvert. Funkcióit két részre célszerű csoportosítani a kommunikációs irányok szerint. Egyik irányban képes kezelni a 10 MB/s, illetve 100 MB/s sebességű ethernetes hálózatot, továbbá teljes TCP/IP protokollveremmel is rendelkezik, valamint web- és e-mail szerver lehetőséget is biztosít. A másik irányban pedig egy egyszerű kétvezetékes soros UART-interfészt biztosít, illetve három egyéb szabadon felhasználható GPIO-lábat.

Az XPORT alkalmazásával a cél az volt, hogy a fejlesztés során távolról könnyen elérhetővé váljon az intelligens hálózati kapcsoló, és már a fejlesztés során is lehessen távolról vezérelni. Lényegében így elérhető, hogy a központi feldolgozóegység UART-perifériáján keresztül is fogadhasson parancsokat, miközben a felhasználó ugyanúgy Ethernet-alapú hálózaton keresztül érheti el az eszközt.

Az XPORT a fejlesztés és a tesztelés szempontjából is fontos lépés, mert a mikrokontrolleren egy UART-interfész könnyedén felkonfigurálható az XPORT számára is, de a teljes Ethernet-modul beállítása és a protokollverem megírása nehezebb és tovább tart. Így, amíg nem teljes mértékben működőképes az Ethernet-interfész, addig is már lehetséges teljes funkcionalitásában használni a kapcsolót.

Fontos kiemelni, hogy ez ideiglenes megoldás, amelynek célja, hogy lerövidítse a fő funkciók fejlesztésének idejét. A végső cél természetesen az, hogy a mikrokontroller Ethernet-perifériáján keresztül fogadhassa a parancsokat.

5.7. CAN fizikai meghajtó

A CAN fizikai meghajtó alapfunkciója az illesztőmodul és a külvilág közötti megfelelő kapcsolat biztosítása. Ezen felül elvégzi a jelek megfelelő formálását, a szabvány szerinti jelszintek és lezárások kialakítását, illetve védelmet is nyújt a kapcsolóeszköz számára.

A fizikai meghajtó modult lényegében három részre lehet bontani a funkciók alapján: a csatlakozóra, az opcionális 120 Ω-os lezáró ellenállásra, és a védelmekre.

Csatlakozóként a CAN- és FlexRay-hálózatokon is gyakran alkalmazott DSUB9-es csatlakozó lett kiválasztva, de az ikercsatlakozós kialakítású, és nem a

hagyományos egycsatlakozós megoldás. Az ikercsatlakozónál két egyforma DSUB9-es csatlakozó található egymás tetején, mindkettőnek mind a kilenc lába ki van vezetve a csatlakozó alján. A választás fő oka, hogy csak így tartható be a panel szélességére méretére vonatkozó követelmény. A választás további előnye, hogy a CAN hálózati illesztő kialakításánál megvalósított architektúrához illeszkedik az ikercsatlakozós kivezetés. A választás hátránya, hogy a magas csatlakozók miatt két egységnyi (2U) magasság szükséges, szerencsére a szerverszekrényben ez nem jelent problémát

A CAN-hálózat szabványa [14] előírja, hogy a busz a két végponton egy-egy 120 Ω-os ellenállással kell lezárni. A kívánt felhasználási mód során a kapcsolóeszköz funkcionálisan a hálózat egy középpontjának tekinthető, és csak forgalomtovábbító szerepű, de fizikai szempontból mind a két irányban végpontnak minősül, így a szabvány alapján szükséges a lezáró ellenállás az adatvonalak között. Külön kérésre azonban a 120 Ω-os ellenállás csak opcionális lehetőség, így eltérő környezet esetén könnyedén kiiktatható ez az ellenállás.

A fizikai meghajtó igen fontos feladata a panel védelme a rácsatlakoztatott CAN-hálózatról érkező nem kívánatos jelváltozások és jelszintek ellen. Közvetlenül a csatlakozók után, minden CAN-buszon található egy-egy PESD1CAN-215 típusú védődióda, amely az ESD védelmet biztosítja, tehát képes megvédeni a kapcsolóeszközt az elektromos kisülésektől (ESD), valamint a túlfeszültségektől is. A védődiódák után minden CAN-buszon található egy-egy RC-szűrő is, amely a nagyfrekvenciás zavarok ellen védi az eszközt. Végül a differenciális vezetékek között két csatolt tekercs teszi védetté tesz a közös módusú zavarok ellen is a CAN buszokat.

Összességében elmondható, hogy a kialakított fizikai meghajtó biztosítja a megfelelő védelmeket és lezárásokat, így a CAN-illesztő modulhoz a megfelelő jelek érhetnek csak el.

5.8. CAN-illesztő

A CAN-illesztőmodul legfontosabb feladata, hogy flexibilisen kösse össze a központi feldolgozóegységet az összes kezelt CAN-hálózattal úgy, hogy közben a hardveres és funkcionális követelmények is teljesíthetők legyenek.

Fizikai felépítése alapján a CAN-illesztőmodult három részre lehet bontani: az első a CAN adó-vevők, a második a láncolt jelvezetékek az adó-vevők között, a harmadik pedig a feszültségszint-illesztők.

CAN adó-vevőként az AMIS-42700 típusú integrált áramkör lett felhasználva az intelligens kapcsolóeszközben. Ez egy kettős, nagysebességű CAN adó-vevő, amely egyszerre képes két CAN fizikai buszt is kezelni, míg a vezérléséhez egyetlen CAN-periféria is elegendő a központi feldolgozóegységben. Mindkét csatornáján megvalósítja akár az ISO 11898-2-es szabvány szerinti maximális 1 Mbit/s sebességű CAN adatvonal-kezelést is, 12, illetve 24 V-os rendszerben is. Ezen felül beépített védelmekkel is rendelkezik, amely óvja az áramkört külső meghibásodások esetén (pl.: rövidzár). Ezt az adó-vevőt kifejezetten arra tervezték, hogy kapcsolatot teremtsen egy CAN-vezérlőegység, és két fizikai hálózat között. [20]

Az adó-vevőnek három működési módja is lehetséges. Az első az elosztó mód, amelynél külső vezérlés nélkül is képes a hozzá csatlakoztatott két CAN-hálózat között az adatok átjátszására bármely irányban. A második a duális meghajtó, amelynél a CAN-vezérlő által küldött adatokat a CAN-szabványnak megfelelően továbbítja mind a két buszára. A harmadik pedig a bővítő funkció, amikor több AMIS-42700 típusú CAN adó-vevő is összeköthető, és így egyetlen CAN-vezérlő segítségével nem csak két, hanem több CAN-hálózat is vezérelhető. Sajátságos belső kialakításának, és vezérlési felületének köszönhetően mind a három funkció egyszerre is használható.

A különböző üzemmódok lehetőségei miatt igen flexibilisen vezérelhető az AMIS-42700 CAN adó-vevő, és emiatt sok vezérlési be- és kimenettel rendelkezik. A két CAN-busz meghajtója külön engedélyezőjelek segítségével tiltható, vagy engedélyezhető a vezérlőegységből. Engedélyezett buszmeghajtók esetén az olvasás a dedikált RX0 lábon, az írás pedig a TX0 lábon keresztül lehetséges. A bővítő funkció miatt további két vezérlőláb egy TEXT és egy RINT is található, amelyek segítségével lehetséges több AMIS-42700 adó-vevő összekapcsolása úgy, hogy azok együtt szinkronban kezeljék a két buszon lévő adatforgalmat. Érdemes megemlíteni, hogy az AMIS-42700 beépített visszacsatolási elnyomással rendelkezik, amely megfelelő időzítés után letiltja a buszon lévő jelszint mérését, és így garantálja, hogy a meghajtott buszról visszamért jeleket nem továbbítja végtelen ideig, azaz nem okoznak zárolt állapotot az elosztó funkció miatt bekövetkező jelismétlések miatt. Fontos kiemelni,

hogy a több helyről történő (pl.: a vezérlő, és a szomszédos adó-vevő irányából történő) együttes meghajtás a CAN-hálózat esetében nem jelent gondot. [20]

A három üzemmód, és a flexibilis vezérelhetőség miatt a kettős CAN adó-vevő könnyen lehetővé teszi a hardveres elosztó funkció megvalósítását, segíti a kapcsoló funkció szoftveres megvalósítását is, valamint jól illeszkedik a CAN fizikai meghajtóknál alkalmazott ikercsatlakozós megoldáshoz is.

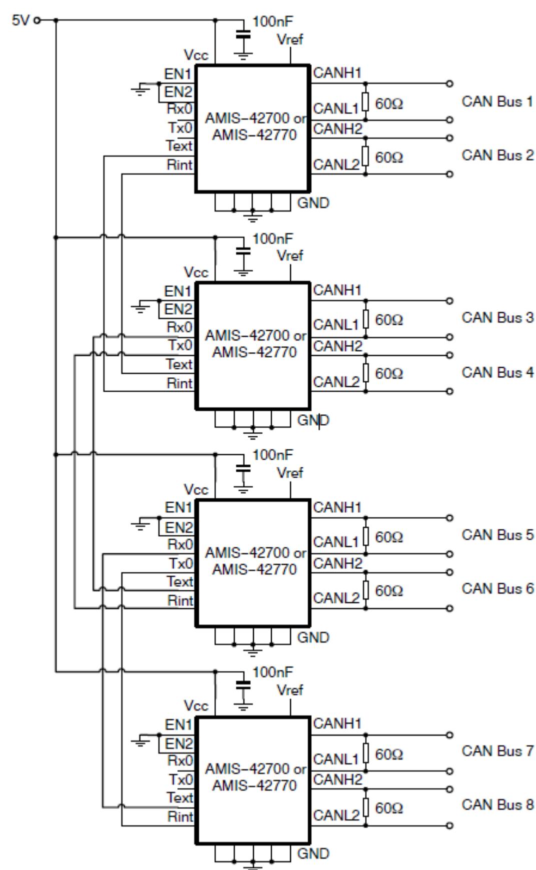
Fontos kitérni egy korlátozásra, amely az adó-vevő belső kialakításából adódik. Több AMIS-42700-s adó-vevő együttes, összekapcsolt alkalmazásakor a kialakított busztopológia nem képes a leggyorsabb, 1 Mbit/s sebességű CAN hálózati átvitelre, mert az adó-vevők összekapcsolása nem a CAN-busz oldalán, hanem csak a feldolgozási oldalon lehetséges. Tehát átjátszáskor mind a fogadó, mind pedig a küldő adó-vevő is késleltetést okoz, amely az adatlap alapján, legrosszabb esetben 230 ns idejű lehet. [21] Egy átlagos rézvezetéken az elektromos jel, a vezeték hullámimpedanciájának függvényében, átlagosan a fénysebesség 0.59-0.77-szeresével képes csak terjedni [22]. Az adó-vevők által okozott késleltetés így átszámolható hosszmértékre, amelyet a vezetéken a jel megtenne a késleltetési idő alatt, az alábbi képlettel, ahol c a fénysebesség, t_{prop} pedig a késleltetési idő, terjedési sebességnek pedig a legrosszabb 0.59-es értéket vehetjük a számításhoz.

$$s = 0.59 \cdot c \cdot t_{prop} = 0.59 \cdot 299\,792\,458\ \frac{m}{s} \cdot 230 \cdot 10^{-9}\ \mathrm{s} \cong 40.681\ \mathrm{m}$$

Látható, hogy az adó-vevők késleltetése több mint 40 m kábelen át történő terjedésnek feleltelhető meg legrosszabb esetben. A CAN-szabvány szerint az 1 Mbit/s-s sebességű CAN-hálózat végpontjai között maximum 40 m távolság megengedett, így belátható, hogy az adó-vevő átjátszási késleltetései miatt nem garantálható az átjátszás a leggyorsabb sebességen üzemelő CAN-hálózat esetén. [14] Természetesen ez nem azt jelenti, hogy a kapcsolóeszköz nem képes az 1 Mbit/s-s sebességű CAN-hálózat kezelésére, mert a vétel és az adás szétválasztásával, azaz a központi feldolgozóegység közbeiktatásával nem hardveresen, de szoftveres úton lehetséges a maximális sebességű hálózatok kezelése is. Az is látható az eredményből, hogy a késleltetés éppen csak az elméleti határ környékén van, emiatt egy adó-vevő által kezelt két csatornán a mérések során lehetséges volt az 1 Mbit/s-os sebességű CAN-hálózat átjátszása is rövid hozzávezetések esetén, a CAN hálózati protokoll robusztussága és tartalékai miatt.

A nyolc CAN-hálózat kezeléséhez összesen négy AMIS-42700 kettős CAN adó-vevőre van szükség, amelyek a bővítő funkciót kihasználva láncolt jelvezeték (daisy chain) segítségével kapcsolódnak össze. [21]

Az állandó és „merev” összekötéshez elegendő lenne a szomszédos adó-vevők között a meghajtó RX0, TX0 és a bővítő RINT, TEXT vezérlőlábakat az adatlapban leírtaknak megfelelően összekötni, továbbá a buszmeghajtók engedélyezőjeleit a földre kötni. Ezzel elérhető, hogy az összeláncolt adó-vevők minden csatornája ugyanarra a CAN-hálózatra csatlakozzon, és bármelyik csatlakozóról érkező adatot átjátsszák az összes többi csatlakozóra is. Ezzel a kialakítással egyszerűen megvalósul az elosztó (hub) funkció. Ez a kialakítás látható az 5-2. ábrán is.



5-2. ábra: Állandó láncolt jelvezeték kapcsolási rajza [3]

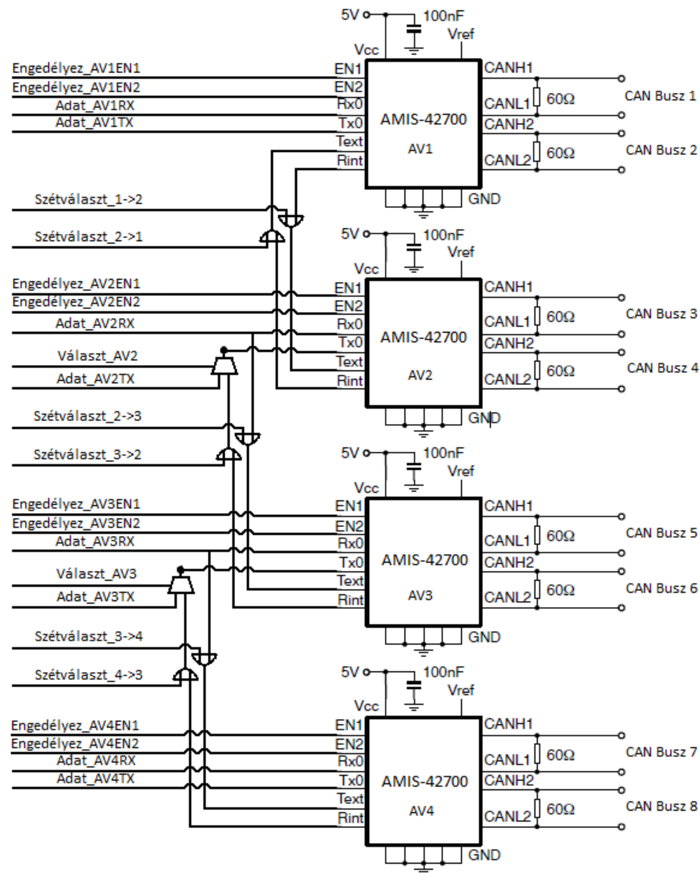
Az intelligens kapcsolóeszköz fő feladata azonban nem az elosztás, hanem a kapcsolás. Ez könnyedén megoldható az engedélyező lábak központi feldolgozóegységről történő vezérlésével. Fontos azonban tisztázni, hogy ilyen értelemben nem a hagyományosnak mondható kapcsolás valósul meg, mert ebben az esetben előre tudni kell a forrás és a cél irányát is, és az adó-vevők ezen buszmeghajtóit

előre engedélyezni kell. Vagyis nem az üzenetben szerepel az üzenet címzettje, és a bejövő adat alapján történik a kapcsolás. Lényegében a busztopológia módosításával lehetséges előre meghatározott eszközök között az adatátvitelt biztosítani. A célfunkció megvalósításához viszont ez is elegendő, hiszen a tesztkörnyezetben előre ismertek és állandóak a tesztelendő eszköz és a tesztrendszerek címei, elhelyezkedései. Az általános felhasználási mód esetén elegendő pont-pont összekötetést megvalósítani, amihez elég csak kettő, a forrás- és a célcsatlakozót irányító buszmeghajtót engedélyezni. Több meghajtó engedélyezésével akár többesadás is megvalósítható, az összes engedélyezésével pedig akár adatszórás is. Belátható tehát, hogy ezekhez a funkciókhoz nem szükséges az adó-vevő architektúra megváltoztatása, csak az engedélyező jelek vezérlését kell megoldani.

