MISKOLCI EGYETEM

GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

Valós idejű ipari Ethernet rendszerek kritikus idő meghatározási módszereinek kidolgozása

című PhD értekezés tézisei

Készítette:

Ferenczi István

okleveles villamosmérnök

Hatvany József Informatikai Tudományok Doktori Iskola

Iskolavezető:

Prof. Dr. Szigeti Jenő

egyetemi tanár

Témavezető:

Dr. Vásárhelyi József

egyetemi docens

Miskolc

2016

„… a folyamatirányító rendszerek szoftversajátossága, hogy alapvetően eseményvezéreltek. Egy-egy esemény bekövetkezésekor, amelynek időpontja előre nem tervezhető, egy technológiailag ésszerű válaszidőn belül a számítógépes rendszernek garantáltan válaszolnia kell…”

Ajtonyi István emlékére

NYILATKOZAT

Alulírott **FERENCZI ISTVÁN** büntetőjogi felelősségem tudatában kijelentem, hogy a **Hatvany József Informatikai Tudományok Doktori Iskolába** beadott PhD értekezés önálló munkám eredménye, az irodalmi hivatkozások egyértelműek és teljesek.

Dátum: 2016.05.05

……
aláírás

KÖSZÖNETNYILVÁNÍTÁS

Az értekezés a Miskolci Egyetem Hatvany József Informatikai Tudományok Doktori Iskola keretén belül készült. Ebből az alkalomból szeretnék köszönetet mondani a doktori iskola minden tanárának és munkatársának, hogy munkájukkal segítették tevékenységemet. Köszönetet mondok volt témavezetőmnek Dr. Ajtonyi István professzor úrnak, aki sajnos már nem lehet közöttünk. Ő volt az, aki elindított engem ezen az úton és átsegített a kezdeti nehézségeken. Köszönöm jelenlegi témavezetőmnek Dr. Vásárhelyi Józsefnek, aki miután megkapta a felkérést, mindent megtett azért, hogy folytassam az elkezdett munkát és elkészüljön az értekezésem.

Köszönettel tartozok a doktori iskola egykori vezetőjének, Dr. Tóth Tibor professzor úrnak, aki a doktori szemináriumokon és fórumokon mindig szigorúan számon kérte a félévi munkámat. Köszönöm Dr. Ormos László kollégámnak, aki szakmailag is sokat segített abban, hogy minél eredményesebb munkát végezzek.

Végül, de nem utolsó sorban köszönöm feleségemnek és családomnak, akik kitartásra bíztattak, mellettem álltak és segítették munkámat.

RÖVIDÍTÉSEK JEGYZÉKE

ADPU	Application Protocol Data Unit
AR	Application Relation
ARP	Address Resolution Protocol
ASIC	Application Specific Integrated Circuit
CAN	Controller Area Network
CCP	Cycle Check Point
CIP	Common Industrial Protocol
CPU	Central Processor Unit
CR	Communication Relation
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DA	Destination Address
DCS	Distributed Control System
DI	Digital Input
DMA	Direct Memory Access
DO	Digital Output
DSAP	Destination Service Access Point
DSCP	Differentiated Service Code Protocol
EDF	Earliest Deadline First
EPL	Ethernet Power Link
FB	Function Block
FCFS	First Come First Served
FCS	Frame Check Sequence
FMMU	Fieldbus Memory Management Unit
FTP	File Transfer Protocol
HMI	Human-Machine Interface
HTTP	Hyper Text Transfer Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFG	Inter Frame Gap
IO	Input-Output
IOCS	Input-Output Consumer Status
IOPS	Input-Output Provider Status
IP	Internet Protocol
IRT	Isochronous Real-Time
LAN	Local Area Network
LLC	Logical Link Control
MAC	Medium Access Control
MIME	Multipurpose Internet Mail Extensions
MTU	Maximum Transfer Unit
NRT	Non Real-Time
OB	Organization Block
ODVA	Open Devicenet Vendors Association
ORTE	Ocera Real-Time Protocol
OSI	Open Systems Interconnection
PII	Process Image of Input
PIO	Process Image of Output

PLC	Programmable Logic Controller
PTCP	Precision Transparent Clock Protocol
PTP	Precision Time Protocol
QoS	Quality of Service
RAM	Random Access Memory
RARP	Reverse Address Resolution Protocol
RM	Rate Monolithic
ROM	Read Only Memory
RPI	Requested Packet Interval
RT	Real-Time
RTA	Real-Time Acyclic
RTEP	Real-Time Ethernet Protocol
SA	Source Address
SMTP	Simple Mail transfer Protocol
SSAP	Source Service Access Point
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
ToS	Terms of Service
TTP	Time Triggered Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VLAN	Virtual Local Area Network
WC	Working Counter
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Tartalomjegyzék

BEVEZETÉS

Ahogy a jelenkori társadalom szereplői sem elszigetelt módon végzik tevékenységüket, úgy a korszerű ipari irányító rendszerek is igénylik a közöttük lévő kommunikációs kapcsolatok kialakítását. Ezek a kapcsolatok hálózatokba rendeződnek és mindegyik elemének jól meghatározott szerep tulajdonítható. A XX. század utolsó évtizedeinek egyik meghatározó tényezője volt a számítógépes hálózatok megjelenése. A világháló rövid időn belül az információszerzés és az információáramlás egyik legnépszerűbb forrásává vált. Segítségével a kommunikációs folyamatok kiléptek a pont-pont kapcsolatok korlátai mögül és lehetővé vált olyan stratégiák kialakítása, melyek egyszerre, látszólag azonos időben akár több hálózati szereplő között is képesek az információáramlást biztosítani.

A disszertáció célja és szükségessége

A számítógépes hálózatok gyors ütemben történő elterjedése hozzájárult a hálózati eszközök és az üzemeltetésükhöz szükséges költségek jelentős csökkenéséhez. Ez a tény egyre inkább az ipari kommunikációs rendszerek fejlesztőit is az Ethernet hálózat irányába terelte. A vezető ipari irányítóberendezéseket gyártó vállalkozásokon belül kutatócsoportok alakultak, hogy kidolgozzák azokat a módszereket, amelyek segítségével a valós idejű adattovábbítás maradéktalanul megvalósítható az egyébként nem determinisztikus Ethernet hálózaton. Ezeknek a kutatócsoportoknak a működése meglehetősen zártkörű. Publikációs tevékenységük korlátozott, inkább csak elvi megoldásokat, vagy már a kész termékeket hozzák nyilvánosságra, de a módszerekről csak nagyon ritkán közölnek információt. Ilyen körülmények között nem csoda, hogy nemcsak a magyar, de a nemzetközi szakirodalom is meglehetősen korlátozott ezen a területen. Témavezetőm Dr. Ajtonyi István professzor úr javaslatára – aki időközben sajnos elhunyt – azért választottam ezt a tématerületet, hogy átfogó képet alkothassak a valós idejű ipari Ethernet rendszerekkel kapcsolatos alapvető kritériumok meghatározását illetően.

A disszertáció célja tehát kidolgozni mindazokat a kritikus idő meghatározási módszereket, amelyek segítségével biztosítható az ipari Ethernet rendszerek valós idejű működése az ipari irányítás minden területén, a számítógépes folyamatirányítástól kezdve a motorvezérlésekig. A kidolgozott módszerek igazolása konkrét gyakorlati mérésekkel vagy számítógépes modell segítségével történik.

A disszertáció első részében bemutatom az Ethernet hálózatokra általában jellemző elveket és megoldásokat, ezeknek fejlődését, valamint azokat a próbálkozásokat, amelyek elvezettek az irodai Ethernet determinisztikus működéséhez. A második fejezetben

részletesen bemutatok három olyan konkrét valós idejű ipari Ethernet rendszert, amelyek a legelterjedtebbeknek tekinthetők és alkalmazásuk lefedi a teljes valós idejű skálát. A további fejezetek a tézisekhez kapcsolódó leírásokat tartalmazza.

A disszertáció négy tézisre épül. Az első tézisben (III. fejezet) kidolgoztam az osztott intelligenciájú ipari irányítórendszerek valós idejű működési feltételeit. Meghatároztam egy valós idejű alaprendszert és az itt megállapított kritériumokat kiterjesztettem a programozható logikai vezérlőkre. Ugyancsak az első tézisen belül történik a master-slave vagy a provider-consumer elven működő ipari Ethernet alrendszerek valós idejű viselkedési feltételeinek elvi meghatározása.

A második tézis (IV. fejezet) az IO kontrollerek ciklusidő-változásainak meghatározását, mérési módszereit és mérésekkel történő igazolását tartalmazza. A ciklikus működést meghatározó legfontosabb jellemző a kontroller ciklusideje. Ennek mérési módszerei kerülnek bemutatásra, valamint a mérési eredmények kiértékeléséből származó általános következtetések felhasználása a valós idejű rendszerek tervezésénél.

A harmadik tézisben (V. fejezet) bemutatásra kerülnek a nem szinkronizált ipari Ethernet rendszerek reakcióidejének vizsgálati lehetőségei. Ezek alapján lehetővé válik a rendszerparaméterek meghatározása és ezeknek az alkalmazása egy konkrét Profinet IO rendszernél. Egy valós idejű rendszer egyik legfontosabb követelménye, hogy a reakcióidő a technológia által támasztott határidőkön belül legyen. A reakcióidők ismeretének függvényében tudjuk meghatározni a nem szinkronizált valósidejű Ethernet rendszerek korlátait, beállítani mindazokat a paramétereket, amelyek segítségével optimális működést biztosíthatunk a különböző sebességű folyamatok számára.

A negyedik tézisben (VI. fejezet) a szigorúan valós idejű ipari Ethernet rendszerek optimális buszfrissítési ciklusának meghatározása történik különböző elvi megoldások alkalmazása mellett. Itt főleg a csomópontok száma, a vezérlési adatok mennyisége, valamint a bitsebesség szerinti szempontokat elemzem. Az itt meghatározott szabályokat felhasználva kidolgoztam egy olyan számítógépes modellt, amely segítségével vizsgálhatóak ezek a rendszerek a fentebb említett szempontok szerint különböző szituációkban.

Idevonatkozó fontosabb szakirodalom rövid áttekintése

Mint ahogy azt már említettem, az ipari Ethernettel kapcsolatos magyar nyelvű szakirodalom meglehetősen szegényes ezen a területen. Míg az informatika, a számítógépes hálózatok és az internet világából számos színvonalas magyar nyelvű könyv, folyóirat, internetes kiadvány létezik [2, 4, 10, 11, 12], addig az ipari Ethernettel kapcsolatosan egyedül csak az Ajtonyi professzor úr köteteiből [3, 14, 28, 53] sikerült használható ismereteket felhasználni.

Az idegen nyelvű szakirodalom az internetnek köszönhetően már sokkal gazdagabb választási lehetőséget biztosít [15, 23, 26, 32, 57, 65]. Megemlítenék néhány elismert szerzőt is, akiknek a munkái nagyban hozzájárultak a disszertációm tudásbázisának megalapozásához. Itt emelném ki Raimond Pigan és Mark Metter „Automating with Profinet” című könyvét [37], amelyben részletesen leírják a Profinet rendszer működési alapjait, protokolljait és alkalmazási területeit. De számos hasznos megoldást ismertem meg az EtherCAT rendszerrel kapcsolatosan Martin Rostan a Beckhoff Automation cég vezetőjének kiadványaiból is [33, 40, 41]. Sokat segített Perry Marshall, és John Rinaldi könyve is [9], amely röviden, tömören csak a lényeges szempontokat kiemelve mutatja be a valós idejű Ethernet rendszerek sarkalatos problémáit.

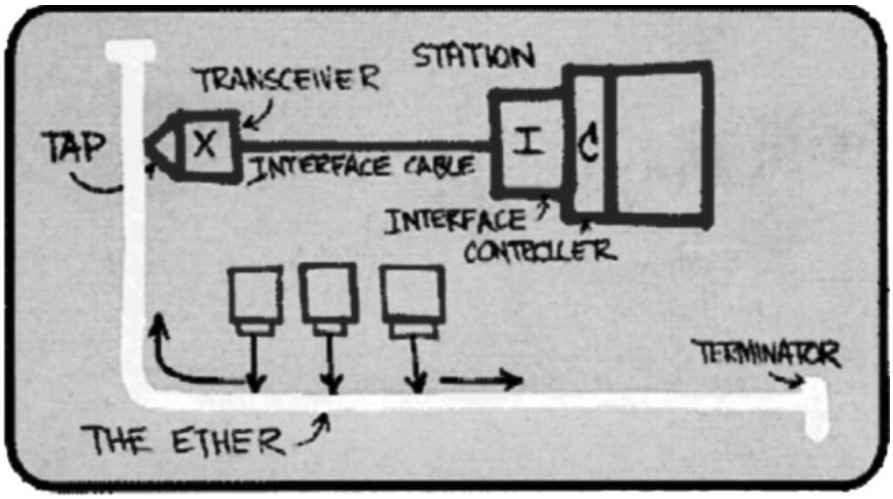
I. FEJEZET

ÚT AZ IPARI ETHERNETIG

Ebben a fejezetben bemutatom az Ethernet hálózati rendszer kialakulásával és fejlődésével kapcsolatos fontosabb mozzanatokat, valmint azt, hogy milyen próbálkozások születtek az egyébként nem determinisztikus Ethernet rendszer ipari kommunikációs célokra történő felhasználása érdekében.

1.1. Az Ethernet kialakulása és fejlődése

Már lassan négy évtizede annak, hogy 1976 nyarán a Xerox cég mérnöke, Robert Metcalf egy számítógépes konferencián (National Computer Conference) bemutatta az első Ethernet hálózatra vonatkozó elképzelésének vázlatát (1.1. ábra). A hetvenes évek elején a Xerox cég mérnökei egy vezetékes hálózat kifejlesztését tűzték ki célul. Azt szerették volna megvalósítani, hogy a Xerox Alto számítógépeket összekapcsolják egymással, a szerverrel és a lézernyomtatóval. Így lehetségessé válhatott, hogy egy nyomtatóra, több, akár nagyobb távolságra lévő számítógépről is lehetett nyomtatni. Az átviteli közegként a két végén impedancia illesztéssel lezárt vastag koaxiális kábelt tasználtak. Mindegyik állomást hálózati interfész vezérlővel láttak el, amely az illesztőn (transceiver) keresztül kapcsolódott a közeghez. Az interfész órajelét az Alto órajeléből származtatták, amely már egy igen jelentős 2,94 Mb/s-os bitsebességet tudott biztosítani.



1.1. ábra. A Metcalf által bemutatott Ethernet vázlat
(forrás: http://www.ieee802.org/3/ethernet_diag.html)

Az Alto Aloha Network nevet viselő kísérleti hálózat jól vizsgázott, így 1973-ban Metcalf el is nevezte a rendszert **Ethernet**nek (éterhálónak), arról az anyagról, amelyet az elektromágneses hullámok terjedési közegének véltek korábban.

A Xerox féle Ethernet olyan sikeres volt, hogy 1978-ban szabadalmaztatták. Röviddel ezután Robert Metcalf kilépett a Xeroxtól, új céget alapított és 1979-ben sikerült meggyőznie három másik nagy céget (Digital Equipment Corporation, Intel és Xerox) a szabadalom támogatására, hogy elfogadtathassák általános szabványként. Így született meg

1982-ben az első igazi, 10 Mbit/s-os Ethernet szabvány az Ethernet II vagy más néven DIX V2.0. Kisebb változtatások után a szabványt az IEEE (Institute of Electrical and Electronics Engineering) is elfogadta és 1983-ban kidolgozta a máig is használatos Ethernet hálózati szabvány alapját az IEEE 802.3-at [1].

A szabványnak azóta számos változata jelent meg. Az eredeti 10Base5 nevű, vastag koaxiális kábeles szabványt (Thicknet) rövidesen (1985-ben) követte a 10Base2, ugyancsak 10 Mbit/s-os vékony koaxiális kábeles (Thinnet) szabvány, amely a kábel viszonylag olcsó ára miatt nagyon gyorsan elterjedt és vált népszerűvé főleg az irodai hálózatok kialakításában. Az áttörést az 1990-ben megjelenő csavart érpáros 10Base-T szabvány jelentette, amely már jóval előnyösebb tulajdonságokkal rendelkezett mint a koaxiális kábeles elődje.

A Fast Ethernet története 1993 júniusában kezdődött, amikor több mint 50 fejlesztő és szolgáltató fogott össze egy közös cél érdekében, hogy kidolgozzák a 100Mbit/s-os Ethernetet. Az eredmény 1995-ben vált teljessé, amikor megjelent az IEEE 802.3u 100Base-TX CAT5-ös csavart érpáros szabvány, amelyet még napjainkban is igen elterjedten használnak szinte minden területen. A következő jelentősebb lépés 1999-ben történt az IEEE 802.3ab Gigabit Ethernetes szabvány megjelenésével, majd ezt követi 2003-ban 10 Gigabit Ethernet. És a fejlődés nem áll meg…

A történelem itt is megismétli önmagát. A 90-es évek elején kezdtek megjelenni az első vezeték nélküli adatgyűjtő terminálok. Ezek a rendszerek nagyrészt egyediek voltak, mindegyik gyártó a saját elgondolása szerint valósította meg a kapcsolatot. Ez azt jelentette, hogy egy mobil adatgyűjtő csak a saját gyártójának eszközeivel kiépített hálózatában működött, a másikében nem. Ennek áthidalására fejlesztette ki az IEEE munkacsoport az évtized közepére az első szabványos WLAN protokollt, az IEEE 802.11-es családot, amelyet 1997-ben véglegesítettek és ratifikáltak. A 802.11 protokoll logikája és az eszközök együttműködését garantáló Wi-Fi a nagysikerű és szinte kizárólagossá vált 802.3 Ethernet protokoll működésén alapult. Az első 802.11 szabvány három fizikai réteget definiált: egyet az infravörös tartományban, kettőt pedig a szabad 2.4 GHz-es tartományban. A szabadon felhasználható frekvencia tartományban az adási teljesítmény nem haladhatja meg a 100mW értéket. Az adatátviteli sebesség induláskor 1 Mb/s volt, ami később 2 Mb/s-ra nőtt. Mindössze két év telt el, és 1999-ben két idevonatkozó szabvány is megjelent. A 802.11b szabvány az előző továbbfejlesztett változata volt. Az adatátviteli sebesség egy új modulációs technikának köszönhetően 11 Mb/s-ra növekedett. A másik megoldás az 5 GHz-es tartományban működő 802.11a protokoll volt, amely már ekkor 54 Mb/s-os sebességet produkált. Ez azonban, az 5 GHz-es korlátozott hozzáférésű (főleg katonai radarok által használt) frekvencia tartomány miatt nem tudott igazán elterjedni.

2003 folyamán megjelent a 802.11g szabvány, amely a 802.11a modulációs eljárását alkalmazva megsokszorozta a 802.11b rendszerek sebességértékét, elérve ebben a frekvenciasávban is az 54 Mb/s-os értéket. Az új szabvány megtartotta kompatibilitását az előzővel, vagyis a 802.11g eszközök automatikusan felismerik és hozzákapcsolódnak a 802.11b elérési pontokhoz, és fordítva, a régi 802.11b eszközöket felismerik az új 802.11g cellák.

Az adattovábbítás sebessége az évek során tovább nőtt. A 2009 szeptemberében ratifikált 802.11n rendszer 20 MHz-es sávszélesség mellett, akár 300 Mb/s sebességet tud biztosítani az éter hullámain. A vezeték nélküli világ egyre nagyobb teret hódít az ipari kommunikációs rendszerek területén is. Lehetővé vált olyan helyeken is érzékelőket és távadókat elhelyezni, ahol egyébként a vezetékek használata miatt ez eddig lehetetlen volt vagy csak nagyon nehezen és költséges megoldások mellett lehetett megvalósítani.

1.2. Adattovábbítás Etherneten keresztül

Ahhoz, hogy az Ethernet hálózaton keresztül az adatok egyik állomástól a másikig zavartalanul és biztonságosan eljussanak, számos bonyolult részfeladatot kell megoldania a küldőnek és a fogadónak is egyaránt. Ráadásul ezeknek a részfeladatoknak a lebonyolítása során, sok esetben együttműködés is szükséges. Azért, hogy ezeket a feladatokat a különböző gyártók által készített interfészek mindegyike kezelni tudja, szabványra volt szükség.

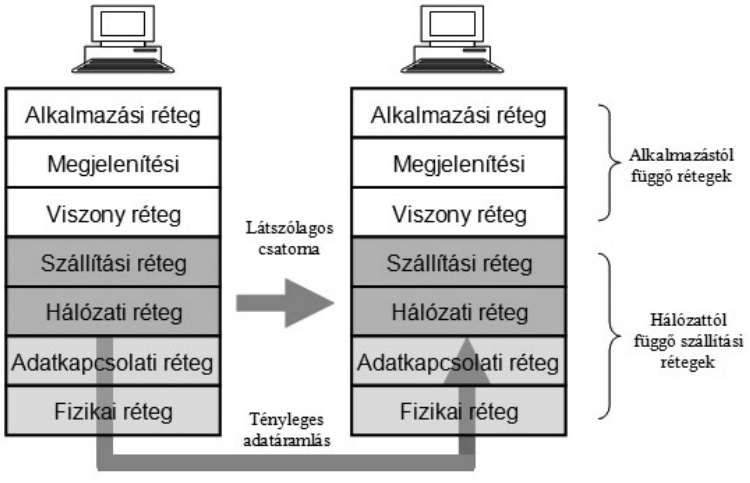
1.2.1. Az OSI modell

A Nemzetközi Szabványügyi Szervezet (ISO) volt az, amely vállalkozott erre a feladatra. Olyan elképzelést dolgoztak ki, mely szerint a hálózati kommunikációt rétegekre bontják, ezen belül pontosan definiálják az egyes rétegek feladatát, illetve a rétegek mentén zajló kommunikációt. A szabványt kicsit megkésve, 1984-ben bocsátották ki és ISO OSI (Open System Interconnection) modellnek nevezték el, mivel a nyílt rendszerek kommunikációjával és összekapcsolásának részleteivel foglalkozik.

Az OSI modell a különböző protokollok által nyújtott funkciókat egymásra épülő rétegekbe sorolja. Minden réteg csak és kizárólag az alsóbb rétegek által nyújtott funkciókra támaszkodhat, és az általa megvalósított funkciókat pedig csak a felette lévő réteg számára nyújthatja. Ez a modell alapjaiban meghatározó volt a hálózatokkal foglalkozó ipar számára. A kezdeményezés legfontosabb eredménye az volt, hogy meghatározták azokat a specifikációkat, amelyek pontosan leírták, hogyan léphet egy réteg kapcsolatba egy másik, azonos szintű réteggel. Ez gyakorlatilag azt jelenti, hogy egy adott gyártó által készített réteg programja együtt tud működni egy másik gyártó által készített programmal. A végeredmény egy olyan rendszer lett, mely két alapelvből állt össze:

- kialakult a 7 rétegből álló struktúra, (1.2. ábra)
- meghatározták az egyes résztevékenységeket ellátó protokollokat.

A modellt az IEEE is elfogadta, beépítette és erre alapozta az IEEE 802.3 szabványsorozatát [1].



1.2. ábra. Az információ OSI modell szerinti látszólagos és tényleges áramlási útja

Az 1.2. ábrán látható rétegek funkcióiról csak röviden tennék említést, ugyanis az ipari Ethernetes rendszereknél, csak néhány, főleg az alsóbb rétegek játszanak meghatározó szerepet. Fentről lefelé haladva ezek a következők:

7. Alkalmazási réteg (Application layer): a felhasználói programok számára biztosítja a hálózati kommunikációt. Ide tartoznak azok a szabványosított protokollok is, mint például a HTTP, FTP, Telnet, SMTP.

6. Megjelenítési réteg (Presentation layer): foglalkozik az adatok szintaktikájával és szemantikájával, azért, hogy a különböző adatábrázolási kódokat alkalmazó rendszerek is kommunikálni tudjanak egymással. Az adatokat átalakítja olyan formátumokká, amelyeket az alkalmazások már közvetlenül tudnak értelmezni (MIME, XML, stb.) Az adatkonverziókon túl, ez a réteg képes az adatok tömörítésére és esetleges titkosítására is.

5. Viszony réteg (Session layer): Lehetővé teszi a kapcsolati viszony létesítését két számítógép között. Vagyis, hogy két állomás között kiépül a kommunikációs kapcsolat, adatok jutnak el egyik géptől a másikig (szállítási réteg feladata), majd az adatáramlás befejezésével a kapcsolat megszűnik. Háromféle kommunikációs módot tesz lehetővé: szimplex (csak egyirányú), half-duplex (kétirányú, de egy időben csak egy irány aktív) és full-duplex (kétirányú, egyidejűleg történő kommunikáció).

4. Szállítási réteg (Transport layer): Adatokat fogad a felső rétegtől, kisebb darabokra vágja szét, átadja a hálózati rétegnek, és gondoskodik róla, hogy az adatcsomagok megbízhatóan és hibamentesen eljussanak az egyik állomástól a másikig. Ezt úgy valósítja meg, hogy kapcsolatot teremt a hálózati eszközök között, nyugtázza a csomagok kézbesítését és újra elküldi az elveszett vagy sérült csomagokat. Küldéskor a szállítási réteg a nagyméretű adatcsomagokat kisebb csomagokká darabolja a hatékonyabb szállítás érdekében. A fogadó számítógép ezeket a csomagokat összeilleszti, és megvizsgálja, hogy minden adatcsomag megérkezett-e? Ezen az elven működik a TCP protokoll [5]. A TCP összeköttetés elsődleges célja a biztonságos szállítás. Feladata, hogy a kapott szelvényeket megfelelő sorrendben rakja össze az eredeti üzenetnek megfelelően. Küldéskor, az adatátvitel hibátlanságának biztosítása érdekében kézfogásos (handshaking) módszert, úgynevezett pozitív nyugtát használ. Az adó elküldi az adatot, közben elindít egy időzítőt is és várakozik a nyugtázásra. Ha nem kap választ az időzítés lejárta előtt, akkor újraküldi a szelvényt. A TCP kapcsolat kétirányú adatáramlást biztosít, pont-pont összeköttetésben, így nem támogatja a többesadást és a szórtadást [2].

Bizonyos esetekben a sebesség és a hatékonyság fontosabb, mint a megbízhatóság. Ilyenkor a küldő fél nem foglalkozik az adatok nyugtázásával, csupán elküldi a csomagokat. Ez az alap filozófiája az UDP protokollnak. A folyamatos nyugtázás, az elveszett szegmensek újraküldése, a torlódásvédelem lassítja az adatátvitelt. Ott ahol minél gyorsabb átvitelre van szükségünk (pl. médiafájlok), vagy ahol a felhasználói adatmennyiség csekély (pl. PLC vezérlési adatok) célszerűbb az egyszerűbb felépítésű UDP protokollt alkalmazni. Ha az üzenetek rövidek, nincs szükség az adatok sorba rendezésére sem, mert azok egy szegmensen belül elférnek. Az UDP leírását az RFC 768-as szabvány tartalmazza, amelyet 1980-ban bocsátottak ki [6].

3. Hálózati réteg (Network layer): Feladata az előző réteg által előkészített csomagok eljuttatása a forrástól a célig, mialatt a csomagok sok esetben több közbeeső csomóponton (router) is áthaladnak. Ha a forrás és a cél eltérő típusú hálózatokban van, ez a réteg gondoskodik a hálózatok közti különbségből fakadó feladatok elvégzésére is.

A hálózati réteg legismertebb és talán a legelterjedtebben használt protokollja az IPv4. Ebben a protokollban minden egyes csomag külön-külön címzést kap, amely egyértelműen azonosítja azt a két csomópontot, amely között felépül az éppen aktuális kapcsolat. Tehát nem feltétlenül azonos a forrás és a cél címével. A címzéshez 4 bájtot (32 bit) használ, ezeket rendszerint decimális formában, a bájtok között ponttal elválasztva, IP címként ismerjük [2]. Az IPv4 elterjedtségét mi sem bizonyítja jobban, mint az, hogy már az IPv6, 16 bájtos változatot is használják, mert a 4 bájtos címzés adta lehetőségek nemrég kimerültek.

2. Adatkapcsolati réteg (Data Link layer): Az adatkapcsolati réteg elsődleges feladata, hogy a hálózat csomópontjai között hibamentes átvitelt biztosítson, illetve az adatkeretek kezelése. Az adó kapcsolati rétege ezeket a kereteket bitenként továbbítja a fizikai réteg felé. Közben megfelelő bitmintákkal ellátja a kerethatárokat, azért, hogy a vételnél, a fizikai réteg felől kapott biteket, a vevő adatkapcsolati rétege vissza tudja alakítani keretekké. Továbbá a küldő gép adatkapcsolati rétege a keret tartalmát felhasználva elvégez egy számítást is az adatkereten, melynek eredményét hozzácsatolja a kerethez. A fogadó gép adatkapcsolati rétege megismétli ezt a műveletet és a kapott eredményt összehasonlítja a kerethez csatolt értékkel, ezzel biztosítva a keretek sérülésmentességének ellenőrzését. Ha a keret sérülésmentes, akkor a fogadó gép adatkapcsolati rétege küld egy nyugtajelet (acknowledgement frame) a küldő gép adatkapcsolati rétegének.

Mivel egy hálózatban az adott fizikai csatornát többen is akarják használni, a hozzáférést az adatkapcsolati réteg egyik különleges alrétege, a *Közegelérés-vezérlési alréteg (Medium Access Control, MAC)* kezeli. Akárcsak a hálózati réteg, a MAC szintű kapcsolat is használ címzést az eszközök azonosítására. Itt az eszköz valóságos fizikai címe kerül felhasználásra. Az IEEE szabványban határozta meg, hogy mindegyik Ethernet hálózati eszköznek rendelkeznie kell egy jól meghatározott egyedi, 48 bites fizikai címmel. Ezt nevezik MAC címnek. Az első 24 bit a gyártót azonosítja, a fennmaradó 24 bit pedig a hálózati eszköz sorozatszámát tartalmazza. Az adatkapcsolati rétegen működő ARP (Address Resolution Protocol) az adónál az előző rétegtől kapott IP címek alapján határozza meg a MAC címeket. A vevőnél ennek a fordítottja történik, a RARP (Revers Address Resolution Protocol) visszaállítja az IP címeket a MAC címek alapján [9].

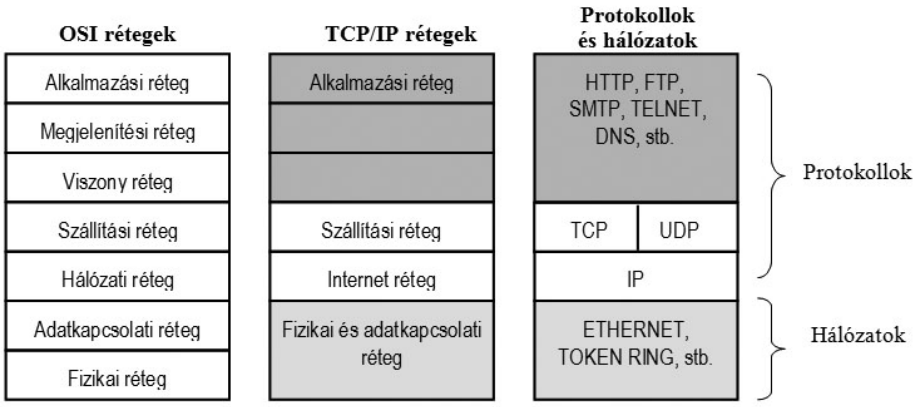
1. Fizikai réteg (Physical layer): Ezen a rétegen zajlik a tényleges, jelszintű adattovábbítás. A rétegnek biztosítani kell azt, hogy az adótól elküldött 1-es bit, a másik oldalon, a vevőnél ugyancsak 1-esként jelenjen meg. Ennek érdekében, az adónál a fizikai réteg bitértékekhez feszültségszinteket rendel, vagyis kódolja az információt, a vevőnél pedig a feszültségszintekből visszaállítja a bináris információt. Az alkalmazott kódolási technikát szabvány írja elő. Pl. a 10 Mb/s-es Ethernet a Manchester-kódolást, a 100Base-TX pedig a 4B/5B kódolást használja [3].

1.2.2. A TCP/IP hivatkozási modell

Megjelenése után az OSI modellt igencsak sok kritika érte. Egyrészt azért, mert túl bonyolultnak tartották, másrészt azért, mert protokolljai nem teljesen tökéletesek. A viszony réteget alig használja néhány alkalmazás, a megjelenítési réteg pedig szinte teljesen üres. Igaz, hogy eredetileg csak öt réteg volt, de mivel a nagy befolyású IBM-nek már volt egy 7 rétegű modellje ezért 7 rétegűre módosították [10].

Az időzítés sem volt túl szerencsés. Egy szabvány megjelentetésének időpontja rendkívül erősen befolyásolhatja annak sikerét. A szabványosításnak a kutatások befejezése után és a beruházások megkezdése előtt kell megtörténnie. Ez azért fontos, hogy a kellő tapasztalat birtokában egységes szabványt hozhassunk létre. Mire az OSI protokollok megjelentek, addigra a Berkeley-féle UNIX TCP/IP protokolljai már széles körben elterjedtek a kutatóegyetemeken [4].

A TCP/IP modell lényegében az OSI modellnek egy egyszerűsített változata, amely csak négy réteget tartalmaz, de megvalósítja mindazokat a funkciókat, amelyek szükségesek a biztonságos kommunikációhoz. A két modell közötti különbséget és az alkalmazott protokollokat az 1.3. ábra mutatja.

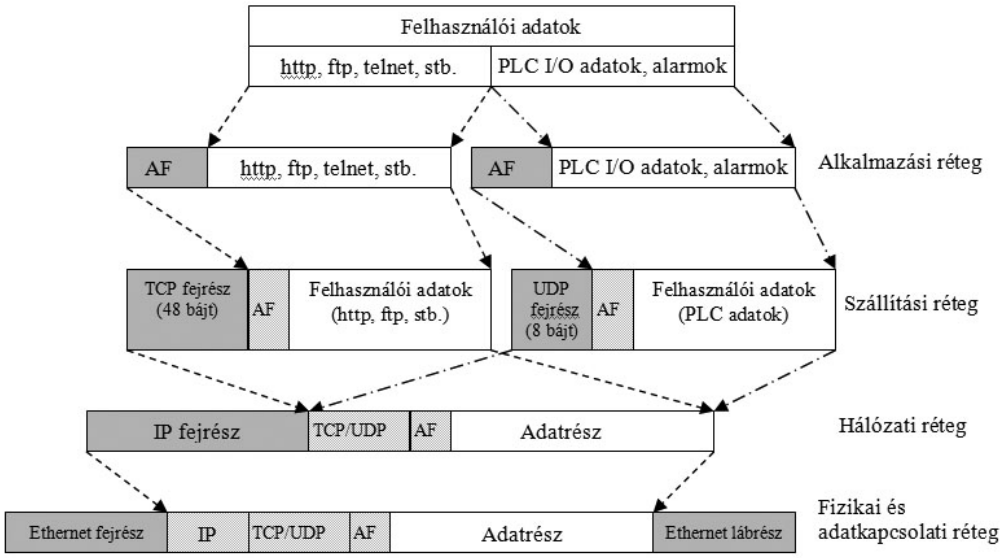


1.3. ábra. Az OSI és a TCP/IP hivatkozási modell és protokolljai

1.2.3. Adattovábbítási stratégia a TCP/IP rétegeken keresztül

Az előző fejezetben bemutatott TCP/IP rétegmodell egyaránt alkalmas általános felhasználói és az ipari Ethernetet alkalmazó rendszerek tárgyalására is. Ezért a továbbiakban ennek működési stratégiáját szeretném bemutatni, kiemelve azokat a sajátosságokat, amelyek leginkább az ipari Ethernetre jellemzőek.

Tételezzük fel, hogy a felhasználói adatok, legyenek azok általános internet információk, vagy speciális ipari alkalmazások adatai, a forrástól a célig jól meghatározott sorrendben végighaladnak a különböző rétegeken és közben járulékos információk hozzáadásával, egymásba ágyazódva (1.4. ábra), a fizikai közegen keresztül érik el a célállomás rétegeit. Ezeken áthaladva rendre lebomlanak a járulékos információk, így a célhoz csak az eredeti üzenet érkezik.



1.4. ábra. Adatáramlás a rétegeken keresztül

A szállítási réteg az alkalmazástól kapott tetszőleges hosszúságú üzenetekből álló adatfolyamból adatszegmenseket készít. A szegmens méretét az MTU (Maximum Transfer Unit) határozza meg, amely minden hálózat meghatározó jellemzője. A gyakorlatban ez néhány ezer bájtot jelent (a legtöbb esetben 1500 bájt), de maximum 64 kilobájt lehet, azért, hogy a fejléccel együtt elférjen a hálózati réteg IP adatmezőjében. Ezeket a

szegmenseket a hálózati réteg IP csomagonként továbbítja. Amikor a szelvények elérik a célállomást, az ottani szállítási réteg azokat ismét adatfolyammá rakja össze és továbbítja a felhasználói folyamatnak.

1.3. A CSMA/CD protokoll működése

Az IEEE 802.3 szabvány szerinti Ethernet hálózatba kapcsolt állomások mindegyike egyenlő jogokkal rendelkezik a közeg használatát illetően. A hozzáférés ellenőrzése és vezérlése a CSMA/CD (Carrier Sense Multiple Access with Collision Detection) közegelérési módszer szerint történik, amelynek röviden a lényege a következő.

Mindegyik állomás állandóan figyeli a hálózati forgalmat. Ha talál olyan adatcsomagot, amelyet neki címeztek leveszi, a többit figyelmen kívül hagyja. Az adatküldésre készülő állomás is hallja a közegen zajló forgalmat, és ha azt érzékeli, hogy valaki már adásban van, azaz vivőt érzékel (Carrier Sense), akkor várakozik az adás megszűnéséig. Amikor elcsendesül a közeg, megkezdi az adást, amelyet egyedül csak a megcímzett állomás fog átvenni. Minél több állomás van az adott hálózati szegmensre kapcsolva, annál nagyobb az esélye annak, hogy egy időben egyszerre két vagy több állomás is küldeni akar. Ilyenkor a közegen a villamos jelek összeadódhatnak, az adatok sérülnek, az információ használhatatlanná válik. Ezt az állapotot érzékelni kell (Collision Detection), sőt figyelmeztetni kell rá a többi állomást is, hogy ütközés történt. Az ütközést az adó érzékeli, amikor visszaolvasáskor nem ugyanazt az adatot kapja. Azért, hogy minden állomás értesüljön az ütközésről, az ütközést észlelő egy rövid ideig egy 32 bit hosszúságú un. JAM jelsorozatot helyez a hálózatra, amelynek hatására mindegyik állomás azonnal beszűnteti az adást, majd véletlenszerű késleltetési idő után újból próbálkozik.

A véletlen időzítésnek köszönhetően valamelyik adó elsőként lefoglalja és megkezdheti az adását, ezt a többiek érzékelni fogják, mielőtt megkezdenék adásukat. A véletlen várakozási idő N ütközés után maximum a következő lehet:

$$t_w = t_s \cdot (2^N - 1) \text{; ahol: } 1 \leq \text{N} \leq 16 \tag{1.1}$$

A fenti relációban szereplő t_s résidő (slot time), vagy körbejárási késleltetés, az az idő mialatt a keret a szegmens két legtávolabbi pontja között körbejár ($t_s = 5{,}12\ \mu s$ Fast Ethernet hálózatban). Ennyi idő szükséges ugyanis ahhoz, hogy mindegyik állomás biztonsággal érzékelje az ütközést.

A várakozási idő egy algoritmus szerint (Binary Exponential Backoff) kerül meghatározásra. Az első ütközés után 0 vagy 1 a szorzó, a második után 0, 1, 2, 3, a harmadik után 0, 1, ..., 7 és így tovább. A 10-edik ütközés után (1023) a szorzó nem nő tovább, a 16.-odik ütközés után az eszköz tovább már nem próbálkozik és hibaüzenettel jelzi az átvitel sikertelenségét.

Az (1.1) összefüggésben szereplő résidő függ a csatorna terjedési sebességétől (v) és a leghosszabb hálózati szegmens hosszától (L). Ennek megfelelően a résidőhöz tartozó minimális keretméret (d_F), ismerve a bitsebességet (b) a következő összefüggéssel határozható meg:

$$d_F = \frac{2 \cdot L}{v} \cdot b \tag{1.2}$$

Az IEEE 802.3 szabvány annak idején a 10 Mb/s-os Ethernet hálózatban, amelynek a hossza négy ismétlővel maximálisan 2500 m lehetett, ezt a keretméretet felkerekítve, 512 bitben határozta meg, ami azt jelenti, hogy a minimális keretméret 64 bájt nagyságú kell

legyen. Vagyis ahhoz, hogy az ütközést elkerüljük a jelenleg használatos nagyobb sebességű hálózatokban vagy csökkentenünk kell a szegmensek hosszát, vagy növelnünk kell a minimális keretméretet [11].

1.4. Az IEEE 802.3 Ethernet keret felépítése

A MAC alréteg egy másik fontos funkciója a keretek létrehozása és ezeknek jól meghatározott elkülönítése egymástól, valamint a hibaellenőrzés. Az IEEE 802.3 keretstruktúráját az 1.5. ábrán láthatjuk.

Előtag	Keret-határoló	Célcím	Forráscím	Adat-hossz	**Adatok**	Ellenőrző összeg
7 bájt	1 bájt	6 bájt	6 bájt	2 bájt	**46-1500 bájt**	4 bájt

1.5. ábra. Az IEEE 802.3 keretformátuma

Az előtag és a kerethatároló igazából nem része a keretnek. A MAC alréteg azért fűzi a keret elejére, hogy biztosítsa a vevő hardverjének ráhangolódását az adó órájára. Tulajdonképpen egy 10101010 mintájú jel 7 bájton keresztül, amelynek a Manchester kódja egy 10 Mhz-es, 5,6 µs időtartamú négyszögjelnek felel meg 10 Mb/s bitsebességnél. Ezt követi a kerethatároló bájt, amely hasonló, kivéve az utolsó bitet, amely ugyancsak 1-es. Ez jelzi, hogy itt kezdődik a keret. A célcím és forráscím az adó illetve a vevő MAC címét tartalmazza, amelyet az adó alrétegének ARP protokollja képez le az IP címből. A vevőnél pedig a RARP éppen az ellenkezőjét teszi, a MAC címből visszaállítja az IP címet a hálózati réteg számára.

A kétbájtos adathossz mező, az adatmezőben található bájtok számát adja meg, amely 0 és 1500 közötti érték lehet. Ha 46 bájtnál kisebb ez a szám, a keret kiegészül töltelékbitekkel, azért, hogy a 64 bájtos minimális keretméret meglegyen. Ismervén az adatmező hosszát, a vevő a vételnél leválasztja ezeket.

Az ellenőrző összeg vagy FCS (Frame Check Sequence) tulajdonképpen egy CRC (Cyclic Redundancy Check) algoritmus szerint kapott osztási művelet 32 bites maradéka, amelyet a keret végéhez fűz hozzá. A vételnél a teljes keretet a CRC-vel együtt, ha elosztják ugyanazzal az értékkel (a generátor polinommal) akkor, ha hibátlan a keret maradék nélküli értéket kapunk. Akár már egyetlen egy bit megváltozását is érzékeli, de 99,9%-ban alkalmas csoportos bithibák felderítésére is.

A keret tartalmaz még az adatmezőbe beágyazva három bájtnyi LLC (Logical Link Control) alréteg információt. Ebből kettő, a DSAP (Destination Service Access Point) és az SSAP (Source Service Access Point) a címzett illetve a forrás protokoll azonosítója, a CONTROL blokk pedig az LLC szolgáltatás módját mutatja [10].

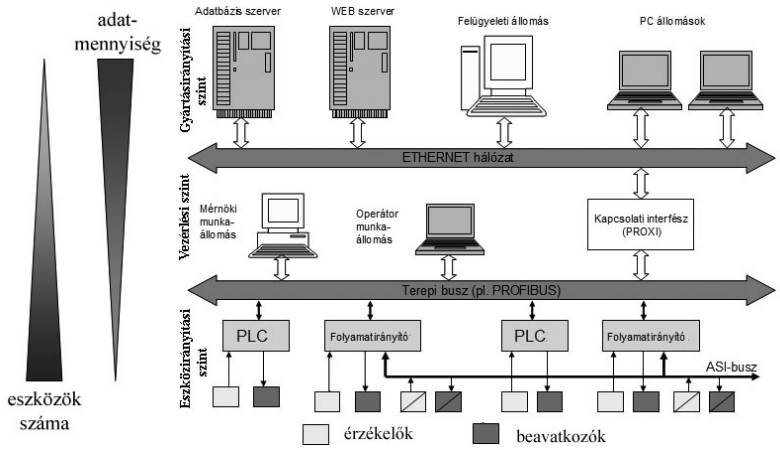
Ezek után pedig újra azzal szembesülünk, hogy van egy szabvány szerinti keretünk, amit gyakorlatilag senki sem használ. Még a Windows alapú operációs rendszerek is az eredeti Ethernet II elnevezésű, természetesen egyszerűbb keretformátumot használják alapértelmezésben. Ebből következik, hogy az ipari Ethernet rendszerek is az utóbbit használják. Az Ethernet II keret formátuma annyiban tér el a 802.3 formátumtól, hogy nem az LLC mezők azonosítják a protokollt, hanem az adathossz mező helyett, az ugyancsak 2 bájtos Ethertype mezőt használják.

Az előzőekben bemutatott, igencsak nagyfokú bizonytalanságot tükröző közeg-hozzáférési algoritmus ismeretében jogosan feltehetnénk magunknak a kérdést, egyáltalán hol használnak ilyen kiszámíthatatlannak tűnő kommunikációs kapcsolatot? A válasz nagyon egyszerű: Mindenütt, kivéve ahol nincsenek időhöz kötött, determinisztikus

folyamatok, ott ahol egy esetleges ütközés után kialakult véletlenszerű várakozási idő a valós idejű kapcsolatokat kedvezőtlenül befolyásolná. Az Ethernet hálózat alkalmazása viszont számos előnnyel rendelkezik. Világméretű kiterjedése, az Internet közvetlen elérhetősége, a hardvereszközök és szoftverek viszonylag alacsony ára, a nagy átviteli sebesség (100 Mb/s, 1 Gb/s, stb.) mind olyan tényezők, amelyek az ipari automatizálás fejlesztői számára célul tűzték ki azt a feladatot, hogy hogyan lehetne az Ethernet kommunikációt determinisztikussá tenni?

1.5. Az ipari Ethernet kialakulásának körülményei

Az ipari irányítóberendezések közötti kommunikációs kapcsolat alapfeltétele egy korszerű, biztonságos decentralizált folyamatirányító rendszer kialakításának. A technológia közvetlen közelébe telepített ipari vezérlők egy-egy részfolyamat közvetlen irányítását látják el. Az innen érkező adatokat továbbítani kell a rendszer adatbázisába, hogy feldolgozható, kiértékelhető esetleg megjeleníthető vagy archiválható legyen. A 70-es évektől kezdődően a vezérlési feladatokat egyre inkább a programozható logikai vezérlők (PLC) látják el. A mikroprocesszorok szinte minden területet átfogó elterjedése, az analóg jeltovábbítást felváltó, egyre több információt hordozó digitális jelközlés lehetővé tette az új irányítási megoldások használatát. Az intelligens távadók és beavatkozók megjelenésével az adatfeldolgozás osztott jellege kiszélesedik. Ezek a készülékek csak villamos segédenergiával működnek, digitális jelekkel dolgoznak és alkalmasak a hozzájuk tartozó irányítórendszerrel akár kétirányú kommunikációra. Az erőforrások megosztása miatt a kommunikációt biztosító buszrendszerek szerepe egyre jobban felértékelődik. Az egykor háromszintű folyamatirányítási hierarchia (1.6. ábra) a fizikai hálózat szempontjából egyre inkább egy, legfeljebb két szintre korlátozódik. A közöttük lévő különféle kommunikációs kapcsolatok pedig az ipari Ethernet irányába fejlődnek.



1.6. ábra. Klasszikus háromszintű folyamatirányítás

Az egyre intelligensebb eszközvezérlők (iDevice) kezdenek egyenrangúakká válni az őket irányító PLC-kkel. Sok esetben képesek önálló kommunikációs kapcsolatot is teremteni egymással. Az egyre fejlettebb rendszerek, egyre nagyobb adatmennyiséget kell továbbítsanak. A klasszikus terepi buszok az alacsony adatátviteli sebességük mellett már nem képesek kiszolgálni ezeket a fejlett rendszereket.

Akárcsak a PLC-k, a hozzájuk tartozó terepi buszrendszerek is gyártó-specifikusan alakultak ki. Ez megnehezítette bizonyos terepi eszközök cseresabatosságát. A rendszerek bővítése vagy esetleges módosítása csak ugyanattól a gyártótól beszerezhető eszközökkel volt lehetséges. Mindezek nagyban hozzájárultak ahhoz, hogy egy egységes

kommunikációs rendszert használjanak az ipari automatizálás területén is. Ez pedig nem más, mint az ipari Ethernet, amelynek alkalmazása számos előnyt jelent a hagyományos terepi buszokkal szemben:

- a tömeggyártás miatt olcsó eszközök használata,
- a különböző gyártmányú eszközök könnyen integrálhatók a rendszerbe,
- rendszerint többféle hálózati topológiát támogatnak (van, amelyik mindegyiket),
- azonos vagy kompatibilis protokollok használhatók az adatfeldolgozási és a technológiai szinten egyaránt, ezáltal a termelési adatok irodai szinten közvetlenül elérhetők,
- nagy átviteli sebesség és nagyobb sávszélesség érhető el, amely lehetővé tesz olyan technikák alkalmazását, amelyek a képinformáció továbbításán alapulnak, mint például a gépi látás,
- könnyű integrálni az Internettel, és akár minden szinten kihasználhatók a web alapú programozás előnyei,
- egységes és ezáltal olcsóbb üzemeltetés és karbantartás,
- vezeték nélküli kapcsolatok alkalmazása olyan helyeken, ahol másfajta kommunikáció nem, vagy csak nagyon nehezen valósítható meg,
- távolról történő beavatkozási, esetleg konfigurálási lehetőség meghibásodás esetén.

Igazi technikai kihívásnak számított még a 90-es évek elején, amikor egy távoli, akár több ezer kilométerre lévő PLC-s irányító rendszer meghibásodásakor a fejlesztők diagnosztizálás céljából modemes kapcsolaton keresztül tudták elérni. Ez ma már az ipari Ethernet hálózatban, megfelelő jogosultságok birtokában bármikor megtehető.

Mindezen előnyök mellett természetesen feltehetjük a kérdést: Vajon jó dolog-e kitennünk a teljes gyártási folyamatot az Internetre? Természetesen ezt csak akkor tehetjük meg, ha biztosítani tudjuk a rendszer hatékony védelmét a támadások ellen. Az Ethernet hálózatok biztonságának kérdései mára már külön ágazattá fejlődtek. Számos olyan fejlett tűzfal-kialakítási, beléptetési (azonosítási és jogosultsági) és titkosítási eljárás létezik, amelyek segítségével megakadályozható az illetéktelen behatolás és szinte teljes biztonságban tudhatjuk rendszereinket.

1.6. Az irodai Ethernet determinisztikussá tételének néhány elvi megoldása

Ahhoz, hogy determinisztikussá tegyük az Ethernet hálózatot, több szempontot is figyelembe kell vegyünk. Elsőként azt célszerű elérni, hogy valamilyen módon megkülönböztessük a valós idejű csomagokat a nem valós idejűektől. Ezen belül a determinisztikus csomagokat besorolni valamilyen prioritási szintbe, attól függően, hogy milyen valósidejű osztályt képviselnek. Számos elméleti és gyakorlat megoldás is született az idők során, amelyhez nagyban hozzájárult az utóbbi idők Ethernetes kommunikációnak nagyarányú műszaki fejlődése. Az alkalmazott megoldások két nagy csoportra oszthatók.

1. Az első csoportba azokat az eljárásokat sorolnám, amelyek az *ütközési valószínűség csökkentését* tűzték ki célul, vagyis lecsökkenteni az ütközési tartományt, korlátozni a backoff algoritmust, illetve prioritással ellátni a valós idejű csomagokat. Sokat segített a kapcsolt Ethernet kifejlesztése, azaz a hubok helyett a switchek használata, melynek következtében az ütközési tartomány mindössze két eszközre korlátozódik, ugyanakkor puffereiben a nem determinisztikus csomagok tárolhatók, amíg a valós idejűek továbbítása be nem fejeződik
2. A második csoportba azokat a megvalósításokat helyezem, amelyek valamilyen jól meghatározott algoritmus szerint vezérlik a buszhasználatot. Ilyenek például a

ciklikus hozzáférési technikákat alkalmazó master-slave, vagy provider-consumer eljárások, vagy a token passing, illetve a token ring módszerek. Ugyancsak ide sorolnám az időosztásos megoldásokat is, amelyeket főleg a szigorúan valós idejű, szinkronizált kommunikációs kapcsolatoknál használnak. Itt is találunk szoftveres, illetve ASIC-t (Application Specific Integrated Circuit) használó hardveres megoldásokat.

1.6.1. Az ütközési valószínűséget csökkentő eljárások

Itt említeném meg az ORTE-t (Ocera Real-Time Ethernet), amely forgalommenedzselő alkalmazással igyekszik ütközésmentessé tenni a kommunikációt, vagy a forgalomfinomítási (Traffic Smooting) eljárást, amely a lyukas vödör elvét használja [14, 15].

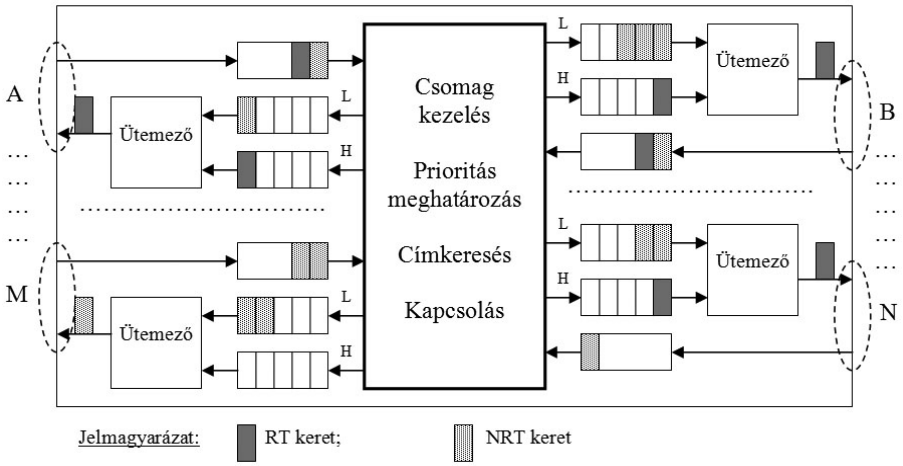
Lo Bello és társai [16] egy olyan megoldást fejlesztettek ki a dinamikus megközelítés kapcsán, mely a hálózati forgalmat az ütközések számával és az áteresztéssel jellemzi egy-egy adott intervallumon belül. A paramétereket ezután egy fuzzy szabályozóra küldik a hálózati bemeneti korlát beállítása végett. A módszerrel nagyon nagy hatékonyságot sikerült elérni a forgalomfinomítás terén.

Más megoldások a versengési backoff algoritmus befolyásolását helyezik előtérbe. Egy átlapoló mechanizmussal egészíti ki a CSMA/CD protokollt, még pedig olyan formában, hogy ütközéskor a valós idejű keretek nem egy 32 bájtos JAM-mel jelzik ezt, hanem egy hosszabb és a várakozási idővel arányos hosszúságú, de felső korláttal rendelkező úgynevezett *fekete burst*-öt (black burst) küldenek. A módszer kidolgozása J. L. Sobrinho és A. S. Krishnakumar nevéhez fűződik [17], akik igazolták, hogy ez determinisztikus működést tesz lehetővé osztott Ethernet hálózatokon. A módszer alkalmazásával megvalósítható, hogy ugyanazon a hálózaton a valós idejű keretek előnyt élvezzenek a nem valós idejűekkel szemben, sőt egymáshoz viszonyítva is attól függően, hogy melyik érkezett hamarabb. Azaz FCFS (First Come First Served) elsőnek érkezett elsőnek kiszolgált hozzáférést biztosít

1.6.2. Kapcsolt Ethernet

A kapcsolt Ethernet igénye akkor merült fel, amikor célul tűzték ki a globális áteresztőképesség javítását, a forgalom izolálását és az eredeti CSMA/CD választó mechanizmus nem-determinisztikus paraméterei okozta hatás csökkentését. Az Ethernet kapcsolók tulajdonképpen leszűkítik az ütközési tartományt az éppen aktuális kommunikációban résztvevő két állomásra, ha csak nem multicast vagy broadcast adás történik. Az állomások tehát nem csatlakoznak direkt módon egymáshoz. Képesek a kapcsolatot duplex módban bonyolítani, vagyis olyan mintha pont-pont kapcsolat épülne ki két állomás között. A legtöbb korszerű Ethernet kapcsoló a „tárol és továbbít” (store and forward) elvet használja, és ha ismeri az IEEE 802.1p, minőségi szolgáltatást (QoS) biztosító protokollt, akkor prioritási szintek szerint tudja pufferelni a kimeneti adatokat [18].

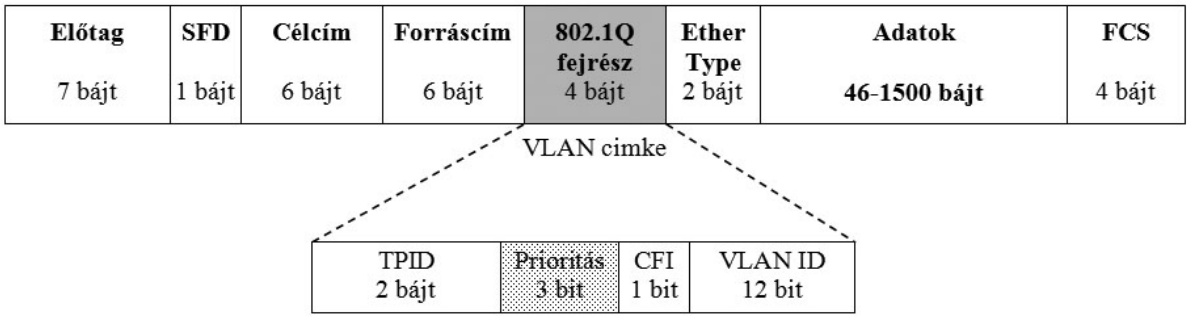
A legtöbb switch kettő, de a menedzselhető, főleg ipari Ethernet switchek akár négy prioritási szintet is tudnak kezelni. Amikor egy üzenet érkezik egy porthoz, akkor az letárolódik a bemeneti pufferbe, megállapításra kerül a prioritás és a cél, majd a prioritásnak megfelelő pufferébe továbbítódik, ahonnan az ütemező mindig a prioritási sorrendnek megfelelően helyezi a buszra. A következő ábrán (1.7. ábra) egy „tárol és továbbít”, két prioritási szinttel rendelkező, duplex kapcsoló elvi felépítési vázlatát láthatjuk.



1.7. ábra. A Store and Forward elvű switch felépítési vázlata

Amikor az egyes portokon az üzenetek érkezési aránya (legyen bemeneti vagy kimeneti port) nagyobb, mint az elküldési arány, az üzenetek sorokban várakoznak. A legtöbb korszerű switch elég gyors az érkező üzenetek lekezeléséhez, így a bemeneti portokon nem alakul ki sor. A sorok a kimeneti portokon jelenhetnek meg, amikor rövid időn belül, egyszerre több port felől érkeznek üzenetek, ugyanarra a kimeneti portra. Ez esetben a sorban lévő üzenetek szekvenciálisan kerülnek továbbításra. Ezért a valós idejű ipari Ethernet hálózatban nélkülözhetetlen a párhuzamos, különböző prioritású szintekhez tartozó kimeneti, FCFS rendszerű pufferek használata.

Most már csak az a kérdés, hogy honnan ismeri fel a kapcsoló a valós idejű csomagokat? Az IEEE 802.1Q, virtuális LAN-okra vonatkozó szabványleírás szerinti adatkeretekben (1.8. ábra), az IEEE 802.1P-nek megfelelően, nyolc különböző prioritási szintet lehet definiálni.



1.8. ábra. A 802.1Q szabvány szerinti VLAN címkés keretformátum

Az előző ábrán (1.8.) láthatjuk, hogy a 802.1Q keret tulajdonképpen Ethernet II keret, amely kiegészül egy 2x2 bájtos VLAN címkével, amelyben 3 bit áll rendelkezésünkre a prioritás meghatározására. A normál TCP/IP adatkeretek a legalacsonyabb 0 szintet kapják, míg a valós idejű keretek 5, 6 vagy a legmagasabb 7 szintű prioritást kapják. A VLAN címke további mezőire a későbbi fejezetekben még visszatérek.

Első ránézésre azt gondolhatnánk, hogy a switch-ek alkalmazásával meg is oldottuk a valós idejű adattovábbítást az Ethernet hálózatban. Ez ugyan sokat javít a helyzeten, de korántsem elegendő. Az automatizált ipari berendezések irányítása a legtöbb esetben úgy valósul meg, hogy egy (vagy több) vezérlő nagyon sok, akár egymástól távol eső érzékelőktől kap adatokat, amelyek kiértékelése után ugyancsak ez a vezérlő továbbítja a

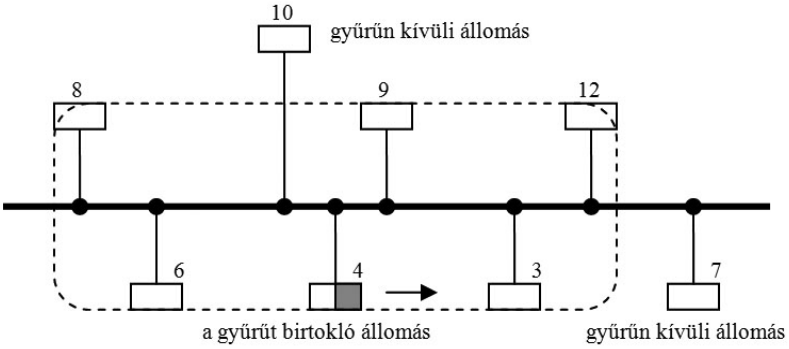
beavatkozáshoz szükséges információt is. Az ilyen Termelő-fogyasztó (Provider-Consumer) modellen alapuló kommunikáció esetében bizony előfordulhatnának olyan helyzetek, hogy egyszerre, egy időben több érzékelő eszköz Ethernetes illesztője elárasztja a vezérlő switchen belüli kimeneti portját. Vagyis szükséges lenne még valami, ami ütemezi ezt a feladatot.

Azt sem szabad elfelejtenünk, hogy a hálózati kapcsolók a csillag topológiát preferálják, az ipari berendezések és vezérlők pedig a legtöbbször busz, vagy gyűrű topológiát használnak. Mindezek mellett még bizonyos késleltetéssel is számolnunk kell, amely akár jelentős is lehet, főleg a szigorúan valós idejű rendszereknél.

