DIPLOMATERV-FELADAT

**Kardos Tamás**
szigorló villamosmérnök hallgató részére

# IO-Link RS-232 átalakító fejlesztése

Az IO-Link kommunikáció egyre jelentősebb területeket hódít meg az ipari automatizálás területén. Segítségével egyszerűvé válik a különböző intelligens eszközök paramétereinek beállítása, ill. a működés során fellépő hibák detektálása. Az IO-Link rendszert minden esetben egy master ill. egy slave eszköz alkotja.

Jelen feladat célja, egy IO-Link slave eszköz kifejlesztése, amely lehetővé teszi az RS-232-es interfésszel rendelkező eszközök (vonalkód olvasók, nyomtatók, RF-ID eszközök stb.) IO-Link rendszerbe történő integrálását. A kifejlesztendő eszköz számos beállítási lehetőséget kell, hogy tartalmazzon, hiszen alkalmas kell, legyen a változatos RS-232-es eszközök zömének lekezelésére. Ilyen paraméterek az RS-232-es átviteli sebesség, stop bitek száma, paritások, stb. Kihívást jelentő feladat lesz a tipikusan ciklikus adatátvitelt használó IO-Link kommunikáció, és az eseményvezérelt RS-232-es protokoll közötti zökkenőmentes átjáró funkció implementálása.

A tervezendő eszköz főbb tulajdonságai a következők:

- IO-Link interfész, amely megfelel a szabványban (IEC 61131-9 [1]) előírt követelményeknek
- RS-232-es interfész, amely paraméterei széleskörűen paraméterezhetők

A diplomatervezés során a következő részfeladatok megoldása vár a hallgatóra:

- Az IO-Link kommunikáció alapos megismerése
- RS-232 –es fizikai réteg, ill. a kommunikációt használó eszközök megismerése
- Hardver terv, kapcsolási rajz elkészítése
- Nyomtatott áramkörök megtervezése, esetleg próbapanelen való felépítése
- Az elkészült hardver alkotóelemek élesztése
- A működtető beágyazott szoftver megtervezése elkészítése

**Tanszéki konzulens:** Krébesz Tamás István, tanársegéd
**Külső konzulens:** Kása Zoltán (Balluff Elektronika Kft.)

Budapest, 2015. szeptember 22.


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

Kardos Tamás

# IO-LINK RS-232 ÁTALAKÍTÓ FEJLESZTÉSE

EGYETEMI KONZULENS
Krébesz Tamás István

KÜLSŐ KONZULENS
Kása Zoltán
(BALLUFF ELEKTRONIKA KFT.)

BUDAPEST, 2016

# Tartalomjegyzék

# HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2016. 12. 18.

...…….

Kardos Tamás

# Összefoglaló

Az IO-Link RS-232 átalakító létrehozásának az elsődleges célja, hogy elősegítse az IO-Link kommunikáció használatának további terjedését az iparban az automatizálás területein.

Jelen feladat célja, egy IO-Link slave eszköz kifejlesztése, amely lehetővé teszi az RS-232-es interfésszel rendelkező eszközök (vonalkód olvasók, nyomtatók, RF-ID eszközök stb.) IO-Link rendszerbe történő integrálását. A konverternek számos beállítási lehetőséget kell, hogy tartalmazzon, hiszen alkalmas kell, lennie a változatos RS-232-es eszközök zömének lekezelésére. Ilyen paraméterek például az RS-232-re jellemző adatátviteli sebesség, a stop bitek száma, paritások, stb. Kihívást jelentő feladat a tipikusan ciklikus adatátvitelt használó IO-Link kommunikáció, és az eseményvezérelt RS-232-es protokoll közötti zökkenőmentes átjáró funkció implementálása.

Logikailag a feladat a hardver kapcsolási rajzának megtervezésével kezdődött. Ezt követően elkészítettem a hardver nyomtatott áramköri rajzát. Miután rendelkezésemre állt a hardver, hozzáláttam a működtető beágyazott szoftver implementációjának. A feladat igazi nehézsége a szoftverben rejlett, amit a kifejlesztés ideje is jól tükrözött.

A diplomaterv feladatomat a Balluff Elektronika Kft-nek a BNI (Balluff Network Interface) nevű részlegén készítettem, amely részleg kifejezetten az IO-Link kommunikációs eszközök fejlesztésére specializálódott.

# Abstract

By now even the cheap sensors and actuators contain embedded micro controllers to create more complex systems than before. Thus the user can receive diagnostics informations from these deviceses and configure them for his purposes. This demand makes IO-Link protocol more and more popular in the automation industry. With the IO-Link communications these „smart devices" can be easily configured and failures can be recognized during operation.

By using the developed device such smart devices can be integrated into an IO-Link system that have RS-232 interface (e.g. barcodescanner, xerox machine, RF-ID devices, etc.). The converter has to provide various range of parameters like data rate, number of stop bits, parity, etc. to satisfy the demands of most devices. It is a challenge to carry out the data transmission without any problems between the acyclic IO-Link and cyclic RS-232 communication.

The first step of developement was the creation of the circuit diagram and then the layout design of the printed circuit. After that I built the hardware and tested its operation. Finally I developed the embedded software. The most difficult part of this project was the firmware which consumed the most develompent time.

I made my MSc thesis at Balluff Electronics Company. The comany department where I work is specialized in the developement of IO-Link communications devices.

# 1 Az IO-Link kommunikáció

Manapság már az olcsó szenzorokban és beavatkozókban is, az egyre komplexebb alkalmazások igényeinek megfelelve, beágyazott mikrokontrollerek találhatóak. Ez lehetőséget nyújt ezen eszközök diagnosztikájára és konfigurációjára, ami megfelel a magasabb elvárásoknak.

Az IO-Link kommunikáció egyre jelentősebb területeket hódít meg az ipari automatizálás területén. Ez több vezető (jellemzően német) cég együttműködésének az eredménye: Balluff, Beckhof, Festo, Leuze electronic, Pepperl+Fuchs, Siemens, ifm, stb.

Az IO-Link rendszer segítségével egy vezérlő egységnek (PC vagy PLC) egyszerűvé válik a különböző intelligens eszközök paramétereinek beállítása, ill. a működés során fellépő hibák detektálása. Az IO-Link rendszert minden esetben egy master, és egy slave eszköz alkotja.

E fejezetnek nem célja az IO-Link szabványt részletesen definiáló specifikáció (IO-Link Interface and System Version 1.1.2) helyettesítése [2], hanem abból csak a feladatom szempontjából lényeges elemeket ragadtam meg és ismertetem mélyrehatóan.

## 1.1 Az IO-Link kommunikáció áttekintése

Az IO-Link egy új, egyszerű, univerzális kommunikációs megoldás szenzorok és beavatkozók számára. Az IO-Link megnevezés mellett, találkozhatunk még az SDCI (Single-drop Digital Communication Interface for small sensors and actuators) elnevezéssel is.

A szenzorok, ill. beavatkozók, amelyek az IO-Link rendszerben slave eszközöknek tekintendők, és az IO-Link master között folyó kommunikáció, kétirányú pont-pont összeköttetés (nem busz). Egyszerű, és olcsó kialakítású, vagyis nincsen szükség bonyolult protokoll ASIC-ekre. Az I/O adatok továbbítása soros kommunikáció segítségével történik. Kompatibilis az „egyszerű”, bináris kapcsolókkal/beavatkozókkal, ill. az elterjedt terepbuszokkal (PROFIBUS, PROFINET, stb.).

Az IO-Link rendszer biztosította lehetőségek és előnyök összefoglalva a következők:

- A szenzorok, ill. beavatkozók kínálta intelligencia maximális kihasználása.
- A szenzorok, ill. beavatkozók paramétereinek kényelmes beállítása (paraméterezés).

- A beállított paraméterek központi, jól definiált tárolása a master egységben.
- Központi, on-line, diagnosztika lehetősége.
- Egyszerű installáció, a kommunikációs csatornák sokféleségének redukálása.
- Csökkenő kábelezés.

## 1.2 Az IO-Link rendszer elemei

Az IO-Link rendszert minden esetben egy IO-Link master és egy IO-Link slave (Device) alkotja. A slave eszköznek a rendszerben a szenzorok és a beavatkozók felelnek meg. A master és slave egységeket összekötő kábel 3-vezetékes, árnyékolatlan, maximum 20 m hosszúságú. Az 1.1. ábra az IO-Link rendszer elemeit mutatja.

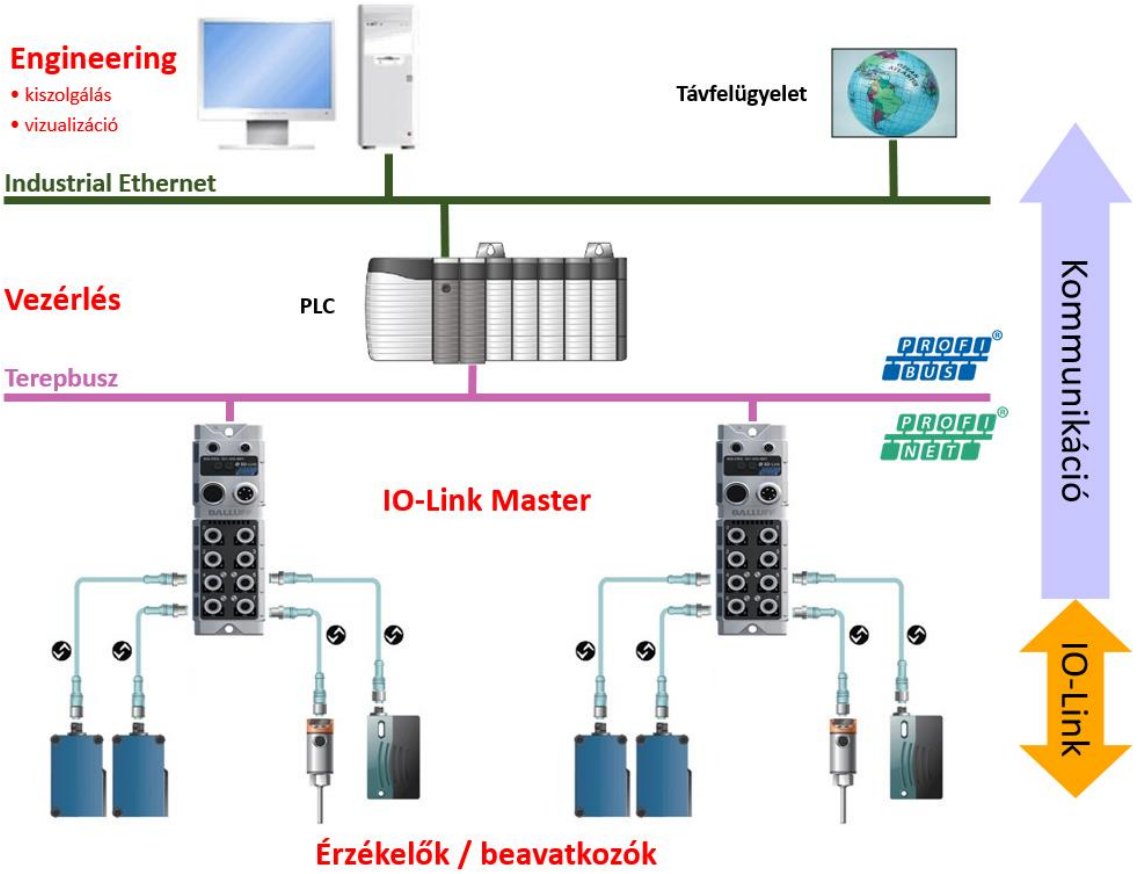

**1.1. ábra: Az IO-Link rendszer elemei**

A topológiának jelentős előnye más rendszerekkel összevetve a kis kábelezési igény, mivel a vezérlőegység (PC vagy PLC) kizárólag az IO-Link master-hez kapcsolódik valamilyen terepbuszon (PROFIBUS, PROFINET stb.) keresztül, amelyhez több slave egység is csatlakoztatható.

## 1.3 Elhelyezkedés az automatizálási hierarchiában

Az 1.2. ábra az IO-Link technológia elhelyezkedését szemlélteti az automatizálási hierarchiában. Látható, hogy az IO-Link kommunikáció a hierarchia alsó szintjén zajlik, az IO-Link master és a szenzorok, ill. beavatkozók között. A felsőbb szintekkel a kommunikációt az IO-Link master végzi, valamilyen terepbusz alkalmazásával (PROFIBUS, PROFINET, stb.) Az

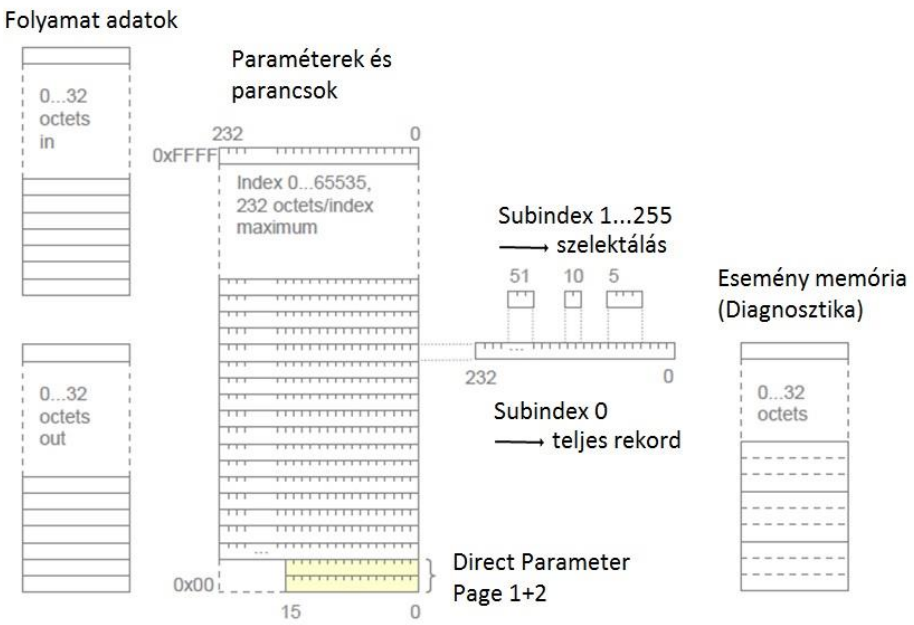
1.2. ábrán látható rendszerben, a szenzoroktól és beavatkozóktól érkező adatoknak a felsőbb rétegekbe való átvitelét nevezzük szokásosan bemenő adatnak, ill. az ellenkező irányban történő átvitelt kimenő adatnak. A vezérlést általában ipari környezetben egy PLC végzi (de lehet PC is), amihez egy PC csatlakozhat Etherneten keresztül, vizualizáció céljából.




**1.2. ábra: Az IO-Link technológia elhelyezkedése az automatizálási hierarchiában**

## 1.4 Az IO-Link kommunikáció főbb jellemzői

Az 1.3. ábrán az általános slave modell látható.




**1.3. ábra: Általános IO-Link slave modell**

A slave eszköz ciklikus jelleggel fogadhat olyan ún. kimenő folyamat adatokat (Process data), amik vezérelnek egy diszkrét vagy folytonos automatizálási folyamatot, ill. küldhet olyan bemenő folyamat adatokat, amik az adott folyamat pillanatnyi állapotát reprezentálják, vagy annak a mért értékei. Mind a kimenő, mind a bemenő folyamat adatok esetében, egy ciklusban maximum 32 byte méretű adat továbbítható.

A slave eszközök általában rendelkeznek különféle paraméterekkel, amiket a felhasználó a számára legoptimálisabban állíthat be. Ennek az igénynek a támogatása céljából, széles paramétertartomány van definiálva a szabványban, ami egy index (0-65535) és subindex (0-255) címzési metódust alkalmaz.

Az első két indexnek, a 0-nak és az 1-nek, kitüntetett szerepe van. Ezeket a szabvány külön névvel is illette, amik a Direct Parameter Page (DPP) 1 és 2 nevet viselik, és ezek, eltérően a többi indextől, csak maximum 16 byte méretűek lehetnek. A Direct Parameter Page 1 főként az olyan adatoknak a tárolója, amiket a master igényel a kommunikáció kiépítéséhez, de találhatóak itt még eszköz specifikus és identifikációs adatok is. A Direct Parameter Page 2 pedig kifejezetten eszköz specifikus paramétereknek a tárolására van fenntartva.