A követelmények között szerepel, hogy az eszköznek képesnek kell lennie egyszerre akár több elkülönített hálózat közötti adatforgalom kezelésére a flexibilis felhasználás érdekében, emiatt az adó-vevők közötti összekötéseknek felszakíthatónak kell lenniük. Ez a felszakítás a CAN-hálózat alaptulajdonsága miatt egy egyszerű VAGY kapu segítségével megvalósítható. Egy CAN-hálózatban a domináns (logikai 0) jelérték mindig „nyer” a recesszív (logikai 1) jelértékkel szemben, így ha a láncolt jelvezeték egy adott összekötésénél folyamatosan logikai 1, azaz recesszív jelérték továbbítódik két adó-vevő között, akkor azok egymástól függetlenül tudnak működni. Emiatt ha egy VAGY kapu egyik bemenetére a láncolt jelvezetéket kötjük be, míg a másik bemenetét a központi feldolgozóegység vezéreli az egyik általános célú lábán keresztül, akkor megvalósul a vezérelhetően felszakítható összeköttetés. Ezzel a megoldással már megvalósítható, hogy szabadon változtatható módon szeparálhatók legyenek a CAN hálózati csatlakozások. Természetesen egy adó-vevő által vezérelt két csatlakozásra ez nem igaz, csak külön adó-vevők által vezérelt csatlakozókra.

Az összetettebb, szoftverből megvalósítható funkciók miatt azonban szükséges az is, hogy az összes adó-vevő a központi feldolgozóegységből is vezérelhető legyen. Az adó-vevők RX0 lábai képesek két logikai bemenetet is meghajtani egyszerre, emiatt a fogadási irányban nem szükséges változtatás, csupán a láncba és a feldolgozóegység felé kell behuzalozni ezt a jelvezetéket. A küldési iránynál viszont egyidejű összehajtás is felléphetne a lánc és feldolgozóegység között a középső adó-vevők esetén, emiatt itt szükséges egy multiplexer alkalmazása, amelynek választójelét a központi feldolgozó egység vezérli. Ezzel a kapcsolással lényegében szabadon felhasználhatóvá válik az

összes adó-vevő és szoftverből bármilyen vezérlés megvalósíthatóvá válik. A vezérelhető adó-vevők és a programozottan szétválasztható láncolt jelvezeték együttes kapcsolása látható az alábbi ábrán. A végső áramköri tervekben ez a kapcsolási rajz került megvalósításra, amely az 5-3. ábrán is látható.



5-3. ábra: Felszakítható és vezérelhető láncolt jelvezeték kapcsolási rajza

Fontos hangsúlyozni, hogy a VAGY kapuk és multiplexerek vezérlőjeleinél lehúzó ellenállások biztosítják, hogy az eszköz a processzoron futó kód nélkül is definit állapotban legyen.

Az AMIS-42700 CAN adó-vevő 5 V-os tápfeszültséget igényel, és ennek megfelelően 5 V-os feszültségszintű vezérlőjeleket is. A központi feldolgozóegységként alkalmazott mikrokontroller azonban csak 3.3 V-os kimeneti jelszintet képes előállítani. Emiatt szükséges a láncolt jelvezetékben lévő adó-vevők és a mikrokontroller közé egy jelszintillesztő egység, amely áthidalja ezt a különbséget. A panelen ezt a Texas Instruments TXB0108-as integrált áramköre valósítja meg, amely egyszerre nyolc vezeték kezelésére is képes, mind a két irányban lehetséges vele a jelátvitel, valamint

magától érzékeli, hogy melyik irányba szükséges éppen a meghajtás. Az egyszerűsége és a kétirányúsága miatt esett erre a szintillesztőre a választás.

Összességében elmondható, hogy a CAN-illesztő a négy párhuzamosan működő kettős adó-vevő, az azokat összekötő láncolt jelvezeték, és a megfelelő szintillesztők alkalmazásával egy igen flexibilis, a kívánt hardveres funkciókat megvalósító, és szoftverből könnyen kezelhető CAN-interfészt nyújt az intelligens kapcsoló számára.

5.9. Követelmények hardveres teljesítése

Az előbbiekben igen részletesen bemutatott modulok felépítését és az alkalmazott tervezési megoldásokat követően célszerű összegezni, hogy az eszközzel szemben állított követelmények közül melyeket hogyan teljesíti a megtervezett hardver.

A legtöbb funkcionális követelményt a CAN-illesztőmodul valósítja meg, vagy teszi megvalósíthatóvá a központi egységbe írt szoftver számára. A fizikai kialakítással, valamint a csatlakozási és vezérlési lehetőségekkel kapcsolatos követelményeket pedig a megtervezett hardver elrendezése, és a felhasznált komponensek együttesen valósítják meg.

Egy AMIS-42700 kettős CAN adó-vevő két fizikai CAN-hálózat kezelésére képes. Így az előírt nyolc CAN csatlakozási lehetőséget négy ilyen adó-vevő párhuzamos működtetésével lehet elérni. Ezek összeköttetését egy láncolt jelvezeték struktúrában egyszerű megvalósítani. A megfelelő vezérelhetőség érdekében szükséges ezek elé jelszintillesztők beiktatása, hogy a különböző működési feszültségszintek között lehetséges legyen a jelátvitel. Ezzel a három elemi résszel összegezhető az architektúra.

A célfunkciót, mely szerint bármelyik hálózati csatlakozóján beérkező kommunikációt minden egyéb csatlakozóra képes legyen továbbítani, a választott AMIS-42700 kettős CAN adó-vevő automatikus átjátszási, és bővíthetőségi funkcióit kihasználva a láncolt jelvezeték struktúra kialakításával lehetséges elérni. Az általános felhasználási mód esetén pont-pont kapcsolatot kell létrehozni, amelyet a feldolgozóegységből vezérelt engedélyezőjelek segítségével lehet megvalósítani. A flexibilis felhasználhatóság érdekében a többesadás és az adatszórás is ugyanígy megoldható.

Az egyéb funkcionális követelmények között szerepel, hogy az eszköz hardveresen képes legyen egy hálózati elosztó szerepét betölteni. A megvalósított architektúrával ez lehetséges, és a megfelelő vezérlési lábakon alkalmazott lehúzó ellenállások miatt csak az engedélyező bemenetek 0 jelszintre állításával, azonnal képes is az eszköz hardveresen, a központi feldolgozóegység beavatkozása nélkül erre a funkcióra.

A további funkciókat és összetett műveleteket pedig majd szoftver segítségével lehet megvalósítani. Szoftverből egyrészt lehetőség van a láncolt jelvezetékek programozható átalakítására, amelyekkel így flexibilisen változtathatók az összeköttetések az egyes adó-vevők között. Másrészről a központi feldolgozóegység szoftvere akár közvetlenül vezérelheti az összes adó-vevőt (fogadhat, illetve küldhet adatot), így szabadon vezérelheti az összes CAN-hálózatot bármilyen igény szerint egy megfelelően megírt szoftver segítségével.

A panel összességében pedig teljesíti az előírt vezérlési és csatlakozási lehetőségeket. Kialakításra került az előírt nyolc független CAN csatlakozási lehetőség a CAN fizikai meghajtóban. A nyolc FlexRay-csatlakozás is elérhető a FlexRay fizikai meghajtó által. A távolról történő vezérelhetőségről ideiglenesen az XPORT alkalmazása, és az UART-modul felélesztése és konfigurálása gondoskodik, későbbiekben pedig az Ethernet-modul, illetve a TCP/IP protokollverem integrálásával válik véglegessé ez a lehetőség. A megtervezett panelre pedig felkerültek a szükséges közvetlen felhasználói felület elemei is.

A megtervezett nyomtatott áramköri panel teljesíti a fizikai méretre és elhelyezkedésére vonatkozó kritériumokat, vagyis egy illesztődobozban betehető a 19 colos szerverszekrénybe, és az adatkapcsolati hálózatok csatlakozói a panel elülső oldalán helyezkednek el. Az egyes modulok panelen történő helyének megfelelő megválasztásával minden hálózat a maximális sebességen is működőképes.

Összességében jól látható, hogy a legtöbb követelményt a megtervezett hardver is teljesíti, és csak az összetettebb funkciókat szükséges szoftverből megvalósítani.

6. A szoftvertervezés lépései

Az előző fejezetekben már ismertettem az intelligens kapcsolóeszközzel szemben támasztott követelményeket, illetve bemutattam azt is, hogy a hardveres megvalósítások hogyan teljesítik ezek többségét. Azonban a két magasabb szintű funkció (a hibainjektálás és a forgalomnaplózás) már szoftveres megvalósítást igényel. Emiatt a megvalósítás során felmerült konkrét kérdések és problémák, valamint az azokra adott megoldások ismertetésének második lépéseként a diplomatervezés során elkészült szoftvert, és a tervezése során alkalmazott megoldásokat is be fogom mutatni ebben a fejezetben.

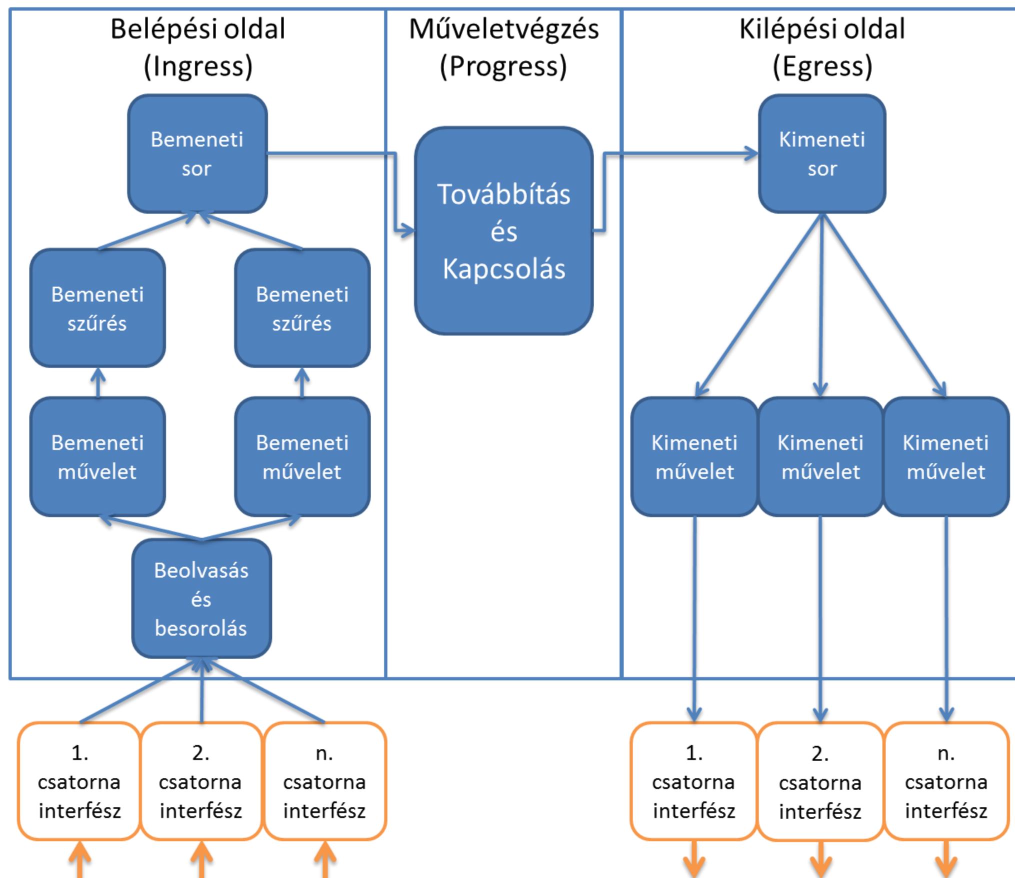
Az elkészült szoftver igen sok kódsorból áll, és minden részletére nincs lehetőségem kitérni a diplomaterv során. Emiatt először a magas szintű szoftverterv átfogó egészét ismertetem, utána pedig csupán a megvalósítás legfontosabb lépéseire térek ki részletesebben.

6.1. A szoftverarchitektúra

Manapság az ethernetes hálózatokban már igen kiforrott technológiákat alkalmaznak, és emiatt az ISO-OSI második rétegében működő kapcsolók szoftverarchitektúrája is jól bevált egységes felépítést követ.

Ez az architektúra három jól elkülöníthető részre osztható: a belépési oldali feladatokra (Ingress Functions), a műveletvégzésre (Progress Functions) és a kilépési oldali feladatokra (Egress Functions). A belépési oldalon először fogadni kell az üzeneteket a csatornákról, illetve a kapcsolási tábla alapján el kell dönteni, hogy milyen műveletek elvégzése szükséges a fogadott csomagon. A besorolás alapján továbbkerül a csomag a megfelelő feldolgozó egységhez, amely átalakíthatja a fogadott csomagot a belső működési protokollnak és jelszinteknek megfelelően, valamint végrehajthat egyéb szükséges műveletet, ha módosítani kell az adaton, illetve ki is szűrheti, ha el kell dobni, vagy ha a terheltség miatt nem képes a további feldolgozásra. A belépési oldalon végül egy sorba (Ingress Queue) kerül a fogadott és átalakított adatcsomag. A műveletvégző az implementált kapcsolási algoritmus és a kapcsolási tábla alapján a szükséges időzítés szerint továbbítja, azaz kapcsolja az üzenetet a kimenet felé. A kimeneti oldalon először szintén egy sorba (Egress Queue) kerül az üzenet. A kimeneti

sorból kikerülve még a kimeneti oldalon is lehetséges különböző adatmanipulációs műveletek elvégzése a kimeneti feldolgozóegységek segítségével. A kilépési oldal a sorból történő kivétel, illetve a szükséges átalakítások után végül kiküldi az adatot a kapcsolási iránynak megfelelő fizika csatornán [23] [24]. Ennek az architektúráját mutatja be a 6-1. ábra.

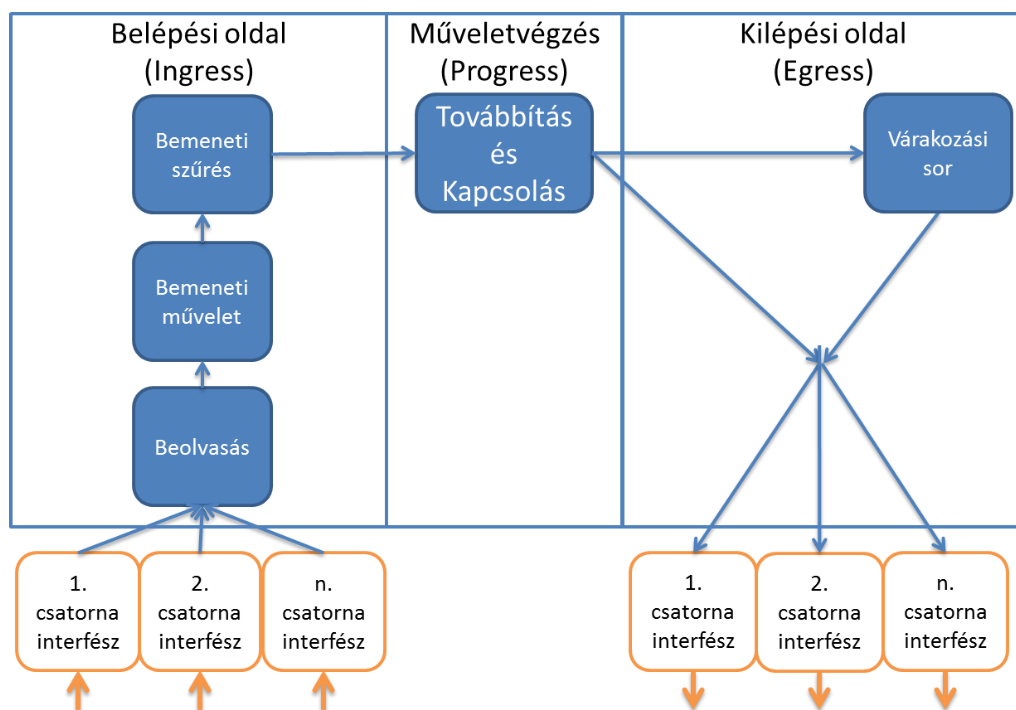


6-1. ábra: Kapcsoló általános szoftverarchitektúrája [4]

Érdemes megemlíteni, hogy a belépési és a kilépési sorokat gyakran összevonják, és ezen az egy soron keresztül történik az adatáramlás a bemenetről a kimenet felé. Ilyenkor a műveletvégző a sorban lévő elemeken tud dolgozni a kapcsolási algoritmus és a kapcsolási tábla alapján.