Mindezek ellenére a hálózati kapcsolók alkalmazása nélkülözhetetlen a valós idejű ipari Ethernet világában, olyannyira, hogy nagyon sok ipari Ethernetes eszközt saját belső kapcsolóval látnak el, hogy bármilyen helyzetben alkalmas legyen további kommunikációs kapcsolat kialakítására.

1.6.3. Vezérjeles sín

Már a 802.3-as szabvány megjelenésének idején egyes gyártásautomatizálással foglalkozó szakemberek komoly fenntartással fogadták ezt a már ismert nem determinisztikus tulajdonságai miatt. Ezért egy egyszerű, kiszámítható legrosszabb esettel rendelkező rendszer kidolgozásába fogtak. Az elképzelés szerint az állomások gyűrűt alkotnak és egy vezérjel, a token körbejár a csomópontok között egy jól meghatározott sorrend alapján. Ezek szerint, ha n állomás vesz részt a gyűrűben, és Δt ideig (token tartózkodási idő) tart egy keret elküldése, akkor egyetlen egy állomásnak sem kell $n \cdot \Delta t$ időnél többet várakoznia egy keret elküldésére. A gyűrű topológia viszont fizikailag nem nagyon illeszkedik a gyártásautomatizálás egyenes vonalú pályájához. Ezért az ide vonatkozó 802.4-es szabványt, amelyet 1986-ban Dirvin és Miller indított útjára, úgy határozták meg, hogy az fizikailag egy lineáris, vagy fa topológiájú hálózat, amelyben az állomások vannak logikai gyűrűbe szervezve (1.9. ábra). A vezérjel a logikai gyűrű mentén körbe jár. Küldési joga csak a vezérjelet birtokló állomásnak van, ezért az ütközés kizárható.



1.9. ábra. A vezérjeles sín elve

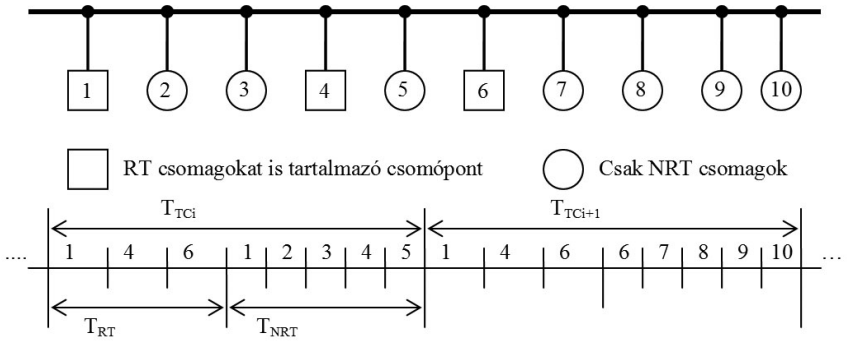
A gyűrűben résztvevő állomások fizikai sorrendje lényegtelen mivel adatszórásos közegben vannak. A lényeg az, hogy a tokent birtokló állomás az utána logikai sorrendben következő állomásnak a címét ismerje. Induláskor mindig a legmagasabb címmel rendelkező állomás kapja meg a vezérjelet, majd ezt követi csökkenő sorrendben egy alacsonyabb címmel rendelkező állomás. (12-9-8-6-4-3 a sorrend, a 10-es és a 7-es állomások a gyűrűn kívül vannak.) Azt, hogy melyik állomás vehet részt a gyűrűben a

MAC protokoll határozza meg, sőt azt is, hogy engedélyt kap-e a kérelmező állomás a belépésre vagy a kilépésre.

A vezérjeles adattovábbítás protokollja négy prioritási szintet is enged definiálni: a 0-ást, a 2-est, a 4-est és a 6-ost. Ez virtuálisan olyan mintha minden állomás belül négy állomást tartalmazna, amelyek az adott prioritási osztályokhoz tartoznak, amelyek közül a legmagasabb prioritású, a 6-os van legfelül. A MAC alrétegbe érkező bemeneti csomagok prioritás szerint kerülnek a megfelelő szintre. Az állomáshoz érkező vezérjel először mindig a 6-os prioritási szintről veszi fel a keretet, ha van, majd a halad beljebb a 0-s szintig, amikor pedig letelt az állomáshoz rendelt idő, továbblép a következőhöz.

A token adott állomáshoz köthető tartózkodási idejét, a sávszélességen belül eloszthatjuk egyenlő arányban (szimmetrikusan) az állomások között, de aszimmetrikusan is, lehetővé téve ezáltal, hogy ugyanahhoz a csomóponthoz a token akár többször is eljusson egy körben járási ciklus alatt.

Az imént bemutatott vezérjeles továbbítási taktikát alkalmazzák a RETHER és a RTEP protokollok.



1.10. ábra. A RETHER protokoll kerettovábbítási stratégiája.

A RETHER esetében a token körbejárás periodikus. CSMA/CD üzemmódban van mindaddig, míg nem érkezik RT kommunikációs kérés, ekkor átvált token üzemmódba és végig megy mindazokon a csomópontokon, amelyek RT üzenetet akarnak továbbítani, majd ha marad elég ideje a cikluson belül, akkor felvesz még NRT kereteket is (1.10. ábra). Ha már nincs több RT kérés, akkor visszakapcsol CSMA/CD módba [19, 20].

Az RTEP estében a közeghez való hozzáférés két fázisban történik: kiválasztás és üzenettovábbítás. A token először az összes csomóponton körbejár, hogy megkeresse a legnagyobb prioritású üzenetet, majd ezt követően közvetlenül ahhoz a csomóponthoz megy, amely a legnagyobb prioritású üzenetet tartalmazza. A valós idejű csomagok az Ethernet keret adatmezejében kerülnek továbbításra. Az átvitel után a csomópont egy új kiválasztási fázist indít. Ha a kiválasztási keretbe beírt prioritásnál nagyobbat talál, akkor az felülírja a keretben lévőt. A protokoll tartalmaz hibatolerancia mechanizmust is, amely segíti a sérült keretek helyreállítását vagy újraküldését [21].

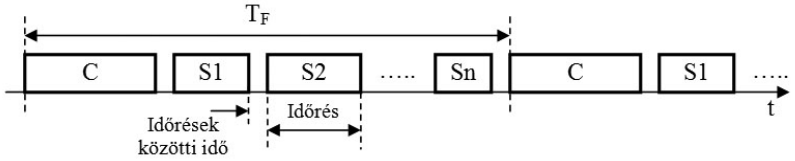
1.6.4. Időosztásos többszörös hozzáférés (Time-Division Multiple Access, TDMA)

Egy másik jól bevált technika az osztott kommunikációs hálózaton, ha ciklikus ütemezés mellett, kizárólagos időréseket (slot time) rendelünk a különböző adatforrásokhoz. Ez a jól ismert időosztásos többszörös hozzáférés (TDMA), mely tartalmaz egy globális szinkron keretet azért, hogy az összes csomópont egyezségre jusson az adott átviteli időréssel kapcsoltba. Ez a megoldás egy ütközésmentes közeg-hozzáférési protokollt eredményez, amelyet kezdetben a nagysebességű autóipari alkalmazások protokolljaként (TTP/C, TTCAN) [29, 30], majd a későbbiekben az osztott Ethernet

legmagasabb, C osztály szerinti szintjén is alkalmazzák, felülbírálva ezzel az eredeti CSMA/CD mechanizmust.

Az időrés lehet fix, vagy változó szélességű. A jobb sávszélesség kihasználása érdekében, természetesen a dinamikus, változó méretű időrések alkalmazása válik előnyössé. Minden egyes időrés alatt a csomópontokban a feladatok ütemezése megtörténik, megakadályozva ezzel a versengést a buszhozzáféréshez. A konfigurálás során lehetőség van újabb csomópontok hozzáadására vagy elvételére. Ennek alapján meghatározható a keretidő (T_F), amelyet ha fix értéken tartunk, akkor kiszámítható, pontos valós idejű működést eredményez.

Minden egyes keret egy kontroll (C) időréssel kezdődik, amely főleg szinkronizálási információkat tartalmaz, majd ezt követik a csomópontokhoz rendelt S_i időrések, amelyek között egy-egy rövid, szabad időköz van hagyva (1.11. ábra).

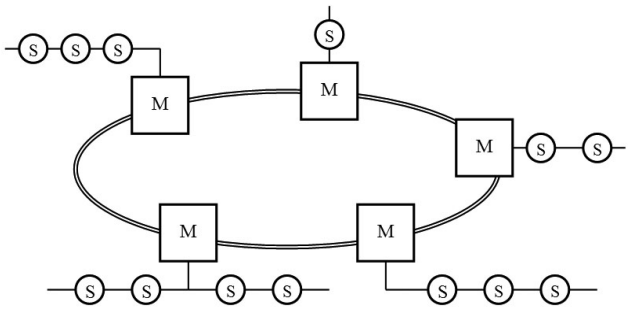


1.11. ábra. TDMA kertstruktúra

A módszer alkalmazásával lehetőség nyílik a nagysebességű kommunikációs kapcsolatok szinkronizálására, kiszámítható algoritmusok bevezetésére, valamint valós idejű és nem valós idejű információk azonos időben történő ütközésmentes továbbítására.

1.6.5. Master-Slave alapú technikák

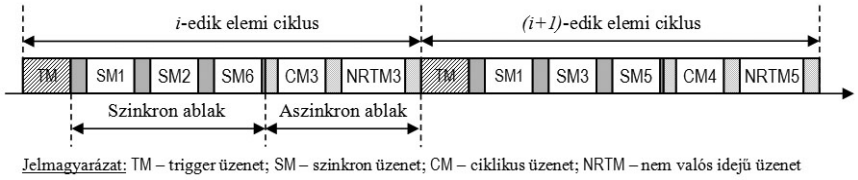
Az egyik legegyszerűbb módja a valós idejű kommunikáció megvalósításának a master-slave megoldás. A közeghozzáférést egy kitüntetett szereppel ellátott csomópont, a master vezérli. Az összes többi, hozzárendelt csomópontok a slavek. Ezek csak a master által kezdeményezett kapcsolatok során kommunikálhatnak. A master-slave kapcsolat egyfajta hálózati hierarchia szintet határoz meg. A masterek a hierarchia valamelyik magasabb szintjén egyenrangú partnerekként szerepelnek. Közöttük legtöbbször gyűrű topológia szerinti tokenes kommunikáció épül fel. Ezek alá vannak rendelve a slave egységek. Ezek lehetnek ugyanarra a buszra kötve, mint a masterek, de jellemzően inkább a felső szintűtől független kommunikációs vonal köti a masterhez (1.12. ábra). A tokent birtokló master üzeneteket küldhet a slaveknek vagy fogadhat ezektől.



1.12. ábra. Master-Slave kommunikáció hierarchiája

Az előbbi eljárás egyik módosított változata az FTT Ethernet protokoll, amely ipari Ethernet hálózatra lett kifejlesztve. Tulajdonképpen egy módosított master-slave technikát használ, amely lecsökkenti a protokoll kommunikációs túlterheltségét. A buszhozzáférési

időt fix hosszúságú, úgynevezett elemi ciklusokra osztják fel. Ezek további két részre tagolódnak: a szinkron és az aszinkron ablakra (1.13. ábra), melyek különböző karakterisztikával rendelkeznek [31].



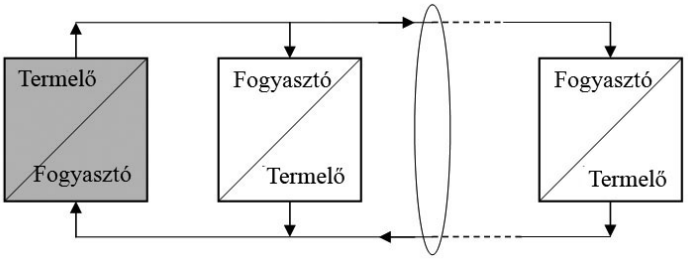
1.13. ábra. Az elemi ciklusok szervezése

A szinkron ablak a master által ütemezett periodikus, idővezérelt forgalmat szállít. Az idővezérelt kifejezés azt jelenti, hogy a forgalom egy közös időalaphoz szinkronizált, melyet ez esetben a master szolgáltat. Az aszinkron ablak ciklikus, eseményvezérelt üzenetekhez kötődő információkat, illetve nem valós idejű forgalmat továbbítja. A két fázis között egy elég jelentős időbeli elkülönülés van jelen, hogy az eseményvezérelt forgalom ne ütközzön az idővezérelt forgalommal.

1.6.6. Termelő-fogyasztó (Provider-Consumer) modell

Az ipari Ethernet rendszerek talán legfejlettebb valós idejű kommunikációs megoldásának tekinthetjük. Működési elve hasonlít a master-slave kommunikációhoz, azzal a különbséggel, hogy full duplex összeköttetést feltételez és itt mindkét fél, akár egy időben is kezdeményezhet kommunikációs kapcsolatot. Az termelő rendszerint adatokat gyűjt az érzékelők felől vagy saját adatbázisából veszi elő és továbbítja a fogyasztó felé. A fogyasztó a kapott adatokat feldolgozza és/vagy továbbítja beavatkozók felé. A kapcsolatot kezdeményezheti bármelyik fél. Ebből a szempontból a felek egyenrangúaknak tekinthetők. A felek bármikor szerepet cserélhetnek, attól függően, hogy mikor milyen feladatot kell ellássanak.

Konfiguráláskor az eszközök között úgynevezett kommunikációs relációk épülnek fel, amelyeket bármikor meglehet nyitni, amikor adatátvitelt akar kezdeményezni valamelyik fél, illetve le lehet zárni, amikor az adattovábbítás befejeződik. A leggyakrabban alkalmazott megoldás az egy termelő, több fogyasztó, illetve a több termelő egy fogyasztó struktúra (1.14. ábra). Ezt a megoldást alkalmazzák például a Profinet IO rendszereknél [37], ahol egy PLC-hez több IO-eszköz kapcsolódik. A PLC is és az eszközök is egyaránt lehetnek termelő illetve fogyasztó szerepben is.



1.14. ábra. Termelő-fogyasztó modell

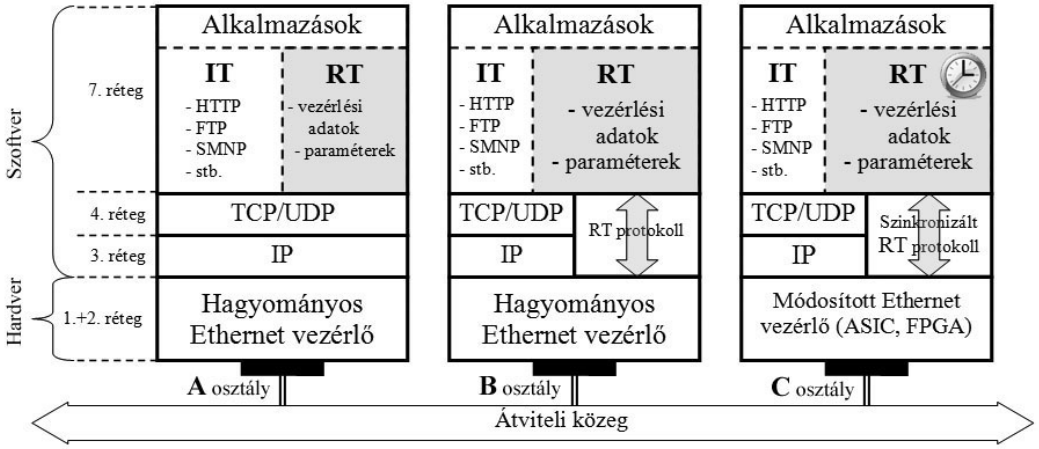
Az előző alfejezetekben bemutatott elvi megoldások egy része soha sem került ipari alkalmazásra, más része már elavult (például a vezérjeles gyűrű), de jelentős szerepet játszottak a jelenlegi, korszerű Ethernet alapú ipari kommunikációs rendszerek kifejlesztésében.

II. FEJEZET

AZ IPARI ETHERNET RENDSZEREK BEMUTATÁSA

A I. fejezetben (1.5) már utaltam arra, hogy az Ethernet hálózat széleskörű elterjedésének következtében az ipari kommunikációs rendszerek is egyre inkább az ipari Ethernet irányában kezdenek fejlődni. Az ipari irányító- és kommunikációs rendszerek gyártói számos ipari Ethernet technológiára épülő megoldást fejlesztettek ki, amelyek jelentős szerepet játszanak a mai, korszerű ipari elosztott vezérlési és folyamatirányítási alkalmazásokban. A hagyományos terepi buszos kommunikáció korlátozott adatátviteli sebessége gátat szab azon korszerű technológiai megoldások alkalmazásának, amelyek nagyobb információmennyiséget és átviteli sebességet igényelnek. A közös Ethernet platform amellett, hogy kétirányú, 100 Mb/s-os bitsebességet képes szolgáltatni, egy lépéssel közelebb hozta egymáshoz a különböző gyártók specifikus termékeit. Ezek most már ha nem is közvetlenül, de megfelelő eszközleírók segítségével implementálhatók a legtöbb rendszerbe. Ez a feladat persze nem olyan egyszerű, mert a gyártók meglehetősen kevés műszaki információt közölnek a termékeik ilyen jellegű alkalmazásaival kapcsolatosan.

A jelenleg elterjedt megoldásokat a gyakorlatban három osztályba sorolhatjuk, attól függően, hogy milyen kommunikációs rétegeket használnak és milyen valós idejű követelményeket képesek teljesíteni. A 2.1. ábrán ezen osztályok rétegstruktúráit láthatjuk.



2.1. ábra. Ipari Ethernet osztályok struktúrái.

Az *A osztályú* ipari Ethernet gyakorlatilag ugyanazt a hardver felépítést és TCP/IP vagy UDP/IP protokollkészletet használja, mint az irodai Ethernet. A valós idejű követelményeket az RT alkalmazások a TCP/IP rétegek felett valósítják meg. Az ütközések elkerülése a valós idejű alkalmazásokban szoftveres úton forgalomfinomítással, vagy kapcsolt Etherneten keresztül történik. Meglehetősen korlátozott valós idejű működést képes biztosítani, általában 50-100 ms-os kézbesítési idővel és legalább ekkora jitterrel. Alkalmazási területe: Főleg monitoring rendszerek, folyamatirányítás, épületautomatizálás, stb. Ebbe az osztályba sorolhatók például a hagyományos Ethernet/IP, a Modbus/TCP vagy a Profinet CBA. [36, 39, 47]

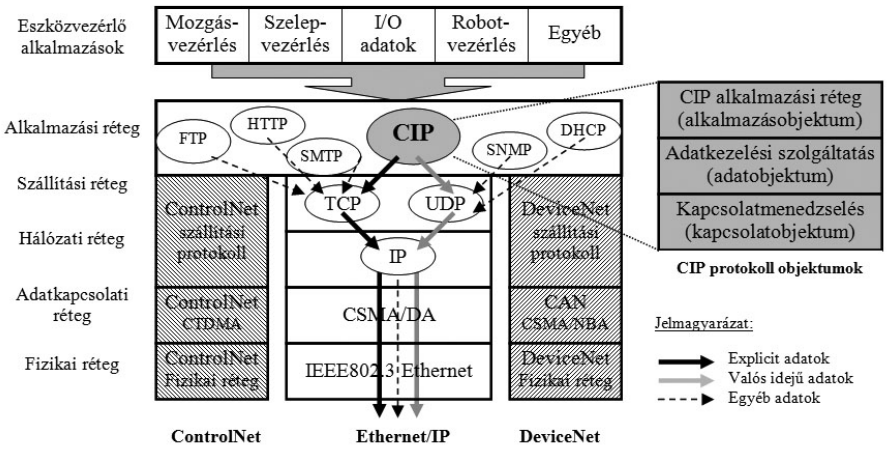
A *B osztály* ugyancsak a szabványos Ethernet hardveres felületet használja, de már jobb valós idejű támogatást biztosít, egyrészt mert a QoS szerint a prioritással ellátott RT csomagokat képes előnyben részesíteni a nem valós idejű csomagokkal szemben, másrészt az egyes RT protokollok képesek kezelni közvetlenül az alsóbb rétegek felől érkező kereteket. Ilyenkor azonban a kommunikáció csak egy szegmensen belül valósítható meg. Az ide tartozó rendszerek általában kielégítik a gyártásautomatizálás követelményeit: 5-10ms-os kézbesítési idők, 10-20 ms-os válaszidők, 2-4 ms jitter. Főleg a PLC-s vezérlésű DCS alapú ipari automatizálás terén alkalmazzák. Az osztály egyik meghatározó megoldása a Siemens támogatottságú Profinet IO vagy a Bernecker & Rainer cég által fejlesztett Ethernet Powerlink.

A harmadik, *C osztályba* a legigényesebb valós idejű működést igénylő hajtásszabályozás és mozgásvezérlés tartozik, amely rendszerint szinkronizált, ciklikus működést feltételez. A ciklusidő 1ms, vagy ennek tört részei (0,5 ms, 0,25 ms), a jitter pedig a legtöbb esetben kisebb, mint 1µs. Mindenképpen 100Mb/s-os Full duplex Ethernet hálózatot feltételez. A hardverfelület módosított (nem szabványos Ethernet), speciális ASIC-t vagy FPGA-t tartalmaz, amelyben a kapcsoló a TDMA technikát alkalmazva mindig szabad utat biztosít a szigorúan valós idejű kereteknek, ugyanakkor a fennmaradó időben a nem valós idejű forgalom is biztosított. A szinkronizált működést az adatkapcsolati rétegbe implementált IEEE 1588 PTP (Precision Time Protocol) protokoll [35] biztosítja. Az eszközöket a MAC címek alapján azonosítják, routolás itt sem valósítható meg. A csoport meghatározó képviselője a Backhoff cég által fejlesztett EtherCAT, de jelentős részt foglal el a Profinet IRT és a Sercos III is.

A fejezet további részében mindegyik osztályból néhány gyakrabban alkalmazott, elterjedtebb megoldást fogok bemutatni.

2.1. Az Ethernet/IP

Az Ethernet/IP protokoll eredetileg a Rockwell cég által került bevezetésre 2001-ben, mint az akkori idők egyik legkorszerűbb és legjobb valós idejű adottságokkal rendelkező ipari Ethernet rendszere. Az IEC 61158 szabvány valós idejű követelményeit, az IEC 61784-1 szabvány ipari Ethernetes specifikációjának megfelelően valósítja meg. Főleg a tengerentúlon, Észak-Amerikában valamint Ázsiában terjedt el. Jelenleg az ODVA (Open Devicenet Vendors Association) menedzseli, melynek köszönhetően a valós idejű működésének alapját képező CIP (Common Industrial Protocol) protokollja szerepel más ipari Ethernetes megoldásokba implementálva is, mint például a ControlNet vagy a DeviceNet és újabban a CompoNet. (2.2. ábra) [35].



2.2. ábra. A CIP protokollon alapuló Ethernet/IP rétegstruktúrája

2.1.1. A CIP protokoll

Valós idejű működés elsősorban az alkalmazási rétegbe implementált CIP protokoll objektumorientált funkcióinak köszönhető. Ezek, funkcióikat tekintve, három szintre bonthatók a következők szerint:

1. Üzenetirányítás és kapcsolatmenedzselés (kapcsolatobjektum),
2. Adatkezelési szolgáltatás (adatobjektum),
3. Objektum orientált alkalmazáskezelés (alkalmazásobjektum).

A CIP protokoll beágyazása az alkalmazási rétegbe (2.2. ábra) nem befolyásolja a többi, már jól ismert protokoll működését, megőrizvén ezáltal az Ethernet hálózat nyitottságát.

Az eszközvezérlő alkalmazások üzeneteit az üzenetirányító objektum a létrejövő kapcsolat rendeltetése szerint két irányba továbbítja. Ennek megfelelően a CIP protokollt használó eszközök alapvetően kétféle üzenetküldési módot használnak:

- *Nem kapcsolt kommunikáció (Unconnected Messaging)* – alacsony prioritású, nem rendszeresen előforduló kapcsolat-felvételi procedúrák kezelésére. Pl. konfigurációs beállítások során keletkező adatforgalom, vagy lekérdezések alkalmával létrejövő kapcsolatok. Ezek az üzenetek a TCP/IP csomagok adatmezőjébe ágyazódnak be és úgy kerülnek továbbításra, mint bármilyen szabványos Ethernet üzenet.
- *Kapcsolt kommunikáció (Connected Messaging)* – elsősorban valós idejű, rendszeresen (ciklikusan) továbbítandó vezérlési adatok továbbítására. A kommunikációt a Kapcsolatmenedzser irányítja, amely miután létre jött egy kapcsolat a csomópontok között, képes szétválasztani az explicit és implicit üzeneteket. Ezeket aztán az Üzenetirányító a megfelelő típusú TCP illetve UDP szállítórétegek felé irányítja.

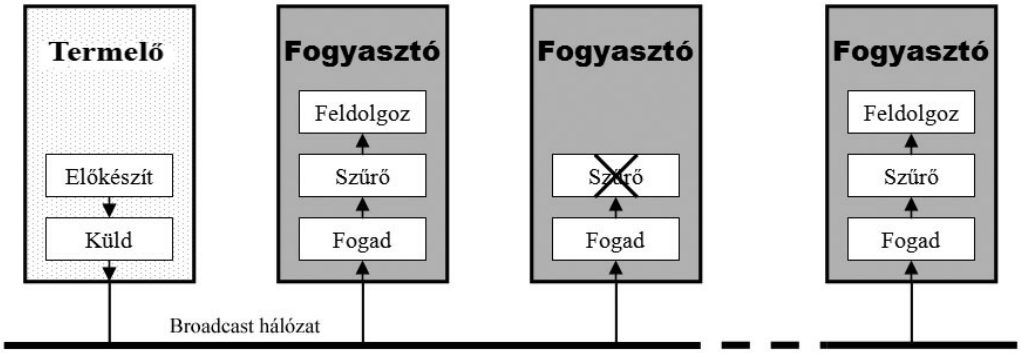
Az imént említett üzenet szétválasztásnak köszönhetően az összes Ethernet/IP hálózati kommunikáció e két üzenettípus szerinti kapcsolaton alapul:

- *Explicit üzenet kapcsolat (Explicit Messaging Connection)* – általános célú pont-pont kapcsolat két eszköz között. Ezek a kapcsolatok kivétel nélkül az Üzenetirányítón (Message Router) keresztül a TCP/IP szolgáltatás adta lehetőségeket használják az adatcsere megvalósítására. Legtöbbször „kérés-válasz" jellegűek ezek a kapcsolatok. Az explicit kérést a vevő dekódolja, értelmezi, majd ugyancsak explicit választ generál.
- *Implicit I/O adat kapcsolat (Implicit I/O data Connection)* – akkor jön létre, amikor valós idejű, alkalmazás specifikus I/O adatokat kell továbbítani az erre a célra létrehozott kapcsolat során. Ezek az adatok szabályos időközönkénti, ciklikus jellegű üzenetek. A legtöbb esetben termelő-fogyasztó modell szerinti pont-multipont kétirányú kapcsolat épül fel a felek között. Az ilyen típusú üzenetek számára mindig az UDP/IP típusú hálózati protokoll szerinti kapcsolat épül ki és fennmarad mindaddig, míg a kezdeményező el nem bontja.

2.1.2. Az Ethernet/IP kapcsolat modellje

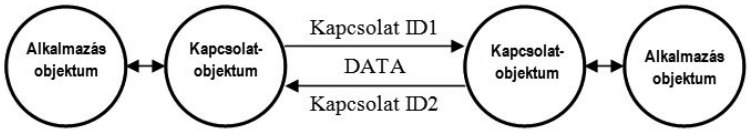
Az Ethernet/IP az eszközök közötti kommunikáció számára a sok szempontból előnyös termelő-fogyasztó modellt alkalmazza. Az esetek többségében ez úgy valósul meg, hogy egy termelő több fogyasztót szolgál ki. Hátránya ennek a megoldásnak, hogy broadcast kommunikációt feltételez, így a fogyasztónak ki kell szűrnie a számára szükségtelen üzenetet. (2.3. ábra). Ha megváltoztatjuk a kommunikációs kapcsolatokat leíró attribútumokat, akkor olyan kapcsolat is létre hozható, amikor több termelő szolgál ki

egy fogyasztót. Full duplex hálózatban, a két kapcsolat akár egyszerre, egy időben is megvalósítható.



2.3. ábra. Termelő-fogyasztó modell szerinti Ethernet/IP kapcsolat

A kommunikációs kapcsolat kiépítéséhez, a kapcsolatban résztvevő állomások két-két objektumának együttműködésére van szükség: az egyik az alkalmazásobjektum a másik pedig a kapcsolatobjektum (2.4. ábra). Ha az állomások egyenrangúak, az alkalmazás jellegétől függően a kapcsolatfelvételt bármelyik fél kezdeményezheti. Az alkalmazásobjektumban az eszközök azonosítására szolgáló gyártó-specifikus adatok szerepelnek, mint például a gyártó ID, az eszköz típusa, sorozatszáma, neve, stb. Ha a kapcsolatfelvétel során megszólított állomás úgy látja, hogy képes az illető eszközzel kommunikálni, akkor elfogadja a kapcsolatot. [34]



2.4. ábra. Kapcsolatfelvétel két eszköz között

A kapcsolatobjektum feladata meghatározni, hogy az adott alkalmazás szerint milyen típusú kapcsolatra van szükség, mert ennek megfelelően kell kiépíteni a kapcsolatot. Mindegyik kapcsolathoz hozzárendel egy jól meghatározott kapcsolatazonosítót (Connection ID). Ha kétirányú kapcsolatra van szükség, akkor kapcsolatazonosítóból is kettőt kell létrehozni. Amikor egy kapcsolat létrejött, akkor a kommunikációhoz szükséges összes erőforrás, beleértve a CIP üzenetirányítót is, lefoglalásra kerül, így stabil összeköttetés biztosítható.

Az Ethernet/IP hálózat eszközeit, rendeletetésük és betöltött funkcióik szerint három osztályba csoportosíthatjuk: üzenet- (Messaging Class), adapter- (Adapter Class) és szkenner osztály (Scanner Class).

1. Az üzenet osztályba tartozó eszközök kizárólag csak explicit üzenetforgalmat (kapcsolt vagy nem kapcsolt kommunikáció szerint) bonyolítanak, valós idejű I/O adatforgalmat nem. Ide sorolhatók például a programozó eszközök, amelyek a PLC-k és HMI eszközök vezérlési programjait töltik fel vagy le.
2. Az adapterek tulajdonképpen távoli I/O eszközvezérlők, amelyek válaszolnak, vagy végrehajtják a szkenner kéréseit. Nem kezdeményeznek adatkapcsolatot, csak akkor küldik vagy fogadják a valós idejű I/O adatokat, amikor a szkenner erre felkéri őket. Explicit üzeneteket is fogadhatnak, más osztálybeli eszközöktől, például a programozó eszköztől, de ilyen jellegű kapcsolatot sem képesek kezdeményezni.

3. A szkenner osztályba tartoznak mindazok az eszközök, amelyek a kapcsolatot kezdeményezhetik az adapterekkel, vagy más szkennerekkel, és mindkét alapfunkciót betölthetik, vagyis termelő illetve fogyasztó szerepük is lehet. A legtöbb esetben erre a célra PLC-ket, robotvezérlőket, stb. használnak.

2.1.3. A CIP protokollt használó Ethernet/IP legfontosabb jellemzői

A Ethernet/IP rendszereknél a kerettovábbítási kapacitás (Packet Rate Capacity – $C_{P/S}$) az egyik legfontosabb meghatározó érték, amelynek segítségével jellemezni tudjuk a hálózat képességeit. Ennek segítségével tudjuk meghatározni a csomópontok számát (N), amelyet kezelni tud az illető rendszer. Ugyancsak ettől az értéktől függ az implicit I/O adattovábbítás buszfrissítési ideje, azaz ciklusideje. Ezt az egyébként nagyon fontos alaptényezőt az Ethernet/IP rendszernél csomagküldési intervallumnak (Requested Packet Interval – RPI) nevezik. Kétirányú I/O kapcsolatnál a ciklusidő számított, minimális átlagértékét a következő összefüggéssel határozhatjuk meg:

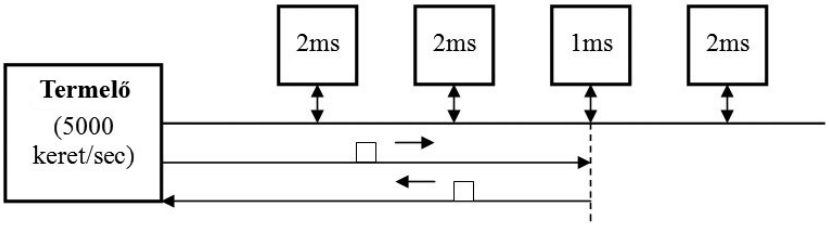
$$RPI = \frac{2 \cdot N}{C_{P/S}} \tag{2.1}$$

A következő táblázatban a ciklusidő értékeket láthatjuk, a csomópontok számától függően, különböző osztályú szkennerek esetében [34].

2.1. táblázat. Az RPI változása a csomópontok szerint

Csomópontok száma (N)	**RPI [ms]**		
	5000 keret/s alap szkenner	**10000 keret/s SRT szkenner**	**25000 keret/s HRT szkenner**
4	**1,6**	**0,8**	**0,32**
8	**3,2**	**1,6**	**0,64**
16	**6,4**	**3,2**	**1,28**
32	**12,8**	**6,4**	**2,56**
64	**25,6**	**12,8**	**5,12**

A valóságban a számított ciklusidő értékeknél kisebb ciklusidő is beállítható egyes csomópontokban akkor, ha a többi csomópontok eszközei nagyobb ciklusidőt igényelnek. Például a Rockwell ControlLogix sorozat 1756-ENBT 5000 keret/sec-os szkenner N = 4 csomóponttal konfigurálható úgy, hogy egy csomópont 1 ms-os, három csomópont pedig 2ms-os ciklusidővel rendelkezzen (2.5. ábra) [36].



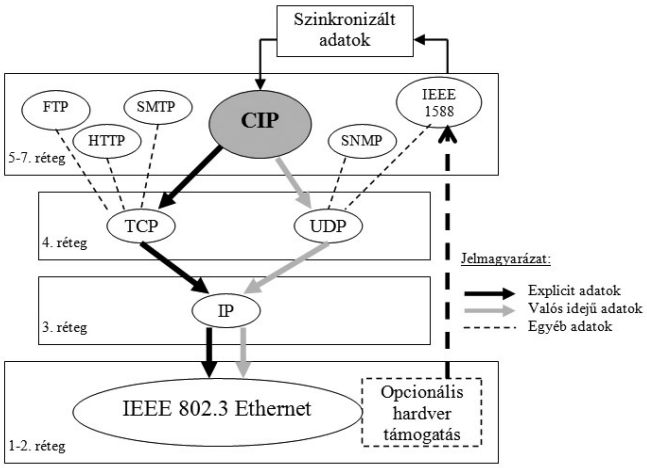
2.5. ábra. Különböző ciklusidővel rendelkező kapcsolatok

A három 2 ms-os ciklusidejű csomópont kiszolgálása az (2.1) szerint másodpercenként 3000 keretet igényel, így még marad egy 2000 keret/sec-os kapacitás, ami pontosan egy 1ms-os fogyasztót tud kiszolgálni.

Az explicit és implicit üzenetek kapcsolatainak száma is korlátozott. A legtöbb Rockwell Ethernet/IP eszköz maximum 64 eszközzel tud létesíteni TCP kapcsolatot. CIP kapcsolatot viszont ennél jóval többet, például a 1756-ENBT 128-at, a 1756-EN2T pedig 256-ot, azért mert egy TCP kapcsolat alatt több CIP kapcsolat is létrehozható [36].

2.1.4. A hatékonyság növelése PTP protokollal

Az Ethernet/IP rendszer valós idejű teljesítményének javítása érdekében a Rockwell bevezette a CIP Sync implementációt, amely szinkronizált adatátvitelt biztosít az eszközök között. Lényege, hogy az IEEE 1588 PTP (Precision Time Protocol) protokollt alkalmazva az eszközök órajelei szinkronizálva lesznek egy meghatározott master órához képest, amely akár az termelő órája is lehet [35]. A ciklusidőt ugyan nem csökkenti a CIP Sync, de a jittert igen, amely viszont a szinkronizálásnak köszönhetően már jelentős válaszidő csökkenést eredményezhet [S16]. A CIP Sync tulajdonképpen a CIP protokoll egy módosított változata, amely a szinkronizálást az alkalmazási réteg szintjén valósítja meg, de opcionálisan az IEEE 1588 hardvertámogatottsága is felhasználható, ezáltal még jobb valós idejű teljesítmény érhető el. (2.6. ábra).



2.6. ábra. A PTP protokoll implementálása az alkalmazási rétegbe

A CIP Sync alkalmazásával az Ethernet/IP A osztályú besorolásból egyfajta C osztályba lép elő, vagyis alkalmassá válik szinkronizált hajtásvezérlésre, de még nem alkalmas szervóhajtások szabályozóinak vezérlésére, ugyanis a ciklusidő még mindig elég magas, tipikusan 5-10ms a reakció idő pedig 15-30ms körüli [34].

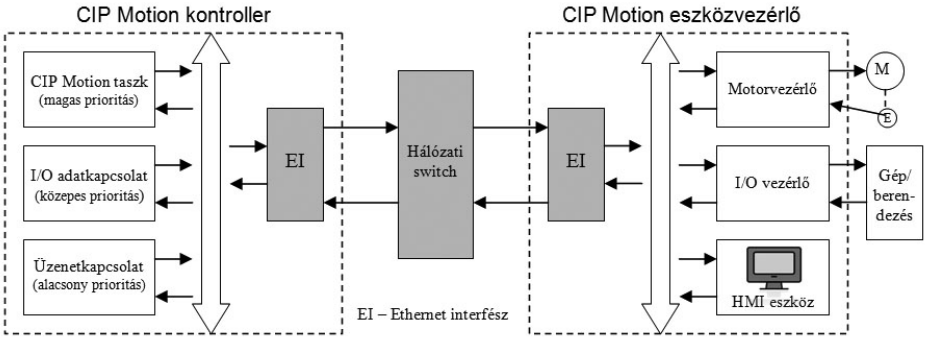
2.1.5. CIP mozgásvezérlő (CIP Motion) lehetőség implementálása

Az ODVA fejlesztései alapján, a Rockwell cég 2009-ben mutatta be az első olyan Ethernet/IP megoldást, amely már képes többtengelyes szervomotoros és frekvenciaváltós mozgásvezérlők ipari Ethernetes adatkapcsolatának bonyolítására [36]. A CIP Motion kettős feladatot lát el: egyrészt biztosítja a mozgásvezérlő eszköz interfész felületéhez az eszközvezérlőt, vagyis azt, hogy a mozgásvezérlő összekapcsolható legyen a frekvenciaváltóval vagy a szervo vezérlővel, másrészt a tengelyvezérlési funkciókat támogatja.

A CIP Motion implementálásával az Ethernet/IP alkalmassá vált mindhárom ipari kommunikációs osztály valós idejű követelményeinek kielégítésére. Ezekhez az osztályokhoz prioritási szinteket rendel az alábbiak szerint:

- mozgásvezérlés (CIP Motion objektum) – magas prioritás,
- I/O adatkapcsolat – közepes prioritás,
- üzenetkapcsolat – alacsony prioritás.

Az ODVA, a prioritással kitűntetett csomagok számára a VLAN címkézett keretek helyett a DSCP-t (Differenciated Service Code Protocol) alkalmazza, amely az IP fejléc Szolgálat típus (ToS) mezőjébe írja be a prioritási szinteket [36]. A 2.7 ábrán egy ilyen megvalósítás vázlata látható.

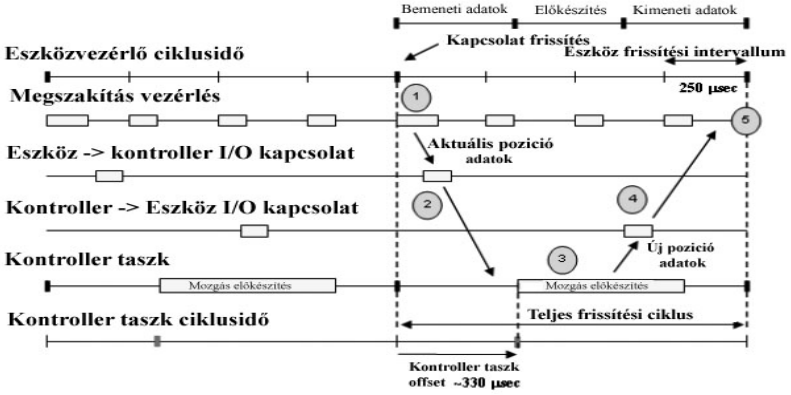


2.7. ábra. Teljes Ethernet/IP vezérlés

A CIP Motion kontroller hajtásvezérlő taszkja szinkronizált módon, ciklikusan kapja az aktuális helyzetadatokat a tengelyek pozícióiról, majd kiszámolja az új értékeket és továbbítja az eszközvezérlő felé. Mindez az Ethernet hálózaton keresztül történik szabványos UDP/IP csomagok formájában. A magas prioritási szintnek köszönhetően ezek az adatok mindig előnyt élveznek a többi adattal szemben, így ütközés nem fordulhat elő. A teljes frissítési ciklus három részre tagolódik:

- bemeneti adatok fogadása,
- az új vezérlési adatok előkészítése,
- kimeneti adatok küldése.

Ha 1ms-os teljes buszfrissítési ciklusidőt feltételezünk, akkor ez négy, 250 μs-os megszakítási intervallummal biztosítható. A megfelelő szinkronizálás biztosításával (CIP Motion kontroller taszk offset 330 μs) elérhető, hogy az új pozíció adatainak előkészítése még az eszközfrissítési intervallum előtt megtörténjen. (2.8. ábra) [36].



2.8. ábra. CIP Motion buszfrissítési ciklus időviszonyai.

Az előző ábrán bemutatott időviszonyok, valamint a CPU teljesítménye alapján meghatározható a CIP Motion kontroller legfontosabb jellemzője, mégpedig az, hogy hány tengelyt képes vezérelni 1ms alatt. A kontroller teljesítőképességének növelése érdekében

optimalizálhatjuk a keretterhelést. Az Ethernet/IP típusú rendszerek ezt úgy oldják meg, hogy bizonyos adatösszetevőket, amelyek hosszú ideig nem változnak, azokat csak egyszer, általában a kapcsolatfelvétel után továbbítják. Vagyis az adattovábbítási stratégia azon alapszik, hogy csak azokat az adatmezőket továbbítják minden ciklusban, amelyek változnak. Ennek a rugalmas adattovábbításnak köszönhetően a keretterhelés közel negyedére csökkenthető, ami pedig a kiszolgált tengelyvezérlők számának a növekedését eredményezheti.

2.2 A Profinet

A PROFINET szabványos (IEC 61158 és IEC 61784 [64]), nyílt ipari Ethernet hálózat, amely kielégíti az automatizálási technológiák és ipari folyamatirányító rendszerek minden követelményét. A PROFINET segítségével kivitelezhetők a decentralizált terepi eszközök közötti biztonságos adatkapcsolatoktól a gyors válaszidőt igénylő motorhajtásokig minden irányítási tevékenység. A valós idejű adatátviteli lehetőségei mellett a teljesen IT technológiára épülő előnyei játszanak fontos szerepet abban, hogy egyre inkább kezd elterjedni az ipari kommunikációban. A PROFINET lehetőséget biztosít a már meglévő terepi buszrendszerek: PROFIBUS, INTERBUS, ASI megoldások kiváltására, anélkül, hogy a meglévő eszközöket lecserélnénk.

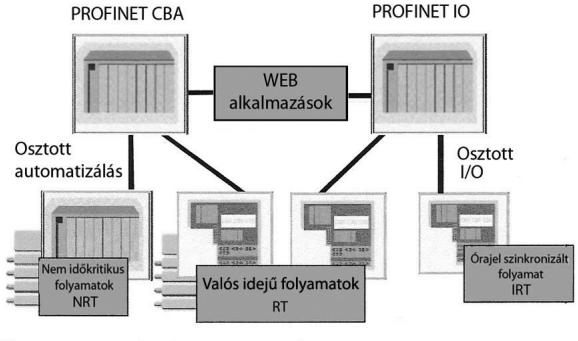
A PROFINET rendszer előnyei elsősorban a gyártásautomatizálás osztott rendszereinek (DCS) valós idejű ipari Ethernetes kommunikációjában nyilvánulnak meg. A rendszer szinte teljes egészében az információs technológia elemeire épül:

- szabványos csatlakozókat (RJ45) és hálózati elemeket (hub, switch, stb.) használ,
- gyakorlatilag az összes hálózati topológiát támogatja,
- biztosítja a hálózaton keresztül a diagnosztizálás lehetőségét,
- biztonságos kommunikációt tesz lehetővé,
- a WLAN segítségével a különleges mozgást végző gépelemek intelligens érzékelői is könnyen kapcsolódhatnak a vezérlőhöz.

A PROFINET használata egyben csökkenti az eszközök telepítési költségeit, kevesebb mérnöki és felügyeleti munkát igényel, de biztosítja a rendszerfejlesztés rugalmasságát és bővíthetőségét.

2.2.1. A Profinet CBA

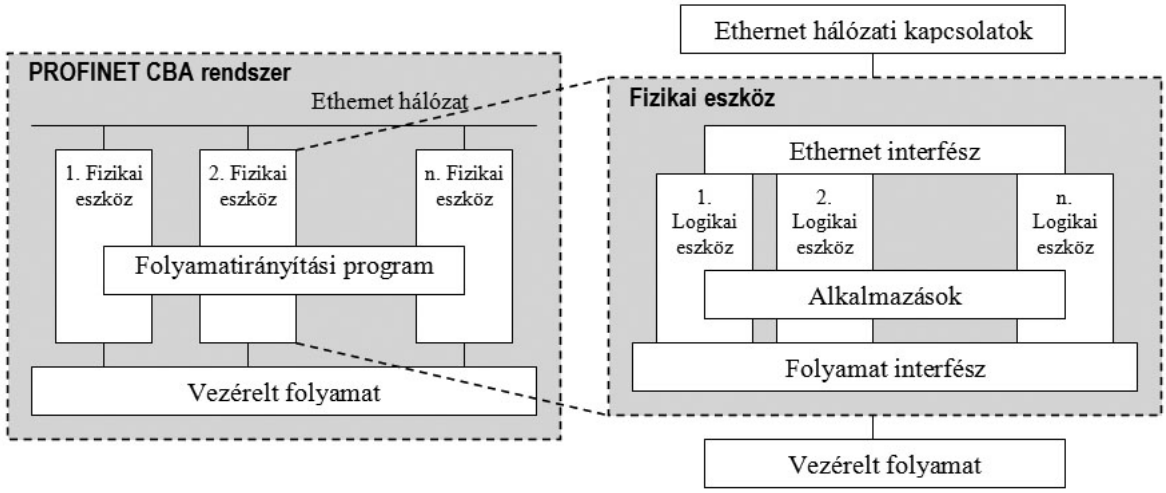
A PROFINET rendszer elemeit, funkcionalitásukat tekintve két külön kategóriába sorolhatjuk attól függően, hogy az automatizálás melyik szintjén teljesítenek szolgálatot, de természetesen közöttük is kiépíthető Ethernet kapcsolat. (2.11. ábra)



2.11. ábra. Egy teljes PROFINET rendszer kommunikációs kapcsolatai [39]

A PROFINET CBA-t komponens alapú, elsősorban programozható funkcionalitású intelligens terepi eszközök valamint kontrollerek számára alakították ki. A komponens modell alatt gépek és technológiai modulok egymástól függetlenül működő részegységeit kell érteni. A fix funkcionalitású komponensek mellett számos programozható funkcióval rendelkező komponens is részt vesz. Ez nagymértékben megkönnyíti a gépek, gépsorok valamint gyártóegységek modulrendszerű objektumainak kialakítását, amelyek jelentősen lecsökkentik a mérnöki költségeket. A komponensek leírásához a PCD-t (Profinet Component Descriptor) használja, amely XML-alapú és a gyártó komponens generátorával hozható létre.

A komponensek tekintetében a PROFINET CBA teljes mértékben alkalmazkodik a 2000-ben kibocsátott, az osztott automatizálási rendszerek IEC 61499-1 szabványához. Hierarchikus struktúra és objektumorientáltság jellemzi, melynek komponensei az eszközök, erőforrások, funkcióblokkok és az alkalmazás. A PROFINET CBA eszközei és erőforrásai között Ethernet kapcsolat van. A hierarchia csúcsán a rendszer áll, amely kapcsolatban lehet más rendszerekkel. A 2.12. ábrán egy ilyen rendszer- illetve eszközmodellt láthatunk [37].



2.12. ábra. PROFINET CBA IEC 61499-1 szerinti rendszermodell

A fenti modellben a fizikai eszközök a rendszert alkotó hardverobjektumokat jelentik, amelyek Ethernet hálózaton keresztül kapcsolódnak egymáshoz, illetve a hierarchia tetejét képező PROFINET vezérlőhöz. Mindegyik eszköz saját MAC címmel rendelkezik, és a konfiguráció során IP címet is kapnak. A logikai eszközök tulajdonképpen az erőforrásokat jelentik, amelyek a fizikai objektumok eszközvezérlő szoftverei. Az ezekhez beprogramozott funkcióblokkok összessége meghatározza az eszközhöz hozzárendelt alkalmazás objektumot. Ez egy futtatható program, amely saját adatterülettel rendelkezik és biztosítja az eszköz technológiai funkcióját. Ezeknek az alkalmazásoknak az összessége alkotja egy adott PROFINET CBA rendszer folyamatirányítási programját, azaz a futási objektumot (Runtime Automation Object).

2.2.2 A Profinet IO

Az Ethernetes kommunikáció szempontjából a PROFINET IO a legkritikusabb. Itt lehet ugyanis megvalósítani az órajel szinkronizált, mozgásvezérlőknél alkalmazott, meglehetősen időkritikus PROFINET IRT kommunikációt is. Architektúrája a jól ismert Profibus rendszer filozófiáját követi azzal a meghatározó különbséggel, hogy az eszközök közötti kommunikáció az ipari Ethernet nyújtotta lehetőségeken alapul. Ebből következik,

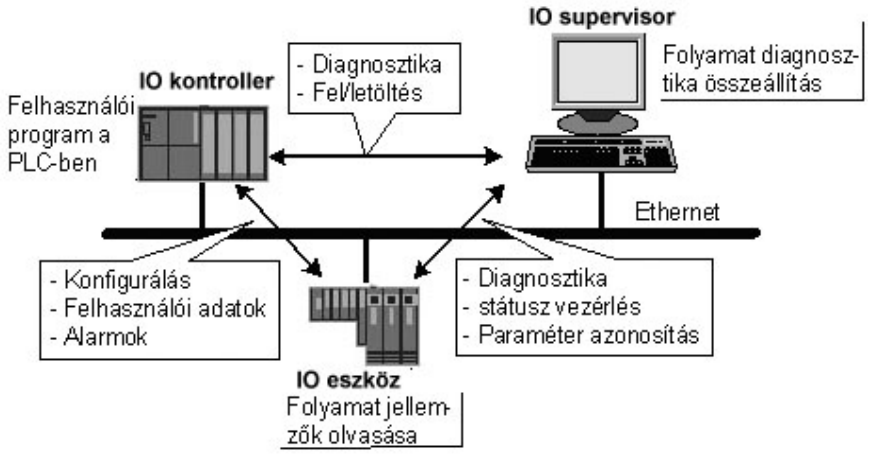
hogy a master-slave modellt az termelő-fogyasztó megoldás váltja fel, amely a kommunikációs kapcsolatok szemszögéből egyenrangú felekként tekinti a kapcsolatban résztvevő eszközöket.

A PROFINET IO rendszeren belül a nyílt internetes alkalmazások mellett mindhárom, a fejezet elején már bemutatott kommunikációs osztálybeli kapcsolat megvalósítható. Támogatja az összes lehetséges hálózati topológiát, beleértve a redundancián alapuló kapcsolatokat is. A rendszer felépítését tekintve alapvetően három elemre épül. Ezek a következők:

- PROFINET kontroller (IO kontroller): ez egy PLC, amelyen a felhasználói program fut,
- PROFINET IO eszköz (IO eszköz): decentralizált terepi eszközvezérlő, amely kommunikál a IO kontrollerrel,
- PROFINET IO supervisor: programozó vagy diagnosztikai eszköz.

Megjegyzés: Egy adott rendszeren belül általában több IO eszköz van konfigurálva. Maximális számuk a kontrollertől függ. (Tipikusan 1024 vagy 2048.)

A rendszer elemeit és az elemek közötti kapcsolatokat és a fontosabb feladatokat a 2.13. ábra szemlélteti.



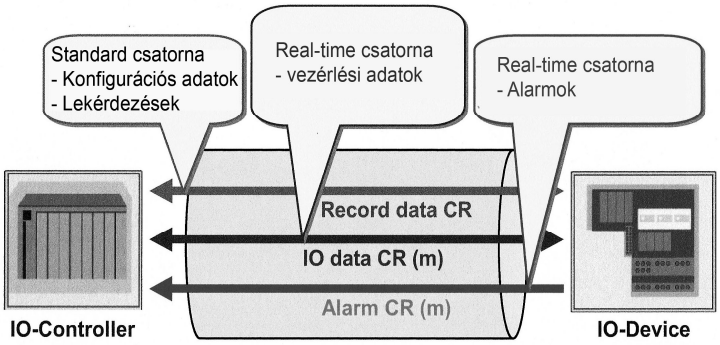
2.13. ábra. A PROFINET IO elemei és kommunikációs kapcsolatai

Míg a PROFINET CBA a komponens alapú modellt támogatja, a PROFINET IO moduláris rendszerű. A modulok fizikai slotokban vannak elhelyezve, amelyek bármikor bővíthetőek újabb modulok hozzáadásával és konfigurálásával. Egy slotnak lehet egy vagy több al-szlotja amelyek az IO csatornákat tartalmazzák 1-től n-ig [S10]. Ezek a csatornák jól meghatározott egyedi IO címmel azonosíthatók, amelyek a konfiguráció kialakítása után leképzésre kerülnek a kontroller adatmemóriájában. Az IO eszközökhöz tartozó konfigurációs fájlok XML alapúak. Ezeket a gyártó biztosítja, de a felhasználó is létrehozhatja a GSD-vel (General Station Description). Ebből következik, hogy adott gyártótól származó IO eszköz más ipari Ethernet rendszerbe is implementálható.

2.2.3 PROFINET kommunikációs stratégia

Az előzőekben már említettem, hogy a Profinet IO rendszer kommunikációs stratégiája az termelő-fogyasztó modellen alapszik. Az IO eszköz adatokat küld a kontrollernek a folyamat állapotáról. Ilyenkor az IO eszköz az termelő, a kontroller pedig a fogyasztó. Amikor a kontroller küldi a folyamat vezérléséhez szükséges kimeneti adatokat, akkor szerepet cserélnek, a kontroller lesz az termelő, az IO eszköz pedig a fogyasztó [S4].

Ahhoz, hogy a kommunikációs csatorna létrejöjjön, az termelő és a fogyasztó között egy virtuális logikai kapcsolatra is szükség van, amit *alkalmazás relációnak* (Application Relation – AR) nevezünk. Az IO kontroller mindegyik IO eszközzel felépít egy-egy AR kapcsolatot. Ezek a kapcsolatok automatikusan felépülnek közvetlenül a rendszer indítása után. Amikor egy kapcsolat létrejön az termelő és a fogyasztó között, az alkalmazás reláción belül épülnek ki a különböző *kommunikációs relációk* (Communication relation – CR) (2.14. ábra, [37, 38]). A szakirodalom szerint, mindezeket a *kontextus menedzsment* (Context management – CM) felügyeli és irányítja.



2.14. ábra. PROFINET IO kommunikációs relációk [39]

A kapcsolatfelvételt mindig az termelő kezdeményezi és a sikeres vétel nyugtázása után elbontja. Alapvetően háromféle kommunikációs kapcsolat alakulhat, attól függően, hogy milyen jellegű kommunikáció épül ki:

- Record Data CR nem ciklikus adatátvitelkor, pl. konfigurálás, lekérdezés,
- IO Data CR ciklikus bemeneti és kimeneti adatok továbbításakor,
- Alarm Data CR hibaesemények alkalmával (2.14. ábra).

Az utóbbi kettő a valós idejű, míg az első a hagyományos Ethernet csatornán keresztül jön létre.

A vezérlési adatok továbbítása tehát az IO Data CR ciklikus kapcsolaton keresztül bonyolódik, ezért ennek a működéséről érdemes kicsit bővebben beszélni. Az IO Data CR alapvető funkciója, hogy továbbítsa az IO adatokat a kontroller és az IO eszközök között a már említett termelő-fogyasztó modell szerint. A kapcsolat létrejötte során az adatokon kívül még az alábbi paraméterek is továbbításra kerülnek:

- egy lista a továbbításra kerülő IO adatobjektumokról,
- a küldési intervallum beállításairól, pl. küldési idő, küldési szorzó.
- az átviteli frekvencia

A létrehozandó kommunikációs kapcsolatok száma a PROFINET IO konfigurációjában van meghatározva attól függően, hogy hány IO eszköz van telepítve. A létrejött kapcsolat kétirányú, vagyis tulajdonképpen két ellentétes IO Data CR jön létre, amelyek állapotinformációt is tartalmaznak. Közvetlen nyugtázás a keretek érkezéséről nincs, viszont ha a fogyasztó nem kapja meg a kapcsolatfelvételkor a továbbítási listán szereplő adatokat három IO busz ciklusidő eltelte után sem, akkor hibaüzenetet generál. Az erre vonatkozó információkat az RT keret ciklusszámláló eleme tartalmazza, amely inkrementálódik, amikor az adott küldési ciklus alatt nem érkezik meg a vezérlési adat [S4].

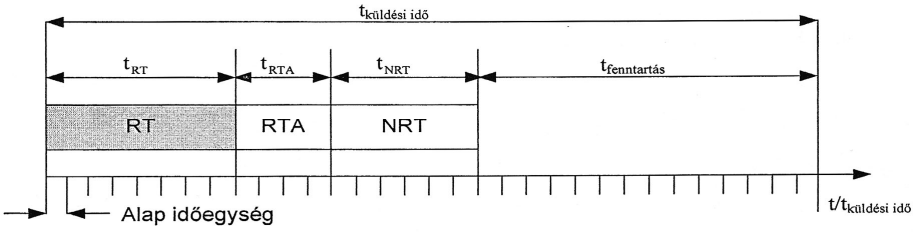
Néhány fontosabb jellemzőt szeretnék kiemelni itt, amelyek feltétlenül szükségesek a kommunikációs stratégiák elemzéséhez. Ezek a következők:

- a *küldési idő* (Send clock time)
- az *osztási tényező* (Reduction ratio)
- a *frissítési ciklusidő* (Update time)
- *küldési ciklusidő* (Send cycle)

A küldési idő az az intervallum, amely alatt a ciklikus adatok továbbításra kerülnek az IO Data CR-en keresztül. Eszköz specifikusan határozható meg a 31,25 µs-os alapidő egész számú többszöröseként.

$$Küldési\ idő = küldési\ tényező * 31{,}25\ \mu s$$

A küldési tényező 1 és 128 közötti egész számú értékeket vehet fel. Nem szinkronizált valós idejű adatok esetén ez a tényező 32 vagy annál nagyobb értékű lehet. Ha 32-vel egyenlő, akkor a küldési idő 1ms lesz. Ezt az esetet szemlélteti a következő, 2.15. ábra.



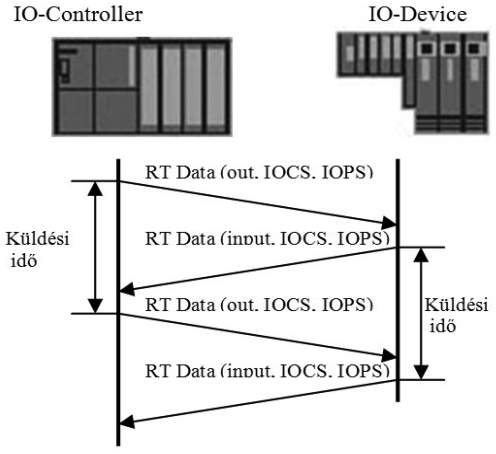
2.15. ábra. A küldési idő RT adatok esetében [37]

A fenti ábrában RT-vel a ciklikus, RTA-val a nem ciklikus valós idejű, valamint NRT-vel a nem valós idejű adatokat jelölik. Alapesetben a három időtartam együttesen nem haladja meg a teljes küldési idő 60%-át.

Mivel nincs mindig szükség nagyon gyors adatátvitelre, a küldési frekvenciák eltérőek lehetnek a különböző IO eszközöknél. Így a leglassúbb állomás nem tartja fel a teljes adatátvitelt. Ezek számára az *osztási tényező* segítségével alacsonyabb küldési frekvenciákat tudunk beállítani. Az osztási tényező mindig 2^n. Következik, hogy egy adott IO eszközvezérlő frissítési ideje:

$$Frissítési\ idő = 2^n \cdot t_{küldési\ idő}$$

Ebben a ciklusban kapják meg az IO eszközök a kontrollertől az adatokat és ugyancsak ilyenkor továbbítják az IO eszközök is a kontroller felé adataikat. Ezt a kétirányú ciklikus adatátvitelt szemlélteti a 2.16. ábra. Mindegyik IO adat mellett jelen van még két-két állapotjelző attribútumot tartalmazó információ is. Az IOPS az termelő, míg az IOCS a fogyasztóra vonatkozó attribútumokat tartalmazza [37].

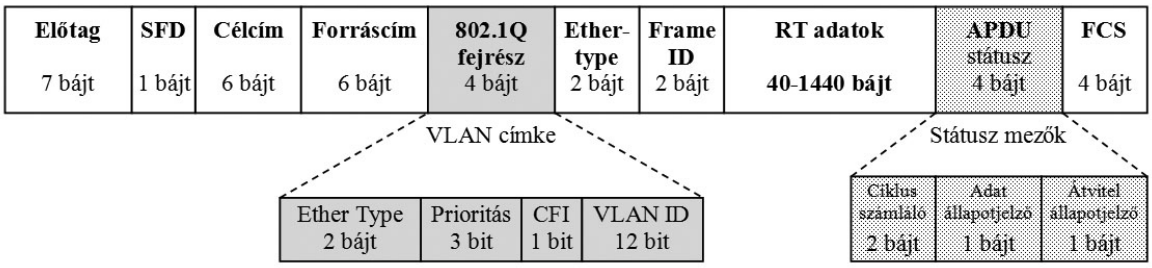


2.16. ábra. PROFINET IO valós idejű ciklikus adattovábbítási stratégia

A *küldési ciklus* azt az időtartamot jelenti, amely alatt az összes IO eszköz adatot cserél, vagyis az alrendszer technológiai ciklusidejét jelenti. Mivel az alrendszeren belül a kontroller és az egyes IO eszközvezérlők között különböző frissítési idők konfigurálhatóak, ez azt eredményezi, hogy a küldési cikluson belül, a tényleges küldési idő is változik, attól függően, hogy az adott frissítési ciklusban hány darab adatkeretet kell elküldjön.