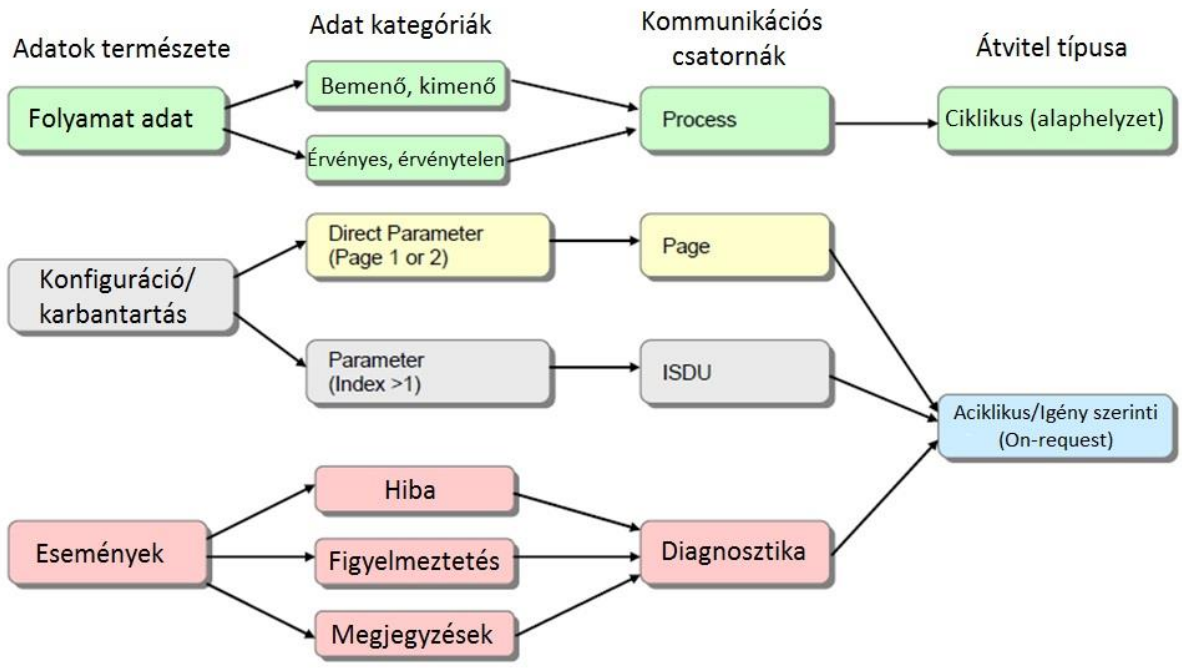
A többi index (2-65535) alatt maximum 232 byte méretű rekord található. A subindex 0, az index által megcímzett egész rekord elérését jelenti, míg a 0-tól eltérő subindex egy-egy adategységet címez meg a rekordon belül. Egy rekordon belül a különböző adat egységek, bármilyen bit eltolással kezdődhetnek, és a hosszuk 1 bit-től egészen 232 byte-ig terjedhet, de

egy rekordon belül az összes adat egységnek a száma nem haladhatja meg a 255-öt. Egy slave eszköz esetében értelmezett adategységek/paraméterek elhelyezkedését a címtartományban, az ún. IODD (IO-Link Device Description) fájl tartalmazza [3], amivel minden eszköznek kötelezően rendelkeznie kell.

A slave eszköz minden állapotváltozása, ami jelzést vagy közbeavatkozást igényel, elmentésre kerül egy ún. esemény memóriába (Event memory) az átvitelt megelőzően. Ilyenkor egy ciklikus adatcserében részt vevő esemény flag bebillenése jelzi, hogy valamilyen esemény bekövetkezett.

A kommunikáció a master és slave eszközök között pont-pont kapcsolat, és azon az elven zajlik, hogy minden esetben először a master küld egy üzenetet, amire a slave eszköz válaszol. A két üzenetet együttesen a szabvány M-sequence-nek (Message-sequence) nevezi. Több féle M-sequence típust definiál a szabvány, hogy a felhasználók igényeit a protokoll maximálisan ki tudja elégíteni.

Az IO-Link kommunikáció az adatokat a természetük szerint kategorizálja, és a kategóriákhoz különböző logikai adatátviteli csatornákat definiál, amelyeket az 1.4. ábra szemléltet.




**1.4. ábra: Az IO-Link kommunikáció logikai adatátviteli csatornái**

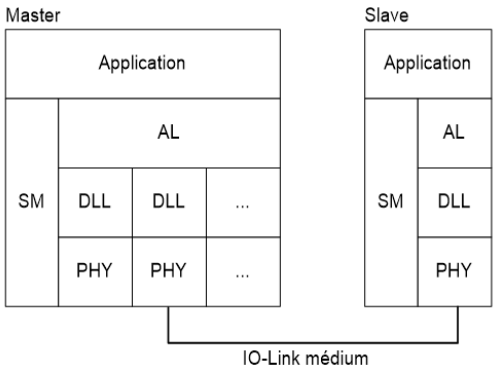
A be és kimenő folyamat adatok (Process data) ciklikus adatcserét végző csatornán (Process csatorna) kerülnek átvitelre. Lehetőség van azt is jelezni, hogy az adott ciklusban átvitt folyamat adatok érvényesek vagy érvénytelenek.

A konfigurációs és karbantartás jellegű adatok aciklikus (On-request) csatornákon kerülnek átvitelre. A Page csatorna egy közvetlen adathozzáférési lehetőséget biztosít a Direct Parameter Page 1 és 2-höz. Az ISDU (Index Service Data Unit) csatorna a paramétertartomány további területein elhelyezkedő adatokhoz biztosít hozzáférést.

Az esemény jellegű adatok az aciklikus (On-request) diagnosztikai csatornán (Diagnosis) kerülnek átvitelre. Az eseményeknek 3 kategóriája került megkülönböztetésre, amik a hibák (Errors), figyelmeztetések (Warnings), és megjegyzések (Notifications).

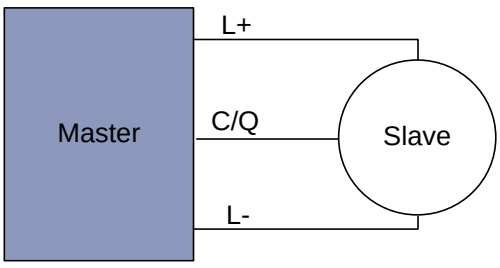
## 1.5 Fizikai réteg

Az ISO/OSI referenciamodellnek 3 rétege került leimplementálásra az IO-Link szabványban, amit az 1.5. ábra szemléltet. Ezek a rétegek a fizikai réteg (PHY – Physical layer), az adatkapcsolati réteg (DLL – Data Link Layer), és a felhasználói réteg (AL – Application Layer). Jelen fejezetben a fizikai réteget mutatom be röviden.



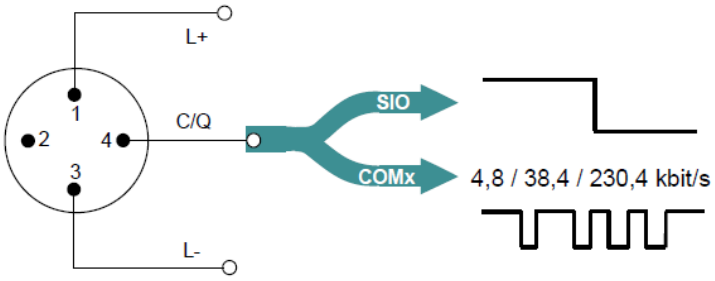

**1.5. ábra: Logikai struktúrája az IO-Link master és slave eszközöknek**

Az 1.6. ábra a 3-vezetékes IO-Link kapcsolatot ábrázolja.




**1.6. ábra: 3-vezetékes IO-Link kapcsolat**

Az IO-Link master eszköznek a 3-vezetékes kialakítást kötelezően támogatnia kell. A gyakorlatban legtöbbször 4 pin-es csatlakozókat alkalmaznak, amelyet az 1.7. ábra szemléltet, valamint a lábkiosztásukat az 1.1. táblázat tartalmazza. Látható, hogy az 1-es pin-en a master az ipari szabványnak megfelelően 24 V-os tápfeszültséget biztosít a csatlakoztatott slave eszköznek. A 2-es pin szabad felhasználási célú jelvezeték, ami alaphelyzetben nincsen csatlakoztatva. Ha az alkalmazás úgy kívánja, a vonal használható digitális be- és kimenetként (DI/DO) egyaránt. A 3-as pin föld. A 4-es pin-en, azaz a C/Q elnevezésű jelvezetéken zajlik az adatátvitel, ahol a ”C” (Communication) az IO-Link, a ”Q” a SIO (Simple Input Output) üzemmódot jelentik. SIO módban a hagyományos, „egyszerű bináris szenzorokat” csatlakoztathatunk a master eszközhöz, de a feladatom szempontjából kizárólag az IO-Link kommunikáció érdekes.

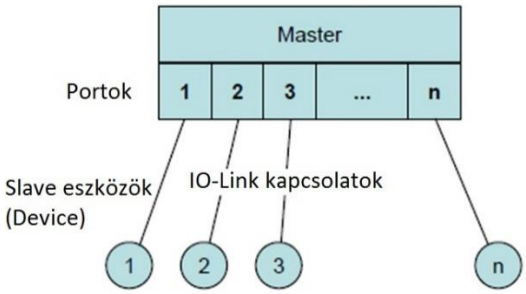



**1.7. ábra: 4 pin-es csatlakozó**

| Pin | Vonal | Megjegyzés |
|---|---|---|
| 1 | L+ | 24 V/1,6 A (tápfeszültség) |
| 2 | | Szabad felhasználási célú (DI/DO) |
| 3 | L- | GND |
| 4 | C / Q | Kommunikáció/Kapcsolójel |

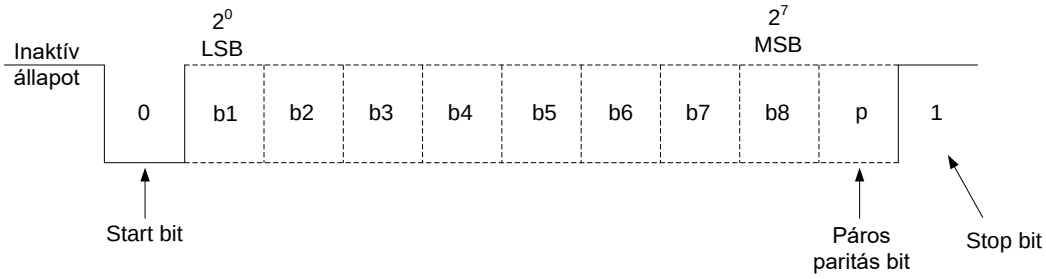
**1.1. táblázat: 4 pin-es csatlakozó lábkiosztása**

Az IO-Link rendszer pont-pont összeköttetést használ a master és slave eszközök között, amit az 1.8. ábra szemléltet. Látható, hogy a master eszköz több porttal is rendelkezhet, amelyekhez slave eszközök csatlakoztathatóak, azonban portonként csak egy eszközt képes kezelni.




**1.8. ábra: Az IO-Link rendszer topológiája**

Az IO-Link kommunikáció az adatátvitelre NRZ (Non Return to Zero) kódolású UART (Universal Asynchronous Receiver/Transmitter) adatkereteket használ, amit az 1.9. ábra szemléltet. Az adatkeretek 8 adatbitet, páros paritást és 1 stop bitet használnak.




**1.9. ábra: NRZ kódolású UART karakter**

A logikai értékekhez tartozó jelszinteket az 1.2. táblázat tartalmazza.

| **Jelszintek** | |
|---|---|
| Logikai 1 | 0 V |
| Logikai 0 | 24 V |
| Inaktív állapot | 0 V |

**1.2. táblázat: Az IO-Link protokoll jelszintjei**

A szabvány három adatátviteli sebességet támogat, amelyekre a COM1, COM2 és COM3 névvel hivatkozik. Az 1.3. táblázat az ezekhez tartózó értékeket tartalmazza.

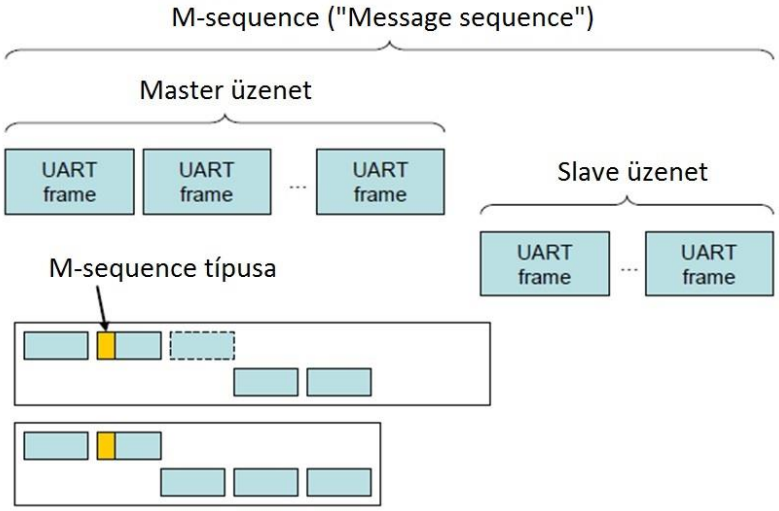
| **Adatátviteli sebességek** | |
|---|---|
| COM1 | 4,8 kBd |
| COM2 | 38,4 kBd |
| COM3 | 230,4 kBd |

**1.3. táblázat: Az IO-Link protokoll adatátviteli sebességei**

## 1.6 Adatkapcsolati réteg

### 1.6.1 Adatátvitel

A kommunikációt a master és a slave eszközök között, minden esetben a master kezdeményezi. A master küld egy üzenetet, amire a slave eszköz válaszol. Mindkét üzenet UART adatkeretekből tevődik össze. A két üzenetet együttesen a szabvány M-sequence-nek (Message-sequence) nevezi. Az adatátvitelre többféle M-sequence típus definiált, amikkel a slave eszközök sokszínűségét támogatja a szabvány. A leírtakat az 1.10. ábra szemlélteti.




**1.10. ábra: M-sequence**

A ciklikus folyamat adatok (Process data) és az aciklikus igény szerinti adatok (On-request data), különböző logikai csatornákon kerülnek átvitelre. A ciklikus folyamat adatok átvitelét, az aciklikus igény szerinti adatok átvitele nem befolyásolja.

Az IO-Link kommunikációban a különféle logikai adatátviteli csatornákat az M-sequence valósítja meg. Többféle M-sequence típus definiált, azért hogy a szenzorok és beavatkozók eltérő hosszúságú folyamat adatai által támasztott követelményekhez illeszkedjen. Az 1.11. ábrán a definiált M-sequence típusok láthatók.


**1.11. ábra: M-sequence típusok**

Ahogyan az 1.11. ábrán megfigyelhető, a Master üzenet első UART kerete az MC (M-sequence Control). Ezt követi a CKT (Check/Type) keret, majd csak ezután kerülnek átvitelre az M-sequence típusától függően, a folyamat (PD), vagy az igény szerinti (OD) adatok. A szaggatott vonallal bekeretezett részek függenek az olvasás vagy írás irányától. Az adatátvitel utolsó kerete minden esetben a CKS (Check/Status), amelyet a slave eszköz küld. Látható, hogy definiáltak fix és változtatható hosszúságú M-sequence típusok.

Az MC byte-ban kerül definiálásra, hogy írás, vagy olvasás műveletet kezdeményez a master. Továbbá itt kerül az is kijelölésre, hogy az aciklikus igény szerinti adatok (OD), mely logikai csatornáról érkeznek, vagyis az aktuális M-sequence-ben az igény szerinti adatokat (OD) a Page, az ISDU, vagy a diagnosztikai csatorna küldte.