A CAN intelligens hálózati kapcsoló számára ez az architektúra megfelelő, azonban a CAN-hálózat és a speciális követelmények miatt jelentős egyszerűsítéseket is lehet alkalmazni a szoftverben. A CAN-protokollnak megfelelően a mikrokontroller perifériája csak a hibátlanul megérkezett csomagokat fogadja a hálózatról, és rögtön egy FIFO sorba teszi a memóriában. Tehát a belépési oldalon a beolvasást már a periféria elvégzi, és azonnal a teljes csomagtartalom rendelkezésre áll. Így rendkívül egyszerűen

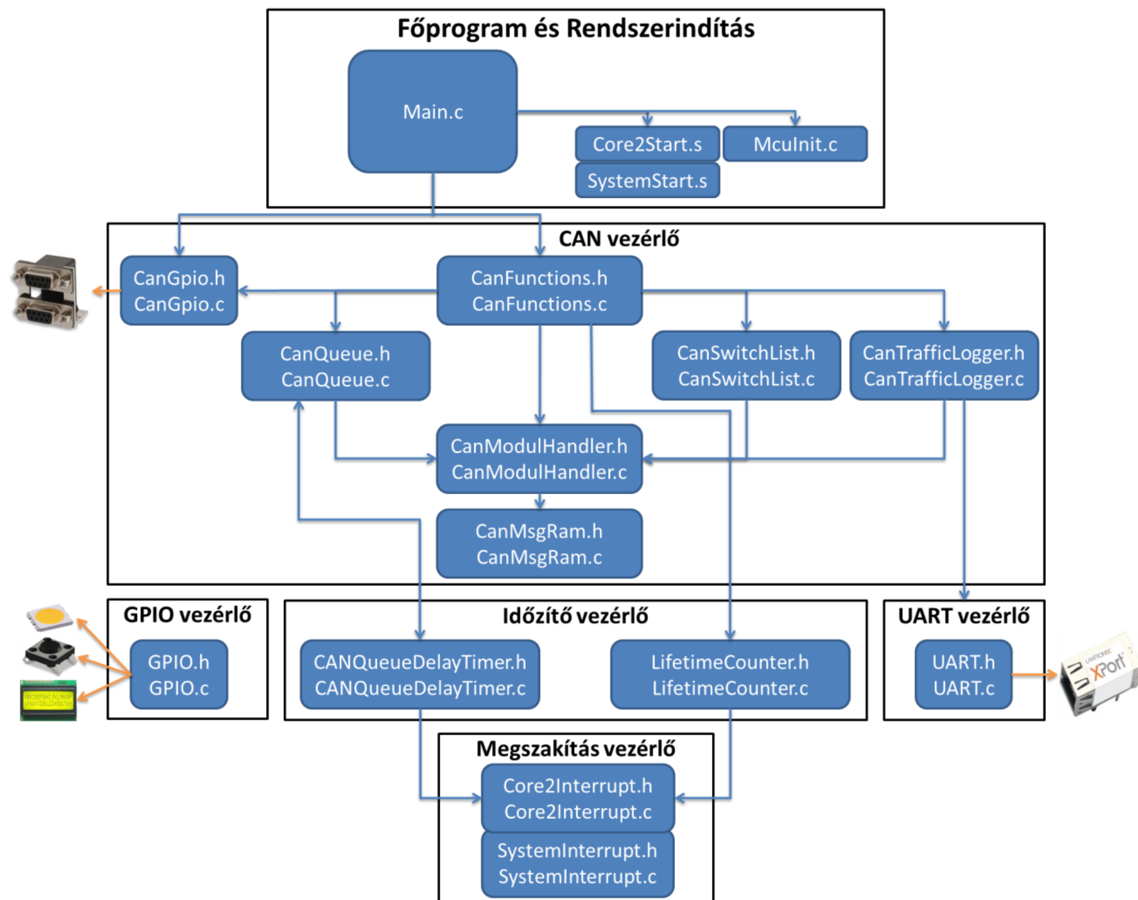
és közvetlenül elvégezhetők a bemeneti adatmanipulációs feladatok. Az adatokat nem szükséges külön feldolgozóegységekhez továbbítani, mert a mikrokontroller közvetlenül is elérheti azokat. A hagyományos kapcsolókkal ellentétben jelen felhasználás során a kapcsolási tábla statikus, és már a tervezési, fordítási időben ismertek az egyes végpontok irányai, amelyek között kapcsolni kell. Emiatt a műveletvégzés és -továbbítás is igen egyszerűen közvetlenül az adat beérkezésekor megoldható, hiszen az üzenetcsomag teljes egésze és a kapcsolási tábla is rögtön a csomag beérkezésekor a rendelkezésre áll, így nem szükséges bemeneti sor alkalmazása sem. A kilépési oldalon a követelmények alapján nem szükséges adatmanipuláció elvégzése, minden szükséges funkció a bemeneti oldalon megvalósítható. A kimeneteken is minden csatornához biztosít egy-egy FIFO sort a CAN-periféria, így az adatok közvetlenül és azonnal továbbíthatók kifelé, és a kisebb torlódások, üzenetfelhalmozódások sem jelentenek gondot. Egyedül a késleltetés megvalósításához szükséges egy várakozási sor. Így tehát vagy a kapcsolási műveletvégzés után vagy közvetlenül a megfelelő kimeneti csatorna periféria FIFO sorába vagy pedig a várakozási sorba lehetséges azonnal továbbítani az üzeneteket. Természetesen a várakozás lejárta utána a sorból egy megszakítási rutin segítségével lehetséges a késleltetett üzenetek továbbítása a kimenet felé. A CAN intelligens hálózati kapcsoló szoftverarchitektúrája a 6-2. ábrán látható.



6-2. ábra: Az intelligens CAN hálózati kapcsoló szoftverarchitektúrája

6.2. A szoftverterv blokkvázlata

A szoftverterv és az elkészült szoftver blokkvázlata a 6-3. ábrán látható.



6-3. ábra: Az elkészült szoftver blokkvázlata

A blokkvázlaton fekete keretekben láthatók a szoftvermodulok, amelyek az egyes felhasznált perifériák kezeléséért, illetve a magas szintű funkciók megvalósításáért felelnek. A kék színű téglalapokban az egyes assembly- (.s), illetve C programozási nyelvű fejléc- (.h) és forrásfájlok (.c) láthatóak, amelyek a diplomatervezés során elkészültek. Az ábrán a nyilak az egyes fájlok közötti függőséget jelölik: a nyíl kezdetén lévő fájl igényli a nyíl végén lévő állomány fejlécét.

A megszakításvezérlő felelős a mikrokontroller összes rendszerszintű kivétel- és a periféria processzormag megszakításkezeléséért. Egyrészről lehetőséget biztosít a kivétel- és megszakításkezelő periféria megfelelő kezdőállapotú beállítására, másrészről pedig definiálja mind a rendszerszintű kivételtáblát, mind pedig a processzorszintű megszakítástáblát, harmadrészt pedig a programfutást megakasztó függvényekkel ki is

tölti a táblákat, amelyeket a felhasználó felülírhat bármelyik fájlban, ha szükségesnek tartja lekezelni az adott megszakítást.

Az időzítő vezérlő a mikrokontrollerben lévő periodikus számláló perifériák kezelését végzi, illetve megfelelő magas szintű időzítési funkciókat biztosít a többi szoftvermodul számára. A modul biztosít egy 64 bites élettartam-számlálót, két 32 bites számláló megfelelő kezelésével, amely a rendszer elindítása után eltelt órajeleket számolja, és aktuális értéke lekérdezhető. Így könnyedén lehetséges időkülönbségek mérése nagy pontossággal, illetve ez az időbélyeg készítésére is felhasználható. A késleltetés mint magasszintű CAN-funkció megvalósításához szükséges egy különálló időzítési rendszer kialakítása. A modul egy különálló periodikus számláló felhasználásával és a számláló megszakítási rutinjával biztosítja a CAN-vezérlő számára a megfelelő időzítések könnyű kialakítását is.

A GPIO-vezérlő felelős a közvetlen felhasználói felület, vagyis a nyomógombok és LED-ek, illetve az LCD-kijelző kezeléséért. A végső program jelenleg nem használja ezeket a funkciókat, de a nyomógombok és LED-ek segítették a fejlesztést, az LCD-kijelzőn pedig későbbiekben tetszőleges információ megjeleníthető a tesztrendszerben.

Az UART-vezérlő felelős az UART-periféria megfelelő kezeléséért, valamint küldési és fogadási függvényeket biztosít különböző alapvető adattípushoz (pl.: 8, 16, 32 bites számok, valamint ezekből álló tömbök átvitelére). A felhasznált UART-periféria jelenleg az XPORT-interfész felé vezet a mikrokontrollerből, vagyis e modul segítségével lehetséges adatok átvitele és fogadása ethernetes hálózatokból, és így megoldható egy személyi számítógéppel való kapcsolattartás is. A forgalomnaplózás mint magasszintű CAN-funkció számára szükséges ez, hiszen így lehetséges az intelligens kapcsolóeszközön áthaladt üzenetek naplózása.

A diplomatervezés során elkészült szoftver legfontosabb része a CAN vezérlőmodul. Egyrészről ezen a modulon belül lehetséges a megtervezett panelen a hardveres funkciók ki- és bekapcsolása, valamint megfelelő beállítása. Másrészről ez a modul felelős a mikrokontroller CAN-perifériáinak alacsonyszintű kezeléséért. Harmadrészt pedig a magasszintű CAN-funkciókat is ez a modul biztosítja.

A főprogram és rendszerindítás modul adja az összes többi szoftvermodul működésének alapját, illetve ebből a modulból lehetséges az egyes funkciók, függvények meghívása. Egyrészről itt találhatóak a mikrokontroller, illetve a

processzormagok elindításához és alaphelyzetbe állításához szükséges assembly kódok, amelyek lefutása után válik csak működőképessé az eszköz. Továbbá ez a modul felelős a legkritikusabb, és minden alkalmazásban szükséges (órajelgeneráló és PLL) perifériák kezeléséért, és az alkalmazás számára megfelelő alaphelyzetbe állításáért. Végül itt található meg a felhasználói program belépési pontja is, vagyis a main függvény, amelyből az összes megvalósított funkció meghívható.

A blokkvázlaton látható, hogy a diplomaterv számára a legfontosabb a CAN vezérlőmodul, és a főprogram csak ennek a modulnak a funkcióit hívja meg közvetlenül. Természetesen a többi modul funkciói is elérhetők, de a diplomatervezés során ezek csak kisegítő szerepűek.

6.3. CAN vezérlőmodul

A CAN-vezérlő szoftvermodul a legfőbb szoftverkomponense az intelligens CAN hálózati kapcsolónak. Ez a szoftvermodul valósítja meg az összes olyan funkcionális követelményt, amelyet hardveresen bonyolult lett volna megoldani.

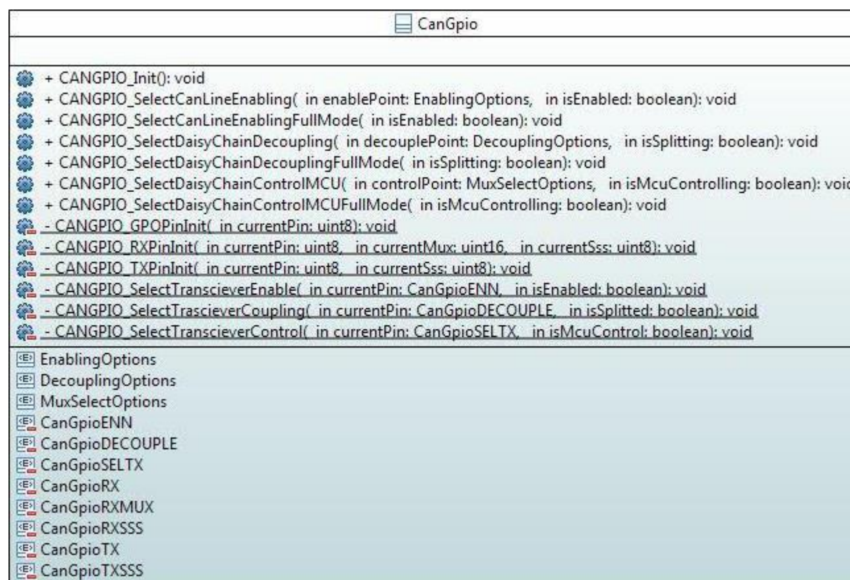
A megtervezett eszköz célfunkciója az, hogy egy beérkező adatcsomagot irányítottan csak egy adott célcsatlakozóra vagy csatlakozókra továbbítson, és ez két módon is megvalósítható. Az első megoldás az, hogy csak a forrás és a vevő adó-vevőit engedélyezi az eszköz, miközben a láncolt jelvezeték teljes hosszában engedélyezett, azaz így közvetlenül hardveresen történik az adattovábbítás a mikrokontrolleren futó funkciók nélkül. A második megoldás az, hogy a mikrokontroller teljes egészében felszakítja a láncolt jelvezetéket és a központi egységen keresztül szoftveresen történik az adattovábbítás. Az első megoldás előnye, hogy rendkívül gyors kapcsolást tehet lehetővé a hardveres megoldás miatt, azonban semmilyen egyéb magasabb szintű funkció nem lesz elérhető. Míg a második esetben bármelyik funkció használható, viszont a szoftveres megoldás miatt előreláthatóan lassabb a kapcsolás.

A követelmények és a diplomaterv kiírása alapján két magasabb szintű funkciót: a hibainjektálást és a forgalomnaplózást is meg kell valósítani, amely szintén ennek a modulnak a feladata. Hibainjektálási funkcióként az üzenetek eldobását, megismétlését, késleltetését és tartalmának felülírását kell lehetővé tenni. A forgalomnaplózásnak pedig kikapcsolhatónak kell lennie, ugyanis csak hibakereséskor érdekes az áthaladt forgalom, tehát előreláthatóan ritkán van szükség erre a funkcióra.

A szoftvermodul fontossága miatt a következőkben részletesebben is ismertetem az egyes almodulok által megvalósított feladatokat és az alkalmazott megoldásokat. Az ismertetés része, hogy UML osztálydiagramokként az egyes almodulok által definiált globális változókat, lokális és globális függvényeket, illetve a felsorolásos típusokat és struktúrákat is bemutatom. A program C nyelven íródott, amely nem objektumorientált nyelv, azonban az egyes almodulok fejléc- és forrásállományai logikailag hasonlóan csoportosítják a kódot, mint más nyelveken az osztályok. Az ábrákon publikusnak minősülnek a fejléc állományokban definiált felsorolásos típusok, adatstruktúrák, és az extern kulcsszóval interfészként nyújtott függvények. Privátnak pedig a csak a forrásfájlokban szereplő felsorolásos típusok, adatstruktúrák és függvények.

6.3.1. CAN ki- és bemenetkezelő (CanGpio)

A CanGpio állományok által definiált változók, függvények, felsorolásos típusok és adatstruktúrák az alábbi osztálydiagramon, a 6-4. ábrán láthatók.



6-4. ábra: A CAN ki- és bemenetkezelő osztálydiagramja

A CAN ki- és bemenetkezelő felelős a processzorban implementált szoftver, és a panelen lévő CAN-hardver megfelelő összekötéséért. Lényegében elfedi a panel hardveres függőségeit, és egy flexibilis interfészt ad a többi szoftvermodul számára.

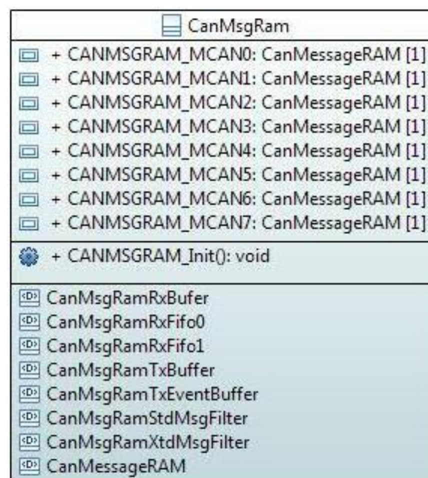
A ki- és bemenetkezelő almodul kettős feladatot hajt végre. Egyrészről a mikrokontroller CAN-perifériáit a megfelelő funkcionalitású processzorlábra köti, és a paneltervek alapján megfelelő helyre vezeti ki azok jeleit. Ez a funkció alacsony szintű, ráadásul változtathatóság sem szükséges, hiszen az adatlap és a panel tervei alapján

egyértelmű, hogy milyen összekötéseket kell megvalósítani egy alaphelyzetbe állító függvénynek. A másik feladata, hogy a hardveres funkciók tiltását, engedélyezését és beállítását lehetővé tegye szoftverből. Ehhez az általános processzorlábakat használja fel, amelyeket három csoportra tagolhatjuk. Az első az adó-vevők letiltásáért vagy engedélyezéséért felel. A második a láncolt jelvezeték felszakításáért vagy összekapcsolását teszi lehetővé. A harmadik pedig a láncban középső adó-vevők küldési irányú bemeneteinek forrását vezérli. Az alaphelyzetbe állító függvény kialakítja a teljes láncolt jelvezetéket, és minden adó-vevőt engedélyez. További függvények biztosítják az egyes adó-vevők vagy az összes együttes tiltását és engedélyezését, valamint a láncolt jelvezeték egy ponton történő vagy teljes szétválasztását, illetve összekapcsolását is. Így magas szintű és flexibilis konfigurációt biztosít a hardveres funkciók alkalmazására a szoftver számára.

