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

Mérésadatgyűjtő és vezérlő rendszer fejlesztése EtherCAT hálózattal

Szakdolgozat

Készítette
Kőrösi Dániel

Konzulens
dr. Orosz György
Csengeri Bálint

2019. december 8.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

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

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

Budapest, 2019. december 8.

Kőrösi Dániel
hallgató

Kivonat

A dolgozat témája egy EtherCAT kommunikáción alapuló rendszer fejlesztése volt. Az EtherCAT protokoll által egy új, modern kommunikációs típust ismertem meg, az Ethernet alapú field-busz kommunikációt. Az ilyen hálózatok robosztusak, gyorsan összeszerelhetőek, az eszközök összeköttetéséhez pedig csak egy Ethernet kábel szükséges. Manapság az iparban az ilyen tulajdonságú hálózatok egyre keresettebbek.

A szakdolgozatomat a ProDSP Technologies Zrt.-nél készítettem. A cégnek több olyan terméke van, illetve több olyan terméket fejleszt, ahol egymástól különálló műszereket vagy egyszerű vezérlőegységeket kell összekötni úgy, hogy közöttük valósidejű adatcsere történjen. Tipikus feladatok közé tartozik szinkronizált mérések elvégzése, a mérési eredmények továbbítása, illetve vezérlési feladatok ellátása.

A feladatom célja egy olyan EtherCAT alapú mérő és adatgyűjtő rendszer kifejlesztése volt, amely nem csak mintavételezni tud, de az EtherCAT kommunikációs protokoll segítségével a Slave-ek működését vezérelni is. A gyűjtött adatokat a rendszer képes volt fájlba menteni és akár ezzel egy időben megjeleníteni, mindezt úgy, hogy közben a kommunikáció valósidejű maradt.

A rendszer létrehozásához több olyan egységet is használtam, amelyet speciálisan az EtherCAT kommunikációra készítettek. A hálózatom irányítása egy PC-n valósult meg, erre specializált szoftverrel.

Ezen prototípusból való kiindulás csökkenti egy specifikusabb, bonyolultabb EtherCAT rendszer kidolgozásához szükséges fejlesztési időt, a fejlesztéshez megfelelő alapot ad.

Abstract

The subject of the thesis was the development of an EtherCAT communication based system. With EtherCAT protocol, I was able to learn about a new, modern communication type, the Ethernet-based fieldbus communication. Networks based on this communation protocol group are robust, easy to set up and connect to all the devices of the network, only an Ethernet cable is required. Nowadays networks with these features are even more in demand.

I've completed my thesis at ProDSP Technologies Zrt. The company develops many products, in which there's a need for realtime data transfer between separate devices. These products have to perform typical tasks like syncronised measuring, transmissing the measured data and manage control tasks.

The goal of my assignment was to develop and build such an EtherCAT based measurement and datacollector system, which is not only able to measure, but with EtherCAT communication protocol, it can control the Slaves of the network. The system was able to store the collected data in a file, and in the same time display it, while retaining it's realtime capabilities.

To create this system, I us many devices made specifically for EtherCAT communication. The network management was developed and done on PC with specialized software.

From this prototype other, more specific and more complicated EtherCAT systems can be created faster, with less development time. It provides an appropriate basis for future development.

1. fejezet

Bevezetés

1.1. A szakdolgozat háttere

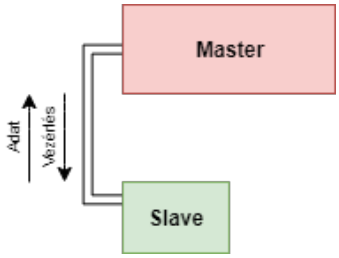
Már az 1990-es években megjelent a szükség valósidejű hálózatokra az iparban. Ezen igényeket elégítette ki a CAN, LIN, és FlexRay. A többféle fieldbusz rendszer több alapvető követelményt támasztott, főleg a fizikai megvalósítás szintjén. Minden buszrendszernek vezetékezését meg kellett tervezni, speciális vezetékekkel, eszközökkel jöhetett létre a kommunikáció. Ezen hálózatok egymásba integrálása is idő- és költségigényes.

A ProDSP Technologies Zrt.-nél is felmerült az a gondolat, hogy egységesíteni kellene a nem National Instruments által gyártott, különálló eszközök kommunikációját és egy közös kommunikációs hálózatra rákapcsolni őket, mindezt úgy, hogy a National Instruments által gyártott eszközök natív kommunikációs megoldásai is megmaradjanak. Fontos volt, hogy a cég által rendelésre tervezett eszközök az időzített adatgyűjtési és beavatkozói elvárásoknak is megfeleljenek. A cég egyes ilyen termékei megkövetelik a valósidejű eszköz-eszköz kommunikációt, így adott volt, hogy olyan protokollt érdemes alkalmazni, amely az előbb említett tulajdonságai mellett erre is képes. Erre jelenthet megoldást az EtherCAT.

Mivel a cég ezt a protokollt nem ismerte, így az én feladatom volt az EtherCAT tulajdonságainak, előnyeinek megismerése. Az így megismert hasznos attribútumok segítségével egy prototípus rendszert alkottam, mely bemutatja az EtherCAT kommunikációs protokoll működésének alapjait. A jövőbeni fejlesztések megalapozásaként a rendszer megvalósít egy mérésadatgyűjtő rendszert, amely ezen felül képes egyszerűbben vezérelni Slave egységeket.

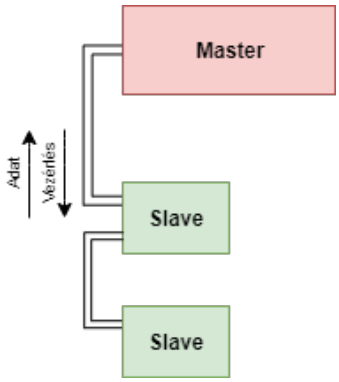
1.2. Célok, követelmények

A cél egy megbízhatóan működő determinisztikus adatgyűjtő és vezérlő rendszer megépítése volt. A prototípussal szemben támasztott követelmény annyiból állt, hogy 1 ms-os gyakorisággal szinkron mintavételezést tudjon megvalósítani, az adatokat lementeni és ezeket az információkat úgy továbbítani és fogadni, hogy a kommunikáció mindvégig valósidejű maradhasson, ne történjen adatvesztés.



1.1. ábra. Egy Slave-es rendszer blokkvázlata

A 1.1-es ábrán látható az alaprendszer felépítése, bár a prototípusban egyszerű a kapcsolat, de összeköttetést kialakítani egy bonyolultabb topológiában nem igényel több eszközt mint a mostani. A 1.2-es ábra a két Slave-es elrendezést mutatja be.



1.2. ábra. Két Slave-es rendszer blokkvázlata

A munkám elején a hagyományos terepi buszokat, majd az Ethernet alapúakat hasonlítottam össze, és végül összevetettem az összes addig vizsgált terepi busz tulajdonságait. Ezek eredményeit figyelembe véve, illetve a cég által támasztott követelmények hatására kiválasztottam az EtherCAT protokollt, amelyet ezután behatóbban tanulmányoztam. Megismertem a protokoll felépítését a fizikai rétegtől kezdve az alkalmazási rétegig. Megvizsgáltam a működését és a fejleszthetőségének további lehetőségeit(EtherCAT G és EtherCAT P). Ezek után a rendszer alkotóegységeit választottam ki, és ezek tulajdonságait vettem szemügyre. Ezt követően a fizikailag létrejött rendszer működéséhez szükséges software-eket és firmware-eket fejlesztettem. A kész rendszer működését leellenőriztem, attribútumait megvizsgáltam és összevetettem a követelményekkel.

2. fejezet

Terepi buszok összehasonlítása

A field-busz rendszerek

Terepi buszok, vagy field-buszok olyan kommunikációs hálózatok, melyeket általában valósidejű elosztott vezérlésre használnak, és melyek általában csak fizikai és adatkapcsolati réteget valósítanak meg. Manapság többféle buszrendszer is alkalmazható különböző ipari megoldásokra. Alapvetően két típus szerint vizsgáltam ezeket a rendszereket, ezen típusok a Nem Ethernet alapú buszok, illetve az Ethernet alapú buszok. Ebben a fejezetben ezen típusok tulajdonságait határozom meg és hasonlítom össze. A következőekben az alábbi field-busz rendszereket mutatom be:

- Nem Ethernet alapú:
  - CAN: 2.1.1[7]
  - LIN: 2.1.2[7]
  - FlexRay: 2.1.3[24]
- Ethernet alapú:
  - SERCOS III: 2.2.1[19]
  - PROFINET IRT: 2.2.2[18]
  - EtherCAT: 2.2.3[8]

2.1. Nem Ehernet alapú rendszerek

2.1.1. CAN:Controller Area Network [7]

Protokoll jellemzők:

Többmester felépítés

A CAN buszrendszer fizikai rétegének működése miatt nincsen szükség különálló Master csomópontra, minden egység egyenrangú, üzeneteit önállóan tudja továbbítani. Egyedül a csomópont azonosítója az amely meghatározza az egység prioritását, ez a buszért való versengésben játszik szerepet.

Broadcast

A buszon való kommunikáció egy telefonkonferenciához hasonló, több fél van, akik egyszerre hallgatják a másikat, de egyszerre csak egy beszél, mivel csak így érthető a kommunikáció. Emiatt minden csomópont üzenete eljut mindegyik más egységnek a hálózaton belül. Az adatmennyiség üzenetenként maximálisan csak 8 bájt lehet.

Rendszer flexibilitása

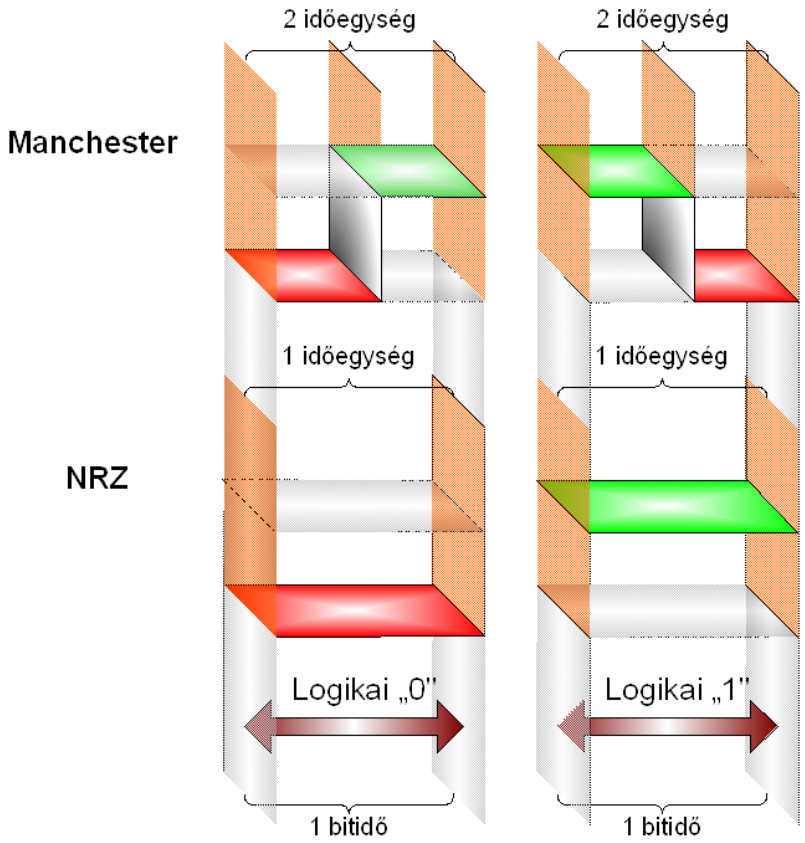
A rendszer felépítése miatt könnyen lehet egységeket hozzáadni, vagy kivenni a többi egység megzavarása nélkül. Összesen csak 32db [7] csomópont lehetséges egy szabványos buszmeghajtó esetén.

Fizikai felépítése:

CAN busz

A CAN buszok fő elrendezése a huzalozott-ÉS konfiguráció, ebben a csomópontok vagy logikai egyest, vagy logikai nullát továbbíthatnak. Az adattovábbítás módjának azért huzalozott-ÉS a neve, mivel ahhoz, hogy logikai egyes jelenjen meg a buszon, minden csomópontnak logikai egyest kell küldenie, de ha bármelyik csomópont logikai nullát továbbít, akkor a busz állapota logikai nulla lesz. Ezért nevezzük a logikai nullás bitet dominánsnak és az egyeset recesszívnek.

A CAN nem nullára visszatérő kódolást, vagyis NRZ-t(Non-Return-to-Zero) használ, amellyel egy kiküldött bit alatt nem szükséges jelszintváltás. Ezzel ellentétes a Manchester kódolás, melynél egy bit reprezentációjához szükséges a jelszintváltás (2.1. ábra). Mivel a CAN a hatékonyságra törekszik, és a Manchaster kódolás két bitidő alatt tud csak egyet értelmezni, így indokolt az NRZ használata. Azonban a Manchaster kódolás jelszintváltása egyben a szinkronizáció szerepét is betölti, így a szinkronizációra az NRZ kódolásban is szükséges egy megoldás. Ehhez a bitstuffing eljárást kerül felhasználásra, amelyben ha ötnél több azonos jelszintű érték továbbítódna, az ötödik után a küldő beszúr egy ellentétes értéket, a fogadó fél pedig fogadásnál kiszelektálja ezeket.



2.1. ábra. NRZ és Manchester kódolás közötti eltérés.[7]

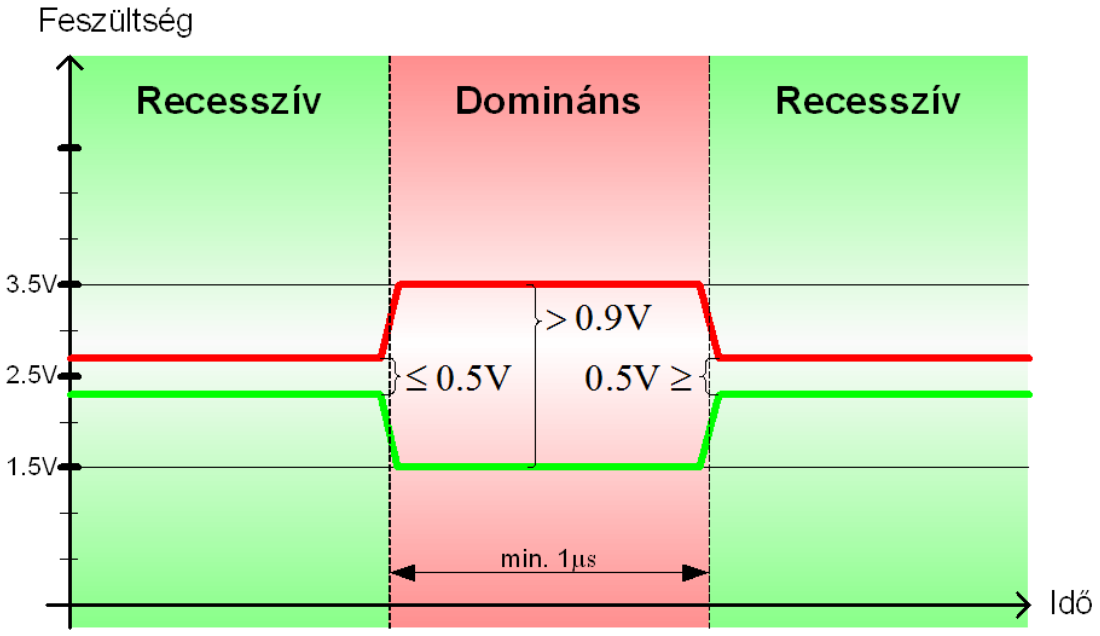
A fizikai réteg felépítésére leggyakrabban az ISO 11898-2 [14] és az ISO 11898-3 [15] csavart érpárt alkalmaznak, szimmetrikus adóval és vevővel. Előbbi a nagy sebességű, maximálisan 1 Mbit/s-os átvitelre képes, utóbbi a kis sebességű, max. 125 kbit/s-ra. A maximálisan használható sebesség függ a busz hosszától, mivel a kibocsájtott jelnek oda vissza be kell futnia a teljes kábelt. Ilyen tájékoztató értékek a 2.1. táblázat-ban láthatóak.

Hossz	Sebesség
40 m	1 Mbit/s
100 m	500 kbit/s
200 m	250 kbit/s
500 m	125 kbit/s
1 km	50 kbit/s

2.1. táblázat. A távolságokhoz tartozó sebesség [7]

Bitszint meghatározása

Az kommunikáció adása és vétele ugyan azon az érpáron történik, és mind két ér aktívan részt vesz mindkét logikai jel létrehozásában, ezek a vezetékek a `CAN_H(High)` és a `CAN_L(Low)`.Ha a két vezeték feszültség szintjei közötti különbség nagyobb mint 0.9 V akkor a bitszint domináns, ha kisebb, akkor recesszív (2.2. ábra). A különbségből való meghatározás miatt ez a felépítés az elektromágneses interferenciák ellen védi a hálózatot.



2.2. ábra. A bitszintek létrehozása.[7]

Arbitráció

Egy állomás akkor kezdhet adni, ha a keret végén 11 recesszív bit érkezett, vagyis az üzenet végét nem szakította meg domináns bit. Ha több csomópont kezd el egyszerre adni, akkor az első lefutó élnél összeszinkronizálódnak és ezután a rendszer huzalozott-ÉS felépítését használják ki az üzenetküldési prioritás meghatározásában. Adás közben folyamatosan veszik a jelet és ezeket összehasonlítják a küldött jelükkel, ha nem egyezik

akkor abbahagyják az adást. Mivel a domináns biteknél elég, ha csak egy adó hajtja meg a hálózatot, így azok a csomópontok nyerik meg az arbitrációs versenyt, amelyeknek az azonosítója a legkisebb(legtöbb domináns bit), ezzel prioritást létrehozva a csomópontok között. Ennek az arbitrációs eljárásnak előnye, hogy nem destruktív, több csomópont adásánál nem kell megszakítani mindegyiknek az adást és a legnagyobb prioritásúnak újrakezdeni, mivel az egyik csomópont mindenképpen képes leadni az üzenetét.

CAN protokoll felépítése:

CAN keret

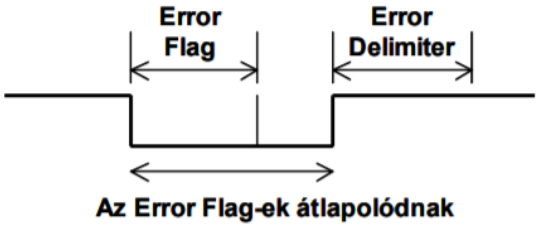
Az adatot melyet különböző alkalmazásokban felhasználunk adatkeretekkel továbbítja a CAN [7]. Ezek felépítése a látható a 2.2. táblázatban.

Név	Hossz	Érték	Leírás
Start of Frame	1 bit	0	A keret kezdetét jelzi.
ID Field	11 bit		Többnyire az adatmező tartalmát azonosítja.
RTR	1 bit	0	Távoli adáskérésnél RTR=1, egyébként RTR=0.
Control Field	6 bit	00xxxx	Az adatmező hosszát adja meg byte-okban.
Data Field	0-64 bit		Maximum 8 byte hosszúságú adatmező.
CRC Field	15 bit		Ellenőrző összeg.
CRC Delimiter	1 bit	1	
ACK Slot	1 bit	1/0	A keretet hibátlanul vevők ezt a bitet nullába állítják.
ACK Delimiter	1 bit	1	
End of Frame	7 bit	1111111	A keret végét jelzi.
Intermision	3 bit	111	Keretek közötti szünet.

2.2. táblázat. A CAN keret felépítése

Hibakeret

Ha bármely adó vagy vevő egység kommunikáció közben hibát észlel, ez a keret kerül kiadásra. Error Active módban az első hat bit logikai nulla míg Error Passive módban egyes. Ezt követi az Error Delimiter, ami nyolc darab recesszív értékből áll. Mivel sérti a bitbeszúrás szabályait, így azok a csomópontok is hibát fognak észlelni emiatt, amelyek eddig nem észlelték. Így ez megnövelheti a Frame hosszát maximálisan 6 bittel, mint ahogy azt a 2.3. ábra mutatja.



2.3. ábra. Az Error Frame felépítése

2.1.2. LIN:Local Interconnect Network [7]

Protokoll tulajdonságai:

Egymester felépítés

A LIN rendszer működése egy master alapú, ez kezdeményezhet csak adatátvitelt, így ez az ütemező eszköz, emiatt nincsenek arbitrációs problémák.

Broadcast

Bár az adatátvitelt csak master kezdeményezhet, de a kiküldött adatot minden egység képes fogadni és feldolgozni. Az ajánlott maximális csomópont szám egy hálózatban 16 [7].

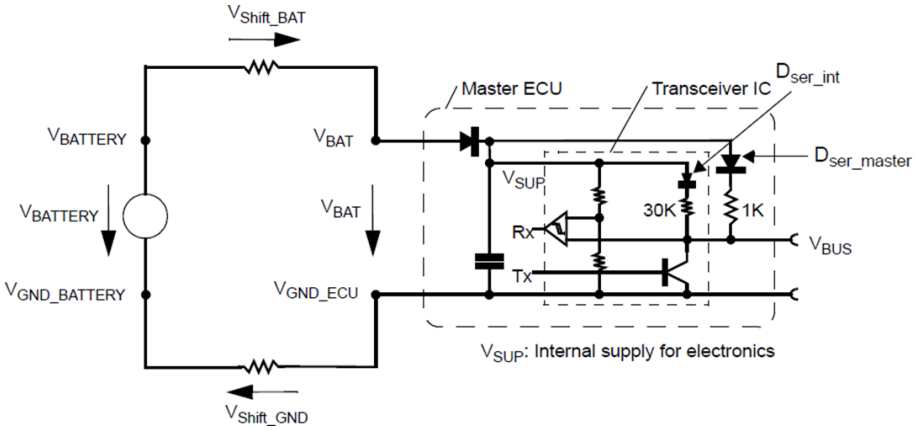
Egyszerűség

A LIN egy kifejezetten egyszerű field-busz rendszer, hiszen a legtöbb mikrokontrollerben megtalálható UART perifériával meg lehet oldani a kommunikációját.

Fizikai felépítése:

Adatküldő és fogadó egység

A küldés és fogadás is az ISO 9141 szabványhoz kötötten zajlik. Az egységek egy diódán és egy ellenálláson keresztül kapcsolódnak a tápellátás pozitív lábára. A tápellátást általában egy autóakkumulátor szolgáltatja.