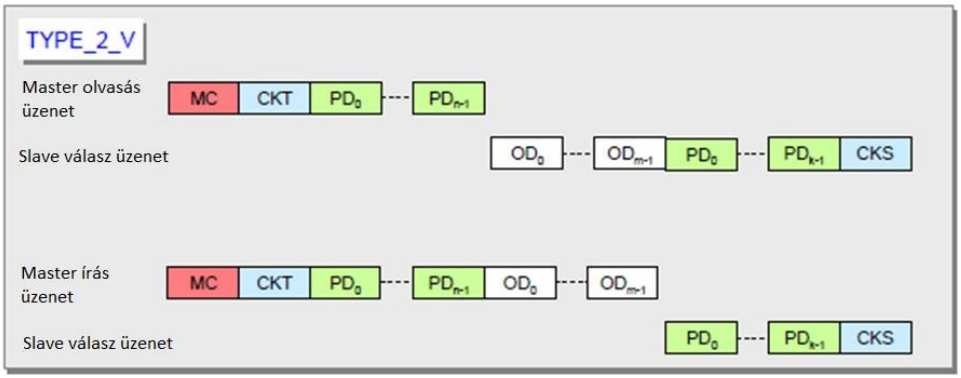
A CKT byte tartalmaz egy adatátvitel helyességének ellenőrzésére szolgáló ellenőrzőösszeget, valamint kijelöli az M-sequence típusát. Az ellenőrzőösszeget a master az összes átküldendő adat-byte alapján kalkulálja, aminek a felhasználásával a slave képes

ellenőrizni a fogadott adatok helyességét. Az M-sequence típusának megadásával a master definiálja, hogy az üzenetek az M-sequence-ben strukturálisan hogyan épüljenek fel.

A CKS byte a válasz üzenet része, amelyet a slave eszköz küld a master-nak. A byte tartalmaz egy ellenőrzőösszeget, amivel a master ellenőrizheti az adatátvitel helyességét. Továbbá tartalmaz egy flag-et, amellyel a slave a bemenő folyamat adatokat érvényes vagy érvénytelen státusszal láthatja el. Ezenkívül a byte-ban egy esemény flag is található, amelyen a slave eszköz jelezheti egy esemény bekövetkeztét, vagyis a flag-gel a diagnosztikai logikai csatornának a használatát kérelmezheti a master-tól.

Az IO-Link RS-232 átalakító kifejlesztése során olyan M-sequence típusra volt szükségem, ami képes nagyobb mennyiségű ki és bemenő folyamat adatok kezelésére. Erre a célra a TYPE_2_V típus a legalkalmasabb, ezért ezt a típust röviden bemutatom.

Az 1.12. ábra a TYPE_2_V típusú M-sequence-et szemlélteti. Ennél a típusnál nem fix értékű az folyamat és igény szerinti adatok száma, vagyis lehetőséget nyújt a fejlesztőnek az értékek megadására. A ki és bemenő folyamat adat byte-ok (PD) száma tetszőlegesen 0-tól 32-ig terjedően állítható (az ábrán az *n* és *k* betűk értéke). Az igény szerinti (OD) adat-byte-ok lehetséges értékei 1, 2, 8 és 32 (az ábrán *m* betű értéke).




**1.12. ábra: TYPE_2_V típusú M-sequence**

## 1.6.2 A kommunikáció kiépítése

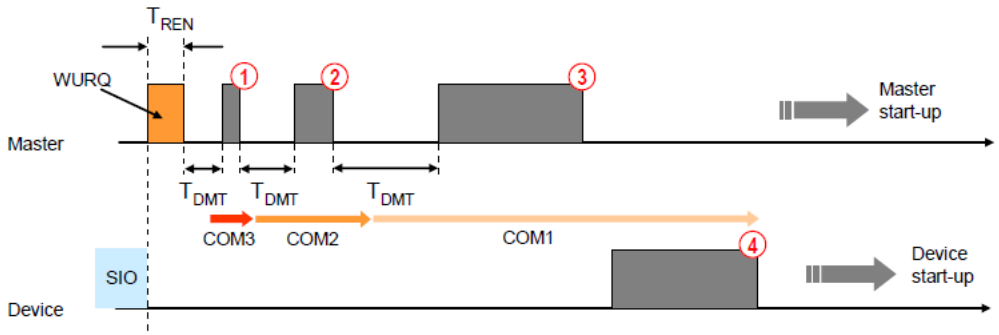
A kommunikációt a master és a slave eszközök között, minden esetben a master kezdeményezi. A master küld egy üzenetet, amire a slave válaszol. A master két megszólítása között eltelt idő a ciklusidő. A ciklusidő a kommunikáció egy fontos paramétere. A ciklusidő megválasztásánál szerepet játszik az adatátvitel sebessége is (COM1, COM2, COM3), hiszen ha az gyors, a ciklusidő is kicsi lehet. Ha a ciklusidő kicsi, a rendszer gyorsabban képes reagálni a változásokra. Természetesen az, hogy ezen a téren egy adott rendszernek milyen

kritériumokat kell kielégítenie, alkalmazásfüggő. Az ajánlott minimális ciklusidők értékeit, az adatátvitel sebességétől függően, az 1.4. táblázat tartalmazza.

| Adatátvitel sebessége | Minimális ciklusidő ($t_{CYC}$) |
|---|---|
| COM1 | 18,0 ms |
| COM2 | 2,3 ms |
| COM3 | 0,4 ms |

**1.4. táblázat: Ajánlott minimális ciklusidők**

A kommunikáció kiépülési fázisában a master még nem tudja, hogy a csatlakoztatott slave eszköz milyen adatátviteli sebességet, ill. minimális ciklusidőt támogat, hiszen ezek a paraméterek slave eszközönként eltérőek lehetnek. A master eszköznek többek között ezeknek a paraméterértékeknek a megállapítására van szüksége, mielőtt a ciklikus folyamat adatok átvitelébe kezdhetne. Az 1.13. ábra a kommunikáció kiépülési folyamatát szemlélteti.




**1.13. ábra: A kommunikáció kiépítése (Wake up sequence)**

Az ábrán látható, hogy a kommunikáció egy ún. WURQ (Wake Up Request) impulzussal kezdődik. A WURQ impulzussal a master ellenkező jelszintet ad ki a vonalra, amellyel felébreszti a SIO módból a slave eszközt. Ezt követően, a megfelelő időkorlátok betartásával, a master egy üzenetet (TYPE_0 típusú M-sequence) küld COM3 adatátviteli sebességgel. Ha nem érkezik válasz az üzenetre az elküldésétől számított $T_{DMT}$ időn belül, az ismét elküldésre kerül immár COM2 adatátviteli sebességen. Végül, ha továbbra sem érkezik válasz, a COM1 sebességen próbálkozik újra a master. Az ábrán látható példán, a slave eszköz COM1 adatátviteli sebességet használ, mivel az a COM1 sebességen elküldött megszólításra reagált.

A master az ábrán látható üzenetben lekérdezi a slave eszköznek a minimális ciklusidejét. A slave-től érkező válasz a master részére azonnal két információt is hordoz,

hiszen a slave a válaszüzenetben megadja a minimális ciklusidejét, valamint a master megtudja, hogy a csatlakoztatott eszköz milyen adatátviteli sebességet támogat (az ábrán látható példában COM1-et).

A választ követően azonban a master és slave eszközök között a ciklikus folyamat adatok átvitele még mindig nem kezdődik meg. A szabvány definiál ún. Start up, Preoperate és Operate kommunikációs szakaszokat. A kommunikáció kiépülési fázisai a Start Up és Preoparate szakaszok, ahol a master lekérdezi a slave által támogatott M-sequence típusokat, az IO-Link verzióját, a bemeneti/kimeneti adatok jellemzőit, és még egyéb más paramétereket is. A Preoperate szakasz legvégén a master visszaírja a slave-be az alkalmazandó ciklusidőt, amivel a slave eszköz aktiválása megtörténik, a kommunikáció az Operate szakaszba lép át, és megkezdődik a ciklikus folyamat adatok átvitele.

## 1.7 IODD fájl

A slave eszközök mindegyike rendelkezik egy ún. IODD (IO-Link Device Description) fájllal, ami egy XML formátumú leíró fájl az adott eszközről. Az IODD fájl szerkezetét szabvány definiálja (IO-Link Device Description Specification Version 1.1) [3]. Az IODD leíró fájl tartalmaz minden információt az adott slave eszközről, beleértve az eszköz paramétereit, és azoknak a címeit is.

Az IO-Link eszközöket gyártó cégek a felhasználóknak biztosítanak olyan szoftvert, amely képes az eszközökhöz tartozó IODD fájlokat beimportálni, és az IODD-ben tárolt információt felhasználó barát módon megjeleníteni. Így a slave eszközt a felhasználó egyszerűen egy PC-ről felparaméterezheti. Végeredményben, a felhasználó egy grafikus felületet lát, ahol ki vannak listázva az adott slave eszköznek a paraméterei, és egy legördülő listából azoknak az értékeit szabadon állíthatja. Az IODD fájl lehetőséget nyújt arra is, hogy különböző nyelvi beállítások közül választhasson a felhasználó.

Az IODD fájl továbbá fejlesztési célokra is felhasználható. A PLC programozás során lehetőség van a projekthez hozzáadni, ezáltal a kód érthetőbbé válik. A programozónak nem byte-okat és biteket kell megcímeznie, hanem az IODD-ben szereplő neveket használhatja fel, amik már a megfelelő címekre mutatnak.

# 2 IO-Link RS-232 átalakító

Az IO-Link RS-232 konverter kifejlesztésének az igényét a General Motors megrendelése adta. Számukra egy olyan konverterre van szükség, amivel a meglévő IO-Link rendszerükbe képesek legyenek RS-232 interfésszel rendelkező vonalkód olvasókat integrálni. A 2.1. ábra egy ilyen vonalkód olvasó készüléket ábrázol.

A vonalkód olvasótól beérkező adatokat, egy PLC dolgozza fel. Tekintve, hogy ez nem tartozik a feladatomhoz, bővebben nem foglalkoztam vele.

**2.1. ábra: RS-232 interfésszel rendelkező vonalkód olvasó (példaalkalmazás)**

A feladatom kivitelezéséhez először az IO-Link kommunikációt kellett mélyrehatóan megismernem. Az IO-Link kommunikáció című fejezetben ismertettem, hogy az IO-Link rendszert minden esetben egy master és egy slave eszköz alkotja. A kifejlesztett konverter egy IO-Link slave eszköz. A kommunikáció a master és a slave eszközök között ciklikus jellegű, ahol a master kérdez, a slave válaszol.

Meg kellett ismernem alaposan az RS-232 fizikai réteget, ahol a kommunikáció, ellenben az IO-Link-kel, eseményvezérelten zajlik.

A feladatom az volt, hogy egy olyan átalakítót készítsek, amely a két protokoll között egy zökkenőmentes átjáró funkciót valósít meg. Azonban kihívást jelentő probléma, hogy a két protokoll alapjaiban eltér egymástól, hiszen az adatátvitel az IO-Link kommunikációban tipikusan ciklikus, míg az RS-232 eseményvezérelten zajlik. A probléma megoldására ki kellett dolgozom egy olyan IO-Link rendszer fölé rendelt kommunikációs logikát/protokollt, amellyel

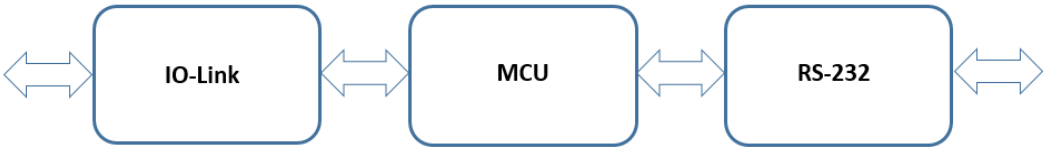
a kifejlesztett konverter és a vezérlés (PC, PLC) jelezhetik egymás részére az adatátviteli igényeket.

A konverternek számos beállítási lehetőséget kell tartalmaznia, hiszen alkalmasnak kell lennie a változatos RS-232-es eszközök zömének lekezelésére. Ilyen paraméterek az RS-232-es adatátviteli sebesség, stop bitek száma, paritások, stb. Az átalakító konfigurációjára, az IO-Link slave eszközöknek megfelelően, az aciklikus ISDU csatornán keresztül van lehetőség.

Figyelembe kellett vennem a hardver tervezésekor, hogy a konverternek EMC (Electromagnetic Compatibility) biztosnak kell lennie, hiszen ipari környezetekben kell megállnia a helyét. Ezért, hogy megbizonyosodást szerezzek arról, hogy a hardver erősen zajjal terhelt környezetekben is működőképes, azt EMC tesztelésnek vetettem alá.

## 2.1 Hardver

Az IO-Link RS-232 átalakító alapvetően nem igényel nagy komplexitású hardvert, azonban a tervezésnél figyelembe kellett vennem, hogy az eszköznek EMC biztosnak kell lennie, hiszen ipari környezetekben kell megállnia a helyét. A konverter blokkvázlatát a 2.2. ábra szemlélteti.




**2.2. ábra: IO-Link RS-232 átalakító blokkvázlata**

Az ábrán megfigyelhető, hogy a konverter hardver specifikációja 3 fő blokkra osztható fel. Az MCU (Micro Controller Unit) nevű blokk az áramkör központi adatfeldolgozó egysége, ahol konverter logikája kerül implementálásra. Az IO-Link és RS-232 blokkok jelátalakító szerepkört töltenek be, vagyis a külvilág és az MCU blokk közötti jelszintek átalakításáért felelnek.

Az MCU blokkot megvalósító mikrovezérlő kiválasztásánál, azt a szempontot vettem figyelembe, hogy kontroller a manapság nagyon elterjedt ARM architektúrával rendelkezzen. Sok gyártó kínál ilyen architektúrájú mikrovezérlőket (NXP, Texas Instruments, ATMEL, stb.), végül a választásom egy STMicroelectronics által forgalmazott, Cortex-M3 architektúra családból származó, 32 bites, ultra alacsony fogyasztású mikrovezérlőre esett (STM32L1 széria) [4].

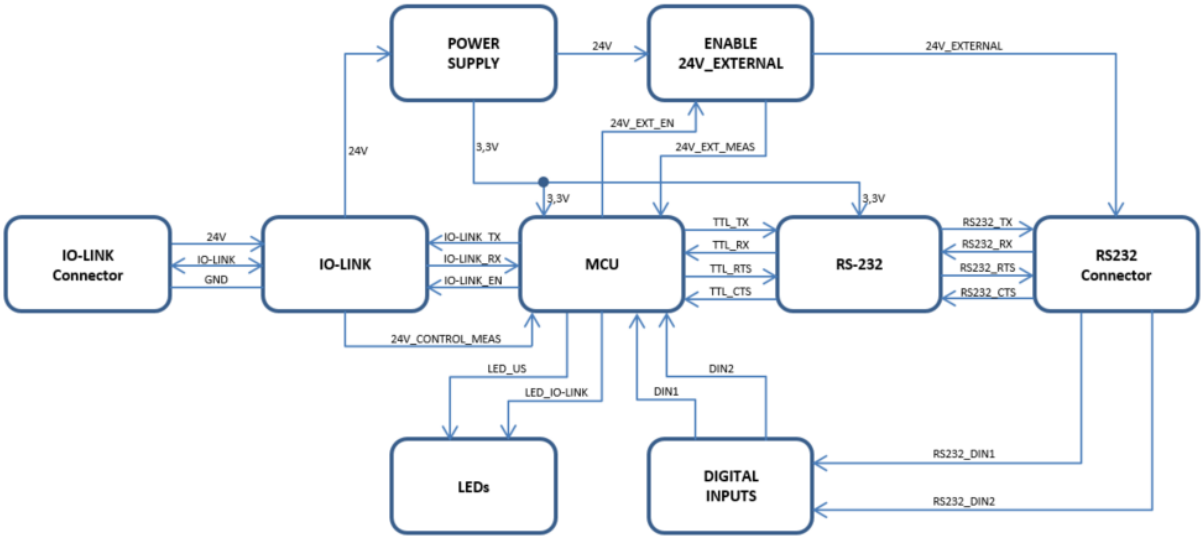
Az IO-Link blokk megvalósítására 2 alkalmas fizikai réteg illesztő integrált áramkört is találtam a piacon, amelyek a következők:

- E981.10 (ELMOS) [5]
- MAX14824 (Maxim Integrated) [6]

Az E981.10 IC tudása ugyan kisebb, de a konverter szempontjából elegendő, ellenben olcsóbb, ezért emellett döntöttem.

Az RS-232 blokk realizációjához számos fizikai rétegillesztő integrált áramkört találtam. A választásom végül, praktikussági szempontból, a cégnél már készleten lévő ESD védett ICL3232IVZ IC-re esett [7]. A későbbiekben azonban megfontolandó lehet optikailag leválasztott IC-re való áttérés, amely nagyobb védelmet nyújt a zavarokkal szemben, ilyen IC például a MAX3250 [8].

A 2.3. ábra a hardver részletesebb rendszertervét ábrázolja.




**2.3. ábra: IO-Link RS-232 átalakító rendszerterve**

Az áramkör központi egysége tehát egy 32 bites, ultra alacsony fogyasztású mikrovezérlő (STM32L151), amelyen a működtető beágyazott szoftver fut. Ez a blokkvázlatban az MCU név alatt szerepel.