A feladatából adódóan ez egy komplexebb egység, hiszen alacsony szinten a processzor lábait is kezelnie kell, miközben magas szinten a panelen meglévő hardveres megoldások szoftveres engedélyezését vagy tiltását is lehetővé teszi.

6.3.2. CAN-üzenet RAM (CanMsgRam)

A CanMsgRam állományok által definiált változók, függvények, felsorolásos típusok és adatstruktúrák az alábbi osztálydiagramon, a 6-5. ábrán láthatók.



6-5. ábra: A CAN-üzenet RAM osztálydiagramja

A CAN-üzenet RAM kialakítja a mikrokontroller CAN-perifériájának szükséges struktúrákat, és helyet foglal a memóriában azoknak, illetve gondoskodik inicializálásukról is. Ezek a struktúrák elengedhetetlenek a CAN-periféria

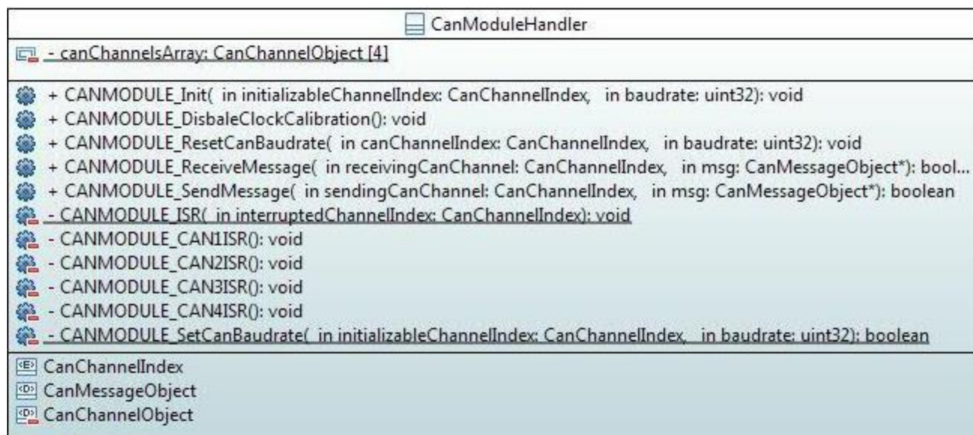
működéséhez, hiszen például ezeken keresztül lehet hozzáférni a fogadott üzenetekhez, vagy a kiküldésre üzeneteket átadni.

A CAN-üzenet RAM almodul a fejléc fájlban definiálja a mikrokontroller adatlapjában leírtak alapján a CAN-perifériákhoz szükséges memóriastruktúrákat, valamint deklarálja és más programmodulok számára elérhetővé teszi a mikrokontroller összes CAN-perifériájához tartozó változókat, és gondoskodik arról is, hogy a megfelelő memóriaterületre kerüljenek. A forrásfájl definiálja ezeket a CAN-perifériák használatához szükséges változókat, és egy alaphelyzetbe állító függvényt is biztosít a megfelelő inicializálás érdekében.

Ez az almodul igen egyszerű, hiszen a CAN-perifériák memóriakezelését valósítja meg, és biztosítja, hogy használhatóvá váljanak más modulok számára.

6.3.3. Alacsonyszintű CAN-vezérlő (CanModuleHandler)

A CanModuleHandler állományok által definiált változók, függvények, felsorolásos típusok és adatstruktúrák az alábbi osztálydiagramon, a 6-6. ábrán láthatók.



6-6. ábra: Az alacsonyszintű CAN-vezérlő osztálydiagramja

Az alacsony szintű CAN-vezérlő célja, hogy elfedje a felsőbb rétegek elől a mikrokontroller CAN-perifériájának függőségeit, és egy egységes könnyen kezelhető interfészt biztosítson a többi programmodul számára.

A vezérlő célja érdekében a CanChannelObject nevű struktúra segítségével összeköti a CAN-periféria struktúrát, a hozzá tartozó üzenetmemóriát, valamint a megszakításkezelési rutint, és kialakít egy egységes leírót, amely egy adott CAN-csatornát szimbolizál. Ennek segítségével megfelelően karbantarthatja a perifériát és a szükséges memóriaterületet is az interfészfunkciók megfelelő biztosítása mellett.

A hardver architektúrájából adódik, hogy négy CAN-periféria kezelése szükséges ahhoz, hogy az összes csatornát használni lehessen. Érdemes kiemelni, hogy a CAN-csatornát leíró struktúrákat egy tömbbe is rendezi, és így egy indexszám segítségével lehet egy-egy csatornát azonosítani. A hibalehetőségek kizárása érdekében pedig a CanChannelIndex felsorolásban külön definiálja az elérhető csatornák neveit és azok sorszámát. Ezzel a megoldással egyrészről elfedhető a CAN-csatornát reprezentáló struktúra felépítése, és így nem szükséges más programrészeknél azok ismerete, másrészről pedig így nem szükséges egy nagyméretű struktúrát paraméterként átadni, hanem egyetlen szám is elegendő a CAN-csatornák azonosításához.

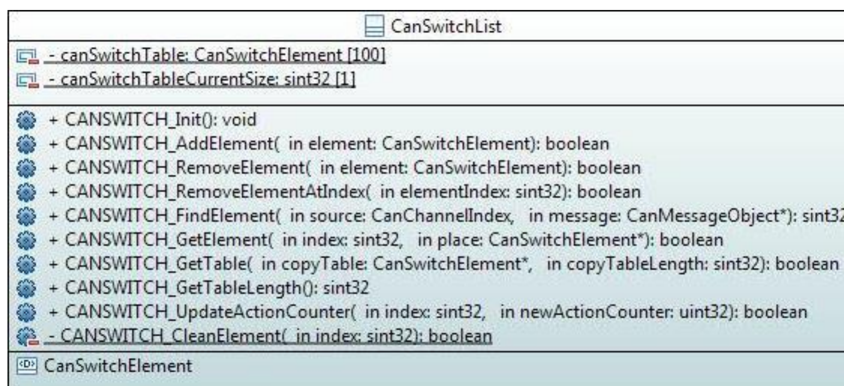
Az egyszerű interfész érdekében egy CanMessageObject struktúra is bevezetésre kerül, amely egy CAN-hálózaton átvihető üzenet minden fontos paraméterének tárolására alkalmas. Ennek a struktúrának része a CAN-azonosító (CANId), egy jelzőbit, amely kifejezi, hogy hagyományos vagy bővített-e az azonosító, egy másik jelző, amely megmutatja, hogy az üzenet kérés típusú-e vagy hagyományos üzenet, az üzenet hossza és az üzenet tartalma, amely maximálisan nyolc bájtnyi lehet. Ez a struktúra más modulokban is fontos és hasznos, hiszen a kapcsoló minden egyéb funkciója ilyen üzenetobjektumokon dolgozik, vagy ezt használja fel. Ez az oka annak, hogy sok más fájl is függ ennek az almodulnak a fejléc fájljától.

Az alacsonyszintű vezérlő interfészeként elérhető egy alaphelyzetbe állító, egy csatorna adatátvitelisebesség-átállító, egy üzenet fogadását, és egy üzenet kiküldését megvalósító függvény. Fontos kiemelni, hogy a CAN-periféria küldési és fogadási irányba is FIFO sorokat használ, így a fogadó és küldő függvény csak a fogadási sor elejéről olvas, míg a küldő függvény csak a küldési sor végére ír. A fogadás vagy a kiküldés mechanizmusát a periféria különálló módon oldja meg, illetve, ha bármelyik irányba betelik a FIFO sor, akkor a további üzenetek elvesznek.

Az alapszintű vezérlő interfész függvényeket biztosít az adatátviteli sebesség beállítására, üzenetek kiküldésére és fogadására bármelyik CAN-csatornán. Ezekkel a függvényekkel minden szükséges CAN hálózati aktivitás flexibilisen megoldható.

6.3.4. CAN kapcsolótábla-kezelő (CanSwitchList)

A CanSwitchList állományok által definiált változók, függvények, felsorolásos típusok és adatstruktúrák az alábbi osztálydiagramon, a 6-7. ábrán láthatók.



6-7. ábra: A CAN kapcsolótábla-kezelő osztálydiagramja

A CAN kapcsolótábla-kezelő felelős a kapcsoláshoz szükséges kapcsolótábla létrehozásáért, karbantartásáért és a kapcsoláshoz szükséges lekérdezőfüggvények biztosításáért, de magát az üzenetek kapcsolását valójában nem ez a kódrészlet végzi el.

Ebben az almodulban kerül definiálásra az egyik legfontosabb struktúra CanSwitchElement néven, amely leírja a kapcsolási táblába tehető bejegyzések részeit. A struktúrában minden lényeges adat szerepel, amely a kapcsolás és a magas szintű funkciók elvégzéséhez szükséges. A felhasználók számára is fontos hogy részletesen ismerjék a felépítését, így érdemes végignézni a kezelő elemeit. A követelmények szerint hibainjektálási funkcióként az üzenetek eldobását, megismétlését, késleltetését és tartalmának felülírását kell lehetővé tenni. Viszont lehetőséget kell biztosítani, hogy ne minden beérkező üzeneten, hanem tetszőleges periodicitású beérkezési ciklusonként hajtódjanak csak végre. A kapcsolótábla-leíró struktúra flexibilitásának következtében lehetővé vált, hogy az egyes funkciók kombinációit is végre lehessen hajtani.

A struktúra első felében azok az adatok szerepelnek, amelyek ahhoz szükségesek, hogy a kapcsolást pontosan le lehessen írni: a forráscsatorna indexe, a kapcsolni kívánt CAN-üzenet azonosítójához tartozó bitminta és maszkminta, a hagyományos vagy bővített azonosító használatát jelző és az üzenet vagy kérés típusát jelző bit, valamint a célcsatorna indexe. A struktúra második felében a kapcsolás közben végrehajtandó hibainjektálási akciók leírásához és megvalósításához szükséges adatok szerepelnek: a hibainjektálási akció periódusa, a perióduson belüli állapot jelzése, a kívánt késleltetési idő, a szükséges ismétlések száma, az ismétlések közötti kívánt időtartam, valamint az üzenettartalom módosításához tartozó bitminta-tömb és maszkminta-tömb és hosszaik.

Az azonosító maszkolás alkalmazásával elérhető, hogy egy bejegyzés ne egy konkrét azonosítóra, hanem egy adott tartományra vonatkozzon. Továbbá az üzenettartalom estén a kimaszkolt bitpozíciókban a bitmintában szereplő bitekre módosul az üzenetben lévő adat.

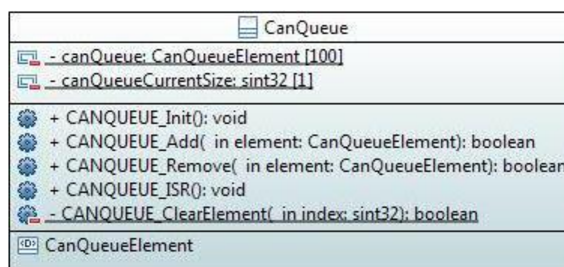
A kapcsolótáblát is itt definiálja a program canSwitchTable néven, illetve mellette külön tárolja a tábla kihasznált hosszát, vagyis az aktuálisan érvényes bejegyzések számát. Maga a tábla statikus módon foglal helyet a memóriában, tehát véges hosszúságú lehet, de egy preprocesszor makroutasítás segítségével könnyedén változtatható a lefoglalt hossz. A tábla egyszerű tömbszervezésű, azaz az új bejegyzéseket a tömb elejétől kezdi el feltölteni, és bejegyzés törlésekor mindig a törölt sor mögötti elemeket a megfelelő pozícióval előrecsúsztatja. Ennek következtében az aktuálisan érvényes bejegyzések mindig a tömb elején folytonosan helyezkednek el, így a cache memória miatt hatékonyan és gyorsan lehetséges kapcsoláskor a szükséges bejegyzés megtalálása és kiolvasása.

Az almodul elérhető kezelőfüggvényekként egy táblakezelésnél általánosan elvárt interfészt nyújt. Az alaphelyzetbe állító függvény törli az összes bejegyzést, és zérus értékekkel tölti fel a táblát. Egy külön függvény új elem hozzáadását is biztosítja a táblához. Törölni egy elemet akár a táblabejegyzés attribútumai, akár a táblában elfoglalt hely alapján is lehetséges. Adott a lehetőség egy bejegyzés táblában elfoglalt pozíciójának meghatározására a forráshely és az üzenetazonosító attribútumok alapján. Továbbá egy adott pozícióban lévő bejegyzés értékei elkérhetők és lemásolhatóak. A teljes tábla, illetve az aktuális bejegyzések száma is lekérdezhető, ami az esetleges listázási műveleteket segítheti. Végül egy külön függvénnyel lehetséges a hibainjektálási perióduson belüli állapotgép léptetése.

Összességében látható, hogy a kapcsolótábla valójában egyszerű tömbszervezésű táblázat, amely sok attribútumos bejegyzéseket tárol. A szervezés gyors kereshetőséget biztosít a táblában, míg a rekordok tárolási módja következtében igen flexibilis hibainjektálás valósítható meg.

6.3.5. CAN várakozásisor-kezelő (CanQueue)

A CanQueue állományok által definiált változók, függvények, felsorolásos típusok és adatstruktúrák az alábbi osztálydiagramon, a 6-8. ábrán láthatók.



6-8. ábra: A CAN várakozásisor-kezelő osztálydiagramja

A CAN várakozásisor-kezelő almodul, a késleltetés funkció megvalósításához szükséges FIFO definiálását, deklarálását és megfelelő kezelését biztosítja.

A késleltetési sor a kilépési oldalon található a blokkvázlaton, ennek megfelelően a hibainjektálás miatt esetlegesen módosított üzenetek kerülhetnek bele, amelyeknek a célcsatlakozója is már eldöntött és ismert. Ezek alapján definiálja ez a rész a CanQueueElement nevű struktúrát, amelyek a késleltetni kívánt üzeneteket írják le. A struktúra része: a szükséges késleltetési idő, a periféria-órajel periódusidejében kifejezve, maga az üzenetleíró és a célcsatorna indexe. Ezzel az egyszerű struktúrával a késleltetés feladata leírható.

Magát a késleltetési FIFO sort canQueue néven hozza létre a program, emellett egy számlálóban az aktuálisan a sorban lévő üzenetek számát is tárolja. A sor statikus módon foglal helyet a memóriában, tehát a maximális mérete véges, de a mérete könnyedén módosítható az igények alapján egy preprocesszor utasítással. A sor alapvetően tömbszervezésű, azonban a benne lévő elemek a hátralévő szükséges késleltetési idő szerint csökkenő sorrendben helyezkednek el, azaz a legkorábban elküldhető üzenet a tömb végén, a legtöbbet várakozó pedig a tömb elején található minden időpillanatban. Fontos kiemelni, hogy a késleltetett üzenetek mindig a tömb elején folytonosan, tömbelemhely kihagyása nélkül helyezkednek el rendezetten.

A FIFO sorba történő beillesztéskor a késleltetni kívánt üzenet struktúrájában a teljes késleltetési idő szerepel, azonban a sorban bent lévő elemek esetén a késleltetésből aktuálisan hátramaradt idő látható. Ráadásul a FIFO sor nem az abszolút hátramaradt késleltetési időt tárolja minden egyes bejegyzés esetén, hanem az időrendben eggyel korábbi kiküldésű, tömbindexben eggyel későbbi bejegyzés kiküldési idejéhez képest szükséges további késleltetési időkülönbséget. Így a FIFO sorban az utolsó, legkorábban kiküldésre kerülő bejegyzés esetén valóban a kiküldésig hátralévő idő szerepel, de az utolsó előtti, azaz a másodikként kiküldésre kerülő

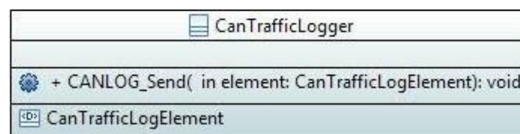
bejegyzéstől kezdve a kiküldési idő a saját indexhelyen és magasabb tömbindexű helyeken lévő késleltetési időkülönbségek összegeként számítható ki. Ezzel elérhető, hogy a periodikus számláló megszakításában csak a tömb utolsó elemének késleltetési idejét kell csökkenteni, és nem az összes bejegyzését.