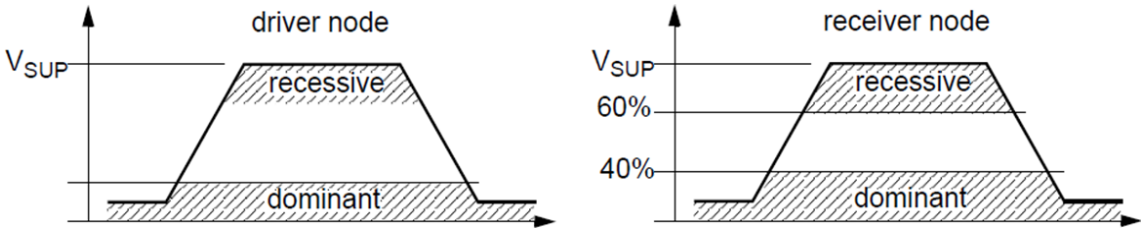
2.4. ábra. Egy LIN csomópont felépítése.[7]

Bitszint meghatározás

LIN rendszereknél a fizikai felépítés egyszerűbb, csupán egy vezeték van, melyet megfelelő szintekkel kell meghajtani. Bár ez elméletben könnyű, gyakorlatban a tápfeszültség esése, vagy a földponteltolódás miatt itt is hibák léphetnek fel. A jel szintje itt is domináns (logikai és fizikai 0) illetve recesszív (logikai és fizikai 1) lehet. Ezeknek elfogadott értéke eltér küldő és fogadó csomópontoknál. A 2.5. ábra mutatja a specifikációkban meghatározott szükséges értékeket.

Időzítési követelmények

A LIN busz esetén a bitsebesség értéke 1-20 kbit/s is lehet [7]. Ez a sebesség sok tényezőtől függ, ilyen például a hőmérséklet vagy feszültségingadozás. A megfelelő kommunikációhoz a csomópontoknak ezeket a változásokat érzékelnie kell és azokra megfelelően

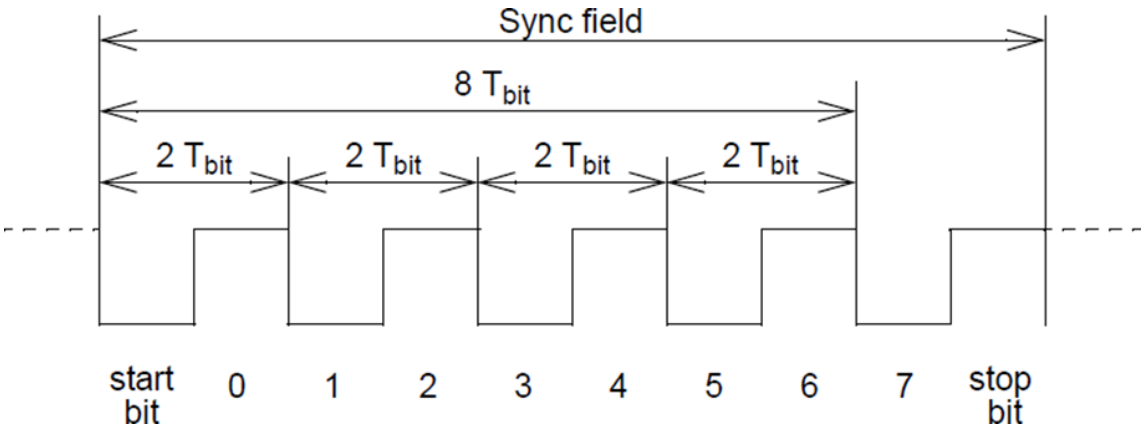


2.5. ábra. A szükséges feszültségszintek küldő- és fogadófeleknél.[7]

reagálni. A különböző hőmérsékleten lévő egységekben az oszcillátorok eltérően működhetnek, a kapacitív és rezisztív terhelés is megváltoztathatja a felfutási időket egységenként, ezzel a sebességet is. Ezeket a változásokat, eltéréseket kell kiegyensúlyozni vagy mérni és belekalkulálni a kommunikációba. A bitsebesség tolerancia leírja a busz referencia sebességtől való eltérést. A kommunikáló egységeknek figyelembe kell venniük ezt az értéket a pontos bitsebesség beállításához. Ennek értékét több tényező is változtatja:

- Órajelforrás stabilitása a slave egységnél.
- Órajelforrás stabilitása a master egységnél.
- Bitidő mérésének hibája a slave egységnél.

Ezen kívül minden üzenet elején lévő szinkronizációs mező segítségével is megtörténik és pontosodik a szinkronizáció.



2.6. ábra. A szinkronizációs mező.[7]

LIN protokoll felépítése:

A LIN kommunikációs hálózatoknál a szorosan összefonódó adatkapcsolati és hálózati réteg összességét hívjuk LIN protokollnak, mely leírja az üzenetek felépítését és ezeknek a típusát.

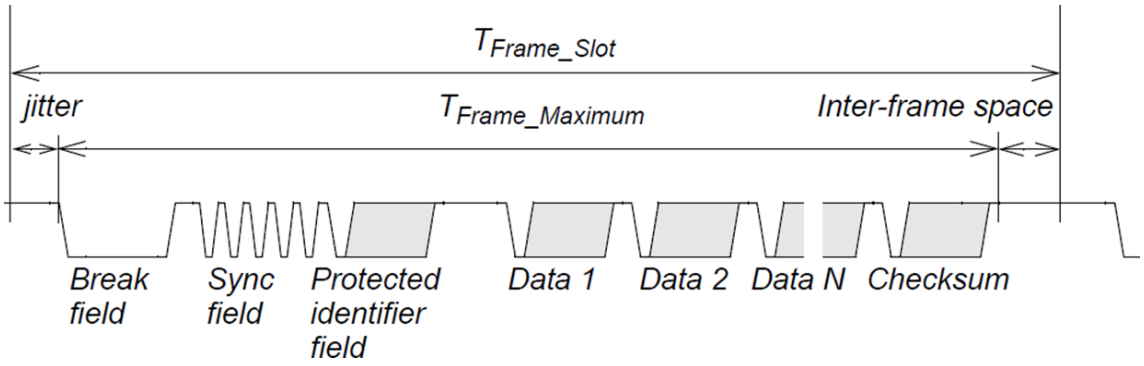
Taszkok

A rendszerben csak egy master van, ez az egység végzi a master taszkot és ha van csatlakoztatva rá periféria, akkor slave taszkot is tud végezni. A maradék slave nodeok pedig csak slave taszkot hajtanak végre. Az egész kommunikációt a master ütemezi a master taszk segítségével. A masterben található egy ütemező táblázat, melyben nyilvántartja,

hogy melyik slave-el kell kommunikálnia. Emiatt a rendszer működése determinisztikus lesz, nincs szükség arbitrációra.

LIN keret

A kommunikáció itt is üzenetekkel történik, melyek egy speciális keretben foglalnak helyet. A LIN üzenet eleje egy break és egy szinkronizációs mezőből áll. Ezzel kérdezi le a master a slave-et melyre a slave egy választ küld, ami tartalmazza az adatmezőt és az ellenőrző összeget.



2.7. ábra. A LIN keret. [7]

Adatcsomagok

Két típusú, jól elkülöníthető adatcsomagot használnak a LIN kommunikációban.

- Signal: Fizikai paramétereket, állapotokat jelöl, egy keretben mindig azonos adatbájtban található.
- Diagnosztikai üzenet: Ezek tartalmaznak diagnosztikai és konfigurációs információkat.

Hibajelzés

Nincs külön aktív hibajelzésre alkalmas megoldás mint a CAN-ben, viszont diagnosztikai üzenetben jelezhető a hiba.

2.1.3. FlexRay [24]

Protokoll attribútumai:

Megbízhatóság

A mai autófejlesztésekben a hagyományos mechanikai vezérlés helyett egy alternatívát jelenthet a vezeték általi vezérlés. Ez lehet akár a kormányzás vagy fékezés is, ezen folyamatoknak hibamentesnek kell lenniük, mivel életek múlnak rajtuk. Legfőképp ezen megoldásokra használják manapság az autógyártók a FlexRay field-buszokat.

Topológia

A FlexRay protokoll egy nagy előnye, hogy sok különböző topológiájú rendszerre felhasználható. A rendszer felépítése lehet csupán pont-pont kapcsolat, aktív, illetve passzív csillag, busz topológia, vagy akár ezek keveredéséből hibrid topológia is előállítható.

Fizikai felépítése:

Vezetékezés

FlexRay egy vagy két kommunikációs csatornán operálhat. Ezeknek a felépítése csavart érpárokból áll, ezzel a külső zavarok hibahatásai csökkenthetőek. Hosszuk topológiánként változhat, de általában 24 méter a maximális távolság két csomópont között. Átviteli sebessége elérheti akár a 20 Mbit/s-ot is [24].

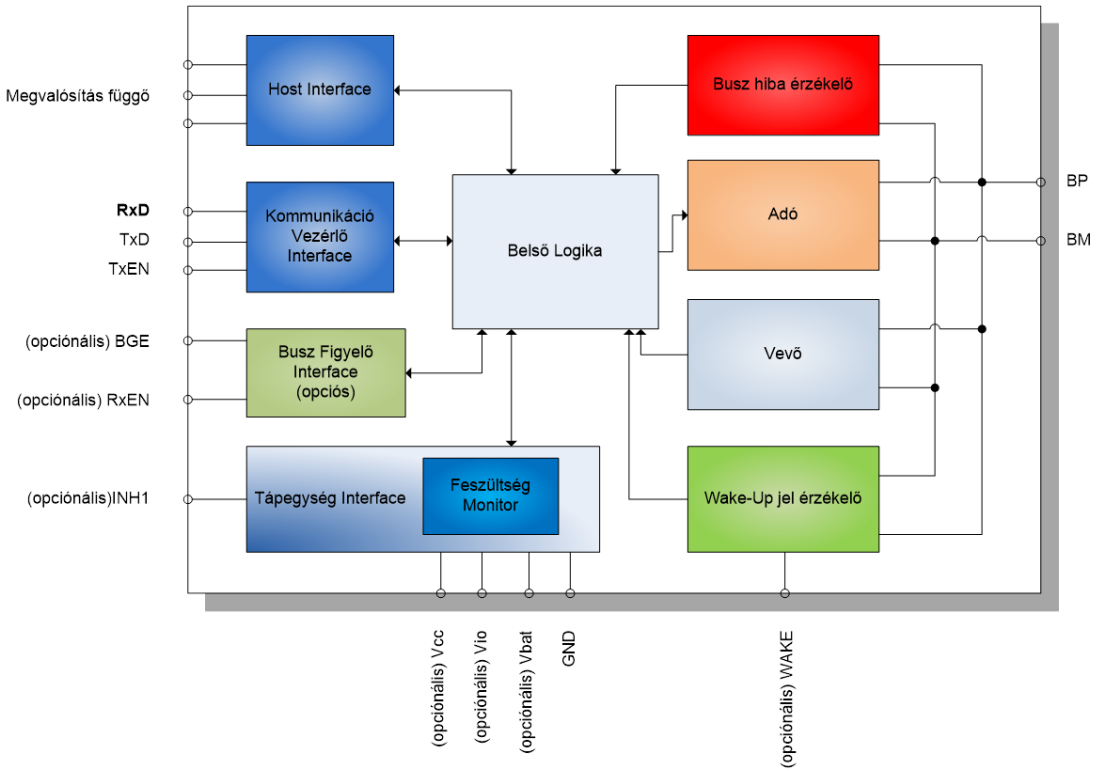
Busz Driver (BD)

A busz driver köti össze a FlexRay csomópontot a csatornával, ez alakítja át a fizikai jeleket logikaivá illetve a logikaiakat fizikaivá, ezt a kommunikációs vezérlő interfész segítségével történik. Ezen kívül feladata a tápfeszültség figyelése, és az eszköz és a busz közötti elektromos kisülések elleni védelem is. Az alacsony energiafelhasználású csomópontműködés is ezen az egységen keresztül realizálható. Felépítése látható a 2.8-as ábrán.

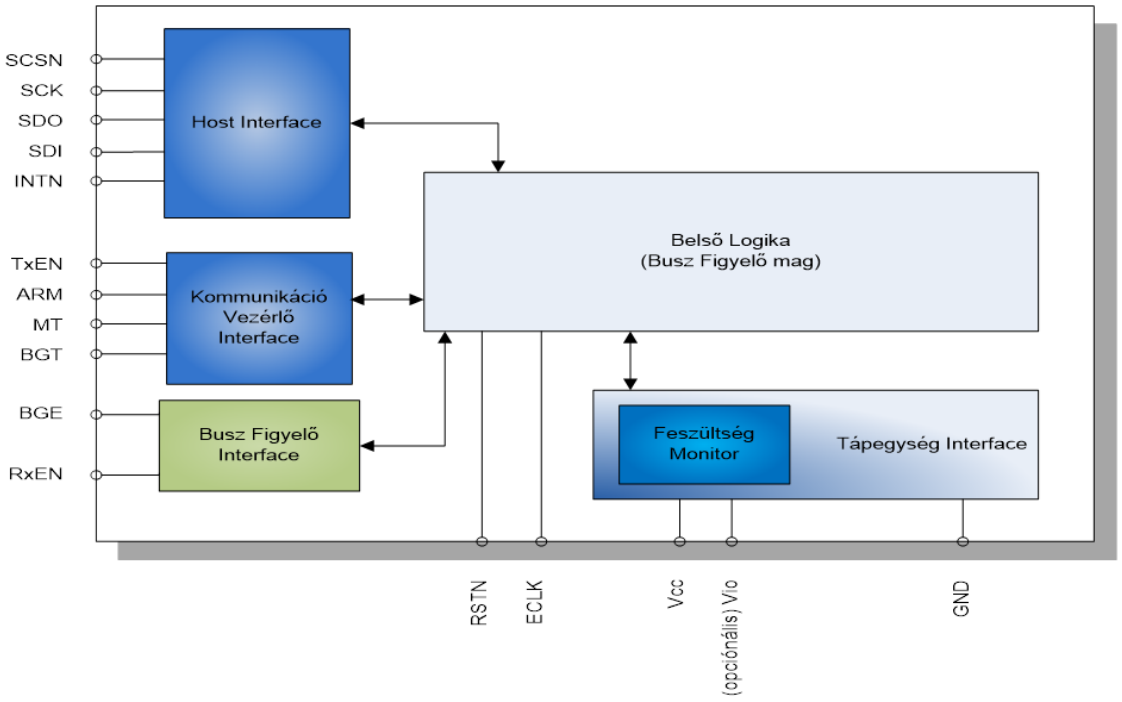
Egy állapotgép definiálja a működési módjait, melyek lehetnek: Normal, Standby, Sleep vagy Receive Only. A host segítségével vezérelhetjük az állapotokat a Host Interfaceen keresztül. Az applikációt megvalósító hardverrel több típusú összeköttetés is lehetséges, lehet közvetlenül huzalozott vagy akár SPI is.

Busz Guardian (BG)

A FlexRay segítségével megvalósított kommunikáció pontossága nagyon fontos, ebben segít a Busz Figyelő. Ez az eszköz osztja be a buszon található egységek üzeneteinek helyét és idejét a kommunikációs ciklusban úgy, hogy a busz ciklikus kommunikációja determinisztikus legyen. Meggátolja a hogy hibás egységek rossz időpontban való üzenetei megzavarják a rendszert. A 2.9. ábra mutatja a felépítését. Az figyelő állapotai: Standby és Normal. mivel időkritikus a rendszer, így a BG-ben is található saját lokális óra, amely segítségével időzíti az üzeneteket. A kommunikációs vezérlő interfész szinkronizálja ezt az időt a BG és a CC között.



2.8. ábra. A busz driver blokkdiagramja [7]



2.9. ábra. A busz figyelő blokkdiagramja [7]

Kommunikációs vezérlő (CC)

Sok implementáció fellelhető kommunikációs vezérlőre, ez az egység az, ami összeköti az applikációt a FlexRay hálózattal. A kommunikáció állapotától függő állapotai vannak

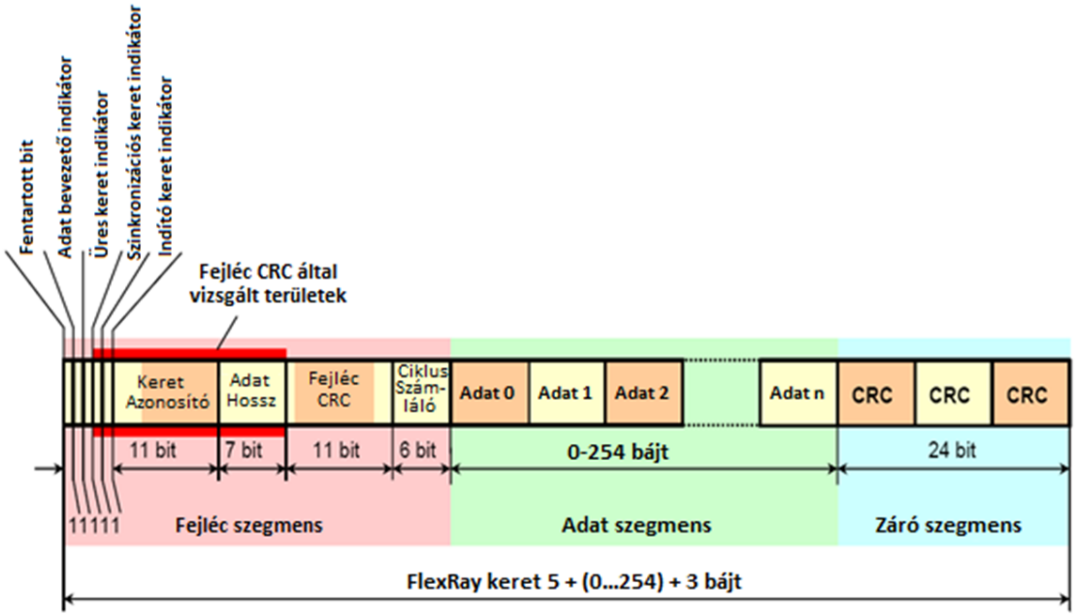
ennek az egységnek (összesen 8 darab), ezek váltásáról a Protocol Operation Control(POC) gondoskodik.

Kommunikációs szegmensek

A FlexRay kommunikáció sorban történik, meg van adva minden csomópont sorrendje. Ezt felhasználva kétféle kommunikációs szegmens van: statikus és dinamikus. Statikus szegmensben a kommunikáció ideje állandó, és minden keret hossza megegyezik. Ezáltal, ha az egyik csomópont nem akar információcserében részt venni, akkor is meg kell várnia a következő csomópontnak az előbbire kiszabott kommunikációs időt. A dinamikus szegmensben ezzel szemben, ha az egyik csomópont nem akar küldeni, akkor nem kell megvárni amíg a hozzátartozó kommunikációs idő letelik, hanem elindulhat a következő csomópont üzenetküldése.

FlexRay keret

Három részből áll: fejléc, adat és hibaellenőrzés. Ha egy csomópont üzenetet küld a hálózatnak, akkor először a fejléc, majd az adat végül a zárószegmens fog megjelenni.



2.10. ábra. A FlexRay keret [7]

Időszinkonizálás

Az összes FlexRay csomópont rendelkezik lokális órával, melyek eltérése akár kevesebb mint 125 μs [1] is lehet. Ez a szinkronizáció időkritikus műveleteknél (pl.:fékezés) nagyon lényeges. A FlexRay protokoll az órák szinkronizálását többféleképp éri el. Az összes statikus üzenet küldésének és fogadásának időpontja ismert az összes csomópont által. Ezek segítségével a node-ok tudják konfigurálni a kezdeti eltéréseiket és az üzenetküldés ciklusidejét[24].

2.2. Ethernet alapú rendszerek

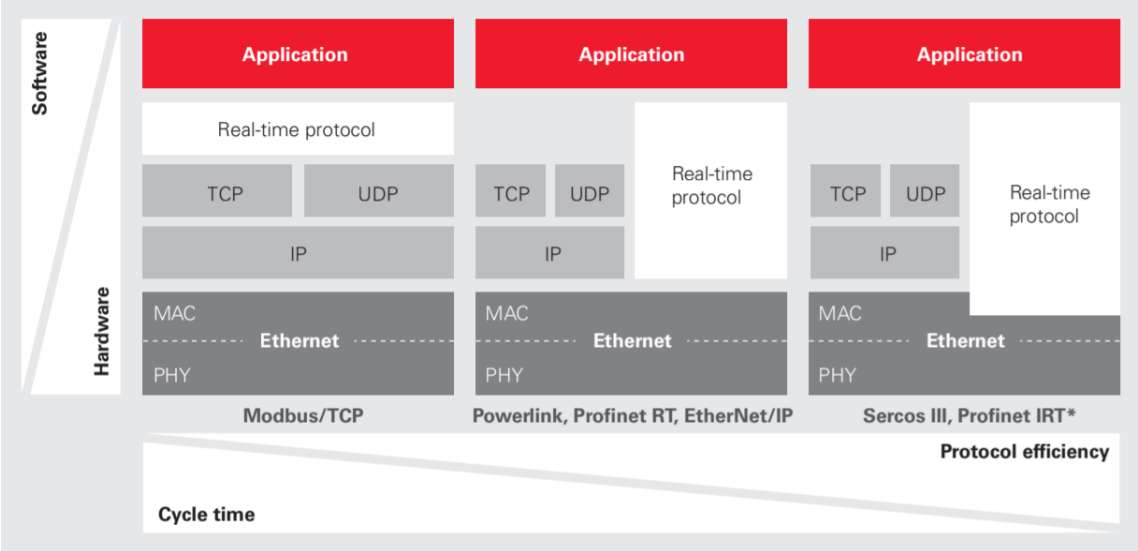
Fizikai réteg, vezetékezés

Minden Ethernet alapú protokollnak nagy előnye a könnyű kapcsolat kialakítása az egyszerű, mégis nagy adatátviteli sebességre képes Ethernet kábel segítségével. Ezzel a kábellel a nem Ethernet alapú buszrendszerekhez képest, olcsóbban és egyszerűbben lehet összekötni a rendszer minden elemét. További előnye, hogy nem csak a valósidejű hálózat létrehozására alkalmas, hanem ezzel a mindennapi alkalmazásokban használt, nem valósidejű eszközöket is össze lehet kötni, így egy hálózatban valós és nem valósidejű kommunikáció is történhet. Emiatt a hálózat bármely részéről elérhetőek a valósidejű elemek, adatok.

Mivel Ethernet protokollra épülnek, így annak a fizikai rétegét is használják ki, mely képes akár 100 Mbit/s-os full duplex adatátvitelre[16].

Valósidejű Ethernet

Legfőbb probléma az Ethernettel, hogy nem valósidejű, mivel az információütközések jelezve vannak de nincsenek elkerülve. Emiatt kiszámíthatatlan időbeli eltérések lehetnek az információáramlásban. Erre megoldásként több különböző szintű módszer áll rendelkezésre, mellyel lehetőség van elérni a valósidejűséget. A legalapvetőbb szinten a TCP/IP réteg fölé van helyezve a valósidejű protokoll. Ennél nagyobb ciklusidő elérését biztosítja az a megoldás ahol a valósidejű protokoll a hálózati és a szállítási réteg mellett helyezkedik el. Ennél is gyorsabb, ha már az adatkapcsolati rétegben is megjelenik az adott protokoll, ezen megoldásoknak már saját Ethernet frame-jük van, így csak a fizikai rétegen keresztül kompatibilisek az Ethernet protokollal (2.11. ábra).