A 2.3. ábrán látható, hogy az áramkör a szabványos 24V-os ipari tápfeszültséget az IO-Link csatlakozón keresztül kapja.

Az IO-Link nevű blokk tartalmaz egy bemeneti zavarvédelmi fokozatot, és egy IO-Link fizikai réteg illesztő integrált áramkört (E981.10). A zavarvédelem egy fontos szempont, hiszen

a konverternek meg kell felelnie az EMC teszt által támasztott követelményeknek. A külvilág felől érkező közös modusú zajok, az eszköz működését nem befolyásolhatják. A zavarvédelmet megvalósító elektronikát nem magam terveztem, hanem a cég termékeinél alkalmazott jól bevált, sztenderd végfokot használtam fel. Az IO-Link szintillesztő IC-nek a rendszerbe integrálásakor, az adatlapon található ajánlást alkalmaztam.

A Power Supply nevű blokk feladata a 24V-os tápfeszültség 3,3V-ra történő átalakítása, amelyről az áramkör további blokkjai/integrált áramkörei üzemelnek. Valójában a blokk mögött egy DC/DC feszültség átalakító IC rejlik (LM2674) [9]. A kapcsolóüzemű DC/DC átalakító használata a hő disszipáció, ill. a hatásfok tekintetében előnyös. A hő disszipáció alacsony értéken tartása különösen fontos a konverter szempontjából, mivel a konstrukcióját tekintve, egy vékony cső alakú házban foglal helyet a PCBA gyantával kitöltve.

Ahogyan a bevezetésben is ismertetésre került, a mikrovezérlő nem kapcsolódik közvetlenül az RS-232 csatlakozóra sem, hanem annak a jelei egy szintillesztő egységbe (ICL3232IVZ) futnak be először, amelyet az ábrán az RS-232 blokk szimbolizál. A blokk funkciója kizárólag az, hogy a mikrovezérlőtől érkező adatokat a szabványos jelszintekre emelje, ill. a kívülről a csatlakozókon keresztül beérkező adatokat, alacsonyabb, TTL logikai jelszinten juttassák el a mikrovezérlőnek. Az IC adatlapja a bekötésre vonatkozólag tartalmazott egy általános javaslatot, amit a kapcsolási rajzban fel is használtam.

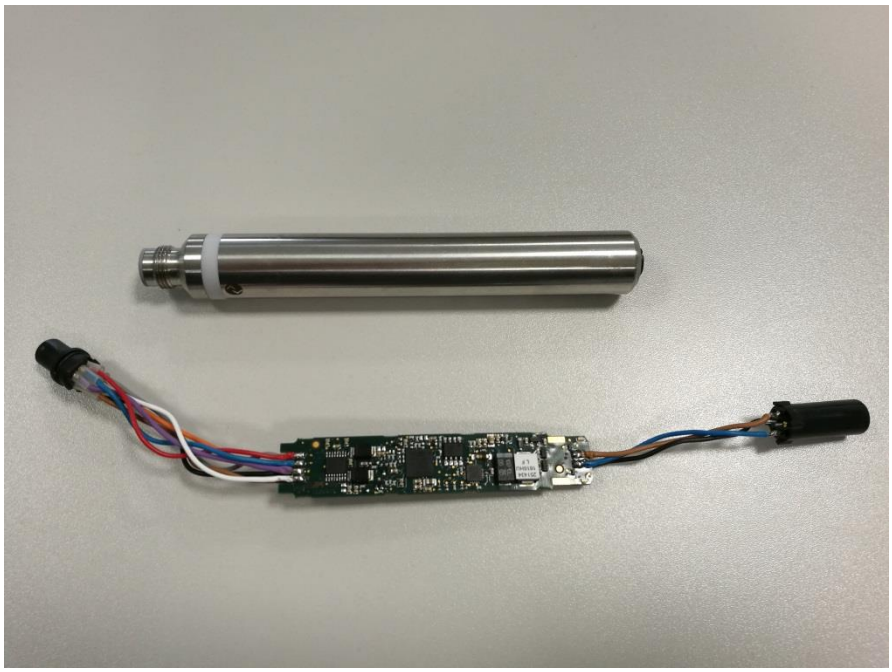
Az Enable 24V External nevű blokk egy kapcsolható 24V-os tápfeszültséget szolgáltat az RS-232 interfésszel rendelkező eszköznek. A kapcsolás egy Hide-Side-Switch IC-vel (ISP762T) történik [10]. Gondolni kellett arra is, hogy előfordulhat az az eset, hogy a felhasználó hibás konfigurációból adódóan a 24V-os tápfeszültséget rövidre zárja, amit a konverternek képesnek kell lenni felismernie és elviselnie is. Így a rövidzár detektálásának céljából, a blokk tartalmaz egy ellenállás-alapú feszültségosztót is a 24V-os kimenetén. A feszültségosztón eső feszültség értékéből a rövidzár ténye felismerhető, amit a mikrovezérlő az AD konverterrel monitoroz.

Mivel az alkalmazott csatlakozón felhasználatlanul maradt 2 db pin, ezért azokat egyszerű digitális bementekké alakítottam, amik különféle trigger jelként hasznosíthatóak. A blokkdiagramban ezeket a Digital Inputs blokk szemlélteti. A bemeneteken a 24V felel meg a logikai magas szintnek, ill. a 0V a logikai alacsony szintnek. A digitális bemeneteknél alkalmazott elektronika, a cég más termékeiben is használt, sztenderd megoldásnak számít, ami az ipari szabványnak megfelelő kapcsolási karakterisztikát valósítja meg.

A LED nevű blokk az áramkör 2 db LED-jét jelképezi. Az egyik LED az IO-Link kapcsolatról, a másik a tápfeszültségről szolgáltat visszajelzést.

A konverter IO-Link felőli oldalán egy M12-es A-kódolású, 4 pin-es apa csatlakozó, míg az RS-232 oldalon egy M12-es A-kódolású, 8 pin-es anya csatlakozó található.

Az elkészült prototípust a 2.4. ábra szemlélteti. Az ábrán a ház is látható, amiben a hardver gyantával kitöltve foglal helyet.



**2.4. ábra: Az elkészített prototípus**

## 2.2 Szoftverspecifikáció

### 2.2.1 Az IO-Link kommunikációs paraméterek

A kommunikációt COM2 (38,4 kBd) adatátviteli sebességre állítottam, ami az alkalmazás szempontjából elegendően gyorsnak bizonyult, és ezt a továbbiakban számítással is igazolom.

Az alkalmazás szempontjából fontos, hogy minél több ki- és bemenő folyamat adat kerüljön átvitelre egy IO-Link cikluson belül, ezért a TYPE_2_V típusú M-sequence-et alkalmaztam, mivel ez a típus engedélyezi a legtöbb folyamatadatot. Így a ki- és bemenő folyamat adatok számát 32-32 byte méretűre állítottam, hiszen ez a maximum érték, amit a szabvány megenged.

Az igény szerinti (On-request) adatok az IO-Link protokollnak megfelelően, a paraméterek kezelésének, ill. diagnosztikai céloknak van fenntartva (Page, ISDU, Diagnosztikai csatornák). Egyrészt a konverter esetében nem számíthatunk sűrűn a paraméterek módosítására/lekérdezésére, és a definiált paraméterek hossza is csak néhány byte méretűek. Másrészt a diagnosztikai csatorna csak ritkán kerül használatra, ezért az igény szerinti (On-request) adatok számát 2 byte-ra állítottam. Nagyobb érték definiálása felesleges lenne, hiszen a kommunikációban csak overhead-ként jelentkezne, ill. azt lassítaná.

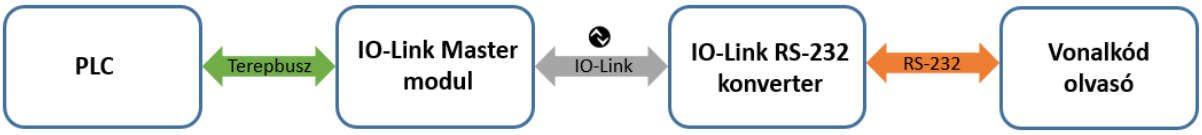
A ciklusidő értékének megadásánál, több szempontot is figyelembe kellett vennem. Számszerűsítve az átvitelre kerülő byte-okat egy ciklusidőn belül, az átvitel sorrendjében, 1 MC, 1 CKT, 32 PDO, 2 OD, 32 PDIN, 1 CKS, amik együttesen 69 byte-ot számlálnak. A kommunikáció sebessége COM2, azaz 38,4 kBd, amiből következik, hogy $\frac{1}{38400} = 26$ µs ideig tart 1 bit átvitele ($T_{bit}$). Egy adatkeret 11 bitből áll (1 start, 8 adatbit, 1 paritásbit, 1 stopbit), így a 69 byte átviteléhez szükséges időtartam $69 * 11 * 26$ µs $= 19734$ µs, ami közelítóleg 20 ms. Továbbá figyelembe kellett vennem azt is, hogy a mikrovezérlő belső EEPROM-jába való írás, időigényes műveletnek számít. A mikrovezérlő adatlapja 4,5 ms időtartamot ír egy EEPROM írás művelet végrehajtására (4 byte méretű memória lapnak), ami idő alatt a programvégrehajtás teljesen megáll, és ez érvényes a beérkező megszakításkérések kiszolgálására is (az IO-Link stack megszakításos elven működik). Ebből kifolyólag, ha több egymást követő EEPROM írás művelet szerepel a kódban, akkor, az hosszú ideig képes blokkolni a programvégrehajtást, ami azt eredményezi, hogy a master által küldött üzenetre, a konverter nem lesz képes a ciklusidőn belül válaszolni. Ennek a problémának az áthidalására az EEPROM-ot kezelő szoftvermodul ugyan tartalmaz megoldást, de szüksége van legalább 12 ms egybefüggő, tétlen időtartamra a ciklusidőn belül. A ciklusidő értékének a megadásakor számításba kellett vennem azt is, hogy a master által küldött üzenetre sem érkezik azonnal válasz a konvertertől, hanem lesz egy rövid, tétlen időtartam a kettő között. Ennek a reakcióidőnek az értékét a szabvány maximum 10 $T_{bit}$ időre korlátozza. Továbbá a master és a konverter is rendelkeznek kis mértékű jitterrel. Ezen okokból kifolyólag, érdemes 2-3 ms időtartam ráhagyással is kalkulálni, mivel így lesz akkora időtartalék a rendszerben, hogy ezek a bizonytalanságok nem okozhatnak problémát a kommunikációban. A ciklusidő értéket, figyelembe véve a leírtakat, $20\ ms + 12\ ms + 3\ ms = 35\ ms$-ra választottam.

Az adatátviteli sebességre is közvetlen befolyással van a ciklusidő értéke, hiszen az minél rövidebb, annál több adat átvitelére képes a rendszer. Az folyamat adatokat tekintve, legyen szó ki, ill. bemenő adatról (hiszen mindkettő értékét 32 byte-ra állítottam), a rendszer

egy másodperc alatt $\frac{1\ s}{35\ ms} * 32\ byte = 914\ byte$ átvitelére képes. Egy vonalkód olvasónak, ill. az RS-232 interfésszel rendelkező eszközök zömének, ez az adatátviteli sebesség elegendő, mivel nem kell folyamatos adatáramlásra számítani (soros kommunikáció eseményvezérelt), és ha adat érkezik az csak maximum néhány 10 byte nagyságrendű.

### 2.2.2 Az IO-Link rendszer fölé rendelt kommunikációs logika

Szükség volt egy olyan IO-Link rendszer fölé implementált protokollra, amivel a konverter működése vezérelhetővé válik. A vezérlést általában egy PLC végzi. A rendszer felépítését a 2.5. ábra szemlélteti. Az ábrán egy példaalkalmazás látható, a konverterhez egy RS-232 interfésszel rendelkező vonalkód olvasó készülék lett csatlakoztatva.




**2.5. ábra: Példaalkalmazás**

A konverter esetében, az IO-Link rendszer fölé rendelt kommunikációs logikának a szükségességére, bemutatok néhány egyszerű esetet. Maradok a 2.5. ábrán bemutatott rendszer példájánál. Tegyük fel, hogy valamilyen vonalkód került beolvasásra, ami 50 byte hosszú adatot tartalmaz. A vonalkód olvasó a kiolvasott adatot egyszerűen továbbküldi a konverternek.

Az IO-Link kommunikáció áttekintésénél bemutatásra került, hogy az IO-Link protokoll ciklikus jellegű, ill. egy cikluson belül maximum 32 byte be- és kimenő folyamat adat kerülhet átvitelre (TYPE_2_V típusú M-sequence engedélyez maximum 32 byte hosszú konfigurációt).

A konverter a szabvány engedte maximum 32 byte méretű bemenő folyamat adattal került konfigurálásra, vagyis ciklikusan (a ciklusidőben definiált időközönként, ami a konverter esetében 35 ms) 32 byte adatot küld a PLC részére. Először is adódik a probléma, hogy valamiféleképpen jelezni kellene a PLC-nek, ha adat érkezik a vonalkód olvasó készüléktől. Nem jó megoldás az, ha a PLC-nek ciklikusan 32 byte 0x00 adatot küldök, és ha a vonalkód olvasótól adat érkezett, akkor magát az adatot küldöm a 0x00 helyett. Hiszen előfordulhat az az eset, hogy a vonalkód olvasótól érkező 50 byte adat tartalmaz 0x00 byte-ot is.

Továbbá a konverter hiába használja ki a szabvány engedte maximum 32 byte hosszúságú folyamat adatoknak az átvitelét, az 50 byte hosszú adatot csak két részletben képes átvinni. Így a konverter az első ciklusban átvisz 32 byte adatot, a második ciklusban pedig a

maradék 18 byte-ot. Azonban a PLC ebben az esetben nem lesz képes megállapítani, hogy a második 18 byte hosszú csomag a korábbi 32 byte-nak a folytatása, vagy ez az adat már egy másik vonalkódtól származik.

Érezhető, hogy a ciklikus IO-Link kommunikáció, és az eseményvezérelt RS-232-es protokoll közötti zökkenőmentes átjáró funkció implementálásához szükség van arra, hogy a PLC és a konverter valamilyen logika felhasználásával kommunikáljon egymással, amely az IO-Link kommunikáció fölé épül.

A helyes megoldás tehát az, hogy a 32 byte-ból kijelölök egy byte-ot, aminek valamely bitjén jelzem a PLC-nek, ha adat érkezett a konvertertől. Egy másik byte-on (esetleg ugyanazon) továbbítom azt, hogy a vonalkód olvasótól beérkezett adat hány byte hosszúságú. Ha ezeket a lépéseket megteszem, mielőtt az adatokat továbbküldeném, a PLC képes lesz eldönteni, hogy egy adat hol kezdődik, ill. végződik.

Az IO-Link rendszerre implementált protokoll, tehát a következőképpen néz ki. A konverter esetében 32-32 byte be- és kimenő folyamat adatot definiáltam, vagyis a konvertertől a PLC-nek továbbított bemenő, ill. a PLC-től a konverternek küldött kimenő ciklikus folyamat adatok hossza megegyezik. A be és kimenő folyamat adatoknál is definiáltam két-két ún. header byte-ot, amik kizárólag a kommunikáció lebonyolításáért felelősek. A header byte-okon adat byte-ok nem továbbíthatóak, ezért ezek a byte-ok a kommunikációban overhead-ként jelentkeznek. A 2.1. táblázat a bemenő folyamat adatok értelmezését szemlélteti.

| Bit / Byte | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 1. Header byte | PW | DIN2 | TO | DIN1 | ER | JE | JA | DA |
| 2. | Adat méret, vagy Hibakód, vagy Adat[1] | | | | | | | |
| 3. – 31. | Adat[2] – Adat[30] | | | | | | | |
| 32. Header byte | PW | DIN2 | TO | DIN1 | ER | JE | JA | DA |

**2.1. táblázat: A bemenő folyamat adatok értelmezése**

A 2.1. táblázatban látható, hogy a bemeneti folyamat adatoknál az első, és az utolsó byte-ok vannak kijelölve header byte-nak, ami nincsen másképpen a kimenő folyamat adatoknál sem. Látható, hogy a header byte-ok bitjeinek külön elnevezésük van, amik különböző funkciókért felelősek. Látható továbbá az is, hogy az első és utolsó header byte-ok bitjei megegyeznek, vagyis ugyanazon header byte redundánsan kétszer is elküldésre kerül egy cikluson belül.