2.2.4 A valós idejű protokoll elemei

Az előző fejezetben bemutatott kommunikációs relációknak megfelelően az adatok továbbításához a PROFINET különböző protokollokat használ, amelyek az Ethernet II keretformátumra épülnek. Ahhoz, hogy a valós idejű adatokat is továbbítani tudjuk Etherneten keresztül anélkül, hogy befolyásolnánk a hagyományos TCP/IP alapú IT kommunikációt, nyilvánvaló, hogy az 1.4. fejezetben ismertetett, IEEE 802.3 szabványnak megfelelő keretet, ki kell egészítenünk további elemekkel. Ezek egyrészt prioritást biztosítanak az RT kereteknek, másrészt segítségükkel azonosítani lehet az RT adattípusokat is, de az átvitel állapotára vonatkozóan is tartalmaznak információkat. A PROFINET erre a célra az IEEE 802.1Q szabványban foglalt VLAN címkés keretformátumot alkalmazza. Ez az előbbi formátumot kiegészíti még 2x2 bájtnyi információval (VLAN címke), a keret végére pedig az APDU (Application Protocol Data Unit) státuszinformációkat hordozó bájtok kerülnek (2.17. ábra) [S4].



2.17. ábra. Valós idejű adatokat szállító keretformátum

A VLAN címke négy fontos azonosítót tartalmaz:

- *EtherType* (2 bájt):
  – 0x8100 a keret VLAN protokoll azonosítója;
  – ha < 0x0600 a keret IEEE 802.3 szabvány szerinti;
  – ha 0x600 Ethernet II típusú keret (FrameID azonosítóval).
- *Prioritás* (3 bit): 0-7 közötti szintek, ahol az RT keretek 6, 7 prioritási szintet kapnak.
- *CFI* (1 Bit, formátum indikátor):
  – CFI = 0, Ethernet;
  – CFI = 1 Token ring.
- *VLAN-ID* (12 bit) prioritásazonosító:
  – 0x000 – a keret prioritással ellátott;
  – 0x001 – nincs prioritás;
  – 0x002-0xFFE – szabadon használható

További keretazonosítók:

– *EtherType* a protokollazonosító. A nem valós idejű adatokat tartalmazó keretek 0x0800, a valós idejű adatokat tartalmazó keretek pedig a 0x8892 PROFINET protokollazonosítót kapják.

– A *FrameID* információi a keret adatainak tulajdonságaira utal. (ciklikus, nem ciklikus, vagy Alarm típusú adatok) Például: 0xFC01 magas szintű hibajel típusú UDP/IP, vagy 0x8000-0xBEFF valós idejű (RT) adatot tartalmaz.

Megjegyzés: A 4 bájtos *VLAN tag* és a 2 bájtos *EtherType* számára a 6 bájtot az adatkitöltő bájtok (PAD) közül használunk fel azért, hogy az IEEE802.1Q-t nem támogató kapcsolók is átengedjék az ilyen típusú kereteket. Ezért az adatmező tartomány 40 és 1440 bájt között tölthető fel. A keret teljes hossza pedig az előtaggal együtt 1522 bájt.

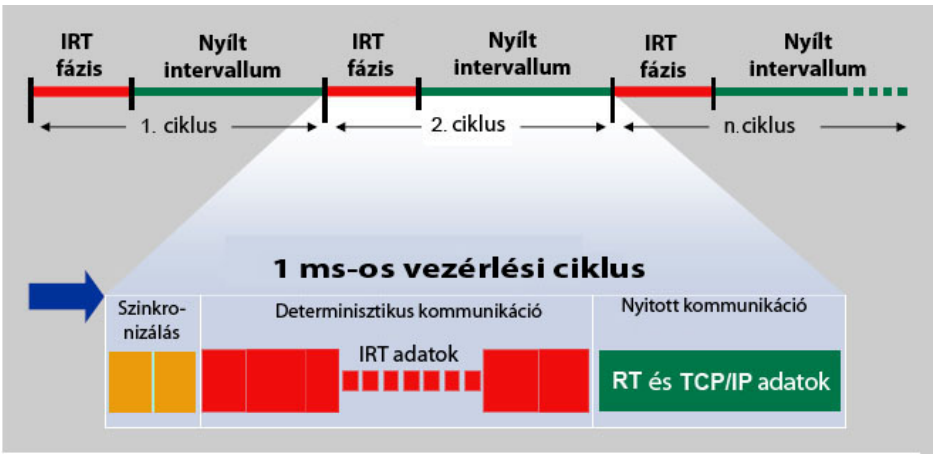
A státuszinformációkat tartalmazó bájtok közül kiemelném az adat-állapotjelző (Data Status) 5. bitjét, amely ha 1-es, akkor az átvitel hibamentes, vagy a Ciklusszámlálót (Cycle Counter), amely inkrementálódik, ha az termelő megismételte a küldést. Minden egyes inkrementálás időben 31,25 µs időt jelent. Ha a számláló túlcsordul (meghaladja a megengedett ismétlések számát), a fogyasztó a keretet eldobja, az alkalmazás pedig hibaüzenettel reagál.

A nem ciklikus valós idejű adatok (Alarm Data CR) hasonló keretformátumot használnak, azzal a különbséggel, hogy mivel itt nincs szükség állapotjelzésre, ezért az APDU státuszbájtok hiányoznak.

2.2.5. A Profinet IRT kommunikáció elve

A motorvezérléseknél, ahol időben akár 2-3 vagy annál több tengely összehangolt, pontos mozgatására van szükség, a szinkronizálás nélküli, valós idejű üzemmód nem alkalmazható. Az ilyen, időszinkronizált feladatok Ethernet hálózati kommunikációja a PROFINET IRT (Isochronous Real-Time) segítségével valósítható meg, amely röviden az alábbiakkal jellemezhető:

- A kommunikáció kizárólag egy hálózati szegmensen belül történik, mert a kapcsolat csak a fizikai és az adatkapcsolati OSI rétegeket használja [38].
- A busz ciklus időben két részre osztódik (2.18. ábra): IRT fázisra (un. piros intervallum), és egy azt követő nyílt intervallumra (un. zöld intervallum).
- 1 ms-nál kisebb ciklusidők és 1µs-os, vagy annál kisebb eltéréssel (*Jitter*),
- A küldési intervallumok szinkronizálása.



2.18. ábra. Az IRT kommunikáció időosztása [39]

A terepi eszközök kommunikációs időintervallumai rugalmasan állíthatók a felhasználó által. A piros intervallum (IRT fázis) nagysága főleg attól függ, hogy hány IRT eszköz csatlakozik a kontrollerhez. A kontroller konfigurálása során, amikor megadjuk az IRT eszközök számát, ez az intervallum automatikusan lefoglalásra kerül az IRT keretek számára. Az IRT keretekkel azonos időben érkező nem időkritikus kereteket az IRT

kapcsoló (ASIC – Application Specific Integrated Circuit) puffereli a zöld intervallumig, vagyis mindig szabad utat biztosít az IRT adatok számára. A zöld intervallumban a valós idejű (RT), prioritással ellátott, IEEE 802.3Q szabvány szerinti, valamint a nem valós idejű (NRT) keretek továbbítása valósul meg.

A zöld intervallumból (nyílt intervallum) a pirosba (IRT fázis) való átmenetet, a szinkronizálás fázisa alatt (narancssárga intervallum) a determinisztikus kommunikáció alapjául szolgáló hardver (ERTEC 400 vagy ERTEC 200) felügyeli. A küldési ciklus mindig a szinkronizálással kezdődik, amikor a kontroller órájához igazodnak az IRT eszközök órái.

2.2.6. A Profinet IRT protokoll

Az előző fejezetben bemutatott időosztásos kommunikációs stratégia, valamint a hardvertámogatás (ASIC) miatt a protokoll prioritási információkat nem tartalmaz. Ezért a VLAN címke használata felesleges. Az Ether Type itt is 0x8892, a 2 bájtnyi keretazonosító (Frame ID) értéke (0x0100 – 0x7FFF) pedig ciklikus IRT kerettípusra utal. További 8 bájt az eszközökre (IO Controller, IO Device) vonatkozó attribútumokat (IOPS, IOCS) és APDU státusz biteket tartalmazza. C osztályú kommunikációról lévén szó, a protokoll közvetlenül az adatkapcsolati rétegben építi fel a keretet, illetve választja le az IRT adatokat. Nincs beágyazott UDP/IP fejléc sem ezért a keret 36 – 1490 bájt között terhelhető. (2.19. ábra)

Előtag	SFD	Cél cím	Forrás cím	Ether Type	Frame ID	IRT adatok	FCS
7 bájt	1 bájt	6 bájt	6 bájt	2 bájt	2 bájt	36-1490 bájt	4 bájt

2.19. ábra. PROFINET IRT keretformátum

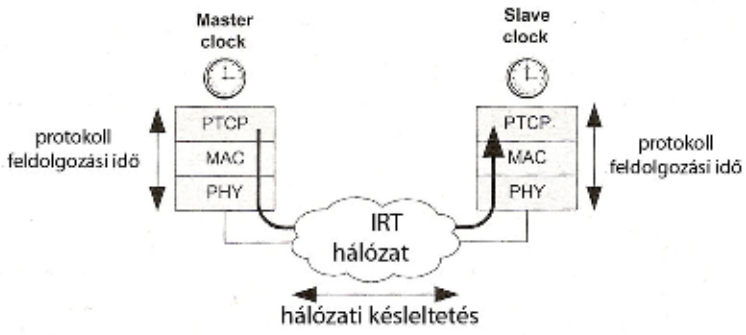
A motorvezérléseknél a vezérlési adatok nagysága csak ritkán haladja meg a 64 bájtot. A legtöbb esetben ez az adatmennyiség még a 36 bájtot sem éri el, ami azt jelenti, hogy csak a minimális 64 bájtos keret kerül továbbításra. Következik, hogy a keretterhelési tényező meglehetősen alacsony, a legtöbb esetben, alig több mint 56%. A keretterhelési tényező (F_{PF}) meghatározására a következő összefüggést használtam:

$$F_{PF} = \frac{d_{DATA}}{d_{FRAME}} \cdot 100\% \tag{2.2}$$

A fenti relációban d_{DATA} a vezérlési adatok nagyságát, d_{FRAME} pedig a teljes keretméretet jelöli bájtban kifejezve. Maximális keretterhelésnél ez az érték elérheti a 98%-ot.

2.2.7. Szinkronizálás PTCP protokollal

Szinkronizálás hiányában bármennyire is igyekszünk csökkenteni a ciklikus adatküldési időt, a mozgásvezérlők számára nem tudunk kellően precíz, összehangolt vezérlési adatokat továbbítani. Az IRT hálózat szinkronizálására a PROFINET a PTCP-t (Precision Transparent Clock Protokol) használja, amely hasonló az IEEE 1588 szabványban leírt PTP (Precision Time Protocol) protokollhoz [35, S16].



2.20. ábra. A PTCP beépülése az adatkapcsolati rétegbe [37]

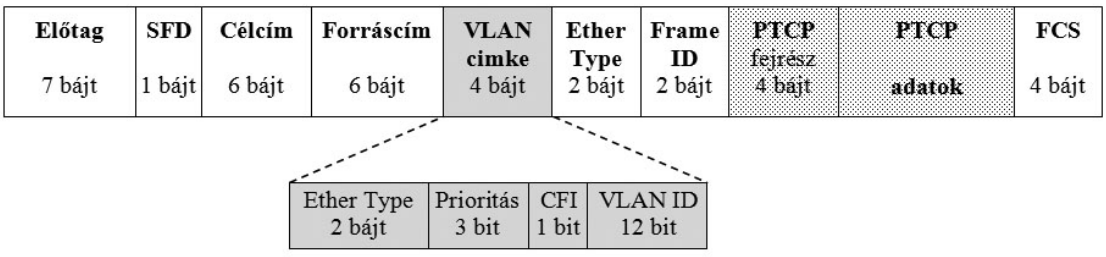
A szinkronizálás megvalósítása az ASIC egyik alapvető feladata. A PTCP az OSI réteg második szintjébe integrálódik, ez egyben azt is jelenti, hogy nem routolható (2.20. ábra). Ebből következik, hogy az IRT keretek csak azonos alhálózati szegmensen belül továbbíthatók.

A PTCP a következő fontosabb tulajdonságokkal rendelkezik:

- tized mikroszekundum nagyságrendű szinkronizálási pontosság (jitter),
- alacsony erőforrás használat (CPU, RAM),
- minimális hálózati sávszélességet igényel,
- alacsony adminisztrációs követelmények [37].

A PTCP segítségével akár 32 IRT eszköz szinkronizálása is megvalósítható egyetlen egy mester óra (Master Clock) alapján. A módszer lényege, hogy a csomópontokban un. átlátszó órát (transparent clock) alkalmaznak, amely a nem neki címzett szinkronkeretet átereszti, csupán egy bejegyzést szúr be a keretbe, amely a csomópont késleltetéséről tartalmaz információt. Így a saját csomópontbeli órajel szabályozási kör szinkronizálási hibája nem kerül továbbításra. Ebből következik, hogy a jitter maximális értéke lineárisan növekszik, és csak az IRT eszközök számától függ [S16].

A mester óra funkcióját betöltő eszköz rendszerint az IO kontroller. A szinkronizáláshoz szükséges adatokat a szinkronkeretekben továbbítja az IO eszközök számára. (2.21. ábra)



2.21. ábra. PTCP keretformátuma

A kereten belül fontos szerepet tölt be a keretazonosító (Frame ID). Ennek értéke mutatja, hogy éppen milyen típusú szinkronkeret kerül továbbításra: Sync, Follow Up, DelayReq vagy DelayRes. A PTCP fejrész számon tartja, hogy hányadik szinkronkeret került továbbításra, valamint tartalmazza a szinkronizálás pontosságára vonatkozó információt. (10ns, 1ns)

A 2.17. és 2.21. ábrákon bemutatott protokollok további mezőinek specifikus értékei részletesen az 1. számú mellékletben találhatók.

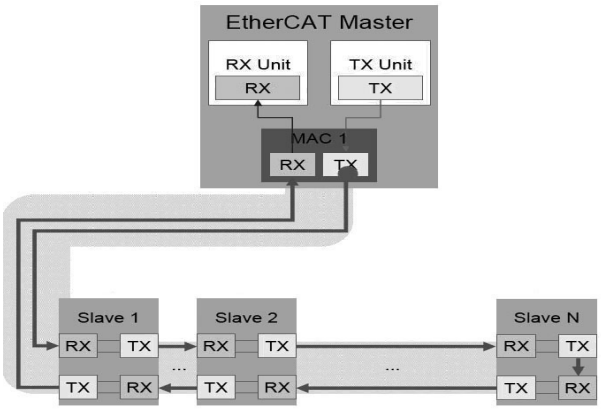
2.3. EtherCAT

A szigorúan valód idejű ipari Ethernet kommunikációs rendszereknél (C osztály) a vezérlési adatok továbbítása két módon valósítható meg. Az egyik megoldás, amikor a kontroller (master) mindegyik, vele Ethernet kommunikációs kapcsolatban lévő IO eszköznek (slave) külön-külön címzi a kereteket, amelyek a megfelelő vezérlési adatokat tartalmazzák. Ezt a módszert alkalmazza az előző fejezetben bemutatott PROFINET IRT rendszer. Láthattuk, hogy ebben az esetben 36 bájtnál kisebb vezérlési adatok gyakorlatilag nem befolyásolják a továbbításhoz szükséges időt, mert így a teljes keretméret nem haladja meg a minimális, 64 bájtot. Hátránya ennek a megoldásnak, hogy az előbb említett esetben a keretterhelési tényező a legjobb esetben is csak 56% lehet. Igazi előnye az 1000 Mb/s-os Gigabit Ethernet hálózaton mutatkozik meg, amikor az adattovábbításhoz szükséges idő gyakorlatilag már nem függ a keret méretétől [S23].

A másik megoldás, amikor arra törekszünk, hogy kihasználjuk a maximális keretterhelést, amely az IEEE 802.3 szerint 1500 bájt lehet. Ebben az esetben az IO eszközök számára továbbítandó vezérlési adatokat (telegramokat) igyekszünk egy vagy több kereten belül úgy elhelyezni, hogy minél nagyobb legyen a keretek kihasználtsága. Ez a módszer kétségtelenül előnyt jelent kisméretű (36 bájtnál kisebb) telegramok továbbítása esetében. Így egy Ethernet kereten belül egyszerre továbbíthatja akár az összes csomópontban lévő eszközök vezérlési adatait. Ezt a megoldást alkalmazza a Beckhoff cég által fejlesztett EtherCAT (Ethernet for Control Automation Technology) [41].

2.3.1. Az EtherCAT rendszer fontosabb jellemzői

Az EtherCAT-et igen rövid ciklusidő, zavarmentes kommunikáció és nagy pontosságú szinkronizálás jellemzi. 100 Mb/s-os full duplex Ethernet hálózaton szegmensenként gyakorlatilag szinte korlátlan számú (max. 65 535) IO eszköz kapcsolható össze egymással. A gyártó adatai szerint 100 IO eszköz, összesen 1000 digitális IO csatorna esetében a frissítési idő mindösszesen 30µs, de 100 darab szervo tengely, egyenként 8 bájtos IO adatokkal történő hajtásakor sem sokkal haladja meg a 100µs-ot [40]. E rendkívüli gyorsaságot annak köszönheti, hogy a csomópontokban lévő IO eszközök „menet közben”, közvetlen memória hozzáféréssel (DMA) olvassák le a nekik címzett, illetve írják fel a továbbítani kívánt adatokat (2.22. ábra). Az erre a célra kialakított Ethernet hardvernek köszönhetően a csomópontokon belüli késleltetési idő rendkívül rövid, mindösszesen 1,35 µs full duplex hálózatban [S5].



2.22. ábra. EtherCAT adattovábbítás

Az EtherCAT kontroller a feladat elvégzéséhez nem szükséges, hogy rendelkezzen semmilyen extra kommunikációs processzorral. Ebből következik, hogy bármilyen kontroller, amelyik rendelkezik szabványos Ethernet interfésszel alkalmas erre a feladatra, függetlenül attól, hogy milyen operációs rendszert vagy alkalmazást használ.

A rendszer a vezérlési adatok nagysága szempontjából sem mondható korlátozottnak. Noha a motorvezérléseknél nincs szükség nagyméretű telegramok továbbítására, a gyártó 60 kilobájtban határozta meg a maximális adatmennyiséget, amely továbbítható a csomópontoknak. (Ebben az esetben természetesen több keretre van szükség) [40].

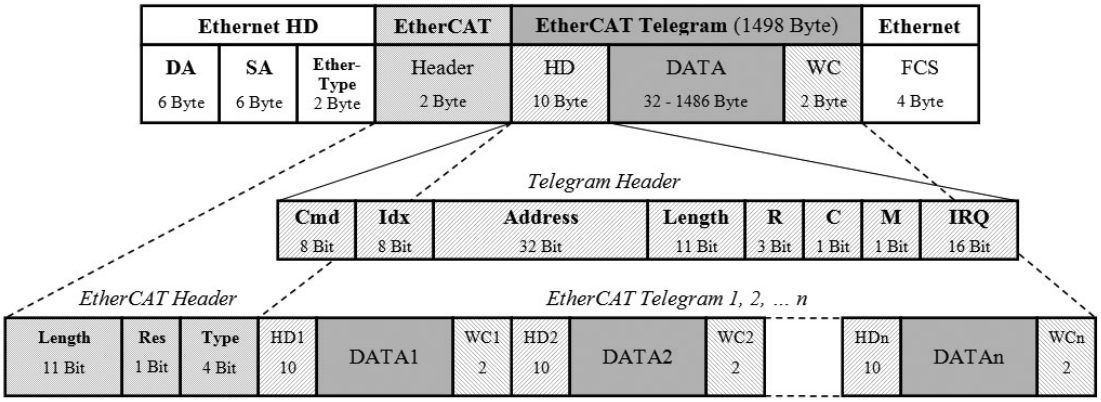
A rendszer elsősorban a busz topológiát támogatja, de jól használható, csillag, fa és vegyes topológiájú hálózatokban is. Vegyes topológiában a ciklusidő azért lényegesen nagyobb, mint például a busz topológiában.

2.3.2. Az EtherCAT protokollok

Egy teljes EtherCAT rendszer alapvetően kétféle kommunikációs protokollt használ, amelyek a szabványos IEEE 802.3 Ethernetre épülnek:

- nem szinkronizált folyamatokhoz, mint például folyamatautomatizálás vagy konfigurációs beállításokhoz, folyamatvizualizáláshoz valamint lekérdezésekhez a TCP/IP vagy UDP/IP alapú EtherCAT Automation Protokollt (EAP), vagy más néven mailbox protokollt,
- szinkronizált, ciklikus vezérlési adatok, mint például motorvezérlések vagy szabályzók adatainak továbbítására az EtherCAT eszköz-protokollt (EtherCAT Device Protocol).

Ez utóbbi esetben a keret azonosítása EtherType-al történik, amelynek értéke 0x88A4. A master így tudja megkülönböztetni a vezérlési adatokat tartalmazó kereteket más típusú Ethernet keretektől. Mindegyik keret tartalmaz még egy kétbájtos EtherCAT fejlécet is, amely főleg a beágyazott adatmennyiség hosszára (Length) és típusára (Type) vonatkozóan tartalmaz információt. (2.23. ábra). Az eszköz-protokoll esetében a Type mező értéke 1-el egyenlő. A motorvezérléseknél használt, erősen hardver-beállítottságú (FPGA) slave eszközök csak az ilyen típusú kereteket képesek kezelni.



2.23. ábra. EtherCAT eszköz-protokoll keret struktúrája

Azért, hogy egy kereten belül, több csomóponti eszköz számára is tudjuk továbbítani a vezérlési adatokat (Type = 1), azonosítás céljából mindegyik telegramot további fejléccel (Telegram Header, HD), illetve működésszámlálóval (Working Counter, WC) kell ellátnunk. Az erre a célra felhasználható maximális keretterhelés 1498 bájt lehet. Ebben az esetben a (2.2) szerinti keretterhelési tényező elérheti akár a 97%-ot is. Ha az összes

telegram nem fér el egy keretben, a master további keretekben helyezi el ezeket. Ha túlságosan kisméretű a telegram és kevés a slave eszköz, akkor töltelékbitekkel (Pad) egészíti ki egészen 64 bájtig [S5].

2.3.3. Címzési módok

A slave eszközök címzése, illetve a telegramok célba juttatása, valamint az, hogy mit kell csinálni a kapott adatokkal, az EtherCAT kommunikációs stratégiájának a legfontosabb faladata. Ennek lebonyolításában nyújt segítséget a telegram fejléc, amelynek mezőit és azok funkcióit a 2.1. táblázat foglalja össze.

2.1. táblázat. Az EtehrCAT telegram fejlécének mezői és funkciói

Mező	Adat típus	Érték/Magyarázat
Cmd	**Bájt (1)**	**EtherCAT utasítás típus**
Idx	**Bájt (1)**	**A master által használt index a dupla vagy elveszett telegramok azonosítására.**
Address	**Dupla szó (4 Bájt)**	**Címzési mód meghatározása (Automatikus címinkrementálás, csomóponti címzés, logikai címzés)**
Length	**11 Bit**	**A telegramhoz tartozó adat hossza**
R	**3 Bit**	**Nem használt, értéke nulla**
C	**1 Bit**	**Keret körforgás: engedélyezés C =1; tiltás C = 0**
M	**1 Bit**	**További telegram: utolsó telegram M = 0; van következő telegram M = 1**
IRQ	**Szó (2 Bájt)**	**Eseményregiszter. A slavek írják és logikai VAGY kapcsolaton keresztül a master elemezi.**

A címzési stratégia során különbséget kell tegyünk keret- illetve telegramcímzés között. Busz topológiában a master a keretet mindig a hozzá közvetlenül kapcsolódó legelső slave eszköznek címzi, vagyis az Ethernet célcím (DA) mezőbe ennek az eszköznek a MAC címe kerül. A forráscím (SA) értelemszerűen a master MAC címe lesz. A kereten belül a telegramok kétféle címzési mód szerint juthatnak célba: eszközcímzéssel illetve logikai címzéssel. Eszközcímzésnél lehetőség van háromféle címzésre: pozíció- illetve csomópontcímzés és broadcast.

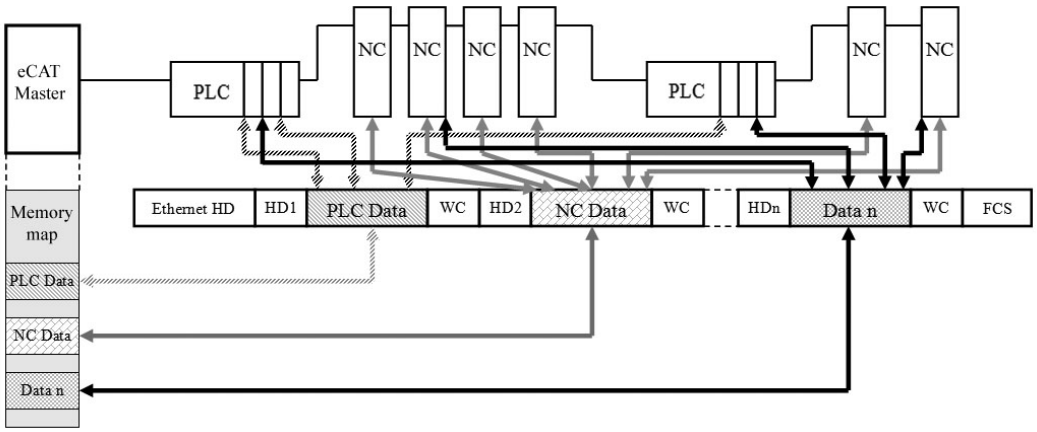
A legegyszerűbb és a leggyakrabban alkalmazott mód a pozíciócímzés. A telegramok, kivéve az elsőt, az eszközök negatív címértékét tartalmazzák. Mindegyik slave, kivéve az utolsót, miután befejezte a saját adatcseréjét, inkrementálja a soron következő telegramok címmezejének pozíció bájtjait (2 bájt pozíció, 2 bájt offset). Az az eszköz lesz megcímezve, amelyik a nullás címet olvassa. Hátránya ennek a módszernek, hogy az eszközök csomópontbeli sorrendje nem cserélhető fel.

Csomópontcímzéskor a master a konfiguráció során beolvasott címek alapján végzi a telegramok címzését. Ezek a címek a slavek EPROM-jaiban vannak tárolva és nem módosíthatóak. Az eszköz azt a telegramot olvassa/írja, amelyiknek a címe megegyezik az EPROM-jába beírt értékkel. Leginkább vegyes topológia esetén használják.

A broadcast címzés során mindegyik csomóponti eszköz kiválasztásra kerül. Ezt a címzési módot inicializáláskor és állapotlekérdezéskor használja a vezérlő.

Logikai címzéskor a vezérlő memóriatérképének a konfiguráció során meghatározott részei kerülnek a telegramokba. A slave eszközök a memóriakezelő egységeik (FMMU – Fieldbus Memory Management Unit) segítségével a saját memóriájukba képezik ezeket az

adatokat. Indításkor, ugyancsak a konfiguráció során meghatározott beállításokkal, a vezérlő tudatja mindegyik eszköz FMMU-val, hogy pontosan melyik memóriaterületet olvashatják, illetve írhatják. A bitcímzési mód is megengedett ebben az esetben, így a digitális IO eszközök adatai is viszonylag könnyen továbbíthatóak. A módszert főleg a folyamatirányítási adatok célba juttatásakor használják, jelentősen csökkentve ezáltal a telegram fejlécek számát, tovább javítva a keretterhelési tényezőt (2.24. ábra) [42].

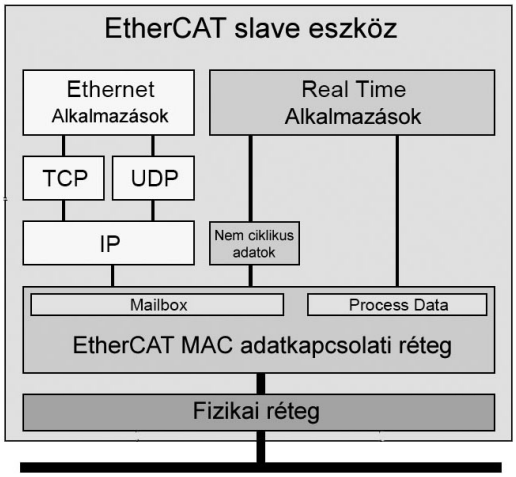


2.24. ábra. A logikai címzés

2.3.4. Ethernet az EtherCAT felett

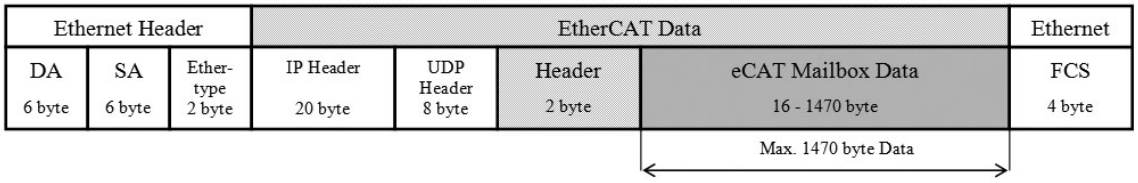
Az EtherCAT slave interfészek teljesen kompatibilisek a szabványos Ethernettel, vagyis beágyazódnak az EtherCAT MAC alrétegébe. Ez az alréteg meg tudja különböztetni a ciklikus adatokat, a nem ciklikus „mailbox" típusú adatoktól. Ezen kívül a szabványos Ethernet alkalmazások számára is teljesen átjárhatóak a TCP/IP vagy UDP/IP protokollok, így az eszközök, legfőképpen a master zavartalanul kommunikálhat nem EtherCAT rendszerbeli eszközökkel is. Ez továbbá azt is jelenti, hogy a topológia bármely szabad Ethernet portjára bármilyen informatikai eszköz csatlakoztatható anélkül, hogy zavart okozna az EtherCAT rendszerben.

A következő ábrán (2.25. ábra) a mailbox protokoll beágyazását látjuk az adatkapcsolati rétegbe, egész pontosan az EtherCAT MAC alrétegbe.



2.25. ábra. A mailbox protokoll beágyazása a MAC alrétegbe

A folyamatirányítási adatok továbbításakor (Type = 4) vagy a nem ciklikus un. „mailbox” típusú adattovábbításhoz (Type = 5) az EAP protokoll az UDP/IP kommunikációt használja. Ilyenkor a fejrész is beágyazódik az EtherCAT adattartományba, így maximálisan már csak 1470 bájtnyi adat továbbítható (2.26. ábra).

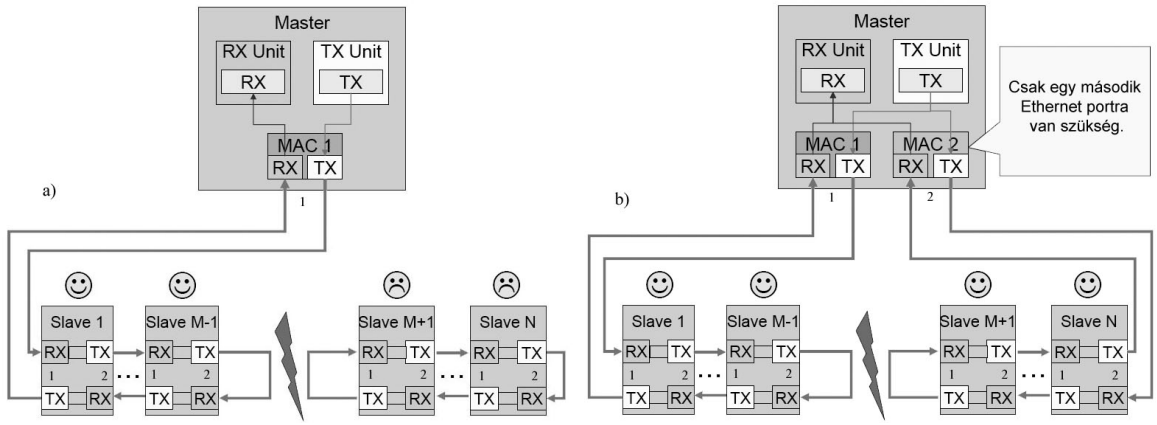


2.26. ábra. Mailbox típusú keret struktúra

Az ilyen típusú EtherCAT kereteknél az EtherType azonosító értéke 0x800 vagyis azonos a szokásos Ethernet keretek azonosítójával és a 0x88A4 UDP portot használja.

2.3.5. EtherCAT redundancia

A lineáris topológiát alkalmazó ipari Ethernet rendszerek egyik gyakran előforduló problémája a kábelszakadás. A slave eszközök felismerik a kábelszakadást. Ilyenkor azok a portok, amelyek érzékeli a szakadást automatikusan lehurkolják, Rx – Tx vonalukat. A vezérlő és a szakadási pont közötti szakasz látszólag továbbra is működőképes marad, viszont a vezérlő érzékeli, hogy a slave-ek egy része nem cserél adatot, mert a nekik címzett telegramok számlálója (WC) nem inkrementálódott. Egy másik probléma, hogy a szakadás utáni szakaszon pedig elkezd „körbeforogni” a legutolsó keret, mert a szakadást érzékelő port itt is lehurkolta vonalait. (2.27.a. ábra) Ez a helyzet akár veszélyes is lehet ezért, hogy ezt elkerüljük, a szakadást érzékelő eszköz a fejlécben a C bitet (lásd 2.1. táblázat) nullára állítja, ezzel megakadályozza a körbeforgást, eldobja a telegramot és hibakóddal reagál.



2.27. ábra. Kábel redundancia megvalósítása [41]

A probléma megoldása nagyon egyszerű. A vezérlőben csak egy második Ethernet port konfigurálására van szükség és az utolsó slave 2. portját összekötni a vezérlő 2. portjával. (2.27.b. ábra) Az így módosított topológia szerit a vezérlő elsődleges portjáról küldött kereteket a másodlagos port fogja fogadni és fordítva. Hibátlan rendszernél, a két keret tartalma azonos kell legyen. Ezt a vezérlő ellenőrzi. Azért, hogy ne duplázódjanak a keretek, a master az elsődleges port által fogadott kereteket eldobja. Ha szakadás történik,

az ezt érzékelő portok azonnal lehurkolják vonalaikat, a kommunikáció nem szakad meg a vezérlővel egyik eszköz felől sem. Viszont a master, miután összehasonlítja a két fogadott keretet látja, hogy azok különbözőek, nem dobja el egyiket sem, hanem a kettőből összerakja a helyes memóriatérképet.

2.4. További ipari Ethernet rendszerekről röviden

Az előző fejezetekben három alapvető, egymástól különböző elveken alapuló, igen elterjedt valós idejű ipari Ethernet megoldást mutattam be. Ezek a megvalósítások mind különböző gyártókhoz kapcsolhatók. Ez nem véletlen, ugyanis a gyártók, amikor a lehetőségek megnyíltak az Ethernet irányába igyekeztek egymástól független megoldásokat implementálni. A továbbiakban nagyon röviden jellemeznék még néhány jelentősebb gyártóhoz kapcsolódó ipari Ethernet megoldást.

Az osztrák B & R Automation cég által kifejlesztett Ethernet PowerLink (EPL) kifejezetten gyors és pontos, C osztályú átviteli sebességet biztosít akár 200 μs-os ciklusidővel és 1 μs-os jitterrel. Gyorsaságát annak köszönheti, hogy az adattovábbítást akárcsak a Profinet IRT két fázisban hajtja végre. A szinkron fázisban mindegyik állomás rögzített szélességű időrést kap a valós idejű adatok továbbítására. A telegramok azonosítását a keretbe ágyazott EPL azonosító biztosítja. Az aszinkron fázisban kerül sor a hagyományos IP alapú információ továbbítására. A vezérlési adtok küldésekor egy időben mindig csak egy állomás férhet hozzá a hálózathoz, így ütközés nem fordulhat elő [43, 44].

A német SERCOS International cég az EtherCAT-hez hasonló elven működő megoldást dolgozott ki, amelyet SERCOS III néven forgalmaznak. A szigorúan valós idejű követelményeket is képes teljesíteni. Busz vagy gyűrű topológiában ciklusideje 31,25 μs-tól 65ms-ig konfigurálható. 100 Mb/s-es Full duplex Ethernet hálózatot feltételez csavart érpáron vagy optikai szálon [45, 46].

A MODBUS/TCP alkalmazása főleg a Schneider Electric gyártmányokban kerül felhasználásra. A MODBUS az egyik legismertebb ipari protokoll, amelyet még a MODICON cég fejlesztett ki. Ezt fejlesztették tovább ipari Ethernetre, úgy hogy tulajdonképpen a MODBUS keretet ágyazzák be a TCP keretbe. Ezáltal A osztályú valós idejű kommunikációt valósít meg kérés/válasz mechanizmussal, amely jól illeszkedik a master-slave technikához. Ezzel akár több ezer eszközt is képes „megszólítani” [47].

Végül, de semmiképp sem utolsó sorban a Mistsubishi Electric CC-Link elnevezésű megoldását említeném, amely 1 Gb/s-os Ethernet hálózaton, gyűrű topológiában a „token passing” stratégiát alkalmazza. Optikai kábelen keresztül akár 66 km átviteli távolságú hálózat kialakítására is alkalmas. A valós idejű adatok számára ciklikus kommunikációt biztosít. A nem valós idejű adatok továbbítása a tranziens fázis alatt történik. Egy kontroller maximum 119 állomást tud kiszolgálni. Ugyan nagy sebességű hálózatot használ, de ennek ellenére meglehetősen lassú, 16 csomópontnál 256 bájt vezérlési adattal a ciklusidő kb. 2 ms [63].

III. FEJEZET

AZ IPARI ETHERNET, MINT VALÓS IDEJŰ RENDSZER

Ebben a fejezetben bemutatom mindazokat az elméleti elemeket, amelyek alapján meghatározhatók a válós idejű rendszerek alapköveteményei annak érdekében, hogy implementálhatóak legyenek az ipari Ethernet hálózatban.

3.1. A valós idejű rendszerekkel kapcsolatos alapfogalmak

A valós idejű rendszerek fogalma a digitális számítógépek automata célgépekként történő alkalmazásával egy időben jelent meg. A mikroszámítógépek és mikrokontrollerek egyre inkább beépülnek a vezérléstechnikai rendszerekbe. Egyre könnyebb programozásuk és egyre kedvezőbb áruk miatt, már a legegyszerűbb automaták is tartalmaznak valamilyen mikroprocesszort, vagy mikrokontrollert. A valós idejű rendszerekben az események feldolgozásában az idő alapvető szerepet játszik. Az alapprobléma, hogy egy vagy több külső eszköz előre nem ismert időpontban valamilyen esemény által kiváltott információt küld a számítógépnek, amelyre az eseményt kiváltó októl függő, előírt időn belüli választ kell küldeni. A környezetből egyidejűleg több eseményhez kapcsolódó üzenet is érkezhet a számítógéphez és ebben az esetben is teljesülnie kell az előírt időn belüli válaszadásnak.

3.1.1. A valós idejű viselkedés meghatározása

Az ilyen rendszerek sajátossága, hogy a fizikai környezetükkel aktív, valós idejű információs kapcsolatban állnak. A rendszer bemenetére érkező információkat kiértékelik, feldolgozzák és meghatározott időn belül kell választ adniuk. Ezeket a rendszereket a befogadó természetes környezet és a mesterségesen létrehozott hardver és szoftver komponensek nagymértékű összekapcsolódása, együttműködése jellemzi és a környezetükkel, – amely jellemzően nem informatikai rendszer – közvetlen információs kapcsolatban állnak. Ezek, általában olyan digitális eszközök, amelyek speciális célra készültek, és nem rendelkeznek bonyolult felhasználói felülettel.

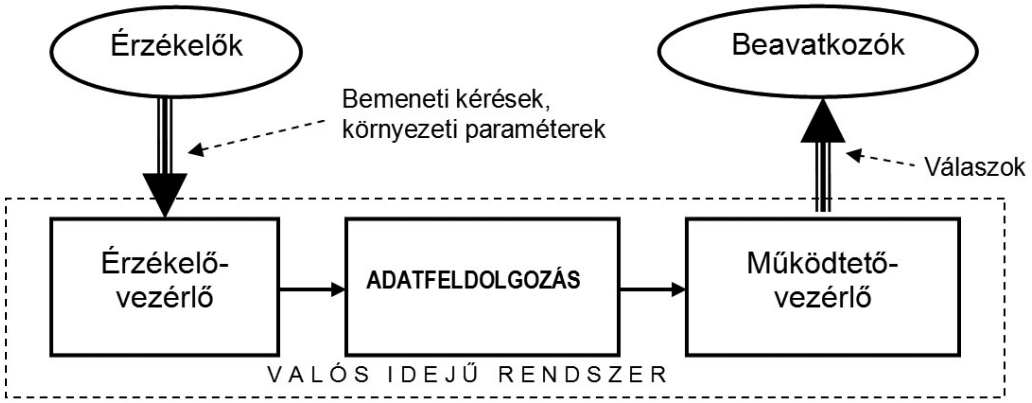
A valós idejű rendszerek definícióját csak a tulajdonságai alapján nem tudjuk meghatározni. Egy meglévő rendszerről addig nem lehet eldönteni, hogy valósidejű-e vagy nem, amíg nem ismerjük meg a rendszer feladatspecifikációját. Bizonyos időhöz kötött tulajdonságok alapján feltételezhetjük, főleg hogyha időmérő eszközöket, vagy időfüggő algoritmusokat használ. A feladat ismerete nélkül azonban nem dönthető el, hogy ezek használata valamilyen időbeli viselkedésre vonatkozó követelmény teljesítéséhez vagy más szempontok miatt szükséges.

A definíciót tehát nem a rendszer tulajdonságai, hanem a vele szemben támasztott követelmények alapján kell megadni. Az évek során számos klasszikus értelemben vett definíció látott napvilágot [23, 24, 25], melyek alapján kijelenthető: *Valós idejűnek tekinthetjük azt a rendszert, amelyek specifikációjában a logikai és számítási funkciókon túlmenően valamilyen időbeli viselkedésre vonatkozó előírás is szerepel a külső, valós időskálához kötötten [27]*.

3.1.2 A valós idejű rendszerek fontosabb jellemzői

Egy valós idejű rendszer lényegét tekintve abban különbözik egy adatfeldolgozó rendszertől, hogy:

- a környezetéből érkező jeleket érzékelők szolgáltatják. Ezeknek feladata a működtetett (irányított) rendszer, környezet állapotáról információkat adni.
- a feldolgozás eredményét (választ) előírt idő alatt kell a beavatkozó eszközre juttatni. (3.1. ábra)



3.1. ábra. Egy valós idejű rendszer általános modellje [27]

A feldolgozó program nagysága természetesen az általa megvalósítandó feladatot leíró algoritmustól függ, de funkcionálisan tartalmazza az 3.1. ábra szerinti három rész funkciót is, amelynek alapján folyamat jól meghatározható. Ez szekvenciális programokban nem mindig kivitelezhető. Ezért a valós idejű rendszereket konkurens, együttműködő folyamtokként szokás tervezni. Ezeket a folyamatokat a valós idejű rendszer specifikusan kialakított operációs rendszere kezeli. A folyamatok futtatása rendszerint ciklikus, ebből következik, hogy egyik alapvető időtényezője a rendszernek éppen a *ciklusidő* nagysága lesz.

Egy valós idejű rendszer fontosabb jellemzői a következők:

- az irántyó berendezés rendszerint valamilyen intelligens rendszer (PLC, mikrokontroller, de akár PC is lehet),
- idő és/vagy esemény vezérelt működés,
- az irányító rendszer online kapcsolatban van a technológiai berendezéssel az érzékelő, beavatkozó eszközökön keresztül,
- az előírt *időkorlát* (határidő, deadline) az irányított technológiai berendezés, folyamat időviszonyaiból határozható meg.

A számítógépek ilyenfajta alkalmazása szükségessé tette, hogy a számítógépes rendszer és a benne futó programok időbeli viselkedését is vizsgáljuk, illetve specifikáljuk, mégpedig a környezetében érvényes, valós időskálán. A számítógépes rendszernek úgy kell működnie, hogy *válaszideje* (response time) a megengedett időkorláton belül maradjon. Ha nem sikerül adott időn belül reagálni, az ugyanolyan hiba, mint egy téves számítási eredmény előállítása, sőt következményeit tekintve bizonyos területeken (pl. ipari biztonsági rendszereknél) még súlyosabb is lehet.

A valós idejű rendszerek általában két csoportra oszthatók:

- **szigorúan valós idejű (hard real-time)** rendszerek, amelyek specifikációja egyértelműen rendszerhibának tekinti valamely időkövetelmény be nem tartását. Ez azt jelenti, hogy a szigorúan valós idejű rendszerek normál működése során nem

engedhető meg, hogy valamilyen időkövetelmény teljesítése elmaradjon. Ha mégis bekövetkezne ilyen esemény, a rendszernek a kidolgozott stratégiának megfelelően kell reagálnia, semmiképp sem szabad határozatlan állapotba kerülnie.

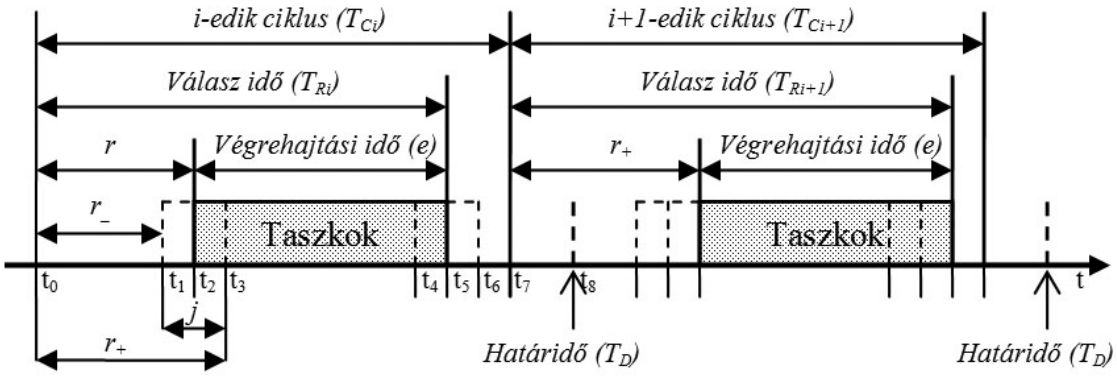
- **lazán valós idejű (soft real-time)** rendszerek, amelyek körében az ilyen mulasztást csak a rendszer működőképességének vagy teljesítményének csökkeneséként fogjuk fel, és erre az esetre is specifikáljuk a rendszer viselkedését.

3.1.3. Egy valós idejű rendszert jellemző fontosabb időtényezők

A valós idejű rendszerek elemzésekor számos időtényezőt kell figyelembe vennünk, azért, hogy optimalizálni tudjuk a működést, azaz minimálisra csökkentsük azokat a tényezőket, amelyek veszélyeztethetik a valósidejű működést. Ezek közül a legfontosabb időintervallumok a következők:

1. előkészületi idő (release time),
2. végrehajtási idő (execution time),
3. válasz idő (response time),
4. ciklus idő (cycle time),
5. remegési intervallum (jitter),
6. határidő (deadline).

Ezeknek az időknek az egymáshoz való viszonyát mutatja a következő, 3.2. ábra.



3.2. ábra. A valós idejű működéssel kapcsolatos időtényezők

Az *előkészületi idő (r)* meglehetősen tág fogalom. Vonatkozhat valamilyen hardver eszközre, amikor azt az időtartamot jelenti, amíg az eszköz elérhetővé vagy használhatóvá válik. De vonatkozhat valamilyen feldolgozásra szánt adatra is, amelyet valamelyik buszról kell levenni. A *végrehajtási idő (e)*, egy adott feladat teljes feldolgozásához szükséges idő. A *válaszidő* az eseményt (feladatot) kiváltó időpillanattól a beavatkozás megkezdéséig eltelt időt jeleni. A *ciklusidő (T_C)* a ciklikus működésű rendszerek legfontosabb jellemzője. Azt az időintervallumot jelenti, amelynek eltelte után a taszkok végrehajtása ismétlődik. A *jitter (j)* rendszerint azt az időintervallumot jelenti, amelyen belül várható valamely fentebb felsorolt tényező kezdési időpillanata. Sok esetben nem tudjuk pontosan meghatározni, vagy bizonyos okok miatt nem mindig azonos például az előkészületi idő esetében. Azt viszont meghatározhatjuk, hogy egy [r_-, r_+] intervallumon belül legyen. Maximális értéke:

$$j_{max} = r_+ - r_- \tag{3.1}$$

A jitternek főleg a szinkronizált időkapcsolatok esetében van jelentősége.

A *határidő (T_D)* meghatározza azt a maximális időtartamot, amely alatt a rendszernek be kell fejeznie a feladatok végrehajtását (3.2. ábra). Ezt rendszerint a külső, technológiai követelmények határozzák meg [14].

A fentebb felsorolt időtényezők korántsem mondhatóak állandónak. Bármelyik változhat ezek közül, akár egyszerre is, de a lényeg az, hogy ne lépje túl a határidőt. Ciklikus működésű rendszereknél a bemeneti értékek beolvasása, és a kimeneti csatornák írása is a ciklikusan történik. A legtöbb esetben ezek az események részei az alapciklusnak. Logikusnak tűnik, hogy a bemeneti változók beolvasása a ciklus első fázisában, még a taszkok elindítása előtt megtörténjen, majd a ciklus végén a válaszok megjelenjenek a kimeneteken. Ez a legtöbb rendszernél így is van, de létezik olyan megoldás is, amikor mindkét esemény a ciklus elején megtörténik, sőt ezen belül, a kimenetek írása megelőzi a bemenetek beolvasását. Erre a problémára egy későbbi fejezetben még visszatérek. (4.1. fejezet)

Az imént említett perifériakezelési stratégia miatt előfordulhat, hogy valamely bementi gerjesztés lekési a bemeneti aktualizálási fázist, így csak a következő ciklusban kerül kiszolgálásra. (Feltételezzük, hogy a gerjesztés megfelelően hosszú időtartamú.) Ezért a legkedvezőtlenebb esethez (worst case) hozzárendelt maximális válaszidő (T_{Rmax}), akár a ciklusidő néhányszorosa is lehet. Ennek következtében a válaszidők mindig nagyobbak, legfeljebb egyenlők lehetnek a ciklusidővel, de mindig a technológiai határidőn belül kell megvalósuljanak. Következik, hogy:

$$n \cdot T_C \le T_{R\,max} \le T_D ; \quad n = 1, 2, \ldots \tag{3.2}$$

3.1.4. Az ipari valós idejű rendszerekkel szemben támasztott egyéb követelmények

Az ipari környezetben alkalmazott valós idejű rendszereknél az is meghatározó szempont lehet, hogy milyen módon tudja kezelni az esetleges időkorlátokra vonatkozó hibákat. Az semmiképp sem megengedett, hogy egy kritikusnak számító időhiba miatt a rendszer csak egy hibaüzenettel reagáljon, és várakozó állapotba álljon, amíg valamilyen operátori beavatkozás nem történik. A rendszertől elvárt viselkedés általában az, hogy próbálja meg bizonyos mértékig csökkenteni a hiba (például határidő elmulasztása) következményeit, és később automatikusan térjen vissza a normál üzemállapotba, vagy ha ez sikertelen, akkor minimum elvárás, hogy legalább a biztonsági funkciókat helyezze működésbe.

Az ilyen rendszerekkel szemben tehát *fokozott biztonsági és megbízhatósági követelményeket* támasztanak. A fokozott biztonság érdekében a rendszer akkor sem adhat ki olyan kimenőjeleket, amelyek a környezetben veszélyes helyzetet idéznek elő, ha meghibásodás miatt nem tudja folytatni működését. A fokozott megbízhatóság elérésének érdekében a rendszereket nemcsak a biztonsági funkciókkal, hanem a *hibatűrés* képességével is fel kell ruházni.

Ugyancsak a biztonsággal és a megbízhatósággal kapcsolatos követelmény a *robosztusság*. Egy robosztus rendszert különlegesen kedvezőtlen, szélsőséges környezeti hatások sem tehetnek tönkre. Átmenetileg leállhat, de a kedvezőtlen hatások elmúltával tovább kell működnie. Egy ilyen a rendszer akkor sem kerülhet határozatlan állapotba, ha nem várt, nem specifikált vezérlőjeleket kap, vagy a program végrehajtása valamilyen meghibásodás, tápkimaradás miatt megszakad, majd újraindul. Az esetek jelentős részében még kezelői segítségre sem lehet számítani, hanem *automatikus újraindítást (watchdog),* esetleg *alapkonfiguráció* betöltést kell megvalósítania.

A valós idejű vezérlési rendszerek gyakran hónapokig, sőt évekig nem állíthatók le, vagyis gyakorta *folyamatos működésűek*. Például egy folyamatosan működő technológiát

irányító rendszerénél üzemszerű működés mellett lehet, hogy egy évben csak egyetlen egy tervezett karbantartási leállás engedhető meg, amelynek időtartama néhány nap, legfeljebb egy hét lehet. Minden egyéb leállás hatalmas anyagi veszteséget vagy biztonsági kockázatot idézhet elő.

Ugyancsak gyakori, hogy egyes rendszerek vagy a nagyobb rendszerek alrendszerei, hosszú időn át felügyelet nélkül működnek. Ilyen *felügyelet nélküli működést* kell biztosítani például a terepre kihelyezett irányító, adatgyűjtő állomásokra, vagy például a műholdak, űrszondák esetében, ahol bármilyen helyi kezelői beavatkozás, diagnosztika, javítás, újraindítás nem lehetséges.

Az eddig felsorolt tulajdonságok miatt a valós idejű rendszerek szoftverei sokkal nagyobb része foglalkozik a talán soha be nem következő, kivételek kezelésével, mint egyéb rendszerekben. Ez még nehezebbé teszi az időkövetelmények teljesítését.

A valós idejű rendszerek feladatainak megfogalmazása leggyakrabban a rendszer viselkedésének leírásával történik. A leírás azt tartalmazza, hogy egy-egy funkció végrehajtása közben milyen üzenetváltások történnek a környezeti szereplők és a rendszer között, és ezek hatására milyen műveleteket kell végrehajtani. Az irányítás tárgyát képező aktív környezeti folyamatok általában egymástól függetlenül, időbeli korlátozások nélkül kezdeményezhetnek beavatkozásokat. A rendszer ennek következtében egyidejűleg több környezeti szereplővel folytathat párbeszédet úgy, hogy a különböző szereplőkkel váltott üzenetek egymáshoz viszonyított sorrendje előre nem határozható meg [27].

3.2. Valós idejű alaprendszer

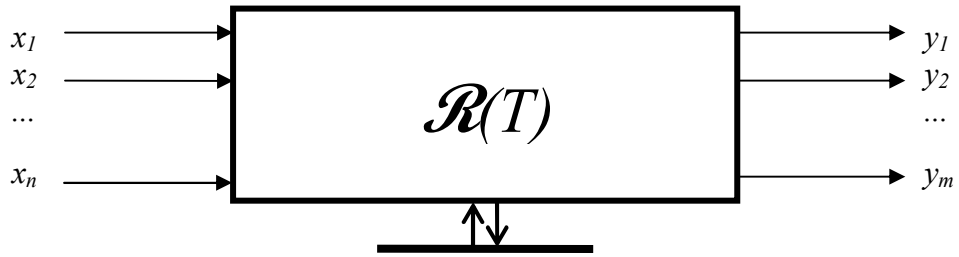
A valós idejű Ethernet rendszerek elemzésénél célszerű kiindulni egy olyan alaprendszerből, amely minden szempontból eleget tesz a valós idejű alapkövetelményeknek. Első sorban ciklikus működésű rendszert feltételezünk, amelyben a feladatok (taszkok) futása ütemezhető, bizonyos feladatok pedig akár párhuzamosan is végezhetőek. A valós idejű rendszer időbeli viselkedésének leírásában a leggyakoribb időhöz kötött feladatok a következők:

- *határidős feladat:* adott időn belül végrehajtandó,
- *periodikus feladat:* adott időközönként ismételten végrehajtandó, (legtöbbször határidős ez is)
- *prioritással kitüntetett feladat:* a prioritási szinttől függően akár megszakítást is kezdeményezhetnek a határidős feladatokon belül is (pl. alarmok),
- *időzített feladat:* bizonyos megengedett eltéréssel egy adott időpontban végrehajtandó,
- *időkorlátos várakozás:* a rendszer külső esemény bekövetkezésekor végez el egy feladatot, de ha a külső esemény nem következik be adott időpont eléréséig, más feladatot kell végrehajtani.

3.2.1. Alapvető tényező a ciklusidő

Ahhoz, hogy az előbbiekben felsorolt idő-specifikus feladatokat a rendszer maradéktalanul képes legyen megvalósítani, alapműködése is valamilyen formában időhöz kötött kell legyen. Itt rendelhető hozzá a már az előzőekben ismertetett *ciklusidő*, amely egy alaprendszer esetében nem más, mint a felhasználói program és a hozzá kapcsolódó rendszerfeladatok ismétlési gyakoriságának ideje [S13]. Az említett feladatok közül, a periodikus és határidős feladatok végrehajtása a legkritikusabb, ezért e két szempont szerint fogom meghatározni az alaprendszer kritériumait. Feltételezzük a következő ábra szerinti alaprendszert, ahol $x_1, x_2, \ldots, x_n$ a rendszer bemeneti, $y_1, y_2, \ldots, y_m$, pedig a

kimeneti, véges számú változói, $\mathcal{R}(T)$ a közöttük fennálló relációk halmaza és a rendszer rendelkezik Ethernet csatlakozással.



3.3. ábra. Valós idejű alaprendszer

<u>Megjegyzés:</u> Az esetek zömében $n \geq m$, mert egy-egy kimeneti eseményt rendszerint több bemeneti feltétel teljesülése határoz meg.

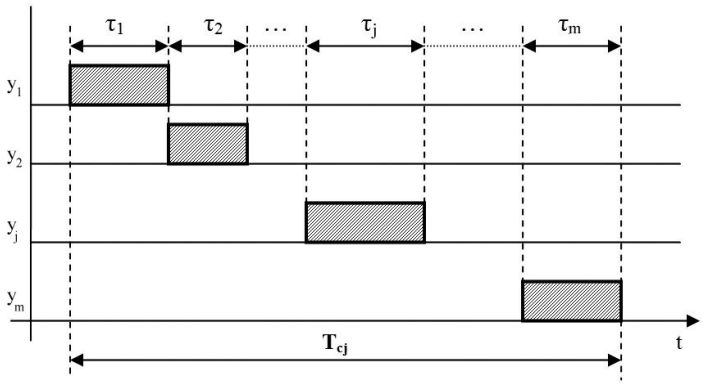
A kimeneti eseményeket meghatározó relációk ($\mathcal{R}_j$, $j = 1,2,...,m$) a bemeneti események közötti logikai kapcsolatoktól (f_j), az általuk kiváltott folyamatoktól, valamint az ezeket meghatározó időktől függenek. Általános formában felírhatjuk a következőket :

$$
\begin{aligned}
&y_1 = \mathcal{R}_1[f_1(x_1,x_2,...x_i,...,x_n),\ T_{R1},\ D_1] \\
&y_2 = \mathcal{R}_2[f_2(x_1,x_2,...x_i,...,x_n),\ T_{R2},\ D_2,] \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&y_j = \mathcal{R}_j[f_j(x_1,x_2,...,x_i,...,x_n),\ T_{Rj},\ D_j]; \quad (i = 1,2,...,n;\ j = 1,2,...,m) \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&y_m = \mathcal{R}_m[f_m(x_1,x_2,...x_i,...,,x_n),\ T_{Rm},\ D_m]
\end{aligned}
\tag{3.3}
$$

ahol T_{Rj} a rendszer adott kimeneti eseményéhez tartozó válaszidejét jelenti, D_j pedig a hozzárendelt relatív határidőt, amely nem szabad túllépje a rendszer specifikációjában meghatározott T_D értéket [S13]

$$\forall D_j \leq T_D \tag{3.4}$$

A válaszidők eseményenként változhatnak, mert függnek a rendszerbemenetekhez hozzárendelt f_j függvények által elindított taszk futási idejétől (τ_j). Ha feltételezzük, hogy egy ciklusidőn belül mindegyik kimeneti esemény megvalósul, akkor ezeknek az összege végső soron meghatározza az aktuális eseményekhez tartozó T_{Cj} ciklusidőt is (3.4. ábra).

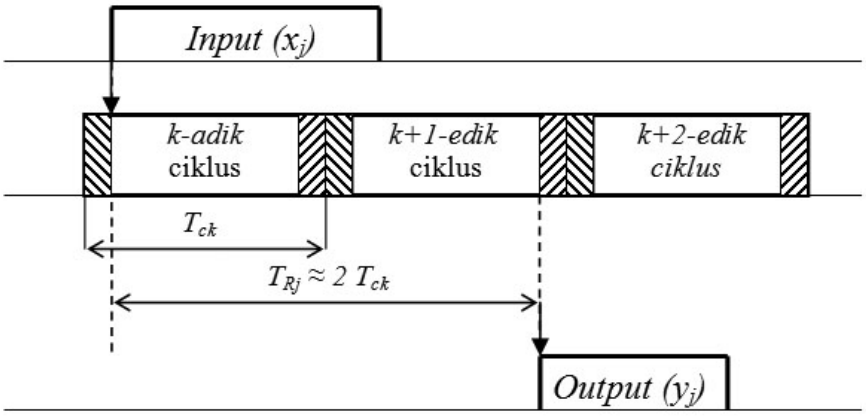


3.4. ábra. A folyamatok (taszkok) futási ideje

Ha alkalmazzuk az előző fejezetben kapott (3.2) összefüggést a válaszidőket illetően és feltételezzük, hogy a ciklusidőt jó megközelítéssel *m* számú taszk végrehajtási ideje (τ_j) határozza meg, akkor felírhatjuk a következő összefüggést:

$$T_{Rj} = q \cdot T_{Cj} = q \cdot \sum_{j=1}^{m} \tau_j \le T_{C\,max}; \text{és} \forall\, \tau_j \le D_j \tag{3.5}$$

A (3.5) összefüggésben $1 \le q \le 2$, valós szám, nevezzük *bemenet-gerjesztési tényezőnek*, a véletlen időpillanatban érkező bemeneti gerjesztés és a ciklusidőn belüli bementi változók aktualizálási időpontjának viszonyát mutatja [S13]. Ha a ciklustmegelőzően a rendszerhez nem érkezik egyetlenegy kérés sem vagy egyetlen egy bemenete sincs, akkor $q = 1$, mert a ciklus alatt lefutó taszkok várhatóan még az adott ciklusban eredményt produkálnak. A legkedvezőtlenebb esetben (3.5. ábra.), amikor a bemeneti gerjesztés éppen lekési az adott ciklus bemeneti aktualizálási idejét, akkor az erre adott válasz csak a következő ciklusban jön létre, vagyis $q = 2$.