Az almodul 4 függvényt biztosít interfészként, amelyekkel minden szükséges funkció megvalósítható: egy alaphelyzetbe állító, egy új bejegyzést beillesztő, egy bejegyzést törlő függvényt és végül egy megszakításkezelő rutint. A beillesztés és a kivétel alatt a késleltetéshez felhasznált periodikus számláló megszakításkérését letiltja a program, hogy biztosan a megfelelő helyen történjen meg a beszúrás és a kivétel. A megszakításkezelő rutint is meghívható interfészfüggvényként nyújtja, mert a CAN késleltetéshez használt különálló periodikus számláló kezelése a jobb átláthatóság érdekében az időzítővezérlő szoftvermodulba került.

A késleltetési sor rendezett tömbszervezésének és az időkülönbségek tárolásának köszönhetően a szükséges megszakítási rutinok rövid idő alatt lefuthatnak, illetve a periodikus számláló megszakításkérését rövid ideig szükséges csak letiltani.

6.3.6. CAN forgalomnaplózó (CanTrafficLogger)

A CanTrafficLogger állományok által definiált változók, függvények, felsorolásos típusok és adatstruktúrák az alábbi osztálydiagramon, a 6-9. ábrán láthatók.



6-9. ábra: A CAN forgalomnaplózó osztálydiagramja

A CAN forgalomnaplózó almodul a kapcsolón áthaladó forgalom naplózásának feladatát látja el, az előírt kéréseknek megfelelően.

A követelmények alapján valójában elegendő csak a beérkező adatforgalmat naplózni, hiszen a kapcsolási tábla előre ismert, így a kimeneti adatforgalom kiszámolható a bemenetek és a kapcsolótábla felhasználásával. Jelenleg a kapcsoló a naplózott információkat az UART-vezérlőn keresztül az XPORT-interfész segítségével juttatja el az Ethernet-hálózatra csatlakozó számítógépekhez. Az adatútból jól látható, hogy a szűk keresztmetszetet az UART-vonal 921600 bps adatátviteli sebessége jelenti. Ez a CAN-hálózatban maximálisan elérhető sebességnél is kisebb, ráadásul egy

naplóbejegyzésben több adat szerepel, mint egy átvitt CAN-üzenetben. Emiatt célszerű a lehető legkevesebb, redundancia nélküli adatot naplózni és átküldeni, így ez a fő oka annak, hogy csak a bemeneti adatforgalom kerül naplózásra.

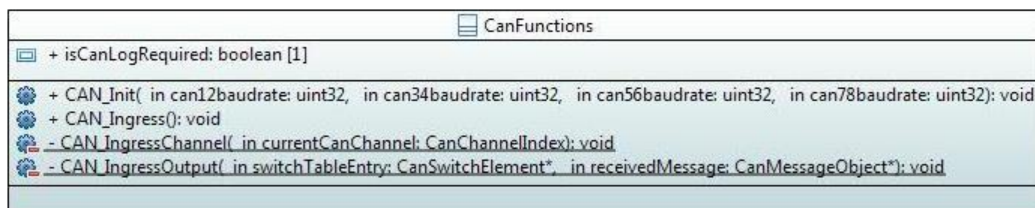
A forgalomnaplózást kezelő almodul CanTrafficLogElement néven definiálja azt a struktúrát, amely a naplózás során feljegyzésre és továbbításra kerül. A struktúra első eleme az időbélyeg, amely a 64 bites élettartam-számláló értékét tartalmazza az üzenet beérkezésekor. A második elem az üzenet, amelyet fogadott a kapcsoló. A harmadik pedig a csatorna indexe, ahonnan az üzenet beérkezett. Ennyi információ naplózásából a kapcsoló teljes hálózati forgalma rekonstruálható a tesztek során.

Az almodul interfésze egyetlen függvényt tartalmaz, amelynek az a feladata, hogy egy paraméterként kapott naplózási struktúra teljes tartalmát elküldje az UART-vonalon az XPORT-interfész felé.

A forgalomnaplózó felépítéséből látszik, hogy csak a naplózási adatáramlást teszi lehetővé, illetve a naplózott adatok struktúráját írja elő. A naplózási adatok begyűjtéséről az ezt hívó kódnak kell gondoskodnia.

6.3.7. Magasszintű CAN-funkciók (CanFunctions)

A CanFunctions állományok által definiált változók, függvények, felsorolásos típusok és adatstruktúrák az alábbi osztálydiagramon, a 6-10. ábrán láthatók.



6-10. ábra: A magasszintű CAN-funkciók osztálydiagramja

A CAN-vezérlő szoftvermodul legfelső szintje teszi elérhetővé az intelligens CAN-kapcsoló célfunkcióját, illetve az egyéb magas szintű funkciókat a felhasználói alkalmazások számára, az eddigiekben ismertetetett interfészfüggvények, funkciók, és adatszerkezetek felhasználásával. A diplomatervezés során megírt főprogram az itt definiált függvényeket hívja közvetlenül.

A felhasználói alkalmazások számára két függvényt tesz elérhetővé ez a rész, egy, az egész CAN-vezérlőt alaphelyzetbe állító függvényt, illetve a szoftveres kapcsolást megvalósító függvényt.

Az alaphelyzetbe állító függvény az összes eddig bemutatott almodul alaphelyzetbe állító függvényét meghívja, és ezzel lényegében a teljes CAN-vezérlő beállításait és tárolóit is kezdeti állapotba hozza. Egyébként ez az oka annak, hogy az összes többi almodul fejlécfájljától függ ez az almodul.

A szoftveres kapcsolást elvégző függvény egymás után az összes csatornára ugyanazt a belépési oldali kódot hajtja végre. Egy csatorna esetén először megpróbál beolvasni egy teljes CAN-üzenetet az adott csatornáról. Ha nem érkezett üzenet, vagyis a periféria olvasási FIFO sora üres, akkor vissza is tér, amennyiben érkezett üzenet, akkor elvégzi az üzenet megfelelő célcsatornára történő kapcsolását és közben az összes szükséges funkciót is. Új érkező üzenet esetén először megnézi, hogy szükséges-e forgalomnaplózás, és ha igen, akkor lekérdezi az élettartam-számláló aktuális értékét, kialakítja a naplóbejegyzést, és el is küldi az XPORT-interfész felé. Második lépésként az érkező üzenet azonosítója és a forráscsatorna alapján megkeresi és elkéri a kapcsolótábla erre az üzenetre vonatkozó bejegyzését. Ha nem található ilyen bejegyzés a táblában, akkor nem csinál semmit az üzenettel, tehát elnyeli, és nem továbbítja. Ha talált ilyen bejegyzést, akkor pedig elvégzi a bejegyzés alapján előírt hibainjekciókat és késleltetést, továbbkapcsolja a megfelelő célcsatorna irányába az üzenetet. Legvégső lépésként frissíti az üzenet manipulációs periódusának állapotgépét.

A hibainjekciót, késleltetést és csomagkapcsolást egy különálló lokálisan használt függvény valósítja meg, amely paraméterként a kapcsolótábla-bejegyzést és a beérkezett üzenetet igényli. Ha a kapcsolótábla-bejegyzés üzenetmanipulációs állapota szerint nem abban ciklusban tartunk, amiben szükséges a hibainjektálás vagy késleltetés, akkor azonnal a célcsatlakozó CAN-perifériájához továbbítja a csomagot, azaz a lehető leggyorsabban kiküldi a cél felé. Ha az állapotgép alapján szükséges a manipuláció, akkor először lépteti az állapotot. Második lépésként elvégzi az üzenet tartalmának adatmanipulációját. Ezután a kapcsolási bejegyzésben leírt ismétlésszámnak megfelelő alkalommal kiküldi az üzenetet. Ha az első üzenet esetén szükséges késleltetés, akkor a csomagot a várakozási sorba illeszti, ha pedig nem szükséges késleltetés, akkor rögtön továbbítja a célcsatorna CAN-perifériájához. Az ismételt üzenetek esetében is ugyanúgy jár el, csak a késleltetési időhöz hozzáadja az ismétlések közötti időtartamot is az aktuális ismétlési számnak megfelelően.

Alaposan átgondolva a csomagkapcsoló függvény működését, a függvény nem csak a követelményben leírt egyes hibainjektálási és késleltetési funkciók

megvalósítására képes, hanem ezek tetszőleges kombinációjára is, ehhez csak megfelelően kell kitölteni a kapcsolási tábla bejegyzését.

Fontos kiemelni, hogy a forgalomnaplózás egy globális jelzőbit átbillentésével tetszőlegesen ki- és bekapcsolható, ugyanis a felhasználási esetek többségében nincs szükség rá, és ilyenkor felesleges lenne a naplózás miatt lassítani a kapcsoló működését.

A magasszintű CAN-funkciók almodul felépítése alapján elmondható, hogy az intelligens CAN hálózati kapcsoló működése három fázisból kell, hogy felépüljön. Az inicializációs fázisban az összes szükséges szoftvermodult alaphelyzetbe kell állítani, illetve a hardveres funkciókat az alkalmazásnak megfelelő állapotba kell kapcsolni. A második fázisban fel kell tölteni a kapcsolási táblát a tesztkörnyezetek csatlakozási csatornái ismeretében. Végül a futási fázisban pedig elegendő csak a szoftveres kapcsolást elvégző függvényt végtelen ciklusban meghívni. Ezek alapján látható, hogy a felhasználói alkalmazás szintjén igen egyszerű kód szükséges a kapcsolóeszköz működtetéséhez.

6.4. Követelmények szoftveres teljesítése

Az intelligens CAN hálózati kapcsoló a vele szemben állított legtöbb követelményt hardveres úton megvalósítja, csak a célfunkciót – az irányított közvetlen kapcsolást – és néhány összetett funkciót – hibainjektálást és forgalomnaplózást – szükséges szoftverből megvalósítani.

A CAN-vezérlő bemutatásából jól látható, hogy ezeket az elvárt funkciókat a magasszintű CAN-funkciókat összefogó almodul mind megvalósítja. Az architekturális egyszerűsítések következtében az összes kívánt funkció a bemeneti oldalon teljesíthető. Emiatt a magasszintű CAN almodul egyetlen interfészfüggvény segítségével az elvártaknak megfelelő módon képes az összes szoftveres szolgáltatatás nyújtására.

7. Megvalósított kapcsolási algoritmusok

Az intelligens kapcsoló kétféle kapcsolási algoritmust is megvalósít. Az előző fejezetekben ismetetett hardver- és szoftverfelépítés, illetve az architektúrák alapján célszerű összegezni, összevetni a hardveresen és a szoftveresen megvalósított kapcsolási algoritmusokat.

Az intelligens CAN hálózati kapcsoló hardveresen a vonalkapcsolást valósítja meg, hiszen az útvonal kialakítása nem függ az üzenetek adattartalmától. Hardveres kapcsolási mód esetén a CAN-illesztőegységben, a láncolt jelvezetéken át történik meg a kapcsolás. Ehhez a láncot teljes egészében ki kell alakítani, és az adó-vevők tiltásával, illetve engedélyezésével lehetséges az üzenet megfelelő irányba történő kapcsolása. Ilyenkor a kapcsolási iránynak előre ismertnek kell lennie, vagyis a vonal felépítése nem egy kezdeményezőjel segítségével, hanem a tesztkörnyezet megfelelő vezérlése alapján valósul meg. A tényleges kommunikáció fázisában teljes sávszélességgel lehetséges a kommunikáció, vagyis a beérkező biteket azonnal, még bitidőn belül, minimális késleltetéssel el is kezdi továbbítani a kapcsoló. A hardveres megoldás előnye, hogy rendkívül gyorsan történhet meg a kapcsolás és az üzenet továbbítása. Hátránya, hogy semmilyen magas szintű funkció sem érhető el, illetve hogy külön előre kell kialakítani a kapcsoló kívánt iránynak megfelelő beállítását.

Az intelligens CAN hálózati kapcsoló szoftveresen a „tárol és továbbít" algoritmust valósítja meg, hiszen az üzeneteket egységként kezeli, és mindig teljes hosszában fogadja azokat, mielőtt továbbküldené. A magasszintű CAN-funkciós almodul interfészfüggvényei szükségesek a szoftveres kapcsolási mód kialakításához. Ilyenkor a mikrokontroller CAN-periféria működési módja határozza meg az algoritmust, hiszen az mindig megvárja a teljes üzenet beérkezését, ellenőrzi a helyességét, és csak hibátlan és teljes vétel esetén teszi elérhetővé az üzenetet a mikrokontroller számára. Emiatt szoftverből a kapcsolási döntés csak akkor hozható meg, ha már a teljes csomag beérkezett. A szoftveres megoldás előnye, hogy az összes magas szintű funkció elérhető, illetve nincs szükség előzetes felkészülésre a kapcsolási irány kialakításához. Hátránya, hogy nagyobb késleltetést okoz a kapcsolás a teljes üzenetfogadás megvárása következtében.

8. Az elkészült eszköz tesztelése

Az előző fejezetekben részleteztem a diplomatervezés során elkészült hardver és szoftver felépítését és a tervezési megfontolásokat. A feladat része, hogy a megvalósított komponenseket és funkciókat le is kell tesztelni, hogy az elvártaknak megfelelően működnek-e. Ebben a fejezetben röviden ismertetem az eszköz tesztelésének módszereit, illetve néhány fontosabb eredményt is bemutatok.

Először a tesztelés közben végrehajtott teszteket, utána a fejlesztés végén elvégzett teszteket, végül pedig a tesztkörnyezetbe történő beillesztést is részletezem.

8.1. Tesztelés fejlesztés közben

Rendkívül fontos, hogy a fejlesztés folyamata során a lehető leghamarabb elkezdődjön a tesztelés is, hiszen minél hamarabb fény derül egy esetleges hibára, annál kisebb erőfeszítéssel és költséggel javítható az.

A hardver fejlesztése során a körültekintő tervezés mellett felülvizsgálatok (review) segítették a lehető legkevesebb hibával történő tervezést. Az egyes részegységek elkészítése során is folyamatosan egy-egy tapasztaltabb hardvertervező mérnök kolléga is átnézte az elkészült tervrészleteket, és figyelmeztettek az esetleges hibákra. Ezenkívül a teljes hardverterv elkészítése után, de még a gyártásba küldés előtt a teljes elkészült tervet is felülvizsgálták még egyszer. A gyakori felülvizsgálatoknak köszönhetően gyártás után a hardver élesztése első próbálkozásra sikeres volt, és minden részegysége működőképesnek bizonyult.

A szoftvermodulok fejlesztése során különböző funkcionális tesztkódokkal bizonyíthattam mind magamnak, mind pedig a későbbi felhasználók számára az adott modul működőképességét. Ezek a tesztek a fejlesztéssel párhuzamosan készültek el, így azonnali visszajelzést kaphattam az esetleges hibákról. A tesztek az almodulok interfészfüggvényeit, illetve kritikus belső függvényeit is meghívták különböző bemeneti paraméterekkel, így igazolva helyes működésüket.

Összességében elmondható, hogy nagyon hasznosnak bizonyultak a fejlesztés közbeni tesztek, hiszen az esetleges hibákat azonnal javítani tudtam.

8.2. Az elkészült eszköz teljesítménytesztje

A CAN-hálózatot kapcsoló hardver és szoftver elkésztése után a teljes eszközt is egy végső tesztelésnek kellett alávetni.

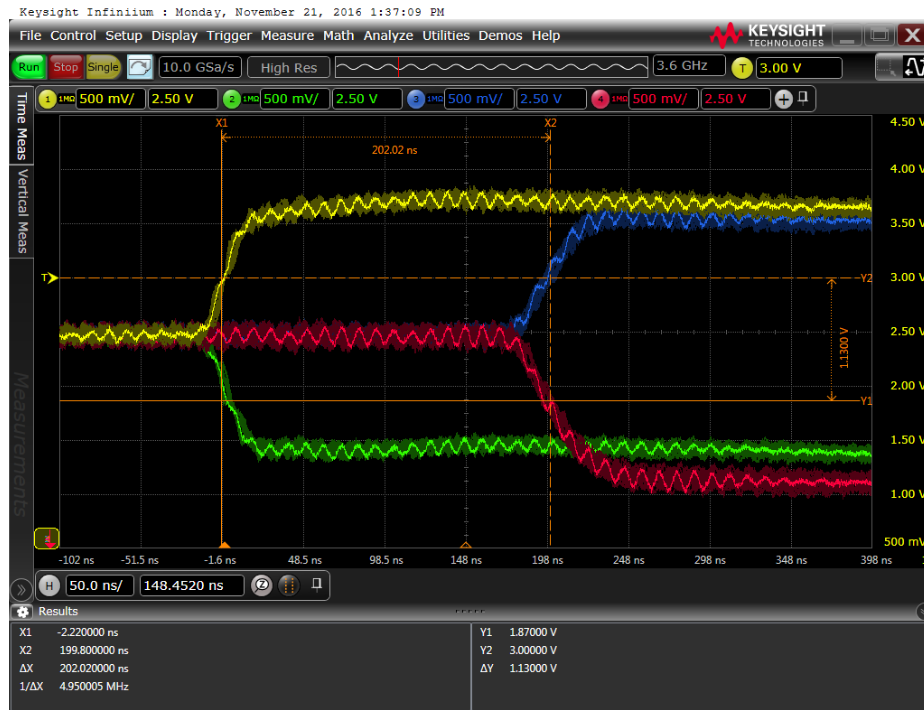
A fejlesztés fő célja, hogy elkészüljön egy a lehető legkisebb késleltetésű CAN-hálózati rétegben működő kapcsolóeszköz, amelynek segítségével a teszteléshez használt adatok és időzítéseik a lehető legkevésbé változnak meg, miközben lehetségessé válik az irányított adattovábbítás a többi kívánt magas szintű funkció végrehajtása mellett is. Ennek következtében a fejlesztés végén mérésekkel kellett igazolni, hogy a kifejlesztett kapcsolóeszköz az elvárt késleltetési időkön belül képes a feladatának ellátására.