A bemenő folyamat adatoknál a 2. byte-on három féle adat is szerepelhet, ahogyan ez a 2.1. táblázatban is látható. Egyrészt ezen a byte-on kerül definiálásra az átvitelre kerülő adat mérete. Ha a konverternek az RS-232 portján adat érkezik, akkor a beérkezett adat méretét ezen a byte-on közli a konverter a PLC-vel. Másrészt a logika tartalmaz hibafelismerő funkciókat is. Ha konverter valamilyen hibát észlel, akkor egy hibakódot küld a PLC részére, ami szintén a 2. byte-on fog megjelenni. Harmadrészt a byte adatátviteli funkciót is betölthet.

A 3-tól a 31-ig terjedő byte-ok kizárólag adatátviteli célokra használatosak.

Azt, hogy a bemenő folyamat adatoknak az egyes byte-jain éppen milyen adat van jelen, vagy azok egyáltalán hordoznak-e információt, a header byte tartalma szabja meg.

A header byte 1. bitje az ún. DA (Data Available) flag, amellyel a konverter jelzi a PLC-nek, hogy adat érkezett az RS-232 vonalon. A flag 1-es értékű az adat beérkezésének a regisztrálásától, egészen az átvitel befejeztéig.

A header byte 2. bitje az ún. JA (Job Accepted) flag, amely nyugtázza a PLC adatátviteli kérelmeit. A flag 1-es értéket vesz fel az adatátviteli kérvény regisztrálásától, egészen a munka befejeztéig.

A header byte 3. bitje az ún. JE (Job End) flag, amelynek a feladata, hogy jelezze az adatátvitel befejeztét. Ha a kívánt mennyiségű adat átvitelre került, a flag bebillen. Amikor a PLC is nyugtázza, hogy az adatátvitel valóban végét ért (kimenő folyamat adatok header byte-jának a JB flag értéke 0-ra vált), akkor ismét 0 értéket vesz fel.

A header byte 4. bitje az ún. ER (Error) flag, ami valamilyen hiba bekövetkezését jelzi a PLC-nek. Ha konverter hibát észlel, a flag-et bebillenti, és ezzel egyidejűleg a 2. byte-ra egy hibakódot helyez. A flag visszabillentéséhez az eszköz szoftveres újraindítást igényel (kimenő folyamat adatok header byte-jának az RD flag-jén 0→1→0 átmenet generálása).

A header byte 5. és 7. bitjei az ún. DIN1 és DIN2 (Digial Input 1, 2) flag-ek, amelyek az RS-232 oldalon található két digitális bemenet értékeit tartalmazzák. A flag-ek, noha a header byte-ba ágyazottak, az IO-link rendszer fölé rendelt kommunikációs logikától teljesen függetlenek.

A header byte 6. bitje az ún. TO (Toggle Output) flag. Abban az esetben, ha olyan adatátvitel zajlik az IO-Link és RS-232 vonal között, amely nagyobb, mint 30 byte, akkor szükség van a TO flag használatára is. Említettem már, hogy az IO-link vonalon egy cikluson belül maximum 32 byte vihető át. Mivel a két header byte overhead-ként jelentkezik, ezért ha

a hasznos adat hossza nagyobb, mint 30 byte, az már nem fér bele egy ciklusba. Ebben az esetben az adat „felszeletelésére” van szükség, mivel az több IO-Link ciklusban kerül átvitelre. Ha egy adat tehát hosszabb, mint 30 byte, legyen szó küldésről vagy fogadásról, akkor az IO-Link vonalon minden új adatszeletnél a TO flag értéke invertálódik, jelezvén, hogy már az előzőtől különböző adatblokk átvitele zajlik. Például egy 100 byte méretű adat átvitele esetén, a TO flag értéke háromszor (30 + 30 + 30 + 10) invertálódik. A TO flag esetében nem a bit értéke, hanem annak a változása hordozza az információt.

A header byte 8. bitje az ún. PW (Power) flag, amelynek 1-es értéke jelzi, hogy a konverter bekapcsolt állapotban van. Ha a PLC inaktív állapotba vezérli a konvertert, vagyis szoftveresen kikapcsolja (kimenő folyamat adatoknál a header byte-nak az RD flag-je 1-es értéket vesz fel), abban az esetben a flag 0 értékű. A konverter inaktív állapotba juttatása egyúttal az eszköz alaphelyzetbe való állítását is eredményezi.

| Bit / Byte | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 1. Header byte | | TI | | | | RD | | JB |
| 2. | Művelet parancs, vagy Adat[1] | | | | | | | |
| 3. | Adat méret, vagy Adat[2] | | | | | | | |
| 4. – 31. | Adat[3] – Adat[30] | | | | | | | |
| 32. Header byte | | TI | | | | RD | | JB |

**2.2. táblázat: A kimenő folyamat adatok értelmezése**

A 2.2. táblázat a kimenő folyamat adatok értelmezését szemlélteti. Az ábrán látható, hogy a kimenő folyamat adatoknál, hasonlóan a bemenő folyamat adatokhoz, az első és utolsó byte-ok header funkciókat töltenek be.

A kimenő folyamat adatoknál a 2. byte-on két féle adat is szerepelhet, ahogyan ez a 2.2. táblázatban megfigyelhető. Egyrészt a byte adatátviteli funkciót is betölthet. Másrészt, az adatátvitelt megelőzően, ezen a byte-on kerül definiálásra, hogy a PLC milyen irányú adatátvitelt kezdeményez. Az ábrán a művelet parancs elnevezés, az irány meghatározására utal. Olvasás műveletnek nevezem azt, ha a PLC adatot vesz a konvertertől. Ha a konverter adatot fogad az RS-232 vonalon, azt jelzi a PLC részére (DA flag bebillentésével) és csak ezután kezdeményezhet olvasás műveletet a PLC. Írás műveletről beszélek, amikor a PLC adatot küld a konverternek, amit a konverter az RS-232 vonalon továbbküld. A művelet parancs értelmezett értékeit a 2.3. táblázat tartalmazza.

| Parancs | Jelentés |
|---|---|
| 0x01 | Olvasás |
| 0x02 | írás |

**2.3. táblázat: A kimenő folyamat adatokban a művelet parancs értelmezése**

A kimenő folyamat adatoknál a 3. byte-on szintén két féle adat szerepelhet, ahogyan ezt a 2.2. táblázat mutatja. Egyrészt ezen a byte-on kerül definiálásra az átvitelre kerülő adat mérete. Ha a PLC írás műveletet kezdeményez, akkor az elküldendő adat méretét ezen a byte-on közli a konverterrel. Másrészt a byte adatátviteli funkciót is betölthet.

A 4-től a 31-ig terjedő byte-ok kizárólag adatátviteli célokra használatosak.

Hasonlóan a bemenő folyamat adatokhoz, azt, hogy a kimenő folyamat adatoknak az egyes byte-jain éppen milyen adat van jelen, vagy azok egyáltalán hordoznak-e információt, a header byte tartalma szabja meg.

A header byte 1. bitje az ún. JB (Job) flag, amellyel a PLC az adatátvitelt kezdeményezi. Olvasás és írás műveleteknél is az adatátvitel elindítása, ennek a flag-nek a bebillentésével vezérelhető. A PLC az adatátvitel befejezésének a sikerességét/nyugtázását a flag visszabillentésével jelzi a konverter részére.

A header byte 3. bitje az ún. RD (Reset Device) flag, amellyel a konverter inaktív állapotba vezérelhető. A flag-gel szoftveres újraindítás végezhető, aminek hatására alaphelyzetbe kerül a konverter. Ha a konverter valamilyen hibát észlelt, amelyet jelzett a PLC-nek (bemenő folyamat adatoknál a header byte-jának az ER flag értéke 1, és ezzel egyidejűleg a 2. byte-on a hibakód szerepel), az minden esetben újraindítást igényel.

A header byte 7. bitje az ún. TI (Toggle Input) flag, amely azonos a bemenő folyamat adatoknál megismert TO (Toggle Output) flag funkcionalitásával. Ha egy adat nagyobb méretű, mint 30 byte, abban az esetben 30 byte-os blokkokra kell „felszeletelni”. Ezek a blokkok több részletben kerülnek átvitelre az IO-Link vonalon. Minden új blokk küldésekor/fogadásakor a TI flag értéke invertálódik, jelezvén, hogy az előzőtől eltérő adatblokk átvitele zajlik. A TI flag esetében nem a flag értéke, hanem annak a változása hordozza az információt.

Konkrét példát írásra, olvasásra és hibadetektálásra a következő fejezetekben mutatok be. A példák bemutatásánál kerül magyarázatra, hogy a byte-ok és bitek, mikor, milyen értéket vesznek fel az adatátvitel során. A kommunikáció handshaking jellegű, a PLC és a konverter

közlik egymással az állapotváltozásaikat, és a saját állapotgépükben addig nem lépnek tovább egy következő állapotba, amíg a másik fél a változást nem nyugtázta.

### 2.2.3 Olvasás művelet

Maga az olvasás, mint művelet, a vezérlés, azaz a PLC szemszögéből tekintendő. A PLC az olvasás parancs kiadásakor, a konverternek az RS-232 vonalán fogadott adatainak a továbbítását kezdeményezi.

A kommunikáció a PLC és a konverter között handshaking jellegű. A PLC és a konverter is a saját állapotgépeikben ugrálnak egyik állapotból a másikba, és ezeket a változásokat jelzik is a másik félnek. Az aktuális állapotaikat a header byte-okon közlik egymással, és addig egyik fél sem léphet tovább egy új állapotba, amíg a másik fél ezt nem nyugtázta.

A következőkben két féle olvasási műveletet fogok bemutatni. A fő különbség a két olvasási folyamat között, hogy az első eset 30 vagy annál kevesebb byte adatátvitelére, a második eset pedig 30 byte-nál nagyobb méretű adatok átvitelére vonatkozik. A 30 byte-nál nagyobb méretű adatátvitel esetén, az adat több IO-Link ciklusban, blokkokra felosztva kerül átvitelre, míg a 30 vagy annál kisebb méretű adatok egy IO-Link cikluson belül.

A PLC és a konverter állapotváltozásait, és a közöttük zajló handshaking kommunikációt az egymásnak küldött ki és bemenő folyamat adatokon keresztül fogom szemléltetni.

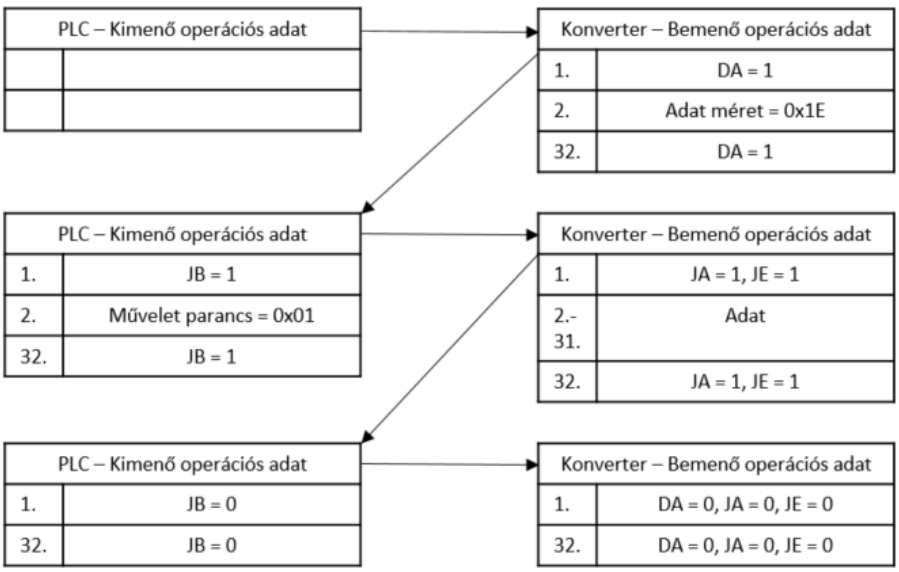
A 2.6. ábra a rendszer alaphelyzetben lévő állapotát szemlélteti. Alaphelyzetben nem zajlik adatátvitel, ezért a header byte-okat leszámítva, a 2-tól a 31-ig terjedő byte-okat nem ábrázoltam, mivel az értékük irreleváns. Látható, hogy a PLC-nek az RD és JB flag-jei 0 értékűek. Az RD (Reset Device) flag 0 értékével a PLC aktív állapotba (engedélyező jel) vezérli a konvertert. A JB (Job Available) flag 0 értéke a tétlen állapotot jelzi, azaz a PLC nem kezdeményez adatátvitelt. A konverter válaszképpen bebillenti a PW (Power) flag-jét, amivel jelzi, hogy működésre kész állapotban van.

| PLC – Kimenő operációs adat | |
|---|---|
| 1. | RD = 0, JB = 0 |
| 32. | RD = 0, JB = 0 |

| Konverter – Bemenő operációs adat | |
|---|---|
| 1. | PW = 1 |
| 32. | PW = 1 |

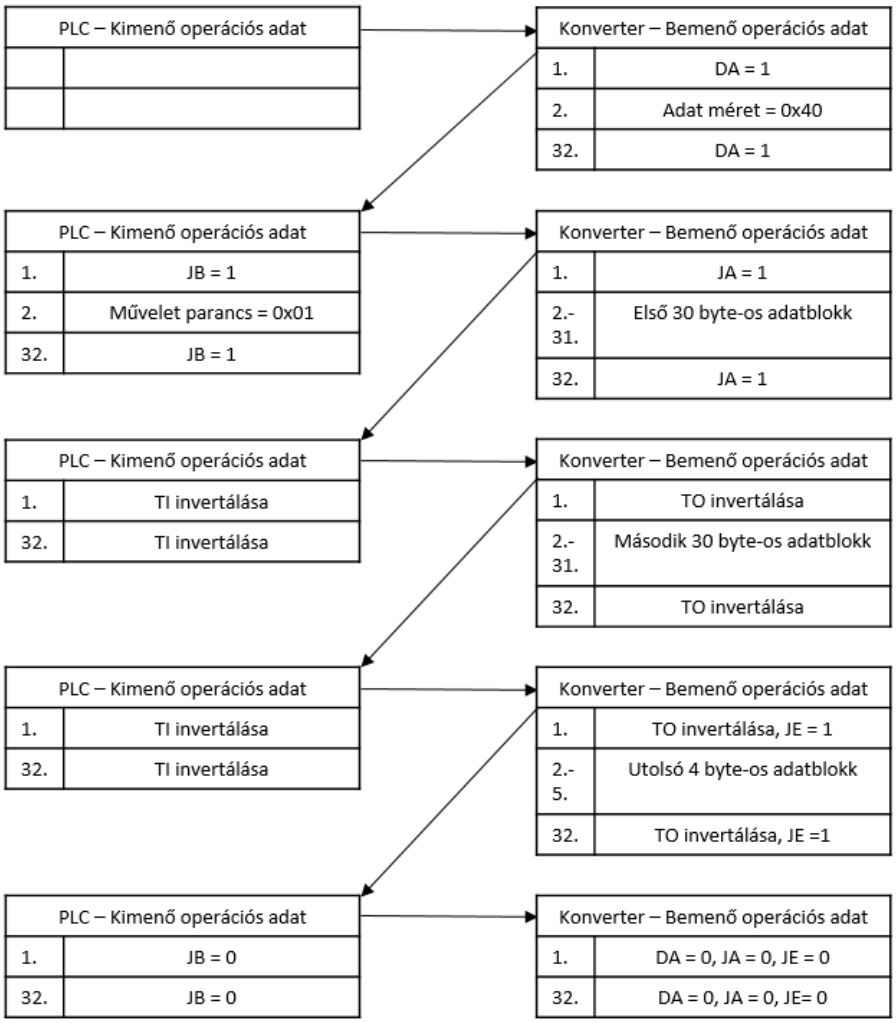
**2.6. ábra: A ki és bemenő folyamat adatok az indítást követő, ill. tétlen fázisban**

A 2.7. ábra egy olyan olvasás művelet végrehajtását szemlélteti, ahol egy 30 byte méretű adatcsomag kerül átvitelre. A példa érvényes minden olyan esetre, amelynél 30 vagy annál kisebb méretű adat kerül átvitelre a konvertertől a PLC-nek. Olvasás parancsot a PLC abban az esetben kezdeményezhet (ellenkező esetben a konverter hibát jelez), ha a konverter jelzi, hogy adatot fogadott az RS-232 vonalon, amelyet szeretne továbbítani az IO-Link vonalán a PLC-nek. A 2.7. ábrán is megfigyelhető, hogy a konverter az RS-232 vonalán érkező adatok regisztrálásakor a DA (Data Available) flag-jének a bebillentésével jelzi, hogy kiszolgálást igényel a PLC-től. Továbbá ezzel egyidejűleg a 2. byte-on közli azt is, hogy mekkora méretű adatot kíván továbbítani, ami jelen esetben 30 byte (30 = 0x1E). Válaszképpen a PLC bebillenti a JB (Job) flag-jét és a 2. byte-ra a 0x01 adatot helyezi, ami az olvasás parancsot reprezentálja. A konverter a JB flag 1-es értékének regisztrálásakor megvizsgálja a 2. byte-ot, hogy azon milyen parancs szerepel. Az olvas parancs értelmezését követően bebillenti a JA (Job Accepted) flag-jét, amivel jelzi, hogy elfogadta a parancsot, és annak végrehajtásába kezd. Mivel az RS-232 vonalon beérkezett adat mérete nem haladja meg a 30 byte-ot, ezért a munka elfogadásával egyidejűleg bebillenti a JE (Job End) flag-jét is, hiszen egy cikluson belül képes a teljes adatmennyiséget átvinni. Az adatokat a konverter a 2-tól a 31-ig terjedő byte-okon jeleníti meg. A PLC az adatok átvitelét követően a JB flag értékét visszabillenti, amivel nyugtázza a konverternek, hogy az adatátvitel a részéről is befejezettnek tekintendő. Válaszképpen a konverter alaphelyzetbe állítja a header byte-jának bitjeit, vagyis a DP, JE, JA flag-eket visszabillenti.