2.11. ábra. Az Ethernet alapú rendszerek típusai [19]

2.2.1. SERCOS III [19]

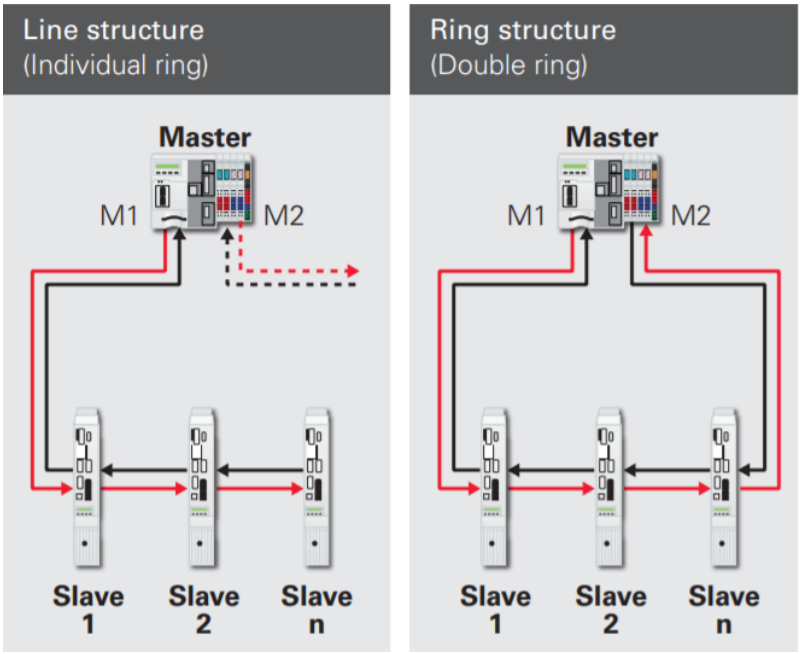
Kommunikáció alapjai

A jel terjedése

Az információ feldolgozása a csomópontokban történik ”on-the-fly” típusúan, vagyis a beérkező telegramot azonnal elkezdi az eszköz olvasni, nem várja meg amíg az egész kért adat megérkezik. Így a rendszernek összesített késleltetése szinte csak a hardver komponensek késleltetései, illetve a terjedési idő.

Megvalósítható topológiák

A SERCOS III protokollal több típusú topológiát is meg lehet valósítani. Ilyen a vonal, illetve a kör topológia. Kapcsolókkal, bővítőkkel lehetőség van fa topológiát is kialakítani[19]. Az Ethernet duplex tulajdonsága miatt könnyű a kör topológiával redundáns hálózatot kialakítani, mivel ha megszakad valahol a kör, az két vonalra oszlik, így nem történik leállás.



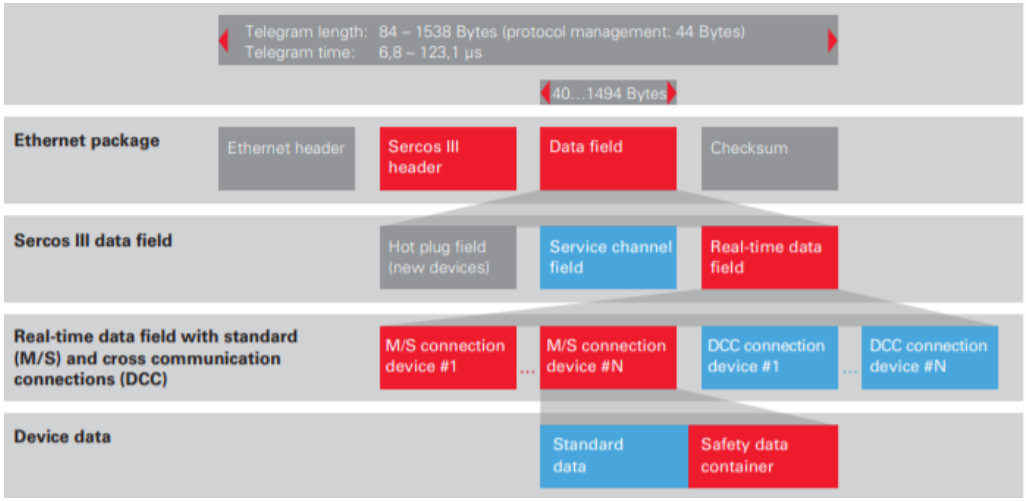
2.12. ábra. Az egyszerűen elérhető topológiák. [19]

SERCOS III kommunikációs ciklus

Telegram

A Sercos hálózatokban az általános Ethernet kommunikáció az Unified Communication Channel (UCC)-ben történik, mely lehetőséget biztosít egységek inicializálására, konfigurálására valósidejű alkalmazás előtt vagy akár a Sercos hálózatra csatlakoztatott számítógép internetes szolgáltatásainak teljes körű kihasználására, mint például a levelezés vagy a böngészés. A telegram másik részében található a hálózat valósidejű kommunikációt lehetővé tévő protokollja. Ennek a headerje leírja, hogy a hálózat épp mely állapotában van, megadja a Master általi Slave ütemezések leírásának, illetve a Slave-ek válaszának, egymás közötti kommunikációjának helyét a telegramban(2.13. ábra). Az adatmező három részből áll:

- Hot Plug mező: Újonnan csatlakoztatott nodeokkal való kommunikáció, melyek már üzemelő hálózatra kapcsolódtak.
- Szervíz Csatorna mező: Az olyan csatornák száma, melyek aciklikus kommunikációt folytatnak Master és Slave-ek között.
- Valósidejű Adat mező: Ez a mező használható aciklikus és ciklikus adattovábbításra is, itt lehet realizálni a valósidejű kommunikációt.



2.13. ábra. Sercos telegramm felépítése[19]

Real-time jellemzők

Maximális eszközök száma

A biztonságosan és elvártaknak megfelelően működő Slave egységek számát nagyban befolyásolja a hálózat kommunikációjának periódusideje.

Ciklusidő (μs)	Egy egységre jutó adatmező nagysága (byte)	Egységek maximális száma (UCC nélkül)	Egységek maximális száma (UCC-vel)
31.25	8	7	2
62.5	12	14	8
125	16	26	21
250	12	61	57
250	32	33	31
500	12	122	120
1000	50	97	95
1000	32	137	134
1000	12	251	245

2.3. táblázat. Sercos eszközök száma [19]

Időzítések

A használható ciklusidők függenek a hálózaton lévő egységek számától, a legkisebb elérhető periódusidő 31.25 μs. A Sercos protokollt használó egységekben szükséges egy hardveres komponens is, mely lehetővé teszi a valósidejű kommunikációt, így garantálja a kis válaszidőt (<0.5 ms) és a pontos szinkronizációnak köszönhetően a kis jittert (<0.1 ms) [16].

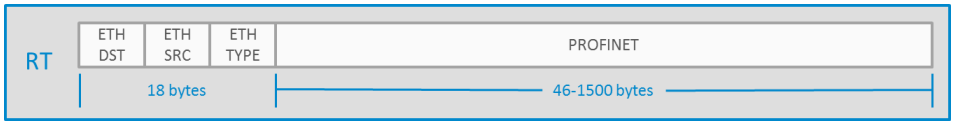
2.2.2. PROFINET IRT [18]

A PROFINET csatornák

A PROFINET IRT protokoll három kommunikációs csatornából épül föl. Ezek a Real Time(RT), Non Real Time (NRT) és Isochronous Real Time (IRT).

Real Time csatorna

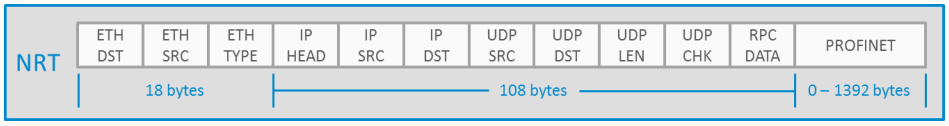
Késleltetés és jitter mind kerülendő real-time alkalmazásokban. Erre egy korlátolt megoldás az RT csatorna, mely kihagyja a hálózati, szállítási és viszonyréteget, így biztosítva az alacsony jittert és késleltetést. De mivel ezek a rétegek kimaradnak, így az IP cím adta differenciálást nem lehet használni, emiatt az eszközök között csak a MAC cím tesz különbséget, ez alapján azonosíthatóak az egységek. Valamint IP cím nélkül nem lehet helyi hálózatok között Real Time frame-eket küldeni.



2.14. ábra. A Profinet RT csatorna felépítése [18]

Non Real Time csatorna

Ezen csatorna az összes OSI réteget használja, így nem valósidejű, de sokkal szélesebb körben képes kapcsolatot létesíteni, akár az interneten keresztül is.



2.15. ábra. A Profinet NRT csatorna felépítése [18]

Isochronous Real Time csatorna

A Profinet IRT-nek eltérő a valósidejűséget biztosító kommunikáció kialakítása. Ahelyett, hogy egy telegramban vagy üzenetben megadott, specifikus helyen tárolná a kérdéses adatokat, a protokoll időszeletekre bontja a ciklikus kommunikációjához tartozó ciklusidőt. Ezen időszeleteknek a 25 százalékát[18] használja a determinisztikusságot erősen megkövetelő valósidejű adatok elküldésére, a többi szeletben pedig a gyengébb valósidejűséget is tűrő adatcserék folynak.

A kommunikációban résztvevő egységek

A résztvevő egységek kommunikációs feladatai három csoportba különíthetőek.

Device

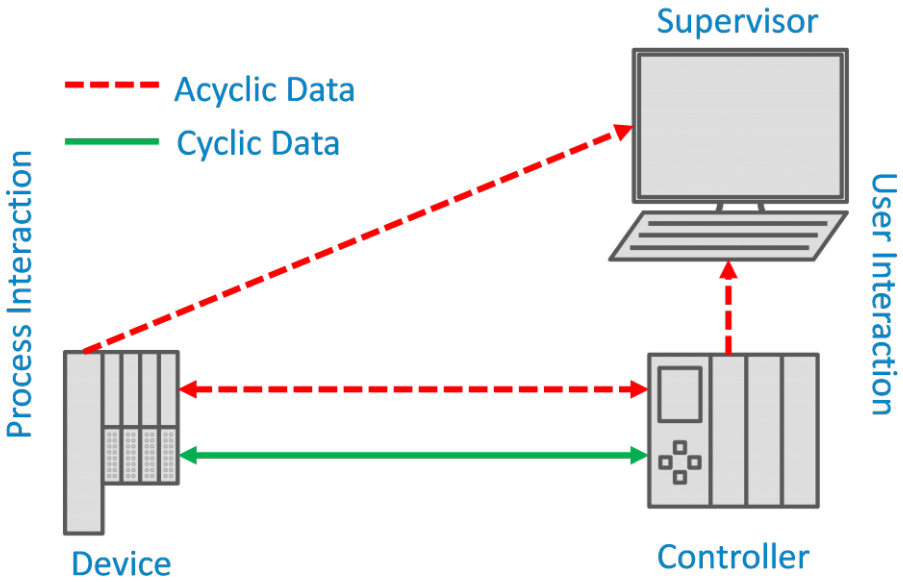
A Profinet IRT eszközök feladata a valósidejű kommunikáció lefolytatása. Ennek folyamata során a ciklikus adataikat elküldik a Controller-nek, és aciklikus diagnosztikai adatot vagy értesítést a Supervisor-nak. Nem folytatnak kommunikációt hasonló feladatú eszközökkel.

Collector

Feldolgozza az összes Device-tól hozzá küldött ciklikus valósidejű információkat, és karbantartási adatokat is tárol ezekről. Megvalósítása általában valamilyen PLC vagy ipari PC.

Supervisor

Ezek az egységek a hibajavításban, illetve a helyi hálózatok közötti kapcsolatok kialakításában vesznek részt, de felvehetik akár egy Controller szerepét is.



2.16. ábra. A Profinet IRT hálózat egységeinek kapcsolata[18]

Real-time jellemzők

Maximális egységek száma

Lényegében nincs fizikai korlátozás az egységek számára(Összes lehetséges MAC cím 2^{48}. Több egységből álló hálózat kialakítása a Profinet IRT protokoll segítségével lehetséges, de nagyobb darabszámnál már komoly konfigurációs komplexitás fordulhat elő, illetve különböző kapcsolókat kell implementálni a rendszerbe, mely költséges.

Időzítések

A Profinet IRT-vel elérhető maximális válaszidő kevesebb mint 1 ms . A kommunikáció jittere kevesebb mint 0.1 ms.

2.2.3. EtherCAT röviden [8]

Tekintettel arra, hogy a következő fejezetben részletesen kifejtem az EtherCAT protokoll működését, így itt csak a Real-time jellemzők összehasonlítását végzem el.

Maximális egységek száma

Az EtherCAT protokollal a maximális egységszám 65535, így az implementációkban e téren szinte nincsen korlát[8].

Időzítések

Az EtherCAT protokollal megvalósított hálózatokban a válaszidő akár 25 μs is lehet, de általánosan 100 μs. Míg a jitter 1 μs alatt is lehet[16].

2.3. Protokollok összehasonlítása

A nem Ethernet alapú protokollok bár lehetnek valósidejűek, mégis egy komplex hálózat kialakításakor több hátrányuk is van. A kapcsolat az egységek között speciális, így nem a legköltséghatékonyabb ezeket kiépíteni, illetve a kommunikációjuk más típusú rendszerekkel újabb megoldandó problémákat vet fel. Ezzel szemben az Ethernet alapú hálózatok összeköttetése gyors és költséghatékony (Ethernet kábel) és a változó szintű Ethernet protokoll megvalósításának köszönhetően egyszerűbben lehet több alrendszert összefűzni, lehetőség van akár nem Ethernet alapú hálózatokat is felkapcsolni a főrendszerre. Az Ethernet alapú hálózatok között kiemelkedik az EtherCAT protokoll, mely egyike a legjobb teljesítményű protokolloknak és egyúttal legköltséghatékonyabbaknak is. Hosszútávú növekedését és életképességét az EtherCAT-nek a piacban betöltött szerepe is indikálhatja(2.4. táblázat).

	EtherCAT Standard	PROFINET IRT Standard	SERCOS Standard
Cégcsoport tagszám	3200	1400	90
Gyártó tagszám	500	80	90
Szervo-motor cégek	150	12	7
I/O cégek	100	6	4

2.4. táblázat. A jelenlegi piac felépítése[16]

Végső összevetés

Bár minden protokollnak megvannak az előnyei és hátrányai is egyben, de az első fejezet alatt leírt protokollok közül is láthatóak az iparban is még sokat alkalmazott, egyre erősebb, illetve a mai felhasználásokhoz kevésbé alkalmazkodó, gyengébben teljesítő rendszerek. Véleményem szerint hosszútávon az Ethernet alapú valósidejű kommunikációs rendszerekben van a jövő, ez a terület napról napra fejlődik, és még sok kiaknázatlan lehetőség van benne.

	CAN	LIN	FlexRay	EtherCAT	PROFINET IRT	SERCOS
Maximális kábeltávolság	1000m	40m	24m	100m	100m	100m
Jitter	x	x	x	<0.1 ms	<1 ms	<0.1 ms
Maximális sávszélesség	1 Mbps	20 kbps	2x10 Mbps	100 Mbps(10 Gbps)	100 Mbps	100 Mbps
Eszközök száma	32	16	Egy topológiában: 22	2^{16}	2^{48}	251
Hálózat topológiája	busz	busz	csillag,busz	tetszés szerinti	tetszés szerinti	vonal,kör
Hálózatépítés komplexitása	közepes	közepes	bonyolult	egyszerű	egyszerű	egyszerű

2.5. táblázat. Fieldbus rendszerek összehasonlítása

3. fejezet

Az EtherCAT protokoll

3.1. Fizikai réteg

Szükséges fizikai eszközök

Ethernet kábel

Mint minden Ethernet alapú protokollnál, itt is órási előnyt jelent, hogy az általános, olcsó Ethernet kábelt lehet használni az EtherCAT egységek között egészen 100 méteres távolságig. Efölött már az üvegszálas kábel biztosítja a megfelelő működést[10].

A jel terjedése

Az üzenet a Master-től indul, csak ez kezdeményez kommunikációt. A kiküldött frame végigfut a rendszeren, a Slave-ek, melyeken áthalad az üzenet "on-the-fly" dolgozzák fel az adatokat, kiolvassák a rájuk eső részeket és írnak az üzenetbe. Ennek köszönhetően a kommunikáció nagyon gyors, szinte csak a jel terjedési sebessége és a Slave egységek hardveres késleltetése határozza meg a kommunikáció sebességét. A telegramm által érintett utolsó egység (Slave), mely érzékeli a telegrammot és csak egy darab nyitott portja van, ahonnan az adat érkezett, visszaküldi ugyan ezen a porton keresztül a frame-et az Ethernet full-duplex tulajdonságát kihasználva. Ha kör alapú a topológia, akkor minden Slave csak továbbít, nincs szükség visszaküldésre. Ezen tulajdonságok miatt a Slave-ek közötti szinte közvetlen kommunikáció is lehetséges.

Master

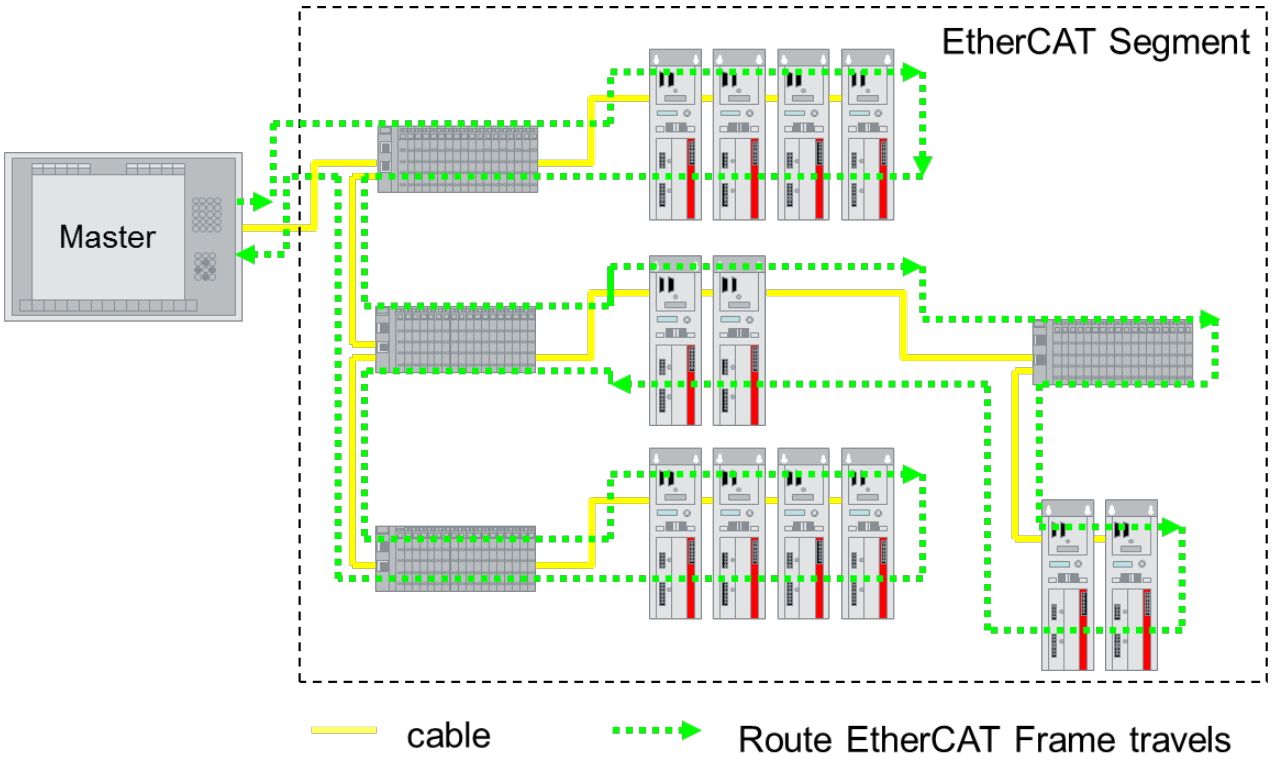
A Master realizálásához nem szükséges semmilyen specifikus hardver, csupán egy Ethernet port. A MAC kontrollerhez közvetlen memória hozzáféréssel indítja a kiküldendő üzeneteket, frameket, emiatt nem függ a Master kontroller teljesítményétől a kommunikáció.

Slave

A gyors és megbízható információcsere miatt szükséges a Slave egységekben speciális ASIC, vagy FPGA, mely kezeli a kommunikációt az Ethernet portokon keresztül a Masterel.

Topológia

Az elérhető topológiákban nincs megkötés, ráadásul a csatlakoztatható maximális egységek száma 65535, így szinte bármilyen rendszer megvalósítható. Ezen tulajdonságok



3.1. ábra. Az EtherCAT jel terjedése[9]

mellett a flexibilitását növeli a ”Hot-Plug” lehetőség, vagyis futás közben lehet kicserélni Slave-eket anélkül, hogy a rendszer működését meg kellene állítani[9].

Redundancia

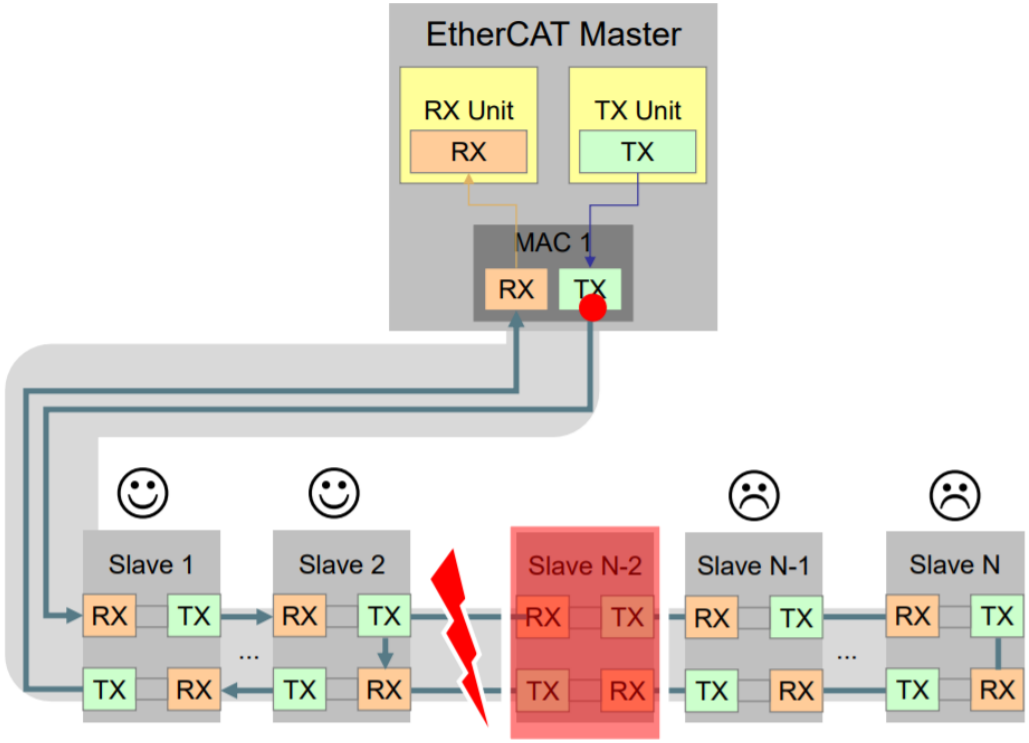
Az Ethernet kábel full-duplex tulajdonságát és a lehetséges topológia rugalmasságát felhasználva redundáns rendszereket lehet létrehozni, ahol a futás közben kiesett egység nem bénítja meg a kommunikációt. Ehhez a Master-nek legalább két Ethernet portal kell rendelkeznie. Ha egy köztes egység kiesik, akkor a kör topológiából két busz topológia alakul ki, így folytatódik a kommunikáció.