A teljesítménytesztek során tehát mind a hardveresen, mind pedig a szoftveresen megvalósított algoritmus késleltetését meg kellett mérni, és az eredmények alapján eldönteni, hogy megfelelőek-e a felhasználási környezetben. A mérések során egy CAN-hálózati forgalmat generálni képes külső terepbusz (Fieldbus Gateway) eszköz küldött ki 500 kbps sebességgel különböző CAN-üzeneteket az intelligens kapcsoló első CAN csatlakozójára, majd a nyolcadik csatlakozóról ugyanaz az külső eszköz fogadta az üzeneteket azonos sebességgel. A mérések alatt a kapcsoló beállításai alapján minden üzenetet, késleltetés és bármilyen manipuláció nélkül, az első csatornáról a nyolcadik csatornára továbbított mind a két algoritmus esetében, és a többi csatornához tartozó adó-vevők tiltott állapotban voltak. A két mérési csatorna kiválasztásának az az oka, hogy ez a két csatorna van fizikai kialakítás alapján a legmesszebb egymástól, vagyis e két csatlakozó közötti kapcsolás tarthat a legtöbb ideig, így ez számít a legrosszabb esetnek. A mérések során a hardveres esetben a mikrokontroller nem játszott szerepet a kapcsolásban, szoftveres esetben pedig csak egyetlen kapcsolótábla-bejegyzést tartalmazott az előbb leírtaknak megfelelően, és a naplózás pedig ki volt kapcsolva.

A késleltetés mérésekor a kapcsolón a be- és kimeneti csatlakozók közvetlen közelében elhelyezett mérőpontokon lehetett megfigyelni a csatlakozón, vagyis az adott CAN-csatorna buszán lévő fizikai jeleket. A mérés egy Agilent MSO9254A típusú négycsatornás oszcilloszkóp segítségével történt, amelynek négy mérőfejét a két használt CAN-csatorna pozitív és negatív vezetékeire kötöttem a mérőpontoknál. Az oszcilloszkóp kurzor opciójának segítségével pedig a bemeneti csatornán érkező, és a kimeneti csatornán kiküldött CAN-üzenet startbitje között eltelt időt kellett megmérni a

késleltetés megállapításához. Ebben a mérési elrendezésben tehát a kapcsoló teljes késleltetését lehetséges megmérni nagy pontossággal.

Alább, a 8-1. és a 8-2. ábrákon látható a hardveresen megvalósított vonalkapcsolás során mért késleltetés.



8-1. ábra: Hardveresen megvalósított vonalkapcsolás késleltetése

8-2. ábra: Hardveresen megvalósított vonalkapcsolás késleltetése üzenetszinten

Az ábrákon a mért időfüggvények láthatók. A bemeneti CAN-csatorna pozitív vezetékén (CANH) mért jelet a sárga szín, a negatív vezetékén (CANL) lévő jelet a zöld szín mutatja, míg a kimeneti csatorna pozitív vezetékét a kék szín, a negatív vezetékét a piros szín reprezentálja. A mérések alapján elmondható, hogy a hardveres vonalkapcsolás esetén a kapcsoló rendkívül kevés, mindössze 202 ns idejű késleltetést okoz a két hálózat között a tényleges kommunikáció fázisában. Az is jól látható, hogy az elvártaknak megfelelően azonnal megtörténik a továbbítás, és a jelvezetékláncban használt elemek összesített késleltetése után, még a bitidőn belül a kimeneten már meg is jelenik az üzenet azonos bitpozíciójában fogadott jelszint. A teljes üzenet idejéhez viszonyítva lényegében zérusnak vehető a késleltetés, és a két hálózaton minden időpillanatban lényegében ugyanaz a tartalom jelenik meg.

A szoftveresen megvalósított „tárol és továbbít” algoritmus során mért késleltetés ábrája az alábbi, 8-3. ábrán látható.

8-3. ábra: Szoftveresen megvalósított „tárol és továbbít” algoritmus késleltetése

Az ábrán a mért időfüggvények láthatók. A bemeneti CAN-csatorna pozitív vezetékén (CANH) mért jelet a sárga szín, a negatív vezetékén (CANL) lévő jelet a zöld szín mutatja, míg a kimeneti csatorna pozitív vezetékét a kék szín, a negatív vezetékét a piros szín reprezentálja. A mérések alapján elmondható, hogy a szoftveres „tárol és továbbít” algoritmus esetén a kapcsoló már jól érzékelhető 248 μs idejű késleltetést

okoz a két hálózat között. Az ábrán az is jól látható, hogy az elvártaknak megfelelően a kapcsoló először fogadja az üzenetet teljes egészében, a beérkezés után történik meg a kapcsolás, és csak utána kezdődik meg a kiküldése a megfelelő irányban. Külön kimérhető a fogadott üzenet ACK bit vége és a kiküldött üzenet START bit eleje közötti idő, amely nagyságrendileg 24 µs. A CAN-üzenet formátuma alapján az ACK bit és az üzenet vége között még nyolc recesszív bit található, amely 500 kbps sebességen 16 µs. Tehát maga a kapcsolás 8 µs időtartamon belül történik meg átlagosan a szoftverből.

Összességében elmondható, hogy mind a két kapcsolási algoritmus megfelel a felhasználási környezetből adódó elvárásoknak, hiszen a tesztkörnyezetekben ms nagyságrendbe eső pontosságú időzítések ellenőrzésére van mód. A mérések alapján a szoftveres megoldás 1, a hardveres megoldás pedig 4 nagyságrenddel kisebb késleltetést okoz az elvárt pontossághoz képest.

8.3. Beillesztés a tesztkörnyezetbe

A követelmények összegyűjtése során már igen pontosan ismertek voltak az intelligens CAN hálózati kapcsoló csatlakozásai a külvilág felé, mind funkcionálisan, mind pedig a fizikai kialakítása szempontjából.

Már a hardver- és szoftvertervezés előtt pontosan ismertek voltak a szükséges funkcionális interfészek és fizikai méretek, emiatt a tervezés egyik céljává vált az, hogy a megvalósított eszköz az elvárt méretű legyen, valamint a követelményként megfogalmazott logikai és fizikai kapcsolatokat tartalmazza. Emiatt a teszkörnyezetbe történő beillesztés könnyedén megvalósult, és a beillesztés után a kapcsolási tábla felprogramozása után azonnal működőképesnek bizonyult, és így felhasználhatóvá vált.

9. Összefoglalás, továbbfejlesztési lehetőségek

A diplomatervezés részeként először megismerkedtem a hálózati eszközökben alkalmazott szoftverek rétegszervezésű felépítésével, beleértve adatkapcsolati réteg feladatait, mert a feladatkiírásban szereplő intelligens kapcsolóeszköz is ebben a rétegben üzemel. Megvizsgáltam a manapság alkalmazott kapcsolási algoritmusokat, amely során utánakerestem a hálózati környezetben használt alapfogalmaknak, az algoritmusok csoportosítási lehetőségeinek, illetve az algoritmusok jellemzéséhez használható mérőszámoknak és elemzési szempontoknak. Az alapok megismerése után mélyrehatóan elemeztem az egyes algoritmusokat hatékonyság és felhasznált erőforrások szempontjai alapján.

Ezen ismeretek segítségével átláthatóvá vált számomra, hogy pontosan milyen feladatok elvégzésre lehet szükség az intelligens kapcsolóeszközben, és összegyűjtöttem a leendő felhasználók igényeit egy megbeszélés keretein belül.

Az elméleti tudás és a követelmények ismeretében már el lehetett kezdeni az eszköz fejlesztését. A feladat megvalósítása két fejlesztési fázisra osztható. Az első fázis az intelligens CAN hálózati kapcsoló hardverének a megtervezése volt. Ennek során figyelni kellett a panel fizikai méretére, a tesztkörnyezethez való kapcsolódási interfészekre, valamint a követelményként előírt hardveres funkciók megvalósítására. A második fázis pedig a megtervezett és legyártott intelligens kapcsoló panel szoftverének kifejlesztéséből állt. Szoftverből az eszköz célfunkcióját, az egyetlen irányba történő adattovábbítást és két magasabb szintű funkciót: a hibainjektálást és a forgalomnaplózást kellett megoldani. A feladat részeként végül le kellett tesztelnem a kapcsolóeszközt, illetve meg kellett győződnöm arról, hogy megfelelően, kis késleltetéssel képes ellátni a feladatát, végül pedig be is kellett illesztenem a tesztkörnyezetbe.

A diplomatervezés során elkészült az intelligens CAN hálózati kapcsolóeszköz, amely képes a CAN-hálózaton az üzenetek kapcsolására az elvártaknak megfelelő sebességgel és késleltetésekkel, valamint a követelményekben felsorolt összes funkciót is képes ellátni. A késznek mondható termék ellenére a fejlesztés során többféle lehetőség és szempont is felmerült, amelyek mentén célszerű lehet a későbbiekben továbbfejleszteni az intelligens kapcsolót.

Az egyik legfontosabb fejlesztési lehetőség, hogy lehetőséget kellene biztosítani a kapcsolótábla futás közbeni átírására. Habár igaz, hogy a szerverszekrényben általában nem változik a tesztelő eszköz, illetve a tesztkörnyezetek elrendezése, de ritka esetekben (pl.: cserék, javítások) ez is előfordulhat. Továbbá a későbbiekben nem garantálható biztosan, hogy a teszteszköznek, amely mindig egy csatlakozó irányában található, ne kelljen az új üzenetek megjelenésével azonos azonosítóval több tesztkörnyezet irányába is adatot küldenie. Mind a két esetben jelenleg újból fel kellene programozni a statikus kapcsolótáblát manuálisan. Ehelyett sokkal egyszerűbb megoldás, ha egy szervergépről ethernetes hálózaton keresztül, futás közben lehetne a kapcsolótáblát megváltoztatni, így automatizáltan lehetne alkalmazkodni a változásokhoz is, ami kényelmesebb felhasználhatóságot eredményezne.

Manapság nemcsak CAN-, hanem FlexRay-hálózatokat is alkalmaznak az autóiparban. A követelmények összegyűjtésekor már kiderült, hogy a későbbiekben FlexRay-hálózatokon történő kapcsolásra is felhasználnák az intelligens kapcsolóeszközt. Emiatt a hardver megtervezésekor már ezt a szempontot is figyelembe vettem, és a kapcsolóeszköz hardveresen képes a FlexRay-hálózatok kezelésére is. Azonban a FlexRay-hálózat bonyolultsága miatt magát az adó-vevőt is fel kellene programozni a központi feldolgozóegység segítségével a hálózati beállításoknak megfelelően, tehát önmagában, hardveresen nem képes a kapcsolás ellátására FlexRay-hálózaton. Emiatt a későbbiekben mindenképp szoftveresen is ki kell egészíteni az intelligens kapcsoló programját ahhoz, hogy a FlexRay-hálózatokat is kezelje.

A megoldások leírása során többször is megemlítettem, hogy jelenleg az ethernetes hálózatokból történő vezérlés és adatcsere az XPORT-interfész eszközön keresztül valósul meg. Tehát a központi feldolgozóegység valójában egy egyszerű UART-hálózaton keresztül képes a külvilággal kommunikálni, mert az UART-periféria felélesztése és beállítása sokkal kevesebb fejlesztési idő alatt megoldható, mint az Ethernet-periféria felprogramozása és egy TPC/IP protokollverem illesztése. Azonban az XPORT eszköz az UART-vonalon maximálisan csak 921600 bps adatátviteli sebességre képes, amely a kapcsoló számára nem elegendő, főleg ha a kapcsolótábla futás közbeni átírását is lehetővé akarjuk tenni. A mikrokontroller rendelkezik a feladat számára megfelelő Ethernet-perifériával, emiatt a későbbiekben mindenképp célszerű áttérni a közvetlen ethernetes kapcsolat kialakítására.

Irodalomjegyzék

[1] Andrew S. Tanenbaum, David J. Wetherall: *Számítógéphálózatok*, harmadik, bővített, átdolgozott kiadás, ISBN 978-963-545-529-4, Panem Könyvek, Taramix Kft., 2013

[2] Pavel Tvrdik: *Topics in Parallel Computing*, Course of University Of Wisconsin-Madison, Spring 1999, http://pages.cs.wisc.edu/~tvrdik/cs838.html#schedule (megtekintés: 2016. május 20. 12:50)

[3] Cisco: *Cut-Through and Store-and-Forward Ethernet Switching for Low-Latency Environments*, http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5020-switch/white_paper_c11-465436.html (megtekintés: 2016. május 20. 12:50)

[4] Chwan-Hwa (John) Wu, J. David Irwin: *Introduction to Computer Networks and Cybersecurity*, ISBN-13: 978-1-4665-7214-0, CRC Press, 2012

[5] D-Link: *What is the difference between Store-and-Forward switching and Cut-Through switching?*, http://www.dlink.com/hu/hu/support/faq/switches/layer-3-gigabit/dgs-series/what-is-the-difference-between-store-and-forward-switching-and-cut-through-switching (megtekintés: 2016. május 20. 12:50)

[6] Joann Zimmerman, Charles E. Spurgeon: *Ethernet Switches*, 1st Edition, ISBN: 9781449367299, O'Reilly Media, Inc., 2013

[7] Olivier Bonaventure: Computer Networking : *Principles, Protocols and Practice*, 2nd Edition, Chapter 6: DataLink Layer – Ethernet Switches, http://cnp3book.info.ucl.ac.be/2nd/html/protocols/lan.html#ethernet-switches (megtekintés: 2016. május 20. 12:50)

[8] Matthew J Castelli: *How a LAN Switch Works*, http://www.ciscopress.com/articles/article.asp?p=357103&seqNum=4 (megtekintés: 2016. május 20. 12:50)

[9] Matthew J Castelli: *LAN Switching First-Step*, 1st Edition, ISBN-10: 1-58720-100-3, ISBN-13: 978-1-58720-100-4, Cisco Press, 2004

[10] Lantronix: *Network Switching tutorial*, https://www.lantronix.com/resources/networking-tutorials/network-switching-tutorial/ (megtekintés: 2016. november 27. 16:42)

[11] Paul Duvall, Steve Matyas, and Andrew Glover: *Continuous Integration: Improving Software Quality and Reducing Risk*, 1st Edition, ISBN-10: 0321601912, ISBN-13: 978-0321601919, Addison-Wesley, 2007

[12] International Standard: *IEC 60297-3-100: Mechanical structures for electronic equipment – Dimensions of mechanical structures of the 482,6 mm (19 in) series – Part 3-100: Basic dimensions of front panels, subracks, chassis, racks and cabinets*. Edition 1.0 2008-11

[13] Hotraco-Agri: CAN Switch,
http://www.hotraco-agri.com/en/product/861244556/can-switch
(megtekintés 2016. november 27. 16:42)

[14] International Organization for Standardization: *ISO 11898 - Road vehicles - Controller area network (CAN)*, Edition: 2, 2015

[15] FlexRay Consortium: *FlexRay Communications System Protocol Specification* Version 3.0.1, 2010 October

[16] Schroff: Main Catalog – Equipment Protection Solutions, 24th Edition October 2012 http://www.jsquared.com/files/Schroff-subracks.pdf
(megtekintés 2016. november 27. 16:42)

[17] Texas Instruments: *TPS65251 4.5-V to 18-V Input, High-Current, Synchronous Step-Down Three Buck Switcher With Integrated FET*,
http://www.ti.com/lit/ds/symlink/tps65251.pdf
(megtekintés: 2016. november 27. 16:42)

[18] Texas Instruments: *DP83848x PHYTER Mini / LS Single Port 10/100 MB/s Ethernet Transceiver*, http://www.ti.com/lit/ds/symlink/dp83848t.pdf
(megtekintés: 2016. november 27. 16:42)

[19] Lantronix: *XPort Embedded Ethernet Device Server,*
https://www.lantronix.com/products/xport/
(megtekintés: 2016. november 27. 16:42)

[20] On Semiconductor: *AMIS-42700 Dual High Speed CAN Transceiver*,
http://www.onsemi.com/pub_link/Collateral/AMIS-42700-D.PDF
(megtekintés: 2016. november 27. 16:42)