3.5. ábra. A válaszidő alakulása a legkedvezőtlenebb helyzetben

Az előbbi megállapításokat és a (3.4) és (3.5) összefüggéseket figyelembe véve kijelenthetjük, hogy *a ciklikus működésű alaprendszer valósidejű, ha a legkedvezőtlenebb eset ciklusideje (a maximális ciklusidő) a határidő felénél nem nagyobb:*

$$T_{C\,max} \le \frac{T_D}{q} = \frac{T_D}{2}; \tag{3.6}$$

ahol:

$$T_{Cmax} = max(T_{C1}, T_{C2}, \ldots, T_{Ck}, \ldots), \tag{3.7}$$

T_D pedig a rendszer specifikációjában megadott időkorlát (határidő).
<u>*Következmény:*</u> A (3.5) és (3.7) összefüggésekből egyértelműen következik, hogy:

$$\forall T_{Ck} \le \frac{T_D}{2}; \quad (j = 1, 2, \ldots, m), \tag{3.8}$$

illetve:

$$\forall\, T_{Rj} \le T_D \tag{3.9}$$

vagyis, *az alaprendszer valósidejűnek tekinthető, ha bármely bemeneti gerjesztésre adott válaszidő sohasem haladja meg az előírt határidőt.*

3.2.2. A legkedvezőtlenebb eset ciklusidejének meghatározása

A (3.5) összefüggés szerint a ciklusidő a taszkok számától és azok időtartamától függ. Szigorúan valósidejű rendszereknél a taszkok időtartamához is hozzárendelhetünk relatív időkorlátokat (D_j). Ebben az esetben viszont ahhoz, hogy a rendszer megtartsa szigorúan valósidejű jellegét, az egyes taszkok futási ideje sem haladhatja meg a hozzájuk tartozó időkorlátot:

$$\forall\, \tau_j \le D_j\,; \quad (j = 1,2,\ldots,m) \tag{3.10}$$

A taszkok sorrendjét, a legtöbb esetben, az őket kiváltó bemeneti események érkezési sorrendje határozza meg. Egy esemény bekövetkezésének elmaradása esetén, a taszk futása az adott ciklusban akár el is maradhat. Előfordulhatnak még periodikusan futó határidős taszkok vagy kitüntetett taszkok (pl. hibaesemények), amelyeket célszerű prioritással ellátni, tekintettel arra az esetre, amikor azonos időben több kérés is érkezik a rendszer bemeneteire. A prioritásspecifikáció nélküli, azonos időpillanatban kiváltott taszkokat a rendszer véletlenszerű sorrendben hajtja végre. Ezeket a taszkokat bármikor megszakíthatják magasabb prioritással rendelkezőek.

Az előbb elmondottak alapján meghatározható a rendszer maximális ciklusideje, ha feltételezzük, hogy az adott ciklusban mindegyik taszk lefut és mindegyikhez a legkedvezőtlenebb eset időtartamát (C_j) soroljuk. Ha ehhez az esethez rendeljük hozzá a taszkok határidejét (D_j), felírhatjuk a következő összefüggést:

$$T_{C\,max} = \sum_{j=1}^{m} C_j \le \sum_{j=1}^{m} D_j \tag{3.11}$$

Figyelembe véve az (3.6) szerinti, valós idejű alaprendszerre vonatkozó megállapítást, vagyis azt, hogy a legkedvezőtlenebb gerjesztési esethez még maximális ciklusidő is társul, akkor:

$$T_{C\,max} \le \frac{T_D}{2} \tag{3.12}$$

A (3.10) és (3.12) összefüggések alapján teljes bizonyossággal kijelenthetjük, hogy *a ciklikusan működő alaprendszer valósidejű, ha a határidős taszkok futási ideje a határidőn belül lezajlik, a maximális ciklusidő pedig a rendszer specifikációjában megadott időkorlát felénél nem nagyobb.*

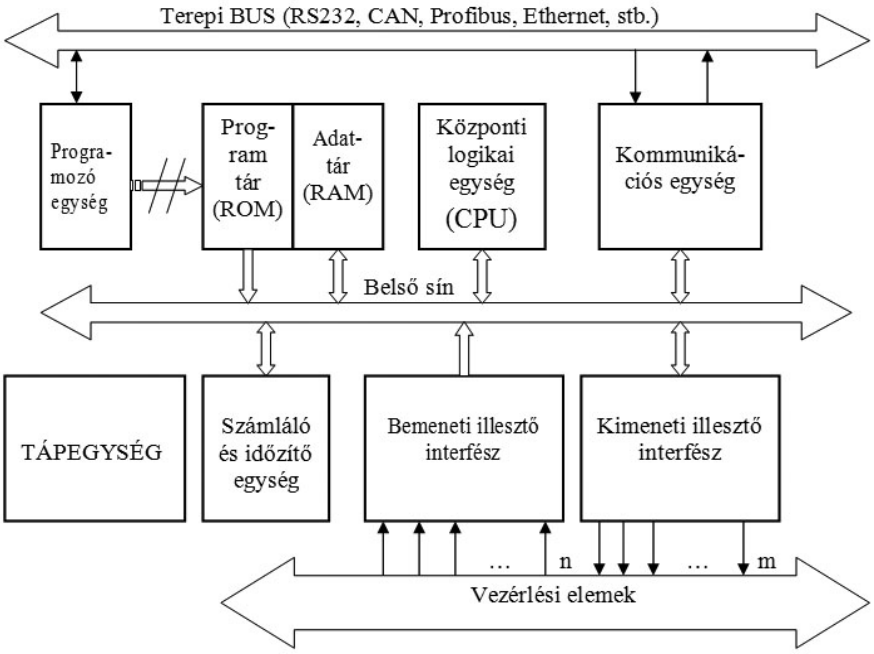
3.2.3. A programozható logikai vezérlő (PLC) mint valós idejű alaprendszer

A programozható logikai vezérlők gondolata akkor merült fel, amikor a hatvanas években a piacvezető nagy amerikai autógyárak, azzal szembesültek, hogy már nem csak az európai, hanem az amerikai piacon is kezdenek megjelenni az egyre inkább vevői igényeket kielégítő, kisebb teljesítményű, de sokkal hatékonyabban működő japán gépkocsik. Huzalozott vezérlők használatával a technológia rugalmassá tétele meglehetősen nehéz feladat. A gyártó- és szerelősorok átállítása valamint a vezérlő rendszerek módosítása gyakran több napot is igénybe vett. Ezért az akkori piacvezető amerikai autógyártó, a General Motors 1968-ban pályázatot írt ki olyan megoldásra, amely képes kiváltani a relés vezérléseket, könnyen programozható és megbízhatóan működik. A

pályázatot a Modicon cég nyerte el, mérnökei pedig nyomban hozzá is kezdtek a munkához.

Az első PLC Richard Morleyhoz és csapatához fűződik, akik „Modicon 084” néven 1969-ben készítették el. Morley túlzásnak tartotta, hogy computernek hívják, így inkább a controller elnevezést javasolta. Mivel elsősorban logikai műveletek végzésére szánták, és ami a lényeg, hogy programozható volt, ezért a máig is ismert elnevezést tulajdonították neki. Központi egységét huzalozott CPU alkotta, mindösszesen 1 kB memóriával rendelkezett, viszont 128 bemeneti/kimeneti csatornát tudott kezelni [62].

Az elmúlt több mint 40 év alatt a PLC-k sokat változtak. Központi logikai egységüket rendszerint RISC típusú mikroprocesszor (akár több processzor) alkotja azért, hogy a valós idejű működés minél inkább támogatott legyen. A kezelhető memóriaterület is jóval nagyobb, bár nem ez a meghatározó, ha minősíteni kell egy PLC-t. A bemeneti és kimeneti csatornák száma pedig moduláris felépítésüknek köszönhetően, akár több ezer is lehet. A korszerű, moduláris felépítésű PLC-k, a hagyományos kommunikációs kapcsolatokon (RS232, USB, CAN, Profibus, stb.) kívül, Ethernet interfésszel is rendelkeznek, vagy Ethernet modul illeszthető hozzá. (3.6. ábra)

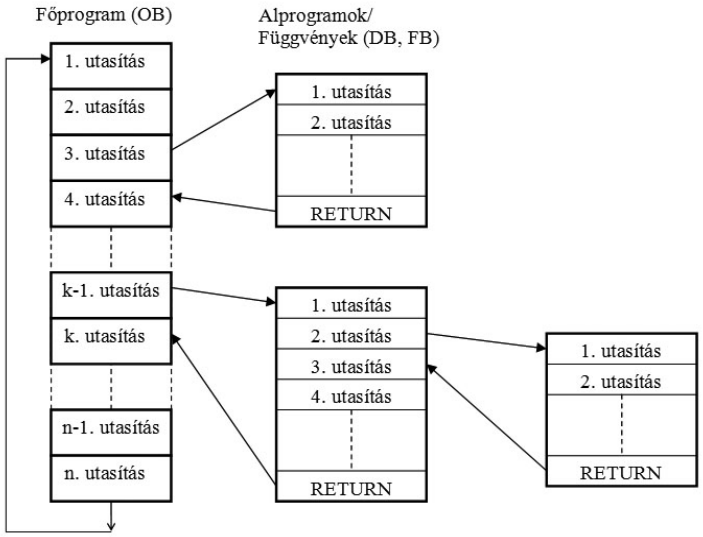


3.6. ábra. Egy PLC általános felépítése

Azok a programozható logikai vezérlők, amelyek rendelkeznek Ethernet csatlakozási lehetőséggel, felépítésüket illetően belátható, hogy eleget tesznek a valós idejű alaprendszer szerkezetének, mert vannak bemeneti/kimeneti csatornái, amelyeken keresztül kapcsolatot tart a vezérléssel. Most lássuk, hogy programvégrehajtás szempontjából teljesítik-e a (3.10) és (3.12) kritériumokat?

A PLC rendszerprogramja ciklikus működést biztosít a felhasználói program számára. Indítás vagy RESET után a rendszerprogram beolvassa a bemeneti változókat és elindítja a főprogram utasításainak szekvenciális végrehajtását. A legtöbb felhasználói program strukturált szervezésű, vagyis a főprogramból szubrutinok (taszkok) vagy függvények hívhatóak. Ezek hívása sok esetben eseményvezérelten történik, de lehetnek periodikusan ismétlődő idővezérelt, vagy prioritással ellátott taszkok is. Az utolsó utasítás után a rendszerprogram az eredményeknek megfelelően aktualizálja a kimeneti csatornákat és visszatér a főprogram elejére (3.7. ábra). Ezt az egyszeri program-végrehajtási időt

nevezzük a PLC ciklusidejének vagy letapogatási időnek (scan time), amely mindenben azonosnak tekinthető a 3.1.3. alfejezetben meghatározottal.



3.7. ábra. A felhasználói program szerkezete

Látható, hogy a ciklusidő az utasítások számától, az utasítások bonyolultságától és természetesen a processzor művelet-végrehajtási sebességétől függ. Nem túlságosan bonyolult vezérlési programoknál ez néhány milliszekundumtól, 10-50, legfeljebb 100ms-ig terjed.

A bementi gerjesztésre adott válaszidők jó esetben egy, de maximum két ciklusidőt igényelhetnek. Ennek megfelelően a határidőt minden esetben tartani tudják, ha a ciklusidőt sikerül a határidő felénél nem nagyobb értéken tartani. Egyes PLC-k ezt képesek ellenőrizni is és „hibaüzenettel" reagálnak ennek túllépésekor. Egy ilyen hibaüzenet természetesen nem a PLC leállását jelenti. Ilyenkor meghív egy olyan taszkot, amely tartalmazza, hogy ilyen esetben mit kell, hogy tegyen a PLC. Ennek ismeretében a továbbiakban a ciklikus működésű, Ethernet hálózati kapcsolattal rendelkező PLC-t valós idejű alaprendszernek tekintem [S17]. Ciklusidejének változása és valós idejű viselkedésének mérésekkel történő igazolása a IV. fejezetben kerül bemutatásra.

3.3. Nem ciklikus működésű valós idejű rendszerek

Az előző fejezetben meghatározott ciklikus működésű, valós idejű alaprendszer legfontosabb tényezője a ciklusidő. A taszkok sorrendje és azok határideje legfeljebb az optimális ciklusidő elérése érdekében lehet érdekes, mert az általuk kiváltott hatás amúgy is a ciklus végén kerül érvényesítésre. A határidő túllépés – megfelelő intézkedéseket követően – rendszerint a taszk megszakítását eredményezi azért, hogy a többi zavartalanul lefuthasson. A taszkok futási idejét, így végső soron a ciklusidőt alapvetően a rendszert alkotó CPU (vagy CPU-k) műveletvégzési sebessége határozza meg. Ezért nem mindegy, hogy a taszkok milyen sorrendben kerülnek feldolgozásra. A cél, hogy a prioritási sorrend betartása mellett, a processzor kihasználtság a legjobb legyen.

A valós idejű rendszerek nem csak ciklikus működésűek lehetnek. Az ilyen rendszerek valós idejű működése sokkal összetettebb feltételek teljesítését kell biztosítsák. Ezek a rendszerek elsősorban arra törekednek, hogy taszkjaik optimálisan ütemezhetőek legyenek adott időintervallumon belül. Több megközelítés is született ebben a témában, főleg egyprocesszoros rendszerekhez. Ezek közül megemlíteném az *RM (Rate Monolitic) ütemezést* és ennek egy továbbfejlesztett formáját az EDF (*Earliest Deadline First)*

algoritmust. Az RM algoritmus elvének kidolgozása Liu és Layland nevéhez fűződik [25]. Az eljárás egyprocesszoros rendszert feltételez, amelyben minden taszk periodikus és a gyakrabban futó taszkok magasabb prioritást élveznek. Az elv a processzor igény és a processzor kihasználtság közötti relációkon alapul [24]. A működés preemptív, vagyis a magasabb prioritással rendelkező taszk megszakíthatja az alacsonyabb prioritásút.

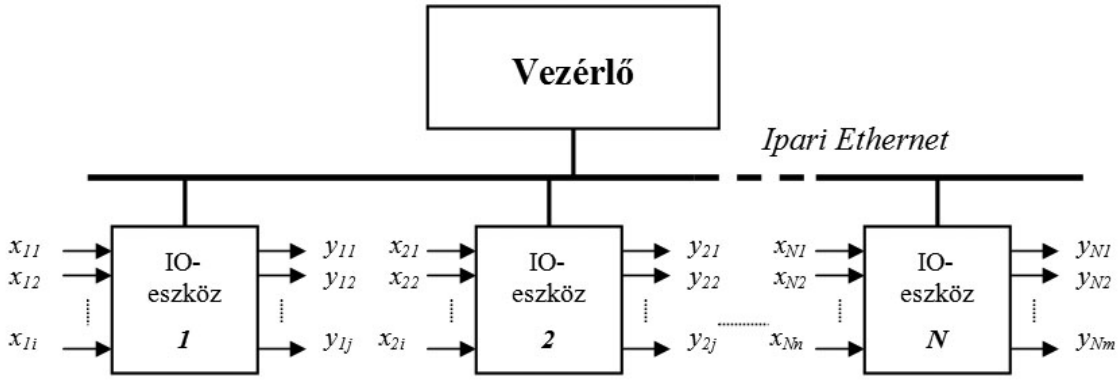
A módszernek egyik hátránya, hogy nem garantálható a teljes processzor-kihasználtság. Másrészt sztatikus jellegű, mivel a taszkok prioritása a periódusuk szerint van meghatározva, és a prioritást illetően egyáltalán nem mondható rugalmasnak. Ráadásul az optimális ütemezés csak akkor biztosítható, ha a taszkok határideje megegyezik a taszkok periódusával.

Az EDF ütemezési algoritmus az előbbinél egy sokkal rugalmasabb, dinamikus megoldást eredményez. A prioritás a taszkok abszolút határidejéhez van kötve, vagyis azé a magasabb prioritás, amely a legkorábban lejáró határidővel rendelkezik. Innen ered az elnevezés is. Dinamikusságának köszönhetően, amennyiben új, végrehajtásra kész folyamat jelentkezik, a prioritási sorrend újragenerálódik és lehetővé teszi, hogy a magasabb prioritású taszkok akár meg is szakíthatják az alacsonyabb prioritásúak futását [26].

3.4. Valós idejű ipari Ethernet alrendszer

Ha a (3.2) fejezetben meghatározott alaprendszert kiterjesztjük az ipari Ethernet hálózatokra, akkor meghatározhatjuk azokat a körülményeket és feltételeket, amelyek mellett, az egyébként nem determinisztikus Ethernet hálózat alkalmassá válik valósidejű feladatokat ellátó egységek közötti kommunikáció lebonyolítására. Ennek megfelelően a valós idejű Ethernet alrendszert úgy tekintjük, mintha az alaprendszer bemeneti és kimenetei eseményeit kezelő csatornák eszközei Ethernet hálózaton keresztül kapcsolódnak a folyamatokat (taszkokat) irányító vezérlőhöz. Feltételezzük, hogy a vezérlő és a perifériák közötti viszonyt valamilyen időhöz kötött, token passing vagy master-slave vagy provider-consumer típusú megoldás biztosítja. Ez utóbbi, olyan szempontból előnyösebb, hogy duplex hálózatban, bármelyik fél kezdeményezhet kommunikációt egymástól függetlenül.

Feltételezzük, hogy az n számú bemeneti (x_i) és az m számú kimeneti (y_j) csatorna N számú Ethernetes periféria eszközön (IO-eszköz) keresztül kapcsolódik a vezérlőhöz. A következő ábrán egy ilyen Ethernet alrendszert láthatunk (3.8. ábra).



3.8. ábra. Valós idejű Ethernet alrendszer

Az előző fejezetben láthattuk, hogy ahhoz, hogy kiszámítható működésűvé tegyük az egyébként nem determinisztikus Ethernet hálózatot, egyik alapvető feltétel az, hogy

valamilyen jól meghatározott időközönként épüljön ki kapcsolat az eszközök és a vezérlő között. Egymástól függetlenek legyenek, rendelkezzenek prioritással a nem valósidejű (NRT) kommunikáció felett, de számottevően ne akadályozzák azokat.

3.4.1 A buszfrissítési ciklus

Egyik módja a ciklikusan működő kapcsolati rendszereknek, a nem Ethernet alapú terepi buszrendszereknél jól bevált master-slave megoldás, amikor a master a konfigurációjában meghatározott időközönként lekérdezi a slave eszközök bemeneteit, illetve adatokat küld ezeknek a kimeneteire. Egy cikluson belül akár az összes eszköz lekérdezésre kerülhet adott sorrendben, jól meghatározott időrések szerint. Ha a sorrend nem változik, akkor gyakorlatilag mindegyik eszköznek azonos ciklusidő szerint van lehetősége adatot cserélni a masterrel. A módszer igazából akkor alkalmazható a leghatékonyabban, amikor az eszközök hasonló paraméterekkel rendelkező céleszközök működtetését végzik [53].

A másik megoldás, amikor az eszközök és a vezérlő között egymástól független kommunikációs kapcsolatok épülnek fel (provider-consumer), vagyis mindegyik eszközhöz hozzárendelhető saját, egyéni buszfrissítési ciklusideje [S12]. Ez főleg akkor előnyös, ha az eszközökhöz kapcsolódó rendszerek más-más időspecifikációt igényelnek. Egy lassú folyamatot felesleges olyan ütemben lekérdezni, mint egy nagyon gyors folyamatot. Ezáltal lehetőség nyílik az adott sávszélesség jobb kihasználására, ami lehetővé teszi, hogy nagyszámú IO-eszköz esetén is a valósidejű elvárásoknak megfelelően működjön a rendszer. Természetesen ez a megoldás is lehetőséget biztosít arra, hogy azonos funkcionalitású céleszközök kiszolgálását biztosítsa nagy precizitással. Ennek érdekében, az egyébként sajátságos egyéni ciklusidőket azonossá kell tegyük és szinkronizálni kell, hogy minél precízebb beavatkozásokat biztosíthassunk a célrendszerek számára, ott ahol ezek megkövetelik.

Bármelyik megoldást is feltételezzük a busz ciklus meghatározó tényezője lesz a valósidejű Ethernet alrendszernek. A szigorúan valósidejű rendszereknél ez a ciklusidő megfelelően rövid és pontos kell, hogy legyen. Sok esetben nem is az a probléma, hogy viszonylag nagy a kimenetek reakcióideje, hanem az, hogy ezek minél pontosabbak legyenek, mert akkor mindig ugyanazzal a késéssel tudunk kalkulálni.

Felhasználva a (3.3) összefüggését és figyelembe véve az egyes eszközök busz ciklusidejét, a 3.8. ábra szerinti rendszer esetében bármelyik kimenetre felírhatjuk:

$$y_{kj} = \mathcal{R}_{kj}[f_{kj}(x_{11},\ldots,x_{ki},\ldots,x_{Nn});\mathcal{T}_k, T_C, T_{Dk}];\ (i = 1,2,\ldots,n;\ j = 1,2,\ldots,m;\ k = 1,2,\ldots,N); \quad (3.13)$$

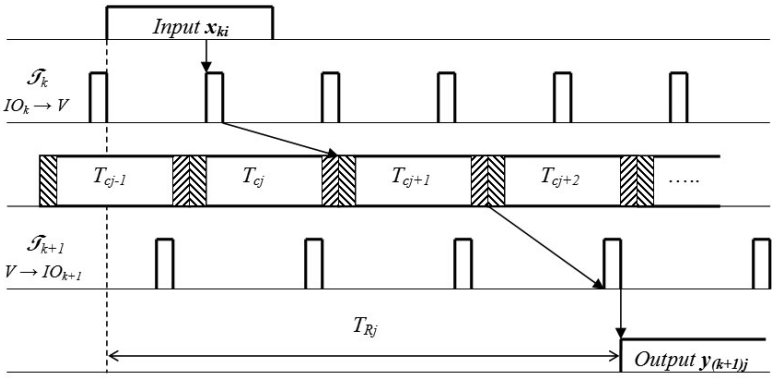
ahol $\mathcal{T}_k$ a k-adik eszköz buszfrissítési ciklusideje, T_C a vezérlő ciklusideje, T_{Dk} pedig a k-adik eseményhez tartozó relatív határidő.

Tételezzük fel, hogy az $y_{(k+1)j}$ eseményt az x_{ki} gerjesztés váltja ki, az eszközökhöz hozzárendelt buszfrissítési ciklusidők pedig különbözőek ($\mathcal{T}_k \neq \mathcal{T}_{k+1}$). A legkedvezőtlenebb eset (3.9. ábra) reakcióidejének meghatározására a következő relációt írhatjuk fel:

$$T_{Rj} = 2 \cdot T_C + \mathcal{T}_k + \mathcal{T}_{k+1}; \quad (3.14)$$

A (3.14) reláció egy általános helyzet reakcióidejét határozza meg, amikor a kommunikációs relációban résztvevő eszközök buszfrissítési ideje eltér egymástól, a vezérlő program-végrehajtási ciklusideje pedig tetszőleges. Láthatjuk, hogy a vezérlő ciklusideje súlyozottan befolyásolja a válaszidőket. Ezért nem is igazán érdemes a buszfrissítési ciklusokat sokkal kisebbre választani, mint a vezérlő ciklusideje, mert, ha

lassúbb a feldolgozás, a gerjesztések felülíródhatnak, a kimenetek pedig nem frissülnek a vevő buszciklusa szerint. Mivel a vezérlő ciklusidejét a taszkok futási idejének összege határozza meg, azaz adott, ennek megfelelően: $\mathcal{T}_{kmin} \geq T_C$. Vagyis az alrendszer minimális buszfrissítési idejét a vezérlő ciklusidejéhez kell igazítani (3.9. ábra).



3.9. ábra. A legkedvezőtlenebb helyzet reakcióideje

Egy másik, ugyancsak szélsőséges esetnek tekinthető az a szituáció, amikor a buszfrissítési idők jóval nagyobbak, mint a vezérlő ciklusideje. Ilyenkor egyértelmű, hogy a válaszidőket csak a buszfrissítési idők nagysága határozza meg, vagyis függetlenek a vezérlő ciklusidejétől.

3.4.2. A valós idejű Ethernet alrendszer meghatározása

Ahhoz, hogy az ipari Ethernet alrendszert valósidejűnek tekinthessük, teljesítenie kell a valósidejű alaprendszer (3.12) szerinti feltételeit, vagyis:

$$T_{Cmax} \leq T_{Dk}/q \tag{3.15}$$

Ahol a (3.11) alapján feltételezzük, hogy:

$$T_{Cmax} = max(T_C, \mathcal{T}_k, \mathcal{T}_{k+1}) = \mathcal{T}_k \tag{3.16}$$

A (3.14) összefüggést alkalmazva a legkedvezőtlenebb esetre, amikor szinkronizálás nélküli a kommunikáció, továbbá a gerjesztés olyan időpillanatban történik, hogy mindkét irányban éppen lekési a buszfrissítési időt és a bemeneti aktualizálást is egyaránt, következik, hogy ebben az esetben a gerjesztésre adott válasz reakcióideje kb. négyszerese lesz a legnagyobb busz ciklusidőnek. (3.9. ábra). Ebben az esetben tehát: $q = 4$. Következik, hogy:

$$\forall\ T_{Rj} = q\ T_{Cmax} = 4\ \mathcal{T}_k \leq T_{Dk} \tag{3.17}$$

Kijelenthetjük, hogy *az Ethernet alrendszer valósidejű, hogyha a legnagyobb ciklusidővel rendelkező eszköz buszfrissítési periódusa legalább négyszer kisebb, mint a rendszer specifikációjában hozzárendelt időkorlát.*

Megjegyzés: Mivel $q = 4$ helyzet bekövetkezési valószínűsége igen csekély, a gyakorlatban, a legtöbb esetben elegendő ha $q = 3$, de természetesen lehetőség van ettől eltérő értékek kiválasztására is. A valóságban ez azt jelenti, hogy ha három buszfrissítési idő elteltével sem érkezik válasz a kérésre, akkor gyanítható, hogy valamilyen

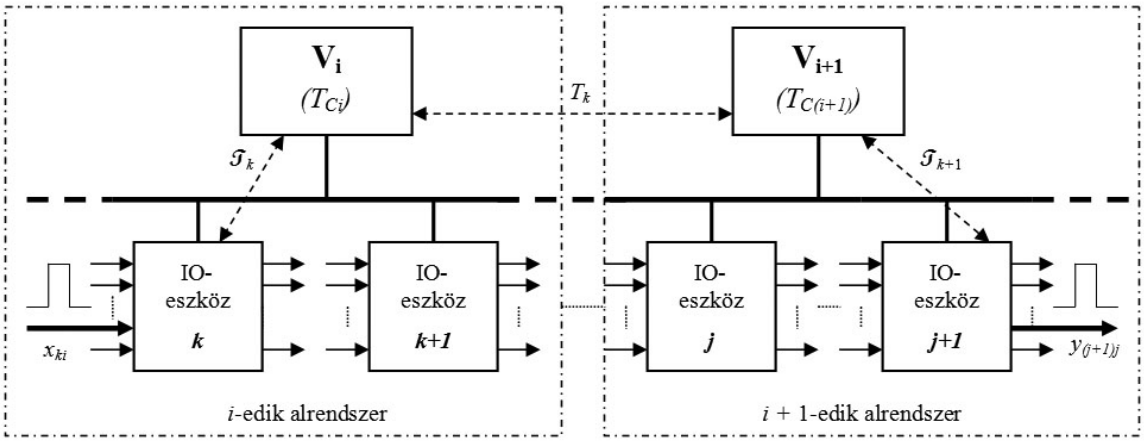
hibaesemény történt, és a rendszer ennek megfelelően kell reagáljon. Az *n, m, N* mennyeiségekre vonatkozó korlátok tágabb értelmezés szerint a vezérlő kapacitásától függnek, de szigorúbb valósidejű feltételek mellett természetesen a rendszerspecifikációban szereplő T_D időkorlát is meghatározó lehet, főleg az eszközök számát illetően.

A (3.17) alapján elmondható, hogy a feltételezett ipari Ethernet alrendszer valósidejű egységes egészként működő valósidejű alaprendszernek tekinthető annak ellenére, hogy a bemeneti és kimeneti változók más és más, egymással Ethernetes kapcsolatban álló IO eszközhöz tartoznak. A valósidejű feltételek teljesülését egyértelműen a buszfrissítési ciklus helyes megválasztása és megfelelő értéken tartása határozza meg. Ahogy szigorítjuk az alrendszer valósidejű követelményeit, úgy egyre szigorúbb feltételeket kell szabnunk a ciklusidőket illetően is. Szigorúan valósidejű rendszereknél arra kell törekednünk, hogy a működés minél kiszámíthatóbb legyen, azaz a 3.9. ábra szerinti legkedvezőtlenebb helyzeteket elkerüljük. Ezt úgy tudjuk elérni, hogy szinkronizáljuk az eszközök ciklusidejét. Ezáltal egyrészt az eszközök ciklusidő periódusát egyenlővé tesszük, másrészt igyekszünk a jittert (ciklusidő eltérés) minél kisebbé tenni [S7].

3.5. A valósidejű Ethernet alrendszer kiterjesztése

Ha kettő vagy több, egymástól független funkcionalitású valósidejű Ethernet alrendszert a hálózaton keresztül összekapcsolunk egymással, egységes egészként működő valósidejű Ethernet rendszert kapunk, amely teljesíteni fogja a (3.9), (3.10) összefüggésekben meghatározott követelményeket a megfelelő feltételek mellett.

A következő ábrán (3.10. ábra) két tetszőlegesen választott alrendszer kapcsolatát látjuk. Feltételezzük, hogy az *i*-edik alrendszer kommunikációs relációt épít a *i+1*-edik alrendszerrel. Továbbá a *k*-adik IO eszköz valamelyik bemenete gerjeszti azt az eseményt, amelyik a *j+1*-edik eszköz kimenetén kell megvalósuljon.



3.10. ábra. Két tetszőleges alrendszer kommunikációs kapcsolata.

Legyen T_{Ci} illetve $T_{C(i+1)}$ a két vezérlő ciklusideje, $\mathcal{T}_k$ valamint $\mathcal{T}_{k+1}$ az érintett IO eszközök és T_k a két vezérlő közötti buszfrissítési idők. A (3.14) relációt kiterjesztve az így létrejövő rendszerre, következik:

$$T_{Rj} = 2 \cdot (T_C + T_{C(i+1)}) + T_k + \mathcal{T}_k + \mathcal{T}_{k+1}; \qquad (3.18)$$

Feltételezzük, hogy a (3.16) relációnak ugyancsak $\mathcal{T}_k$ tesz eleget, tulajdonképpen a (3.17) összefüggést kapjuk $q = 7$-re.

$$\forall\, T_{Rj} = q\; T_{Cmax} = 7\; \mathcal{T}_k \leq T_{Dk} \tag{3.19}$$

Kijelenthető, hogy *az Ethernet hálózaton keresztül egymással összekapcsolt ciklikusan működő alrendszerek valós idejű Ethernet rendszert alkotnak, ha a legnagyobb ciklusidővel rendelkező eszköz buszfrissítési periódusa legalább hétszer kisebb, mint a rendszer specifikációjában meghatározott időkorlát.*

3.6. Új tudományos eredmények

Definiáltam egy Ethernet csatlakozással bíró alaprendszert, amely ciklikus működésű, véges számú IO csatornákkal rendelkezik és maradéktalanul teljesíti a valós idejű feltételeket.

A meghatározott kritériumok alapján bizonyítottam, hogy az Ethernet csatlakozással rendelkező PLC-k teljesítik a valós idejű alaprendszer feltételeit. Ez fontos lehet abból a szempontból, hogy ha PLC-t alkalmazunk rendszervezérlőként, egyedül csak a ciklusidő helyes megválasztását kell biztosítanunk a valós idejű működéshez.

A továbbiakban az alaprendszert kiterjesztettem alrendszerré úgy, hogy Ethernet hálózaton keresztül további IO csatornával rendelkező elemeket csatlakoztattam hozzá. Az alrendszerre vonatkozóan is meghatároztam a valós idejű működési kritériumokat. Két vagy több alrendszer összekaocsolásából kapott rendszer ugyancsak valós idejű rendszert alkot amennyiben teljesíti az erre vonatkozó követelményeket.

***1. TÉZIS** [S4, S9, S17, S18]*

Kidolgoztam egy olyan elméleti modellt, amelynek segítségével meghatározható, hogy a ciklikus működésű ipari Ethernet rendszerek milyen kritériumoknak kell eleget tegyenek, hogy valósidejű működést biztosítsanak.

1.1. Altézis. *A ciklikusan működő alaprendszer valós idejű, ha a határidős taszkok futási az előírt határidőn belül van, a maximális ciklusidő pedig a rendszer specifikációjában megadott technológiai időkorlát felénél nem nagyobb.*

1.2. Altézis. *Az ipari Ethernet alrendszer valósidejű, hogyha a legnagyobb ciklusidővel rendelkező eszköz buszfrissítési periódusa legalább négyszer kisebb, mint a rendszer specifikációjában hozzárendelt időkorlát.*

1.3. Altézis. *Az ipari Ethernet alrendszer egységes egészként működő valós idejű alaprendszernek tekinthető annak ellenére, hogy a bemeneti és kimeneti változók más és más, egymással Ethernetes kapcsolatban álló IO eszközvezérlőhözhöz tartoznak.*

1.4. Altézis. *Két vagy több alaprendszer Etherneten történő összekapcsolásából származó rendszer ugyancsak valós idejű működést biztosít, ha a legnagyobb ciklusidővel rendelkező eszköz buszfrissítési periódusa legalább hétszer kisebb, mint a teljes rendszerre meghatározott időkorlát.*

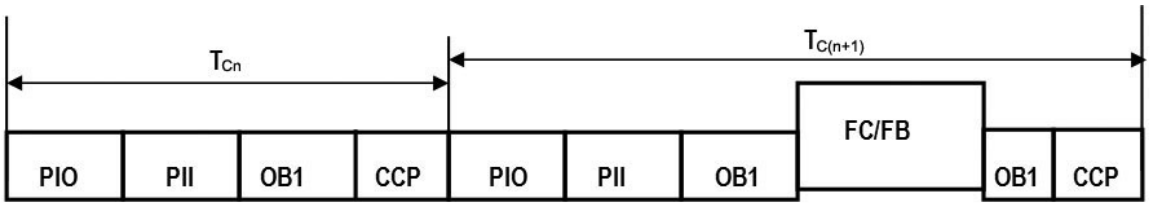
IV. FEJEZET

AZ IO KONTROLLEREK CIKLUSIDŐ-VÁLTOZÁSÁNAK MEGHATÁROZÁSA ÉS MÉRÉSI MÓDSZEREINEK KIDOLGOZÁSA

Az ipari Ethernet rendszerek vezérlői a valós idejű működés biztosítása érdekében az esetek többségében elsősorban ciklikus működésűek. Alapprobléma tehát az ilyen rendszereknél a ciklusidő meghatározása, elemzése és mindazoknak a kritériumoknak a vizsgálata, amelyek befolyásolhatják ezt. A III. fejezetben bizonyítást nyert, hogy a ciklikus működésű, programozható logikai vezérlők valós idejű alaprendszert alkotnak. Ebben a fejezetben pedig mérésekkel igazolom ezt, valamint meghatározásra kerülnek mindazok a tényezők, amelyek alapvetően befolyásolják a ciklusidőt. A ciklusidőt nem csak meghatározni, de sok esetben mérni is szükséges. A PLC-k operációs rendszere (rendszerprogramja) a legtöbb esetben, ugyan ezredmásodperces pontossággal méri a ciklusidőt, sőt némelyiknél konfigurálni is lehet ennek állandó vagy maximális értékét, de ez nem minden esetben elégséges.

4.1. A ciklusidőt meghatározó tényezők

A PLC-k ciklikus működésének felügyelete az operációs rendszer feladatai közé tartozik. Bekapcsolás vagy újraindítás után, a bemeneti és kimeneti folyamati térképek aktualizálását követően az operációs rendszer a programszámlálót a főprogram kezdőcímére állítja. Ezzel megkezdődik a felhasználói program utasításainak feldolgozása. Az utolsó utasítás végrehajtása után, az operációs rendszer a biztonságos működéshez szükséges rendszerfeladatok elvégzése után újrakezdi a folyamati térképek aktualizálását majd a program végrehajtását. A felhasználói program futása közben érkezhetnek megszakítási kérelmek, vagy szekvenciális programszerkezet esetén alprogram-, illetve függvényhívások, amelyek növelhetik a vezérlési program végrehajtási idejét. (4.1. ábra)



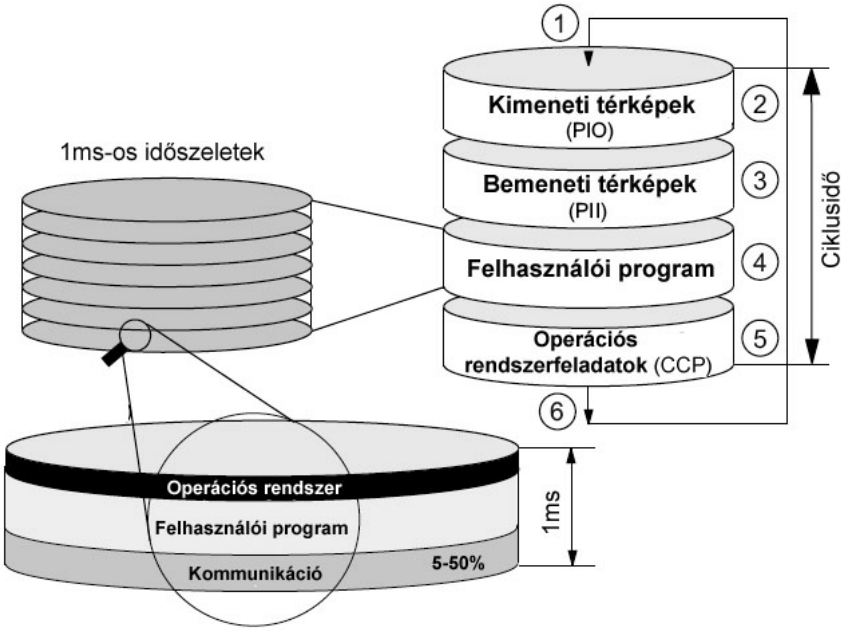
PIO/PII: - Process Image of Output/Input (Kimeneti/bemeneti folyamati térképek aktualizálása)
OB1: - Organization Block (Főprogram vagy szervező modul Siemens)
CCP: - Scan cycle check point (Ciklusellenőrzési pontok vizsgálata)
FC/FB: - Függvény vagy funkció blokk hívása

4.1. ábra. A ciklusidő fontosabb elemei [48]

A *ciklusidő* (scan time vagy cycle time) azt az időintervallumot jelenti, amely szükséges a felhasználói program futásához, a megszakítások kiszolgálásához valamint a rendszerműveletek (pl. folyamati térképek aktualizálása) végrehajtásához.

Látható, hogy a ciklusidő (T_C) megnövekszik azon ciklusokban, amelyekben alprogram (funkció blokk FB) vagy függvényhívás (FC) történik. Ezen kívül függ a program hosszától, az utasítások bonyolultságától a megszakítások számától és a CPU

sebességétől. Általános esetben a ciklusidő tipikusan néhány milliszekundumtól néhány 10ms, különleges esetekben néhány 100ms-ig is terjedhet. Ez meglehetősen hosszú idő ahhoz, hogy közben a CPU ne foglalkozzon egyéb, a működéshez szükséges feladatokkal. Ezt a problémát úgy kezeli a legtöbb PLC mikroprocesszora, hogy felosztja a felhasználói programot 1 ms-os időszeletekre, vagyis 1 ms-ként megszakítja a felhasználói programot. (4.2. ábra). A felhasználói program végrehajtásán kívül az operációs rendszer minden ciklusban végrehajtja a bemeneti és kimeneti folyamati térképek (PII és PIO) aktualizálását, a ciklus végén pedig más adminisztrációs feladatokat, mint például az átmeneti változók törlése (CCP). [48]



4.2. ábra. A felhasználói program időosztása [48]

A 4.2. ábra az 1998 októberétől gyártott Siemens PLC-k ciklusszerkezetét és a felhasználói program időszeleteit mutatja. Az ábra szerint a ciklusidő összesen hat fázisra bontható:

1) Az operációs rendszer inicializálja a ciklusidő kezdetét.
2) A CPU a folyamat kimeneti memóriatérképek (PIO) tartalmát a kimeneti csatornákra másolja.
3) A CPU a bemeneti csatornák állapotát bemásolja a folyamat bemeneti memóriatartományába (PII).
4) A CPU a felhasználói programot időszeletekre bontja és megkezdi az utasítások végrehajtását.
5) A ciklus végén az operációs rendszer elvégzi a szükséges adminisztrációs feladatokat, pl. törli az átmeneti változókat.
6) A CPU ellenőrzi a ciklusidőt, lezárja a ciklust és újraindítja a következőt.

A programszelet összetételét vizsgálva láthatjuk, hogy az operációs rendszerutasítások mellett még kommunikációs feladatokat is le kell vezérelnie. A Siemens ezt *kommunikációs terhelésnek* (Communication load) nevezi. Az erre fordított idő tág határok között (5-50%) konfigurálható. Alapértelmezett értéke 20%. Ettől eltérő értéket beállítani csak indokolt esetben érdemes. Ez tehát egy újabb tényező, amely jelentősen befolyásolhatja a ciklusidőt, és amelyet figyelembe kell vegyünk a maximális ciklusidő kiszámításakor. Kommunikációs terhelés alatt itt többféle lehetőséget értelmeznek. Például kapcsolat valamilyen operátor panellel vagy HMI rendszerrel, monitor üzemmód

alkalmazása diagnosztizáláskor, de idetartoznak az ipari Ethernetes kommunikációval kapcsolatos rendszerműveletek is. A kommunikációs terhelés konfigurációbeli változtatása a ciklusidő nem lineáris változásához vezet [48], amelyre a későbbiekben még visszatérek.

4.2. A ciklusidő kiszámítása

A valós idejű ipari Ethernetes vezérlési rendszerek tervezésénél fontos kiindulási pont lehet az alaprendszer ciklusidejének ismerete, ugyanis ez határozza meg az alapvető feltételt ahhoz, hogy a kialakított rendszer valós idejű legyen. (Lásd 3.2.1. fejezet (3.6) reláció.) Ennek érdekében röviden összefoglalom, melyek azok a fontosabb tényezők, amelyek befolyásolhatják a legkedvezőtlenebb helyzetben kialakuló maximális ciklusidőt:

- bementi/kimeneti csatornák száma,
- az utasítások száma,
- az utasítások bonyolultsága (logikai, fixpontos vagy lebegőpontos utasítások),
- a CPU sebessége, és más hardver elemek időbeli viselkedése,
- időzített és véletlenszerű megszakítások (FB-k, FC-k futtatása),
- diagnosztizálás és hibakezelés,
- kommunikációs folyamatok HMI interfészekkel vagy a programozó eszközzel,
- operációs rendszerfeladatok futása,
- ipari Ethernetes kommunikáció (pl. Profinet IO vagy Profinet CBA).

A fenti tényezők közül vannak állandó, jól meghatározható elemek, amelyek nem változnak az adott rendszer működése során. Gondolok itt a felhasználói program utasításaira, vagy az IO csatornák számára. De ugyanígy állandónak tekinthetők a ciklus végén futó operációs rendszerfolyamatok is.

A továbbiakban egy konkrét rendszeren keresztül fogom bemutatni a ciklusidő kiszámítását, majd ezeknek mérésekkel történő igazolását. A választott alrendszer egy Profinet IO, Siemens S7-300 CPU315F-PN/DP IO kontrollerrel.

4.2.1. Alap ciklusidő meghatározása

Ha eltekintünk a diagnosztizálástól és a kommunikációs eseményektől, a 4.1. ábra alapján a ciklusidőt négy időtényezőre bonthatjuk. Ebből következik, hogy az alap ciklusidőt felírhatjuk a következő formában:

$$T_C = T_{PIO} + T_{PII} + 1{,}1 \cdot T_r + T_{CCP} \tag{4.1}$$

A fenti relációban T_r a felhasználói program futási idejét (runtime) jelöli. Ez függ az utasítások számától és az utasítások típusától. Az S7-300-as CPU-knál az utasítás-végrehajtási idő igencsak széles skálát foglal magába (0,1 µs-tól több száz mikroszekundumig). A CPU315F-PN/DP processzor esetében egy átlagos bitművelet végrehajtása 0,1 µs időt vesz igénybe, ha közvetlen címzést használunk, míg egy egyszerű lebegőpontos művelet elvégzéséhez már kb. 3 µs szükséges. De egy bonyolultabb művelet, például a gyökvonás akár 100 µs ideig is tarthat [49]. A futási idő tehát kiszámítható, de a külső vezérlési eseményektől függően tág határok között változhat. Maximális értékét úgy kapjuk meg, hogy a teljes felhasználói program (főprogram, funkcióblokkok és függvények) utasításait figyelembe vesszük. Egy nem túl bonyolult felhasználói program futási ideje a fentebb említett processzorral kb. 5-20 ms. A kapott értékhez még hozzá kell adnunk kb. 10% processzor időt (1,1 szorzó) amely a műveletvégzéssel párhuzamosan zajló operációs rendszer-műveletekhez köthető. (4.2. ábra szerinti időosztás).

Sokkal egyszerűbb és adott konfiguráció esetében állandónak tekinthető a folyamati térképek frissítési ideje. Itt egy alapidőből (K) kell kiindulni és ehhez hozzáadni a helyi és távoli (Profibus vagy Profinet) IO modulok bájtjaihoz tartozó időt (t_{PI}, t_P). Ennek alapján felírhatjuk:

$$\begin{aligned} T_{PII} &= K + n \cdot t_{PI} + n' \cdot t_P \\ T_{PIO} &= K + m \cdot t_{PI} + m' \cdot t_P \end{aligned} \tag{4.2}$$

A fenti relációban *n, m* a helyi, *n'*, *m'* a távoli bemeneti illetve kimeneti modulok bájtjainak számát jelenti.

A fenti értékek nagyságrendileg általában jóval kisebbek, mint a futási idő, de mindenképpen számolnunk kell vele főleg nagyszámú IO modul esetében, vagy amikor a felhasználói program egyszerű.

Ugyancsak állandónak tekinthető a ciklus végén az operációs rendszer által végzett ciklusellenőrzési pont műveleti ideje (T_{CCP}) is. Az S7-300-as processzorok műszaki leírásaiban, a gyártó 90 és 250 µs közötti értéket ad meg. Az általam használt processzornál ez az érték 140 µs [48].

4.2.2. A teljes ciklusidő

Ha elszigetelt rendszert feltételezünk, akkor az alap ciklusidőn kívül legfeljebb csak a megszakításokhoz rendelhető extra időket kell figyelembe vegyük, illetve a hibaesemények során bekövetkező ciklusidő növekedést. A megszakításokkal kapcsolatos extra idők a megszakítás jellegétől függően több csoportba sorolhatók. A legjelentősebb időnövekedést a hardveres illetve a diagnosztizálás során fellépő megszakítások jelentik. Ezek az értékek is processzortól függőek. Általában 100 és 400 µs közötti értékekkel számolhatunk. (A CPU315-nél 200 µs.) Gyorsabb processzoroknál ez 40 és 160 µs között van. A hibaesemények jóval kisebb, 100 – 150 µs-os ciklusnövekedést idéznek elő, de a gyors processzoroknál 20 µs alatt is lehet.

A korszerű irányítási és vezérlési rendszerek már nem elszigetelt módon működnek. Rendszerint valamilyen hálózatba vannak rendezve és kommunikációs kapcsolatban állnak egymással. Ezen kívül gyakorta a folyamati állapotok és adatok is vizualizálva vannak. Mindezek jelentősem megnövelhetik a vezérlők alap ciklusidejét. Az ilyen feladatokat ellátó rendszereknél a kommunikációra fordított idővel számolnunk kell, mert ez jelentősen megnövelheti a teljes ciklusidőt. A 4.1. fejezetben már utaltam erre a problémára, így most csak ennek a hatását fogom bemutatni. Ennek alapján a teljes ciklusidőre a következő összefüggés írható fel:

$$T_{Ctotal} = T_C \frac{100}{100 - C_L} \tag{4.3}$$

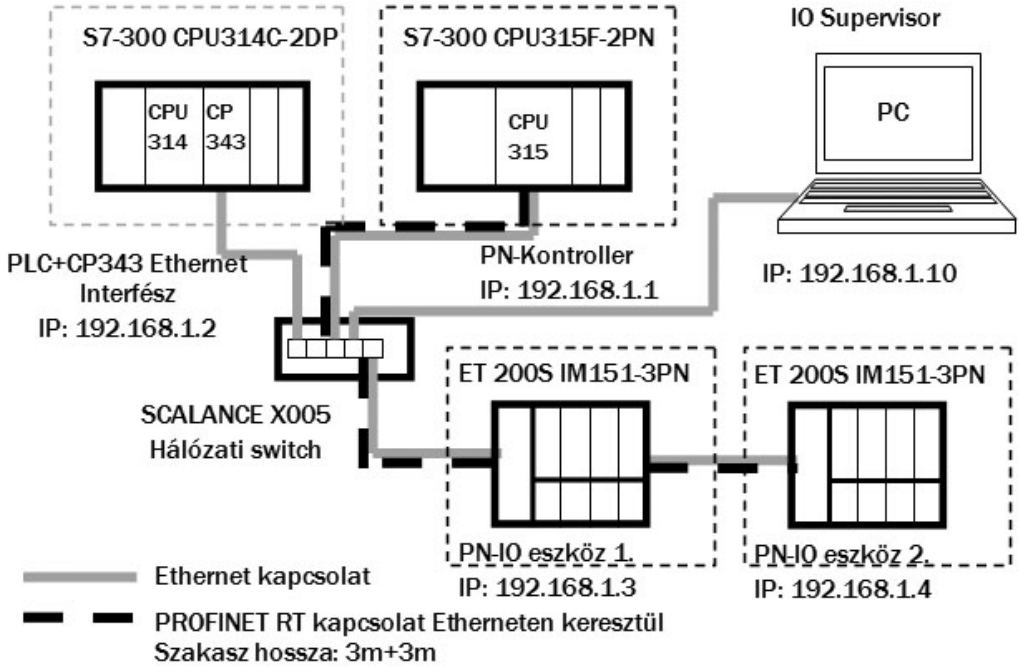
A fenti relációban C_L a kommunikációs terhelést jelenti %-ban kifejezve.

A kommunikációs terhelés hatása nem lineáris, mértéke a konfigurált értéktől függ. 20%-os alapbeállításnál ez 1,25-szörös növekedést jelent [48].

4.2.3. Az alkalmazott IO vezérlő ciklusidejének kiszámítása

A vizsgált Profinet IO rendszer vegyes topológiában egy S7-300 CPU315F-PN/DP kontrollert, két ET-200 IM151-3 PN IO eszközvezérlőt, IO supervisorként egy laptopot,

egy SCALANCE X005-ös nem menedzselhető hálózati kapcsolót tartalmaz, valamint még egy PLC-t, egy S7-300 CPU314-2DP-t, amely a CP343-1 Lean interfészen keresztül kapcsolódik a hálózathoz, de lényegében nem része a Profinet rendszernek (4.3. ábra).



4.3. ábra. A ciklusidő vizsgálatára konfigurált rendszer

A Profinet kontrollerhez közvetlenül kapcsolódnak saját IO csatornák is. Az egyik modul 4 analóg bementet (AI) és 2 analóg kimenetet (AO), míg a másik 16-16 bemeneti illetve kimeneti csatornát (DI/DO) tartalmaz. A ciklusidő-számítások során természetesen ezeket is figyelembe kell venni. Az IM151-3PN IO eszközvezérlőkhöz összesen 8-8 darab, egyenként 2-2 csatornás digitális bemeneti illetve kimeneti modulok kapcsolódnak.

A számításokhoz szükséges IO kontroller és a CPU314C-2DP PLC ide vonatkozó katalógus adatait az alábbi táblázat tartalmazza.

4.1. táblázat. A CPU ciklusidőt befolyásoló rendszeradatok [48]

	K	t_{PI}	t_P	T_{CCP}
CPU315F-PN/DP	**100µs**	**20µs/bájt**	**46µs/szó**	**140µs**
CPU314C-2DP	**150µs**	**35µs/bájt**	**-**	**150 µs**

Ezeket felhasználva az n, m, n', m' bájtok értékei a következőképpen alakulnak:

$$\begin{aligned} n &= (4 \times AI) + 2 \times DI = (8) + 2bájt \\ m &= (2 \times AO) + 2 \times DO = (4) + 2bájt \\ n' &= 8 \times 2 \times DI = 2bájt \\ m' &= 8 \times 2 \times DO = 2bájt \end{aligned} \tag{4.4}$$

Megjegyzés: Az analóg csatornákat általában közvetlenül címzik, ezért a (4.4) relációkban a zárójelben lévő részek elhanyagolhatók.

A (4.2) reláció alapján $T_{PII} = 186\ \mu s$ és $T_{PIO} = 186\ \mu s$ időtartamú lesz.

A tesztrendszer felhasználói programja vezérlési feladatokat nem végez. Csak a ciklusidő kiszámításához és bizonyos helyzetek szimulációjához szükséges programmodulokat tartalmazza. Ennek megfelelően az alapprogram kevés számú utasítást tartalmaz. Ezek összesen 135 logikai illetve bitművelet, és 23 egyszerű lebegőpontos műveletet tartalmaz. A 4.2.1. fejezetben megadott művelet-végrehajtási időket felhasználva a program futási ideje meglehetősen rövid.

$$T_r = 135 \times 0{,}1 + 23 \times 3 \approx 82{,}5\,\mu s \tag{4.5}$$

Következik, hogy az alap ciklusidő értéke az (4.1) reláció alapján:

$$T_C = 186 + 186 + 1{,}1 \times 82{,}5 + 140 \approx 603\,\mu s \tag{4.6}$$

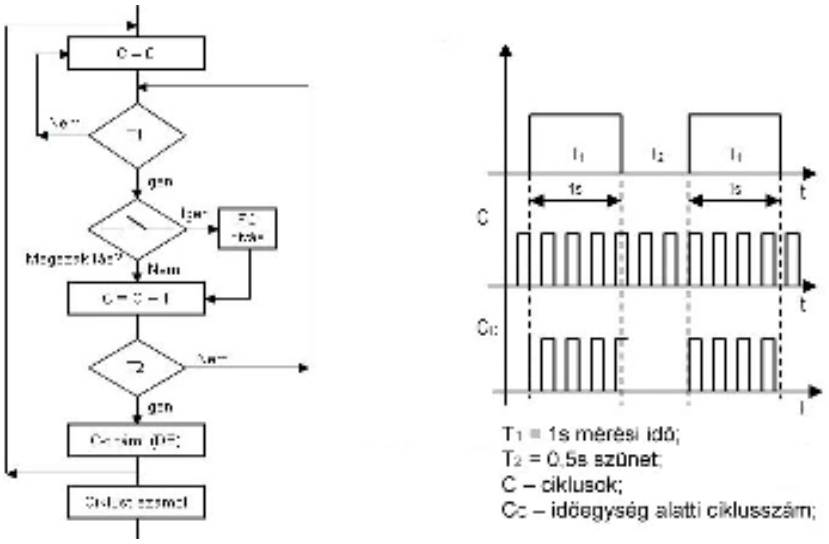
A 20%-ra beállított kommunikációs terhelést is figyelembe véve, a (4.3) reláció alapján a teljes ciklusidő értéke legfeljebb *0,75 ms* lesz, ha más tényezők nem változnak.

Egyes PLC-nél konfigurálható még a minimálisan, illetve a maximálisan megengedett ciklusidő. Ennek túllépésekor a rendszer automatikusan meghív egy hibamodult (OB80), amelyikben tervezéskor leprogramozható, hogy ilyen esetben hogyan reagáljon a rendszer.

4.3. Ciklusidő mérési módszerek [S18]

Mivel a PLC-k általában csak ezredmásodperces pontossággal mérik a ciklusidőt, indokolt kidolgozni egy olyan mérési eljárást, amellyel, még ha közvetett módon is, de pontosabban mérhető a ciklusidő. Így kimutathatóak lesznek mindazok a tényezők, amelyek befolyásolják ennek változását. A továbbiakban két mérési módszer elvi és gyakorlati megvalósítását fogom bemutatni.

Az alkalmazott egyik mérési elv a következő: megszámoljuk az időegység alatt lefutott ciklusokat. Az *1 s*-os mérési időt *0,5 s* szünet követi, majd ez automatikusan ismétlődik addig, amíg kívánjuk a mérést folytatni (4.4. ábra). Közben lehetőség van adott bemeneti csatornákon keresztül megszakításokat kérni, melyeknek eredményeként elindíthatunk különböző lebegőpontos műveleteket végző vagy analóg jeleket kezelő függvényeket, hogy lássuk, hogyan befolyásolja a ciklusidő változását.



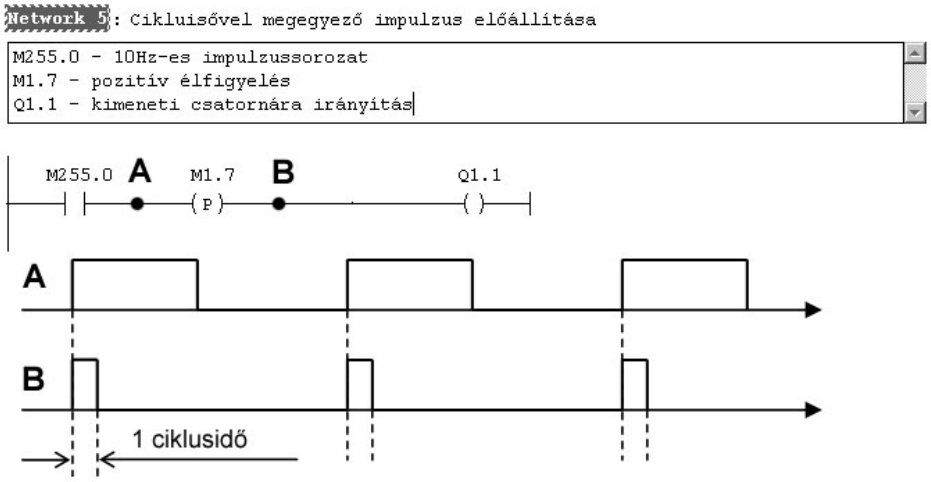
4.4. ábra. A ciklusidő mérési elve

A C_C időegység alatti ciklusszámokat a program adatblokkban (DB) tárolja, a ciklusidőt (T_C) pedig a következő összefüggéssel számíthatja ki:

$$T_C = \frac{1000}{C_C}[ms] \tag{4.7}$$

A felhasználói program a SIMATIC STEP7 V5.4 fejlesztői környezetben készült létradiagramos illetve utasításlistás formában. Programrészleteit a 2. melléklet tartalmazza. Az eljárás a főprogram részét képezi, de funkcióblokként is elkészíthető. Így alkalmazhatóvá válik bármelyik vezérlő ciklusidejének mérésére. Csak a legszükségesebb utasításokat tartalmazza, hogy kevésbé befolyásolja a tényleges felhasználói programot. A kiszámított ciklusidő felhasználható további mérési feladatokra (pl. reakció idő mérése) és szükség esetén HMI panelen bármikor megjeleníthető. A módszer előnye, hogy szoftveres úton végzi a mérést, semmilyen külső eszközt nem igényel, és sokkal pontosabban méri a ciklusidőt, mint a PLC. Hátránya, hogy nem tudjuk megmérni a rövid ideig tartó, az átlagostól jelentősen eltérő szélsőséges ciklusértékeket. Ezért a ciklusidő viselkedésének közvetlen tanulmányozására egy másik módszert fogok alkalmazni, amellyel akár az előző módszer hitelessége is igazolható.

A PLC kimeneti moduljának egyik csatornájára kivezetem azt az impulzust, amelynek szélessége megegyezik a ciklusidő nagyságával. Ilyen impulzust a legegyszerűbben úgy kaphatunk, hogy például a rendszer által generált, 10 Hz-es impulzussorozatnál felfutó-él figyelést végzünk, melynek során egy általunk választott bitmemória (Merker) értéke egy ciklusidőig logikai magas szinten lesz. Az itt kapott értéket valamelyik digitális kimeneti csatornára továbbítjuk és megmérjük a keletkezett impulzus szélességét. (4.5. ábra)



4.5. ábra. Ciklusidővel megegyező impulzus előállítása

Az előbbi módszerrel mért ciklusidő értéke hitelesnek tekinthető, ha a célnak megfelelő eszközzel mérjük az impulzus szélességét. Hátránya, hogy 50 Hz-nél nagyobb frekvenciájú impulzussorozatot nem tudunk leprogramozni, így csak 20 ms-nál nagyobb, egymást követő ciklusidőket tudunk megmérni. Ebből következik, hogy 2 ms-os ciklusidőnél csak minden tízedik mérhető.

A 4.5. ábrán látható megoldás szerint az M255.0-re konfigurált impulzussorozat frekvenciája 10 Hz, vagyis 100 ms-ként van egy felfutó él. Az előbbi példánál maradva ez azt jelenti, hogy minden ötvenedik ciklus kerül a kimenetre.

4.4. A két mérési módszer összehasonlítása

Az előző fejezetben bemutatott időegység alatti ciklusszámlálás helyességét az 4.5. ábrán bemutatott módszer szerint fogom ellenőrizni. Ha a felfutó él frekvenciája 10 Hz, akkor mindegyik időegység alatt végzett ciklusszámlálásból 10 mintát lehet külső eszközökkel megmérni. A mérést többször elvégezve egyrészt meggyőződhetünk a közvetett módszer helyességén, másrészt kiszűrhetünk olyan nagyon rövid ideig tartó ciklusidő-változásokat, amelyek különböző okok miatt előfordulhatnak a működés során. Ha ehhez még hozzáfűzzük az operációs rendszer által mért maximális ciklusidő értéket, akkor egy teljes képet kapunk a PLC ciklikus viselkedéséről.

A kimeneti csatornára kivezetett ciklusidő-impulzus időtartamát egy általam fejlesztett, LabView 8.2 környezetben futó alkalmazás segítségével mérem. Az interfészként használt USB6221 eszköz, 20 MHz-es órajelet használva, a gyártó által garantált mérési pontossága 0,05 μs [50]. Az alkalmazás alapértelmezés szerint 100 mérési eredményt tárolására alkalmas (beállítható 10000-ig), amely bármikor Excel táblába konvertálható.