3.2. Adatkapcsolati réteg

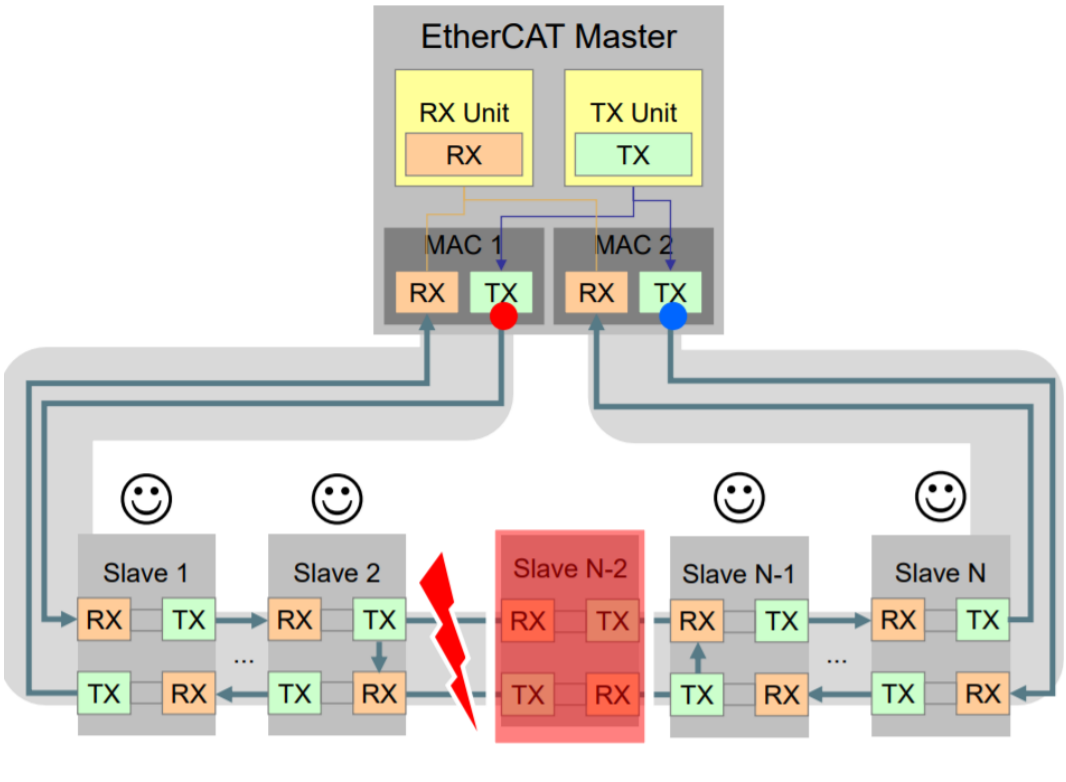
EtherCAT frame felépítése

Az EtherCAT üzenet egy frame-ből áll, melyet a Master ciklikusan küld ki a Slave-eknek. Ez a frame minden egyes Slave-en keresztülhalad, minden Slave a kiküldött frameben található adatot helyben elemzi, ahogy az áthalad rajta. Az EtherCAT üzenet bele van ágyazva egy általános Ethernet üzenetbe. Ezen üzeneteknek két típusa van, melyeket az üzenet datagramm header-je specifikál:

- Olvasás, írás és olvasás-írás
- Egy adott Slave-hez való hozzáférés direkt címzéssel, vagy több Slave-hez logikai címzéssel



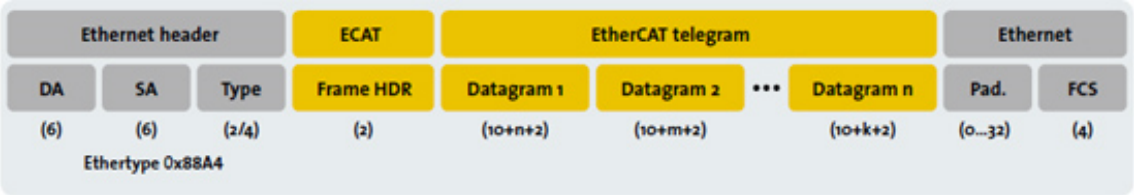
3.2. ábra. EtherCAT rendszer redundancia nélkül[9]



3.3. ábra. EtherCAT rendszer redundanciával[9]

A datagrammokban tárolódik az adat, ezt tudják írni vagy olvasni a Slave és Master egységek. A Slave-ek belső memóriája hozzá van rendelve egy-egy datagramm címhez, így be lehet állítani, hogy egy datagrammot melyik Slave(Slave-ek) írják, olvassák. Master oldalról a DMA végzi a datagramm átmásolását, így csökkentve a CPU terhelését.

A datagramm a Slave-en elvégzett utasítások számát is nyilvántartja, a Master ezáltal tudja ellenőrizni, hogy a Slave-ek működése megfelelő-e. A rendszer felállásakor minden Slave-hez kötődni fog egy vagy több cím, esetleg egy logikai címmel el lehet több Slave-et is érni. Mivel a datagrammok minden adatot tartalmaznak, így a Master eldöntheti, hogy adott pillanatban mely adatokat kérdezi le. Ezáltal a konvencionális field-bus rendszerekhez képest a Master kevésbé van leterhelve, nem kell minden egység adatát kiolvasni, és ezeket a memóriába mentenie[10].



3.4. ábra. EtherCAT frame felépítése[9]

Címzés és utasítások

A hálózatba csatlakoztatott egységek mind rendelkeznek egy EtherCAT címmel, melyet a Master egység szab meg. A címzésnek két módja van: fix címzés, illetve automatikusan inkrementálódó címzés. Előbbi módszernél minden egység a rendszer indulása előtt kap egy fix 16 bites címet, míg az utóbbinál a hálózat első Slave-je megkapja a 0-ás címet és a soron következők mindig egyel kisebb címet kapnak(0xFFFF,0xFFFE stb), ezt általában konfigurációs beállításoknál használják[9]. Mindkét címzési módra külön utasítások használhatóak, így optimalizálva a Slave-ekhez való hozzáférést.

SyncManager

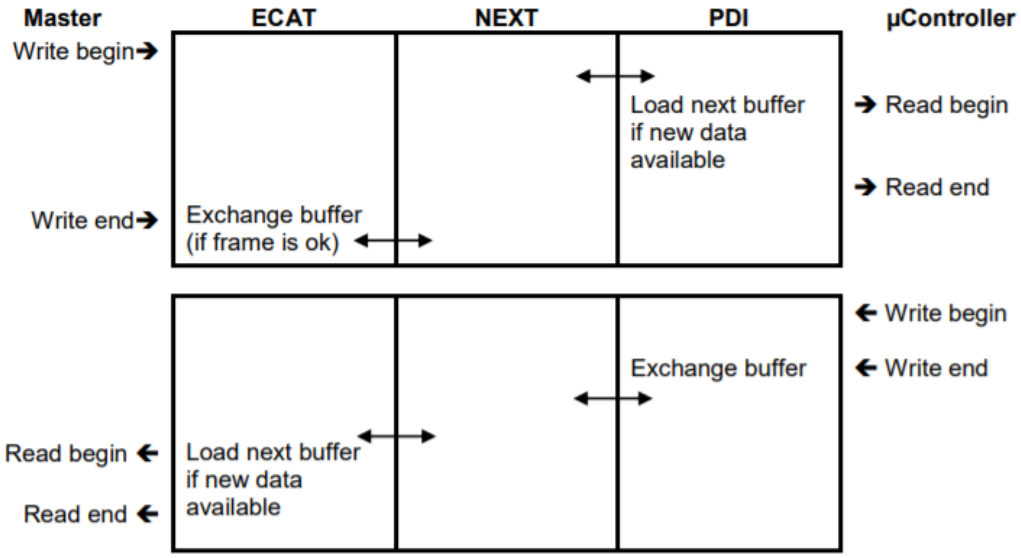
Minden Slave-ben található EtherCAT ASIC egy részegysége, mely összehangolja, szinkronizálja a Master által küldött frame és a Slave implementációját végző mikrokontroller adatcseréjét. Segít megtartani a cserélendő adat konzisztenciáját, biztonságos kommunikációt tesz lehetővé. Két működési módja van: Buffered Mode és Mailbox Mode[11].

Buffered Mode

A konzisztens adatcsere érdekében létrehoz egy kommunikációs buffert, melyhez a Master és az adott Slave megvalósítás is bármikor hozzáférhet. A beleírt adatokat elraktározza, melyet a másik oldal bármikor kiolvashat. Ha gyorsabb az írás mint az olvasás sebessége a maximális buffer hely elérése után a legrégebbi adatok törlődnek. Ezt használják általában a ciklikus adatok továbbítására.

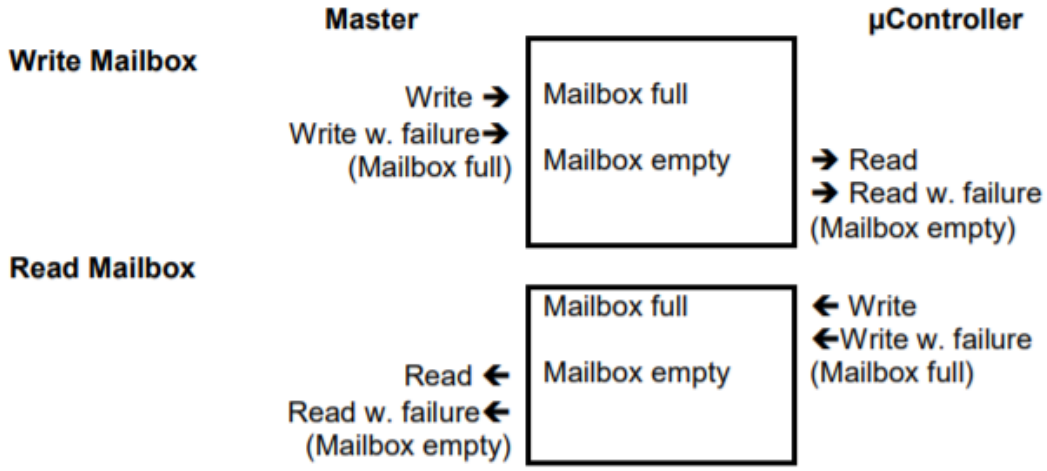
Mailbox Mode

Egy kézfogásos mechanizmust valósít meg ez a módszer, így garantáltan nincs adatvesztés. Az egyik oldal által írt adatot előbb ki kell olvasnia a másik oldalnak, és csak



3.5. ábra. Buffered Mode működése[11]

ezután lehet újra írni. Lassabb megoldás és körülményesebb, ezért ezt általában szervíz és beállítási üzenetek továbbítására használják. Több protokol is értelmezhető ezeken az üzeneteken keresztül mint például: CANopen, Ethernet, Servo Drive vagy File Access.



3.6. ábra. Mailbox Mode működése[11]

FMMU

Fieldbus Memory Management Unit az a részegység, mely a képes a logikai címzéseket fizikai címekké alakítani, így lehetséges, hogy több Slave egység alakít ki egy adatszegmenst, vagy ugyan azt az adatot több egység is írhatja és olvashatja.

Diagnózis

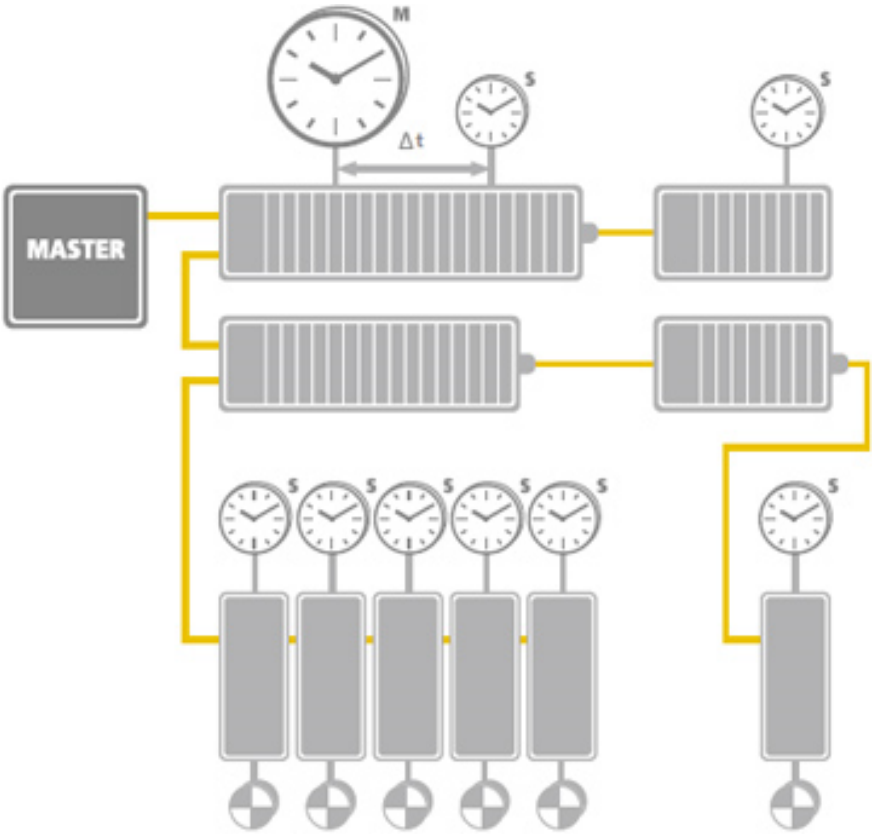
Az áthaladó üzenet tartalmaz egy ellenőrző összeget, melyet minden egyes Slave újraszámol az üzenet áthaladásánál, ha hibás, akkor megnövekedik a hibaszámláló. A Master

ezáltal képes visszaszámolni, hogy melyik egységnél történhetett a hiba. Minden datagrammban van egy úgynevezett Working Counter, melyet minden Slave megnövel egyel, ha az adott Slave memóriája van a datagrammban megcímezve. Így a Master képes meghatározni, hogy minden egység megfelelően működik-e, ha nem, akkor a kapott adatot nem továbbítja a kontroll alkalmazásnak.

3.3. Elosztott órák

Olyan applikációkban, ahol szükséges a maximális pontosság, megjelenik az EtherCAT Slave-ek újabb előnye: Az elosztott órák(Distributed Clocks, DC). Ez azt jelenti, hogy minden Slave egységben található egy különálló óra, mely nanoszekundum pontossággal számol. Ezeket felhasználva a Master képes egy órabeállító telegramm kiküldésével összehangolni az órákat, úgy, hogy azok kevesebb mint 100 ns-os eltéréssel működjenek[9]. Ez a telegramm képes a terjedési késleltetést is figyelembe venni, így növelve a pontosságot. Ezek az órák képesek minden órajelre egy megszakítást generálni, az úgynevezett SYNC jelet, mely a mikrokontrollerhez eljuttatva nagy pontosságú mintavételezést tesz lehetővé.

A elosztott órák tehermentesítik a Mastert, mivel a folyamatok nem a frame kiküldésére és annak megérkezésére reagálnak, hanem a belső órájukra, így a Master telegrammjainál nem követelmény a legnagyobb pontosság. Ez olyan megoldásoknál jelent nagy előnyt ahol párhuzamosan, azonnal kell végrehajtani feladatokat az idő függvényében (például sebességmérésnél a pozíciómérése).



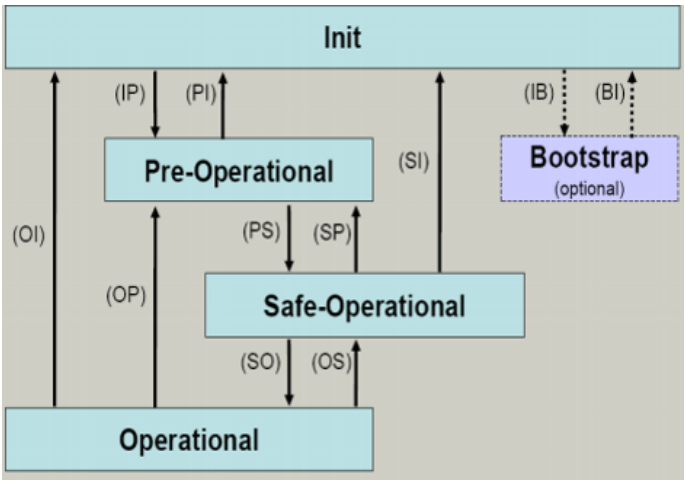
3.7. ábra. Az elosztott órák hardver alapú szinkronizálása[8]

3.4. Alkalmazási réteg

Állapotgép

A Slave megfelelő működéséhez össze kell hangolni a Slave-et és a mikrokontrollert. Ehhez segítséget ad az EtherCAT Slave-ek általános állapotgépe, mely négy állapotból áll.

- INIT: Elnevezéséből adódóan a rendszerek inicializálása itt történik meg, itt engedélyezzük az egyszerűbb kommunikációkat a két eszköz között.
- BOOT: Ebben az állapotban frissülhet a firmware-je a Slave-nek.
- PRE-OP: Pre-Operational állapotban már mailbox típusú kommunikáció működőképes, bár PDO-n(4.3 alfejezet) keresztül nem lehet még adatot továbbítani.
- SAFE-OP: Safe-Operational állapotban már PDO Input-ot is tudunk értelmezni, melyet a Master küld a Slave-eknek, de Output még nem aktív, a Slave nem tud adatot küldeni ilyen módon.
- OP: Operational: Ebben az állapotban már az összes típusú kommunikáció működőképes, ez a végső állapot melyet a hálózatban minden Slave-nek el kell érnie a kívánt működés érdekében.

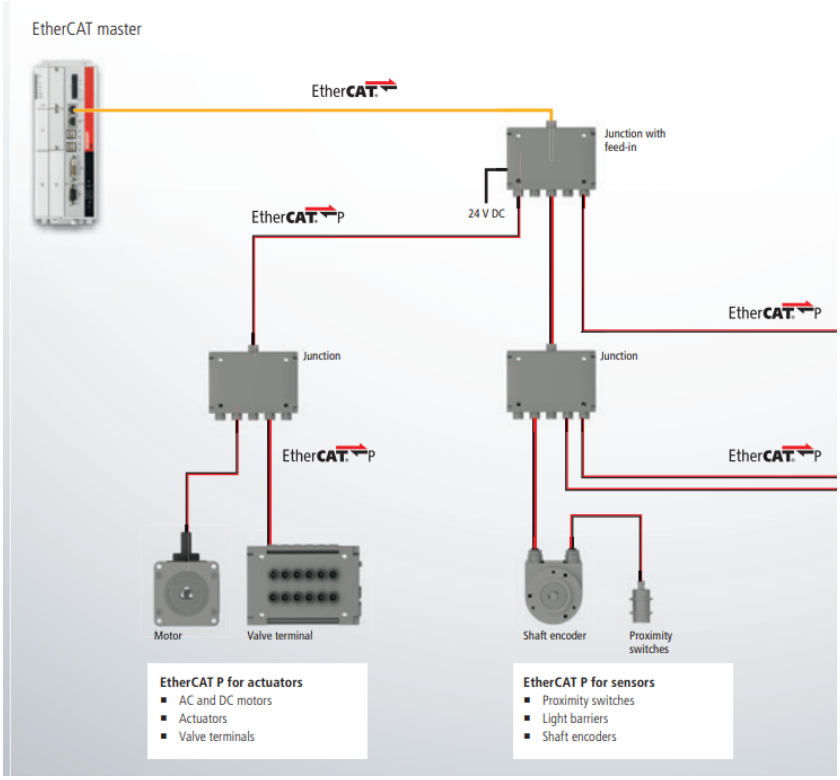


3.8. ábra. Az engedélyezett váltások az állapotok között[8]

3.5. További lehetőségek

EtherCAT P

Az általános Ethernet kábeleken is lehetséges tápfeszültséget továbbítani a jel mellett, így egy vezetékkel meg lehet valósítani két funkciót, ezzel is csökkentve az eszközökhöz szükséges kábeleket. A kábel ezen képességét az EtherCAT eszközök is képesek felhasználni az EtherCAT + Power keretein belül, így lehetséges olyan hálózatot kialakítani, ahol az eszközök között elég egyetlen, univerzális kábel.



3.9. ábra. Hálózat megvalósítása EtherCAT Power segítségével[8]

EtherCAT G

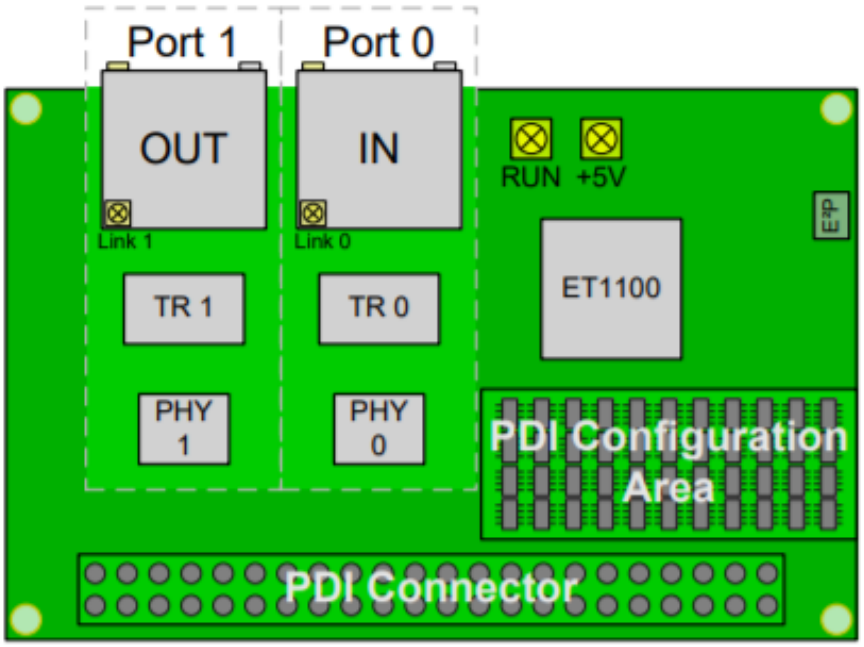
A kommunikáció alapértelmezett sebessége 100 Mb/s, de más típusú kábelekkel ezt akár 1 Gb/s-ra is fel lehet növelni. Sőt a kommunikációban új szegmens integrálásával akár 10 Gb/s is lehetséges. Ezek a különböző sebességű alrendszerek integrálhatóak az alap rendszerrel.

4. fejezet

Mérésadatgyűjtő és vezérlő rendszerben felhasznált eszközök

4.1. FB111-0141

Az FB1111-0141 egy tesztelésre fejlesztett kontroller lap, mely tartalmaz egy EtherCAT kommunikációt folytató ET1100 ASIC-et, ezt az ASIC-et az FB1111-en található Ethernet portok kötik össze a EtherCAT hálózattal, illetve az FB1111 csatornát biztosít a kommunikációra az ET1100 és az alkalmazás mikrokontrollere között.



4.1. ábra. FB111 felépítése[2]

Process Data Interface

A Process Data Interface, röviden PDI, valósítja meg az összeköttetést az alkalmazást futtató mikrokontroller és a kommunikációt végző ASIC között. Ennek formája az FB1111 variánsainál többféle is lehet (lásd a 4.1 táblázatban), de az FB1111-0141 csak SPI perifériát köti az ET1100-hoz, így én ezt használtam.