[21] On Semiconductor: *AND8360 AMIS-42700 Multiple CAN Bus Network*,
http://www.onsemi.com/pub_link/Collateral/AND8360-D.PDF
(megtekintés: 2016. november 27. 16:42)

[22] Buddy Shipley: *Installer's Guide to Local Area Networks*, ISBN 0-7668-3374-7, Thomson Delmar Learning, 2004

[23] Cisco Systems Inc: *Cisco IE 3000 Software Configuration Guide, Release 12.2(50)SE*,
http://www.cisco.com/c/en/us/td/docs/switches/lan/cisco_ie3000/software/release/12-2_50_se/configuration/guide/ie3000scg/swqos.html
(megtekintés: 2016. november 27. 16:42)

[24] Cisco Systems Inc: Cisco Catalyst 3750-E Series Switches,
http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-3750-e-series-switches/prod_qas0900aecd805bbea5.html
(megtekintés: 2016. november 27. 16:42)

Ábrajegyzék

[1] Andrew S. Tanenbaum, David J. Wetherall: *Számítógéphálózatok*, harmadik, bővített, átdolgozott kiadás, ISBN 978-963-545-529-4, Panem Könyvek, Taramix Kft., 2013

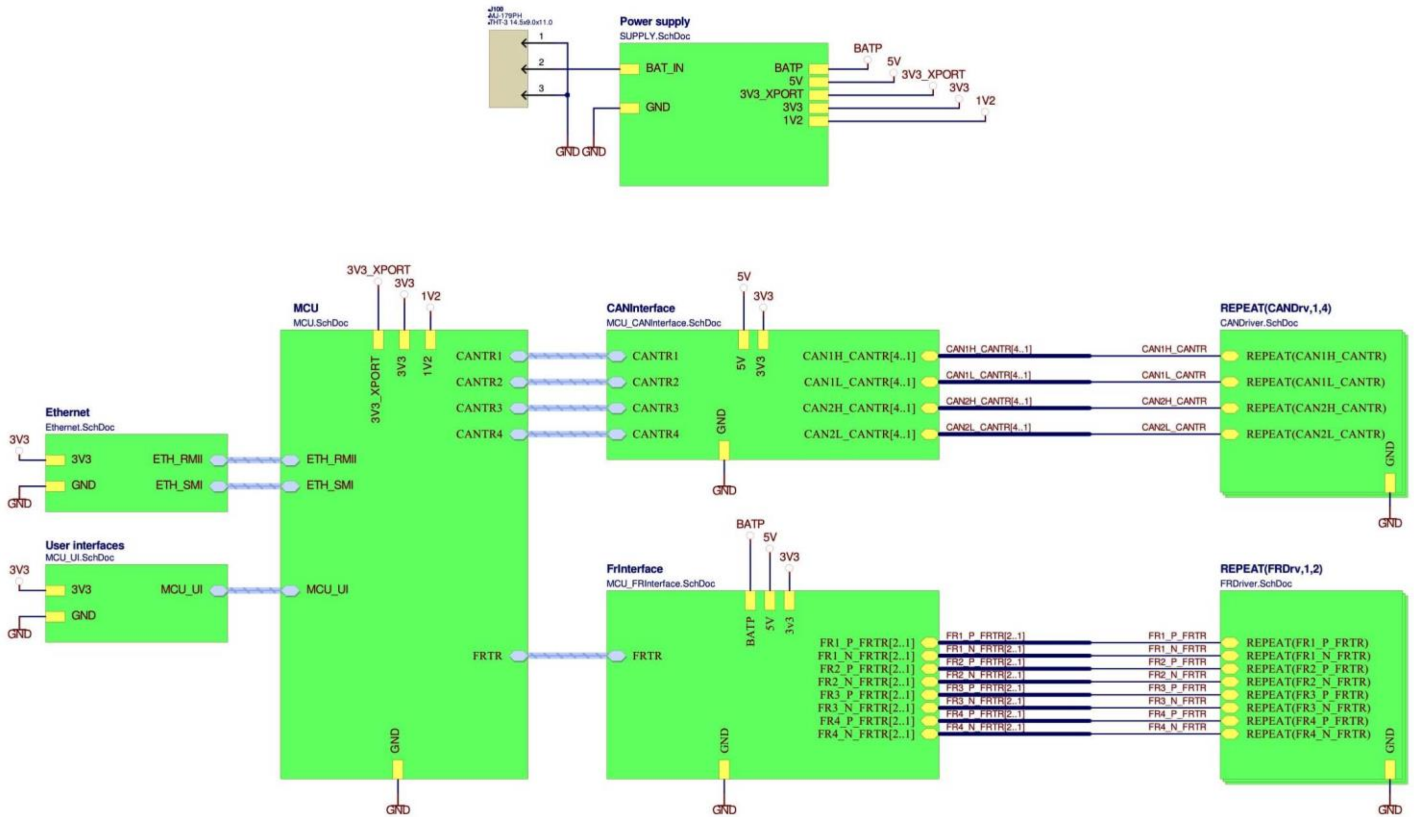
[2] Pavel Tvrdik: *Topics in Parallel Computing*, Course of University Of Wisconsin-Madison, Spring 1999, http://pages.cs.wisc.edu/~tvrdik/cs838.html#schedule (megtekintés: 2016. május 20. 12:50)

[3] On Semiconductor: *AND8360 AMIS-42700 Multiple CAN Bus Network*, http://www.onsemi.com/pub_link/Collateral/AND8360-D.PDF (megtekintés: 2016. november 27. 16:42)

[4] Cisco Systems Inc: Cisco Catalyst 3750-E Series Switches, http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-3750-e-series-switches/prod_qas0900aecd805bbea5.html (megtekintés: 2016. november 27. 16:42)

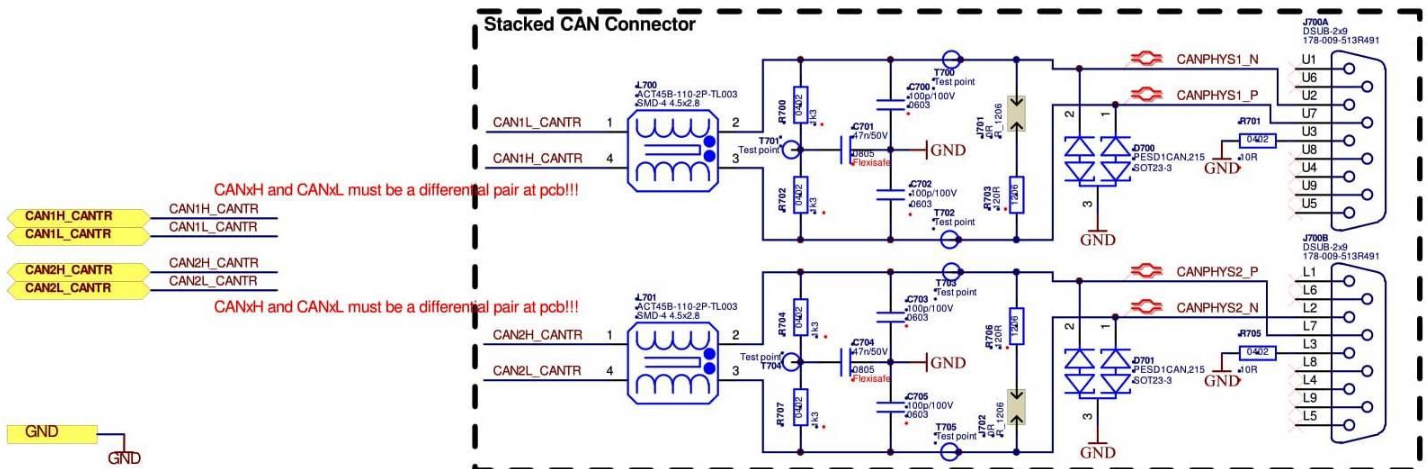
Függelék

A hardverterv blokkszintű kapcsolási rajza



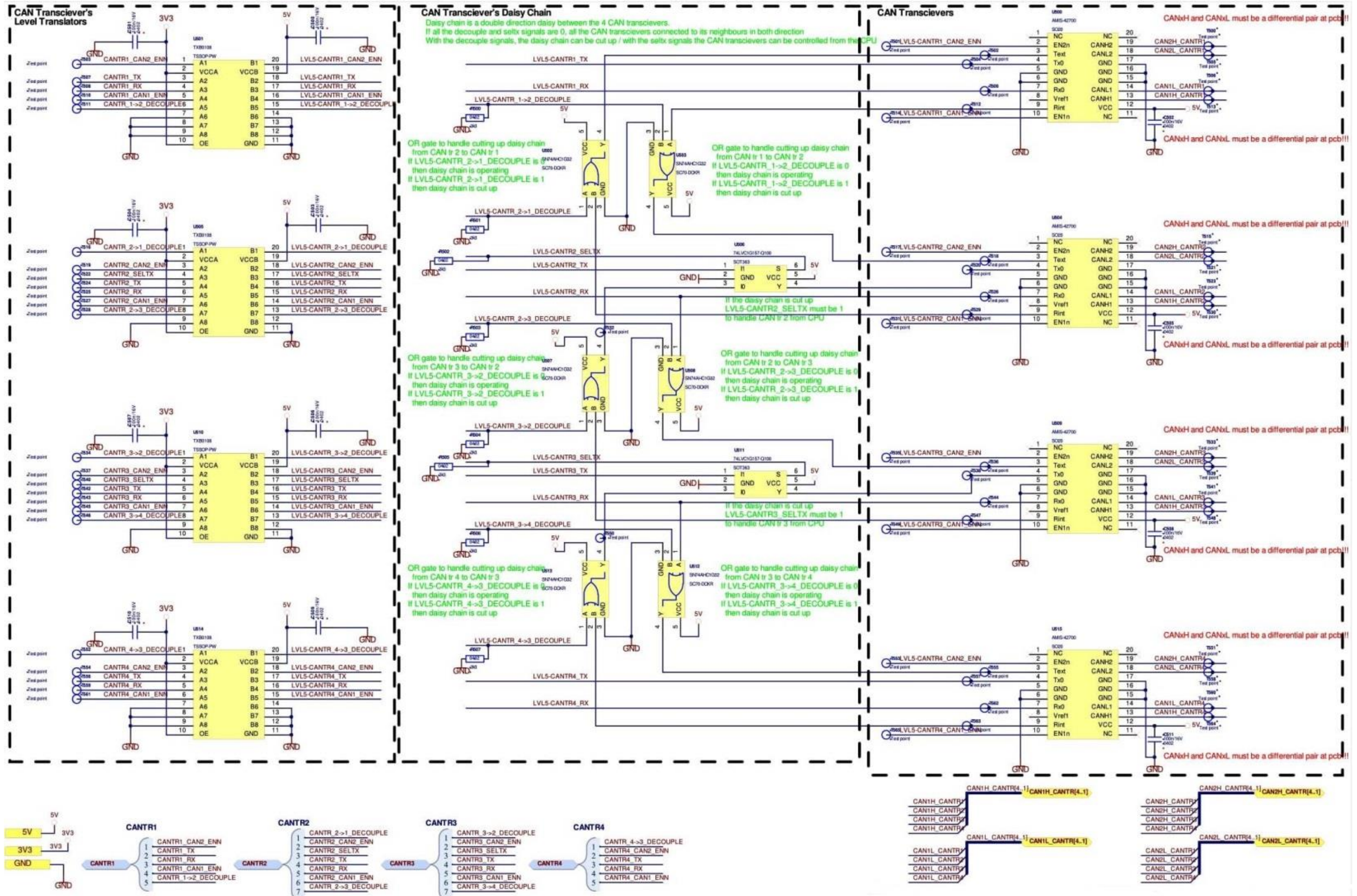
F-1. ábra: A hardverterv blokkszintű kapcsolási rajza

A CAN fizikai meghajtó kapcsolási rajza



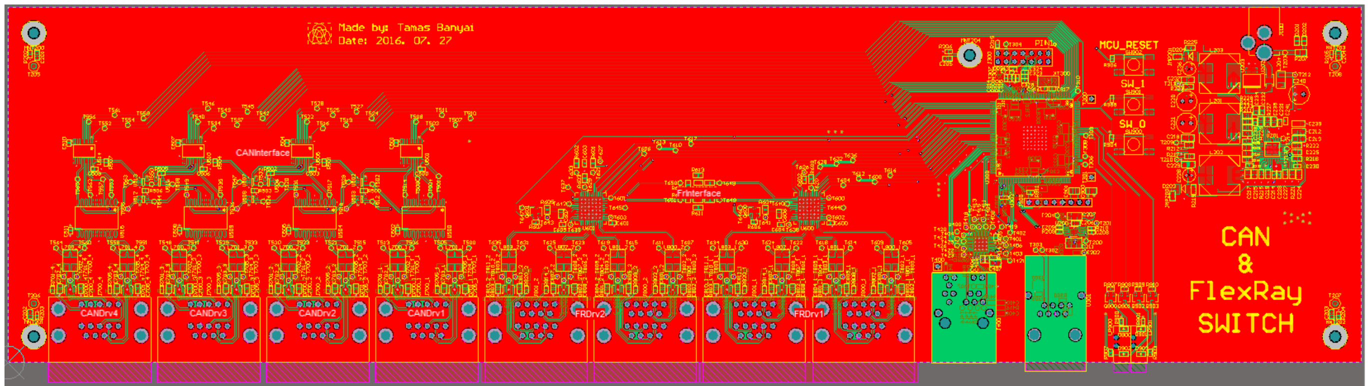
F-2. ábra: A CAN fizikai meghajtó kapcsolási rajza

A CAN-illesztő kapcsolási rajza

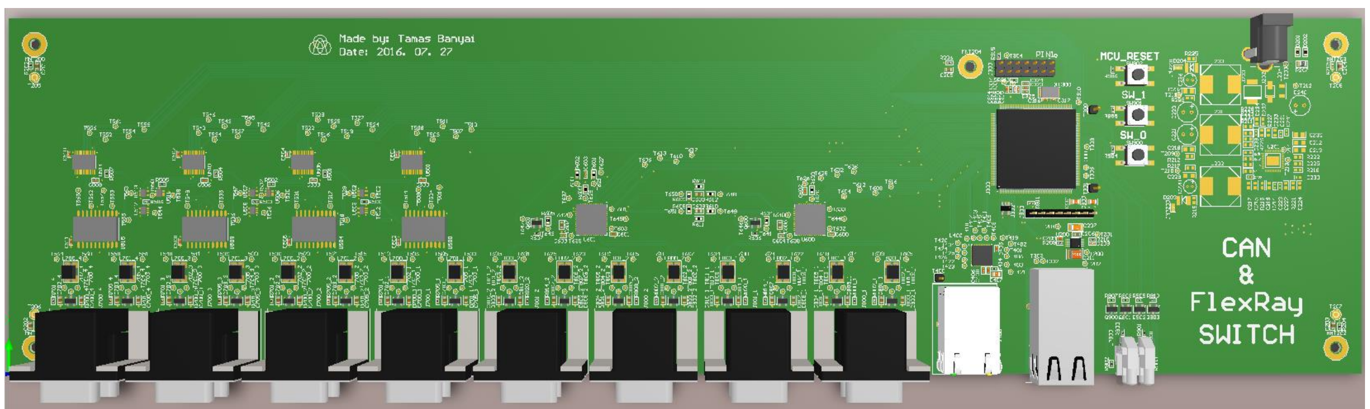


F-3. ábra: A CAN-illesztő kapcsolási rajza

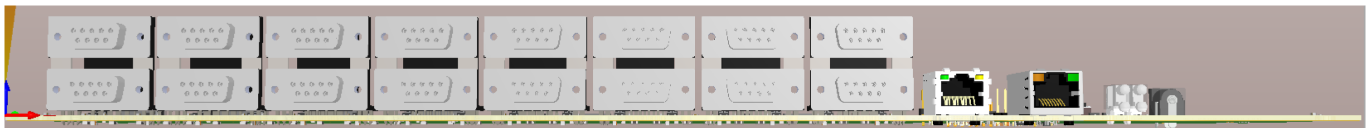
A megtervezett panel áramköri rajzolata



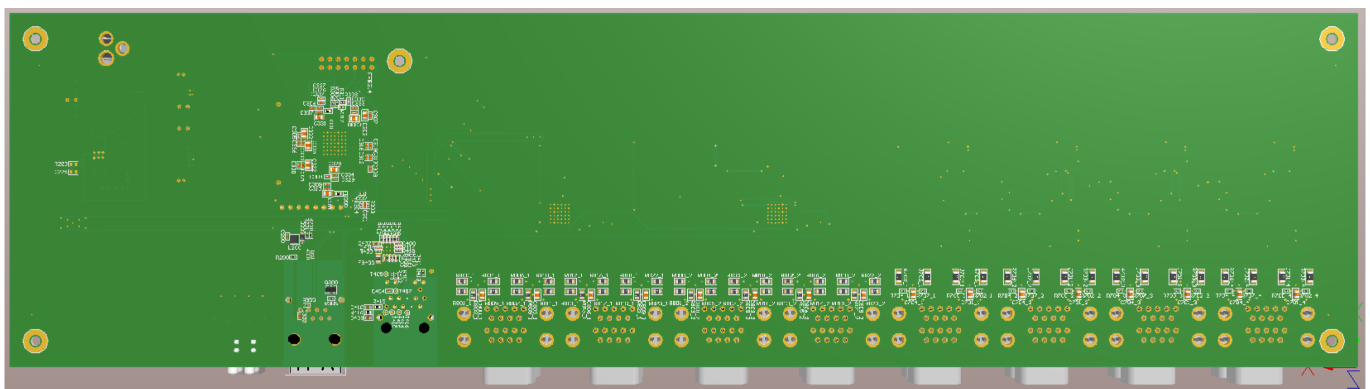
F-4. ábra: A megtervezett panel 2 dimenziós rajzolata