**2.7. ábra: 30 byte méretű adat olvasására példa**

A 2.8. ábrán egy olyan olvasás művelet végrehajtása látható, ahol már egy 64 byte méretű adatcsomag kerül átvitelre. A példában bemutatott logika érvényes minden olyan esetre, amelynél 30 byte-nál nagyobb méretű adat kerül átvitelre a konvertertől a PLC-nek. Mint az korábban említésre került, egy IO-Link cikluson belül maximum 30 byte adat vihető át, így 30 byte-nál nagyobb méretű adatok esetén, azoknak a „felszeletelésére" van szükség. 64 byte méretű adatnál 3 IO-Link ciklus szükséges a teljes adat átviteléhez. A 2.8. ábrán látható, hogy hasonlóan az előző példához a DA flag bebillenése jelzi a PLC-nek, hogy a konverter adatot fogadott az RS-232 vonalon, és a 2. byte-on megjelenik az adat mérete. Ennek a hatására a PLC bebillenti a JB flag-et és kiadja a 2. byte-on az olvasás parancsot (0x01). A konverter az olvasás parancsra reagálva bebillenti a JA flag-jét, és kihelyezi a 2-tól a 31-ig terjedő byte-okra az adat első 30 byte-os szeletét. A PLC válaszul arra, hogy az adat első blokkját átvette, invertálja a TI (Toggle Input) flag-jét. Ezt érzékelve a konverter kihelyezi az adat második 30 byte-os szeletét a 2-től 31-ig terjedő byte-okra és invertálja a saját TO (Toggle Output) flag-jét, hogy jelezze, hogy az adat byte-okon már új adatok találhatóak. A PLC a második adatszelet kézhezvételekor ismét invertálja a TI flag-jét, amivel jelzi, hogy készen áll az újabb adatblokk fogadására. A konverter ezt észlelve invertálja a saját TO flag-jét és a 2-tól az 5-ig terjedő byte-okra kihelyezi az utolsó 4 byte-os adatblokkot. Mivel az utolsó adatblokkról van szó, ezért a JE flag-et is bebillenti, ami az adatátvitel végét jelenti. A konverter az adatátvitel kezdetét megelőzően közölte a PLC-vel, hogy mekkora méretű adatot kíván átvinni (DA flag bebillenésekor a 2. byte-on), így a PLC képes lesz felismerni, hogy a 2-tól az 5-ig terjedő byte-ok tartalmaznak csak hasznos adatot. Végül a PLC az adatátvitel befejezésének a nyugtázását, a JA flag visszabillentésével jelzi, amire a konverter válaszképpen alaphelyzetbe állítja a flag-jeit.




**2.8. ábra: 64 byte méretű adat olvasására példa**

### 2.2.4 Írás művelet

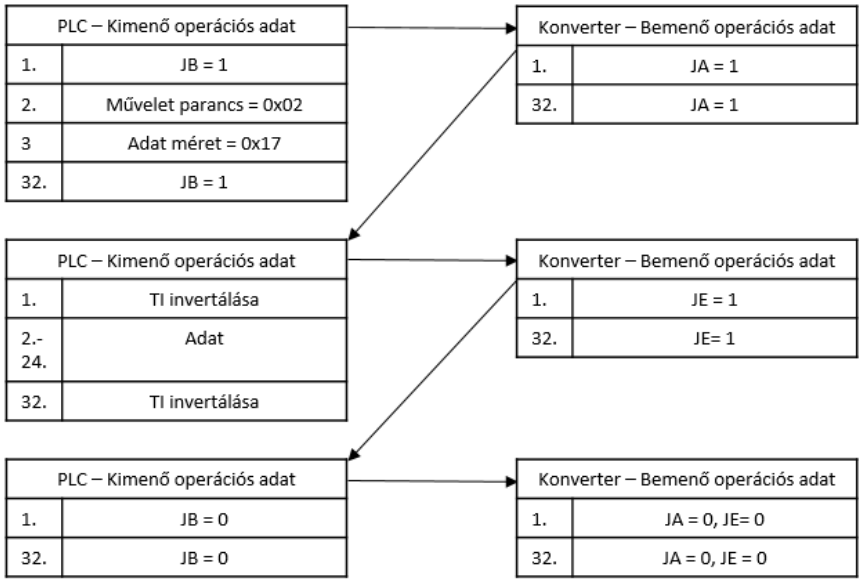
Az írás művelet az olvasással ellentétes irányú adatátvitelért felelős. A rendszer itt is a PLC szemszögéből tekintendő, vagyis írás művelet alatt, az adatoknak a PLC-től a konverter RS-232 vonalára csatlakozó eszközhöz való eljuttatását értem.

A kommunikáció az írás műveletnél is handshaking jellegű. A megismert módon a PLC és a konverter az állapotváltozásaikat az folyamat adatokban kijelölt header byte-okon tudatják egymással.

A következőkben két féle írási folyamatot fogok bemutatni. A fő különbség a két írási művelet között, hogy az első eset 30 vagy annál kevesebb byte adatátvitelére, a második eset pedig 30 byte-nál nagyobb méretű adatok átvitelére vonatkozik.

A 2.9. ábra egy olyan írási műveletet szemléltet, ahol kevesebb, mint 30 byte méretű adatátvitel zajlik. Az adatátvitelt, ellentétben az olvasás művelettel, a PLC kezdeményezi. A

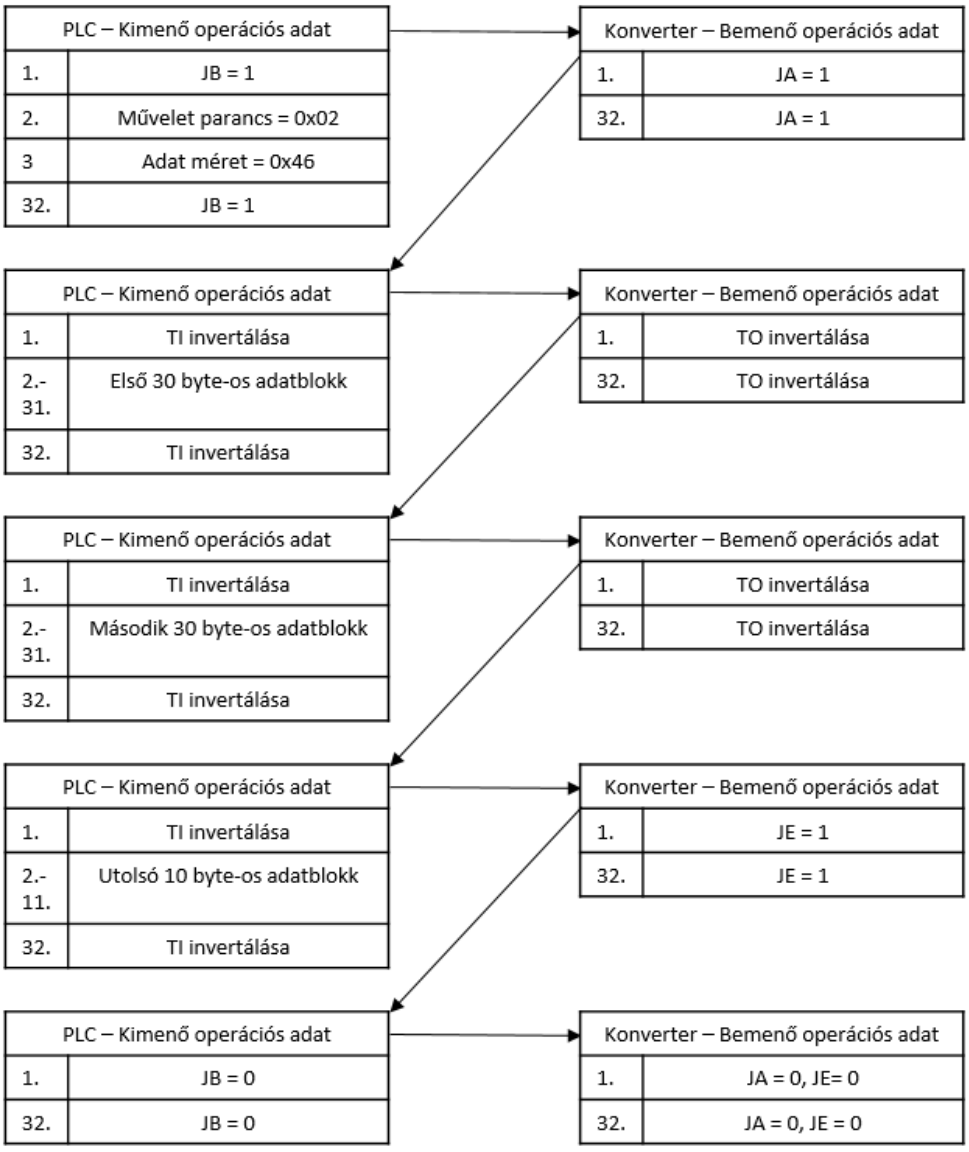
PLC az adatátvitel kezdetét a JB (Job) flag bebillentésével jelzi, és a 2. byte-ra az írás parancsot reprezentáló 0x02 adatot, a 3. byte-ra pedig az átvinni kívánt adat méretét (23 = 0x17) helyezi. A konverter a JB (Job) flag bebillenésének észlelésekor megvizsgálja, hogy a 2. byte-on milyen művelet definiált. Miután megállapította, hogy írás műveletről van szó, elmenti a 3. byte tartalmát, amiből megtudja az átvitelre kerülő adat méretét. Jelzésként a PLC-nek a konverter bebillenti a JA (Job Accepted) flag-jét. A PLC erre reagálva, a példában 23 byte méretű adatot a 2-tól a 24-ig terjedő byte-okra helyezi, és az adatok jelenlétét a TI (Toggle Input) flag invertálásával jelzi. A konverter, mivel korábban már eltárolta az adat méretét, átveszi a 23 byte méretű adatot a megfelelő adat byte-okról, továbbítja azokat soros kommunikációval az RS-232 vonalon, és bebillenti az átvitel befejezését jelző JE (Job End) flag-et. A PLC válaszul visszabillenti a JB (Job) flag-jét és befejezettnek tekinti az átvitelt. A konverter ezt nyugtázva visszaáll az alaphelyzetébe, azaz visszabillenti a JE (Job End), és a JA (Job Accepted) flag-eket.




**2.9. ábra: 23 byte méretű adat írására példa**

A 2.10. ábra olyan írás műveletet szemléltet, ami nagyobb, mint 30 byte méretű adatátvitel lezajlását szemlélteti. Az adatátvitel hasonlóképpen kezdődik, mint az előző példában. Látható, hogy a PLC bebillenti a JB (Job) flag-jét, a parancs byte-jára az írást reprezentáló 0x02 adatot helyezi, és közli az adat méretét a 3. byte-on. Ezután a konverter bebillenti a JA (Job Accepted) flag-jét, amivel jelzi, hogy észlelte a kérést. Az átvitelre szánt adat 70 byte méretű, ami 3 IO-Link ciklusban továbbítható (30+30+10). A PLC az első 30 byte méretű adatblokkot a 2-tól a 31-ig terjedő adat byte-okra helyezi és invertálja a TI (Toggle Input) flag-jét. A TI flag értékének változására reagálva a konverter eltárolja az első 30 byte-os

adatszeletet, és a következő blokk továbbítását igényli a TO (Toggle Output) flag invertálásával. A PLC az adat byte-okra kihelyezi a második adatblokkot és ezt jelzi a TI (Toggle Input) flag-jének az invertálásával. A konverter a második szeletet is eltárolja és invertálja a TO flag-et. A PLC a maradék 10 byte méretű blokkot a 2-tól a 11-ig terjedő byte-okra helyezi és ismét invertálja a TI flag-jét. A konverter az adat méretének az ismeretében tudja, hogy hasznos adatok csak a 2-től a 11-ig terjedő byte-ok tartalmaznak, ezért csak azoknak a tartalmát menti el. A konverter a következő lépésben összeilleszti az átvitt adatblokkokat az átvitel sorrendjében (Adat[1]- Adat[30] + Adat[1]- Adat[30] + Adat[1]- Adat[10] ), és soros kommunikációval elküldi azokat az RS-232 vonalon. Jelezve a PLC-nek a feladat teljesítését, bebillenti a JE flag-et. A PLC ezt nyugtázva visszabillenti a JB flag-jét, aminek hatására a konverter visszaáll alaphelyzetébe, azaz visszabillenti a JE, JA flag-eket.




**2.10. ábra: 70 byte méretű adat írására példa**

### 2.2.5 Hiba diagnosztikai funkciók

A konvertert hiba diagnosztikai funkciókkal is elláttam. A header byte-ban (bemenő folyamat adatok) található egy ER (Error) flag, ami a hiba bekövetkezését hivatott jelezni. Az ER flag bebillenésekor a 2. byte hibakód átvitelére van kijelölve. A PLC-nek hiba fellépésekor a konvertert minden esetben szoftveresen újra kell indítania, hogy az ismét alaphelyzetbe kerüljön.

A konvertert a 2.4. táblázatban felsorolt hibafelismerő funkciókkal láttam el, amikhez a következő hibakódokat rendeltem.

| Hiba típusa | Hibakód |
|---|---|
| Nincsen elérhető adat | 0x01 |
| A header byte-ok különbözőek | 0x02 |
| Érvénytelen parancs | 0x03 |
| Az adat mérete 0 | 0x04 |
| Írás parancs megszakítva | 0x05 |
| Olvasás parancs megszakítva | 0x06 |
| Adat túlcsordulás | 0x07 |
| RS-232 vonalon paritás hiba | 0x08 |
| RS-232 vonalon zaj hiba | 0x09 |
| RS-232 vonalon keret hiba | 0x0A |
| RS-232 vonalon sikertelen adattovábbítás | 0x0B |

**2.4. táblázat: A bemenő folyamat adatoknál értelmezett hibakódok**

A 0x01 kódú hiba akkor generálódik, ha a PLC olvasás műveletet kezdeményez, de a konverter nem fogadott beérkező adatokat az RS-232 vonalon, ezért nem áll rendelkezésére adat, amit továbbíthatna.

A 0x02 kódú hibát akkor jelez a konverter, ha a PLC által küldött kimenő folyamat adatokban, a header funkcióknak kijelölt byte-ok tartalma nem egyezik.

A 0x03 kódú hiba akkor áll fenn, ha a PLC bebillenti a JB (Job) flag-jét, de az ilyenkor parancs byte-ként funkcionáló 2. byte-on érvénytelen adat szerepel. Az értelmezett értékeket lásd a 2.3. táblázatban. A konverter ebben az esetben nem tudja, hogy a PLC milyen műveletet (olvasás, írás) akar végrehajtani, ezért hibát jelez.

A 0x04 hibakód esetén, a PLC a JB (Job) flag-jének bebillentésével adatátvitelt kezdeményez, de a 3. byte-on, ami ilyenkor az átvinni kívánt adat méretét tartalmazza, a 0x00 adat szerepel, vagyis az 0 byte méretű adatátvitelt jelöl.