Alkód	PDI interfész	Leírás
0140	µController	16/8 bit asynchronous Microcontroller Interface
0141	SPI	Serial Peripheral Interface (Slave)
0142	Digital IO	32 bit In/Out digital interface

4.1. táblázat. FB1111 fajtái[2]

ET1100

Az EtherCAT kommunikáció egyik követelménye a Slave egységekben lévő ASIC vagy FPGA mag. Azért csak ilyen magokat használnak, mert szükség van a kommunikáció gyorsaságára és pontosságára, amely tulajdonságokat ilyen elemekkel könnyebb megvalósítani. Az általam létrehozott rendszerben ennek a szerepét az ET1100 ASIC fogja betölteni. A Slave működésének konfigurációja két forrásból ered, az egyik a Master által, a rendszer felállásánál kiküldött beállítások, a másik pedig az ET1100-ban lévő EEPROM-ban tárolt, a Slave bekapcsolásakor betöltődő beállítások.

EEPROM

Az EEPROM minden EtherCAT eszközbe szükséges, mivel ez tárolja a Slave-et jellemző adatokat az EtherCAT Slave Információt(ESI). A kötelezően tárolt, beállításhoz szükséges információk a következőek:

Cím(Word)	Név	Leírás
0x0	PDI Control	Inicializációs érték, meghatározza a PDI típusát
0x1	PDI Configuration	Inicializációs értéke, adott típus működésének beállítása
0x2	SYNC jel pulzusának hossza	
0x3	Extended PDI Configuration	További PDI beállítások
0x4	Configured Station Alias	Az egység elérési nevének beállítása
0x5	Fenntartott	
0x6	Fenntartott	
0x7	Ellenőrző összeg	

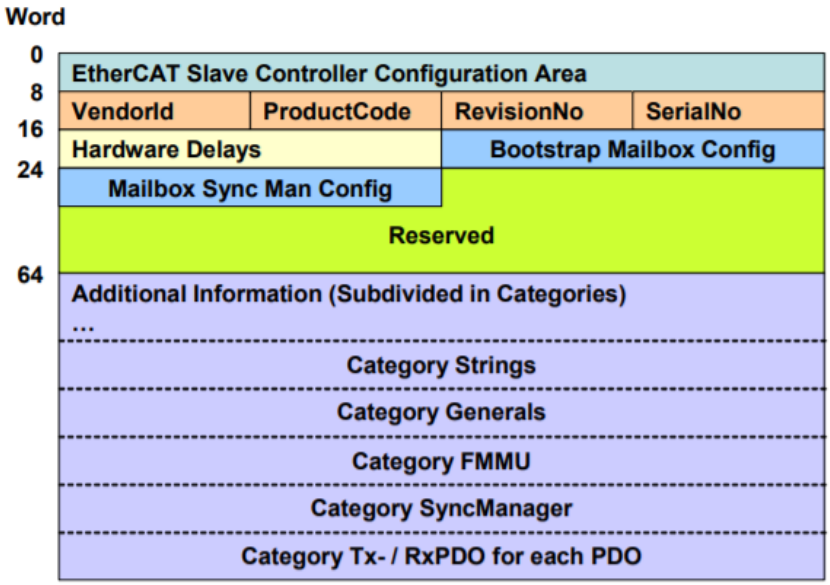
4.2. táblázat. Főbb tárolt beállítások[11]

Ezen kívül általában megtalálhatóak még a különböző mailbox beállítások, illetve hardver leírások. A többi bejegyzés gyártófüggő.

Distributed Clock

Az ET1100 is rendelkezik elosztott órával, mely több dologra is használható:

- Slave egységek (és Master) közötti órajel szinkronizáció
- Egyidejű kimeneti jelek generálása (SYNC jelek)
- Input események pontos időbélyegezése
- Egyidejű digitális input/output frissítés

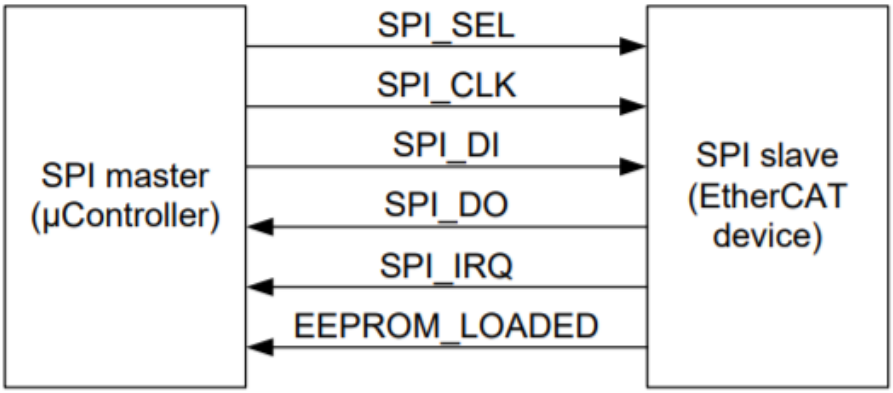


4.2. ábra. EEPROM felépítése[11]

Az órák pontossága azon alapszik, hogy az összes egység egy rendszeróráról működik, mely lehet vagy a Master, vagy a Master által kijelölt Slave egység. Erre az órajelre hangolódik az összes eszköz a rendszer felállásakor, a Master által kiküldött beállító telegramm segítségével. Az időt nanoszekundum pontossággal mérik, 2000. január 1 00:00-tól kezdődött a számítás, és mivel 64 bites az ehhez tartozó regiszter, így több mint 500 évig kell járnia az óráknak addig, amíg túlcsordul.

SPI

Mivel az FB1111 board csak SPI perifériával való csatlakoztatást tesz lehetővé, így az EtherCAT egység(ET1100) SPI kommunikációs protokolljának megismerése is fontos volt. A 4.3-as ábrán is látható, hogy nem csak a szokványos SPI összeköttetés valósul meg, hanem az úgynevezett EEPROM LOADED vezeték is részét képezi a kommunikációnak. Amikor a hálózat inicializálódik a Master szétküld egy parancsot a Slave egységeknek, hogy töltsék be az EEPROM-jaikban található adatokat. Ezen betöltődés után jelez az ET1100 az EEPROM LOADED vezetéken a mikrokontrollernek, hogy készen áll az SPI kommunikációra.



4.3. ábra. SPI master és slave kapcsolata[11]

Címzés

Az EtherCAT egység memóriájának kétféle címzési módja létezik. Az egyik a 2 byte-os címzés, amely által 8 Kbyte memóriát lehet elérni, illetve a 3 byte-os mely által 64 Kbyte-ot. A teljes eszközkezelés és elérés érdekében az utóbbit választottam. Az olvasás tipikus címzésének felépítése a 4.4-es ábrán látható[13].

Az olvasás úgy történik, hogy a kiküldött cím után folyamatosan 0x00 értékű byte-okat küldünk, és ezekre válaszolva küldi el az EtherCAT egység az adott című memória tartalmát. Az olvasás kezdetét egy 0xFF várakozó mezővel jelezzük, és a végét is ezzel zárjuk le. Az írás annyiban különbözik, hogy nincs szükség várakozó mezőre, és bármelyik kiküldött byte után megszakítható, nem szükséges lezáró mező sem.

Az ET1100 címe kiolvasásnál és írásnál is automatikusan inkrementálódik, így lényegében egy címzésből(0x0-ra) kiolvasható lenne a teljes címtartománya.

3 Byte address mode	
A[12:5]	address bits [12:5]
A[4:0] CMD0[2:0]	address bits [4:0] 3 byte addressing: 110b
A[15:13] CMD1[2:0] res[1:0]	address bits [15:13] read command: 011b two reserved bits, set to 00b
0xFF	wait state byte
D0[7:0]	data byte 0
D1[7:0]	data byte 1

4.4. ábra. SPI 3 byte olvasás[11]

Mód

Négy SPI átviteli mód létezik a mintavétel időpontjának (órajel felfutó vagy lefutó éle) és az órajel polaritásának függvényében. Én a mikrokontroller és ET1100 kommunikációjára az EtherCAT Group által ajánlott módot, a 3. módot választottam, ahol a invertált a polaritás és a felfutó élnél történik a mintavétel.

4.2. TMS320F28069 Piccolo

A projekt elején olyan tesztboardokon található mikrokontrollereket vettem szemügyre, melyek már integrálták magukba az ET1100-at, ilyen volt pl.: a TMS320F28377D Deflino[20], vagy a LAN9252[21]. Végül már egy, a ProDSP Technologies Zrt. által bejáratott mikrokontrollert választottam, a TMS320F28069 Piccolo-t. Ennek több oka is volt, az egyik az, hogy ez már egy ismert mikrokontroller, így több, a fejlesztést és működés megfigyelését könnyebbé tévő szoftver rendelkezésemre állt már, illetve így nekem kell megoldani a rendszer összes összeköttetését, összehangolását, ezáltal még mélyebben felfedezve a tulajdonságait, képességeit.

Maga a mikrokontroller 32 bites akár 90 MHz-es órajelű működésre is képes. SPI periféria, AD átalakító, illetve külső megszakítást lehetővé tévő megszakításkezelés is elérhető volt rajta, így minden szükséges funkciót meg tudtam rajta valósítani.

A mikrokontroller a cég által korábban kidolgozott projekt kártyáján volt, így én is ezt használtam, de a kártyán csak a már meglévő csatlakozókat és egy piros LED-et vettem igénybe.

4.3. TwinCAT 3

A TwinCAT 3 egy univerzális szoftver EtherCAT rendszer tervezésre. Nem csak konfigurálni és tesztelni lehet vele, de a Windows mellett futó valósidejű kernelének segítségével akár 50 μ-os időzítés is megvalósítható. Így egyszerre valósíthat meg rendszereszközt, fejlesztő és futásidejű környezetet. Utóbbi két megoldáshoz szükség van Visual Studio 2015 vagy 2017-es verziójára, mivel a TwinCAT mint egy kiegészítő csatlakozik a Visual Studiohoz. A TwinCAT 3 szoftverrel több típusú modul is fejleszthető, legyen az C++ vagy bármelyik a PLC programozási nyelvek közül, ez annak köszönhető, hogy minden mobulnak létezik egy TwinCAT Component Object Model(TcCOM) leíró struktúrája, amely definálja a modul viselkedését és karakterisztikáját[6].

PC mint EtherCAT Master

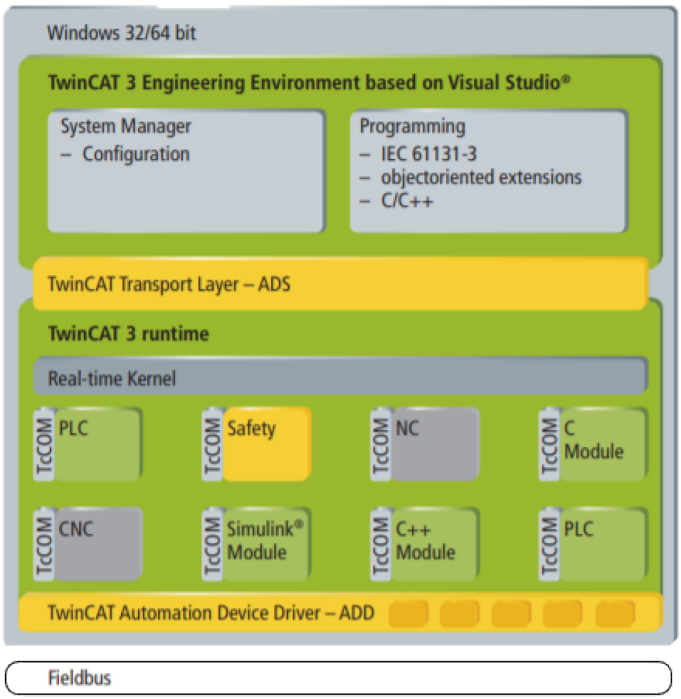
A 3.1 alfejezetben említettem, hogy Master-hez egyedül egy Ethernet port szükséges. Ezt felhasználva egyedül egy drivert kell telepíteni, mellyel a szoftver képes az Ethernet porton keresztül EtherCAT frame-eket küldeni és fogadni. Ezután ehhez a porthoz csatlakoztatott Slave egységek felismerik a PC-t mint Master egységet.

Rendszer

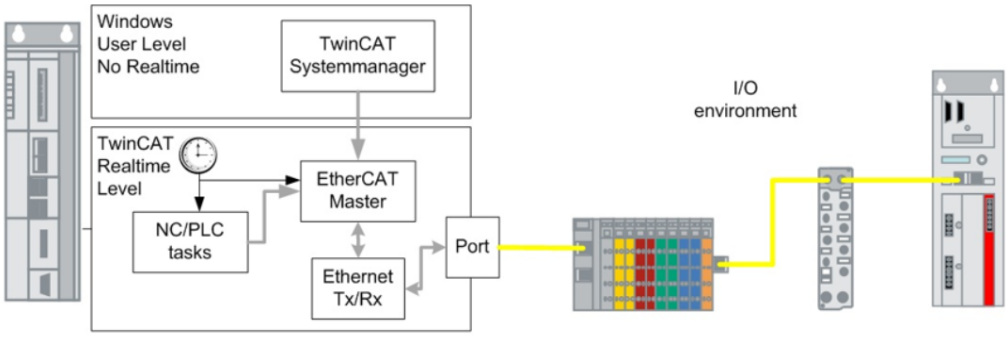
Mivel a TwinCAT szoftver mélyen a Windows működésébe van beágyazva, így több lehetőség van a program által a rendszert az igényeinkhez szabni.

Taskok

Az általunk fejlesztett vezérlő program, illetve a kommunikáció időzítésére taskokat kell alkalmazni. Minden taskhoz hozzá lehet rendelni egy, vagy több programot, és a rendszer ezeket a taskokat és így a hozzájuk rendelt programokat fogja a beállításaiktól függően indítani. Ezek a beállítások kiterjednek a task prioritására és ciklusidejére. Az EtherCAT kommunikációnak és a taskoknak is van egy sorrendje, melyet a TwinCAT 3 szoftver kezel.



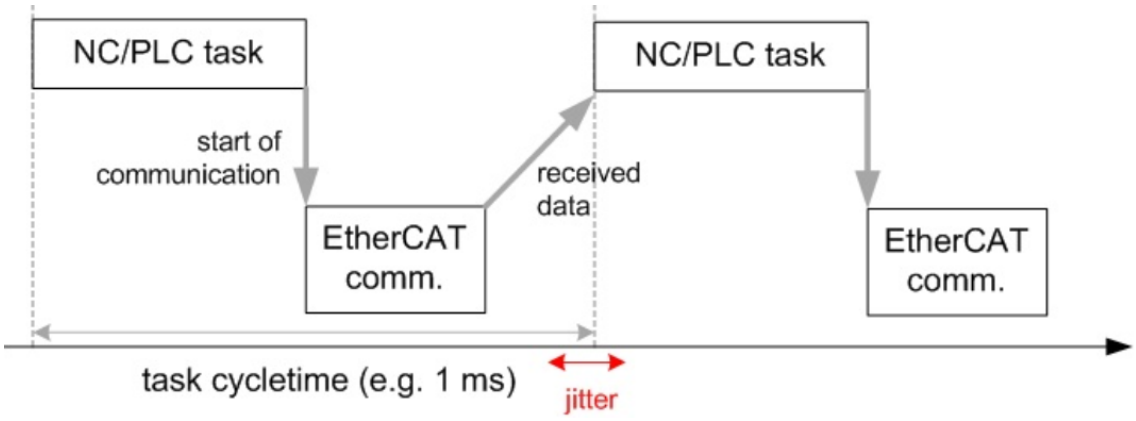
4.5. ábra. TwinCAT 3 helye a számítógép rendszerében[3]



4.6. ábra. EtherCAT Master megvalósítása Windows rendszerben

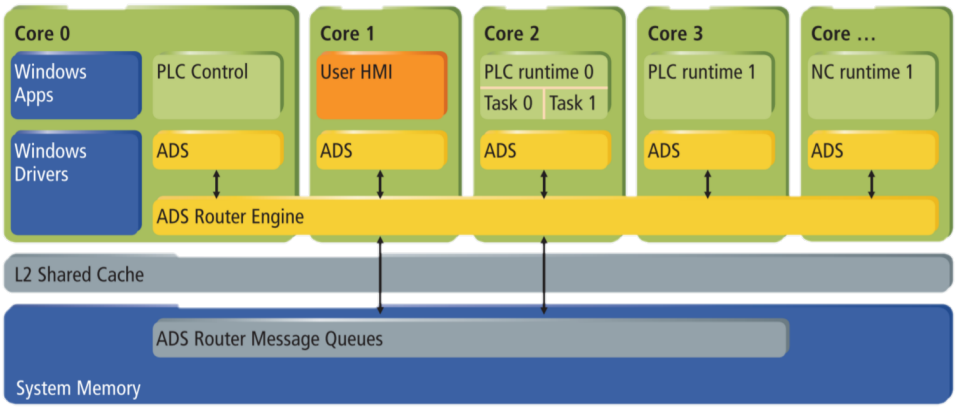
Valósidejűség

Egy általánosan működő Windows platformon a valósidejűség nem garantálható. TwinCAT 3-ban lehetőségünk van a több magú processzorok működésébe beleszólni. Megszabhatjuk a magok hozzárendeltségét, ezáltal megadjuk, hogy a Windows, vagy a TwinCAT számára van-e lefoglalva az adott mag. A Windows által lefoglalt magokon is képesek vagyunk finomabb beállításokat megadni, például limitálhatjuk, hogy az a mag mennyit foglalkozzon az operációs rendszer kéréseivel, és mennyit a TwinCAT programmal. Minden maghoz taskot kell rendelni, a kernel a beállított magon keresztül tudja meghívni a taskot és abból a programot. A magfelosztás segítségével elkülöníthetünk magokat, melyek mint egy kis PLC futnak a gépünkben, és csak a számukra megadott feladatokat hajtják végre.



4.7. ábra. Egy rendszer ciklus felépítése[3]

Minden magnak be lehet állítani a saját bázisidejét ami 50 μ-tól egészen 1 ms -ig terjedhet, ez azt jelenti, hogy a kernel időzítője ilyen gyakran kezeli a maghoz csatolt taskokat[3].



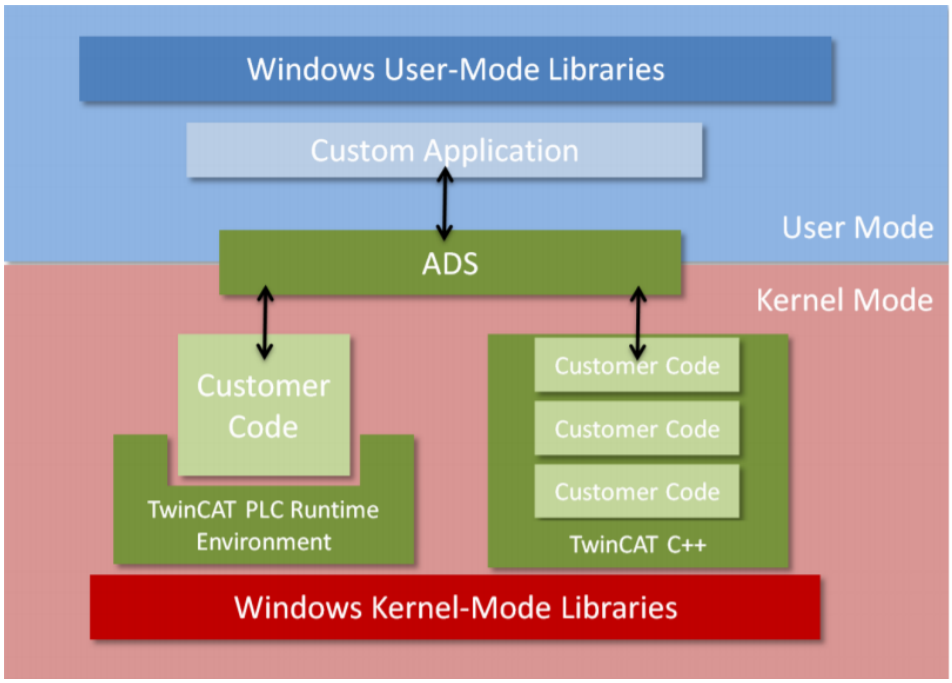
4.8. ábra. A több magos felépítés[3]

PLC

A szoftver segítségével több féle PLC programozáshoz megszokott nyelven is lehet fejleszteni, mint például a Ladder Diagram, Function Block Diagram vagy Structured Text. Én az utóbbit választottam tekintve, hogy ez áll a szememben a legközelebb az általam ismert programozási nyelvekhez, mint például a C vagy a C++. Több projektet is létre lehet hozni, ezeket hozzárendelni taskokhoz, így akár egyszerre több PLC-t is lehet egy számítógéppel szimulálni. Az összes ilyen kód a TwinCAT Runtime Enviroment-ben fut, mely lehetővé teszi a valósidejűséget[3].

C++

Egyik nagy újítása a TwinCAT 3-nak az, hogy képes C++ programot is valósidejűen futtatni, így lehetőség van ezen a nyelven is PLC-re fejleszteni. A C++ modulok nem részei a Runtime Enviroment-nek, ezért a valósidejűségük úgy biztosítható, hogy kernel modulokként lesznek végrehajtva (.sys), így a Windows Kernel Mode könyvtár segítségével állíthatóak össze ezeket a modulok.



4.9. ábra. A PLC és C++ modulok felépítése[4]

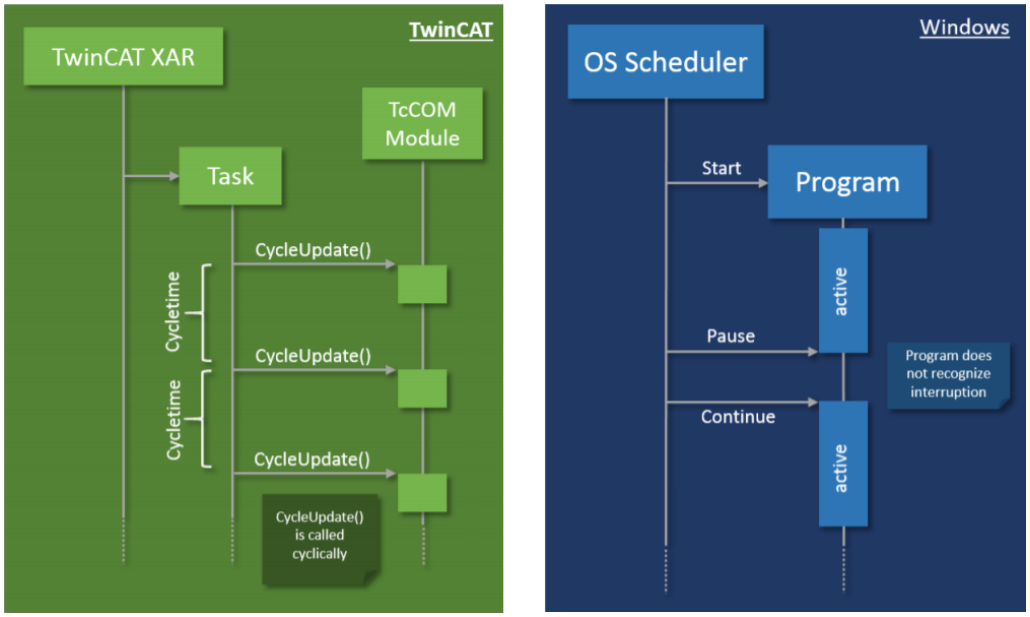
Felhasználó módú programfejlesztésnél a program megalkotása után az operációs rendszer végrehajtja azt, ezután a program önállóan tud futni. De a multi tasking miatt, az OS bármikor megállíthatja a program működését, úgy, hogy az általa futtatott program ezt nem érzékeli. Ezzel a viselkedéssel nem lehetséges real-time rendszert alkotni, vagyis egy blokkoló, eseményvezérelt rendszerből egy determinisztikus, ciklikus rendszerre kell áttérni. A PLC működésének a példájára a C++ modulnál a vezérlést és IO működését is a TwinCAT valósidejű környezet kezeli. A C++ program ciklikus kialakítása miatt ez a környezet hívja meg, és ez a korábban megszabott ciklusidő alatt lefut. Míg hagyományosan a futó program kezeli az egyidejűséget (Threadek nyitása, zárása), addig ebben a megvalósításban a futó program erre nem képes, egyedül az őt meghívó környezet(4.10 ábra).