F-5. ábra: A megtervezett panel 3 dimenziós modellje felülnézetben

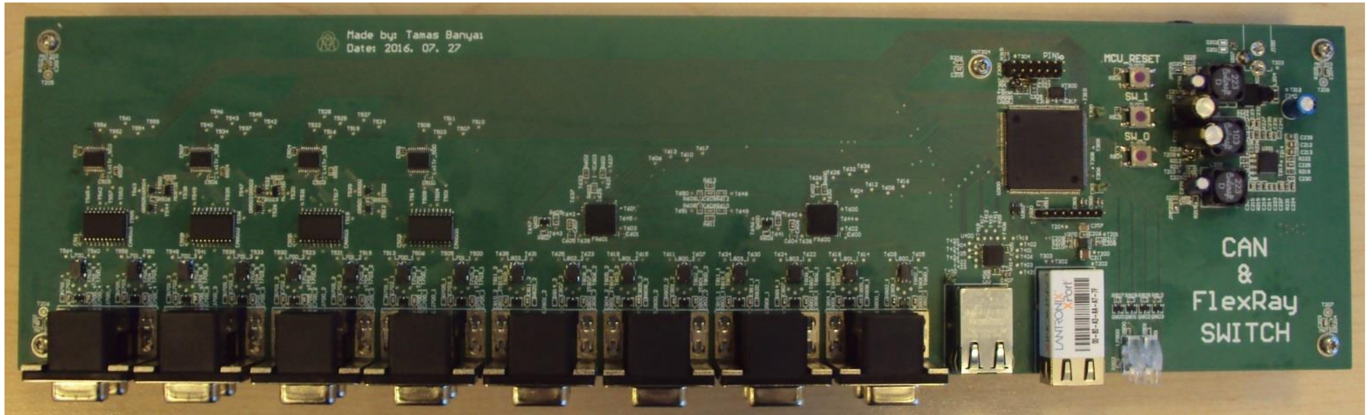


F-6. ábra: A megtervezett panel 3 dimenziós modellje elölnézetben



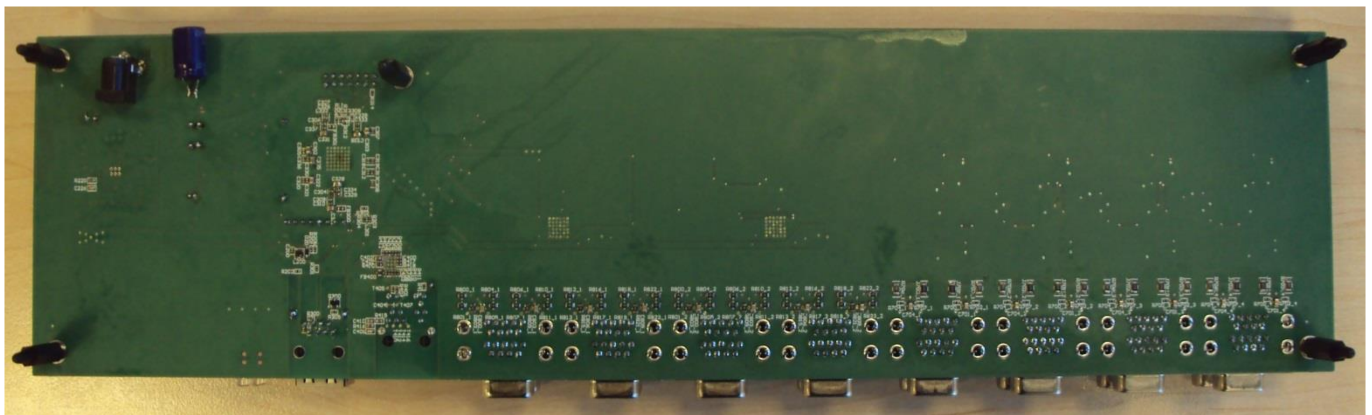
F-7. ábra: A megtervezett panel 3 dimenziós modellje alulnézetben

Az elkészült panel



F-8. ábra: Az elkészült panel felülnézete

F-9. ábra: Az elkészült panel elölnézete



F-10. ábra: az elkészült panel hátulnézete

A szoftver almodulokban használt adatstruktúrák definíciói

A CAN ki- és bemeneti almodul felsorolásos típusai, amelyek az adó-vevők letiltásáért vagy engedélyezéséért, láncolt jelvezeték felszakításáért, és a láncban középső adó-vevők küldési irányú bemeneteinek forrásának vezérléséért felelősek.

```
typedef enum {
    CANTR1 = 0,
    CANTR2 = 1,
    CANTR3 = 2,
    CANTR4 = 3,
    CANTR5 = 4,
    CANTR6 = 5,
    CANTR7 = 6,
    CANTR8 = 7
} EnablingOptions;

typedef enum {
    BETWEEN_12_34 = 0,
    BETWEEN_34_56 = 1,
    BETWEEN_56_78 = 2
} DecouplingOptions;

typedef enum {
    SEL_34 = 0,
    SEL_56 = 1,
} MuxSelectOptions;
```

A CAN-üzenet RAM adatstruktúrája, amely a CAN-perifériától kapott üzenet elemek szerkezetét írja le.

```
typedef struct {
#if CANMSGRAM_NUMBER_STD_FILTERS != 0
        CanMsgRamStdMsgFilter CANMSGRAM_StdFilter [CANMSGRAM_NUMBER_STD_FILTERS];
#endif
#if CANMSGRAM_NUMBER_EXT_FILTERS != 0
        CanMsgRamXtdMsgFilter CANMSGRAM_XtdFilter [CANMSGRAM_NUMBER_EXT_FILTERS];
#endif
#if CANMSGRAM_NUMBER_RXFIFO0_ELEMENT != 0
        CanMsgRamRxFifo0 CANMSGRAM_RX_FIFO_0 [CANMSGRAM_NUMBER_RXFIFO0_ELEMENT];
#endif
#if CANMSGRAM_NUMBER_RXFIFO1_ELEMENT != 0
        CanMsgRamRxFifo1 CANMSGRAM_RX_FIFO_1 [CANMSGRAM_NUMBER_RXFIFO1_ELEMENT];
#endif
#if CANMSGRAM_NUMBER_RXBUFFER_ELEMENT != 0
        CanMsgRamRxBuffer CANMSGRAM_RX_Buffer [CANMSGRAM_NUMBER_RXBUFFER_ELEMENT];
#endif
#if CANMSGRAM_NUMBER_TXBUFFER_ELEMENT != 0
        CanMsgRamTxBuffer CANMSGRAM_TX_Buffer [CANMSGRAM_NUMBER_TXBUFFER_ELEMENT];
#endif
#if CANMSGRAM_NUMBER_TX_EVENTS != 0
        CanMsgRamTxEventBuffer CANMSGRAM_TXEvent_FIFO [CANMSGRAM_NUMBER_TX_EVENTS];
#endif
} CanMessageRAM;
```

Az alacsonyszintű CAN-vezérlő CAN-üzenet struktúrája, illetve az elérhető csatornák felsorolása.

```
typedef struct {
        uint32 ID;   /* Message unique CAN ID */
        uint8 XTD;   /* Boolean Flag for Extended Frame Format */
        uint8 RTR;   /* Boolean Flag for Remote Transmission */
        uint8 data_length;                   /* Data length in bytes */
        uint8 data[CANMSGRAM_MAXDATASIZE];     /* Data buffer in Bytes */
} CanMessageObject;
typedef enum {
    CAN_12 = 0,
    CAN_34 = 1,
    CAN_56 = 2,
    CAN_78 = 3
} CanChannelIndex;
```

A CAN kapcsolótábla-bejegyzést leíró struktúra.

```
typedef struct {
        CanChannelIndex source;
        uint32 messageIDBits;
        uint32 messageIDMask;
        uint8 messageXTD;
        uint8 messageRTR;
        CanChannelIndex destination;
        uint32 actionCycle;
        uint32 actionCounter;
        uint32 delayTime;
        uint8 repeatNumber;
        uint32 repeatInterval;
        uint8* overrideMask;
        uint8* overridePattern;
        uint8 overrideArraysLength;
} CanSwitchElement;
```

A CAN várakozási sorba tehető késleltetni kívánt üzenetek struktúrája.

```
typedef struct {
        uint32 delayTime;
        CanMessageObject message;
        CanChannelIndex destination;
} CanQueueElement;
```

A CAN forgalomnaplózás során felküldött elemek struktúrája.

```
typedef struct {
        uint64_t lifetime;
        CanMessageObject message;
        CanChannelIndex channel;
} CanTrafficLogElement;
```

A magasszintű CAN-vezérlő forráskódja

A CanFunctions.h és CanFunctions.c állományok, mint magasszintű CAN-vezérlőt megvalósító fájlok integrálják egybe az összes megvalósított funkciót, és teszi egyszerűen elérhetővé a felhasználói alkalmazások számára azokat.

A CanFunctions.h állomány.

```
#ifndef CANFUNCTIONS_H_
#define CANFUNCTIONS_H_

/**
 * @file CanFunctions.h
 *
 * @brief This header file contains the required functions above the CAN network
 * These are "smart functions" that can be done during the switching of a CAN message.
 *  - Initialization of the whole CAN periphery and all the other CAN SW components
 *  - Ingress function to do all the required modification of the message during the message
switching.
 *
 * $Author:  tamas.banyai
 * $Date:  2016. 10. 18.
 * $Revision:  $
 *
 * @copyright ThyssenKrupp Presta
 *
 */

/**** INCLUDES **************************************************************/
#include <Std_Types.h>
#include <SPC58Ne84_v3.h>

#include <LifetimeCounter.h>

#include "CanGpio.h"
#include "CanMsgRam.h"
#include "CanModuleHandler.h"
#include "CanQueue.h"
#include "CanSwitchList.h"
#include "CanTrafficLogger.h"

/**** PREPROCESSOR DIRECTIVES ***********************************************/

/**** TYPE DEFINITIONS ******************************************************/

/**** EXPORTED FUNCTION PROTOTYPES ******************************************/
extern  void  CAN_Init(uint32  can12baudrate,  uint32  can34baudrate,  uint32  can56baudrate,
uint32 can78baudrate);

extern void CAN_Ingress(void);

/**** EXPORTED OBJECTS DECLERATIONS *****************************************/
extern boolean isCanLogRequired;

#endif /* CANFUNCTIONS_H_ */
```

A CanFunctions.c állomány.

```
/**
 * @file CanFunctions.c
 *
 * @brief This source file contains the required functions above the CAN network
 * These are "smart functions" that can be done during the switching of a CAN message.
 * - Initialization of the whole CAN periphery and all the other CAN SW components
 * - Ingress function to do all the required modification of the message during the message
 *   switching.
 *
 * $Author:  tamas.banyai
 * $Date:  2016. 10. 18.
 * $Revision:  $
 *
 * @copyright ThyssenKrupp Presta
 *
 */

/**** INCLUDES ***********************************************************/
#include "CanFunctions.h"

/**** PREPROCESSOR DIRECTIVES ********************************************/

/**** GLOBAL VARIABLES ***************************************************/
boolean isCanLogRequired = FALSE;

/**** EXTERNAL GLOBAL VARIABLES ******************************************/

/**** LOCAL VARIABLES ****************************************************/

/**** LOCAL FUNCTION DECLARATIONS ****************************************/
static void CAN_IngressChannel(CanChannelIndex currentCanChannel);
static void CAN_IngressOutput(CanSwitchElement* switchTableEntry, CanMessageObject*
receivedMessage);

/**** EXTERNAL FUNCTION DECLARATIONS *************************************/

/**** LOCAL FUNCTION DEFINITIONS *****************************************/
static void CAN_IngressChannel(CanChannelIndex currentCanChannel) {
    boolean isMessageFromCanChannel = FALSE;
    CanMessageObject receivedMessage;

    CanSwitchElement switchTableEntry;
    sint32 switchTableEntryIndex;
    boolean isSwitchTableEntryExists = FALSE;

    //try to receive a message from the current channel
    isMessageFromCanChannel = CANMODULE_ReceiveMessage(currentCanChannel, &receivedMessage);

    if (isMessageFromCanChannel) {
        //if there was a received message

        if (isCanLogRequired) {
            CanTrafficLogElement receiveLog;
            //log the received message in the CAN logger if the user required
            receiveLog.lifetime = LTCNTR_GetTime();
            receiveLog.message = receivedMessage;
            receiveLog.channel = currentCanChannel;
```

```
            CANLOG_Send(receiveLog);
        }

        //find the first entry in the switch table that matches for that message
        switchTableEntryIndex = CANSWITCH_FindElement(currentCanChannel, &receivedMessage);
        isSwitchTableEntryExists = CANSWITCH_GetElement(switchTableEntryIndex,
&switchTableEntry);

        //if there was an entry for that message in the switch table
        if (isSwitchTableEntryExists) {
            //do the manipulation and the forwarding
            CAN_IngressOutput(&switchTableEntry, &receivedMessage);
            //also update with the decreased action counter the switch table entry
            CANSWITCH_UpdateActionCounter(switchTableEntryIndex,
switchTableEntry.actionCounter);
        }
        //if there was not any switch table entry, then don't do anything.
    }
    //if there was no received message, then move on until the next time.
}

static void CAN_IngressOutput(CanSwitchElement* switchTableEntry, CanMessageObject*
receivedMessage) {
    if (switchTableEntry->actionCounter == 0) {
        //Do the manipulating functions at every x.th time.
        uint32 overrideLength = 0;
        uint32 i = 0;

        switchTableEntry->actionCounter = switchTableEntry->actionCycle - 1;

        //do the override function
        if (switchTableEntry->overrideArraysLength < receivedMessage->data_length) {
            overrideLength = switchTableEntry->overrideArraysLength;
        } else {
            overrideLength = receivedMessage->data_length;
        }
        for (i = 0; i < overrideLength; i++) {
            receivedMessage->data[i] =
                    ((receivedMessage->data[i]) & (~(switchTableEntry->overrideMask[i]))) |
                    ((switchTableEntry->overridePattern[i]) &
                     (switchTableEntry->overrideMask[i]));
        }

        //do the repeating function
        for (i = 0; i < switchTableEntry->repeatNumber; i++) {
            if (switchTableEntry->delayTime > 0) {
                //do the delay function
                CanQueueElement delayedMessageElement;
                delayedMessageElement.delayTime = switchTableEntry->delayTime;
                delayedMessageElement.message = *receivedMessage;
                delayedMessageElement.destination = switchTableEntry->destination;
                CANQUEUE_Add(delayedMessageElement);
            } else {
                //do the forward function
                CANMODULE_SendMessage(switchTableEntry->destination, receivedMessage);
            }

            switchTableEntry->delayTime += switchTableEntry->repeatInterval;
            //repeat with interval
```

```
        }
    } else {
        (switchTableEntry->actionCounter)--;
        //By default (not the x.th time) just forwarding.
        CANMODULE_SendMessage(switchTableEntry->destination, receivedMessage);
    }
}

/**** GLOBAL FUNCTION DEFINITIONS *********************************************/
void CAN_Init(uint32 can12baudrate, uint32 can34baudrate, uint32 can56baudrate, uint32
can78baudrate) {
    CANMSGRAM_Init();
    CANMODULE_DisbaleClockCalibration();
    CANGPIO_Init();

    CANSWITCH_Init();
    CANQUEUE_Init();

    CANMODULE_Init(CAN_12, can12baudrate);
    CANMODULE_Init(CAN_34, can34baudrate);
    CANMODULE_Init(CAN_56, can56baudrate);
    CANMODULE_Init(CAN_78, can78baudrate);
}

void CAN_Ingress(void) {
    //Do the ingress function on every channel
    CAN_IngressChannel(CAN_12);
    CAN_IngressChannel(CAN_34);
    CAN_IngressChannel(CAN_56);
    CAN_IngressChannel(CAN_78);
}

/* End of file CanFunctions.c */
```