Mindkét módszerrel több száz mérést végeztem a CPU314C-2DP processzoros PLC-n, különböző szituációkban, melyeknek összesített eredményeit a következő táblázatban (4.2. táblázat) foglaltam össze:

4.2. táblázat. A két mérési módszer összehasonlítása

Mérési események	Közvetett [ms]	LabView 8.2 [ms]	Eltérés [%]
Alap ciklusidő	**0,657**	**0,688**	**4,71**
Egyszerű lebegőpontos műveletek	**0,806**	**0,831**	**3,10**
Analóg bemenetek és kimenetek kezelése	**1,237**	**1,252**	**1,21**
Ciklikusan végzett lebegőpontos műveletek	**3,256**	**3,342**	**2,64**

A legnagyobb eltérés, 4,71% a legkisebb ciklusidő mérésénél mutatkozik. Ez első ránézésre soknak tűnik, de ha figyelembe vesszük, hogy mindösszesen alig több mint 30 μs az eltérés, akkor azt mondhatjuk, hogy elfogadható, a célnak megfelelő. Itt jegyezném meg, hogy a PLC által mért értékek az első három esetben 1 ms, míg az utolsóban 3ms lenne, amely kb. 45% eltérést jelent.

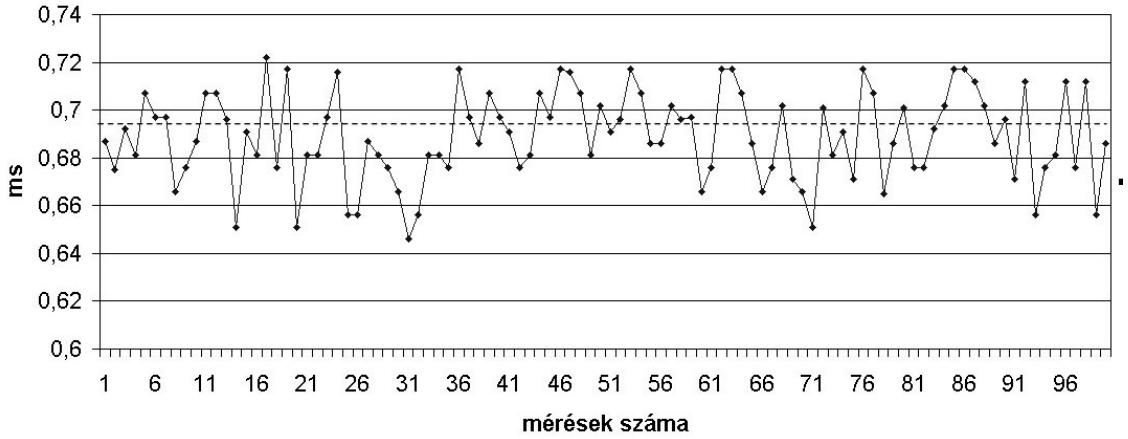
4.5. A ciklusidő mérése

Összehasonlításként, a tesztrendszerben mindkét PLC ciklusidő-változását mértem teljesen azonos körülmények között, ugyanazzal a felhasználói programmal, természetesen az adott PLC-re szabott konfigurációban. Kezdetben azért, hogy a PLC-k ciklusidejének összehasonlítását ne befolyásolja a Profinet IO rendszer, a CPU315F-PN/DP kontroller konfigurációjában nincs beállítva a Profinet IO kapcsolat. Ez utóbbi akkor kerül majd konfigurálásra, amikor a Profinet IO a ciklusidő változásra gyakorolt hatását fogom vizsgálni. Az alkalmazott mérési módszer a LabView 8.2 NI interfészes megoldás.

4.5.1. Az alap ciklusidő mérése

Ha feltételezzük, hogy csak a főprogramhoz tartozó taszkok futnak, nincs megszakítási kérelem és nincs kommunikáció sem, vagyis a kontroller elszigetelt módban

működik, akkor viszonylag állandó értékű ciklusidőt mérünk. Ezt láthatjuk a következő ábrán a CPU314-2DP PLC esetében. A mérések során több ezer mérésből véletlenszerűen rögzített 100 minta értéke 0,646ms és 0,722ms között változik 0,688ms-os átlagértékkel. (4.6. ábra)



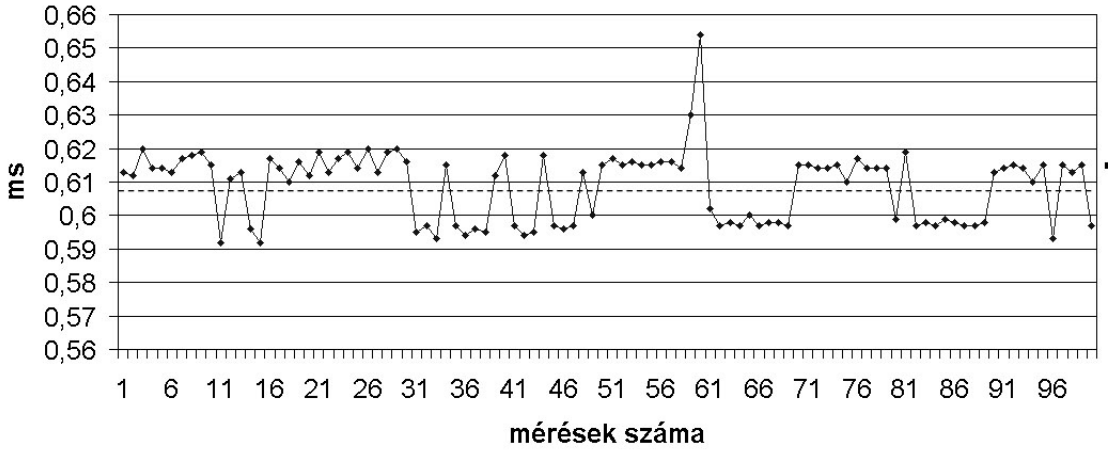
4.6. ábra. A CPU314-2DP PLC alap ciklusidő változása megszakítások nélkül

A mért legnagyobb különbség két ciklusidő között 76 µs, amely 5,52% eltérést jelent az átlagértékhez (T_{Cmed}) képest. Nevezhetem ezt jitternek vagy *ciklushibá*nak (ε_C), amely meghatározóan a hardverműködésből adódik (3.1.3. alfejezet 3.2. ábra), ugyanis ebben az esetben az (4.1) relációban szereplő futási idő (T_r), jóval kisebb, mint a ciklusidő. (4.2.3. alfejezet (4.6) reláció)

$$\varepsilon_C = \frac{T_{C\,max} - T_{C\,min}}{2 \cdot T_{Cmed}} \cdot 100 \tag{4.8}$$

A ciklushiba egyre kevésbé lesz meghatározó a felhasználói program futási idejének növekedésével, vagyis minél nagyobb a ciklusidő annál kevésbé érzékelhető a hardverműködés okozta pontatlanság.

Hasonló eredményeket kapunk a CPU315F-PN/DP PLC esetében is azzal a jelentős különbséggel, hogy ennek a processzora már valamivel gyorsabb, mint az előző PLC-é, főleg a lebegőpontos műveletvégzést illetően. [48] A következő ábrán (4.7. ábra) láthatjuk ennek a PLC-nek az alap ciklusidő változását függvényhívások nélkül.

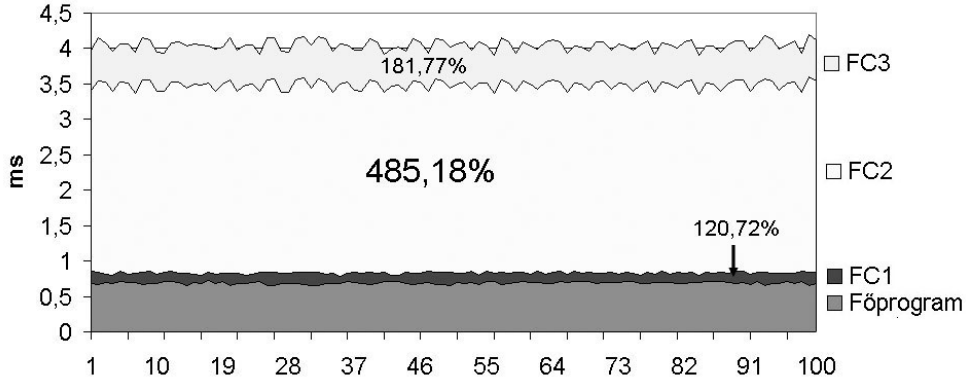


4.7. ábra. A CPU315F-PN/DP kontroller alap ciklusidő változása megszakítások nélkül

A gyorsabb működés egyértelműen látszik és ez nem csak az utasítás végrehajtás gyorsaságában nyilvánul meg, hanem a hardverműködés kismértékű javulásában is. Ez abból következik, hogy a ciklushiba ennél a PLC-nél már csak 5% körül van.

4.5.2. A megszakítások hatása a ciklusidőre

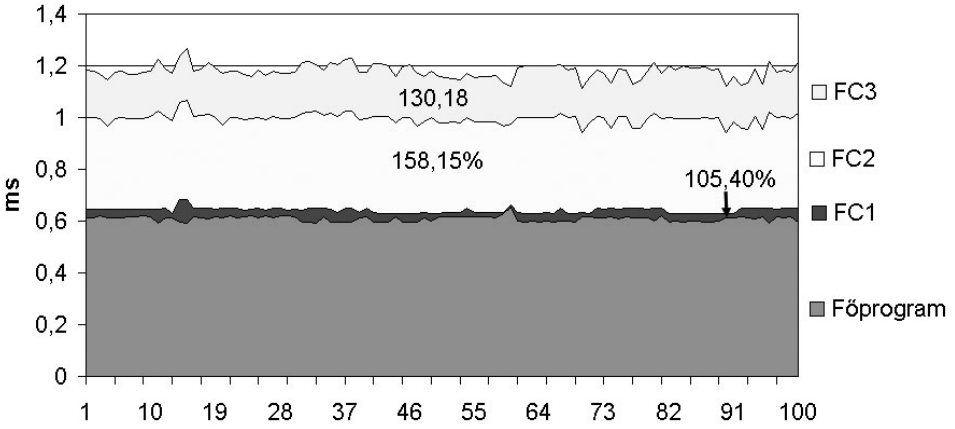
Ebben az alfejezetben végzett mérési eredmények bizonyítják a 4.2. fejezetben megfogalmazott tényezők közül az utasítások számára és bonyolultságára vonatkozó állítást. Három, különböző bonyolultságú és típusú megszakítást, vagyis függvényhívást kezdeményezek a felhasználói programban, amelyek külön-külön vagy egy időben is indíthatóak. Az első függvény (FC1) egy egyszerű lebegőpontos műveletet tartalmaz, amely két-két szám összeszorzásából, összeadásából valamint egy gyökvonásból áll. A második (FC2) két számot négyzetre emel, összeadja, gyököt von belőle, osztja és szorozza. Ezt ciklikusan végzi és összeadja, ameddig az eredmény kisebb, mint ezer. A harmadik függvény (FC3) két analóg bement feszültségeit kivonja egymásból, majd az eredményt az egyik analóg kimenetre továbbítja. A CPU314-2DP PLC ciklusidő növekedését a függvényhívásoknak megfelelően a következő ábra mutatja.



4.8. ábra. A CPU314-2DP PLC ciklusidő növekedése a függvényhívások szerint

Láthatjuk, hogy már az egészen egyszerű lebegőpontos műveletvégzés is több mint 120%-ra emeli a ciklusidőt, a ciklikusan ismételt műveletek pedig majdnem ötszörösére. Ezzel szemben az analóg jelek kezelése nem okoz rendkívüli ciklusidő változást.

A következő ábrán azt láthatjuk, hogy a CPU315F-PN/DP PLC esetében mindezek a megszakítások sokkal kevésbé növelik a ciklusidőt, főleg lebegőpontos műveletvégzéskor. (4.9. ábra)

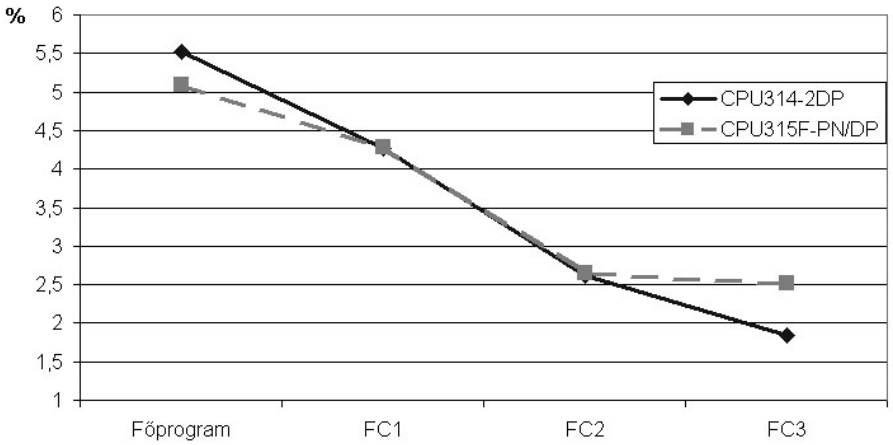


4.9. ábra. A CPU315F-PN/DP PLC ciklusidő változása a függvényhívások szerint

A 4.8. és 4.9. ábrák alapján kétségtelenül megállapítható, hogy *a programozható logikai vezérlők, valamint a valós idejű ipari Ethernet alaprendszerek ciklusideje függ az utasítások számától és azok bonyolultságától és típusától, a megszakításoktól valamint a vezérlő processzorának sebességétől.*

Ami a FC3 függvényhívást illeti, a ciklusnövekedést itt elsősorban nem a műveletek sokasága okozza, hanem az analóg csatornák A/D – D/A átalakítóinak késleltetési ideje.

Érdemes még megemlíteni a 4.5.1. alfejezetben meghatározott ciklushiba alakulását a futási idő növekedésének függvényében a két PLC esetében. (4.10. ábra)



4.10. ábra. A ciklushiba alakulása a vizsgált PLC-knél

Az ábra első szegmense azt mutatja, hogy a 315-ös PLC processzora és a hozzá tartozó hardver elemek időbeli viselkedése valamivel jobb, mint a 314-es PLC-é. A középső szakasz egyértelműen a szoftveres működésre utal. Ebből a szempontból, noha láthattuk, hogy a 315 PLC lebegőpontos műveletvégzése sokkal gyorsabb, a két PLC hasonló jelleget mutat. A harmadik szakasz, akárcsak az első, ugyancsak a hardver teljesítmény miatt eltérő, ugyanis itt az A/D – D/A átalakítók játszanak meghatározó szerepet. Itt a 314-es PLC analóg modulja sokkal kevésbé befolyásolja a ciklusváltozást, mint a 315-öshöz kapcsolódó IO modulé.

4.5.3. A kommunikációs terhelés hatása

Az ipari Ethernet rendszerek rendeltetésükből adódóan egymással összekapcsolva, hálózatban működnek. A 4.5.1. alfejezetben már szó volt róla, hogy a cikluson belül, a kommunikációra szánt idő bizonyos esetekben konfigurálható. A kérdés meglehetősen összetett. Egyrészt azért, mert külön kell kezeljük a folyamatvizualizáláshoz kapcsolódó kommunikációs terhelést az ipari Ethernetes kapcsolatokból származó ciklusnövekedéstől, bár ez utóbbi, mint látni fogjuk kevésbé befolyásolja a ciklusidőt és elég jól meghatározható. Másrészt a vizualizáláshoz kapcsolódó kommunikációt korántsem lehet állandónak tekinteni, ugyanis nagymértékben függ a megjelenített adatok számától, az adatfrissítéstől, de a lekérdezés módjától is. A kérdés tehát az, hogy hogyan válasszuk meg a helyes kommunikációs terhelés beállítását, és mi történik az így beállított ciklusidővel, ha időközben megnövekednek a kommunikációs igények?

A vizualizálásból fakadó kommunikációs terhelés hatását 10, 20 és 50%-os beállítások mellett tanulmányoztam.

Először azt vizsgáltam, hogy a különböző beállítások kommunikáció hiányában hatással vannak-e az alap ciklusidőre? Akárcsak az előző alfejezetben, ha nincsenek

megszakítások, nincs ciklusnövekedés, a válasz itt is triviális, *ha nincsenek kommunikációs igények a ciklusidő sem növekszik, bármekkora is legyen a kommunikációs terhelés beállítási értéke.* Vagyis a beállítást követően nem automatikusan növekszik meg a ciklusidő a (4.3) reláció szerint, hanem csak akkor, ha kommunikációs igények merülnek fel. A 10% és 50% beállítások során az alap ciklusidő értéke gyakorlatilag nem változott a 4.5.1. alfejezetben a 20%-os alapbeállításnál mért átlagértékhez viszonyítva. Egyenként ezer minta rögzítése után az eredményeket a 4.3. táblázatban foglaltam össze:

4.3. táblázat. Az alap ciklusidő alakulása
10, 20 és 50%-os kommunikációs terhelésnél terhelés hiányában

Alap ciklusidő 1000 minta után	**Beállított értékek**		
	10%	**20%**	**50%**
Átlag [ms]	**0,602**	**0,608**	**0,598**
Maximum [ms]	**0,673**	**0,659**	**0,626**
Minimum [ms]	**0,586**	**0,592**	**0,583**

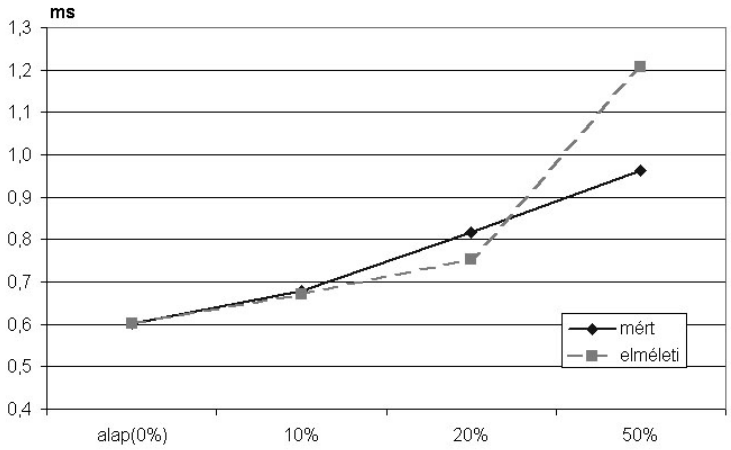
A kommunikációs terhelés által előidézett ciklusváltozás viselkedését a legegyszerűbben monitor program bekapcsolásával vizsgálhatjuk. Mindegyik esetben azonos mennyiségű információ került monitorozásra. Ezt a módszert alkalmaztam mindhárom beállítási módnál, az alap ciklusidőre, valamint a megszakítások által kiváltott esetekben is. Ezeket összehasonlítottam a számított értékekkel. Az 3. számú mellékletben a 10, 20 és 50 százalékos beállítási értékek szerinti változást látjuk monitor üzemmódban.

A 10 százalékos beállításnál rögzített mintákból az látszik, hogy több mint 60 mintánál a ciklusidő gyakorlatilag az alapértéken marad. Időközönként viszont több mintában is a ciklusidő jelentősen, több mint duplájára nő. Ez azért következik be, mert a meglehetősen nagy mennyiségű információt, ha csökkentett minőségben is, de igyekszik megjeleníteni. Az átlagérték 100 mintára számolva 0,68ms, amely csak kevéssel haladja meg a számítások szerinti 10%-os növekedést.

A 20%-os alapértelmezett beállításnál jól látható, hogy itt az esetek több mint 60 százalékánál a ciklusidő 0,8 és 0,9 ms között van. Az átlagérték 0,81 ms, amely szintén nagyobb, mint a számított 20%-os emelkedés. Itt is láthatóak időközönként kiemelkedő ciklusértékek, amelyek azt mutatják, hogy a monitor üzemmód 20%-nál nagyobb beállítást igényelne.

A harmadik 50%-os beállításnál a mérési adtok átlagértéke 0,96ms. A rögzített minták közel fele 1-1,4 ms között van, vagyis az 1,2 ms-os számított érték körül. Kimagaslóan magas érték mindösszesen csak egy van, és ez nem feltétlenül kommunikációs eredetű. Azt is láthatjuk, hogy viszonylag sok minta (több mint 20% esetében) az alapértéken vagy annak közelében van. Mindezek azt bizonyítják, hogy jelen esetben sok az 50%-os kommunikációs terhelés beállítása.

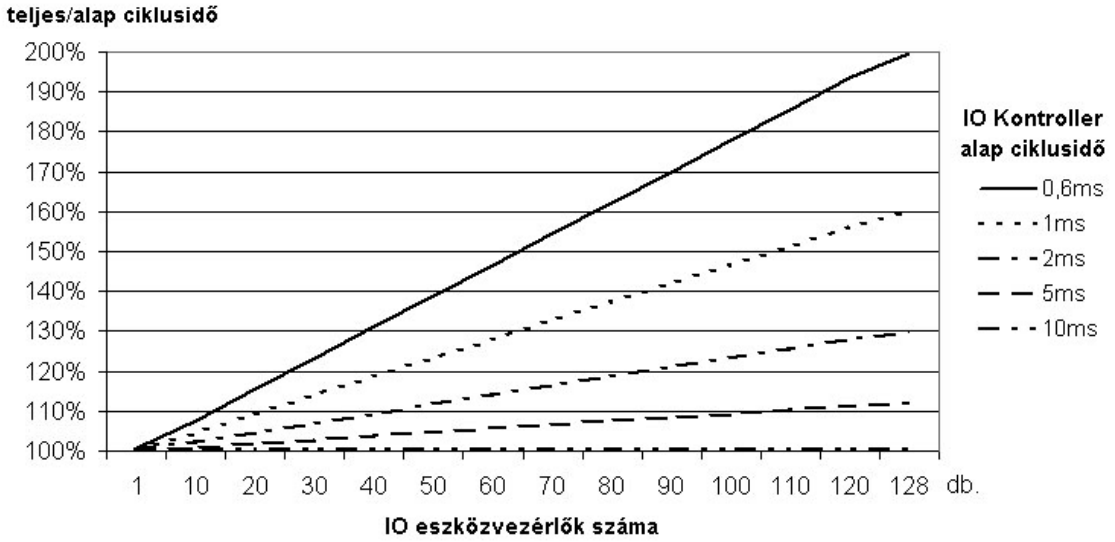
A következő ábrán a mért és a számított értékek valamint a kommunikációs terhelés szerinti ciklusidő változás alakulását láthatjuk. A fenti szituációhoz igazodva, kommunikációs szempontból az optimális beállítás a két görbe metszéspontjában, kicsivel több, mint 20% felett van. (4.11. ábra)



4.11. ábra. Kommunikációs szempontból optimális kommunikációs terhelés beállítása

4.5.4. A kommunikációs relációk hatása a ciklusidőre

Ebben az alfejezetben arra keresem a választ, hogy a Profinet IO eszközvezérlők hozzáadása milyen mértékben befolyásolja a ciklusidő változását? A gyártó adatai szerint [49] egy Profinet IO kontrollerhez maximum 128 darab IO eszközvezérlő csatlakoztatható. Mindegyik eszköz és a vezérlő között a konfigurálás során kétirányú full duplex kommunikációs relációk épülnek fel. Csupán az eszközvezérlők csatlakoztatása, az IO modulok nélkül, számottevően nem befolyásolja a ciklusidőt, mert a kapcsolat-menedzselési funkciókat a kommunikációs processzorok végzik. Az ezzel kapcsolatban végzett mérési eredmények vezérlőnként átlagosan 4-5 µs ciklusnövekedést idéztek elő. Ez a maximális számú IO eszközvezérlő esetében is kevesebb, mint 600 µs ciklusidő növekedést jelent. Ha a teljes ciklusidő növekedést az IO kontroller alap ciklusidejéhez viszonyítjuk, akkor láthatjuk, hogy amikor az alap ciklusidő növekszik, az eszközvezérlők számának növelése egyre kevésbé befolyásolja a ciklusidő változást. 5 ms alap ciklusidő esetében például alig több mint 110%-ra növekszik a teljes ciklusidő. (4.12. ábra)



4.12. ábra. A ciklusidő növekedési rátája az IO eszközvezérlők számának növelésével

A fenti eredmények alátámasztják a 3.4.2. alfejezetben tett megállapítást miszerint *az ipari Ethernetes alrendszer egységes egészként viselkedik, úgy mintha az eszközvezérlők a kontroller saját részei lennének, annak ellenére, hogy közöttük a kapcsolat Etherneten keresztül valósul meg.*

A tesztek során digitális bemeneti és kimeneti modulokat csatlakoztattam rendre, összesen három IO eszközvezérlőhöz. A tesztrendszerben használt IM151-3PN HF eszközvezérlőkhöz egyenként maximum 63 darab IO modul csatlakoztatható és 256 bájt IO adatot tudnak kezelni. A három IO eszközvezérlőhöz, különböző konfigurációkban csatlakoztatott digitális IO modulokkal végzett több mint ezer ciklusidő minta vizsgálata során a bementi modulok bájtonként *87 µs*, míg a kimeneti modulok *144 µs* ciklusidő növekedést (extra ciklusidő) produkáltak. Ebben az esetben ez szinte teljes egészében a folyamati térképek kezelési idejének növekedésében nyilvánul meg. Ez bemeneti modulonként *22 µs*-ot, kimenetinél pedig *36 µs*-ot jelent. A különbség minden bizonnyal a modulok csatornáinak eltérő olvasási és írási idejével magyarázható.

A következő táblázatban néhány gyakoribb, valamint szélsőséges konfiguráció extra ciklusidejének alakulását foglaltam össze. A szürke árnyalatos területek *50ms*-nál nagyobb ciklusidő növekedést okoznak, ezért ezekhez a konfigurációkhoz már gyorsabb IO vezérlőre van szükség. (4.4. táblázat).

4.4. táblázat. A ciklusidő növekedése az IO modulok számától függően

IO Eszköz-vezérlők [db]	**Bemeneti/Kimeneti modulok száma** [db] *						
	4/4	**8/4**	**16/8**	**16/16**	**24/16**	**32/24**	**32/32**
1	0,23	0,32	0,64	0,92	1,10	1,56	1,81
16	3,69	5,08	10,16	14,77	17,55	24,93	28,97
32	7,39	10,16	20,32	29,54	35,10	49,87	57,94
48	11,08	15,24	30,49	44,32	52,64	74,80	86,90
64	14,77	20,32	40,65	59,09	70,19	99,74	115,87
80	18,47	25,41	50,81	73,86	87,74	124,67	144,84
96	22,16	30,49	60,97	88,63	105,29	149,60	173,81
112	25,85	35,57	71,13	103,40	122,84	174,54	202,77
128	29,54	40,65	81,30	118,18	140,38	199,47	231,74

* a ciklusidő értékek *ms*-ban vannak megadva

A fenti táblázatban lévő értékek a digitális IO modulokra vonatkoznak, amelyek ugyanúgy kerülnek leképzésre a vezérlő folyamati térképmemóriájába, mint a saját közvetlen IO csatornái, azzal a különbséggel, hogy a bájtonkénti leképzési idő eltérő lehet. Ez attól is függ, hogy milyen gyorsan tudja az eszközvezérlő alkalmazási rétegének kommunikációs protokollja az IO adatokat az Ethernet keretekben elhelyezni, illetve az IO kontroller kibontani ezekből.

Az eddig végzett mérési szituációkban a kontroller és az IO eszközvezérlők között az adatfrissítési ciklusidő az alapértelmezett *128 ms* értékre volt beállítva. Ez egy meglehetősen alacsony frissítési idő. A kommunikációs terhelés 20%. A tesztrendszer két IO eszközvezérlője összesen öt bemeneti és öt kimeneti modult tartalmaz. (A harmadik eszközvezérlő nem tartalmaz IO modulokat). Az így mért ciklusidő átlaga a már említett 128ms-os frissítési időbeállítással *0,89 ms* volt. Feltevődik a kérdés, hogy elégséges-e a beállított 20%-os kommunikációs terhelés? A (4.3) relációt alkalmazva a ciklusidő számított értéke *1,11 ms* értékre növekszik. A mérési eredmények, 500 minta után átlagosan *0,96 ms*-os ciklusidőt eredményeztek. A mért maximális érték *1,045 ms* volt. Vagyis nem indokolt a nagyobb kommunikációs terhelés beállítása.

4.6. Új tudományos eredmények

Az ipari vezérlő- és kommunikációs rendszerek tervezésekor valamint időkritikus viselkedéseinek vizsgálatakor elengedhetetlen feltétel a ciklusidő ismerete. Ebben a fejezetben kidolgoztam két olyan eljárást, amelyek segítségével sokkal pontosabban mérhető a ciklusidő, mint a vezérlők rendszerszoftvere által biztosított mérési értékek.

A ciklusszámláláson alapuló közvetett módszer bármilyen PLC-nél alkalmazható, hiszen nem igényel semmilyen külső berendezést. Segítségével nyomon követhető a ciklusidő változás, a mért értékek felhasználhatóak a reakcióidők kiszámításakor, amelyek a valós idejű működés kritériumait határozzák meg.

A második módszer közvetlen mérési lehetőséget biztosít külső időmérő számára. Csak olyan PLC-knél használható, amelyek rendelkeznek elektronikus kimeneti csatornával is. Elsősorban hasonló területen történő kutatási célokra használható. Sokkal érzékenyebb az előzőnél és akár néhány mikroszekundumos változások is mérhetőek. A mérési eredmények korlátlan mennyiségben, tetszés szerint tárolhatóak. A módszer segítségével sikerült mérnem a ciklushibát. Ennek alapján igazoltam a ciklushibára vonatkozó relációt, valamint a ciklushiba és a vezérlési program futási ideje közötti kapcsolatot.

Ugyancsak ezzel a módszerrel vizsgáltam a kommunikációs terhelés hatását különböző konfigurációs beállítások mellett és megállapítottam, hogy ha nincsenek kommunikációs igények, a ciklusidő nem változik, bármekkora is legyen a beállított érték. A konfiguráció során beállított érték ciklusidő korlátot jelent a kommunikációs igénybevétel során, amelyet a vezérlő igyekszik több-kevesebb sikerrel betartani. Ugyanitt határoztam meg a kommunikáció szempontjából optimálisnak tekinthető kommunikációs terhelés értékét is.

További kísérleti eredményekkel igazoltam az előző fejezetben megfogalmazott és kidolgozott valós idejű Ethernet alaprendszerre vonatkozó megállapítást, miszerint az egységes egészként viselkedik, úgy mintha az Ethernet hálózaton keresztül kapcsolódó IO modulok saját közvetlen elemei lennének.

Az általam kidolgozott mérési eljárásokkal és az így kapott mérési eredmények egyértelműen alátámasztották a következő tézist.

2. TÉZIS [S11, S13, S17, S18]

Kidolgoztam két olyan új ciklusidő mérési eljárást, amelyeknek segítségével, sokkal pontosabban mérhetők az IO kontrollerek vagy PLC-k ciklusidői, mint a rendszerszoftverek által mért értékek.

2.1. Altézis. *Megállapítottam, hogy a programozható logikai vezérlők, valamint a valós idejű ipari Ethernet alaprendszerek ciklusideje függ az utasítások számától, azok bonyolultságától és típusától, a megszakításoktól valamint a vezérlő processzorának sebességétől.*

2.2. Altézis. *A ciklushiba egyre kevésbé meghatározó a felhasználói program futási idejének növekedésével, vagyis minél nagyobb a ciklusidő annál kevésbé érzékelhető a hardverműködés okozta pontatlanság.*

2.3. Altézis. *Kommunikációs igények hiányában a ciklusidő nem növekszik, bármekkora is legyen a kommunikációs terhelés beállítási értéke.*

V. FEJEZET

NEM SZINKRONIZÁLT IPARI ETHERNET RENDSZEREK REAKCIÓIDEJÉNEK MEGHATÁROZÁSA ÉS MÉRÉSE

A folyamatirányításban és a vezérléseknél alkalmazott ipari Ethernet rendszerek jelentős része, nem igényli a kontroller és az eszközvezérlők közötti időbeli szinkronizálást. Az ilyen rendszerek vezérlői általában 1 ms-nál nagyobb ciklusidővel rendelkeznek és változó sebességű, olykor kimondottan lassú folyamatokat irányítanak. Ebből következik, hogy a rendszer időkorlátja sem kritikus értékű, vagyis a szinkronizálás hiányából adódó néhány milliszekundumos eltérés számottevően nem befolyásolja a rendszer valós idejű működését. A II. fejezet elején az ilyen rendszereket A, illetve B valós idejű osztályba soroltam. Ebben a fejezetben az ilyen, ciklikus alapműködésű rendszerek reakcióidejére vonatkozó kritériumokat és szabályokat fogom meghatározni és mérési eredményekkel igazolni ezeket.

5.1. A reakcióidő fogalma és fontossága

A gyakorlatban sok esetben meghatározó lehet, hogy mennyi idő telik el egy bemeneti csatorna gerjesztésének kezdeti pillanatától egészen addig a pillanatig, amíg a megcímzett kimeneti csatornán megjelenik a válasz. A bementi csatorna gerjesztése és a kimeneten megjelenő válasz között eltelt időt szokás *reakcióidőnek*, vagy *válaszidőnek* nevezni. Ezt főleg olyan esetekben tárgyalom a továbbiakban, amikor a bemeneti és a kimeneti eszköz vezérlői közötti kapcsolat az ipari Etherneten keresztül valósul meg. A vezérlési rendszerek tervezésénél az is fontos lehet, hogy milyen gyakorisággal követhetik egymást a bemenetre érkező jelek, amelyeknél még egyértelmű választ kapunk a kimeneten? Mekkora lehet az a minimális gerjesztési időtartam, amelyiket még „észreveszi” a bemenet. Befolyásolja-e az ipari Ethernet hálózat forgalma a válaszidők alakulását?

Hogy ezekre a kérdésekre választ kapjunk, meg kell határozni mindazokat a tényezőket, amelyek befolyásolhatják a válaszidők alakulását.

5.1.1. A reakcióidőt meghatározó tényezők

A III. fejezetben tárgyalt IO kontrollerek ciklusidejének elemzése során láthattuk, hogy bármely vezérlési eredménynek, a kimeneten történő megjelenéséhez legkevesebb egy ciklusidőre van szükség. Ebből következik, hogy a válaszidőket meghatározó egyik tényező a vezérlő ciklusideje lesz.

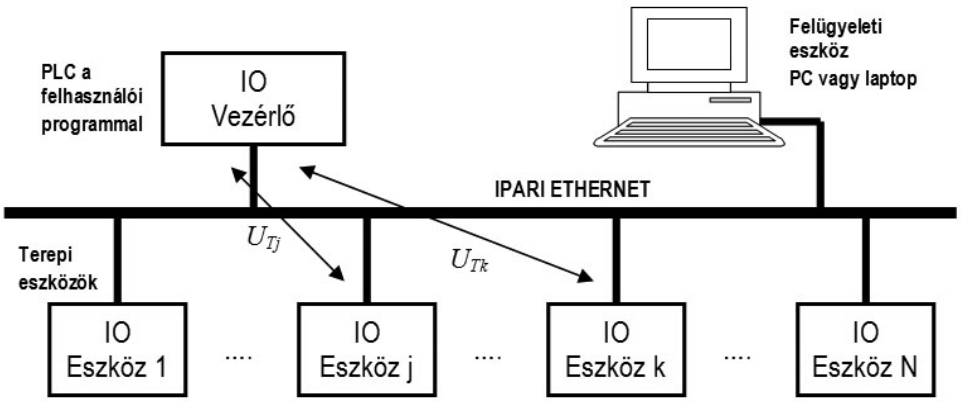
A PLC-k valamint az eszközvezérlők bemeneti csatornái zajszűrő áramköröket tartalmaznak annak érdekében, hogy megakadályozzák a nemkívánatos villamos zavarok bejutását a rendszerbe. A minél hatékonyabb és biztonságosabb szűrés érdekében ezek a

szűrőáramkörök bizonyos késleltetéseket is előidéznek. Ezeket a késleltetéseket ismerjük és egyes rendszereknél konfigurálhatók is [55].

A 2.2.3. fejezetben bemutatott küldési idő a Siemens Profinet IO rendszereknél *0,25* és *4 ms* között konfigurálható. Ennek tipikus beállítási értéke az *A* és *B* valós idejű osztályoknak megfelelően *1 ms*. Ez minden esetben alapját képezi a frissítési időnek. Ennek megfelelően az ilyen rendszereknél a legkisebb frissítési idő *1 ms* lehet. A további értékek 2 hatványainak megfelelő szorzóval állíthatók be egészen *512 ms*-ig [48].

A 3.4.2. fejezetben meghatároztam az ipari Ethernet alrendszer valós idejű működésének feltételét, miszerint *a legnagyobb ciklusidővel rendelkező eszköz buszfrissítési periódusa legalább négyszer kisebb kell legyen mint a hozzárendelt időkorlát*. A válaszidők vizsgálatának szempontjából ez azt jelenti, hogy a buszfrissítési ciklust a válaszidőknek megfelelően kell megválasztanunk a rendszer tervezésekor, hogy valós idejű működést biztosítsunk.

Tételezzük fel az 5.1. ábrán látható ipari Ethernet alrendszert, amely egy IO vezérlővel, N darab eszközvezérlővel és egy felügyeleti állomással rendelkezik. A vezérlő mindegyik eszközzel kétirányú kommunikációs relációt épít fel U_{Tj} frissítési periódussal. Legyen a vezérlő ciklusideje T_C.



5.1. ábra. Ipari Ethernet alrendszer felügyeleti eszközzel

Összefoglalva az előzőeket a reakcióidők alakulását a legtöbb esetben az alábbi tényezők határozzák meg:

- a ciklusidő (T_C),
- a bemeneti csatornák késleltetési ideje (I_D),
- a küldési idő (S_T)
- a konfiguráció során beállított frissítési idők (U_{Tj}, U_{Tk}).

Ezek közül egyedül a frissítési idő az, amelyet a valós idejű követelményeknek megfelelően megválaszthatunk. A másik három tényezőt érdemben nem nagyon tudjuk csökkenteni. Azt is figyelembe kell venni, hogy nem mindegy, hogy a ciklusidő és a frissítési időhöz képest mikor érkezik a bemeneti jel. Ennek megfelelően a válaszidőknek lesz egy alsó és egy felső elméleti határa.

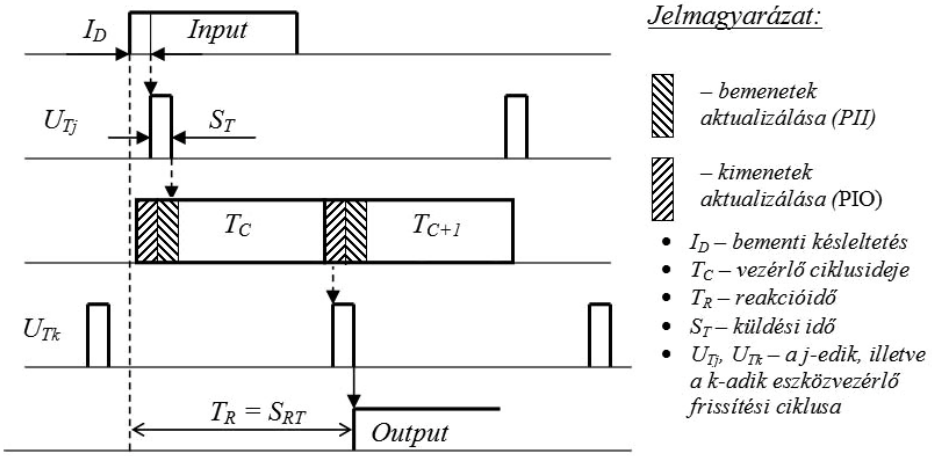
5.1.2. A reakcióidő elvi meghatározása

Az 5.1. ábra szerinti alrendszerben általános esetként feltételezzük, hogy a *j*-edik eszközvezérlő egyik bemeneti csatornájára érkező jel hatására a *k*-adik eszközvezérlő valamelyik kimeneti csatornája kell reagáljon. A szituáció elemzése érdekében két szélsőséges esetet kell meghatározni.

Az alsó határ (S_{RT}) a legkedvezőbb elméleti helyzetet feltételezi, amikor egy ciklusidőn belül a válasz is megérkezik. Ebben az esetben a válaszidőt csak a kontroller ciklusideje és a bemeneti csatorna késleltetési ideje határozza meg [48].

$$S_{RT} = T_C + I_D \tag{5.1}$$

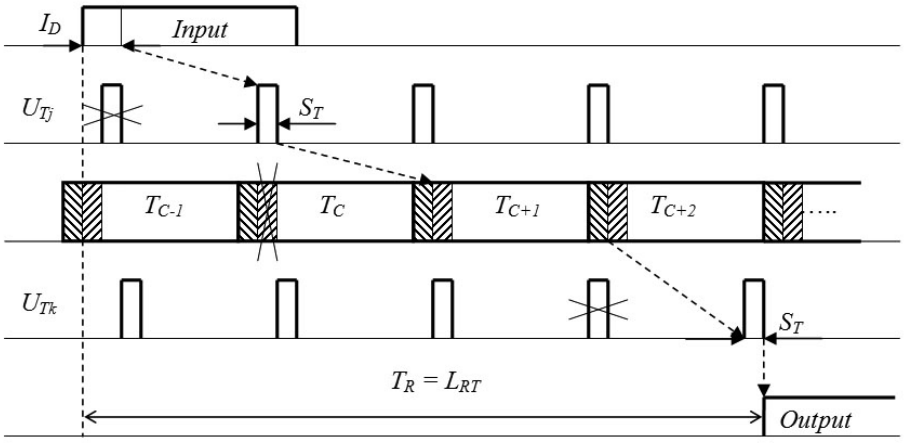
Ez az az eset, amikor a bemeneti gerjesztés éppen a küldési idő idején érkezik, és a bementi folyamati kép (*PII*) aktualizálását még eléri, a válasz pedig a következő kimeneti kép (*PIO*) aktualizálásakor ugyancsak a küldési időnek éppen a legkedvezőbb pillanatában érkezik. (5.2. ábra)



5.2. ábra. A legkedvezőbb esethez tartozó válaszidő

Az előbbi eset a valóságban legfeljebb egyirányú kommunikáció esetében fordulhat elő, ugyanis nagyon csekély annak a valószínűsége, hogy ez a három feltétel kedvezően teljesüljön. De nem is igazán lényeges, legfeljebb ki tudjuk számolni az elméletileg lehetséges legrövidebb válaszidőt.

A legkedvezőtlenebb helyzet (L_{RT}) az lesz, amikor az IO eszköz bemenetére érkező jel éppen lekési az aktuális frissítési időt, majd a következő frissítés után a ciklus bementi folyamati kép aktualizálási (*PII*) pillanatát is, ezért csak a következő ciklusban töltődik be és kerül végrehajtásra. A válasz esetében pedig a következő ciklus kimeneti folyamati kép aktualizálásakor (*PIO*) éppen lekési a frissítési időt, így csak a következővel kerül továbbításra. Ezt az esetet láthatjuk a következő ábrán. (5.3. ábra)



5.3. ábra. A legkedvezőtlenebb eset válaszideje

A fenti ábra alapján a legkedvezőtlenebb esethez tartozó válaszidő értékét általános formában a következő reláció alapján határozhatjuk meg:

$$L_{RT} = I_D + U_{Tj} + 2 \cdot T_C + U_{Tk} + 2 \cdot S_T \tag{5.2}$$

Mindkét szélsőérték előfordulási valószínűsége meglehetősen csekély, főleg nagyobb értékű frissítési idővel rendelkező konfigurációk esetében. A válaszidők várható középértéke a következő összefüggéssel számítható ki.

$$T_{RM} = \frac{U_{Tj} + U_{Tk} + 3 \cdot T_C}{2} + I_D + S_T \tag{5.3}$$

Nem szinkronizált ipari Ethernet rendszereknél a legkisebb frissítési idő szerinti válaszidő számított átlaga *4 ms*. (ha $U_{Tj} = U_{Tk} = T_C = S_T = 1\ ms$; $I_D = 0{,}5\ ms$)

Megjegyzés: Az U_{Tj}, U_{Tk} frissítési időket értelmetlen sokkal kisebb értékekre beállítani, mint a ciklusidő értéke, mert ekkor a reakcióidők változását igen jelentős mértékben a vezérlő ciklusideje fogja meghatározni, amely igencsak érzékeny a megszakítások által indított taszkok futási idejére.

A jitter számított maximális értéke a következő relációval adható meg:

$$j_{max} = U_{Tj} + U_{Tk} + T_C + 2 \cdot S_T \tag{5.4}$$

A legkisebb frissítési idővel számítva ez az érték meglehetősen nagy, *5 ms* lesz.

A középértékhez viszonyított válaszidő *relatív eltérés* százalékos mutatóját pedig az alábbi kifejezés határozza meg:

$$\varepsilon_R = \frac{j_{max}}{2 \cdot T_{RM}} \cdot 100\% = \frac{U_{Tj} + U_{Tk} + T_C + 2 \cdot S_T}{U_{Tj} + U_{Tk} + 3 \cdot T_C + 2 \cdot I_D + 2 \cdot S_T} \cdot 100\% \tag{5.5}$$

Az előbbi példánál maradva ez az eltérés a legkisebb frissítési idő szerint számítva 62,5% lesz. Ha a frissítési időt növeljük, akkor ez az érték is növekedni fog. Csökkenést legfeljebb akkor érhetnénk el, ha növelnénk a kontroller ciklusidejét, illetve a bementi csatornák késleltetési idejét. Ezeknek a növelése viszont kedvezőtlen hatással van a válaszidőkre.

A tesztrendszerben végzett mérési eredmények azonban azt mutatják, hogy az előbbi elméletileg számított értékeknél sokkal kedvezőbb értékeket kapunk, mind az átlagérték, mind a maximális jitter, mind pedig a válaszidő relatív eltérését illetően. Ez utóbbi akár 27%-ra is csökkenhet. Ez azzal magyarázható, hogy a legkedvezőtlenebb esethez tartozó szituáció előfordulási valószínűsége sokkal kisebb, mint az alsó határhoz tartozó szituációé [S3]. (lásd 5.3.2.1. fejezet 5.3. táblázat)

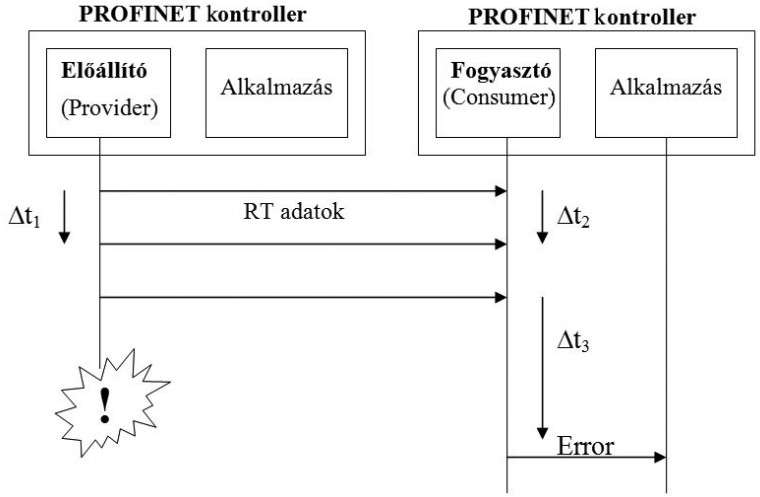
5.1.3. Fontosabb sajátos esetek

Az előző fejezetben bemutatott általános helyzeten kívül még két fontosabb szituációt érdemes tárgyalni. Az egyik az, amikor az IO kontroller és az IO eszközvezérlők között létrejövő kommunikációs reláción belül csak egyirányú adatátvitel történik. Ez lehet a PLC felől valamelyik IO eszközvezérlő irányába, ilyenkor a PLC az termelő, az

eszközvezérlő pedig a fogyasztó, vagy fordítva, az eszközvezérlő az termelő, a PLC pedig a fogyasztó. Ugyanilyen kommunikációs kapcsolat kialakítható két PLC között is, vagy az OPC szerver és egy PLC között.

A másik gyakran előforduló konfiguráció az, amikor az IO kontroller és az eszközvezérlő között kétirányú adatkommunikáció zajlik. Ilyenkor mindkét eszköz egyaránt betölti az termelő és a fogyasztó szerepét is. Mivel a 100 Mb/s-os ipari Ethernet hálózat alapértelmezésből full duplex jellegű, a kétirányú kommunikációs relációk akár egyidőben is kialakulhatnak.

A ciklikus adatok egyirányú továbbításának egyik legegyszerűbb módja a **visszaigazolás nélküli** simplex adatátvitel [2], amely feltételezi, hogy a csatorna jó minőségű, hibamenetes és a fogyasztó képes olyan sebességgel fogadni az adatokat, ahogyan az termelő küldi. A Profinet CBA esetében ez sok esetben meg is valósítható, mert a kommunikáció általában egyenrangú eszközök között zajlik. (5.4. ábra).



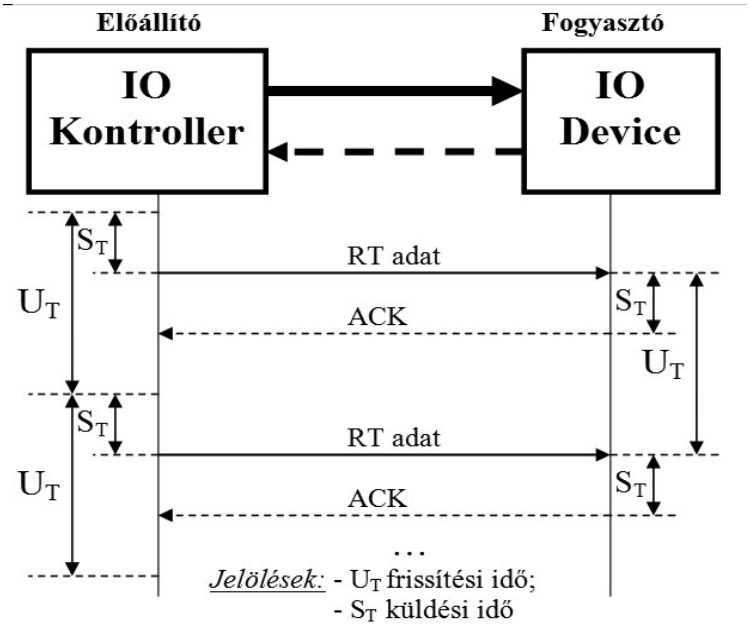
5.4. ábra. Nyugtázás nélküli ciklikus adattovábbítás [37]

Az előbbi ábrában: Δt_1 a frissítési időt, Δt_2 a feldolgozási időt, Δt_3 pedig a várakozási időt jelenti. A frissítési időnek minden esetben igazodnia kell a fogyasztó felhasználói programciklusához. A feldolgozási időt a fogyasztó határozza meg, de számottevően nem haladhatja meg a frissítési időt, mert akkor hibás működéshez vezetne. A várakozási idő maximális értéke a konfigurációban határozható meg. Ennek az időnek a túllépése már nem fogadható el, ezért ilyenkor hibaüzenetet generál.

$$\Delta t_3 = n\,\Delta t_2 \tag{5.6}$$

Az alapbeállítás szerint $n = 3$, RT adatok küldése esetén. Vagyis ilyen jellegű hibaüzenetet a fogyasztó akkor produkál, amikor a várt adatok három ciklusidőn belül sem érkeznek meg.

A másik, főleg a PROFINET IO rendszereknél alkalmazott RT adattovábbítási módszer a **megáll és vár** (stop and wait) adatátvitel [2]. Itt már a vétel visszaigazolásaként a fogyasztó visszaküld egy úgynevezett ál-keretet (ACK), amelyet pozitív nyugtaként (Positive ACKnowledgement) tekinthetünk. A keret a hibamentességről nem tartalmaz információt, az termelő viszont megismétli az adatküldést, ha meghatározott időn belül nem érkezik nyugtázás. Lényegében kétirányú kapcsolatot feltételez, ahol azonban valós idejű adattovábbítás csak az egyik irányban történik. Viszont adattovábbítás fordítva is lehetséges, vagyis az IO eszközvezérlő felől az IO kontroller irányába. (5.5. ábra).



5.5. ábra. Egyirányú adatforgalom nyugtázással

Megjegyzés: Az eszközök közötti adattovábbítási idő mindkét esetben elhanyagolható, mert jóval kisebb, mint a frissítési vagy az előkészítési és feldolgozási idő.

A válaszidőket tekintve, a legkedvezőbb eset kivételével, mindegyik reláció módosul, ugyanis ebben az esetben csak egy frissítési és egy küldési idővel kell számolnunk. Ennek megfelelően a (5.2), (5.3), (5.4), (5.5) relációk a következőképpen módosulnak:

$$L_{RT} = I_D + U_T + 2 \cdot T_C + S_T \tag{5.2'}$$

$$T_{RM} = \frac{U_T + 3 \cdot T_C + S_T}{2} + I_D \tag{5.3'}$$

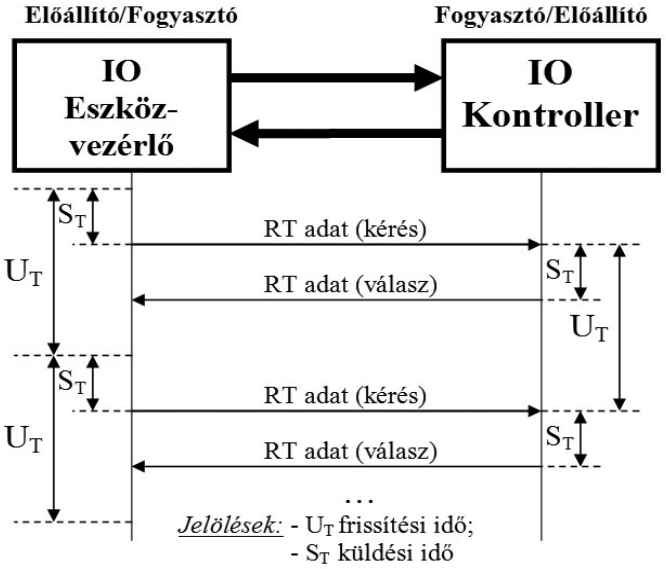
$$j_{max} = U_T + T_C + S_T \tag{5.4'}$$

$$\varepsilon_R = \frac{U_T + T_C + S_T}{U_T + 3 \cdot T_C + 2 \cdot I_D + S_T} \cdot 100\% \tag{5.5'}$$

Ha ismét a legkiélezettebb helyzetet vesszük alapul (U_T=T_C=S_T=*1 ms;* I_D=*0,5 ms*), a várható válaszidő középérték *3 ms*-ra, a jitter maximális értéke ugyancsak *3 ms*-ra, a relatív eltérés pedig *42,85%*-ra csökken.

A mérési eredmények ebben az esetben is valamivel kedvezőbb értékeket mutatnak, akárcsak az előző szituációban.

Az osztott intelligenciájú ipari Ethernet rendszerek talán leggyakrabban előforduló esete az, amikor az IO eszközvezérlő valamelyik bemeneti csatornájára érkezik valamilyen villamos jel, amelynek hatására ugyanennek az eszközvezérlőnek valamelyik kimeneti csatornáján megjelenő villamos jel vezérli a beavatkozást. A szituáció nagyon hasonló az 5.5. ábrán bemutatotthoz, azzal a különbséggel, hogy az eszközvezérlő hol termelő, hol fogyasztó szerepet tölt be, a valós idejű adatátvitel pedig minden esetben kétirányú. Ezt az esetet mutatja az 5.6. ábra.



5.6. ábra. Kétirányú RT adattovábbítás

Mivel a bemeneti és kimeneti csatornák ugyanahhoz az eszközvezérlőhöz tartoznak, az (5.2), (5.3), (5.4), (5.5) összefüggések az alábbiak szerint változnak:

$$L_{RT} = I_D + 2 \cdot (U_T + T_C + S_T) \tag{5.2"}$$

$$T_{RM} = I_D + U_T + S_T + \frac{3 \cdot T_C}{2} \tag{5.3"}$$

$$j_{max} = T_C + 2 \cdot (U_T + S_T) \tag{5.4"}$$

$$\varepsilon_R = \frac{T_C + 2 \cdot (U_T + S_T)}{3 \cdot T_C + 2 \cdot (I_D + U_T + S_T)} \cdot 100\% \tag{5.5"}$$

5.1.4. További sajátos esetek és következmények

Az előző fejezetben (5.1.3.) tárgyalt általános és sajátos szituációk mellett érdemes még megemlíteni azokat az eseteket, amelyeknél a bemeneti késleltetés és a kontroller ciklusideje számottevően már nem befolyásolja a válaszidőket. Ennek megfelelően kijelenthetők:

1. A küldési időt elhanyagolhatjuk, ha feltételezzük, hogy jóval kisebb, mint a frissítési periódus (pl. $U_T = 8\ ms$ feletti beállításoknál).
2. Ha a két frissítési idő közül a nagyobbikat a maximális ciklusidőhöz igazítjuk ($max\{U_{Tj};U_{Tk}\} = U_T \geq T_{Cmax}$), és ezek összege jóval nagyobb, mint a bemeneti késleltetés akkor a bementi késleltetést elhanyagolhatjuk és a legkedvezőtlenebb eset válaszideje:

$$L_{RT} = 4U_T \tag{5.6}$$

   Ezzel a konkrét, gyakorlatias szituációval igazolható a 3.4.2. fejezetben meghatározott, ipari Ethernet alrendszerek valós idejű működésére vonatkozó (3.24) reláció.
3. Ha a frissítési idők jóval nagyobbak, mint a ciklusidő, akkor a válaszidők a legrosszabb esetben is két frissítési intervallum alatt megérkeznek és függetlenné válnak a ciklusidőtől:

$$L_{RT} = 2 \cdot U_T ; \quad \text{ha } U_T >> T_C \tag{5.7}$$

Általános formában az (5.1) és (5.6) relációk alapján kijelenthetjük, hogy *a nem szinkronizált ipari Ethernet rendszer bármely eszközvezérlőéhez tartozó kimeneti csatornák reakcióideje (T_R) a bementi késleltetéssel bővített kontroller minimális ciklusideje és a konfigurációban meghatározott frissítési idő négyszerese között véletlenszerűen változik.*

$$I_D + T_{C\,min} < T_R < 4 \cdot U_T \tag{5.8}$$

Mivel a Profinet IO rendszereknél a frissítési idő a küldési idő kettő hatványa szerinti szorzata, felírhatjuk, hogy:

$$U_T = 2^n \cdot S_T \tag{5.9}$$

Következik, hogy a (5.8) relációt kifejezhetjük úgy is mint:

$$I_D + T_{C\,min} < T_R < 2^{n+2} \cdot S_T ; \qquad n=\{0, 1, 2, \ldots, 9\} \tag{5.10}$$

Ha figyelembe vesszük, hogy a minimális bemeneti késleltetés kb. *0,1 ms*, a minimális ciklusidő és a küldési idő nem szinkronizált rendszereknél *1 ms*, akkor láthatjuk, hogy a valós idejű Profinet IO rendszerek reakcióideje elméletileg igen tág határok között konfigurálható. Vagyis az elméletileg lehetséges minimális válaszidő nem lehet kisebb, mint *1,1 ms*. Az elméletileg lehetséges felső határ pedig *2048 ms*-nál nem lehet nagyobb, alapértelmezett küldési idő esetén.

5.2. A válaszidők mérési módszereinek bemutatása [S2]

Az általam alkalmazott mérési eljárások, akárcsak a ciklusidő mérésénél, itt is két kategóriába sorolhatók. Az egyik az, amikor az IO kontroller vezérlési programja tartalmaz olyan funkcióblokkokat, amelyek szoftveres úton mérni tudják a reakcióidőt. Előnye ennek a módszernek, hogy a mérés nem igényel semmilyen külső eszközt, a kontroller pedig azonnal tud megfelelőképpen reagálni azokra az esetekre, amikor esetlegesen reakcióidő túllépés fordul elő. Akár egymástól távol levő eszközvezérlők esetében is alkalmazható. Hátránya, hogy a mérési pontosság a kontroller ciklusidejéhez igazodik, vagyis tulajdonképpen azt méri, hogy hány ciklusidő alatt érkezik meg a válasz. Ez viszont éppen elégséges, ha a mérés a biztonságos, valós idejű működést hivatott ellenőrizni.

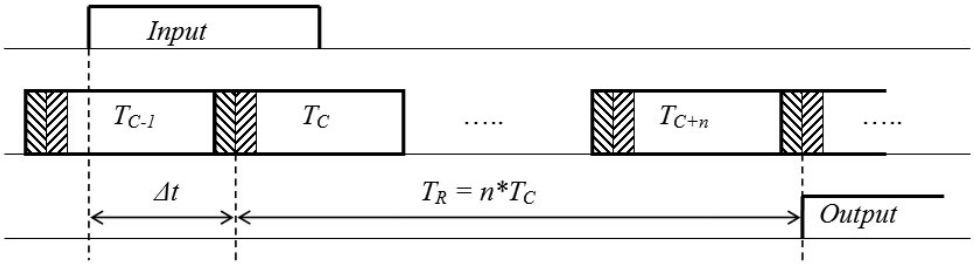
A másik csoportba azokat az eljárásokat sorolom, amelyek valamilyen külső eszközt használnak a válaszidők mérésére. Ezek közül két megoldást fogok bemutatni.

Az egyik, amikor a mérés univerzális számlálóval történik, a másik pedig amikor egy LabView 8.2. alkalmazás figyeli és rögzíti is a mérési adatokat. Ezeket egy NI interfész szolgáltatja, amelyik kapcsolatban van az IO vezérlő(k) megfelelő csatornáival. Ez utóbbi megoldásnak az a nagy előnye, hogy a mérési adatok tetszőleges mennyiségben rögzíthetők, kiértékelhetők és közvetlenül Excel táblázatba konvertálhatók. A mérések nagy pontossággal végezhetők. Hátrányt jelent viszont a külső eszközök használata és ezek beállításai. Inkább csak tesztrendszerben kivitelezhető, mert gondot jelenthet a csatornakapcsolat kiépítése az egymástól nagyobb távolságra elhelyezett eszközvezérlők esetében.

5.2.1. Mérés szoftveres úton

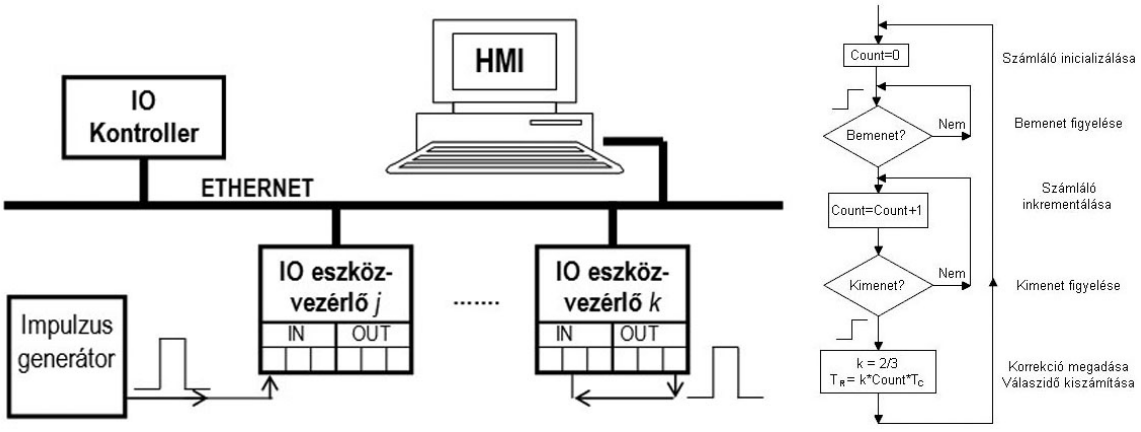
Ennél a módszernél készítettem egy funkcióblokkot (FB), amely a mérést végzi. Ezt a főprogramból hívjuk minden egyes alkalommal, amikor az adott bemenet válaszidejét akarjuk mérni. A mérési eredményeket akár adatblokkban is tárolhatjuk és HMI-vel megjeleníthetjük. A mérési elv azon alapszik, hogy megszámoljuk hányszor fut le az IO kontroller ciklusa a bemenet gerjesztési pillanata és a kimenet állapotváltozásáig eltelt idő alatt.