Fejlesztési követelmények

A TwinCAT 3 többi moduljával szemben a C++ modul működéséhez több alapvető elem szükséges:

- Windows Driver Kit: Kernel Driver íráshoz szükséges alapcsomag
- A fejlesztett driver digitális aláírása
- A futtató környezet digitális aláírása

A digitális aláírások megszerzéséhez magunknak kell generálni egy Windows teszt aláírást, és az operációs rendszert teszt üzemmódba kell léptetni, hogy ez az igazolás érvényes legyen, illetve egy online digitális aláírást kell kérvényezni a TwinCAT 3 szoftveren keresztül[4]. Ezeket hozzáadva a Visual Studio C++ projekthez, lehetőség van fejleszteni és futtatni a programot.



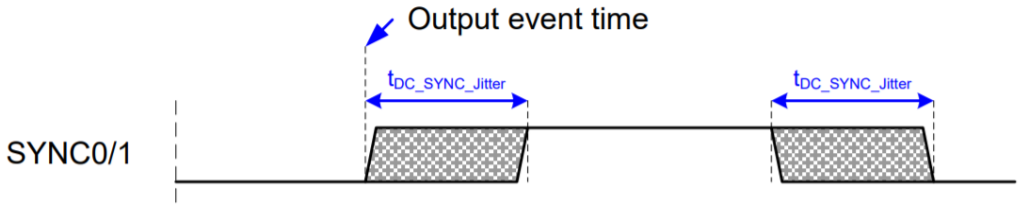
4.10. ábra. A TwinCAT C++ és a normál program futása[4]

IO

A TwinCAT 3-al realizált EtherCAT Master is képes a hozzácsatlakoztatott Slave-eket felderíteni. A kommunikáció megkezdése előtt szkenneli le a hálózatot a Master, és ezután állítja fel az egységek hálózati térképét. A TwinCAT adatbázisában sok Slave leíró fájlját tartalmazza, így automatikusan képes ezeket az eszközöket felkonfigurálni, új, általunk fejlesztett eszköznél a ki és bemeneteket illetve a Slave paramétereit mind a detektálás után kézileg kell megadni. Kész terméknél ezeket az adatokat meg kell határozni egy XML fájlban, aminek a segítségével a TwinCAT 3 képes automatikusan detektálni az ilyen eszközöket. A következőkben az általam használt Slave eszköz beállításainak lehetőségeit írom le.

Distributed Clock

Korábban említett elosztott óra megtalálható az ET1100-ban is, a működése nagyon fontos a megvalósított rendszeremben. Használatáról bővebben az 5. fejezetben fogok írni. A DC kimeneti jelei a SYNC jelek generálásának beállításait itt lehet elvégezni. A beállítható periódusidő és az jel pulzushossza is minimum 50 ns [12], de efelett bármekkora időtartomány beállítható. Az óra képes egyszeri impulzust is kiadni. A jel maximális jittere 15 ns(4.11.ábra).



4.11. ábra. A SYNC jel és a jittere[11]

PDO

Ahhoz, hogy az EtherCAT hálózat kommunikációja jól össze legyen hangolva a PC-n futó modulokkal (PLC, C++ projektek) szükségesek be és kimenetek, melyek a PC-t mint EtherCAT Master-t csatolják össze egy Slave egységgel. Ezen kívül segítenek a PC-n megvalósított projektek ciklikus időzítésének és az EtherCAT kommunikáció ciklusának szinkronizációjában. Ezeknek a ki és bemeneti változóknak az összesített neve a kimeneti vagy bemeneti Process Data Object vagyis PDO. Mindegyik változó négy részből áll: név, adattípus, az aktuális adat és az az EtherCAT Slave memória cím, amelyre hivatkozva el lehet érni őket. Ezeket meg lehet adni futás közben is, vagy akár a Slave-et leíró XML fájlban, így a Slave felderítésénél ezen változók azonnal inicializálódnak a Master oldalán is. A ki és bemeneti PDO-kat hozzá kell rendelni a Slave-ben található SyncManager-eknek(3.2 alfejezet), amik így ügyelnek a változók értékeinek folyamatos frissülésére, és az adatok pontos írására/olvasására. Ezen PDO-kban található változók megjelennek a TwinCAT 3 programban mint Slave-hez tartozó ki és bemenetek, amiket össze lehet kapcsolni a Master által futtatott program változóival, vagy akár másik Slave egység ki és bemeneteivel.

SDO

A Service Data Objet-ek(SDO) olyan üzenetek melyek nem időkritikusak, nem szükséges hozzájuk a valósidejűség. Ezek általában inicializációs beállítások, konfigurációs üzenetek, vagy állapotlekérdezések. Mivel nem időkritikusak, így nem a PDO-ban rögzülnek, hanem Mailbox módszerrel kerülnek kiküldésre és fogadásra.

5. fejezet

A rendszer fejlesztése

5.1. Az adatgyűjtő rendszer várt tulajdonságai

A célom egy olyan adatgyűjtő rendszer fejlesztése volt, amellyel egy adott jelet AD átalakítás segítségével, periodikus és ekvidisztáns mintavételezéssel lehet vizsgálni. Ezen kívül, ha több Slave egység is a rendszerre van csatlakoztatva, akkor mindegyik képes legyen ugyanabban az időpillanatokban mintavételezni. Ezen tulajdonságok az EtherCAT protokollnak hála elérhetőek, és felkonfigurálhatóak, így létrehozható egy olyan hálózat, amely képes több jelet ugyanazokban az időpontokban szinkron módon mintavételezni, és így a rendszer működését szabályozni. A teszt célkitűzése az volt, hogy az adatgyűjtő rendszer 1 kHz-es mintavétellel tudjon stabilan működni.

5.2. Kommunikáció az ET1100 ASIC és Piccolo mikrokontroller között

Összeköttetés megtervezése

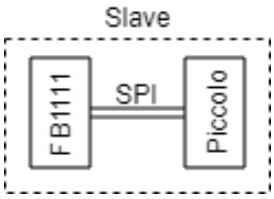
Legelőször az FB1111(4.1 fejezet) és a Piccolo mikrokontroller közötti kapcsolatot terveztem meg. Ehhez mindkét eszköz lábkiosztását meg kellett vizsgálnom, és a megfelelő lábakat összekötő kábelt terveznem[22]. Ebből két típus készült.

Tápkábel

A feladat elején csak ki akartam próbálni az FB111-et, meg akartam nézni, hogy a TwinCAT 3 szoftver képes-e az eszközt detektálni. Ezért lett egy tápkábel tervezve, amelyet labor tápról tudok meghajtani. Az ezzel való tesztelés nem volt sikeres, mivel a TwinCAT 3 felismerte az FB1111 eszközt, de tekintve, hogy az nem rendelkezett mikrokontrollerrel, így csupán az alap kommunikációs kalibrációkat lehetett beállítani rajta, minden magasabb szintű művelethez szükséges a megfelelő EtherCAT állapotba lépni, és ez csak a mikrokontrollerrel való párbeszéddel lehetséges.

SPI és tápkábel

Ezt az összeköttetést minden szükséges pin helyének figyelembe vételével terveztem, így sikerült a táp, SPI, SYNC és EEPROM jeleket is bekötnöm a mikrokontrollerbe.



5.1. ábra. Slave felépítése

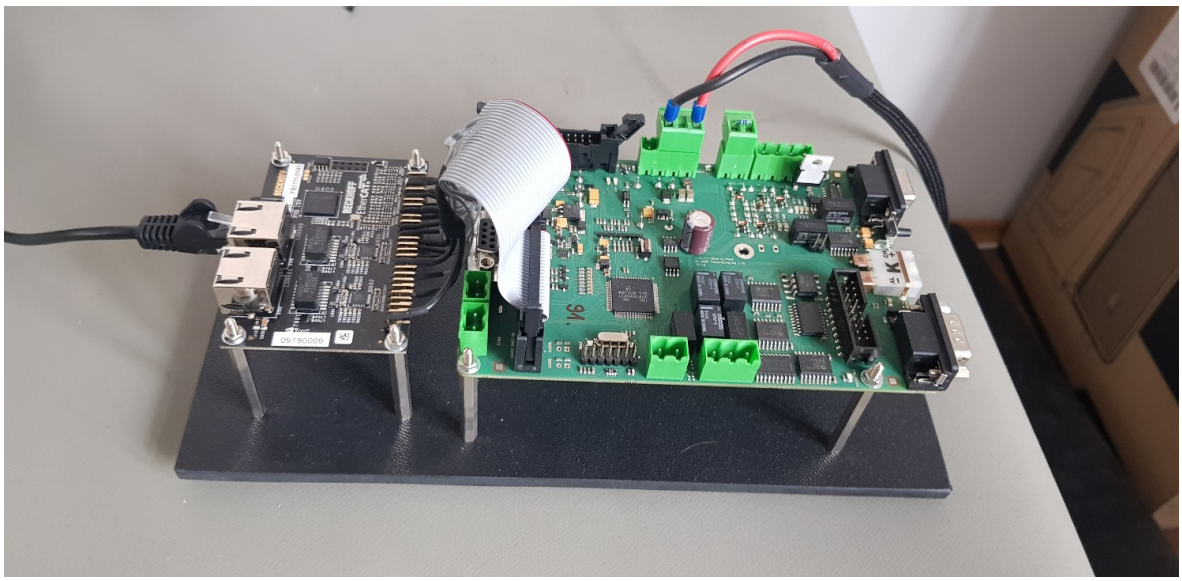
SPI

Legelőször a mikrokontrolleren található SPI modul inicializálása volt a feladatom. A mikrokontroller maximális órajele 90 MHz, de egy külső szoftver, mellyel futási időben tudom a változók értékét lekérni, vagy módosítani korlátozta ezt 80 MHz-re. Az SPI modul maximális sebessége függ az aktuális órajeltől, de ennek csak egy osztása jut el a modulba, mely jelen esetben 40 MHz, és a modulban történik még egy, előre beállított leosztás(divider)[23], így

$$maxbaudrate = orajel/divider$$

A beállítható legkisebb osztó a 4, így 10 Mbps vagyis 1.25 MBps a maximális sebesség. Ez a teszt alkalmazásra elegendő, de ha nagyobb sebességgel szeretnénk adatcserét folytatni, akkor szükséges jobb SPI modullal rendelkező mikrokontrollert keresni, vagy valamelyik másik ET1100 számára is elérhető kommunikációs formát választani. Az SPI szükséges beállításait az előző fejezetben már taglaltam, ugyan az a konfiguráció történt meg a mikrokontrolleren is.

Az előző fejezetben említett SPI címzési módszer tanulmányozása után megvalósítottam a szükséges SPI függvényeket, ezek 1, 2, 4 és 8 byte-ot írnak vagy olvasnak az ET1100 választott címéről. Az ET1100 regiszter elrendezése miatt ezen függvények minden kommunikációs szükségletet kielégítettek. Illetve a kommunikáció csak akkor kezdődik ha az EEPROM LOADED jel magas értéket vesz fel, ugyanis addig az ET1100 nincs fölkészülve az adatcserére.



5.2. ábra. A kész összeköttetés az FB1111(bal) és a mikrokontrollert tartalmazó lap(jobb) között

5.3. A mérésadatgyűjtő firmware Piccolo mikrokontrolleren

Az SPI kommunikáció leteszt elése után szükséges volt a kommunikációt és a mintavételezést lebonyolító firmware készítése is a Piccolo mikrokontrollerbe, amely képes folyamatos futás mellett kommunikációs hiba nélkül ellátni a feladatát.

Főbb függvények

Még a teljesen felépített működő program előtt szükséges volt kikeresnem és használnom az ET1100 PDI-n keresztüli inicializálásához, beállításához szükséges regiszterek nevét, címét és tartalmát. A program első része ezeket az alapvető regisztereket olvassa ki, és a beállításoknak megfelelően írja át.

Két fő regiszter található az ET1100-ban amelyek folyamatos kezelése elengedhetetlen egy stabil rendszer működéséhez: AL Control és AL Status[12] . Előbbit periodikusan olvasni kell, és ezt állítva jelzi a Master, hogy a Slave-nek épp milyen állapotba kell átlépnie. Utóbbi pedig az aktuális állapotot tartalmazza, illetve jelzi, ha bármilyen hiba lépett fel az állapotváltásnál. Így ezen regisztereket a futás folyamán folyamatosan, 50 ms-onként kérdezem le. Tesztjeim során találtam ezt az értéket, eléggé folyamatos a lekérdezés, hogy a lehető legkönnyedebb legyen az állapotváltás, és mégse terhelje túl a folyamatos lekérdezés az SPI kommunikációs csatornát. Ezeken kívül a SyncManager-ekhez tartozó regisztereket és a SYNC jelekhez tartozóakat kell beállítani és kezelni, ezekkel már működő rendszer épülhet fel.

EtherCAT állapotgép megvalósítása

Egy olyan állapotgépet kellett létrehoznom, mely folyamatosan, aktívan képes kezelni a Master által kérvényezett állapotokat, mindezt úgy, hogy a Slave konfigurálásához szükséges regiszterek is megfelelően be legyenek állítva. A program vázát egy switch szerkezet alkotja, mely az EtherCAT állapotokat reprezentálja. Szükséges teendők az állapotváltásoknál:

- INIT->PREOP: Mailbox kommunikáció engedélyezése, ha kell DC beállítása
- PREOP->SAFEOP: PDO IN engedélyezése
- SAFEOP->OP: PDO OUT engedélyezése

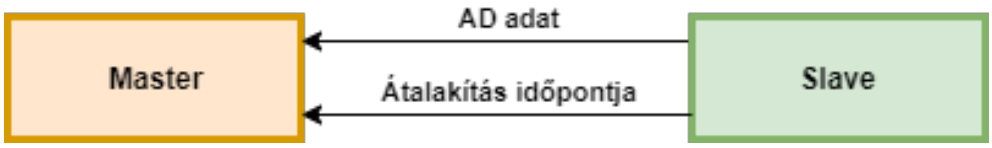
Az állapot visszaváltásoknál a korábban engedélyezett funkciókat tiltani kell.

Szinkronizációs jelek

A mikrokontroller AD átalakítóját olyan módba állítottam, mely minden egyes külső megszakításra végez egy AD átalakítást. A külső megszakítást az ET1100 DC SYNC0 jele biztosítja. Ez azért fontos, mivel az ET1100 órajele szinkronizált a Master-évl, emiatt ahhoz képest pontosabb mint a Piccolo belső órája.

Periodikus jelek

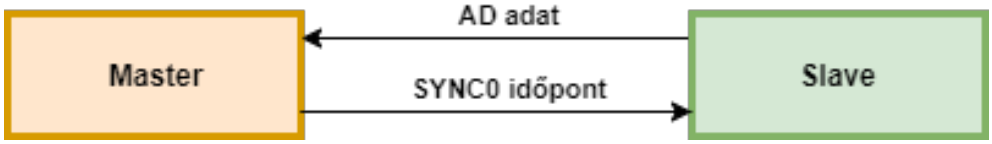
Az ET1100 beállítható arra, hogy bizonyos időközönként adjon impulzust a SYNC csatornákon, ezt kihasználva mindig ugyanakkora időközönként tudom mintavételezni a kérdéses jel feszültségét. Ennek a módnak két változója van, mindkettőt a Slave küldi a Masternek, az egyik maga az átalakított adat, a másik pedig az átalakítás időpontja.



5.3. ábra. A változók a ciklikus jelnél

Egyszeri pulzus

A mikrokontroller és az ET1100 kommunikációja révén nano szekundum felosztással beállítható az az időpont, amikor az ET1100 jelezzen a SYNC csatornán. Ebben az esetben ez egy egyszeri pulzus és újra kell kalibrálni az időpontot, ha megint használni akarjuk, mivel a korábban beállított időpont már letelt, a 64 bites belső óra áthaladt rajta. Ennek a módnak is több előnye van, például, ha speciális időpontokban akarjuk mintavételezni a jelet. Ennél a beállításnál a Master oldala küld egy 64 bites időpontot, ami megszabja, hogy mikor keletkezzen a pulzus. Így a kommunikációnak két fő változója van, ez az időpont, illetve a megszakítás után keletkezett AD adat, amit visszakap a Master.



5.4. ábra. A változók az egyszeri pulzus jelnél

Megszakítások

A Piccolo mikrokontrollerbe érkező periodikus jelek mint külső megszakítások érkeznek, így beindítják az AD átalakítást. Ezen megszakítások alatt az AD átalakítás által létrehozott értékeket a mikrokontroller azonnal átküldi az ET1100-nak, amely ezt az információt továbbítja a Master PC-nek.

PDO

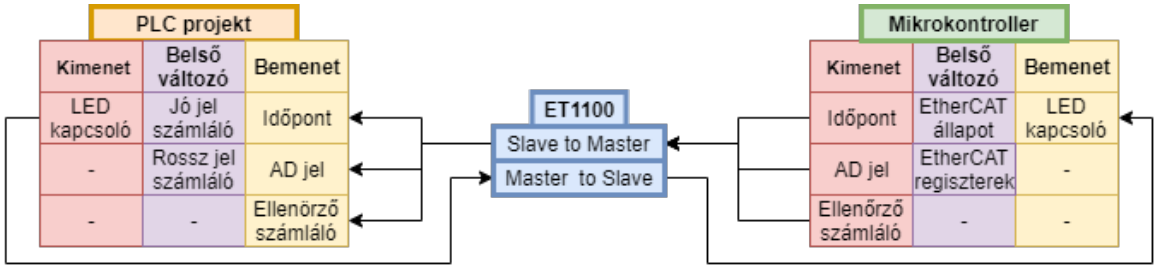
Az alkalmazás nem csak az AD átalakítás eredményét küldte ki, hanem több változó aktuális értékét is, továbbá több változót is fogadott a Master oldalról, amelyek beérkező értékei valamilyen teszt funkciót valósítottak meg. Például egy LED nevű egy bites beérkező változó egybe billentésével, a mikrokontroller boardon található piros LED felkapcsolódik. Mivel a fontosabb AD érték kiküldése megszakításkezelésben van, de az általánosabb PDO jelek nem, így egy olyan időbeosztást választottam, mely szerint a PDO jelek 10 ms-ként frissülnek, az AD jel pedig 1 ms(mivel a SYNC jeleknek ez a periódus ideje). A PDO jelek kiírása közben tiltott a megszakítás, így nem történhet meg az, hogy az általános PDO változók írása közben a megszakítás miatt egy új SPI kommunikáció ékelődik be.

5.4. TwinCAT 3 projekt készítése, beállítása

PLC projekt

A TwinCAT 3 szoftver főként PLC programozásra és írásra való, így egy Structured Text nyelven írt program készítését láttam a legtermészetesebbnek. A főprogramom igen

egyszerű volt, a bejövő adat pontosságát akartam megfigyelni. A Slave oldalról 3 féle adat jött: Az AD átalakítás értéke, az időpontja és egy számláló. A számláló azért fontos, mivel minden egyes AD átalakításnál nő az értéke egyel, a Master ezt az értéket leellenőrzi, és ha egyel nagyobb mint a korábban beérkezett, akkor biztos, hogy az AD átalakítás adata is már a következő ciklus adata, nem történt semmilyen kommunikációs hiba, vagy rendellenesség a ciklikusságban. A PLC projekten belül számoltam a helyes, konzisztens adatok számát és a helytelen adatok számát. Megfigyelhető volt, hogy az elosztott órák kalibrálásával a két ellenőrző számláló értéke hogyan változott. A megfelelő beállításokkal 1.5 órás tesztüzemen hagyva sem történt hiba.



5.5. ábra. A rendszer alapvető kapcsolata

Slave egység beállítása

A rendszer felállításához nem elég a Slave-et a Master-hez csatlakoztatni, sok olyan beállítás van, melyet a Master-nek kell elvégezni a Slave-en ahhoz, hogy működőképes kapcsolat alakuljon ki.

EEPROM

A főbb hardver beállítások mind az EEPROM írásával történnek meg, ehhez először is át kellett írni az FB1111-0141 gyári XML fájlját, a saját igények szerint. Ehhez tudni kell a PDO-k típusát, nevét és címét. Még szükséges egy 12 byte-os EEPROM konfigurációs beállítás, melyet az EtherCAT Tehcnology Group által szolgáltatott táblázat alapján kell megalkotni. Az én beállításaim alapján a Slave egység SPI-al kommunikál 3. üzemmódban, normál mintavétellel. A SYNC jelek impulzushossza 50 μs. Bár a legkisebb beállítható impulzushossz 10 ns , de mivel a mérés célja az volt, hogy 1ms-os periódusidővel legyen mintavétel, így az 50 μs, elegendő, bőven elfér 1 ms-os tartományon belül.

PDO

A PDO-k beállítására két módszer is létezik, az elsőnél a rendszer felépítésénél minden egyes újrakonfigurálásnál a TwinCAT 3 szoftveren belül megadtam a változókat, azok helyét és hozzárendeltem őket a ki vagy bemeneti SyncManager-ekhez, de a fejlesztés vége felé elkészítettem egy olyan Slave leíró XML-fájlt, melyről a TwinCAT nem csak az EEPROM adatokat tudja helyesen beolvasni, de a változók adatait is, így egy ”plug and play” típusú rendszert hoztam létre ezeknek a Slave-eknek.

SDO

A jelenlegi teszt rendszerem nem használ még SDO-t, vagyis üzeneteket, nem szükséges neki futás közbeni konfiguráció. Így ez a funkció nincs implementálva.

Distributed Clock

A legtöbb konfigurációs lehetőség az elosztott órákban van. Minden Slave óráját be lehet konfigurálni a TwinCAT 3 szoftveren keresztül, az órajel ciklusától kezdve, a SYNC jelek ciklusáig. Ha kell az adott jel fázisát is el lehet tolni, illetve beállítható, hogy a rendszer folyamatosan érzékelje az apró fázisváltozásokat, és küszöbölje ki ezeket. Az én általam megvalósított Slave egységeknek 1 ms-os SYNC0 ciklusa van, az egység és Master közötti fázistolás pedig a rendszer által figyelt és konfigurált.