A 0x05 kódú hibát akkor jelez a konverter, ha egy folyamatban lévő írás műveletet a PLC megszakított, vagyis a JB (Job) flag-jét az adatátvitel befejezése előtt visszabillenti.

A 0x06 kódú hibát abban az estben generál a konverter, ha egy folyamatban lévő olvasás műveletet szakított félbe a PLC. Azaz hasonlóan az előző esethez, a JB flag itt is a folyamat befejezése előtt billen vissza.

Az eddig bemutatott hibajelenségek, a rosszul implementált PLC szoftverből fakadnak. Ezekkel a hibajelzésekkel a konverter támogatást nyújt a PLC programozójának. A további hibajelzéseknek a forrásai már a rendszer más részeiben keresendők.

A 0x07 hibakódot a konverter akkor generál, ha az RS-232 vonalon beérkező adatokhoz rendelt tároló túlcsordul. A tároló kapacitását 255 byte-ra maximalizáltam, így ez a hibajelenség abban az esetben fordulhat elő, ha több, mint 255 byte adat érkezik be, mielőtt a korábban beérkező adatokat kiolvasta volna a PLC.

A 0x08 hibakódot a konverter akkor jelez, ha az RS-232 vonalon a soros kommunikáció során paritás hibát észlel.

A 0x09 hibakód estén a konverter zajos RS-232 vonalat jelez.

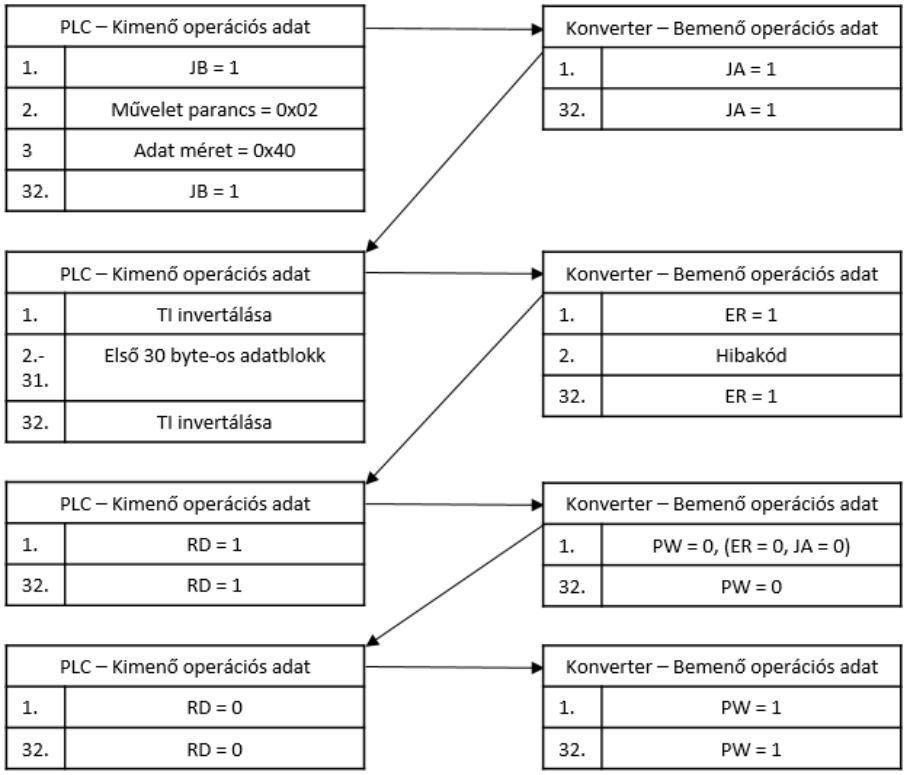
A 0x0A hibakódnál a konverter az RS-232 vonalon a soros kommunikáció során kerethibát érzékel.

A 0x0B kódú hiba akkor következik be, ha valamilyen okból kifolyólag a konverter nem volt képes elküldeni a PLC-től érkező adatot az RS-232 vonalon, noha a PLC és a konverter között az adatátvitel szabályosan lezajlott.

A paritás, zaj, keret hibákat, valamint az adat sikertelen elküldésének a jelzését, a mikrovezérlő képes felismerni, ezért ezeket a hibákat is felvettem a hiba diagnosztikai listára.

A 2.11. ábra egy olyan adatátvitelt szemléltet, ahol a konverter a folyamat közben hibát diagnosztizált. Az ábrán látható, hogy a PLC egy írás műveletet kezdeményezett, azonban a konverter az átvitel során valamilyen hibát észlelt, és az átvitelt felfüggesztve az ER flag-jét bebillentette. A PLC válaszként, a konverter ER flag-jének kinullázása céljából az RD (Reset Device) flag-jét bebillenti. A konverter erre reagálva a PW (Power) flag-jét 0-ba állítja, és

inaktív állapotba kerül. Ezt követően a PLC visszabillenti az RD flag-et, aminek a hatására a konverter alaphelyzetbe áll.




**2.11. ábra: Hiba diagnosztizálása adatátvitel során**

## 2.2.6 Konfigurációs lehetőségek

A konverternek számos beállítási lehetőséget kell, hogy tartalmazzon, hiszen alkalmas kell, legyen a változatos RS-232-es eszközök zömének lekezelésére. Ilyen paraméterek az RS-232-es átviteli sebesség, stop bitek száma, paritások, stb.

A konverter paramétereit, az IO-Link kommunikációnak megfelelően, az aciklikus ISDU csatorna kezeli. A paraméterek ennek megfelelően, az IO-Link kommunikáció bemutatásában leírtak szerint, egy index és subindex címzési metódussal érhetőek el. Azaz, minden paraméterhez tartozik egy index és subindex érték.

A paramétereket továbbá nem felejtő módon menteni is szükséges, mivel a konverternek az újraindulást követően is, a már korábban beállított konfigurációval kell működnie.

A paramétereknek két csoportját különböztetjük meg. Vannak az ún. identifikációs paraméterek, amelyek kötelező jellegűek, és vannak az eszköz paraméterek, amik az adott eszközre jellemzőek.

Az identifikációs paraméterek tehát minden eszköznél kötelező jellegűek, és a címek is adottak. Így, ha kezünkbe akad egy ismeretlen IO-Link-es eszköz, tudni fogjuk, hogy melyik címeken kell az eszköz ID-t, a gyártót, és egyéb identifikációs paramétereket keresni. A konverter identifikációs paramétereit a 2.5. táblázat tartalmazza.

| **DPP** | **ISDU** | | **Paraméter** | **Adat mérete** | **Jogosultság** | **Értéke (alaphelyzetben)** |
|---|---|---|---|---|---|---|
| **Index** | **Index** | **Subindex** | | | | |
| 0x07 | | 0 | Forgalmazó ID | 2 byte | Olvasás | 0x0378 |
| 0x08 | | 0 | | | | |
| 0x09 | | 0 | Eszköz ID | 3 byte | | 0x0504xx |
| 0x0A | | 0 | | | | |
| 0x0B | | 0 | | | | |
| | 0x10 | 0 | Forgalmazó név | 8 byte | | BALLUFF |
| | 0x11 | 0 | Forgalmazó elérhetősége | 16 byte | | www.balluff.com |
| | 0x12 | 0 | Termék név | 20 byte | | BNI IOL-760-002-E066 |
| | 0x13 | 0 | Termék ID | 7 byte | | BNI00xx |
| | 0x14 | 0 | Termék leírás | 16 byte | | IO-Link to RS-232 Converter |
| | 0x15 | 0 | Sorozat szám | 16 byte | | 0x00 |
| | 16hex | 0 | Hardver verzió | 1 byte | | 0x00 |
| | 17hex | 0 | Szoftver verzió | 1 byte | | 0x00 |
| | 18hex | 0 | Application-specific tag | 32 byte | Írás, Olvasás | 0x00 |

**2.5. táblázat: A konverter identifikációs paraméterei**

Látható, hogy az Application specific tag paramétert leszámítva csak olvasás jogosultság értelmezett, hiszen ezek a paraméterek a konverter azonosítására szolgálnak. Az Application specific tag funkciója az, hogy a felhasználónak a rendelkezésére bocsájt 32 byte méretű tárhelyet (ASCII karakternek értelmezendők), amit szabadon felhasználhat. Rendszerint az azonos típusú eszközök közötti, felhasználói szemmel egyszerűbb azonosítási szerepet tölti be (hiszen a sorozatszám is egyértelműen azonosítja az eszközt). A paraméter értékek oszlopban, az x-el kitöltött mezőknél még az értékek nem kerültek meghatározásra.

A konverterre jellemző paraméterek listáját a 2.6. táblázat tartalmazza.

| ISDU | | Paraméter | Adat mérete | Jogosultság | Értéke alaphelyzetben |
|---|---|---|---|---|---|
| Index | Subindex | | | | |
| 0x40 | 0<br>1-4 | RS-232 paraméterek | 4 byte | Olvasás/Írás | - |
| 0x45 | 0 | Kimenő 24V engedélyezése | 1 byte | Olvasás/Írás | 0x00 |
| 0x46 | 0 | Eltelt üzemidő | 4 bytes | Olvasás | - |
| 0x47 | 0 | Újraindítások száma | 4 byte | Olvasás | - |
| 0x52 | 0<br>1-4 | Hőmérséklet adatok | 5 byte | Olvasás | - |

**2.6. táblázat: A konverter paraméterei**

A paraméterekhez tartozó címek (index, subindex) kiosztását illetően, az IO-Link szabvány szabad kezet nyújt a fejlesztőnek, azonban igyekeztem olyan címeket választani, amik más termékek esetén is használatban vannak, ill. az olyan funkciók esetén, amik már más eszközben is léteznek (eltelt üzemidő, újraindítások száma, hőmérsékleti adatok), azoknál értelem szerűen ugyanazokat a címeket használtam.

Az RS-232 paraméterekről részletesebb leírást a 2.7. táblázat tartalmaz. Az RS-232 vonalon a soros kommunikáció paramétereinek a konfigurálási lehetősége, egy nagyon fontos szempont, hiszen a konverternek képesnek kell lennie az RS-232 interfésszel rendelkező eszközök zömének a kezelésére.

| Byte | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Subindex | 1 | 2 | 3 | 4 |
| Leírás | Adatátviteli sebesség | Stop bitek száma | RS-232 Adatkeret opciók | Hardware Flow Control |

**2.7. táblázat: RS-232 paraméterek**

A 2.8. táblázatban megfigyelhető, hogy mely RS-232 paraméter, mely subindexen érhető el. Azoknál az indexeknél, amelyek alá subindexek is tartoznak, a specifikáció szerint a 0-ás subindexen, az összes paramétert egyszerre elérhetővé kell tenni, ezért látható az első

sorban, hogy az egyes tulajdonságok melyik byte-okon helyezkednek el (ill. így az adott paraméter mérete is látható).

Az 1-es subindexen szereplő adatátviteli sebességnél a következő felsorolásban szereplő értékekre konfigurálható a konverter. Az adatátviteli sebességek mellett szereplő értékek reprezentálják az adott sebességet. Például, ha a 40-es index 1-es subindex címen a 0x04-es érték található, akkor a konverter RS-232 vonala 115200 bit/s adatátviteli sebességre van felkonfigurálva.

- 0x00 – 9600 bit/s
- 0x01 – 19200 bit/s
- 0x02 – 38400 bit/s
- 0x03 – 57600 bit/s
- 0x04 – 115200 bit/s (alaphelyzetben)

A 2-es subindexen a stop bitek száma állítható, ahol a következő értékek definiáltak:

- 0x01 – 1 stop bit (alaphelyzetben)
- 0x02 – 2 stop bit

A 3-as subindexen az adatkeret opcióknál az adatbitek száma, és a paritás állítható be. Praktikusabb lett volna az adatbitek számát és a paritást külön subindexekre helyezni, azonban ezt nem tehettem meg, mert a mikrovezérlő meghatározza, hogy 7 bites adathossznál kötelező paritást használni, 8 bitnél már a használata választható, 9 bitnél pedig nem elérhető funkció. Ezért döntöttem a következő felsorolásban látható opciók mellett, mivel így maximálisan kihasználom a mikrovezérlő nyújtotta lehetőségeket.

- 0x00 – 7 bit, Páros
- 0x01 – 7 bit, Páratlan
- 0x02 – 8 bit, Nincs (Alaphelyzetben)
- 0x03 – 8 bit, Páros
- 0x04 – 8 bit, Páratlan
- 0x05 – 9 bit, Nincs

A 4-es subindexen a Hardware Flow Control paraméter állítható be, amelynek a lehetséges értékei a felsorolásban láthatóak:

- 0x00 – RTS és CTS (Alaphelyzetben)
- 0x01 – RTS
- 0x02 – CTS
- 0x03 – Nincs

A 2.6. táblázatban látható, hogy a 45-ös index és 0-ás subindex címen az RS-232 oldalra a külső 24V-os tápfeszültség kapcsolása engedélyezhető, ill. tiltható. A paraméter értelmezett értékei az alábbiak:

- 0x00 – Nincs engedélyezve (Alaphelyzetben)
- 0x01 – Engedélyezve

A további paraméterek a konverter működését érdemben nem befolyásolják, ezek csak diagnosztikai célokat szolgálnak. Ebből kifolyólag a felhasználó csak olvasási jogosultsággal rendelkezik felettük.

A 46-os index és 0-ás subindex címen lekérdezhető a konverter első bekapcsolásától számított, eltelt üzemidő értéke másodpercekben mérve.

A 47-es index és 0-ás subindex címen a konverter újraindításainak a száma olvasható ki.

Az 52-es indexen a hőmérsékleti adatok értékei kérdezhetőek le. Az egyes subindexekről kiolvasható paramétereket a 2.8. táblázat szemlélteti.

| **Byte** | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **Subindex** | 1 | 2 | 3 | 4 | 5 |
| **Leírás** | Aktuális hőmérséklet | Maximum hőmérsékleti érték az újraindítást követően | Minimum hőmérsékleti érték az újraindítást követően | Maximum hőmérsékleti érték az első bekapcsolástól | Minimum hőmérsékleti érték az első bekapcsolástól |

**2.8. táblázat: Hőmérsékleti adatok**

## 2.2.7 A LED-eken definiált jelzések

A hardvernél ismertetésre került, hogy a konverter rendelkezik 2 LED-del, amelyek közvetlenül a felhasználónak küldenek visszajelzést a konverter aktuális állapotáról. Az egyik LED az IO-Link kommunikáció, a másik a tápfeszültség állapotáról tájékoztatja a felhasználót. A definiált jelzéseket a 2.9. táblázat tartalmazza.

| LED | Jelzés | Funkció |
|---|---|---|
| IO-Link kommunikáció | Folytonos zöld | Nincs kommunikáció |
| | Negatívan pulzáló zöld (1 Hz) | Kommunikáció aktív |
| | Kikapcsolt | A modul nem áll feszültség alatt |
| Tápfeszültség | Folytonos Zöld | A modul feszültség alatt áll |
| | Lassan pulzáló zöld (1 Hz) | Tápfeszültség értéke túl alacsony (< 18V) |
| | Kikapcsolt | A modul nem áll feszültség alatt |
| | Gyorsan pulzáló zöld (2 Hz) | A kapcsolható külső 24V-os tápfeszültségen rövidzár áll fenn |