Ez a módszer akkor alkalmazható, amikor a ciklusidő jelentősen kisebb, mint a Profinet IO frissítési ideje, ugyanis ebben az esetben a mérési pontosságot a ciklusidő fogja meghatározni. A ciklusidőt a PLC milliszekundum pontossággal méri, de a 4.3. fejezetben bemutatott módszerrel az ennél pontosabb átlagértéket vehetjük alapul. A bemenet gerjesztésekor ugyan rövid ideig megnövekedhet a ciklusidő, de ez alapvetően nem befolyásolja a mérést. A mérési hiba (Δt) maximálisan egy ciklusidő lehet, amikor a gerjesztés a legkedvezőtlenebb időpillanatban érkezik a folyamati térképek aktualizálásához képest (5.7. ábra). Nagy előnye viszont, hogy a mérések során impulzus generátoron kívül nem kell használnunk semmilyen más eszközt.



5.7. ábra. A szoftveres mérés elve

Azért, hogy a tényleges kimenti állapotváltozást észlelni tudjuk, az adott kimenetet közvetlenül egy tetszőleges bemenetre irányítjuk. Ez viszont 1/3 egységgel megnövelné a mért időt. Ezért ezt egy 2/3-os korrekciós tényezővel kompenzálom. Ez a megoldás látható a hozzákapcsolódó algoritmussal együtt a következő ábrán. (5.8. ábra).

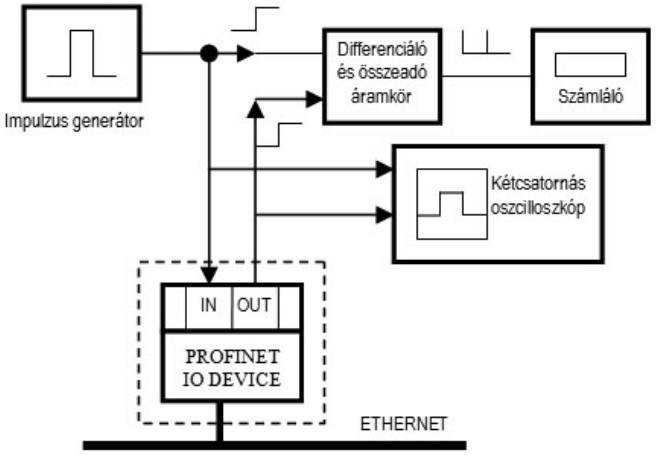


5.8. ábra. Válaszidő mérése szoftveres úton

A bemenetek gerjesztése 0,5-5 s között véletlenszerűen előállított 24V amplitúdójú, 10-500ms-os változtatható szélességű impulzusokkal történik. A véletlenszerű impulzusgenerálás azért szükséges, hogy minél valóságosabb helyzetet lehessen modellezni. Ezeket az impulzussorozatokat a LabView 8.2. fejlesztőkörnyezetben generáltam, és az NI interfész egyik kimeneti csatornájára irányítottam.

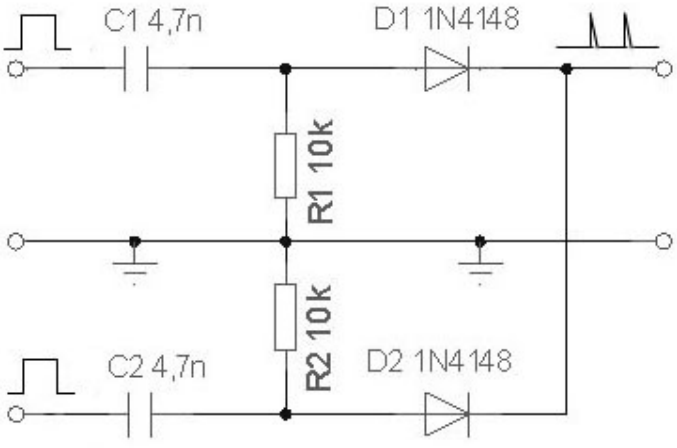
5.2.2. Mérés univerzális számlálóval

Ennél a mérési eljárásnál egy hagyományos, meglehetősen régi típusú, viszont nagy pontosságú univerzális számlálót alkalmaztam. A bemenetek gerjesztésére az imént már bemutatott impulzus generátort használtam. A bemenetre érkező és a kimeneten megjelenő feszültségváltozást egy-egy differenciáló áramkört követően összeadtam és az univerzális számláló bemenetére kapcsoltam. A vizualizálás kedvéért egy kétcsatornás oszcilloszkópot is bekötöttem (5.9. ábra).



5.9. ábra. A válaszidők mérése univerzális számlálóval

A differenciáló áramkör segítségével a négyszögjelből tűimpulzusokat kapunk. Ezek azért szükségesek, hogy egyrészt detektálni lehessen a bementi illetve a kimeneti villamos jel felmenő élét, másrészt összeadható legyen a két jel. Így a számláló mindig a két egymást követő tűimpulzus között eltelt időt fogja mérni. Az áramkör kapcsolási rajza a 5.10. ábrán látható.

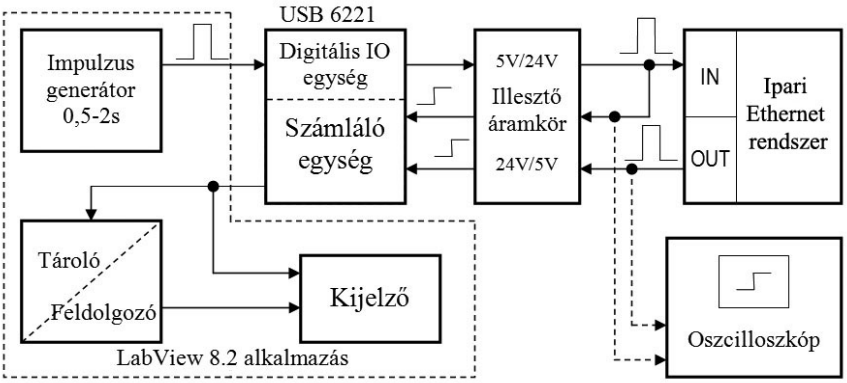


5.10. ábra. Differenciáló és összeadó áramkör

Ennek a mérési módszernek az a nagy hátránya, hogy a mért értékek csak manuálisan rögzíthetők, illetve tárolhatók. Ezért kidolgoztam egy sokkal korszerűbb eljárást, amely ugyanezen a mérési elven alapszik, de lehetőség nyílik az adatok tárolására, akár bizonyos fokú feldolgozására is.

5.2.3. Mérés LabView NI interfésszel

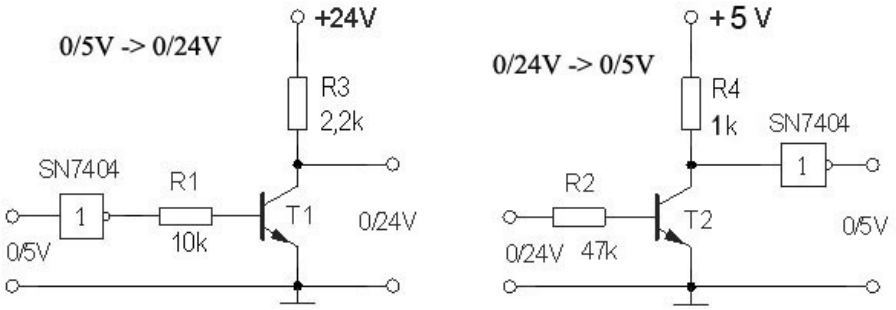
A mérést a 4.4. fejezetből már ismert NI interfész, speciálisan ilyen célra kialakított része végzi. A mérőkártya két számlálóbemenetére csatlakozik a bemeneti illetve a kimeneti jel. A két bemenet felfutó élfigyelésre van beállítva. Differenciáló, illetve összeadó áramkörre már nincs szükség. A két felfutó él közötti időintervallumban történik a számlálás. A mérési adatokat egy egydimenziós tömb tárolja, amely alapértelmezésben 100, de akár több 1000 mért érték tárolására is alkalmas. A kijelzőn mindig az éppen aktuális válaszidő olvasható. (5.11. ábra).



5.11. ábra. Az NI interfészes válaszidő mérés fontosabb egységei

A mérés teljesen automatikusan történik. Csak a mérőimpulzusok szélességét kell megadnunk, valamint a minták számát, vagyis azt, hogy hány mérést végezzen. A mérés indítása után minden egyes eredmény azonnal megjelenik a kijelzőn és a tömb következő rekordjában tárolódik. Miután az összes mérési eredményt elmentette, a feldolgozó egység megkeresi a szélsőértékeket (maximum, minimum), kiszámítja az átlagot, a jittert, valamint a relatív eltérést. Ezeket ugyancsak tárolja, és végül, ha konvertálási opció be van kapcsolva, Excel formátumba is átalakítja [S1].

Az illesztő áramkörök feladata a jelszintek átalakítása. Az USB6221-es interfész digitális illetve számláló egységei 0-5V-os TTL, míg az IO eszközvezérlő bemeneti és kimeneti csatornái a 0-24V-os ipari feszültségszintekkel dolgoznak [55]. Az illesztő áramkörök által okozott késleltetési idő elhanyagolható. Az illesztő áramkörök egy-egy csatornájának kapcsolási rajza az 5.12. ábrán látható.



5.12. ábra. A 0/5 – 0/24 V és a 0/24 – 0/5 V illesztő áramkörök kapcsolási rajza

5.3. A válaszidők mérési eredményeinek elemzése

Az előzőekben bemutatott mérési eljárások közül, a továbbiakban az 5.11. ábra szerinti, LabView-val rögzített mérési eredményeket fogom elemezni és kiértékelni, azért mert ezzel a módszerrel az adott szituációban akár több ezer eredményt is tárolni lehet. A mérésekhez pedig a 4.2.3. alfejezetben bemutatott Profinet IO tesztrendszert használtam.

A kapott mérési eredmények segítségével igazolhatók az előző alfejezetekben meghatározott kritériumok az IO csatornák reakcióidejét illetően. Ezek ismeretében határozható meg, hogy egy nem szinkronizált ipari Ethernet alrendszernél milyen frissítési időket kell beállítani a konfiguráció során azért, hogy a valós idejű működés minden körülmények között biztosítható legyen.

Ebben a fejezetben a mérések során a következő kommunikációs szituációkban vizsgáltam a reakcióidők alakulását:

1) *egyirányú kommunikáció:*
   a) IO kontroller bemeneti csatornája gerjeszti az IO eszköz kimenti csatornáját,
   b) IO eszköz bemeneti csatornája gerjeszti az IO kontroller (PLC) kimeneti csatornáját.
2) *kétirányú kommunikáció:*
   a) adott IO eszköz bemeneti csatornája gerjeszti ugyanazon IO eszköz kimenti csatornáját,
   b) adott IO eszköz bemeneti csatornája egy másik IO eszköz kimenti csatornájára küld adatokat.

A gerjesztett bemenetek közvetlenül a kimenetekre vannak irányítva, tehát az adatokkal semmilyen műveletvégzés nem történik. Az IO kontroller felhasználói programja nem bonyolult, így a ciklusidő csak ritkán közelíti meg a *2ms*-ot. Általában ez *1 ms* körüli érték, vagy kevéssel ez alatt van (*0,96ms*, lásd 4.5.4. fejezet). A vizsgált frissítési idők: *128, 64, 32, 16, 8, 4, 2 és 1 ms*.

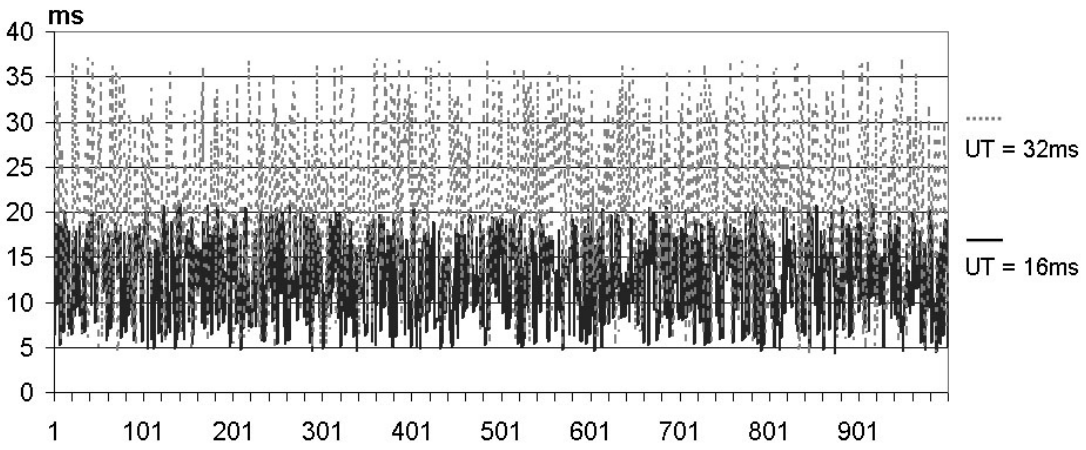
Az IO eszköz bementi csatornáinak késleltetése (I_D) alapértelmezésben *3 ms*-ra van beállítva, de a legkiélezettebb helyzetekben *0,5 ms*-os beállítást is használtam. Az IO kontroller bemenetein ez az érték minden esetben *3 ms* és nem módosítható.

A továbbiakban a mérések során a nagyszámú mintavételezésből esetenként ezer-ezer minta kerül rögzítésre és kielemzésre.

5.3.1. Egyirányú kommunikáció válaszidejének mérése

Először azt az esetet vizsgáltam, különböző frissítési idők beállítása mellett, amikor a reakciót kiváltó bemeneti jel az IO kontroller, továbbiakban PLC valamelyik bemenetére érkezik, a válasz pedig az IO_1 és/vagy az IO_2 eszközvezérlők valamelyik kimeneti csatornáján jelenik meg. A két eszközvezérlő fizikailag teljesen azonos, de különféleképpen konfigurálható. Ennek megfelelően elsőként egy átlagosnak tekinthető szituációt elemzek, amikor az IO_1 frissítési ideje *16 ms*, az IO_2-é pedig *32 ms*. A bemeneti késleltetések *3 ms*-ra vannak beállítva azért, hogy igazodjanak a PLC bemeneti moduljában, egyébként nem konfigurálható értékekhez. A kontroller ciklusideje a mérések ideje alatt átlagosan *0,9 ms* volt.

A mindkét frissítési időhöz tartozó válaszidők alakulása az 5.13. ábrán látható. A lehetséges maximum és minimum értékek közötti véletlenszerű változás jól látható mindkét esetben. Azt is megfigyelhetjük, hogy a minimum értékek a frissítési idő megkétszerezése ellenére is közel azonosak maradnak. Ugyanis a minimum érték az (5.1) reláció szerint független a ciklusidőtől.



5.13. ábra. A válaszidők változásának alakulása egyirányú kommunikációnál

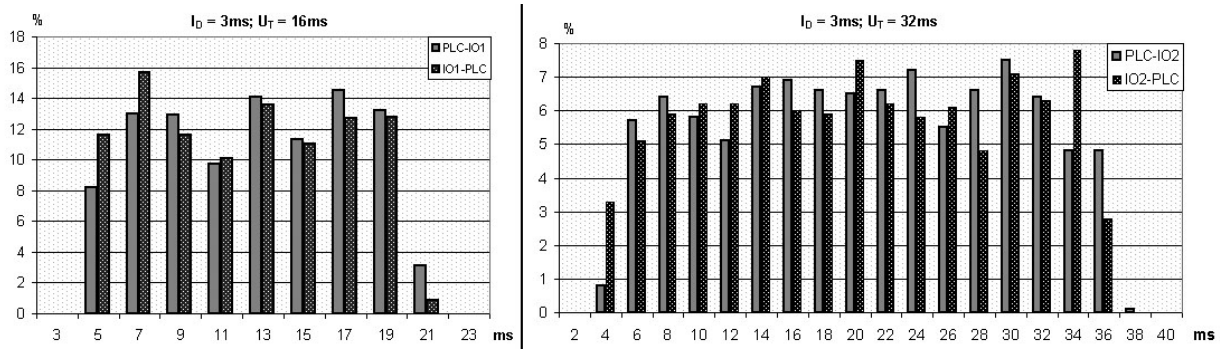
Ha megváltoztatjuk az adatáramlás irányát, vagyis a bemeneti jel az IO_1 vagy IO_2 eszközvezérlők valamelyikére érkezik és a válasz pedig a PLC egyik kimeneti csatornáján fog megjelenni, megtartva az előbbi konfigurációt, várhatóan hasonló értékeket fogunk kapni. Ez látható a következő 5.1. táblázatban is, valamint azok a fontosabb jellemző értékek, amelyek segítségével összehasonlítható a két szituáció.

5.1. táblázat. A válaszidők jellemző értékei 16 és 32 ms-os frissítési idők esetében

Jellemző értékek	U_{TIO1} = 16 ms				U_{TIO2} = 32 ms			
	PLC → IO_1		IO_1 → PLC		PLC → IO_2		IO_2 → PLC	
	mért	elméleti	mért	elméleti	mért	elméleti	mért	elméleti
Maximum [ms]	21,066	21,8	20,337	21,8	37,005	37,8	36,318	37,8
Minimum [ms]	4,27	3,9	4,025	3,9	4,226	3,9	3,994	3,9
Átlag [ms]	12,712	12,850	12,061	12,850	20,783	20,850	20,278	20,850
Jitter [ms]	16,796	17,9	16,312	17,9	32,779	33,9	32,324	33,9
Relatív eltérés	66,06%	69,65%	67,62%	69,65%	78,86%	81,29%	79,70%	81,29%

A fenti táblázatban az elméleti értékeket az 5.1.2. (5.1) valamint az 5.1.3.1. bekezdésben bemutatott (5.2'), (5.3'), (5.4') és (5.5') relációk szerint határoztam meg.

A szélsőértékek közötti véletlenszerű válaszidő eloszlást az 5.14. ábrán láthatjuk. Az eloszlást illetően egyértelműen látszik, hogy a frissítési idő megnövelésével a két milliszekundumonkénti eloszlási gyakoriság jelentősen csökken, átlagosan mintegy 11%-ról kevesebb, mint 6%-ra. Átlag alatti eloszlások inkább csak a szélsőértékek közelében tapasztalható.



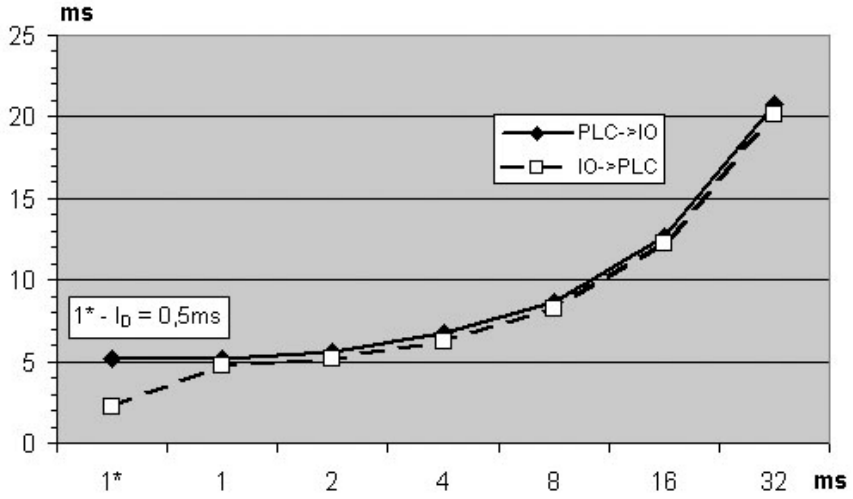
5.14. ábra. A válaszidők eloszlásának alakulása 16 és 32 ms-os frissítési idők szerint

A következő 5.2. táblázatban további válaszidőket jellemző értékeket találunk alacsonyabb frissítési idők szerint, mindkét irányban. A táblázat utolsó oszlopai a *0,5 ms* bemeneti késleltetésnek megfelelő értékeket tartalmazzák.

5.2. táblázat. A válaszidők jellemző értékei 1 és 8 ms-os frissítési idők esetében

Jellemzők [ms]	I_D = 3 ms												I_D = 0,5 ms		
	U_{T2} = 8 ms			U_{T1} = 4 ms			U_{T2} = 2 ms			U_{T1} = 1 ms			U_{T1} = 1 ms		
	PLC-IO2	IO2-PLC	Elm. ért.	PLC-IO1	IO1-PLC	Elm. ért.	PLC-IO2	IO2-PLC	Elm. ért.	PLC-IO1	IO1-PLC	Elm. ért.	PLC-IO1	IO1-PLC	Elm. ért.
Max.	13,099	12,301	13,800	9,393	8,552	9,800	7,312	6,410	7,800	6,152	6,525	6,800	x	3,112	4,300
Min.	3,983	3,979	3,900	4,316	3,962	3,900	4,027	3,967	3,900	4,048	3,923	3,900	x	1,451	1,400
Átl.	8,613	8,178	8,850	6,751	6,206	6,850	5,576	5,141	5,850	5,135	4,717	5,350	x	2,249	2,850
J	9,116	8,322	9,900	5,077	4,590	5,900	3,285	2,544	3,900	2,104	2,602	2,900	x	1,661	2,900
%	52,92	50,88	55,93	37,60	36,98	43,07	29,46	24,74	33,33	20,49	27,58	27,10	x	36,93	50,88

Megjegyzés: A PLC → IO irányban mért értékek valamelyest nagyobbak. Ez azért van, mert az SN323 IO modul bementi késleltetésének tipikus értéke ugyan 3 ms, de a katalógus adatok szerint ez 1,2 – 4,8 ms közötti érték lehet [56]. Ez látható az 5.15. ábrán.



5.15. ábra. A válaszidők átlagértékének alakulása mindkét irányban

Az 5.15. ábrán az *1 ms** jelölt részben láthatjuk a legkiélezettebb helyzetet is, amikor az IO eszközvezérlő bemeneti késleltetése nem *3*, hanem *0,5 ms*. Ebben az esetben ez akár 50%-al is csökkentheti a válaszidőket. A mérések során rögzített legkisebb válaszidő *1,451 ms* volt, vagyis alig több mint az elméletileg lehetséges legrövidebb válaszidő (*1,4 ms*). A jitter is csökkent, viszont nőtt a relatív eltérés százalékos értéke. (5.2. táblázat)

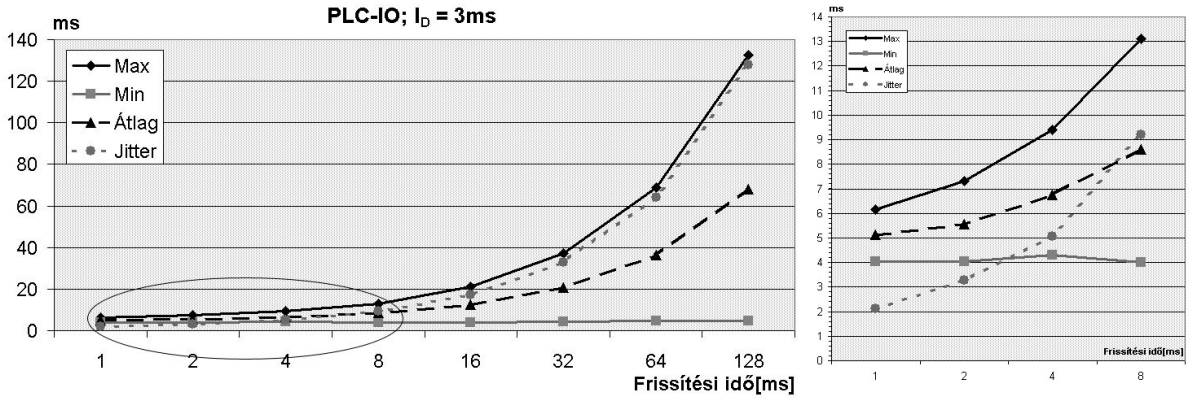
A két táblázat alapján az alábbi következtetéseket érdemes megjegyezni:

- Az elméletileg meghatározott jellemző értékek helyességét a mérési eredmények kétségtelenül igazolják.
- Ha a bemeneti paraméterek azonosak, az egyirányú kommunikáció során a válaszidők nagysága nem függ attól, hogy melyik eszköz tölti be az termelő illetve a fogyasztó szerepét.
- A minimum közeli értékek nagyon keveset változnak a frissítési idő növelésével.
- Az előbbiből következik, hogy a válaszidők átlagértéke sem nő olyan arányban, mint a frissítési idő.

- A frissítési idő növelése a legkedvezőtlenebb hatást a jitterre gyakorolja. Ez gyakorlatilag arányosan nő a frissítési idővel.

Az 5.2. táblázatban az elméleti értékek $T_C = 0{,}9$ *ms*-al lettek meghatározva. A több ezernyi rögzített adat alapján az 5.1.4. alfejezet (5.7) relációja már 8 ms-os frissítési időtől igaznak bizonyul, vagyis *ha a frissítési idő jóval nagyobb, mint a vezérlő ciklusideje, akkor a válaszidők nem haladják meg a frissítési idő kétszeresét.*

A teljes adattábla alapján az egyirányú kommunikáció fontosabb jellemző értékeinek alakulását az 5.16. ábra mutatja.



5.16. ábra. A válaszidőket jellemző értékek alakulása a frissítési idő szerint

Az 5.16. ábrán jól látható, hogy a minimális válaszidő esetében szinte alig tapasztalható növekedés, sőt bizonyos esetekben, akár kisebb is lehet, mint például *8 ms*-nál. Ebből arra lehet következtetni, hogy akár nagyobb értékű frissítési idő esetében is, ugyan jóval kisebb valószínűséggel, de előfordulhatnak az elméleti minimum értékhez közeli válaszidők. Így például *128 ms*-os frissítési időnél is előfordult *4,181 ms*-os válaszidő *3 ms* bemeneti késleltetés mellett.

5.3.2. A kétirányú kommunikáció válaszidejének elemzése a mérési eredmények alapján

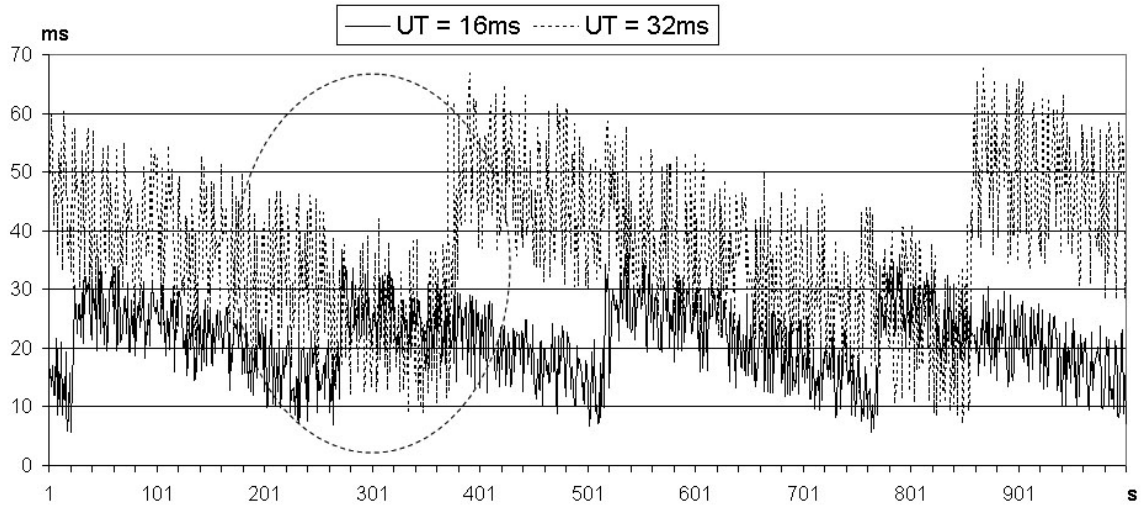
Az IO eszközvezérlők bementi és kimeneti csatornái közötti adatátvitel kizárólag csak kétirányú kommunikáció felépítésével valósulhat meg. Vagyis a PLC és az IO eszközvezérlő(k) között mindkét irányban létrejön a kommunikációs reláció. Az 5.1.2. fejezetben leírtak alapján a válaszidőket meghatározó alapvető tényező a két frissítési idő lesz. Ha a bemeneti és kimeneti csatornák ugyanahhoz az IO eszközvezérlőhöz tartoznak, akkor ezek egyenlő periódusúak, és a jellemző értékeket az (5.1), illetve a (5.2'), (5.3'), (5.4'), (5.5') relációkkal határozhatjuk meg. Ha pedig a legáltalánosabb esetet vesszük figyelembe, amikor az IO csatornák tetszőleges eszközvezérlőkhöz tartoznak, akkor a frissítési intervallumok is különbözőek lehetnek és a jellemző értékeket az (5.1), (5.2), (5.3), (5.4), (5.5) relációk szerint számítjuk.

Egy adott IO eszközvezérlő válaszidejének vizsgálata

Akárcsak az 5.3.1. fejezetben, itt is először vizsgálni fogok két átlagosnak tekinthető esetet *16* illetve *32 ms* frissítési periódus esetében. A bementi késleltetés *0,5* és *3ms*, a kontroller ciklusideje *0,9 ms*. Minden egyes mérési szituációban 1000-1000 minta került rögzítésre és kiértékelésre.

A következő ábrán (5.17. ábra) a válaszidők alakulását láthatjuk mindkét frissítési intervallum esetében, *3 ms*-os bementi késleltetésnél. Akárcsak az egyirányú

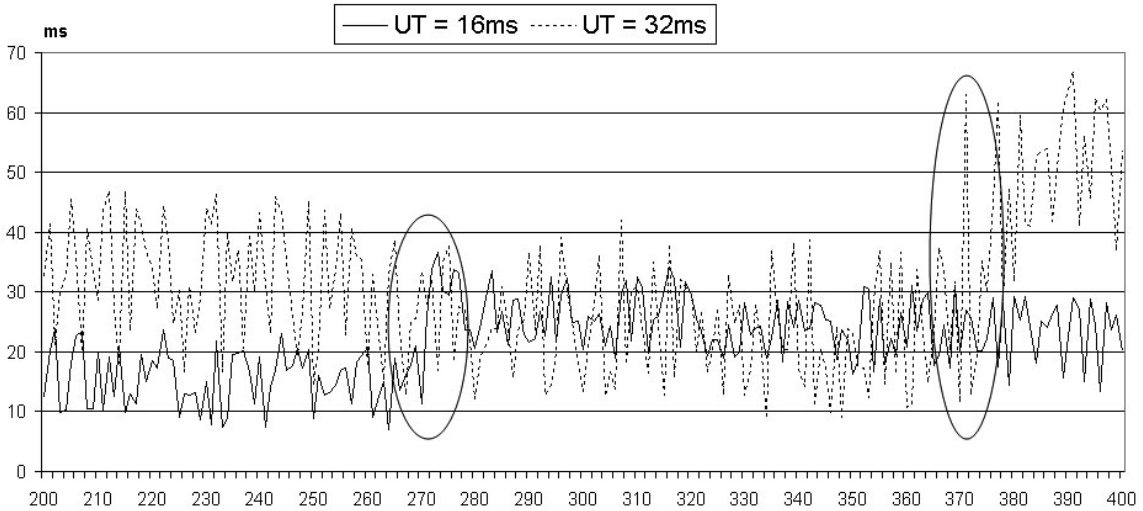
kommunikáció esetében, itt is a legrövidebb válaszidők közel azonos értékűek lesznek (5,65 és 7,16 ms), noha a frissítési idő duplájára nő.



5.17. ábra. A válaszidők alakulása kétirányú kommunikáció esetében

Megfigyelhető, hogy mindkét frissítési időnél a válaszidők csökkenő tendenciát mutatnak, majd hirtelen egy-egy jelentős ugrás tapasztalható, amely periodikusan ismétlődik. Azt is vegyük észre, hogy ez a periódus arányosan növekszik a frissítési idővel. A jelenség a szinkronizálás hiányával magyarázható, ugyanis a kontroller és az eszközvezérlő órajelei nincsenek egymáshoz szinkronizálva. Ennek következtében a két kommunikációs reláció frissítési ciklusa is egymáshoz képest „csúszik”. Az ugrás akkor következik be, amikor az a szituáció áll elő, hogy az adattovábbítás éppen lekési valamelyik küldési időt.

Az 5.17. ábra megjelölt részét kinagyítottam, hogy jobban látható legyen mi is történik. Ezt szemlélteti az 5.18. ábra.



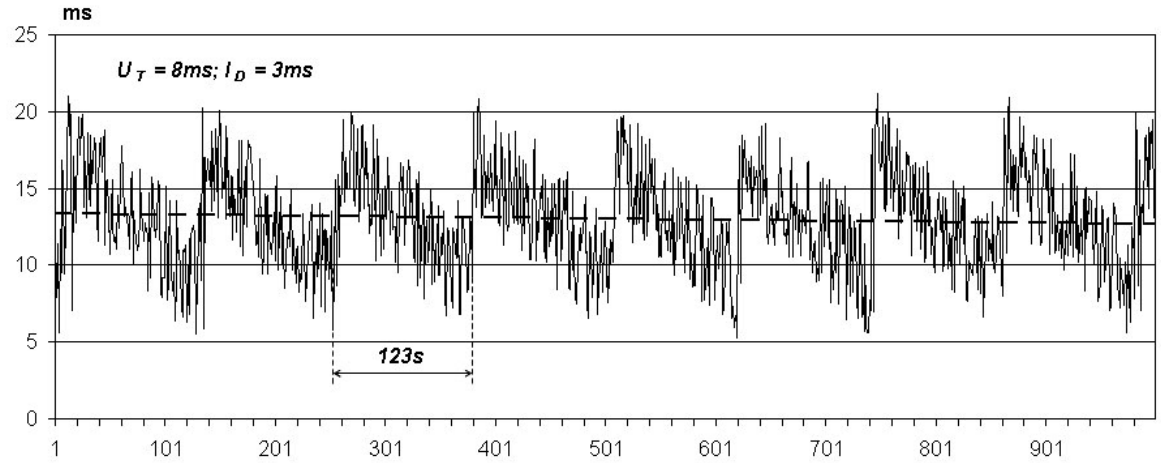
5.18. ábra. Válaszidő ugrás a küldési idő elvesztésekor

A megjelölt részeken mindkét frissítési időhöz tartozó görbén egy-egy ugrás látható, amelynek értéke minden esetben a frissítési intervallumhoz közeli, de legtöbb esetben annál nagyobb mértékű. Az $U_T = 16$ *ms*-os görbéhez tartozó 270 és 271-edik minta között

18,24 ms, míg a 370-371-edik minta között *51,48 ms* a különbség. Ez utóbbi az $U_T = 32$ *ms*-os görbéhez tartozik.

Az 5.17. ábrához tartozó adattáblából pontosan kiolvasható, hogy az $U_T = 16$ *ms*-os görbénél a 24-edik mintától kezdődően 249-251 mintánként következik be az ugrás. A másik esetben 370-371 és 853-854 történik az ugrás. Vagyis kétszerannyi mintánként, mint az első esetben. Ha figyelembe vesszük, hogy a mérések 0,5 – 1,5 másodpercenként történtek, vagyis átlagosan egy másodpercenként, ebből következik, hogy a két frissítési periódus egymáshoz képest körülbelül *65 μs*-al csúszik. Más szóval a két frissítési idő periódusa közötti fázisidő másodpercenként *65 μs*-al változik.

Ugyanez a jelenség tapasztalható 8 ms-os frissítési időnél is (5.19. ábra). Ebben az esetben az ugrások 122-124 mintánként fordulnak elő, amely úgyszintén *65 μs*-os csúszást jelent. Vagyis független a beállított frissítési időktől.



5.19. ábra. Szinkronizálás hiányában a válaszidőknél periodikusan fellépő ugrás

A következő táblázatban (5.3. táblázat) annak a mérési szituációnak a fontosabb jellemzői vannak összefoglalva, amikor a mérésben résztvevő bemeneti és kimeneti csatornák ugyanahhoz az IO eszközvezérlőhöz tartoznak. Az IO_1 és IO_2 eszközvezérlőinek frissítési ideje rendre 1-, 2 ms; 4-, 8 ms illetve 16-, 32 ms értékre vannak beállítva. A kontroller átlagos ciklusideje 0,9 ms. A táblázat utolsó oszlopai itt is egy rövidebb bementi késleltetési idő szerinti értékeket tartalmazzák.

5.3. táblázat. A válaszidők jellemző értékei kétirányú kommunikációnál.

Jellemzők [ms]	***I_D = 3 ms***												***I_D = 0,5ms***	
	U_{T2} = 32 ms		**U_{T1} = 16 ms**		**U_{T2} = 8 ms**		**U_{T1} = 4 ms**		**U_{T2} = 2 ms**		**U_{T1} = 1ms**		**U_{T1} = 1ms**	
	IO2-IO2	**Elm. ért.**	**IO1-IO1**	**Elm. ért.**	**IO2-IO2**	**Elm. ért.**	**IO1-IO1**	**Elm. ért.**	**IO2-IO2**	**Elm. ért.**	**IO1-IO1**	**Elm. ért.**	**IO1-IO1**	**Elm. ért.**
Max.	67,668	70,82	36,715	38,82	21,131	22,800	13,252	14,800	9,202	10,800	7,287	8,800	4,820	6,300
Min.	7,161	3,91	5,658	3,91	5,296	3,900	5,344	3,900	5,246	3,900	5,300	3,900	2,833	1,400
Med.	37,172	37,365	21,252	21,365	13,011	13,350	9,198	9,350	7,099	7,350	6,135	6,350	3,671	3,850
Jitter	60,507	66,91	31,057	34,91	13,842	18,900	7,908	10,900	3,956	6,900	1,987	4,900	1,987	4,900
[%]	81,39	89,54	73,07	81,70	60,88	70,79	42,99	58,29	27,86	46,94	16,19	38,58	27,06	63,64

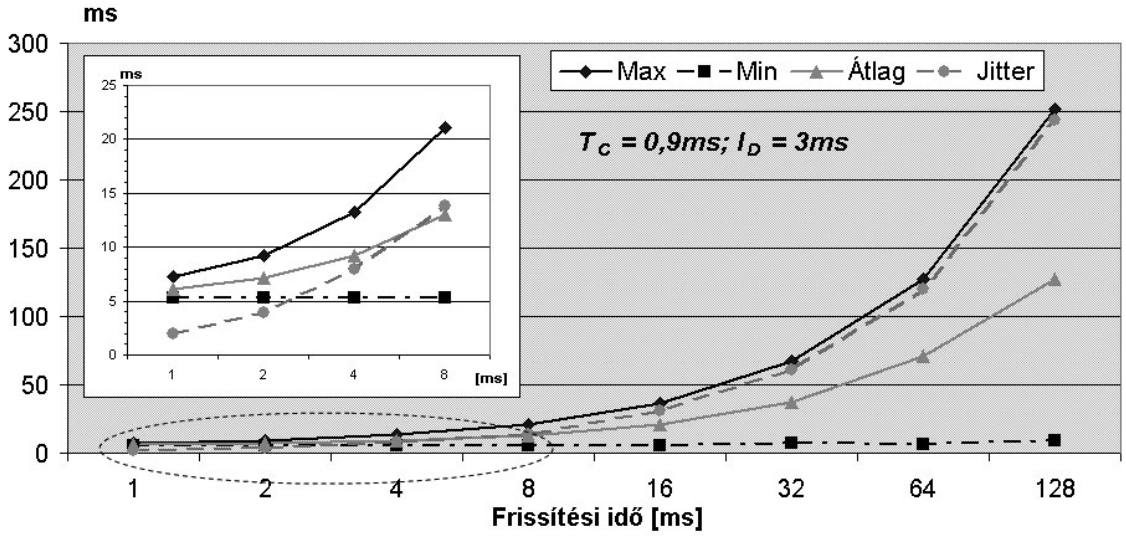
A táblázatban rögzített eredmények ismételten alátámasztják az elméletileg meghatározott számításokat. A szürke oszlopokban az elméleti értékek meghatározása az

5.1.2. fejezet (5.1) illetve az 5.1.4. fejezet (5.2”), (5.3”), (5.4”) és (5.5”) relációi alapján történt. *Több ezer mérés után egyetlen egy esetben sem fordult elő, hogy a mért érték valamelyik, de főleg a legkedvezőtlenebb esethez tartozó számított értéket túllépte volna.* Következtetésképpen igazolást nyert a fenti relációk helyessége, a kapott értékek pedig az 5.1.4. fejezet (5.8) reláció alapján megfogalmazott állításnak is eleget tesznek.

A várható átlag érték is csak 1-2 tized milliszekundummal tér el a számított értéktől és minden esetben kisebb tőle. Jelentősebb különbség a jitter és a relatív eltérés százalékos értékénél tapasztalható, de ezek is kedvező irányban alakulnak. Ennek az a magyarázata, hogy a válaszidők csak ritkán közelítik meg a számított szélsőértékeket. A jitter legkedvezőbb értékét *1 ms* frissítési idő mellett kaptuk, viszont itt tapasztalhatjuk a legnagyobb különbséget is az elméletileg számított értékhez viszonyítva. Mivel ez független a bemeneti késleltetéstől, ezért mindkét esetben azonos értéket kaptam, holott a szélsőértékek különbözőek voltak. A legkisebb relatív eltérés ugyancsak *1 ms*-nál tapasztalható, mégpedig a *3 ms*-os bemeneti késleltetésnél.

Az 5.3. táblázatból az is látható, hogy a legkiélezettebb helyzetben (U_T = *1 ms*; I_D = *0,5 ms;* T_C = *0,9 ms*) a valós idejű működés időkorlátja *5 ms*-on belül van, a jitter pedig valamivel kevesebb, mint a frissítési idő kétszerese. A jitter esetében ez az arány a további frissítési időknél is megmarad.

A következő ábrán adott IO eszközvezérlő válaszidejéhez fűződő fontosabb jellemzők alakulását látjuk a frissítési idő függvényében, kétirányú kommunikáció esetében (5.20. ábra).

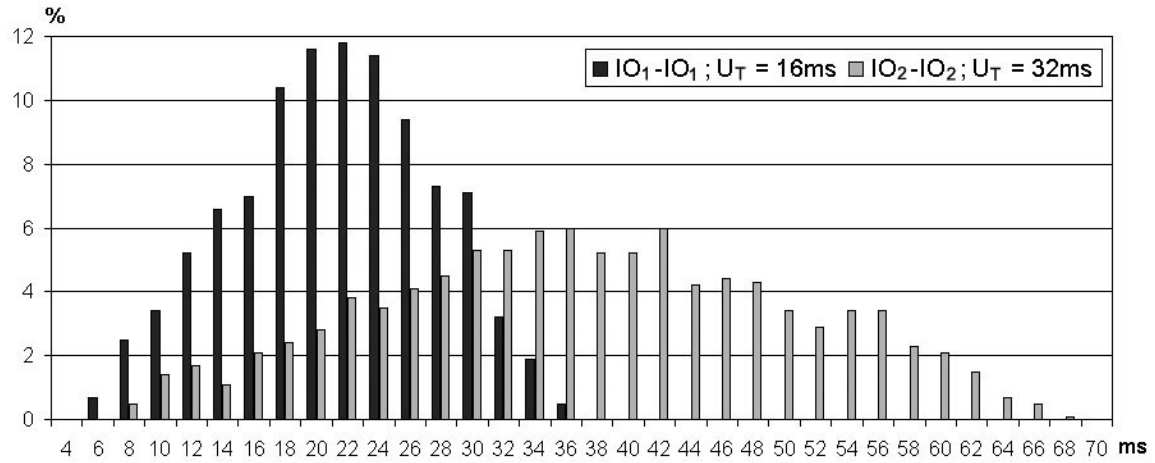


5.20. ábra. A válaszidőket jellemző értékek alakulása kétirányú kommunikációban

A minimális válaszidők ebben az esetben is szinte azonos értékűek. Egészen *16 ms*-ig *5,2 – 5,7 ms* között van, de a nagyobb értékű frissítési időknél sem növekszik *2 – 3 ms*-nál nagyobb értékkel. *128 m*s-nál is csak *8,74 ms*. Ebből következik, hogy a jitter szinte párhuzamosan követi a maximális értéket, amely meghatározó a rendszer valós idejű működése szempontjából. Ezeknek az értékeknek az ismerete igen fontos szerepet játszik a rendszer konfigurálása során. Így például, ha egy olyan rendszert szeretnénk konfigurálni, amelynek határideje *15 ms*, akkor az IO eszközvezérlőhöz tartozó frissítési időt legfeljebb *4ms*-ra lehet beállítani, *3 ms*-os bementi késleltetés mellett.

A maximum és minimum értékek közötti válaszidő eloszlást illetően az tapasztalható, hogy ezek az értékek sokkal inkább koncentrálódnak az átlagérték környezetében, mint az egyirányú kommunikáció esetében, viszont sokkal szélesebb

időintervallumon oszlik meg és az eloszlási gyakoriság is alacsonyabb. Ezeknek a változása látható az 5.21. ábrán.



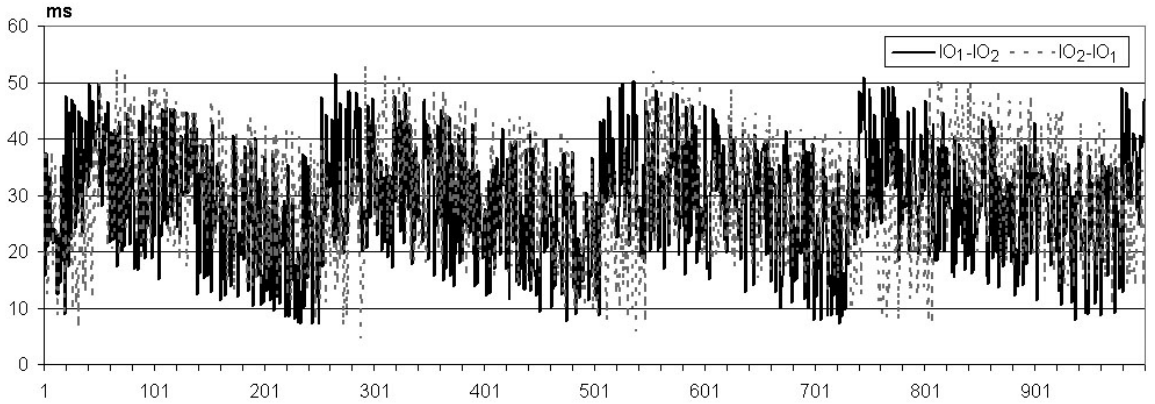
5.21. ábra. A válaszidők eloszlásának alakulása a frissítési idő szerint

A fenti ábrán jól látható, hogy ha a frissítési periódust duplájára növeljük az eloszlási intervallum is ennek arányában növekszik, a két milliszekundumonkénti gyakoriság százalékos értéke pedig a felére csökken.

Két tetszőlegesen konfigurált IO eszközvezérlő egymás közötti válaszideje

Ebben a fejezetben a kétirányú kommunikáció válaszidejére vonatkozó, az 5.1.2. fejezetben általánosan meghatározott jellemző értékek alakulását és ezek helyességét vizsgálom a már előzőekben ismertetett tesztrendszer és mérési eljárás segítségével. Különböző frissítési idők szerint azt vizsgálom, hogy van-e lényeges különbség a között, hogy melyik eszközvezérlő tölti be az termelő, illetve a fogyasztó szerepét. Ennek érdekében először IO_1 egyik bemeneti csatornájára irányítom a tesztimpulzus, a választ pedig IO_2 eszközvezérlő kimenetén kapom, majd az IO_2-t gerjesztem és az IO_1-en fogadom a választ. A kontroller ciklusideje továbbra is *0,9 ms*, a bemeneti késleltetés *3 ms*, a frissítési idők pedig $U_{T1} = 16$ *ms* illetve $U_{T2} = 32$ *ms* értékekre vannak beállítva.

Mindkét esetben 1000-1000 mintát rögzítettem, 0,5-1,5 s időközönként. A válaszidők alakulását ebben az esetben az 5.22. ábra mutatja.



5.22. ábra. Az egymás közötti válaszidők alakulása 16 – 32 ms-os frissítési időknél

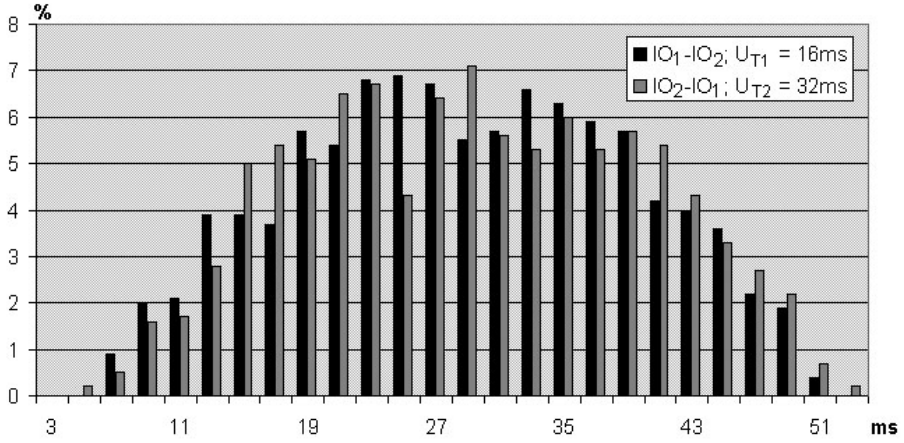
A két eset teljesen hasonló válaszidő változást mutat. Ezt a hasonlóságot megerősítik az 5.4. táblázat adatai is, amely a fontosabb jellemző értékeket tartalmazza, különböző szituációkban. A táblázat adatainak rögzítésekor a ciklusidő átlagosan *0,91 – 0,52 ms* között változott. Az elméleti értékek kiszámításánál $T_C = 0{,}9$ *ms*-ot használtam. A bemeneti késleltetés *3 ms* volt.

5.4. táblázat. Tetszőlegesen konfigurált IO eszközvezérlők fontosabb jellemző értékei

Jellemzők [ms]	U_{T1}/U_{T2} = 2/8 ms			U_{T1}/U_{T2} = 16/32 ms			U_{T1}/U_{T2} = 64/128 ms			U_{T1}/U_{T2} = 256/512 ms		
	Mért [ms]		Elm. [ms]	Mért [ms]		Elm. [ms]	Mért [ms]		Elm. [ms]	Mért [ms]		Elm. [ms]
	IO_1-IO_2	IO_2-IO_1		IO_1-IO_2	IO_2-IO_1		IO_1-IO_2	IO_2-IO_1		IO_1-IO_2	IO_2-IO_1	
Maximum	15,176	15,303	16,8	51,53	52,765	54,8	193,744	191,847	198,8	709,187	767,402	774,8
Minimum	4,748	4,556	3,9	7,241	4,845	3,9	9,577	8,361	3,9	39,626	13,03	3,9
Átlag	10,103	9,891	10,35	28,892	29,284	29,35	100,877	97,650	101,35	377,365	378,681	388,95
Jitter	10,428	10,747	12,9	44,289	47,92	50,9	184,167	183,486	194,9	669,561	754,372	770,9
ε_R [%]	51,61%	54,33%	62,31	76,65	81,82	86,71	91,28%	93,95%	96,15	88,72%	99,61%	99,10

Az 5.22. ábrán is tapasztalható a válaszidők enyhe csökkenési tendenciája, majd egy idő után a hirtelen ugrás, amely ebben az esetben is az előző fejezetben már bemutatott frissítési időciklusok közötti csúszással magyarázható.

Érdemes még összehasonlítani a két eset válaszidő eloszlását a maximum és minimum értékek között. A két milliszekundumonkénti értékek hasonló eloszlást mutatnak mindkét esetben (5.23. ábra). A legnagyobb eltérés 24 és 26 ms között látható. Ez 2,6% eloszláskülönbséget mutat. A legtöbb intervallumban ez mindössze 1% vagy ez alatti érték. Az átlagértékhez viszonyított ±16 ms-os intervallumban található az esetek közel 90%-a.



5.23. ábra. Az egymás közötti válaszidők eloszlása mindkét szituációban

Az elvégzett mérések arra engednek következtetni, hogy *a tetszőlegesen konfigurált IO eszközvezérlők közötti kétirányú kommunikációban, az eszközök egyenrangúak, és bármelyik egyaránt betöltheti az termelő illetve a fogyasztó szerepét.*

Két azonosan konfigurált IO eszközvezérlő válaszideje

A vezérléstechnikában számos esetben előfordulnak olyan szituációk, amikor több teljesen azonos távoli eszköz vezérlését kell megoldani. Ilyen esetekben indokolt lehet a távoli eszközvezérlők teljesen azonos konfigurálása. Ez azt jelenti, hogy azonosak a bemeneti késleltetések és a kommunikációs relációk frissítési ideje is. Ebből viszont az következik, hogy két vagy több azonosan konfigurált, távoli eszközvezérlő a reakció idők

szempontjából nézve úgy fog viselkedni, mintha az IO csatornák ugyanahhoz az eszközvezérlőhöz tartoznának. Ezt alátámasztják az 5.1.2. fejezet (5.1), (5.2), (5.3), (5.4), (5.5) valamint az 5.1.3.2. fejezet (5.2"), (5.3"), (5.4"), (5.5") relációi is.

A továbbiakban ezeket a megállapításokat mérésekkel fogom igazolni. A már előzőekben bemutatott Profinet IO tesztrendszer IO_1 és IO_2 eszközvezérlőit teljesen azonos paraméterekkel konfigurálom. A tesztek során különböző frissítési idők szerint végeztem méréseket. Ezek közül a legrövidebb, egy átlagos és egy viszonylag nagy frissítési időhöz tartozó mérési eredmények az 5.5. táblázatban láthatók. A bemeneti késleltetési idő mindegyik csatorna esetében $I_D = 3\ ms$.

5.5. táblázat. Azonos konfigurációjú IO eszközvezérlők fontosabb jellemző értékei

Jellemzők [ms]	$U_{T1}= U_{T2} = 1\ ms$				$U_{T1}= U_{T2} = 16\ ms$				$U_{T1}= U_{T2} = 64\ ms$			
	IO_1-IO_1	IO_1-IO_2	IO_2-IO_1	IO_2-IO_2	IO_1-IO_1	IO_1-IO_2	IO_2-IO_1	IO_2-IO_2	IO_1-IO_1	IO_1-IO_2	IO_2-IO_1	IO_2-IO_2
Maximum	5,49	5,146	4,954	5,52	36,715	36,317	36,528	36,622	130,78	129,75	130,08	129,164
Minimum	1,96	2,093	2,05	2,768	5,658	5,496	5,032	5,247	9,689	7,05	6,57	7,461
Átlag	3,604	3,691	3,551	3,657	21,252	21,104	21,049	21,472	66,979	70,88	64,61	66,630
Jitter	3,53	3,053	2,904	2,752	31,057	30,821	31,496	31,375	121,091	122,70	123,51	121,703
ε_R [%]	48,97	41,36	40,89	37,62	73,07	73,02	74,82	73,06	90,40%	86,55	95,57	91,33%

A táblázatban rögzített értékek igazolják, hogy a jellemző értékekben történő eltérések meglehetősen alacsonyak, függetlenül attól, hogy milyen frissítési idők vannak beállítva. A legkiélezettebb helyzetben (1 ms frissítési időnél) az átlagértékekben történő eltérés mindösszesen 0,14 ms, de a válaszidők maximális értékei is csak alig több mint 0,5 ms-mal térnek el. Hasonló a helyzet 16 ms frissítési időnél is. Sőt ha figyelembe vesszük a frissítési idő 16-szoros növelését, az átlag és a maximális értékeknél regisztrált 0,4 ms körüli eltérési értékek igencsak jónak mondhatóak. Nagyobb frissítési idők esetében (64-, 128-, 256 ms) természetesen nőnek ezek az eltérések, de az eltérések még így is alig haladják meg a frissítési idő 10%-át.

Összegezve a mérések eredményeit kijelenthető, hogy *az azonosan konfigurált távoli IO eszközvezérlők bementi és kimeneti csatornái a válaszidők szempontjából úgy viselkednek mintha ugyanahhoz az eszközvezérlőhöz tartoznának.*

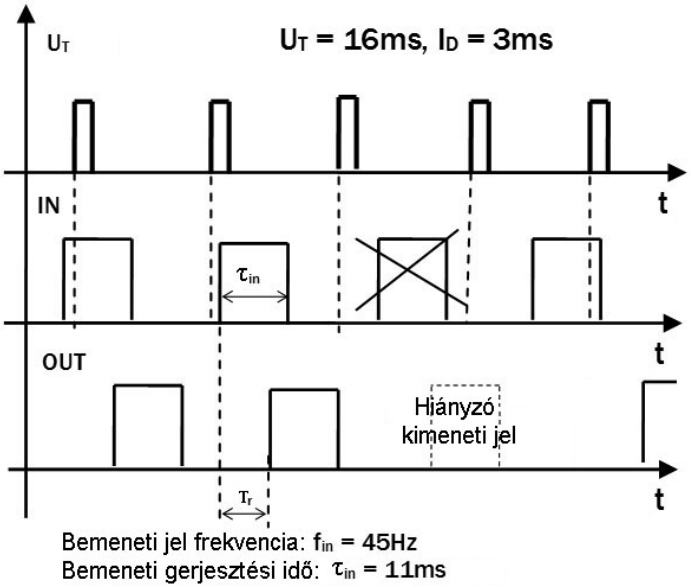
5.4. A minimális gerjesztési idő kérdése

Ebben a részben arra kerestem a választ, hogy mekkora lehet az a minimális gerjesztési időtartam, amely még kiváltja a bemenet reakcióját. Ez egyben azt is jeleni, hogy különböző frissítési idők mellett, milyen frekvenciájú négyszögjel kapcsolható a bemeneti csatornákra, amely még hibamentes választ generál a kimeneten. Ezt lényegében két tényező határozhatja meg: a bementi csatorna késleltetési ideje és a frissítési idő. Nevezzük ezt bemeneti időtartamnak és jelöljük τ_{in}-el. Kijelenthetjük:

$$\tau_{in} > I_D + U_T \tag{5.11}$$

Vagyis *a csatornagerjesztési idő nagyobb kell legyen mint a frissítési idő és a csatornakésleltetési idő összege*. Ez azt jelenti, hogy például *16 ms* frissítési idő és *3 ms* késleltetési idő esetén ez az érték *19 ms*-nál nagyobb kell legyen. A tesztek során alkalmazott *0,5 Hz/1%* kitöltésű impulzusok időtartama *20 ms* volt, tehát *16 ms* frissítési időig biztonsággal alkalmazható volt. *36 ms*-nál viszont már a (5.11) összefüggés alapján minimum *2%*-ra kell emelni a kitöltési tényezőt [S2].

Az általam vizsgált legkritikusabb esetben, amikor a frissítési idő *1 ms* és a késleltetési idő pedig *0,5 ms*, vagyis τ_{in} = *1,5 ms*, azt jelenti, hogy *330 Hz* frekvenciájú négyszögjelre még reagálnak a bemeneti csatornák. A mérések igazolták ezt a feltevést. Többszöri, hosszabb ideig tartó tesztelések után sem tapasztaltam kimaradást. A válaszidők pedig érdemben nem változtak. 350 Hz feletti frekvenciáknál már előfordulhatnak olyan esetek, amikor válasz nélkül maradnak azok a bemeneti impulzusok, amelyek éppen két frissítési idő között jelentek meg. Egy ehhez hasonló jelenséget ábrázol az 5.24. ábra is, ahol a *45 Hz* frekvenciájú bementi jelnél, már előfordulhatnak kimeneti válasz-kimaradások, ha a frissítési idő 16ms és a bementi késleltetés 3ms.



5.24. ábra. Helytelen bementi csatornagerjesztés

Mivel az időbeli folyamatok szinkronizálás nélküliek, a fenti jelenség teljesen véletlenszerűen fordul elő. *45 Hz*-nél még igen nehéz volt észrevenni az oszcilloszkópon, hogy kimaradnak válaszok. Észrevehető kimaradás *50 Hz* felett volt tapasztalható, majd *100 Hz* közelében gyakorlatilag teljesen megszűnt a válaszadás.

A következő táblázatban összefoglaltam az (5.11) reláció szerinti minimális bemeneti időtartamokat és a hozzá tartozó maximális négyszögjel (50% kitöltés) frekvencia értékeket különböző frissítési idők és bemeneti késleltetések szerint.

5.5. táblázat. A minimális bemeneti gerjesztési időhöz tartozó frekvenciák

U_T	τ_{in}[ms]			f_{in}[Hz]		
[ms]	I_D=3ms	I_D=1ms	I_D=0,5ms	I_D=3ms	I_D=1ms	I_D=0,5ms
1	4	2	1,5	125	250	330
2	5	3	2,5	100	166	200
4	7	5	4,5	71	100	110
8	11	9	8,5	45	55	58
16	19	17	16,5	26	29	30
32	35	33	32,5	14	15	15
64	67	65	64,5	7	7	7
128	131	129	128,5	3	3	3

A táblázat segít abban, hogy meghatározott frekvenciájú bementi jel esetében milyen frissítési időt konfiguráljunk az adott kapcsolathoz, hogy biztonságos válaszadást kapjunk

a kimeneti csatornákon. Így például 50 Hz frekvenciájú bementi jelnél a frissítési idő legfeljebb 8 ms lehet, 1ms vagy annál rövidebb bementi késleltetés mellett.

5.5. Új tudományos eredmények

A nem szinkronizált ipari Ethernet rendszerek valós idejű működésének alapvető meghatározója a reakcióidő. Ennek pontos elméleti meghatározása nem lehetséges, ugyanis az eszközvezérlők bemeneti csatornáit gerjesztő jelek véletlenszerűen érkeznek, a ciklikusan beállított frissítési időhöz képest. Érdemes viszont a szélső értékeket, főleg a lehetséges maximális értéket meghatározni, azért hogy a valós idejű működést meghatározó időkorlát rögzíthető legyen. Vagy ahogy az a gyakorlatban lenni szokott, adott technológiai időkorláthoz kell igazítani a reakcióidőket. Ennek érdekében meghatároztam az elméletileg lehetséges minimális és maximális reakcióidőket, a maximális jittert, és a relatív eltérést.