Kommunikáció időzítése

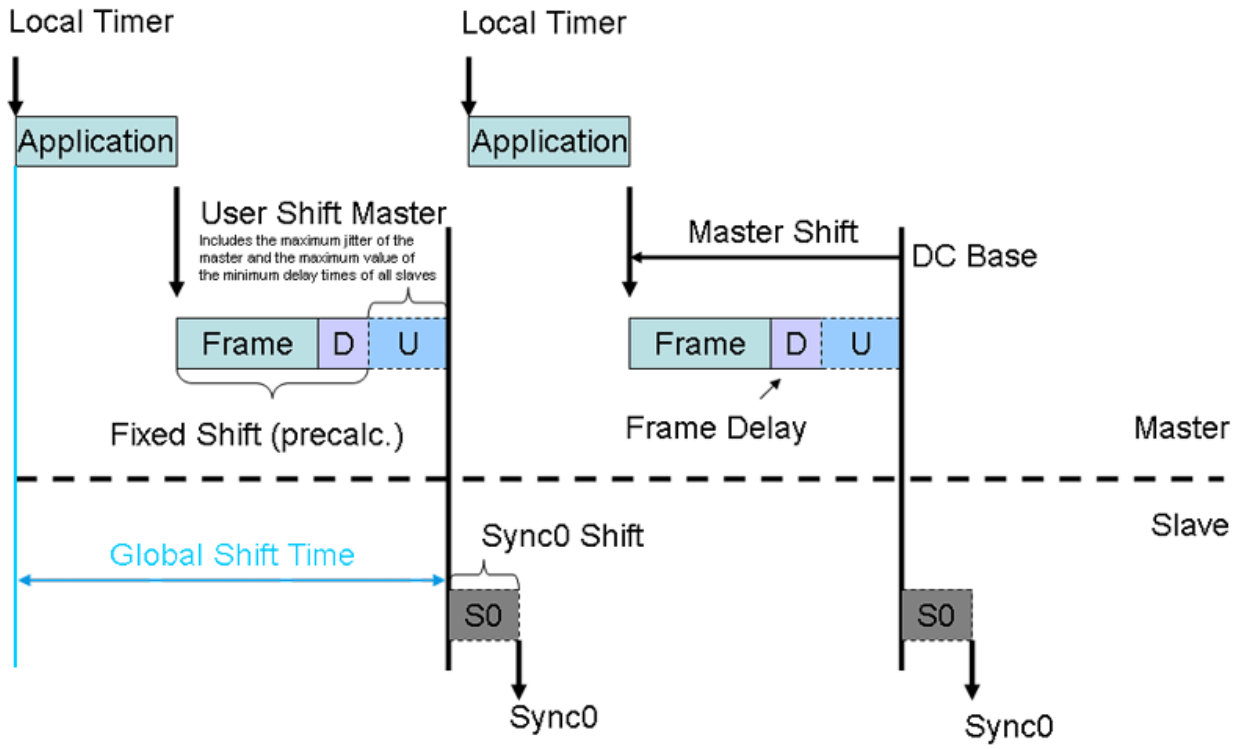
A Slave egységek belső órája beállítható a hálózat referenciaórájának, így minden egység ahhoz igazodik. Ilyenkor a Master is ehhez az órához igazodik, és ehhez képest igazítja a saját órajelét. Általában a Master-hez legközelebb álló Slave a referenciaóra.

Mikrokontrolleren keresztüli beállítások

Lehetőség van a mikrokontrolleren keresztül is beállítani az elosztott órákat, ezt az egyszeri pulzusú SYNC módban teszteltem. Ilyenkor a Master nem szól bele az elosztott órák működésébe.

Kommunikáció időzítése

Sokáig kérdés volt, hogy az EtherCAT rendszerben hogyan lehet meghatározni a PC-n lefutó taskok, az EtherCAT kommunikáció és a Slave külső megszakításai által generált adat időpontjának sorrendjét. A hálózatban a kommunikáció és a PC-n futó PLC task felcserélhető sorrendű, de a legmegbízhatóbb működést akkor kapjuk, ha hagyjuk hogy a Master rendszerezze a sorrendet. A 5.6 ábrán látható, hogy milyen felosztásban következnek egy determinisztikus rendszernél az adott elemek.



5.6. ábra. A Master és Slave ciklikus futásának összehangolása[5]

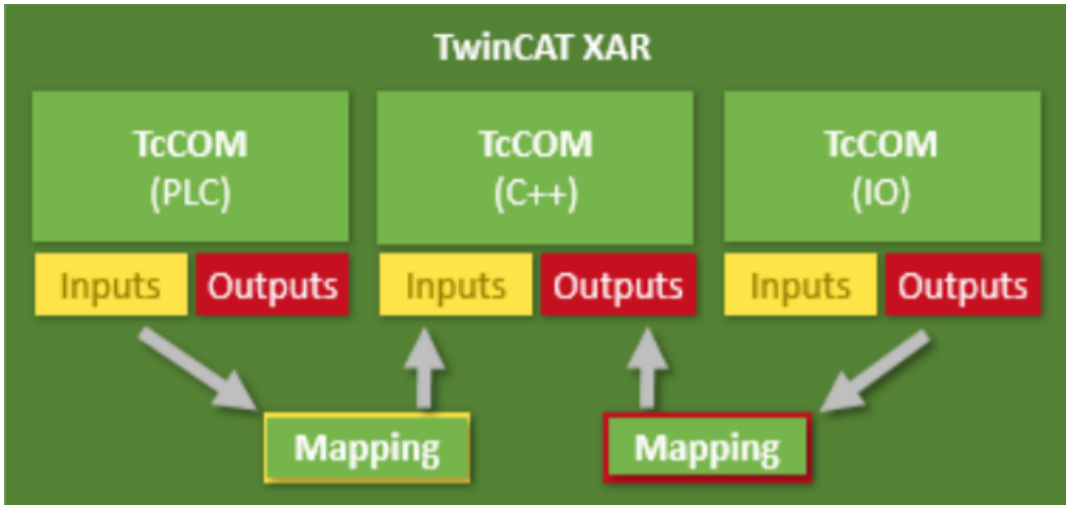
Adatok gyűjtése és lementése

Az adatok gyűjtésére a már korábban említett PLC alapprogramot használtam, mely leellenőrzi, hogy a beérkezett adat jó-e. Emellett egy másik modult készítettem, ami csak az adat összegzéséért és írásáért felelős. A beérkezett AD adatot és időpontot átalakítom sztringekké, majd ezeket megfelelő módon konkatenálva kapok egy végső sztringet, mely az egy ciklusban beérkezett adatokat tartalmazza. A task gyorsaságától és az IO kommunikációtól függően eltérő nagyságú halmazokat készítek ezekből a sztringekből, melyek ha elérték az adott méretet, ami általában 4-8 ciklus adata, akkor az almodul által kiíródnak egy szöveges dokumentumba.

Ha a kiírás modulja és az adatelemző modul is ugyanahhoz a taskhoz volt kötve, mely egy saját processzor maghoz volt rendelve, akkor előfordulhatott, hogy a kiírás lefoglalta a processzor magot, így kimaradt egy ciklusnyi adat. Ezzel az akadállyal találkozva beállítottam, hogy mind a kiírás PLC modulja, mint az adatgyűjtő PLC modul külön processzormaghoz legyenek rendelve, így egymástól függetlenül futva nem zavarják a másikat.

C++ projekt

Létrehoztam egy C++ projektet is, hogy megvizsgáljam a fájlba való kiíratást ebben a környezetben is. A projekt építéséhez és futásához igen sok előkövetelmény létezik, kezdve a digitális aláírásoktól egészen a Windows test üzemmódban való futtatásáig, de ezeket megoldva már egy futtatható modul készült el. A modulban van egy bizonyos függvény, amelyben meg lehet adni azokat az utasításokat, melyeket ciklikusan szeretnénk megvalósítani. A modul létrehozása után a modult össze kell kötni egy taskal, mely folyamatosan meghívja minden egyes ciklus kezdeténél ezt a bizonyos függvényt. A PLC és C++ taskokat a TwinCAT össze tudja kötni, így az adatvételezés ugyanúgy futhat a PLC modulon.



5.7. ábra. A TwinCAT modulok kapcsolata[4]

A fejlesztés és tesztelés alatt úgy éreztem, hogy bár a C++ alatt megvalósítható rendszer nagy előny, mégis véleményem szerint ez a megoldás kissé kiforratlan, nem triviális a használata, ezért esetleg egy külön projekt keretein beleül lehetne rá elegendő időt szánni. Így a megbízhatóan működő PLC-s adatkiírást választottam a rendszerem adatmentési megoldásaként.

Több Slave egység csatlakoztatása

Kiegészítésként mind a két elosztott óra üzemmódot és ezzel együtt mintavételi módot is kipróbáltam két darab FB1111-Piccolo EtherCAT Slave-el. Összeköttetésük egyszerű volt, mivel az FB1111 két Ethernet portal rendelkezik. A Master-ből vezet egy Ethernet kábel az egyik Slave-be, annak másik portjából pedig még egy Ethernet kábel az új egységbe. Működésükhöz csupán fel kellett programoznom az új Slave eszközt, és az készen állt a kommunikációra és a mintavételre.

PLC projekt

Mivel új adatok is érkeznek a Master felé, ezért szükséges volt a projektet is módosítani, az új változókat beiktatni. Tekintve, hogy az adott időpontban mintavételezett jel adatai mind a két Slave-ről kiküldésre kerültek, a kiíró modulban előállított sztring már mindkét adatot tartalmazta, a mintavétel időpontjával egyetemben.

Referenciaóra

Csak egy referenciaóra lehet a hálózatban, erre a Master-hez közelebb csatlakoztatott Slave-et jelöltem ki. A másik SYNC egysége is ugyanúgy üzemel, de óraszinkronizációkor a referenciaórás Slave-hez igazodik.

5.8. ábra. A működő EtherCAT hálózat két Slave egységgel

6. fejezet

Mérésadatgyűjtő és vezérlő rendszer tulajdonságainak mérése, kiértékelése

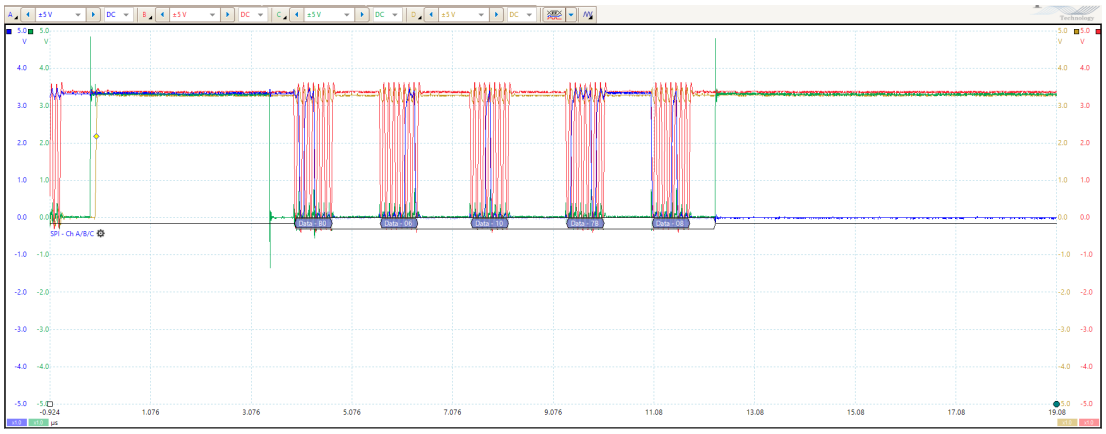
A kész rendszert többféle módon is mértem, az SPI kommunikációtól kezdve a SYNC jeleken át. Az AD átalakításhoz szükséges jelként szinusz jelet választottam, mivel ennél a jelnél az ekvidisztáns mintavételezés hiánya szembetűnőbb, jobban leellenőrizhető, mint például egy négyszögjelnél. Piccolo mikrokontroller AD átalakítója 12 bites, így a kapott adatok 0-4095-ig vehetnek fel értéket. A mérésekhez 4 csatornás Pico Scope 3403D MSO típusú oszcilloszkópot használtam[17], illetve innen generáltam egy 1 Hz-es szinusz jelet is, melyet mintavételeztek a mikrokontrollerek. Azért csak ekkora frekvenciájú jelet mértem, mivel szerettem volna egy periódust minél több adatból visszaállítani és az 1 Hz-es jel egy periódusából ezer darab mintát tudtam venni, amelyet már megfelelőnek gondoltam. Az oszcilloszkóp képes volt dekódolni az SPI jelet, így ezt tesztelésre és ellenőrzésre is használtam.

SPI

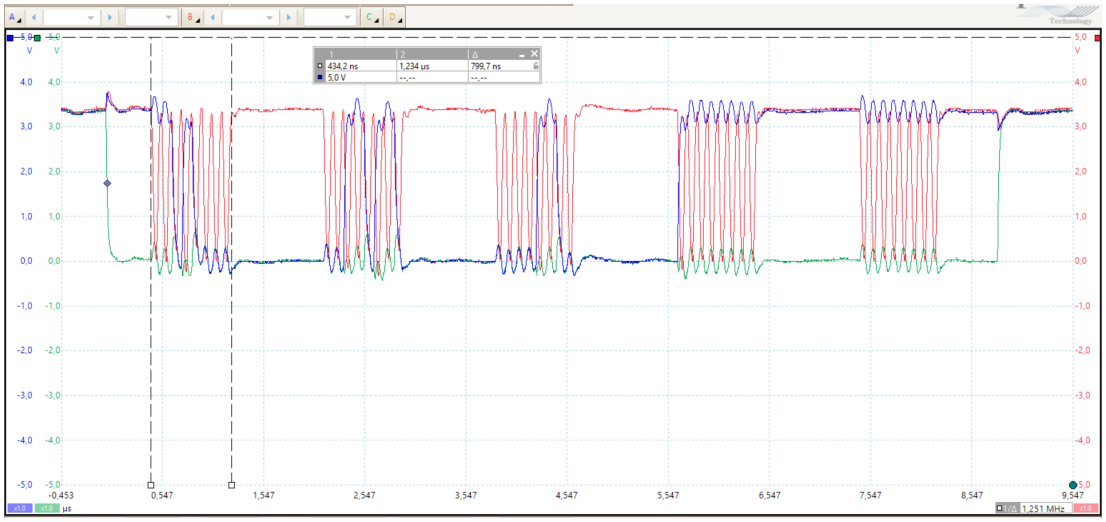
A rendszer tulajdonságainak mérése előtt az SPI kommunikációt is megfigyeltem, az elméleti és gyakorlati gyorsaságának összevetése miatt, illetve a kimenő adatok helyességének leellenőrzése miatt. AZ SPI ábrák jeleinek feloldása:

- Sárga jel: Külső megszakítás(SYNC0)
- Zöld jel: Chip Select
- Piros jel: SPI órajel
- Kék jel: SPI adat

A 6.2 ábrán látható egy kiküldött byte, melynek sebessége 799.7 ns, így a tényleges sebesség 1.251 MBps, amely egyezik a várt 1.25 MBps-el



6.1. ábra. A külső megszakítás(SYNC0) utáni adat kiküldése SPI-on keresztül



6.2. ábra. Egy bájt kiküldésének ideje

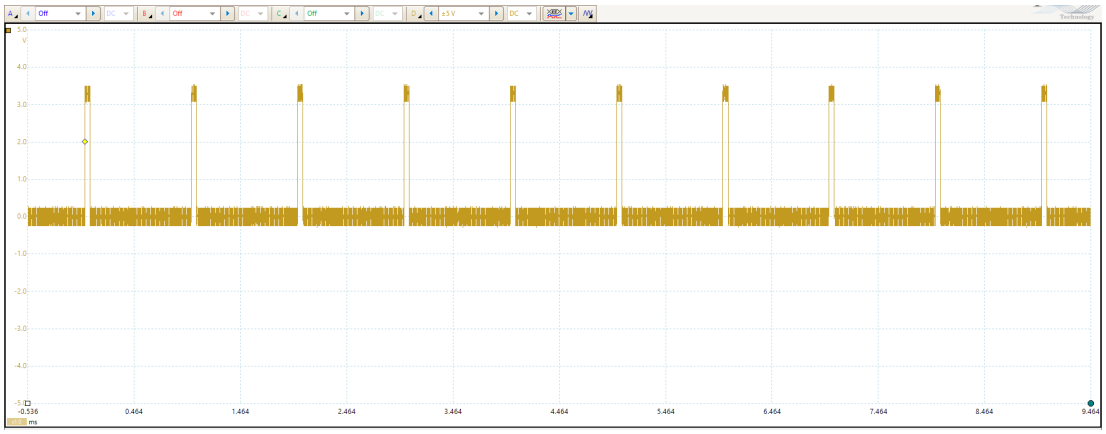
6.1. Egy Slave-es elrendezés SYNC jele

6.1.1. Periodikus jel

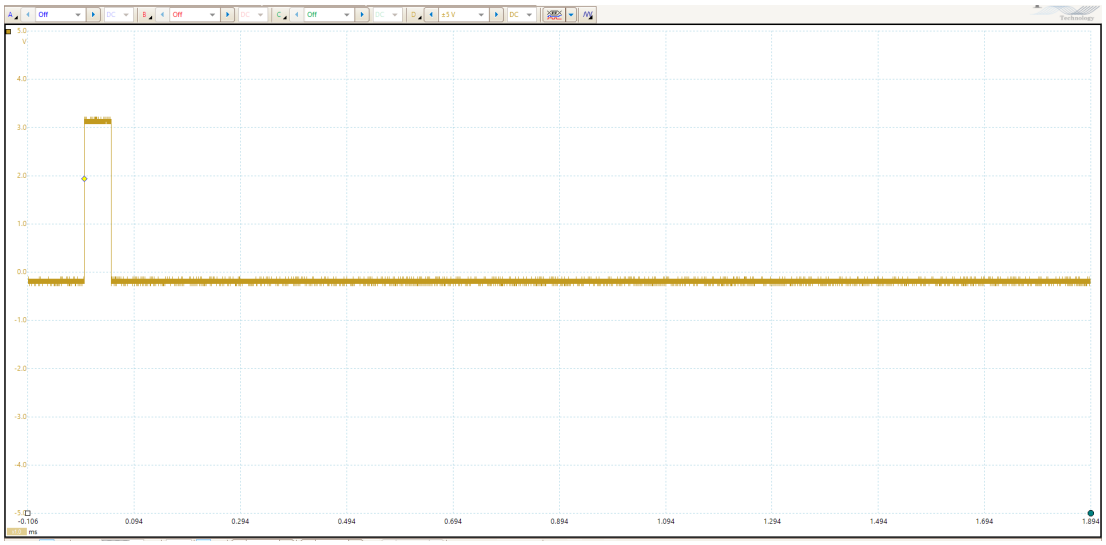
A rendszer összehangolásának a legfontosabb eleme a jól kalibrált SYNC jel, mivel a mintavételt, és ezzel a Master-el való kommunikációt és a Master-en futó modulokat is szabályozza. A rendszer teszteléséhez 1 ms-os SYNC periódus időt állítottam be. A 6.3 ábrán látható a periodikus SYNC0 jel.

Egyszeri pulzus

Az egyszeri pulzusú üzemmódra állított SYNC0 jel csak egy előre beállított időpontban lép működésbe. Viselkedése a periodicitáson kívül azonos az előző jellel.



6.3. ábra. A periodikus SYNC0 jel



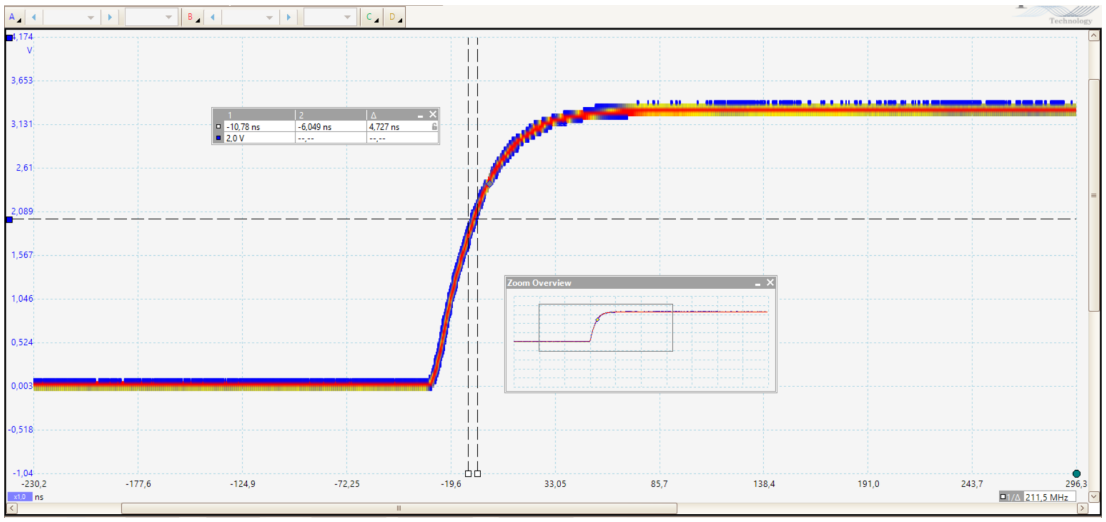
6.4. ábra. Az egyszeri impulzusra állított SYNC0 jel

6.1.2. Jitter

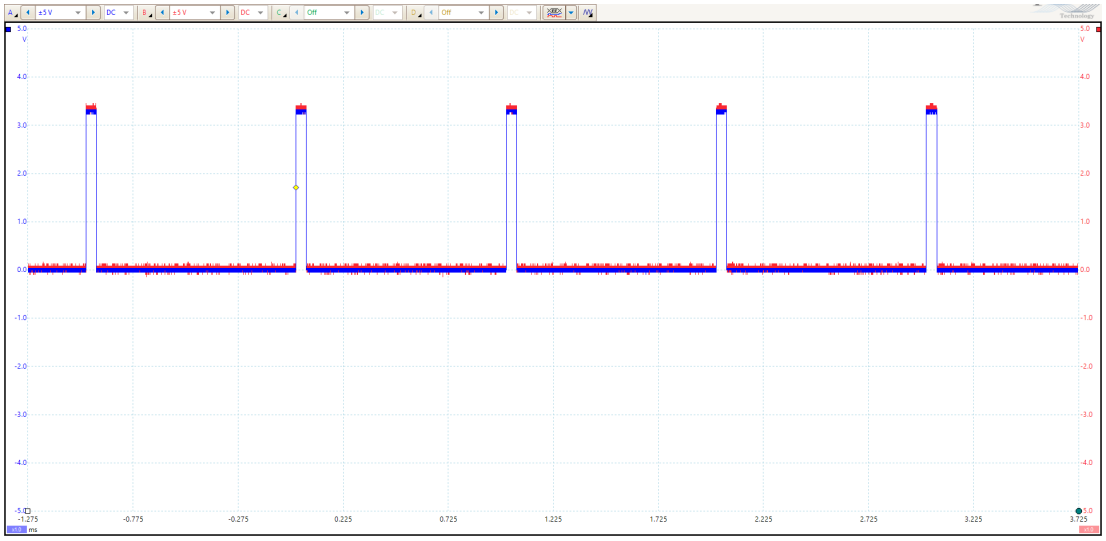
A jitter méréséhez az oszcilloszkóp mérőfejét rákötöttem a rendszerben lévő egyetlen Slave SYNC0 jelére, és a triggernek mindig az előző periódus SYNC0 felfutó élét választottam. Így volt két felfutó él, amelyek közül az elsőre triggereltem, eközben a másodikat nagyítva az oszcilloszkópon megjelenítettem. A megengedett jitter maximálisan 15 ns, a méréseim során ennél alacsonyabb értéket, 4.727 ns-ot mértem. A mérés helye a 6.5 ábrán 2 V feszültségnél történik, mivel a mikrokontroller bemenete a logikai magas értéket ekkora feszültségtől érzékeli.