A méréseket egy általam kidolgozott új eljárással végeztem. A mérések, az eredmények rögzítése, valamint a jellemző értékek kiszámítása egy LabView 8.2 alkalmazás és egy NI USB6221 mérőinterfésszel történik teljesen automatikusan. A mérési pontosság 0,05 µs. A mérések során igyekeztem minden meghatározó szituációt figyelembe venni. Ezt követően több ezer, különböző szituációkban végzett mérési eredmények feldolgozása után meghatároztam a következő tézist:

3. TÉZIS [S1, S2, S3, S5, S12, S14, S19]

Kidolgoztam egy új, automatikusan működő mérési eljárást, amelynek segítségével mérhető a reakcióidő. Megállapítottam, hogy a nem szinkronizált ipari Ethernet rendszer bármely eszközvezérlőéhez tartozó kimeneti csatornák reakcióideje a kontroller bemeneti késleltetéssel bővített minimális ciklusideje és a konfigurációban meghatározott frissítési idő négyszerese között véletlenszerűen változik.

3.1. Altézis. *Egyirányú kommunikáció esetén, ha a bemeneti paraméterek azonosak, a válaszidők nagysága nem függ attól, hogy melyik eszköz (a kontroller vagy az eszközvezérlő) tölti be az termelő, illetve a fogyasztó szerepét.*

3.2. Altézis. *Kétirányú kommunikációnál a válaszidők változásának elemzése során bizonyos periodicitás figyelhető meg, amely arányos a frissítési idővel. Ha a bementi jelek frekvenciáját állandó értéken tartjuk, akkor ebből a változásból meghatározható a szinkronizálás hiányában fellépő, a két eszköz közötti frissítési idők egymáshoz viszonyított csúszása.*

3.3. Altézis. *Az azonosan konfigurált távoli IO eszközvezérlők bemeneti és kimeneti csatornái a válaszidők szempontjából úgy viselkednek, mintha ugyanahhoz az eszközvezérlőhöz tartoznának.*

3.4 Altézis. *A bemeneti jel érzékelése akkor biztonságos, ha a csatornagerjesztés időtartama nagyobb, mint a frissítési idő és a csatornakésleltetési idő összege.*

VI. FEJEZET

SZIGORÚAN VALÓS IDEJŰ IPARI ETHERNET RENDSZEREK BUSZFRISSÍTÉSI IDEJÉNEK MEGHATÁROZÁSA ÉS MODELLEZÉSE

A szigorúan valós idejű ipari Ethernet kommunikációs rendszereknél a vezérlési adatok továbbítása két módon valósítható meg. Az egyik megoldás, amikor a kontroller mindegyik, vele Ethernet kommunikációs kapcsolatban lévő IO eszköznek külön-külön címzi a kereteket, amelyek a megfelelő vezérlési adatokat tartalmazzák. Ezt a módszert alkalmazza a már korábban bemutatott Profinet IRT rendszer. Ebben az esetben 36 bájtnál kisebb vezérlési adatok gyakorlatilag nem befolyásolják a továbbításhoz szükséges időt, mert így a teljes keretméret nem haladja meg a minimális, 64 bájtos keretet. Hátránya ennek a megoldásnak, hogy az előbb említett szituációban a keretterhelési tényező a legjobb esetben is csak 56% lehet [S15]. Igazi előnye az 1000 Mb/s-os Gigabit Ethernet hálózaton mutatkozik meg, amikor az adattovábbításhoz szükséges idő gyakorlatilag már nem függ a keret méretétől [S8].

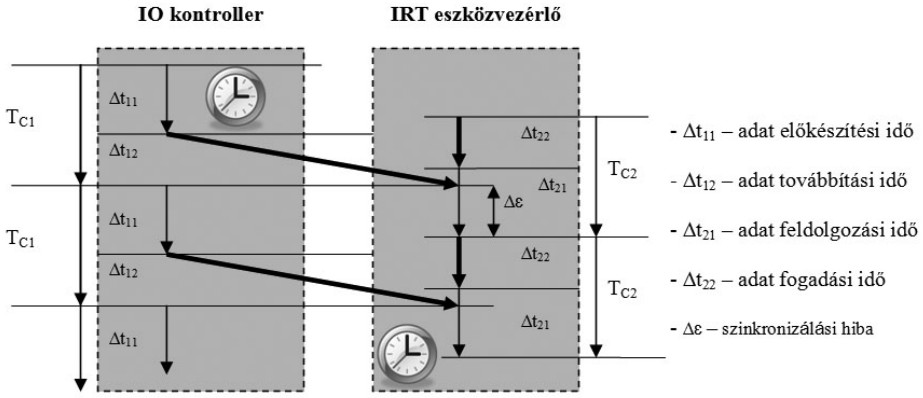
A másik megoldás arra törekszik, hogy kihasználja a maximális keretterhelést, amely az IEEE 802.3 szerint 1500 bájt lehet. Ebben az esetben az IO eszközök számára továbbítandó vezérlési adatokat (telegramokat) igyekszünk egy vagy több kereten belül úgy elhelyezni, hogy minél nagyobb legyen a keretek kihasználtsága [S5]. Ez a módszer kétségtelenül előnyt jelent kisméretű (36 bájtnál kisebb) telegramok továbbítása esetén. Ezt a módszert alkalmazza a II. fejezetben már bemutatott EtherCAT.

Ebben a fejezetben az a célom, hogy kidolgozzak egy olyan modellt, amelynek segítségével mindkét módszer tanulmányozható, közvetlenül összehasonlítható egymással és segíthet abban, hogy adott vezérlési szituációkban melyik módszer alkalmazható a leghatékonyabban. A témával korábban már többen is foglakoztak, mint például Juergen Jaspetnite professzor [58] vagy Gunnar Prytz az ABB AS Corporate Research Center mérnöke [59]. Mindketten más-más, de hasonló modelleket használtak a két rendszer elemzésére. Az általuk kidolgozott elveket felhasználva, de más számítási módszereket alkalmazva sikerült továbbfejleszteni ezeket. [S5, S8, S15]

6.1. A szinkronizálás szükségessége

Az ötödik fejezetben ismertetett nem szinkronizált valós idejű ipari Ethernet rendszerek átlagos reakcióideje kétirányú kommunikáció során a legjobb esetben is meghaladta a *3,5 ms*-ot, a jitter pedig alig kevesebb, mint *2 ms* volt. (5.3. táblázat utolsó oszlopa) Ezek az értékek a C osztálybeli, szigorúan valós idejű rendszerek követelményeit már nem tudják kielégíteni. A motorvezérléseknél, ahol időben akár 2-3 vagy annál több tengely összehangolt, pontos mozgatására van szükség, a szinkronizálás nélküli, valós idejű üzemmód már nem alkalmazható. Az ilyen rendszereknél az IO eszközvezérlők óráinak szinkronizálása nélkülözhetetlen. A szinkronizálás alapját a legpontosabb órajel, rendszerint a kontroller órajele képezi.

Hatékony és pontos szinkronizálással a jitter minimálisra csökkenthető. A szigorúan valós idejű ipari Ethernet rendszereknél ez mindenképpen 1 μs alatti értéket feltételez. A szinkronizálás szükségességét a 6.1. ábrán láthatjuk.

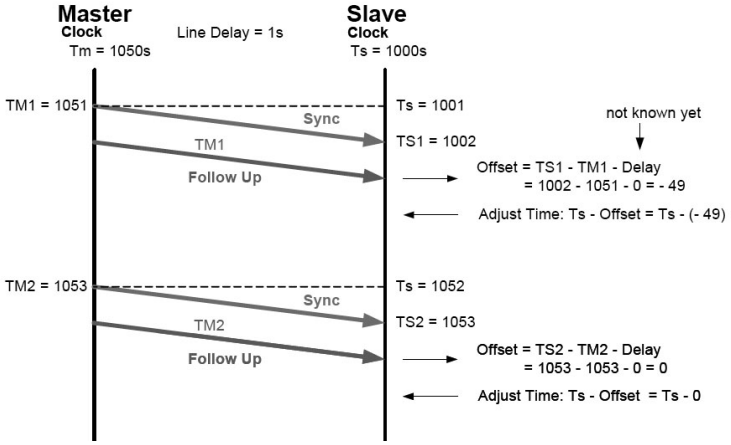


6.1. ábra. Az adatküldés fázisai szinkronizálás hiányában

A6.1. ábrán látható, hogy annak ellenére, hogy a T_{C1} és T_{C2} buszfrissítési ciklusok egyenlők, szinkronizálás hiányában az IRT eszközvezérlő $\Delta\varepsilon$ időig nem tudja átvenni az érkezett adatokat, mert a T_{C2} ciklus még nem kezdődik. Az egyik cél tehát az, hogy az eszközvezérlő órajelét úgy igazítsuk minden alkalommal, hogy a szinkronizálási hiba minél kisebb legyen. A másik cél természetesen az, hogy a buszfrissítési ciklusidők is minél kisebbek legyenek. Ennek a kritériumait is ebben a fejezetben fogom bemutatni.

6.1.1. A szinkronizálás fázisai

Az órajel alapú kommunikációs rendszerek szinkronizálását az IEEE 1588 2002-ben kiadott, majd 2008-ban módosított szabvány írja le [35]. A PTP (Precision Time Protocol) néven ismert protokoll beépül az adatkapcsolati rétegbe és az UDP/IP-n keresztül néhány szinkronkeretbe ágyazott információcsere után megvalósítja az órajelek beállítását. A feladat tehát az, hogy adott hálózati csomópont pontos órája (Master Clock) szinkronizálja az alhálózat összes eszközvezérlőinek óráit (Slave Clock). A szinkronizálás két lépésben valósul meg. Először a master elküld egy szinkronizálást kérő keretet (Sync), majd egy következő keretben (Follow Up) elküldi a saját órájának idejét. A kettő különbsége megadja a két óra közötti eltérést (*Offset* = $T_{S1} - T_{M1} - Delay$). A kommunikációs vonal késleltetési idejét viszont még nem ismerjük, ezért ezt nullának vesszük (6.2. ábra).

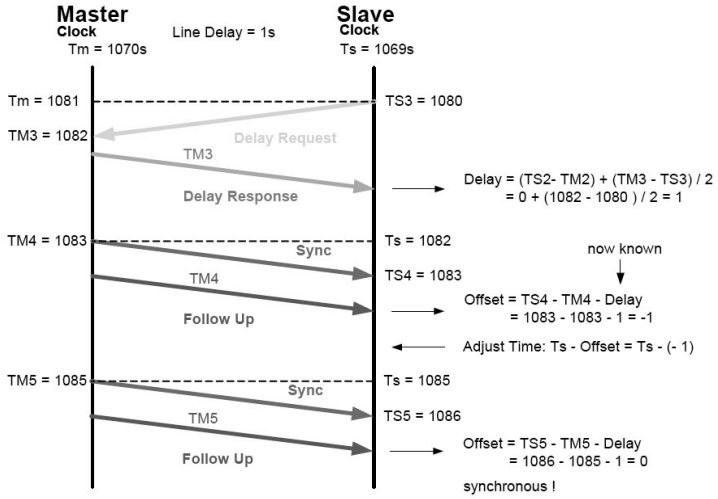


6.2. ábra. Az offset kiszámítása [57]

Az így beállított idő között éppen a késleltetési idő okoz különbséget. A második fázisban (6.3. ábra) a slave küld egy késleltetési kérelmet (Delay Request), a master pedig válaszol rá, vagyis újra elküldi a saját idejét (Delay Response). A négy, most már ismert időpillanat alapján, meghatározható a késleltetés [66].

$$T_D = \frac{(T_{S2} - T_{M2}) - (T_{S3} - T_{M3})}{2} \tag{6.1}$$

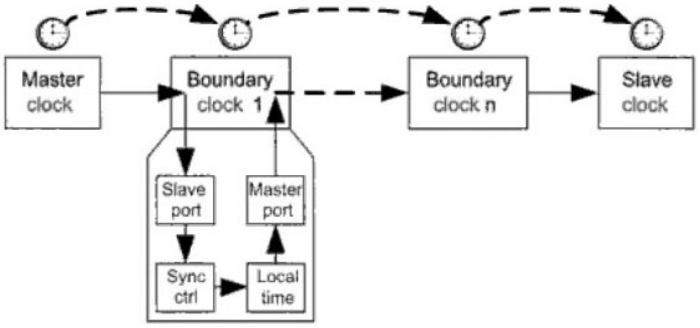
A következő szinkronizálási folyamat így már ismert késleltetéssel történik, így pontosan meghatározható lesz a slave órája. Az offset nulla lesz, és ameddig ez nulla, addig újabb órabeállításra nincs szükség.



6.3. ábra. A slave órabeállítása [57]

6.1.2. Szinkronizálási módszerek

Az IEEE 1588 szabványban alkalmazott egyik korábbi megoldás az, amikor a master órához minden csomópontban egy-egy logikai órát rendelünk hozzá (boundary clock), amelyek szinkronizálják az adott csomópont eszközeinek óráit és egyben mester óraként viselkednek a távolabbi csomópontok irányában. (6.4. ábra).



6.4. ábra. A boundary clock módszer elve [60]

Ennek a módszernek az a nagy hátránya, hogy a csomópontok késleltetési hibáihoz még hozzáadódnak a csomóponton belüli lokális órajel szabályozási hibái is. Ezek már ezzel a hibával képezik a következő csomópont master óráját. Több csomópont esetén ez

jelentősen megnövelné a master és főleg a távolabbi slave csomópontok közötti szinkronizálási hibát (jittert). A módszert főleg számítógépes hálózatokban használják.

Legyen J_0 a csomópontonkénti, J_i pedig egy adott csomópont maximális jitter értéke. Ha az előbbiekben meghatározott módszer szerint számolunk, akkor:

$$\begin{aligned} J_1 &= 1J_0; \\ J_2 &= J_1 + J_1 + J_0 = J_0(1+2); \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ J_n &= J_0\,(1+2+\ldots.+n) \end{aligned} \tag{6.2}$$

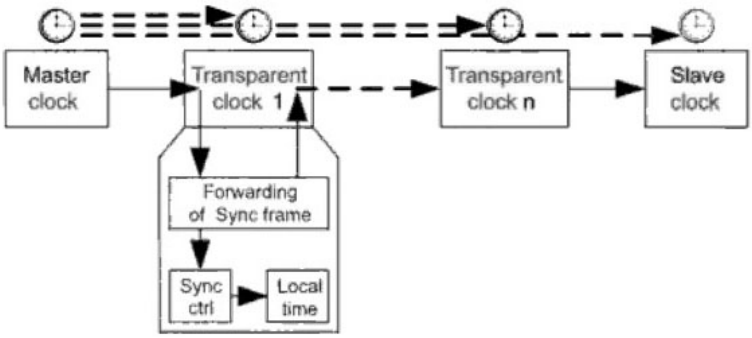
Általános formában felírható a következő összefüggés:

$$J_{n\max} = J_0 \cdot \sum_{i=1}^{n} i = J_0 \frac{n \cdot (n+1)}{2} \tag{6.3}$$

Látható, hogy a csomópontok számának növelésével a maximális jitter hatványozottan nő. Ebből következik, hogy a legtöbbször busz topológiát alkalmazó ipari Ethernet rendszereknél a távolabbi csomópontok szinkronizálási hibái egyre nagyobbak.

A II. fejezetben bemutatott Profinet IO az IRT hálózat szinkronizálására a PTCP-t (Precision Transparent Clock Protokol) használja. Ez elsősorban hardver alapú megoldás, amelynél a szinkronizálás megvalósítása az ASIC egyik alapvető feladata. A PTCP az adatkapcsolati rétegbe integrálódik és ez azt jelenti, hogy az IRT keretek csak azonos alhálózati szegmensen belül továbbíthatók [37].

A PTCP segítségével akár 32 IRT eszköz szinkronizálása is megvalósítható egyetlen egy mester óra (Master Clock) alapján. A módszer lényege, hogy a csomópontokban un. átlátszó órát (transparent clock) alkalmaznak, amely a nem neki címzett szinkronkeretet átereszti, csupán egy időbélyeget szúr be a keretbe, amely a csomópont késleltetéséről tartalmaz információt. Így a saját csomópontbeli órajel szabályozási kör szinkronizálási hibája nem kerül továbbításra. (6.5. ábra).



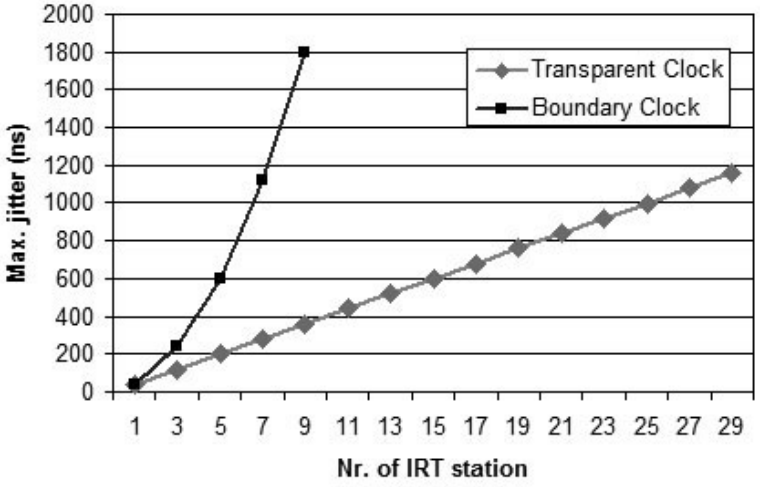
6.5. ábra. Átlátszó óra alkalmazása [37]

Ebben az esetben a jitter maximális értéke, az előző megoldáshoz képest lineárisan fog növekedni, és továbbra is függ az IRT eszközök számától, az alábbiak szerint:

$$J_{i\max} = J_0 \cdot i\,; \quad i = 1,2,\ldots,n \tag{6.4}$$

A két módszer szinkronizálási hibáit összehasonlítva azt láthatjuk, hogy a PTC protokoll alkalmazása felel meg leginkább a valós idejű IRT adatátvitel szinkronizálására.

(6.6. ábra). A maximális jitter alakulása a (6.3) illetve (6.4) összefüggések alapján lett meghatározva, $J_0 = 40\ ns$ maximális csomópontonkénti jitterrel. Ezt a megoldást alkalmazzák a Siemens S7 Profinet IRT rendszerei is, ahol az ET200-as IRT csomópontok által bevitt jitter kb. 40 ns nagyságrendű [S16].



6.6. ábra. A maximális jitter változása a csomópontok számának növekedésével

6.2. A ciklusidő kiszámítása [S15]

A szigorúan valós idejű ipari Ethernet kommunikációs rendszerek közös jellemzője a ciklikus adattovábbítás. Ebből következik, hogy a fejezet elején említett eljárások közül bármelyiket is választjuk, a legfontosabb feladat mindig a ciklusidő meghatározása lesz.

A ciklusidő itt ebben az értelemben az az időintervallum lesz, amely szükséges ahhoz, hogy a kontroller elküldje az összes vele kommunikációs kapcsolatban álló eszközvezérlők számára a vezérlési adatokat, illetve fogadja ezek állapotadatait. A laza valós idejű kommunikációval ellentétben itt már figyelembe kell venni a továbbításhoz szükséges időt. A viszonylagosan rövid frissítési idők miatt, a keretek továbbítási ideje mellett, már a közeg és a hálózati eszközök késleltetési idejével is számolni kell.

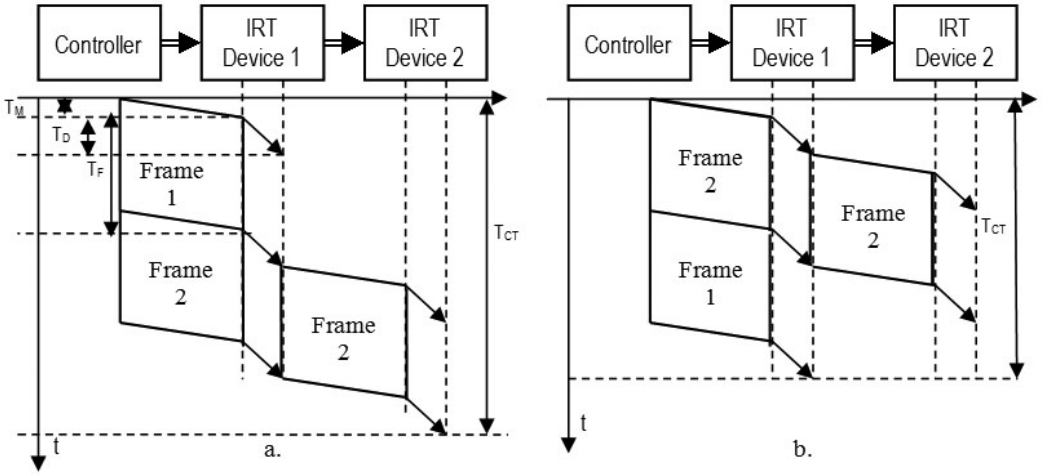
A továbbiakban a második fejezetben bemutatott Profinet IRT és EtherCAT rendszerek ciklusidő változását fogom vizsgálni, adott körülmények között, 100 Mb/s full duplex Ethernet kommunikációt és busz topológiát alkalmazva. A két rendszer kommunikációs stratégiáját, valamint protokolljait a 2.2.6. illetve a 2.3.2. fejezetekben már bemutattam. Ezeket felhasználva dolgoztam ki azokat a módszereket, amelyek segítségével a ciklusidők kiszámíthatók.

6.2.1. Profinet IRT ciklusidő

Ha abból indulunk ki, hogy a Profinet IRT kontroller mindegyik IRT eszköz számára külön-külön keretet küld vagy fogad, akkor a ciklusidő meghatározásakor négy fontos tényezőt kell figyelembe vegyünk:

1. a keretek továbbítási sorrendjét,
2. a vezérlési adatok nagyságát, amely meghatározza a keret méretét,
3. a bitsebességet,
4. a kerettovábbítási idő és az eszközök, illetve a közeg késleltetési idejének arányát.

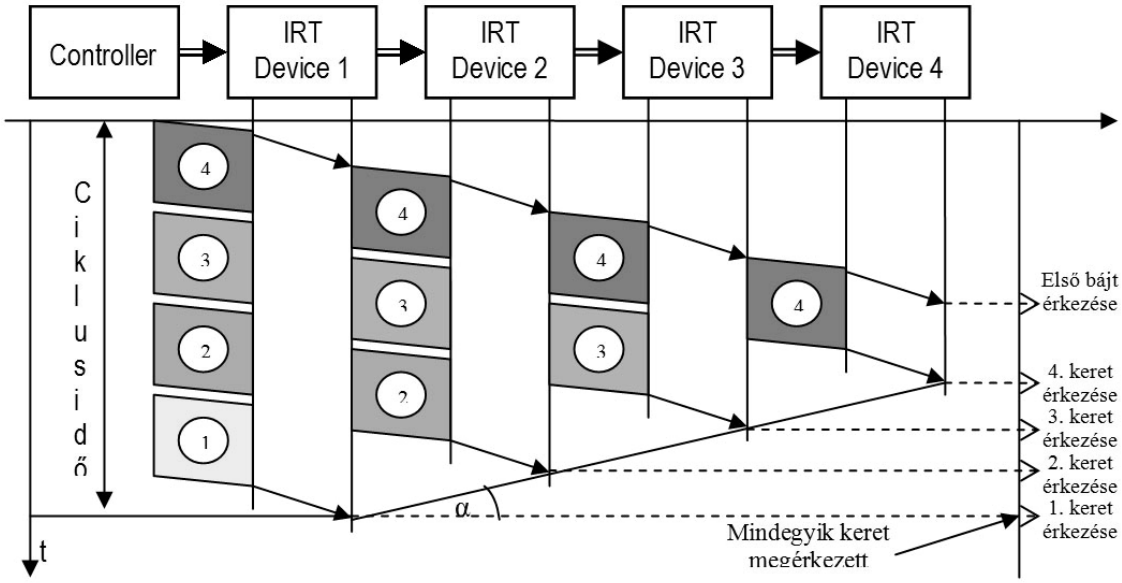
A keretek továbbítási sorrendjének meghatározására a 6.7. ábrán látható grafikus kerettovábbítási modellt fogom használni, amelyen egy-egy tér-idő diagramot láthatunk, egy kontroller és két IRT eszköz esetében. A 6.7.a ábrán a keretek küldési sorrendje megegyezik az eszközök busz-sorrendjével. Látható, hogy ebben az esetben nagyobb a minimálisan meghatározható ciklus idő, mint a 6.7.b. ábrán bemutatott megoldás során, ahol fordított a keretek sorrendje, vagyis a kontroller először a legtávolabbi eszköznek címzi a keretet.



6.7. ábra. Tér-idő diagram két IRT eszköz esetén különböző sorrendben [57]

Ez a különbség főleg akkor jelentős, amikor nagyobb méretű kereteket továbbítunk, vagyis amikor a keretek továbbítási ideje jóval nagyobb, mint a közeg és az eszközök késleltetési ideje.

A 6.7.b. ábrán bemutatott megoldást természetesen kiterjeszthetjük N darab IRT eszközre is. A következő ábrán a Profinet IRT kerettovábbításának tér-idő modelljét láthatjuk, négy eszköz esetében. Ha feltételezzük, hogy igaz N számú eszközre is, akkor ennek alapján meghatározható a ciklusidő [57].



6.8. ábra. Tér-idő diagram $\alpha > 0$ esetében [57]

A 6.8. ábra alapján felírhatjuk a következő összefüggést, amely a ciklusidő maximális értékét fogja meghatározni:

$$T_{CT} = T_M + T_D + N \cdot (T_F + T_{IFG}); \qquad \text{ha } \alpha > 0 \tag{6.5}$$

- T_F – kerettovábbítási idő, függ a keret nagyságától és a bitsebességtől,
- T_{IFG} – keretek közötti időrés (IFG),
- T_D – az IRT eszközök hálózati kapcsolóinak késleltetési ideje,
- T_M – a közeg (kábel) késleltetési ideje (kb. 5 ns/m),
- T_{CT} – az IRT ciklusidő,
- N – az IRT eszközök száma.

Ugyancsak a 6.7. ábra alapján felírhatjuk az α szögre a következő összefüggést:

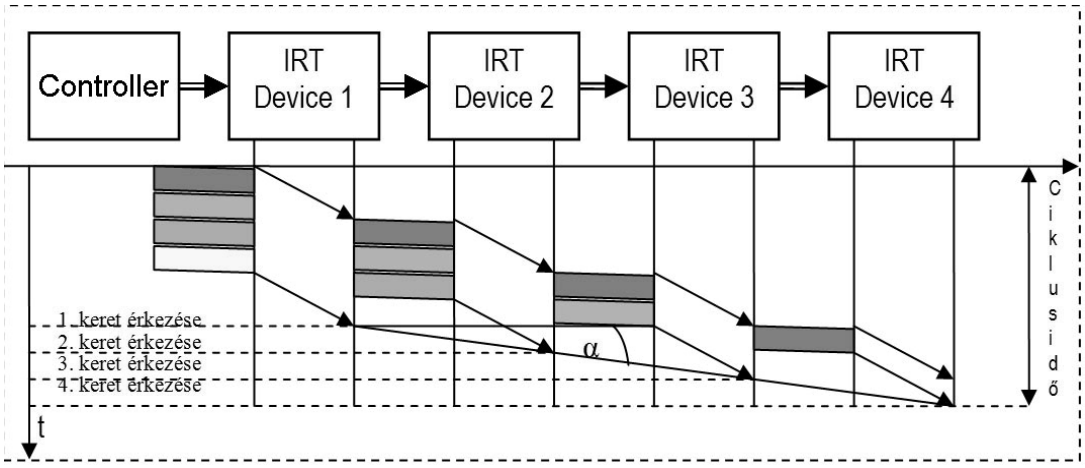
$$tg\alpha = \frac{T_M + T_D + N \cdot (T_F + T_{IFG}) - [N \cdot (T_M + T_D) + T_F + T_{IFG}]}{N - 1} = T_F + T_{IFG} - T_M - T_D; \tag{6.6}$$

Belátható, hogy független csomópontok számától ($N>1$) és a legoptimálisabb adatátvitel akkor érhető el ha $\alpha=0$. A (6.6) reláció alapján ez akkor teljesül, amikor a kerettovábbítási idő megegyezik az eszközök, illetve a közeg késleltetési idejével:

$$T_F + T_{IFG} = T_M + T_D \tag{6.7}$$

Ahhoz, hogy a (6.7) reláció teljesüljön, vagy a keret méretét kellene csökkenteni, vagy növelni az átviteli sebességet. Ha feltételezzük, hogy a maximális *100 m* a távolság az eszközök között, és tudjuk, hogy az Ethernet kábel méterenként kb. *5 ns* késleltetést hoz létre, következik, hogy $T_M = 0{,}5\ \mu s$. Ha az IRT eszközök késleltetési idejét kb. 3μs-nak vesszük (gyártó adatai), a keretek közötti rés $T_{IFG} = 0{,}96\ \mu s$ (12 bájt továbbítási ideje 100 Mb/s bitsebességnél) a kerettovábbításhoz szükséges idő $T_F + T_{IFG} = 3{,}5\ \mu s$ lesz. Az abszolút kerettovábbítási idő így *2,54 μs*. Ez az érték kevesebb, mint 32 bájtos kereteket feltételezne, de az IEEE 802.3 szabvány szerint ez nem lehet kisebb 64 bájtnál. Viszont *192 Mb/s*-os bitsebességnél teljesülne az optimális feltétel. Ebből következik, hogy a Profinet IRT igényelné a nagyobb átviteli sebességű hálózatot.

A helyzet tehát meglehetősen összetett, mivel a kerettovábbítási idő függ egyrészt a keret méretétől, másrészt a bitsebességtől valamint fennállhat az a helyzet is, amikor az α szög kisebb lesz, mint nulla, például Gigabit Ethernet hálózatban.



6.9. ábra. Tér-idő diagram α < 0 esetében [57]

A 6.9. ábrán láthatjuk, hogy egyrészt megváltozott a keretek érkezési sorrendje, másrészt a további átviteli sebesség növelése már számottevően nem befolyásolja a ciklusidőt, mert ebben az esetben a ciklusidő leginkább a közeg és az eszközök késleltetési idejétől függ:

$$T_{CT} = N \cdot (T_M + T_D) + T_F + T_{IFG}; \quad \text{ha } \alpha < 0 \tag{6.8}$$

A fenti reláció 100 Mb/s-os hálózatban, az előzőekben említett okok miatt fizikailag nem alkalmazható, viszont Gigabit Ethernet hálózatban már lenne értelme egészen addig, míg a keret mérete ezt lehetővé teszi.

Az IEEE 802.3 Ethernet keretszabványhoz igazodva, a keret mérete akkor is 64 bájt kell legyen ha 36 bájtnál kevesebb felhasználói adatot tartalmaz. Az adatmező ilyenkor „töltelék bitekkel" egészül ki. 36 adatbájtig a keret mérete ugyan nem változik, viszont a keret kihasználtsága a legjobb esetben is csak 56%. Hogy ezt mérni tudjuk, bevezetjük a F_{PF} (Frame Payload Factor) keretterhelési tényezőt:

$$F_{PF} = \frac{d_{DATA}}{d_{FRAME}} \cdot 100\% \tag{6.9}$$

Ahol a d_{DATA} az IRT felhasználói adat, d_{FRAME} pedig a keret nagysága bájtban.

A legjobb kihasználtságot, 98%-ot, 1490 bájt IRT adat esetében kapjuk, amikor a keret mérete maximális (d_{Fmax}) lesz. Felhasználva a (6.5) összefüggést kapjuk, hogy a maximális IRT keret ciklusideje 100 Mb/s-os hálózatban:

$$T_{CT\max} = T_M + T_D + N(8 \cdot \frac{d_{F\max}}{b} + T_{IFG};) \quad \text{ahol: } b = 100\ Mb/s \tag{6.10}$$

Ez az érték kb. *126 μs* lesz, vagyis mintegy 12-szerese a 64 bájtos ciklusidőnek. Ez semmiképp sem indokolja a maximális keretkihasználtság gyakori alkalmazását, mert ez az IRT eszközök számának csökkentését jelentheti. Célszerű tehát egy optimális terhelést meghatározni, ahol az eszközök számát is figyelembe vesszük. [S7, S15]

Ha a (6.5) relációban a kerttovábbítási időt kifejezzük a bitsebesség (b) és a vezérlési adatok (d_{DATA}) függvényében a 100 Mb/s-os hálózatban, az alábbi gyakorlatban alkalmazható összefüggéseket kapjuk:

$$T_{CT} = T_M + T_D + 6{,}72 \cdot N = 3{,}5 + 6{,}72 \cdot N; \quad \text{ha } d_{DATA} \le 36\ bájt \tag{6.11}$$

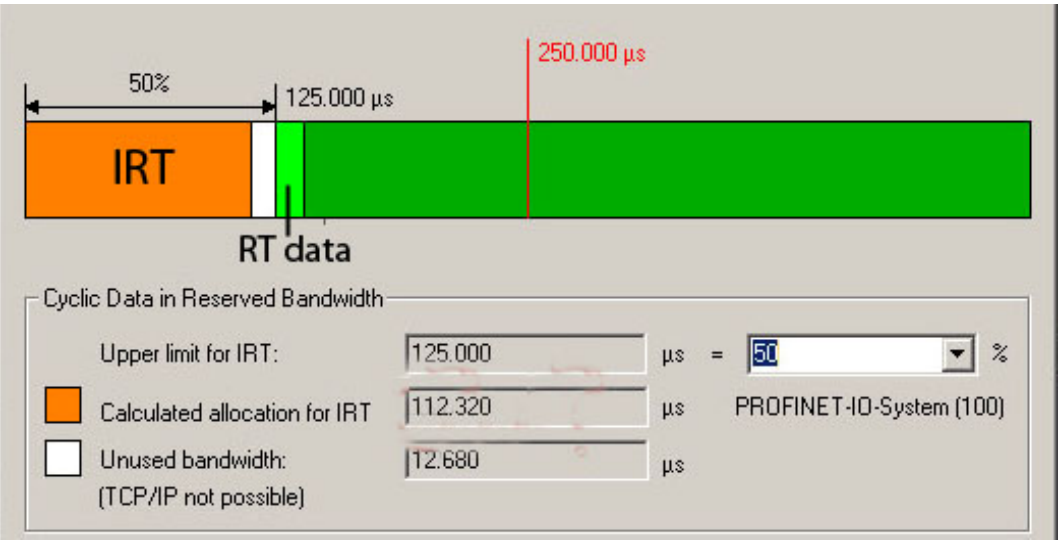
$$T_{CT} = T_D + T_M + \frac{8 \cdot (48 + d_{DATA}) \cdot N}{b} = 3{,}5 + 0{,}08 \cdot N(48 + d_{DATA}); \quad \text{ha } d_{DATA} > 36\ bájt \tag{6.12}$$

A fenti összefüggésekben TM és TD a közeg illetve az eszközök késleltetési idejét jelenti. Az eszközök késleltetési ideje a Profinet IRT rendszereknél 3 μs, a közeg késleltetési ideje pedig a maximális 100 méteres szegmensen kb. 0,5 μs (egész pontosan 0,454 μs). Látható, hogy 36 bájnál kisebb vezérlési adatok esetében a ciklusidő csak az eszközök számától függ, míg az ennél nagyobb értékeknél már a vezérlési adatok nagysága is hatással van rá. Az ilyen rendszerek tervezésénél célszerű egy optimális terhelést meghatározni, figyelembe véve az eszközök számát is. Ennek megfelelően tartalmaz ciklusidő értékeket (T_{CT}) a 6.1. táblázat, amely a (6.12) alapján készült. (Az értékeket μs-ban adtam meg.)

6.1. Táblázat. A ciklusidő változása az eszközszám és a terhelés függvényében

N	d_{DATA} [bájt]															
	36	44	52	60	68	76	84	92	100	108	116	124	132	140	148	156
2	16,94	18,22	19,5	20,78	22,06	23,34	24,62	25,9	27,18	28,46	28,46	29,74	31,02	32,3	33,58	34,86
4	30,38	32,94	35,5	38,06	40,62	43,18	45,74	48,3	50,86	53,42	53,42	55,98	58,54	61,1	63,66	66,22
8	57,26	62,38	67,5	72,62	77,74	82,86	87,98	93,1	98,22	103,34	103,34	108,46	113,58	118,7	123,82	128,94
12	84,14	91,82	99,5	107,18	114,86	122,54	130,22	137,9	145,58	153,26	153,26	160,94	168,62	176,3	183,98	191,66
16	111,02	121,26	131,5	141,74	151,98	162,22	172,46	182,7	192,94	203,18	203,18	213,42	223,66	233,9	244,14	254,38
20	137,9	150,7	163,5	176,3	189,1	201,9	214,7	227,5	240,3	253,1	253,1	265,9	278,7	291,5	304,3	317,1
24	164,78	180,14	195,5	210,86	226,22	241,58	256,94	272,3	287,66	303,02	303,02	318,38	333,74	349,1	364,46	379,82
28	191,66	209,58	227,5	245,42	263,34	281,26	299,18	317,1	335,02	352,94	352,94	370,86	388,78	406,7	424,62	442,54
32	218,54	239,02	259,5	279,98	300,46	320,94	341,42	361,9	382,38	402,86	402,86	423,34	443,82	464,3	484,78	505,26
36	245,42	268,46	291,5	314,54	337,58	360,62	383,66	406,7	429,74	452,78	452,78	475,82	498,86	521,9	544,94	567,98
40	272,3	297,9	323,5	349,1	374,7	400,3	425,9	451,5	477,1	502,7	502,7	528,3	553,9	579,5	605,1	630,7
44	299,18	327,34	355,5	383,66	411,82	439,98	468,14	496,3	524,46	552,62	552,62	580,78	608,94	637,1	665,26	693,42
48	326,06	356,78	387,5	418,22	448,94	479,66	510,38	541,1	571,82	602,54	602,54	633,26	663,98	694,7	725,42	756,14
52	352,94	386,22	419,5	452,78	486,06	519,34	552,62	585,9	619,18	652,46	652,46	685,74	719,02	752,3	785,58	818,86

A táblázat szürke tartományában lévő ciklusidő értékek gyakorlatilag nem használhatóak IRT adatátvitelre, mert meghaladják a maximálisan lefoglalható sáv méretét (500 µs). A 250 µs-ot meghaladó értékek is csak bizonyos korlátozások mellett alkalmazhatóak. A táblázat segítségével könnyen meghatározhatóak a rendszer konfigurálási paraméterei is. Például, ha 12 IRT eszközt kell működtetnünk, és tudjuk, hogy a vezérlési adatok nagysága nem haladja meg a 60 bájtot, akkor a táblázat alapján kapjuk, hogy a ciklusidő minimális értéke 107,18 µs. Ennek megfelelően a konfigurációban választunk egy közeli IRT fázisidőt, jelen esetben 112,320 µs-ot. 50%-os lefoglalási arány mellett, még marad 18,68 µs tartalék intervallum. A nyílt intervallum 250 µs lesz (6.11. ábra). A teljes busz ciklusidő pedig 500 µs. A TCP/IP kommunikáció számára fennmaradó mintegy 368 µs idő alatt kb. 4,5 megabájtnyi egyéb adat is továbbítható. Ebből az a következtetés is levonható, hogy az IRT allokáció gyakorlatilag nem befolyásolja a TCP/IP alapú adattovábbítást.



6.10. ábra. Az IRT ciklus konfigurálása egy Profinet IO rendszerben

A lefoglalási arányt ha megnöveljük, akár 60 bájtnál nagyobb vezérlési adatok is továbbíthatóak, vagy növelhető az IRT eszközvezérlők száma a TCP/IP rovására. Az eredeti példánál maradva akár 250 μs-os busz ciklusidő is konfigurálgató 90%-os lefoglalási aránnyal.

Gigabit Ethernet hálózatban, a keretterheléstől függően, ameddig a keretek továbbításához szükséges idő kisebb, mint az eszközök és a szegmensek késleltetési ideje ($\alpha < 0$), addig a (6.8), majd pedig amikor $\alpha > 0$, akkor a (6.5) összefüggés alkalmazható. Ezeket az elvi összefüggéseket gyakorlatban alkalmazhatóvá tehetjük, ha a kerettovábbítási időt kifejezzük a bitsebesség (b) és a vezérlési adatok (d_{DATA}) függvényében. 36 bájtnál kisebb vezérlési adatok esetében $\alpha < 0$, ezért ezek számára a (6.13) összefüggés használható. A közeg késleltetése (T_M) továbbra is *0,454 μs* (itt már lényeges lehet a pontos érték használata), viszont az IRT eszközök által bevitt késleltetés (T_D) kb. *0,6 μs*-ra csökken. (Ez utóbbi a gyártó adatai alapján.)

$$T_{CT} = N \cdot (T_M + T_D) + 0{,}672 = 1{,}054 \cdot N + 0{,}672 \text{ ; ha } d_{DATA} \leq 36 \text{ bájt} \tag{6.13}$$

36 bájtnál nagyobb vezérlési adatok számára két relációt kell alkalmaznunk, ugyanis egy adott d_{DATA} adatmennyiségtől kezdődően $\alpha > 0$, ezért itt már a (6.5) szerint kell eljárni. Ezeket figyelembe véve a következő gyakorlati összefüggések alkalmazhatók a ciklusidő meghatározására:

$$T_{CT} = 0{,}008 \cdot (48 + d_{DATA}) \cdot N + 1{,}054 \text{ ; ha } d_{DATA} > 36 \text{ bájt és } \alpha \geq 0 \tag{6.14}$$

$$T_{CT} = 0{,}008 \cdot (48 + d_{DATA}) + 1{,}054 \cdot N \text{ ; ha } d_{DATA} > 36 \text{ bájt és } \alpha < 0 \tag{6.15}$$

Az előbbi összefüggések alapján érdemes lenne meghatározni, hogy mekkora adatmennyiségnél lesz a optimális az adatátvitel, illetve mikor lesz minimális a ciklusidő. Ehhez a (6.7) relációból kell kiindulni, amikor $\alpha = 0$. Az egyenlőség bal oldalát kifejezzük az optimális adatmennyiség függvényében, a jobb oldalon pedig a közegkésleltetést a szegmens hosszával arányos módon. Ennek megfelelően a következő gyakorlati összefüggést kapjuk:

$$d_{OPT} = 27 + 0{,}5657 \cdot l \text{ ;} \tag{6.16}$$

A fenti relációban l a szegmensek hosszát (max. 100 m) jelöli, és $T_D = 0{,}6\ \mu s$ állandó, valamint $\tau = 0{,}00454\ \mu s$ méterenkénti kábelkésleltetési idő értékkel számoltam. Érdekes következtetést vonhatunk le, mégpedig azt, hogy Gigabit Ethernet hálózatban a maximális 100 méter hosszúságú szegmenseknél lesz a legnagyobb az optimális adatmennyiség (83 bájt). Egy átlagosan 50 méteres szegmensekből álló hálózatban ez az optimális adatmennyiség 55 bájt lesz. Azt is észre kell vegyük, hogy 36 bájtnál kisebb optimális adatmennyiségről nem beszélhetünk, mivel ezalatt az érték alatt a keretméret már független az adatmennyiségtől (64 bájt). Következik, hogy $l_{min} \approx 16\ m$-nél rövidebb szegmensek lényegében nem befolyásolják az adatátvitelt.

A bitsebesség növelése viszont maga után vonja a CSMA/CD protokoll alapján az ütközési tartomány méretének csökkenését, vagyis csökkenteni kell az állomások közötti távolságot. Full duplex hálózatban ez 10 m-t jelentene, ami jelentősen korlátozná az alkalmazhatóságát. Ha meg akarjuk tartani a 100 m-es szegmenseket, növelni kell a keret méretét 512 bájtra, vagy alkalmazni kell a Gigabit Ethernet MAC protokollja által megengedett „frame bursting” módszert. Ez lehetővé teszi, hogy egy állomás maximum 64 Kbit-g terjedően elárassza egymásután keretekkel a hálózatot. Ez a megoldás áll

legközelebb a Profinet IRT rendszerhez. Így a 84 bájtos optimális vezérlési adatok minden probléma nélkül továbbíthatóak, ha legalább öt állomás kapcsolódik a kontrollerhez. Ebben az esetben a minimálisan elérhető ciklusidő elméleti értéke a (6.15) alapján kb. 6,3 μs lesz.

6.2.2. Az EtherCAT ciklusidő

Az EtherCAT rendszernél a ciklusidő meghatározása meglehetősen összetett feladat. A 2.3.2. fejezetben bemutatott keretstruktúra alapján legelőször meg kell határoznunk, hogy hány telegram helyezhető el egy keretben. Figyelembe véve, hogy mindegyik telegramhoz tartozik egy 10 bájtos fejléc (*HD*) és egy 2 bájtos ellenőrző rész (*WC*), következik, hogy egy kereten belül elhelyezhető telegramok számát (*k*) a következő összefüggéssel határozhatjuk meg:

$$k = INT[\frac{Max\_eCAT\_Data\_Size}{Header + Data + Working\_Counter}] = INT[\frac{1498}{12 + d_{DATA}}] \qquad (6.17)$$

(*Megjegyzés:* A fenti relációban INT[ ] a tört egész részét jelenti.)

Ha már ismerjük az egy keretben elhelyezhető telegramok számát (*k*) és ez a szám kisebb, mint az eszközök száma, akkor meghatározhatjuk, hogy a teljes adatmennyiség továbbításához hány keretre (n_F) van szükség.

$$n_F = INT\left[\frac{N}{k}\right] + 1; \qquad \text{ha } N > k \qquad (6.18)$$

Ellenkező esetben ($N \leq k$) a teljes adatmennyiség elfér egyetlen egy keretben és természetesen ilyenkor *k* egyenlő lesz *N*-el.

Figyelembe véve az előbbi kritériumokat és azt a tényt, hogy a minimális Ethernet keretméret nem lehet kisebb, mint 64 bájt, 3 esetet különböztethetünk meg:

1) $N = k$; és a teljes EtherCAT telegramok mérete (fejléccel és működés számlálóval együtt) nem haladja meg a 44 bájtot.
   Ez a legegyszerűbb eset, ilyenkor a teljes keretméret 64 bájt, az előtaggal (7+1 bájt) és a keretek közti réssel (IFG) együtt pedig 84 bájt lesz. A ciklusidő (T_{eCAT}) meghatározásához figyelembe kell venni még a csomóponti eszközök, valamint a közeg késleltetési idejét ($T_D = 1{,}35\ \mu s$ az EtherCAT rendszernél, és T_M pedig a már előzőekben ismert érték). Következik:

$$T_{eCAT} = \frac{672}{b} + N \cdot (T_D + T_M) \qquad (6.19)$$

   <u>*Megjegyzés:*</u> A fenti eset a valóságban gyakorlatilag nem, vagy csak nagyon ritkán fordul elő, ugyanis például 3 csomópont és 4 bájt vezérlési adat esetében a keretterhelés a fejlécekkel együtt már meghaladja a 44 bájtot!

2) $N = k$ vagyis a teljes vezérlési adatmennyiség elfér egyetlen egy keretben ($n_F = 1$). Ez a legoptimálisabb eset, de itt is figyelembe kell vegyük az EtherCAT telegramok mellett az Ethernet fejléccet és egyéb járulékos bájtokat. A keret nagysága változó lehet, de nem haladja meg a maximális méretet. A ciklusidő a következő összefüggéssel számítható ki:

$$T_{eCAT} = \frac{8N \cdot (12 + d_{DATA}) + 320}{b} + N(T_D + T_M) \cdot \qquad (6.20)$$

3) A legáltalánosabb eset az, amikor a csomópontok száma nagyobb, mint ahány telegram elhelyezhető egy keretben ($N > k$). A (6.17) alapján kapjuk meg, hogy hány keretre van szükség a teljes vezérlési adatmennyiség továbbítására. A ciklusidőt pedig az alábbi általánosított összefüggéssel határozhatjuk meg:

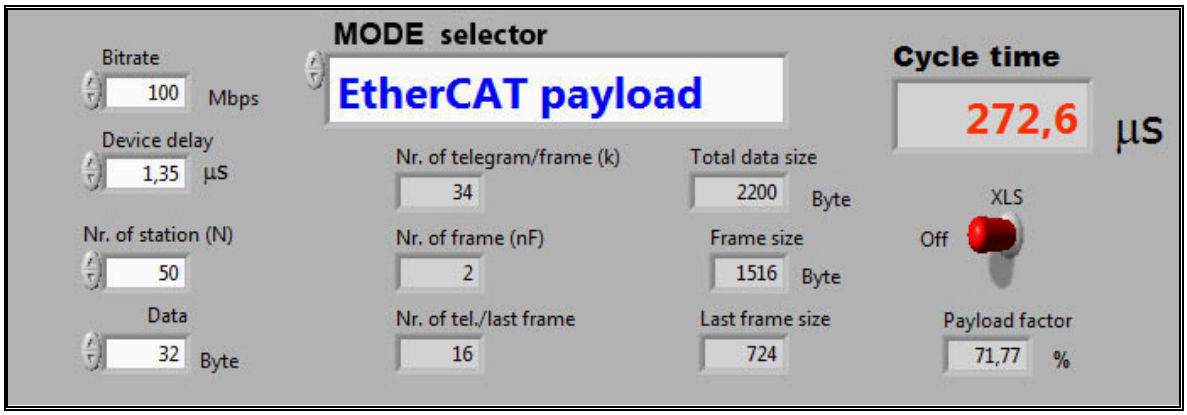
$$T_{eCAT} = \frac{8}{b}\left[40 \cdot n_F + N \cdot (12 + d_{DATA})\right] + N \cdot (T_D + T_M) \qquad (6.21)$$

Megjegyzés: Az eszközkésleltetési idő (T_D) itt is függ a bitsebességtől. 100 Mb/s-os hálózatban 1,35 µs 1Gb/s-es hálózatban pedig 0,85µs. (A gyártó adatai)

6.3. A számítási modell kidolgozása LabView 8.2. fejlesztői környezetben

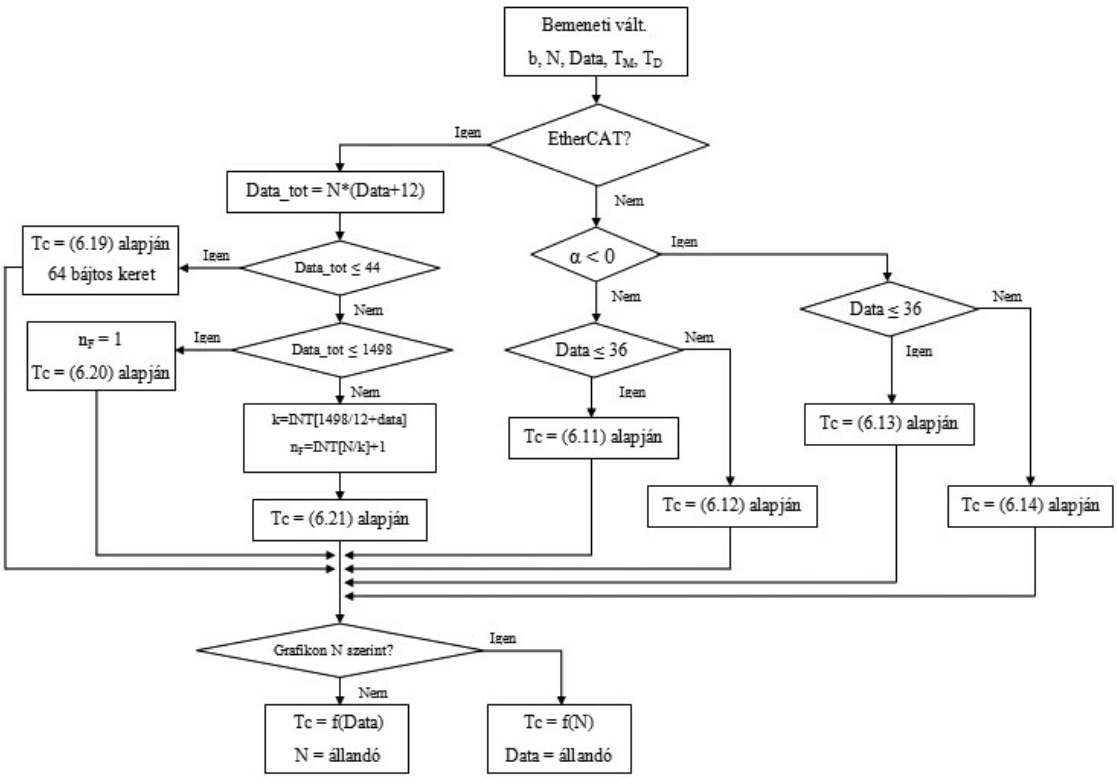
A LabViewban kidolgozott algoritmus mindkét, az előzőekben bemutatott (6.2.1; 6.2.2.) megoldást kezelni tudja és minden esetben lehetőséget biztosít a ciklusidő meghatározására gyakorlatilag korlátlan számú csomóponti eszköz (N) és 1-től egészen 1486 bájtig terjedő keretterhelés függvényében. Egyaránt alkalmazható Fast Ethernet és Gigabit Ethernet hálózatra is, ennek megfelelően beállíthatók az eszközkésleltetési idők. Az eszközök közötti szegmenshossz alapértelmezésben a maximális 100 méter hosszúságú. Ez bizonyos esetekben, adott menüpontok alatt változtatható. A ciklusidő kiszámítása mellett, ábrázolni tudja ennek változását a csomópontok, illetve a keretterhelés függvényében, a keletkező adatokat pedig akár Excel fájlban is eltárolja.

A 6.11. ábrán az alkalmazás EtherCAT üzemmódjának egyik kezelő felülete látható. Számos hasznos információt is megjelenít, mint például a keretek számát, a keretekbe beágyazott telegramok számát, a teljes továbbítandó adatmennyiséget, vagy akár a keretterhelési tényezőt. Ezen kívül alkalmas még a két módszer ciklusidő-változásának egyidejű megjelenítésére, valamint ezek összehasonlítására különböző szituációkban.



6.11. ábra. Az alkalmazás egyik kezelőfelülete

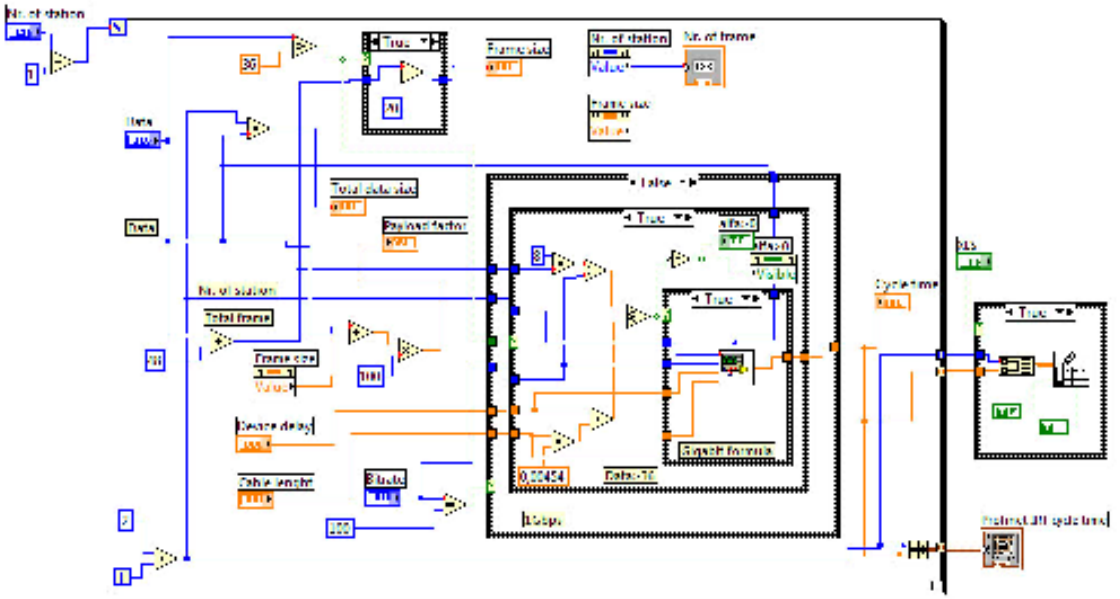
Az alkalmazás algoritmusa a 6.2.1. és 6.2.2. fejezetekben bemutatott matematikai modelleken és számításokon alapszik. A LabView környezetben implementált modell alkalmassá teszi a kidolgozott eljárások vizsgálatát különböző szituációkban. Modellezhetőek mindazok a körülmények, amelyek befolyásolják a szigorúan valós idejű ipari Ethernet rendszerek ciklusidejét. A következő ábrán a ciklusidő-számítási modell algoritmusának egy részlete látható.



6.12. A ciklusidő számítási algoritmusa

A LabView fejlesztői környezet kiválóan alkalmas műszaki modellezési, szimulációs és mérési feladatok implementációjára. A grafikus programozási eljárásoknak köszönhetően a kód elkészítése egyszerű. Számos beépített eljárás és függvény felhasználható, de nagyon egyszerűen készíthetünk saját alprogramokat és függvényeket.

A következő ábrán a Profinet IRT ciklusidő csomópontok szerinti számítási és ábrázolási kódja látható.



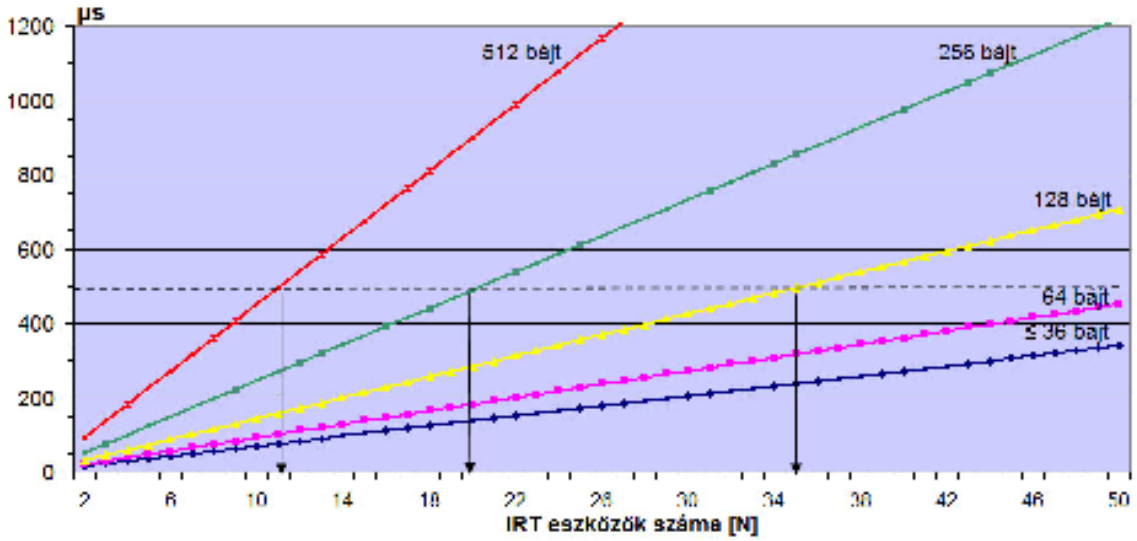
6.13. Ciklusidő-számítási kódrészlet

6.4. A vizsgált szituációk elemzése

Ebben a fejezetben összefoglalom és kiértékelem azokat az eredményeket, amelyeket az elkészített alkalmazás segítségével rögzítettem. Elsőként a Profinet IRT rendszerrel kapcsolatos eredményeket mutatom be, majd pedig az EtherCAT rendszer kerül kiértékelésre. Végül pedig az elviekben is különböző két rendszer összehasonlítására kerül sor különös tekintettel a Fast Ethernet és a Gigabit Ethernet rendszerekben történő alkalmazásukra.

6.4.1. A Profinet IRT Fast Ethernet hálózatban [S8]

Az előzőek alapján egyértelműen következik, hogy a Profinet IRT rendszer valós idejű korlátai nagymértékben függnek a hálózatba kapcsolt eszközök számától és a keretterhelés mértékétől. Az eszközök számának növelésével arányosan nő a ciklusidő. Ezt még jelentősen növeli a keretterhelés mértéke akkor, amikor a felhasználói adatok nagysága meghaladja a 36 bájtot (6.14 ábra). Láthatjuk, hogy a keretkihasználtság növelése nem vezet eredményre, de különösebben nincs is rá szükség, mert a vezérlési adatok nagysága ezeknél a rendszereknél csak ritkán haladja meg a 100 bájtot.

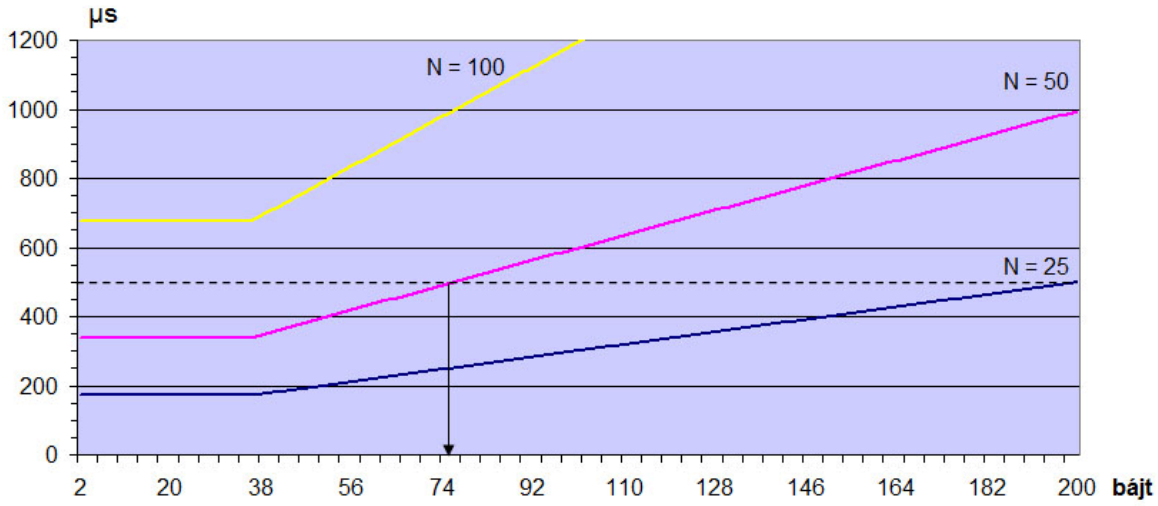


6.14. ábra. A ciklusidő növekedése az eszközök száma szerint

A 6.14.-es ábrán látható görbék meredekségét jelentősen befolyásolja a keretterhelés. Míg a 36 bájt vagy annál kisebb vezérlési adatmennyiség esetében a ciklusidő csomópontonként kevesebb mint 7 µs-al növekszik, addig 512 bájtnál ez az érték 43µs, vagyis több mint hatszorosa. Az is látható, hogy 50 csomóponti eszközt használva 64 bájtnál még maradéktalanul teljesülnek a Profinet IRT ciklusidővel kapcsolatos feltételek, sőt még növelni is lehetne az állomások számát. Viszont 128 bájnál már csak legfeljebb 35 állomás csatlakoztatható. Az állomások száma még tovább korlátozódik 256 és 512 bájtnál 20 illetve 11 csomópontra.

A Profinet IRT kommunikációs stratégiájának megfelelően az 500 µs-os ciklusidő a legkedvezőbb esetben is maximálisan 73 IRT eszközvezérlő csatlakoztatását teszi lehetővé.