6.2. Két Slave-es elrendezés SYNC jelei

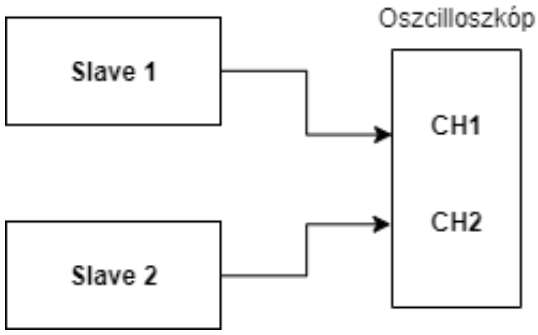
Két Slave jeleit is lemértem, hiszen nagyon fontos volt az egységek SYNC0 jeleinek szinkronitása(6.6 ábra).



6.5. ábra. SYNC jel jittere



6.6. ábra. Különböző Slave-ek SYNC0 jelei

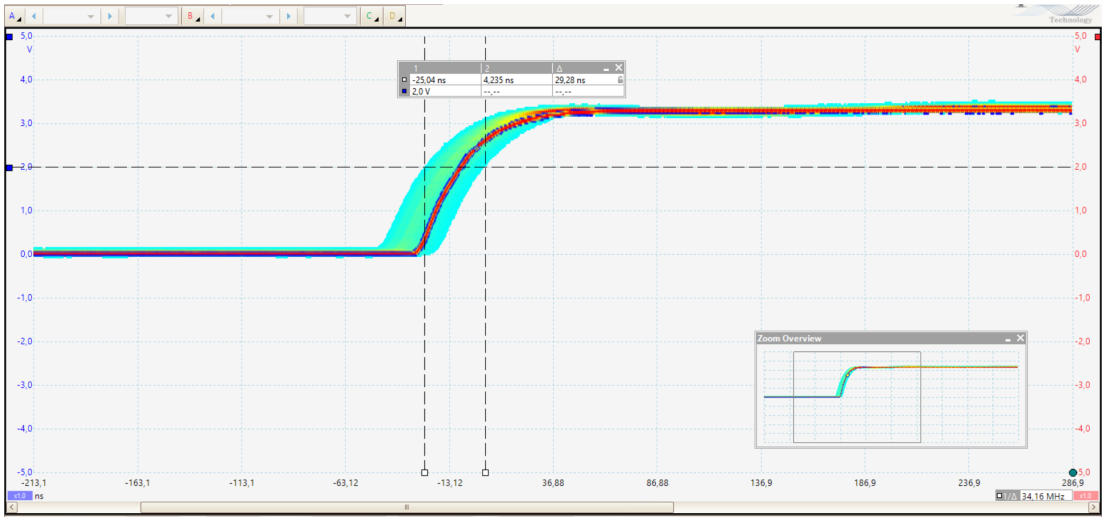


6.7. ábra. Két SYNC jitter mérésének blokkdiagrammja

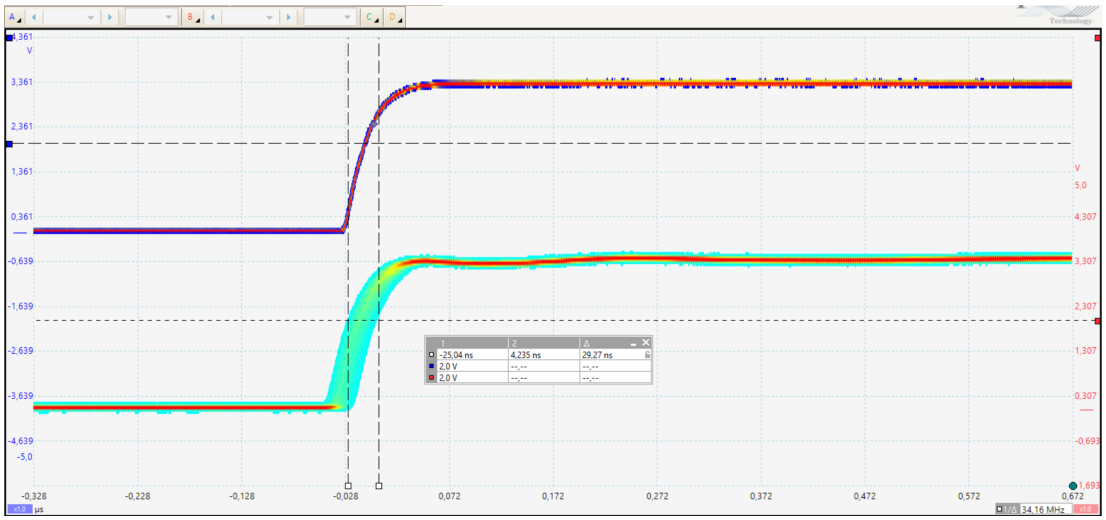
6.2.1. Jitter

A méréshez az oszcilloszkóp egy-egy mérőfejét rákötöttem a hálózatban lévő két Slave egység SYNC0 jelére(6.7 ábra). A választott trigger itt az egyik Slave SYNC0 jele, ehhez képest vizsgáltam a két Slave-es rendszer jitterét. A 6.8 ábrán látható jelek jitterének

összessége. A két jel maximális jittere 30 ns körüli, ami, ha a két jel legnagyobb elfogadható esetét tekintjük, ahol mindkettő jittere 15 ns, akkor is még elfogadható tartományban mozog, de a pontos szinkronizáció miatt ez csak egy határérték, a perzisztencián látható, hogy az átlag jitter ennél jóval kisebb. A két jel jittere külön megjelenítve a 6.9-as ábrán látható. A triggerelés módjának következtében az oszcilloszkóppal mért jitter kisebb az alsó jelnél, míg a felsőnél nagyobb.



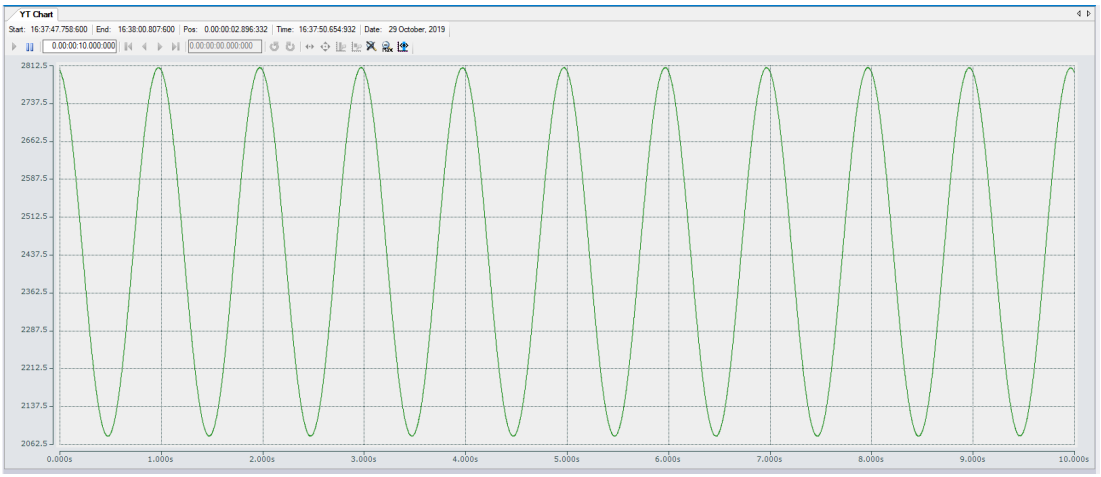
6.8. ábra. Két SYNC jel összesített jittere



6.9. ábra. Két SYNC jel jittere külön

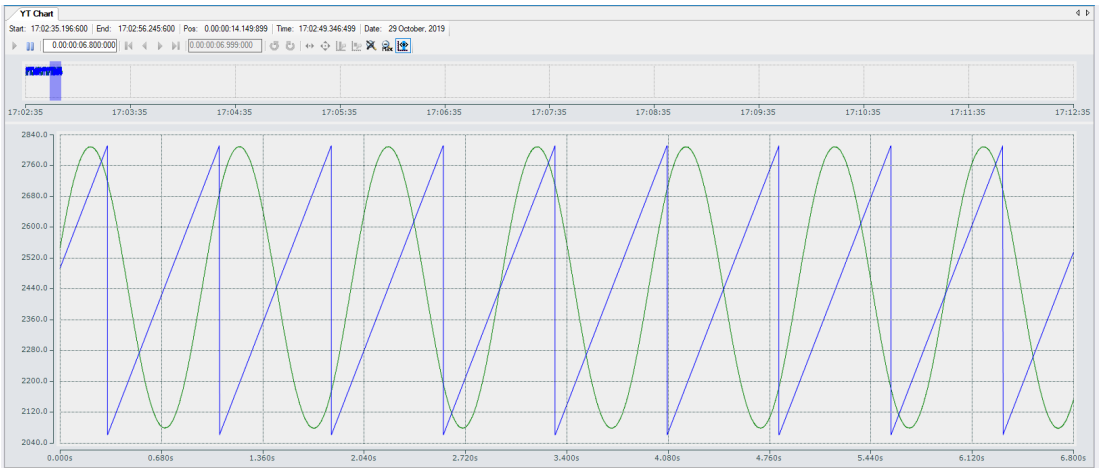
6.3. Szinusz jel mérése

Mivel az adatvételezés periódusideje 1 ms, így az 1 Hz-es szinusz jelet veszteség nélkül tudom mintavételezni. A TwinCAT 3 egyik moduljának segítségével, oszcilloszkópszerűen, élőben tudtam monitorozni a bejövő adat változását, így már a PLC projekten belül ki tudtam rajzolni a mért szinusz jelet. A szinuszjel amplitúdóját 0.5 V-ra és az offset értékét 1 V-ra állítottam. Erre azért volt szükség, mivel a mikrokontroller lábai a negatív feszültségtartományt kizárják, így a jel sem vehetett fel ilyen tartományból értéket.



6.10. ábra. Mintavételezett 1 Hz-es szinusz jel a TwinCAT-ben megjelenítve

Az 5.4-es fejezetben említettem, hogy egy számlálót is kiküldtem az átalakított adat mellett. Ennek további szerepe is volt, mivel a TwinCAT belső oszcilloszkópjában megjelenítve le tudtam ellenőrizni, hogy olyan értékeknél ahol két mintavételezés között a szinusz jel értéke nem változik, a periodikus mintavételezés-e a hibás, vagy csak az AD átalakító ad abban a speciális esetben ugyanolyan értéket. Mivel minden új szinusz értéknél láttam a számláló egyel való növekedését, így biztos voltam benne, hogy a mintavételezés megfelelően működik. A 6.11 ábrán látható számláló folyamatosan túlcsordul egy bizonyos érték

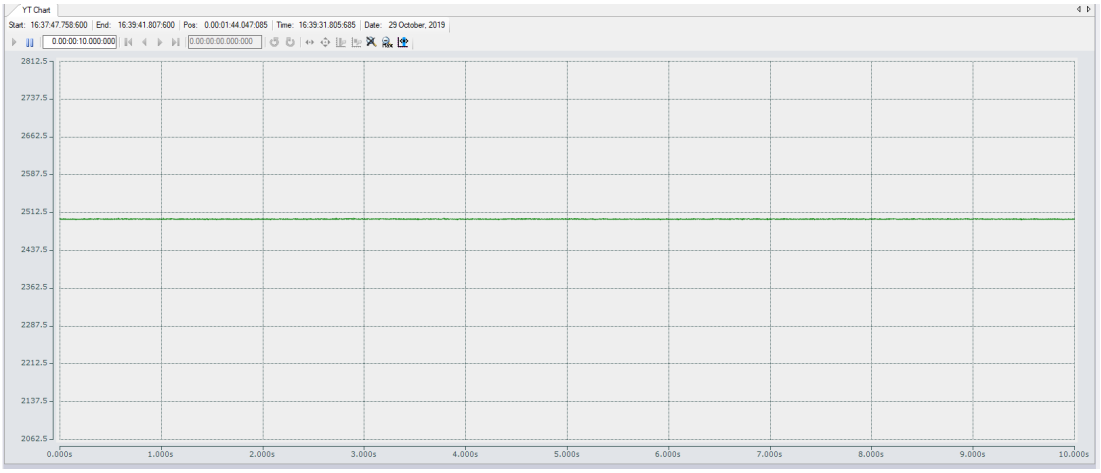


6.11. ábra. Mintavételezett 1 Hz-es szinusz jel a TwinCAT-ben megjelenítve

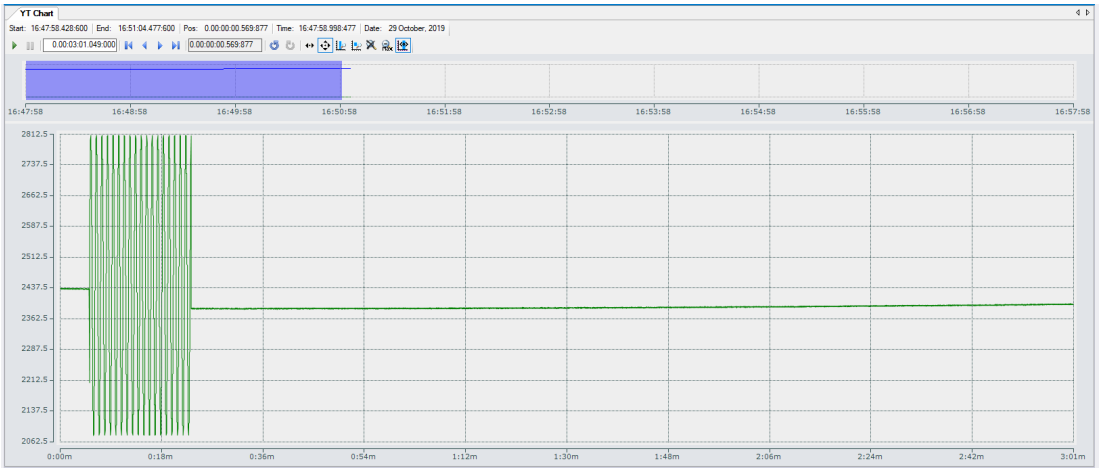
után, így mindig egy értéktartományban marad a szinusz jellel, így követhető a számláló és a szinusz változása is egy ablakban.

1 kHz-es szinusz vizsgálata

Érdekességképpen az 1 kHz-es mintavételi frekvenciával 1 kHz-es jelet vizsgáltam. A várakozásom az volt, hogy ugyanazt az adatot kapom minden egyes mintavételnél, hiszen a két jel szinkronban van, ezáltal minden periódusnak mindig ugyanazon pontját méri az AD. Ez lényegében beigazolódott, de hosszabb várakozás után látszott, hogy a bejövő értékek szép lassan eltérnek egymástól, mivel az EtherCAT hálózat és az oszcilloszkóp órajele nem teljesen azonos, a jel fázisa kissé csúszik, így kirajzolódik az 1 kHz-es szinusz. Ennek a megjelenítésnek a mértéke nagyon apró és a rendszernek percekig kell futnia, míg szemmel feltűnő lesz az eltérés.



6.12. ábra. Az 1 kHz-es jel mintavétele



6.13. ábra. Az 1 Hz-es jelből 1 kHz-es jelre való áttérés, és az órajel eltérés miatt megjelenő szinusz kirajzolódása

7. fejezet

Összefoglalás

A célom egy determinisztikus adatgyűjtő és vezérlő rendszer megépítése volt, amely valósidőben képes 1 ms-os gyakorisággal szinkron mintavételezni egy bejövő jelet, az így mért adatot elraktározni és kijelezni.

A prototípus rendszer felépítése a megfelelő kommunikációs protokoll megtalálásával kezdődött. Az ehhez szükséges alkotóegységek (TwinCAT 3 Master, FB1111-0141 és Piccolo mikrokontroller) kiválasztásával, és azok beüzemelésével jött létre a prototípus.

A SYNC jelek periodikusságának és a jitterének mérés után azt tapasztaltam, hogy a rendszer megfelelően teljesít, egy Slave-es elrendezésben a jitter maximális 15 ns-os értékét nem haladja meg (4.727 ns). Két Slave-es felállásban pedig az egymás jeléhez mért jitter is megfelelő (29.28 ns). Így a rendszer az elvártaknak minden szempontból eleget tesz. A periodikus ekvidisztáns mintavételezés megvalósítható volt. Tesztelésként egy szinusz jel mérését valósítottam meg. A jel mintavételezése a tapasztaltok alapján megfelelő minőségű volt. Az adatgyűjtést és kijelzést is leteszteltem, a rendszer hosszabb futása után sem történt hiba. Ezáltal a rendszeren végzett mérésekkel és tesztekkel alátámasztottam, hogy az adatgyűjtésre vonatkozó követelményeknek megfelel a rendszer.

A vezérlési tulajdonságait is kipróbáltam, a mikrokontroller boardon található LED-et kétféleképpen is tudtam villogtatni. Az egyik módszerben a Master által küldött PDO segítségével, míg a másikban a Master által futás közben felkonfigurált ET1100 SYNC0 egyszeri pulzusát használva.

A protokoll megismerése és a projektek vázainak kidolgozása bár hosszú folyamat, de ezek után újabb elemek hozzáadása és a rendszer konfigurációja már gyorsan megoldható. A mérésadatgyűjtő és vezérlő rendszerem sikeresen használja az EtherCAT által nyújtott kommunikációs és applikációs lehetőségeket, a Slave egységek a pontosan az elvárt időben történő a mintavétellel mért feszültséget helyes értékekkel továbbítják a Master számára, mely ezeket az adatokat elraktározza, illetve vizuális formában kijelzi.

A rendszer a jövőben több irányban továbbfejleszthető. Egyike lehet ezeknek az irányoknak az EtherCAT által nyújtott fejleszthetőségi lehetőségek alkalmazása (3.5.fejezet). Emellett az ET1100-at és a Piccolo mikrokontrollert is fel lehetne helyezni egy boardra, így létrejönne egy kompakt Slave. Másik irány lehet az EtherCAT Master alkalmazás mikrokontrollerre, vagy egykártyás számítógépre való fejlesztése. Végül, mivel ez is egyike volt a céloknak, ezzel a prototípussal, mint kiindulási alappal, új bonyolultabb rendszert lehet fejleszteni.

Köszönetnyilvánítás

Köszönöm dr. Orosz György tanulmányaiban való folyamatos segítségét, illetve hasznos tanácsait, melyekkel hozzájárult a szakdolgozatom elkészültéhez.

Külön köszönöm a ProDSP Technologies Zrt.-nek ezt a kiváló lehetőséget a tanulásra és fejlődésre. Köszönöm Csengeri Bálint külső konzulensemnek a türelmét és szaktudását a kérdéseim megválaszolásában és a fellépő problémák megoldásában.

Irodalomjegyzék

[1] Alexander Hanzlik: A case study of clock synchronization in flexray (2019.11.29). https://pdfs.semanticscholar.org/5f47/869a5155af44b44edba862034f0cfc58f3fe.pdf.

[2] Beckhoff Automation GmbH and Co. KG: Fb1111 datasheet (2019.11.29). https://download.beckhoff.com/download/Document/io/ethercat-development-products/beckhoff_fb1111-014x_v22.pdf.

[3] Beckhoff Automation GmbH and Co. KG: Twincat 3 (2019.11.29). http://download.beckhoff.com/download/Document/catalog/Beckhoff_TwinCAT3_042012_e.pdf.

[4] Beckhoff Automation GmbH and Co. KG: Twincat 3 c++ (2019.11.29). http://download.beckhoff.com/download/document/automation/twincat3/TC1300_C_EN.pdf.

[5] Beckhoff Automation GmbH and Co. KG: Twincat 3 dc (2019.11.29). https://infosys.beckhoff.com/english.php?content=../content/1033/ethercatsystem/2469118347.html&id=/.

[6] Beckhoff Automation GmbH and Co. KG: Twincat 3 tccom (2019.11.29). https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/27021600255258507.html&id=/.

[7] Dr. Fodor Dénes and Dr. Szalay Zsolt: Autóipari kommunikációs rendszerek (2019.11.29). http://moodle.autolab.uni-pannon.hu/Mecha_tananyag/autoipari_kommunikacios_rendszerek/index.html.

[8] EtherCAT Technology Group: Ethercat base (2019.11.29). https://www.ethercat.org/download/documents/ETG_Brochure_EN.pdf.

[9] EtherCAT Technology Group: Ethercat communication (2019.11.29). https://www.ethercat.org/memberarea/download/EtherCAT_Communication_EN.pdf.

[10] EtherCAT Technology Group: Ethercat distance (2019.11.29). https://www.ethercat.org/en/technology.html.

[11] EtherCAT Technology Group: Ethercat et1100 section 1 (2019.11.29). https://download.beckhoff.com/download/document/io/ethercat-development-products/ethercat_esc_datasheet_sec1_technology_2i2.pdf.

[12] EtherCAT Technology Group: Ethercat et1100 section 2 (2019.11.29). https://download.beckhoff.com/download/Document/io/ethercat-development-products/ethercat_esc_datasheet_sec2_registers_2i7.pdf.

[13] EtherCAT Technology Group: Ethercat et1100 section 3 (2019.11.29). https://download.beckhoff.com/download/document/io/ethercat-development-products/ethercat_et1100_datasheet_v1i9.pdf.

[14] International Organization for Standardization: Iso 11898-2 (2019.11.29). http://read.pudn.com/downloads557/ebook/2297674/ISO11898/ISO11898-2.pdf.

[15] International Organization for Standardization: Iso 11898-3 (2019.11.29). https://www.sis.se/api/document/preview/907445/.

[16] KINGSTAR: The best ethernet protocol for your plc (2019.11.29). https://www.automation.com/pdf_articles/kingstar/Best_Ethernet_Protocol_for_Your_PLC.pdf.

[17] Pico Technology Ltd.: Pico scope (2019.11.29). https://www.picotech.com/download/datasheets/PicoScope3000SeriesDataSheet.pdf.

[18] Profinet University: Profinet basics (2019.11.29). https://profinetuniversity.com/category/profinet-basics/.

[19] Sercos International e.V: Sercos iii (2019.11.29). https://dc-pl.resource.bosch.com/media/pl/sercos/sercos3_en.pdf.

[20] Texas Instruments: Delfino (2019.11.29). http://www.ti.com/lit/ug/tidubq6a/tidubq6a.pdf.

[21] Texas Instruments: Lan9252 (2019.11.29). http://ww1.microchip.com/downloads/en/DeviceDoc/00001909A.pdf.

[22] Texas Instruments: Piccolo datasheet (2019.11.29). http://www.ti.com/product/TMS320F28069/technicaldocuments.

[23] Texas Instruments: Piccolo mikrokontroller (2019.11.29). http://www.ti.com/lit/ug/spruh18h/spruh18h.pdf.

[24] Vector Informatik GmbH: Flexray (2019.11.29). https://elearning.vector.com/mod/page/view.php?id=371/.