Hasonlóképpen elemezhetjük a ciklusidő változását a keretterhelés függvényében is. 36 bájtos keretterhelésig a ciklusidő nem változik és a csomópontok függvényében a minimális értékű. Ezt a változás látható a 6.15. ábrán.

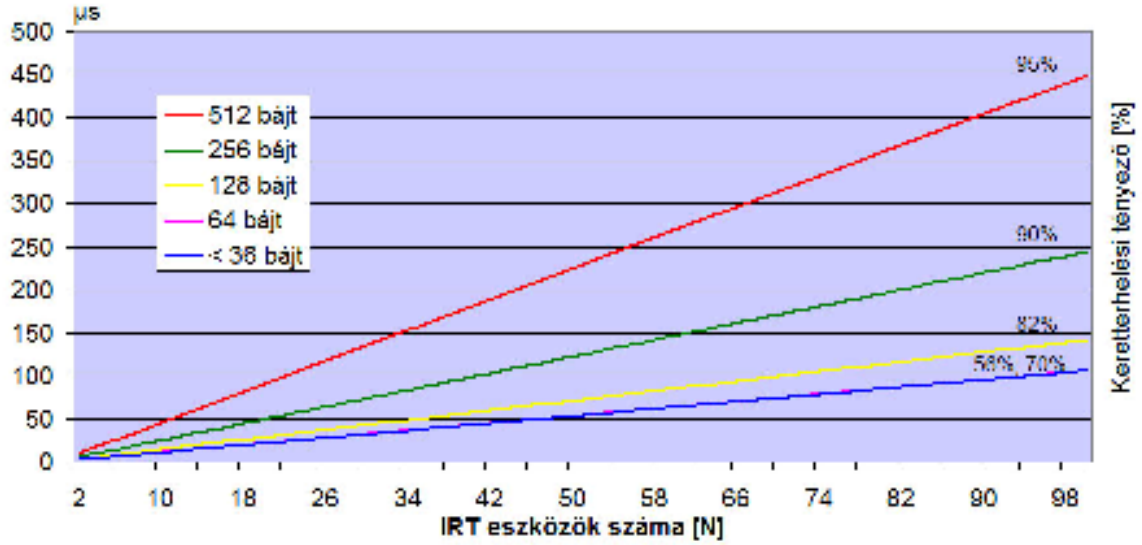


6.15. ábra. A ciklusidő változása a keretterhelés függvényében 200 bájtig

Az előző ábra szerint 25 csomópontig teljesíthető akár 200 bájt vezérlési adatmennyiség továbbítása. Ha a csomópontok számát megduplázzuk, a szigorúan valós idejű működési feltétel már csak 74 bájtig biztosított. Az is látható, mint ahogy az előzőekben már szó volt róla, hogy 100 csomópontnál ez a feltétel már nem biztosított.

6.4.2. A Profinet IRT Gigabit Ethernet hálózatban [S20]

Ebben a fejezetben először azt vizsgáltam, hogy viszonylag alacsony vezérlési adatmennyiségnél (≤ 36 báj, 64 bájt) valamint a nagyobb értékeknél, például 128, 256, 512 bájt továbbításakor hogyan változik a ciklusidő az IO eszközök számának függvényében Gigabit Ethernet hálózatban (6.16. ábra).



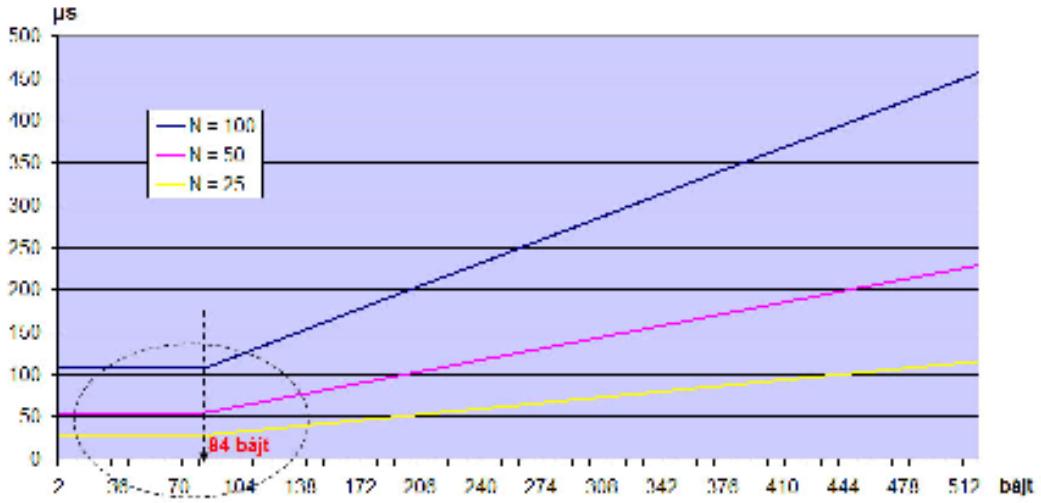
6.16. ábra. A ciklusidő változása a csomópontok függvényében Gigabit Etherneten

Mindegyik esetben a maximális 100 méter a szegmensek hossza. Azt tapasztalhatjuk, hogy állandó értékű keretterhelés mellett, akárcsak a Fast Ethernet hálózatban, a ciklusidő lineárisan növekszik az IO eszközök számának növekedésével. A bitsebesség növelésének köszönhetően viszont a ciklusidő jelentősen csökken. A nagyobb keretterheléseknél (128, 256 bájt) majdnem 10-szer kevesebb, míg a kisebb adatmennyiségnél (36, 64 bájt) csak kb.

6,5-szer lesz kevesebb. Ez azért van így, mert a (6.16) összefüggésből adódóan ebben az esetben 100 méternél rövidebb szegmenshossz lenne az optimális. Továbbá, ha $\alpha<0$, vagyis 84 bájtnál kisebb adatmennyiségnél a (6.15) szerint a ciklusidőt jelentősen befolyásolja a közeg késleltetési ideje (a szegmensek hossza). Ugyancsak ebben az esetben a ciklusidőt számottevően nem befolyásolja a keretméret, ezért van az, hogy a 6.16. ábrán a 36 és a 64 bájtos görbék gyakorlatilag egybeesnek.

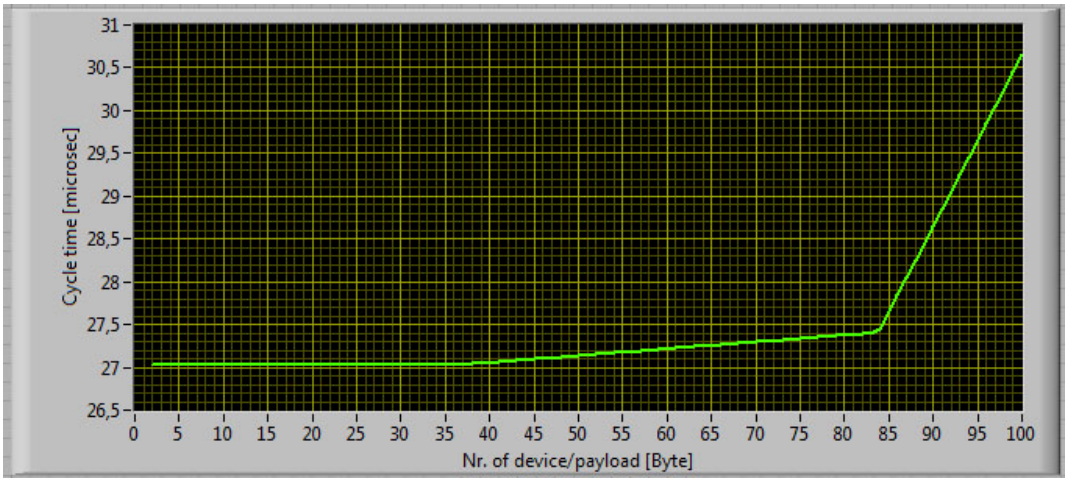
Az előbbiek alapján mindenképpen előnyösnek látszik a Profinet IRT esetében a Gigabit Ethernet hálózat alkalmazása. Ilyen körülmények között akár 100 csomóponti IO eszköz mellett, 256 bájt keretterheléssel is a ciklusidő kevesebb, mint 250µs. Vagyis akár 50%-os IRT időréssel is bőven a szigorúan valósidejű időkorláton (1ms) belül vagyunk. Nagyobb időréssel még az 512 bájos terhelés is biztosítható 100 csomópont mellett.

A továbbiakban a keretterhelés szerinti ciklusidő változást elemeztem. Ez látható a 6.17. ábrán.



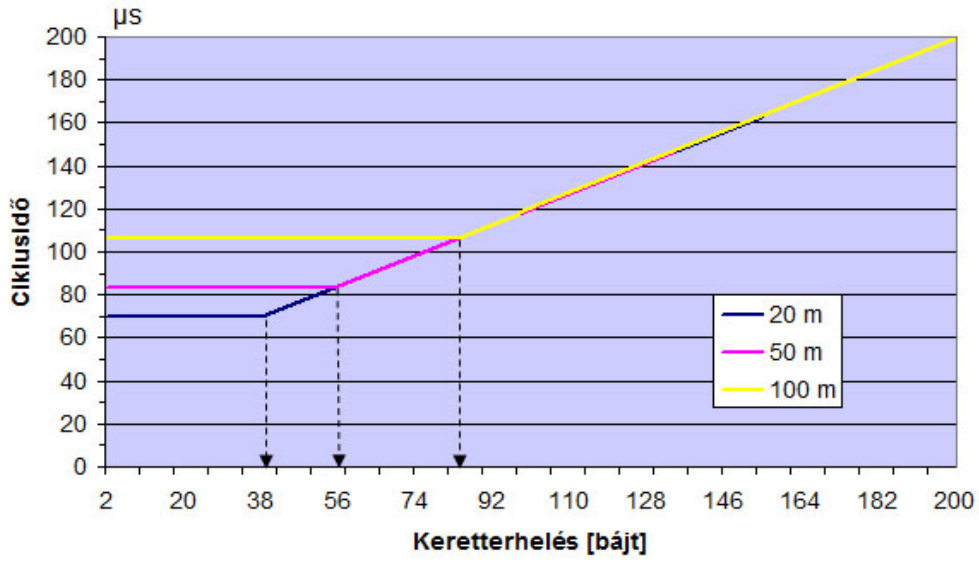
6.17. ábra. A Profinet IRT ciklusidő változása a keretterhelés arányában

A fenti ábra kiemelt részén azt láthatjuk, hogy a (6.16) szerint meghatározott optimális keretterhelésig (100 méteres szegmenseknél 84 bájt) a ciklusidő gyakorlatilag alig változik, vagyis független a keretterheléstől (6.18. ábra). Ez egyértelműen előnyt jelent – a ciklusidő csökkenése mellett – a Fast Ethernettel szemben, ahol csak 36 bájt alatt kaptunk keretterheléstől független értékeket. A 36 és 84 bájt közötti változás a LabView alkalmazás megjelenítő rendszerén kevesebb mint 0,4µs, 25 csomópont esetében.



6.18. ábra. 84 bájt keretterhelésig a ciklusidő független a keretmérettől (N = 25).

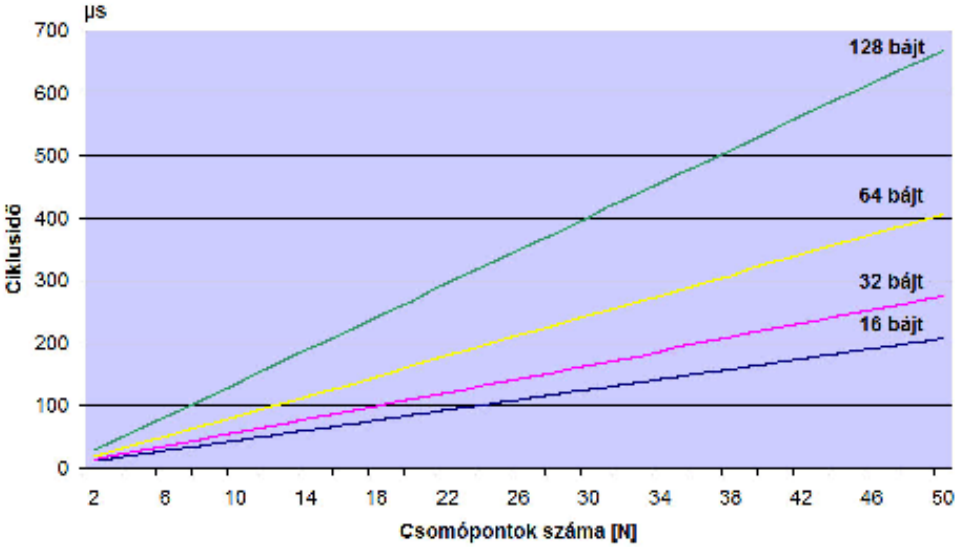
Érdemes megvizsgálni még, az optimális keretterhelés változását a szegmensek hosszúságának függvényében (6.19. ábra). Az eredmény első ránézésre meglepőnek tűnik, hogy a szegmensek hosszának a növekedésével, növekszik az optimális keretterhelés mértéke is, 36 bájttól egészen 83 bájtig. (A 36 bájt 16 méternél valósul meg.) De ha a (6.16) reláció igaz, akkor valóban ez kell történjen.



6.19. ábra**.** Az optimális keretterhelés változása a szegmenshossz szerint

6.4.3. Az EtherCAT rendszer ciklusideje Fast Ethernet hálózatban [S5]

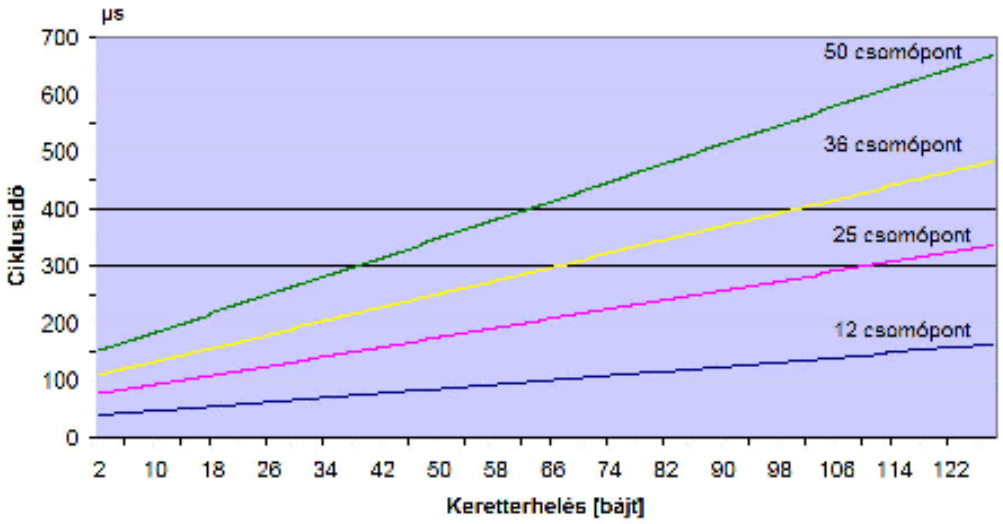
Először azt az esetet vizsgálom, amikor a ciklusidő a csomóponti eszközök száma szerint változik, például 50 csomópontig különböző, 16, 32, 64 illetve 128 bájt keretterhelés mellett. (6.20. ábra).



6.20. ábra. Az EtherCAT ciklusidő változása a csomópontok száma szerint

A szigorúan valós idejű rendszereknél, így az EtherCAT-nél is a ciklusidő legfeljebb 1ms, vagy ennek törtrésze 1/2, 1/4, 1/8, stb. lehet. 50 csomópont esetében 16 bájtos eszközönkénti keretterheléssel még a 0,25 ms-os ciklusidő is megvalósítgató. Nagyobb keretterheléseknél, például 64 bájtnál már csak 0,5 ms-os ciklusidő biztosítható.

A 6.21. ábrán ugyancsak a ciklusidő alakulását láthatjuk, csak most a keretterhelés függvényében.



6.21. ábra. Az EtherCAT ciklusidő változása a keretterhelés szerint

Az előző ábrákból egyértelműen látszik, hogy mindkét meghatározó tényező (N, Data) egyértelműen befolyásolja a ciklusidő növekedését. A növekedés mértéke azonban sokkal alacsonyabb, amikor az EtherCAT terhelést növeljük.

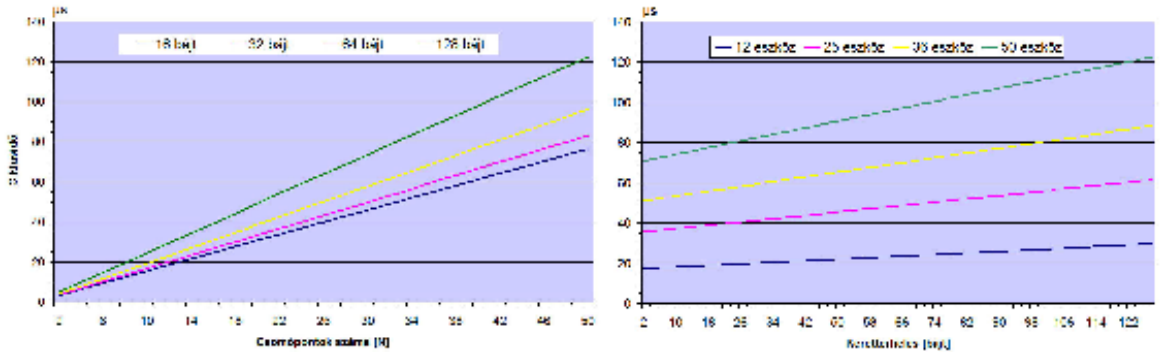
Mindkét ábrán, nyomon követhetjük, hogy azonos terhelés mellett, amikor az eszközök számát megduplázzuk, akkor a ciklusidő is kb. duplájára nő. Például 16 bájt terhelésnél, ha az eszközök számát növeljük 24-ről 48-ra, akkor a ciklusidő 101µs-ról 200µs-ra növekszik. De ugyanez mondható el 128 bájtos terhelésnél is. Itt is kb. duplájára növekszi a ciklusidő. Viszont, ha a terhelést duplázzuk (pl. 32-ről 64-re), azonos eszközszám mellett, a ciklusidő jóval kisebb arányban növekszik, átlagban mintegy 50%-al. Ebből azt a következtetést vonhatjuk le, hogy az EtherCAT rendszer busz frissítési ciklusideje kevésbé érzékeny a vezérlési adatok nagyságának a változására. Ez abból is adódik, hogy a ciklusidő egy jelentős összetevője nagymértékben függ az eszközök által meghatározott késleltetési időtől.

Azt is észre kell vegyük, hogy 128 bájtos terhelésnél, már csak körülbelül 36 eszközt tudunk kiszolgálni, hogy ne haladjuk meg az 500 µs-os korlátot, amely 50-50%-os időréssel (50% EteherCAT, 50% UDP/IP adatok) összesen, a szigorúan valós idejű rendszereknél a maximális 1ms-os buszfrissítési időt adja.

6.4.4. Az EtherCAT rendszer viselkedése Gigabit Ethernet hálózatban

A továbbiakban azt vizsgáltam, hogy milyen hatással van a ciklusidőre a bitsebesség növelése. Várhatóan a keretek továbbítási ideje kb. tízszeresére csökken. De mi a helyzet az eszközök késleltetési idejével? Ez bizony csak kb. a felére csökken, így $t_D = 0{,}85\ \mu s$ értékkel számolok [58].

A 6.22.-es ábrákon bemutatott elemzésekből látható, hogy jelentősen, mintegy öt és félszeresére csökkent a ciklusidő. A görbék jellege nem változott. A baloldali ábrán látható, hogy ugyanúgy duplájára nő a ciklusidő, amikor az eszközök száma megduplázódik. Viszont a jobb oldali ábrán azt is észre kell vegyük, hogy ha az EtherCAT adatokat duplázzuk meg, például 32-ről 64-re, a ciklusidő már csak kb. 15%-al növekszik.



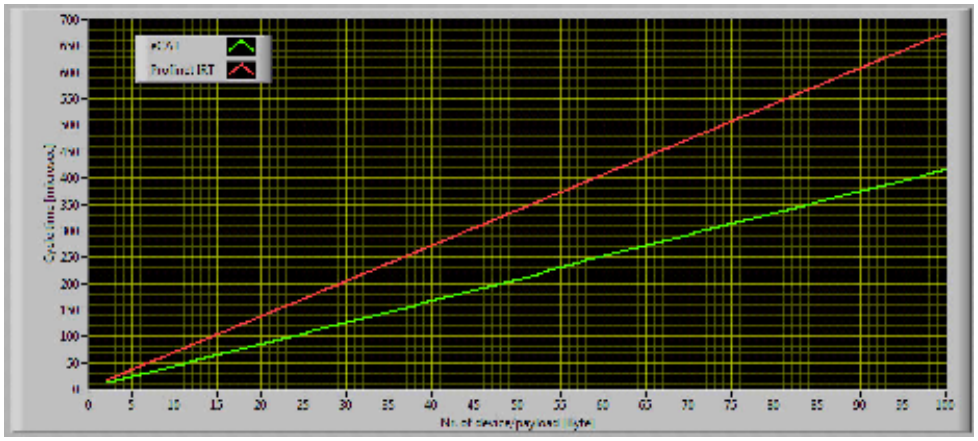
6.22. ábra. A ciklusidő alakulása Gigabit Ethernet hálózatban

Összefoglalva az előzőeket kijelenthető, hogy az EtherCAT a ciklusidőt illetően 100 Mb/s-os hálózatban igen jól viselkedik főleg kisebb adatterhelések (16, 32, 64 bájt) esetében, a bitsebesség növelése viszont nem hozza meg a várt eredmény, mert a ciklusidő csak kb. 5,5-szörös csökkenést ér el. Viszont az is látható, hogy 128 bájtnál, nemhogy 50, de akár 100 eszközt is képes kiszolgálni a fentebb már említett időkorláton belül.

6.4.5. A Profinet IRT és az EtherCAT rendszerek összehasonlítása

A 6.3. fejezetekben bemutatott LabView alkalmazásban lehetőség van arra, hogy látványos összehasonlítást végezzünk a két rendszer viselkedését illetően különböző szituációkban. Az összehasonlítás valójában nem is a két rendszer közötti különbséget, hanem inkább az általuk alkalmazott módszereket hivatott elemezni [S8]. Ezek ismeretében lehetőség nyílik olyan módszerek kidolgozására, amelyek bármilyen körülmények között teljesítik a szigorúan valós idejű ipari Ethernet rendszerek működési feltételeit.

Először azt vizsgáltam, hogy viszonylagosan alacsony vezérlési adatmennyiség (16 bájt) továbbításakor hogyan változik a ciklusidő az IO eszközök számának függvényében Fast Ethernet, illetve Gigabit Ethernet hálózatban. Mindkét esetben a maximális 100 méteres szegmenseket használtam N = 100 csomópontba kapcsolt eszközvezérlővel. Az eszközkésleltetési idők beállított értékei 3 µs a Profinet IRT hálózatban, illetve 1,35 µs az EtherCAT rendszer esetében. A 6.23. ábrán a két rendszer ciklusidejének változása látható 100 Mb/s bitsebesség mellett.

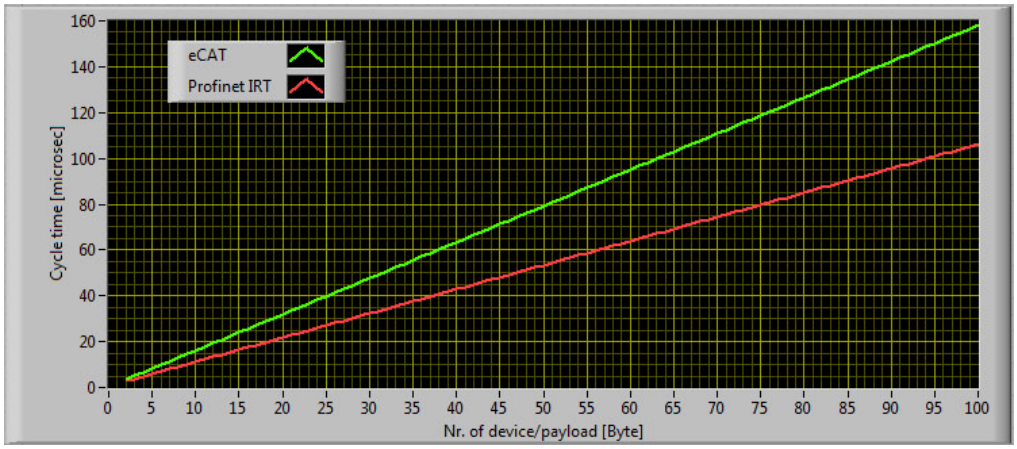


6.23. ábra. A ciklusidők változása a csomópontok száma szerint Fast Ethernet hálózatban

A fenti ábra szerinti eredmények ebben a szituációban egyértelműen az EtherCAT rendszert hozza ki előnyösen. Jól látható például, hogy 250 µs-os ciklusidőn belül akár 60

EtherCAT csomópont is kiszolgálható, míg ugyanez a Profinet IRT-nél csak 36 IRT eszközig biztosítható. Ez úgy is értelmezhető, hogy az EtherCAT alacsony keretterhelés mellett kb. 1,6-szor gyorsabb, mint a Profinet IRT. Ez az arány állandónak tekinthető, hiszen alig változik a csomópontok szerint. 10 csomópontnál 1,61; 200 csomópontnál valamivel kevesebb, mint 1,64.

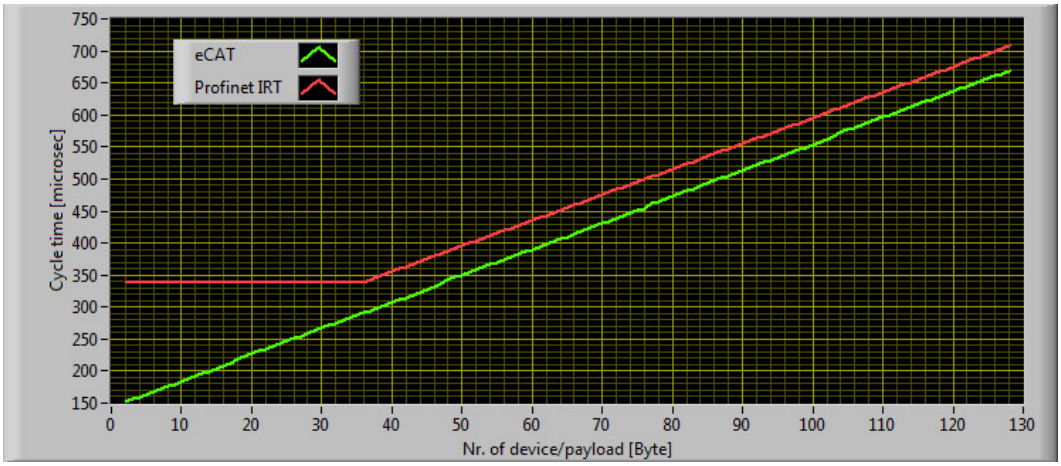
Radikálisan megváltozik a helyzet Gigabit Ethernet hálózatban. Ez látható a 6.24. ábrán 32 bájtos keretterhelés mellett.



6.24. ábra. A Profinet előnye Gigabit Ethernet hálózatban.

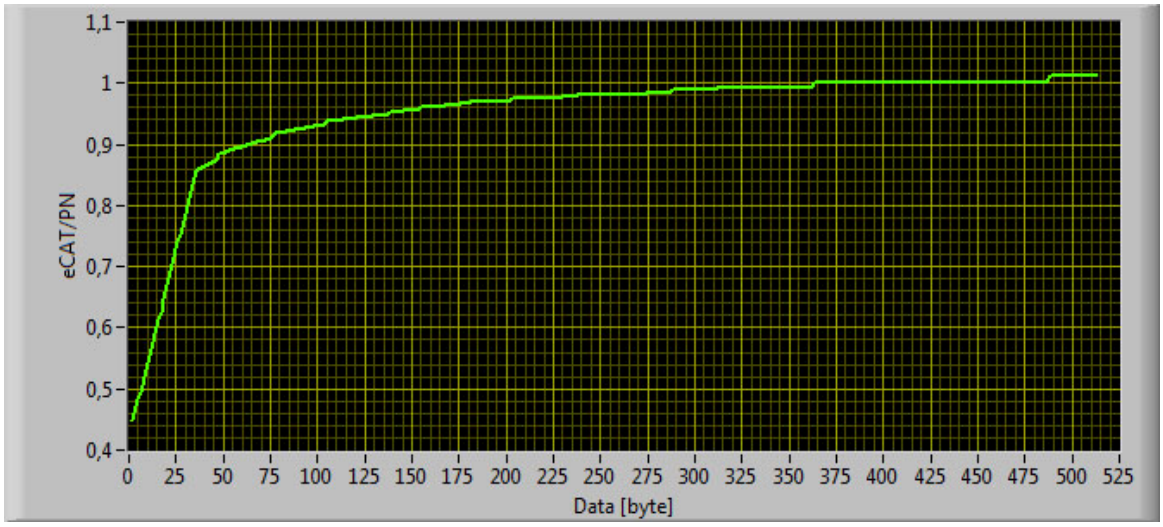
A telegramküldési stratégiának köszönhetően ebben a szituációban a Profinet IRT a nyerő. A ciklusidő arány ugyan kevesebb, mint az előző szituációban, mert csak kicsivel több, mint 1,4 de ugyancsak állandónak tekinthető. Viszont ez a rendszer sem tudja teljes egészében hasznosítani a tízszeres bitsebesség növekedést. A közeg késleltetési ideje ugyan független a bitsebességtől, de az IO eszközök késleltetési ideje csak kismértékben csökken. A Profinet IRT-nél 3-ról 0,6 mikroszekundumra, míg az EtherCAT-nél 1,35-ről 0,85-re. Ugyanakkor kétségtelen, hogy mindkét rendszer esetében biztosítható a 250 µs-os ciklusidő, akár száznál több IO eszközvezérlő estében is. EtherCAT-nél 160, Profinet IRT-nél pedig 230 csomópont tud eleget tenni ennek a feltételnek.

Az előzőektől kicsit eltérő, de hasonló helyzet alakul ki, ha megvizsgáljuk a két rendszer viselkedését a keretterhelés növelését követően. Ennek a szituációnak megfelelően egy átlagos bonyolultságú, 50 darab IO eszközből álló, egyenként 50 méteres szegmensekkel összekapcsolt busz topológia szerinti hálózatot feltételezek ugyancsak 100 Mb/s, illetve 1 Gb/s bitsebességek mellett.



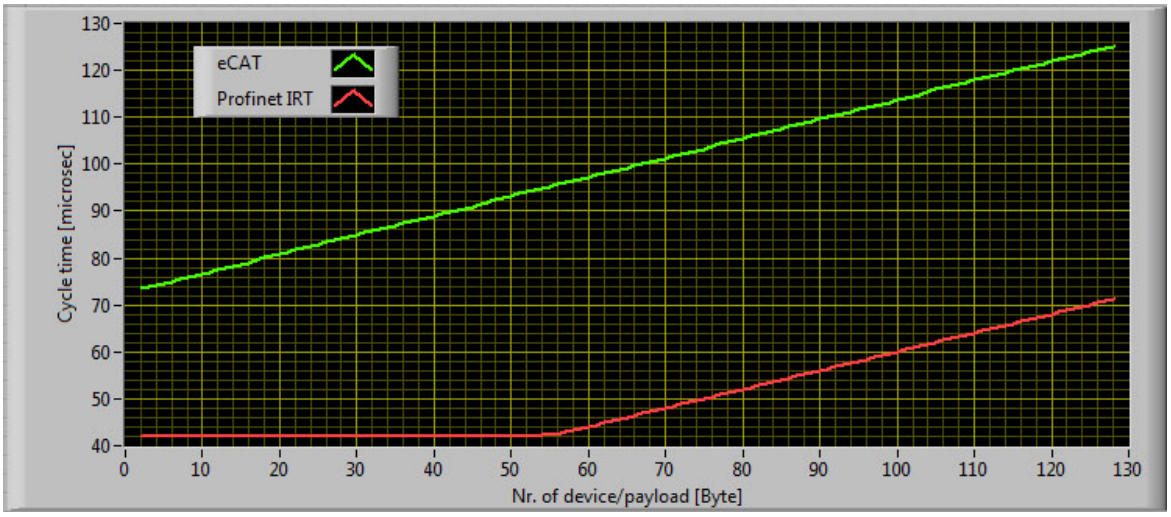
6.25. ábra. A keretterhelés szerinti változások 100 Mb/s bitsebességnél.

A 6.25. ábrán az látható, hogy a 100 Mb/s-os hálózatban az EtherCAT főleg a kisebb keretterheléseknél, egészen 36 bájtig jelentősen jobb ciklusidőt biztosít, mint a Profinet IRT. 36 bájtnál az EtherCAT javára még megmaradó mintegy 50 µs-os ciklusidő különbség a további keretterheléssel csak nagyon lassan csökken. Az alkalmazott modell számításai alapján a két rendszer közötti különbség csak 364 bájt keretterhelés fölött egyenlítődne ki, majd 480 bájt fölött mutatna némi előnyt a Profinet IRT javára. Ezt a helyzetet nagyon jól szemlélteti a 6.26. ábra. Ekkor azonban a ciklusidő már jóval 1 ms fölött van, ami már alkalmatlan a szigorúan valós idejű rendszerek adattovábbítására.

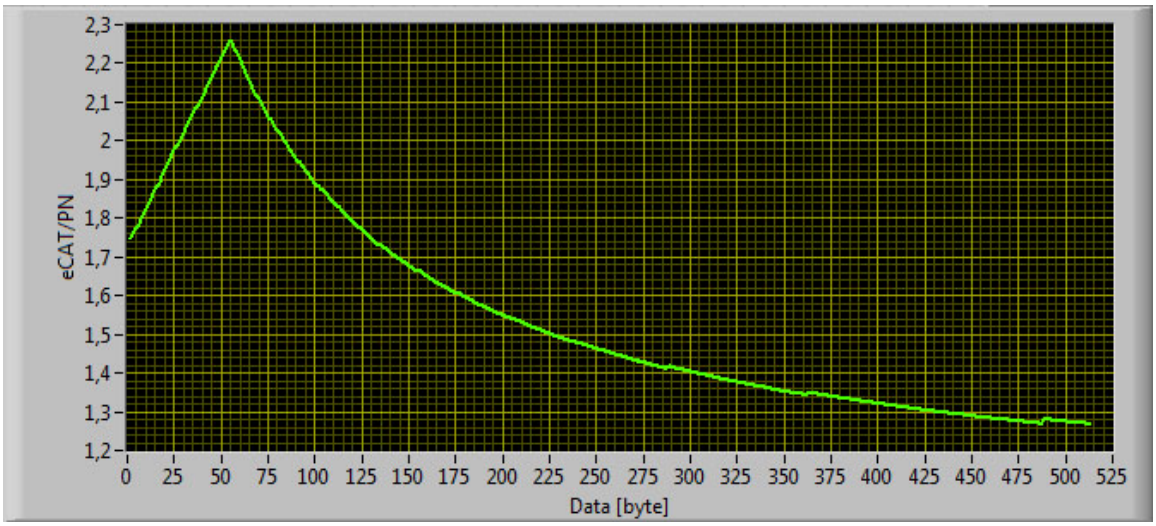


6.26. ábra. Az EtherCAT és Profinet IRT ciklusidő arányának változása
a keretterhelés függvényében Fast Ethernet hálózatban

Egészen más a helyzet áll elő a gigabites hálózatban (6.27. ábra). Itt a Profinet IO előnye már egész alacsony keretterhelés mellett is megmutatkozik, majd pedig fokozódik egészen a (6.16) alapján meghatározott optimális értékig. Jelen esetben ez 55 bájt, de 100 méteres szegmenseknél 84 bájt lenne. (6.2.1.4. fejezet) Ezen értékek fölött a Profinet IRT előny csökkenni kezd ugyan, de mindvégig előnyt fog élvezni akár egészen a maximális keretterhelésig (1490 bájt a Profinet IRT-nél). A 6.28. ábrán 512 bájtig láthatjuk az EtherCAT és a Profinet IRT ciklusidő arányának változási tendenciáját 50 csomópont esetében.



6.27. ábra. A keretterhelés szerinti változások 1000 Mb/s bitsebességnél. (N = 50)



6.28. ábra. Az EtherCAT és Profinet IRT ciklusidő arányának változása
a keretterhelés függvényében 1Gbps bitsebességnél

6.4.6. Az elemzések alapján levonható következtetések

A bemutatott két rendszer teljesen különböző elvi és technológiai megközelítéssel azonos célt állít fel: minél több IO eszközt kiszolgálni és minél rövidebb ciklusidőt biztosítani a vezérlési adatok számára. Az alkalmazott modell rávilágít egy sokat vitatott kérdésre is, mégpedig arra, hogy érdemes-e növelni az ipari Ethernet hálózatok bitsebességét, vagyis alkalmazni a Gigabit Ethernet nyújtotta előnyöket olyan ipari körülmények között, ahol a továbbítandó adatok nagysága meglehetősen kicsi.

Két alapvető tényező került meghatározásra: a topológiából adódó késleltetési idő (T_D és T_M) valamint az adattovábbítási idő. Az előbbi független az adatmennyiségtől és a bitsebesség növelése sem okoz jelentős csökkenést, míg az utóbbit jelentősen befolyásolja az adatmennyiség, ugyanakkor csökkenthető a bitsebesség növelésével.

A bemutatott elemzések azt mutatják, hogy alacsony keretterhelés mellett, 100 Mb/s-os hálózatban, busz topológiában az EtherCAT sokkal gyorsabb, mint a Profinet IRT. Itt megjegyezném, hogy ez a topológia leginkább az EtherCAT-nek kedvez, viszont a komplexebb struktúrák, mint például a fa topológia már inkább a Profinet IRT-nek kedvezőbb. A beágyazott telegramoknak köszönhetően, – amelyek a keretekkel együtt végighaladnak az összes hálózati csomóponton – az EtherCAT nem sokat profitál a bitsebesség növeléséből, míg a Profinet IRT akár több mint kétszer gyorsabb lesz az optimális keretterhelés környékén (28 és 83 bájt között). Ez azt jelenti, hogy az önálló kereteket alkalmazó megoldásokat talán érdemes lenne a gigabit Ethernet irányába fejleszteni.

A valós idejű kommunikációban fontos szerepet játszik a megbízhatóság is. Az összes csomóponton áthaladó beágyazott kereteket alkalmazó rendszerek érzékenyebben reagálnak a kommunikációs hibákra, mint azok, amelyek közvetlen, egyéni kereteket továbbítanak. Nem mindegy az, hogy egy hibás keret ismétlésekor, csak egy állomás vagy az összes állomás adatait újra kell küldeni.

Mindezek a teljesítmény-értékelések azt mutatják, hogy az egyéni adatokat tartalmazó keretek továbbítási módszerérén alapuló rendszerek (pl. Profinet IRT) alkalmasabbak gigabit Etherneten, mint azok, amelyek a beágyazott kereteket használják (pl. EtherCAT), de ezek sem elég hatékonyak ahhoz, hogy érdemes legyen kidolgozni hozzá a technikát. Talán a két koncepció ötvözete adná az igazi megoldást, amikor a

keretek mérete minden esetben az optimális adatértékhez igazodna, vagy új elveken alapuló módszereket és protokollokat kell kidolgozni a jövőben.

6.5. Új tudományos eredmények

A szigorúan valós idejű kommunikációs követelményeknek minden szempontból eleget tevő, sok csomópontot tartalmazó tesztrendszer kiépítése meglehetősen időigényes és költséges feladat. Ebben a fejezetben megterveztem és kifejlesztettem egy olyan számítógépes modellt, amelynek segítségével mindkét, elviekben teljesen különböző technikai megoldás vizsgálható bármilyen szituációban. Az alkalmazás segítségével könnyen eldönthető, hogy adott technológiai követelményekhez milyen megoldást célszerű tervezni. A módszer egyaránt alkalmas Fast Ethernet és Gigabit Ethernet hálózatbeli szituációk elemzésére.

A Gigabit Ethernet hálózatban elvégzett elemzések alapján megállapítottam, hogy azonos körülmények között a két rendszer teljesen eltérő módon viselkedik. Ebben az esetben a ciklusidő függővé válik a szegmensek hosszától. Míg a beágyazott telegramokat tartalmazó kerettovábbításnál jelentősen növekszik a ciklusidő, addig a külön-külön címzett kereteket alkalmazó megoldásnál az optimális keretterhelésen túl szinte alig tapasztalható növekedés a szegmenshosszt illetően. Ez utóbbi esetben viszont *a szegmenshosszal arányosan növekszik az optimális keretterhelés*.

A gyakorlatban a legtöbb szigorúan valós idejű ipari Ethernet rendszer több eszközvezérlőt, de viszonylag kis mennyiségű (16, 32 bájt) vezérlési adatokat használ. Az alkalmazott modell eredményeinek elemzése alapján az alábbi tézist fogalmaztam meg.

***4. TÉZIS** [S5, S7, S8, S15, S20, S21]*

Megterveztem és kifejlesztettem egy olyan új számítógépes alkalmazást, amely mindkét alapmegoldás esetében képes meghatározni az optimális buszfrissítési időt a csomópontok számától, a keretterheléstől és a bitsebességtől függően.

4.1. Altézis. *Fast Ethernet hálózatban az alacsony keretterhelést igénylő, sokcsomópontos, szigorúan valós idejű ipari Ethernet rendszereknél az optimális ciklusidőt a beágyazott telegramokat alkalmazó megoldás biztosítja*.

4.2. Altézis. *Fast Ethernet hálózatban a csomópontonként külön-külön címzett keretekben elhelyezett, 36 bájtnál nem nagyobb, de változó keretterhelést feltételező vezérlési adtok továbbítása állandó ciklusidőt biztosít*.

4.3. Altézis. *A buszon történő ütközések elkerülésére vonatkozó szabályokat figyelembe véve meghatároztam, hogy Gigabit Ethernetes hálózatban 84 bájt optimális keretterhelés mellett, 100 méteres szegmenseket használva legalább 5 csomópontot kell tartalmaznia a Profinet IRT típusú rendszereknek. A beágyazott telegramokat tartalmazó rendszereknél legalább 512 bájtos keretméret biztosítása függ a csomópontok számától, illetve telegram mérettől.*

ÖSSZEFOGLALÁS

A XXI. század első évtizedének egyik kimagasló fejlődési területe kétségtelenül a kommunikációs kapcsolatok, ezen belül pedig számítógépes hálózatok és az internet elterjedése volt. Egyes becslések szerint az internetes csomópontok száma hamarosan eléri a 15 milliárdot, 2020-ra pedig meghaladhatja az 50 milliárdot is. A hagyományos kommunikációs csatornák eszközeit egyre inkább felváltják az internet alapú eszközök. Nem csoda tehát, ha az ipari kommunikációs eszközök is ebbe az irányban fejlődnek.

A korszerű, gyors Ethernet alapú hálózatok, ma már lehetővé teszik, hogy megfelelő kommunikációs stratégiák betartása mellett, akár a szigorúan valós idejű ipari Ethernet rendszerek is teljesíteni tudják a technológiai követelményeket anélkül, hogy befolyásolnák az egyébként nyílt rendszert.

Az általam elkészített doktori disszertáció azokat a módszereket és eljárásokat elemzi, amelyek segítségével biztosítható az egyébként nem determinisztikus Ethernet hálózaton az ipari irányító rendszerek valós idejű viselkedése.

Az értekezésemben bemutatott módszerek jelentős részét konkrét gyakorlati mérési eredményekkel igazoltam. Több éves munkám eredményeként, a több tucat lehetséges szituáció szerinti közel százezer mérési eredmény kiértékelésén alapuló megállapítások reményeim szerint segítséget nyújtanak a területen dolgozó, fejlesztő és üzemeltető mérnökök számára, akik nap, mint nap igyekeznek minél jobb, üzembiztosabb és olcsóbb ipari kommunikációs eszközöket készíteni vagy ezeket üzemeltetni.

Ugyancsak a valós idejű ipari Ethernet rendszerek tervezését segítheti az általam kidolgozott Profinet IRT és EtherCAT rendszerek számítógépes szimulációs modellje. Ennek segítségével bármilyen bonyolultságú lineáris buszrendszerű, sok csomópontos, szigorúan valós idejű hálózat vizsgálható virtuális körülmények között. Segítségével meghatározhatók mindazok a kritikus paraméterek, amelyeket tervezéskor figyelembe kell venni. Az eljárás továbbfejleszthető, bármikor újabb ipari Ethernetes rendszer hozzáadható és implementálható.

SUMMARY

One area of outstanding development in the first decade of the 21st century was undoubtedly the improvement in communication technology, more specifically the spread of the Internet and computer networks. According to some estimates, the number of Internet connections will reach 15 billion soon. By 2020 it may even exceed 50 billion. The traditional devices of communication channels are increasingly being replaced with web-based tools. It is not a surprising development that industrial communication devices are also evolving in that direction. The modern, fast Ethernet-based networks, subject to compliance with appropriate communication strategies, now allow even strictly real-time Industrial Ethernet systems to meet the technological requirements without affecting the otherwise open system. In my PhD thesis, I analyse methods and procedures that help to ensure the real-time behaviour of industrial control systems in non-deterministic Ethernet networks.

The majority of the methods presented in my thesis is supported and confirmed by concrete empirical measurements. Close to one hundred thousand measurements from many dozens of possible scenarios generated throughout my several years of work will hopefully provide help to numerous developers and operator engineers who work every day create better, more reliable and cheaper industrial communication tools and operate those effectively.

Also, the simulation model I developed for Profinet IRT and EtherCAT systems has the potential to help with the design of real-time industrial Ethernet systems. As a result, linear bus systems of any complexity, featuring many nodes in strictly real-time network can be tested under virtual conditions. With its help, it is possible to define all the critical parameters that must be considered during the design process. The procedure can be upgraded, with new industrial Ethernet systems that can be added and implemented at any time.

The following four dissertation thesis sum up the research results and experience achieved during my several years of work:

Thesis 1: *I have defined a cyclical real-time operating system base which can be declared to be real time if the worst-case cycle time (the maximum cycle time) is not more than half of the technological limit. Since the subsystem response time is clearly dependent on the cycle time, the above statement can be interpreted that the base system can be considered real-time if any excitation input response time will never exceed the time limit prescribed by the technology. Furthermore, I extended the core system by connecting more items via Ethernet IO channels. I also determined the real-time performance criteria of the subsystem according to which the Ethernet subsystem is to be stated real time if the bus update time of the device with the largest cycle time is at least four times less than the assigned time limit in the system specification.*

Thesis 2: *I developed two methods that help to measure the cycle time more accurately than measurements provided by the control system software.*
The indirect method based on cycle count can be applied to any PLC as it does not require any external equipment. It makes it possible to track changes in cycle time. The measured values can be used in the calculation of the reaction times, which define the criteria for real-time operation. The second method provides a direct measurement option for an external timer. Only such PLC can be used which also have an electronic

output channels. Primarily it can be used for research purposes in a similar area. It is much more sensitive than the previous one, and even a few microseconds changes can be measured. The measurement results can be stored in unlimited quantities. Using this method I was able to measure the cycle of failure. I determined the relation of the cycle for the error and the relationship between the cycle and run-time error in the control program.

***Thesis 3**: The basic determinant of the real-time operation of the non-synchronized industrial Ethernet systems is the reaction time. The exact theoretical definition of the reaction time is not possible because the excitation signals running into the input channels of the device drivers arrive at random with respect to the cyclically adjusted refresh time. It is, however, useful to determine the extreme values, especially the maximum possible values in order to record the time limit for real-time operation. I determined the limits of the non-synchronized real-time Ethernet systems based on their known reaction times and the configuration options of their parameters, which help to ensure the optimum operation of variable speed processes. Or, as is usually the case in practice, the time limit is adjusted to the given technological response times. I determined the theoretical minimum and maximum response time, the maximum jitter and the relative deviation. After processing thousands of measurements in different situations, I found that the output channel response time related to the non-synchronized industrial Ethernet system device drivers varies randomly between the minimal cycle time of the input delay extended controller and the four-fold value of the refresh time specified in the configuration.*

***Thesis 4:** I examined the definition of the optimal bus refresh cycle time in strictly real-time industrial Ethernet networks. In this contribution, I focused on the number of nodes, the amount of control data, as well as aspects of the bit rate. Using the rules set forth herein I developed a computer model using which these systems can be examined considering the various aspects mentioned above and in different situations. There are basically two theoretical solutions feasible. One is when the controller transmits control data to the device controllers in separately addressed frameworks. The other method is trying to exploit the maximum Ethernet frame load and to place more control telegrams within one frame.*

I have designed and developed a computer model that allows both completely separate strictly real-time technology solutions to be tested practically in any situation. The application makes it easy to decide what solution should be designed based on the given technological requirements.

I found that a solution applying embedded telegrams is the one providing the most optimal cycle time in the case of strictly real-time Industrial Ethernet systems, requiring low payload, and multiple nodes in a Fast Ethernet network. Also, I concluded that forwarding control data that is stored in separately addressed data frames in each node and presupposes changing frame loads that do not exceed 36 bytes, results in constant cycles.

IRODALOMJEGYZÉK

Saját publikációk

[S1] Ferenczi István: *Relationship between input channel excitation time and Profinet IO refresh time,* HUNGARIAN JOURNAL OF INDUSTRY AND CHEMISTRY 38:(2) pp. 95-98. ISBN 0133-0276, 2010.

[S2] Ferenczi István: *Measuring an Industrial Ethernet IO Device Reaction Time by LabView 8.2,* Scientific Bulletin Series C: Fascicole Mechanics, Tribology, Machine Manufactoring Technology, 2013, Vol. 27, pp. 101-104.

[S3] Ferenczi István, Vásárhelyi József: *A Methode to Measuring and Analyzing the Reaction Time of an Industrial Ethernet Network*, Journal of Elecrical Engineering, ISSN 1582-4594, Vol. 16/1, pp. 132-137, 2016.

[S4] Ferenczi István: *Profinet Real-Time kommunikációs stratégiák,* GÉP, Vol. 60/12. pp. 53-56., ISSN 0016-8572, 2009.

[S5] Ferenczi István: *Beágyazott telegramokat tartalmazó ipari Ethernet keretek ciklusideje*, GÉP, Vol. 63/5, pp. 51-54. ISSN 0016-8572, 2012.

[S6] Ferenczi István: *PROFINET - több mint ipari Ethernet II*., X. ENELKO - XIX. SzámOkt: Nemzetközi Energetika - Elektrotechnika és Számítástechnika Konferencia, Marosvásárhely, 2009. október 8-11, 2009, pp. 197-202. ISSN 1842-4546.

[S7] Ferenczi István: *Szinkronizált adatátviteli késleltetések elemzése a Profinet IRT rendszernél,* XI. ENELKO – XX SzámOkt Nemzetközi Energetikai-elektrotechnikai és Számítástechnikai Konferencia, Szatmárnémeti, Románia, 2010.október 7-10, pp. 118-123. ISSN 1842-4546

[S8] Ferenczi István: *Comparing a two approaches of Hard Real-Time Industrial Ethernet protocols,* Proceedings of the 1st Regional Conference – Mechatronics in Practice and Education (MECH-CONF 2011), December 8-10, 2011 Subotica, Serbia, ISBN 978-86-85409-67-7

[S9] Ferenczi István: *Korszerű valós idejű ipari Ethernet rendszerek*, II. Nyíregyházi Doktorandusz (PhD/DLA) Konferencia, Nyíregyháza, Magyarország, 2008.11.21 Bessenyei György Könyvkiadó, 2009. pp. 241-248. ISBN 978-963-9909-19-9.

[S10] Ferenczi István: *PROFINET - több mint ipari Ethernet*, microCAD 2009, XXIII. International Scientific Conference, Miskolc, Magyarország, 2009.03.19-2009.03.20. Automation and Telecommunication Section, pp. 21-26. ISBN 978-963-661-874-2

[S11] Ferenczi István: *Profinet IO ciklusidő változás vizsgálata,* III. Nyíregyházi Doktorandusz (PhD/DLA) Konferencia. 2009. november 20., Nyíregyháza, Magyarország, pp. 121-126. ISBN 978-963-87809-6-6.

[S12] Ferenczi István: *Válaszidők vizsgálata egy Profinet IO rendszernél*, Proceedings of Factory Automation 2010: 15-16th April, 2010, Kecskemét, Hungary. pp. 105-110. ISBN 978-963-7294-83-9.

[S13] Ferenczi István: *Profinet IO kontroller ciklusidő változásának vizsgálata,* XXV. MicroCAD Nemzetközi Konferencia kiadvány, 2010 március.

[S14] Ferenczi István: *Profinet IO válaszidők elemzése*, MTEAR 2010 Konferencia. Nyíregyháza, Magyarország, 2010.05.18-2010.05.19. pp. 169-174. ISBN 978 963 706 424 1.

[S15] Ferenczi István: Methods to determine a Profinet IRT bus cycle time, Proceedings of the XXV. microCAD International Scientific Conference, Miskolc, Magyarország, 2011.03.31-2011.04.01. Miskolci Egyetem, pp. 7-12. ISBN 978-963-661-691-9.

[S16] Ferenczi István: *PTC protokoll alkalmazása a Profinet IRT kapcsolatok szinkronizálására,* IV. Nyíregyházi Doktorandusz (Phd/DLA) Konferencia Nyíregyháza, Magyarország, 2010.12.07. ISBN 978-963-318-174-4.

[S17] Ferenczi István: *Programmable Logic Controllers as Real-Time Industrial Ethernet Base System Devices,* International Symposium on Applied Informatics and Related Areas: AIS 2013, Óbudai Egyetem Székesfehérvár, Magyarország, 2013.11.07-2013.11.09. pp. 46-49. ISBN 978-615-5018-88-6.

[S18] Ferenczi István: *A programozható logikai vezérlők ciklusidejének mérési módszerei,* Műszaki Tudomány az Észak-kelet Magyarországi Régióban 2013, Debrecen, Magyarország, 2013.06.04 pp. 299-307. ISBN 978-963-7064-30-2.

[S19] Ferenczi István, Vásárhelyi József: *Reaction time measurement of non-synchronized Industrial Ethernet IO device,* Proceedings of the 16th International Carpathian Control Conference, Szilvásvárad, Magyarország, 2015.05.27-2015.05.30. pp. 121-124. ISBN 978-1-4799-7369-9.

[S20] Ferenczi István: *Modeling the behavior of Profinet IRT in Gigabit Ethernet network,* TÁMOP-4.2.2/B-10/1-2010-0008 pályázat kiadványa, Miskolc, Magyarország, 2014.04.15 Miskolc-Egyetemváros, Miskolci Egyetem, pp. 41-46.

[S21] Ferenczi István: *A Profinet IRT ciklusidő meghatározásának módszerei,* Doktoranduszok fóruma: Gépészmérnöki és Informatikai Kar Szekciókiadványa. Miskolc, Magyarország, 2010.11.10.

[S22] Ferenczi István: *Nem szinkronizált, kritikus időkapcsolatok elemzése egy Profinet IO rendszernél,* Doktori szemináriumi munkaanyag, Miskolc-Egyetemváros, 2010. 06.

[S23] Ferenczi István: *Beágyazott telegramokat tartalmazó ipari Ethernet keretek ciklusidejének meghatározása*, Doktorandusz szemináriumi munkaanyag, Miskolc-Egyetemváros, 2011.06.

További felhasznált szakirodalom:

[1] IEEE 802.3 Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, 2008.

[2] Hosszú Gábor: Az internetes kommunikáció alapjai, ISBN 963-9442-51-8, 2005.

[3] Ajtonyi István: Ipari kommunikációs rendszerek I., Aut-Info Kft. Miskolc, 2008.

[4] Andrew S. Tanenbaum: Számítógép hálózatok, ISBN 978-963-545-529-4, 2011

[5] RFC 793, Transmission Control Protocol, 1981.

[6] RFC 768, User Datagram Protocol

[7] RFC 826, An Ethernet Address Resolution Protocol, 1982.

[8] RFC 1918, Address allocations for privat Internets, 1996.

[9] Perry S. Marshal, John Rinaldi: Industrial Ethernet, 2. edition, ISBN 1-55617-892-1, 2005.

[10] Petrényi József: TCP/IP alapok, 2009

[11] Hartványi Tamás, Kovácsházi Tamás: Számítógép hálózatok, 2001.

[12] Kovács Szilveszter: A hálózattervezés alapjai, 2007.

[13] RFC 894, Charles Hornig: A Standard for the Transmission of IP Datagrams over Ethernet Networks, 1984.

[14] Ajtonyi István: Ipari kommunikációs rendszerek III., Aut-Info Kft. Miskolc, 2009.

[15] Seok-Kyu Kweon, Kang G. Shin: Statical Real-Time communication over Ethernet for Manufactoring Automation Systems, 1999.

[16] Lucia LoBello, Giordano Kaczynski, Oratio Mirabella: Improving the Real-Time Behavior of Ethernet Networks Using Traffic Smoothing, 2005.

[17] J. L. Sobrinho, A. S. Krishnakumar: EQuB-Ethernet quality of service using black burst, 2006.

[18] Q. Zang, W. Zang: Priority Scheduling in Switched Industrial Ethernet, 2005.

[19] Chitra Venkatramani: The design Implementation and Evaluation of RETHER: Real-Time Ethernet Protocol, PhD dissertation, 1996.

[20] Chitra Venkatramani, Tzi-cker Chiueh: Design, Implementation, and Evaluation of a Software-based Real-Time Ethernet Protocol, 1995.

[21] I. L. Badiola, I. Radovanović, G. Russello, M. Chaudron: Design and implementation of a Real-Time protocol over Ethernet, 2004.

[22] H. Hoang, G. Buttazzo, M. Jonsson, S. Karlsson: Computing the Minimum EDF Feasible Deadline in Periodic Systems, RTCSA 2006.

[23] Jane W. S. Liu: Real-Time Systems, Prentice Hall, 2000

[24] Tei-Wei Kuo: Introduction to Real-time Process scheduling, National Taiwan University http://www.csie.ntu.edu.tw/~ktw/rts/ch-short-course-uni.pdf

[25] C. L. Liu, J. W. Layland: Scheduling algorithm for multiprograming in a Hard Real-time Environments, Journal of ACM, 1973.

[26] S. K. Baruah, A. K. Mok, L. E. Rosier: Preemtively scheduling Hard Real-Time sporadic task on one processor, IEEE Real-Time System Symposium, 1990.

[27] Kondorosi Károly, Szirmay-Kalos László, László Zoltán: *Objektum orientált szoftverfejlesztés,* 5. fejezet, http://www.tankonyvtar.hu/hu/tartalom/tkt/objektum-orientalt/ch05.html#id517610

[28] Ajtonyi I., Gyuricza I., Programozható irányítóberendezések és hálózatok, 2002.

[29] Ross Bannatyne: Time Triggered Protocol: TTP/C, Embended System Programming, march 1999.

[30] B. Müller, T. Führer, F. Hartwich, R. Hugel, H. Weiler, Fault tolerant TTCAN networks, http://www.canopen.org/fileadmin/cia/files/icc/8/mueller.pdf

[31] P. Pedreiras, L. Almeida, P. Gai, G. Buttazzo: FTT-Ethernet, a platform to implement the Elastic Task Model over message streams, http://www.ieeta.pt/lse/ftt/pub/wfcs02_ftt-eth_elas_task_model.pdf

[32] Ronald Dietrich: Industrial Ethernet, Harting Electric GmbH, 2004.

[33] Martin Rostan: Industrial Ethernet Technology, EtherCAT Technology Group, 2011. augusztus.

[34] Rockwell Automation Publication ENET-RM002A-EN-P, 2011. július.

[35] Dirk Mohl, IEEE 1588 - Precise Time Synchronization as the Basis for Real Time Applications in Automation, 2004.

[36] Mark Chaffee, CIP Motion Implementation Considerations, CIP Networks Conference, február 2009.

[37] Raimond Pigan, Mark Metter: Automating with Profinet (2006)

[38] PROFINET – Technology and Application, 2006. április.

[39] PROFINET System Description – Technology and Application, Version June 2011.

[40] EtherCAT – The Ethernet Fieldbus, 2009, www.ethercat.org

[41] EtherCAT Master – Application Developers Manual, Doc. Nr. P.4500.21/Rev. 1.4, 2011.

[42] EtherCAT Slave Controller Technology, Ver. 1.7 by Backhoff Automation GmbH, 2010.

[43] Ethernet Powerlink, Perfection in Automation, 2004. www.br-automation.com

[44] Ethernet Powerlink Basics, 2011, www.ethernet-powerlink.org

[45] SERCOS III, by SERCOS International e.V. 2011. www.sercos.org

[46] Peter Lutz, SERCOS III – Universal Real-Time Communication for Automation, MOF Summit, Tokyo, 2011. november.

[47] Introduction to Modbus TCP by Acromag Inc. Wixom, USA, 2005. www.acromag.com/sites/default/files/Acromag_Intro_ModbusTCP_765A.pdf

[48] Siemens AG, SIEMENS SIMATIC S7-300 CPU 31xC and CPU 31x, Technical Specification manual 2011. március.

[49] Siemens AG, S7-300 Instruction list manual, Edition 1, 2001.

[50] National Instruments, NI 622x Specification manual, 2007. június.

[51] Schneider Magazin, VIII. évfolyam 1. szám, 2008. február.

[52] Siemens AG, ET200S Distributed IO Interface modul IM151-3PN high features manual, 2009. március.

[53] Ajtonyi István, PLC és SCADA-HMI rendszerek II. & Ipari kommunikációs rendszerek II. Aut-Info Kft. ISSN 1789-5456, ISBN 978-963-661-833-9, pp. 408, 2008. május.

[54] International Standard IEC 61131-5, Programmable Controllers Part 5, Communication, Edition 2000.11.

[55] Siemens AG, Simatic ET200 Distributed IO Digital electronic modul 2DI 24VHF manual, 2007. 04.

[56] Siemens AG, Programmable Logic Controllers S7-300 Module Data Reference Manual, 2005.

[57] Dirc S. Mohl, IEEE 1588 - Precise Time Synchronization as the Basis for Real Time Applications in Automation, 2009.

[58] Jaspernite J., Schumacher M., Weber K.: Limits of increasing the performance of Industrial Ethernet Protocols, ETFA 2007 Proceedings, p. 17-24.

[59] Gunnar Prytz, A performance analysis of EtherCAT and Profinet IRT, 13. IEEE International Conference EFTA 2008, pp. 408-415.

[60] Caleb Gordon, White Paper Introduction to IEEE 1588 & Transparent Clocks, www.tekroninternational.com, 2009.

[61] M. Rostan, EteherCAT Introduction, Hannover messe 2008.

[62] Schneider magazin, VIII. évfolyam, 1. szám, 2008. február.

[63] Mitsubishi Electric FA, CC Link IE, 07.2008.

[64] Max Felser, Thilo Sauter: Standardization of Industrial Ethernet – the next battlefield? 2004. IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.

[65] J. D. Decotignie, Ethernet-Based Real-Time and Industrial Communication, Proceedings of the IEEE Volume:93, Issue: 6, 2005, pp. 1107-1117.