**2.9. táblázat: A LED-eken definiált jelzések**

## 2.3 Szoftver terv

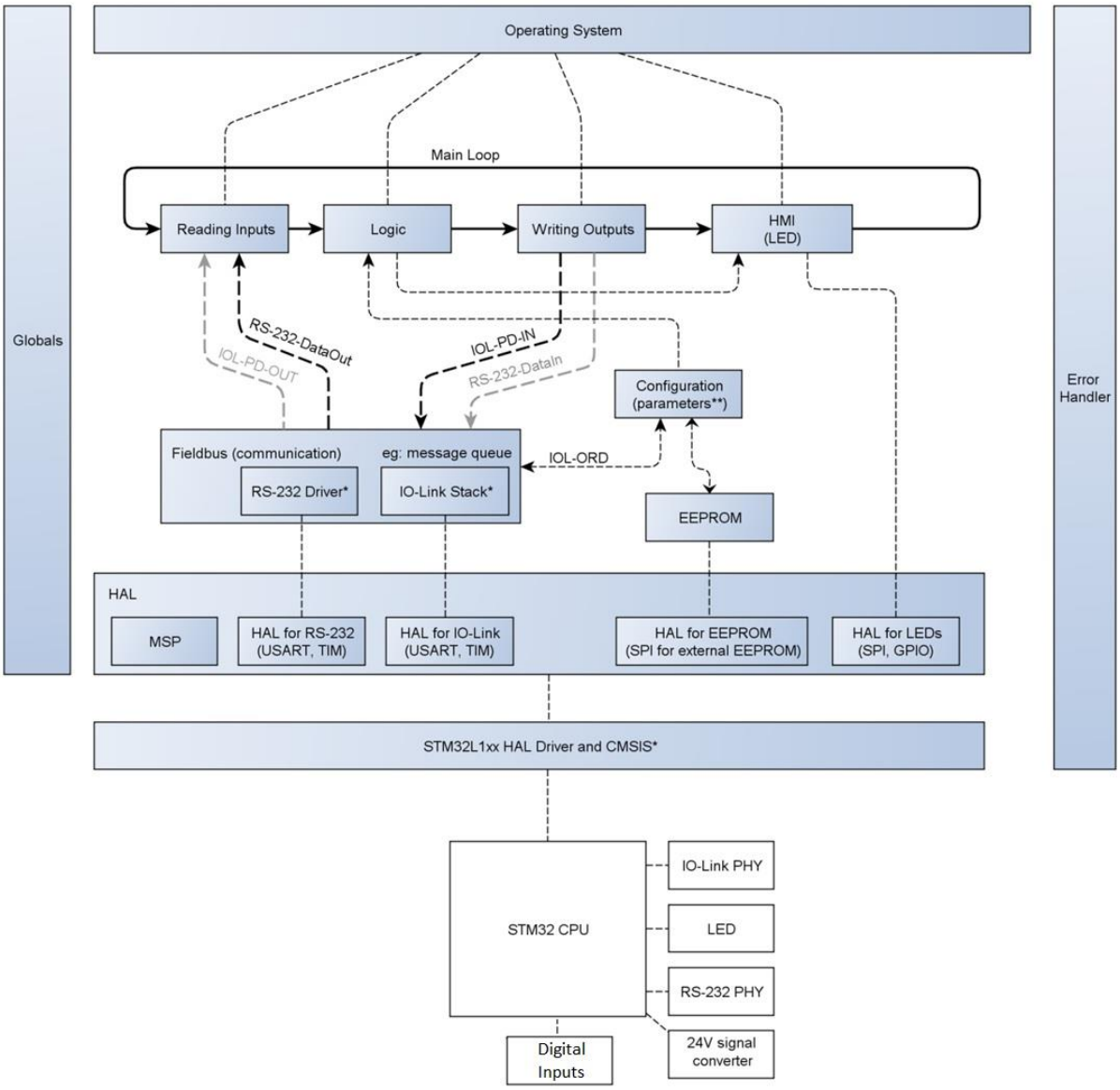
A szoftvert nem a nulláról kellett kifejlesztenem. A cégnek van egy kifejezetten az IO-Link slave eszközökhöz kifejlesztett szoftver sablonja, ami egy nagyon jó kiindulási alapot nyújt. A sablon tartalmaz egy IO-Link stack-et, és egy aköré felépített szoftver architektúrát.

Az IO-Link stack az IO-Link kommunikációt valósítja meg a szabványban definiált módon. A stack teljes egészében egy cégen belüli fejlesztés eredménye, nekem kizárólag az interfész függvényeivel kellett megismerkednem. Segítségével a programozó magas szinten programozhat, biztosít minden logikai csatornához hozzáférést, és azokat a megfelelő időzítésekkel kezeli. A stack tartalmaz egy konfigurációs célú header állományt (iol_cfg.h), amiben a programozó az IO-Link kommunikáció paramétereit define-ok értékeinek a módosításával állíthatja be. Itt állítottam be a szoftver specifikációban megadott paramétereket, vagyis, hogy a kommunikáció TYPE_2_V típusú M-sequence-et használjon, 32-32 byte méretűek legyenek a ki- és bemenő folyamat adatok, 2 byte legyen az igény szerinti adatok (On-request) száma, ill. a minimális ciklusidő értéke 35 ms legyen.

A sablon továbbá útmutatást nyújt a szoftver strukturális felépítésére vonatkozóan. Ez a követendő szoftver architektúra előnyös, mert a biztonságos kódolás jegyében készült, könnyen átláthatóvá teszi a rendszert, és a cégen belül a szoftverek felépítését egységesíti, ami előnyös a továbbfejlesztéseknél.

Az elkészített szoftver architektúrája a 2.12. ábrán látható. A sablon itt már ki lett egészítve a feladat által megkívánt plusz modulokkal is. Ahogyan az ábrán megfigyelhető, a

szoftver moduláris felépítésű, az újrahasznosíthatóság és áttekinthetőség miatt. Minden modul saját bemeneti és kimeneti bufferral rendelkezik, ami feltétele a biztonságos kódolásnak, hiszen a modulok így nem férnek hozzá egymás adataihoz. A modulok közötti adatátvitel, amit a folytonos nyilak szemléltetnek, másolófüggvények használatával történik, amivel minden modul rendelkezik. Az adatokat igénylő modul meghívja annak a modulnak a másolófüggvényét, a saját bemeneti bufferének címével paraméterezve, amitől adatokat vár, és a másolófüggvény a paraméterében megkapott címre másolja a saját kimeneti bufferének a tartalmát. Továbbá a modulok mindegyike rendelkezik saját inicializáló függvénnyel.




**2.12. ábra: Szoftverarchitektúra**

A 2.12. ábrán látható, hogy a szoftver hierarchia tetején az Operating System modul található, ami igazából nem egy operációs rendszer, hanem csak egy egyszerűbb ütemező funkciót lát el. Itt található az alkalmazás kiindulási pontja a main() függvény.

A 2.12. ábrán továbbá látható, hogy az Operating System modul szaggatott vonalakkal van összekötve a Reading Inputs, Logic, Writing Outputs és HMI modulokkal. Ezek a modulok rendelkeznek egy-egy olyan függvénnyel, amelyeknek az ütemező ciklikusan futási jogot biztosít, ezt a kapcsolatot hivatottak szolgálni a szaggatott vonalak.

A szoftver sablon a modulok feladatköreit egyértelműen definiálja. A Reading Inputs nevű modul, az eszköznek mindenféle mérendő paraméterét begyűjti, és azokat egy struktúrába rendezi.

A begyűjtött adatokat a Logic modul már rendezett formában átveszi, és feldolgozza, majd a kalkulált eredményeket a modul kimeneti struktúrájába helyezi. A Logic modul tartalmazza az eszköz logikai részeinek az implementációit, ezért rendszerint ez a legnagyobb méretű modul.

A Writing Outputs modul funkciója, hogy a logika által kalkulált eredményeket a megfelelő csatornákon a külvilág felé továbbítsa.

A HMI (Human Machine Interface) modul, szintén a logika eredményeit felhasználva, a LED-ek vezérlését látja el.

A szoftver architektúrában a modulok között ezt a folyamatot szemlélteti a vastag folytonos nyíl. A leírtakat meg lehetne valósítani egy modulban is, de nagyobb méretű kódok esetén az átláthatatlanságához, és nehezebb hibakereséshez vezetne.

A Fieldbus modul tartalmaz minden olyan részt, ami az eszköz portjain zajló kommunikáció lebonyolításáért felelős. Látható, hogy a modul tartalmazza az IO-Link stack-et, és az RS-232 almodulokat. A szoftver hierarchiában tipikusan a középső szinten helyezkedik el, mivel a modul, ellentétben az alsó szinten elhelyezkedő driver-ekkel, már nem egy periféria kezeléséért felelős, hanem több periféria együttes vezérlésével egy komplex funkciót valósít meg, amihez interfész függvények útján hozzáférést biztosít a magasabb szinten elhelyezkedő modulok részére.

A Configuration modulban található minden olyan tartalom, ami a konverter konfigurációval kapcsolatos. A szoftverspecifikációban (Konfigurációs lehetőségek című fejezetében) ismertetett paraméterek, amelyek egy index és subindex címen érhetőek el, ebben

a modulban kerültek implementálásra. Mivel az eszköz minden paraméterét a Configuration modul kezeli, ezért logikusan ez a modul áll kapcsolatban az EEPROM modullal, ami az EEPROM-ba író/olvasó függvényeket tartalmazza. Az EEPROM a tápfeszültség megszűnése után is megőrzi a tartalmát, ezért minden olyan paraméter, amire a következő újraindítás alkalmával is szükség lesz, ide kerül eltárolásra.

A hierarchia szinten alul a HAL (Hardwer Application Layer) modul található, ahol a hardver közeli kódrészletek találhatóak. Ezek a kódrészletek közvetlenül a mikrovezérlő egyes perifériáival kapcsolatosak. Például itt kerültek implementálásra a perifériák megszakítás kezelő függvényei is.

A hierarchiában a legalsó szinten a mikrovezérlő gyártója által készített függvénykönyvtár (driver-ek) található, amit a STM32L1xx HAL Driver and CMSIS (Cortex Microcontroller Software Interface Standard) nevű modul szemléltet. A függvénykönyvtár driver-eket kínál az egyes perifériákhoz, amik a perifériák regisztereit kezelik.

Az ábrán találóan függőleges oszloppal jelölt Globals modul, a modulszintű globális változókat, ill. define-okat tartalmazza.

A szintén függőleges oszloppal jelölt Error Handler modul funkciója a hibakezelés. Ha valamilyen hiba lépne fel, bármely modulban is keletkezzen, a programvégrehajtás ebbe a modulba fog eljutni, ezért van jelen a hierarchia minden szintjén.

Az implementáció bemutatásánál, a szoftver architektúrában a fontosabb modulokat mutatom be nagyobb mélységekben.

## 2.4 Szoftver implementáció

A konverter kifejlesztését alapvetően egy szoftver orientált feladatnak tartom, mivel a feladat igazi nehézsége ebben rejlik, és ezt a kifejlesztés ideje is jól tükrözte. A szoftver a beágyazott rendszerek hagyományos programozási nyelvének megfelelően C nyelven íródott.

Az implementáció bemutatása során, a szoftverarchitektúrában a felsőbb szinteken elhelyezkedő moduloknak a rendszerben betöltött szerepüket, működésüket mutatom be részletesebben.

### 2.4.1 Operating System modul

A 2.12. ábra szemléltette, hogy a szoftver hierarchia tetején az Operating System modul található, ami valójában csak egy egyszerű ütemező szerepkört tölt be. Itt található a main függvény, ami tartalmaz egy egyszerű a Round Robin elvű ütemezőt. A main függvény definícióját a következő kódrészlet szemlélteti.

```
int32_t main(void)
{
    /* Initialize return code of called functions */
    enReturn_t errorCode = RET_ERROR;

    /* Initialisation of modules */
    errorCode = OS_Init();

    /* Watchdog handler */
    HAL_Watchdog_Init();
    HAL_Watchdog_Start();

    while (errorCode == RET_OK) /* MAINLOOP */
    {
        if (RET_OK == errorCode)
        {
            /* Take snapshot of all input data (RS-232, IO-Link (PLC)) */
            errorCode = IN_ReadingInputs(COMPL_FLAG_READING_INPUT);
        }

        if (RET_OK == errorCode)
        {
            /* Compute output data from the previously collected input data */
            errorCode = LOGIC_ComputeData(COMPL_FLAG_LOGIC);
        }

        if (RET_OK == errorCode)
        {
            /* Flush output data to GPIO ports and forward to Fieldbus */
            errorCode = OUT_WritingOutputs(COMPL_FLAG_WRITING_OUTPUT);
        }

        if (RET_OK == errorCode)
        {
            /* LED Handler */
            errorCode = HMI_LedHandler();
        }

        if (RET_OK == errorCode)
        {
            HAL_Watchdog_Refresh();
        }
    } /* end of main loop */
    #ifdef __DBG_ITM
       printf("<<< ERROR >>> Exit from MAIN loop. Terminate.");
    #endif

    return(1);
}/* End of main() */
```

A kódban látható, hogy először az OS_Init() függvény kerül meghívásra, amely a modulok inicializáló függvényeit egyesével meghívja.

Ezt követi a mikrovezérlő egy Watchdog perifériájának az inicializálása és elindítása, amely újraindítja a rendszert, ha nem várt hiba lépne fel.

Végül a programvégrehajtás eljut a while ciklushoz, ami egy nagyon egyszerű ütemezőt valósít meg. Említésre került, hogy a modul kapcsolatban áll a Reading Inputs, Logic, Writing Outputs és HMI modulokkal. A felsorolt modulok mindegyike rendelkezik egy-egy olyan függvénnyel, amely a while ciklusban körkörösen meghívásra kerül.

A modulok implementációinak a bemutatása során, az ütemezőben meghívott függvényeiket fogom taglalni.

### 2.4.2 RS-232 Driver modul

A modul a nevéből fakadóan, az RS-232 interfész kezeléséért felelős kódrészleteket tartalmazza. A modult a sablon nem tartalmazta, ezért ezt nekem kellett létrehoznom.

A soros kommunikációra a mikorvezérlő USART2 perifériáját használtam. Az adatok ellentétben az IO-Link vonallal itt aciklikus jellegűek. A modul fő feladata, hogy az adatok fogadására és küldésére függvényeket biztosítson a szoftver hierarchiában magasabban elhelyezkedő modulok részére. Figyelmet fordítottam arra, hogy a modul rendelkezzen minden olyan függvénnyel (inicializáló, alaphelyzetbe állító függvények), ami alkalmassá teszi arra, hogy más projektek során is felhasználható, újrahasznosítható legyen.

A konverter kifejlesztésének során, egy komoly problémába ütköztem, aminek az oka ehhez a modulhoz vezetett vissza. Mivel komoly fejtörést és időráfordítást igényelt a probléma orvoslása, ezért magát a probléma észlelését és javítási folyamatát is bemutatom.

Egy kifejlesztett eszközt a piacra kerülését megelőzően EMC kell alávetni. A konverter EMC tesztelése során azonban folyamatos adatforgalmat kellett generálni a mérések elvégzéséhez. Az állandó jellegű adatforgalom generálásához, készítettem egy PC-én futó alkalmazást, amely a számítógép soros port-ján egy adatot küld ki, majd egy időkorláton belül ugyanannak az adatnak a vételére várakozik. Ha az időkorláton belül nem érkezik vissza visszhangszerűen az elküldött üzenet, vagy eltérő adat érkezik, azt egy hibaszámlálón kijelzi a program. Továbbá egy kollégám készített egy olyan PLC-én futó szoftvert, amely vezérli a konvertert, és az attól érkező adatokat hurokszerűen visszaküldi. Vagyis a kialakított rendszer úgy néz ki, hogy a PC-én futó szoftver adatot küld, amit a konverter fogad, és továbbküldi az

IO-Link-en a PLC irányába. A PLC a vételt követően, a fogadott adatot azonnal visszaküldi, ami a konverteren keresztül a PC-én futó szoftverhez kerül vissza. A PC-én futó applikáció, amint a helyes adatot visszakapta, azonnal küldi a következő adatot, így a konverter intenzív adatforgalom lebonyolítása alatt tesztelhető.

A fejlesztés során ugyan volt eszközöm mind az IO-Link, mind az RS-232 oldal szimulálására, azonban csak manuálisan tudtam adatforgalmat generálni, ahol mindent rendben is találtam. Azonban amikor az EMC tesztelés céljából kialakított elrendezéssel teszteltem a konvertert, ott az intenzív átvitel során kb. 3%-ban hiba lépett fel, aminek az oka a kódban volt keresendő.

A hibakeresés első lépése az volt, hogy a PC-én futó programot úgy konfiguráltam, hogy mindig azonos és ismert tartalmú adatot küldjön. A PLC-én lévő szoftverben implementálásra került egy hibaszámláló, amely akkor inkrementálódik, amikor olyan adatot fogadott, amely eltért az ismert üzenettől. Végül a teszt futtatását követően összevetettem a PC és PLC-én futó szoftverek hibaszámlálóit. Mivel mindkét hibaszámláló azonos értéket mutatott, kiderült, hogy a konverter megkapta az üzenetet, de valami okból kifolyólag az átvitel során byte-ok vesztek el a feldolgozás során. Ebből arra lehetett következtetni, hogy vagy az RS-232 adatok vételével lehet a gond, vagy az adatok feldolgozását végző kódrészletek tartalmaznak hibát. Nyilvánvalóan az IO-Link kommunikáció lebonyolításáért felelős IO-Link stack modul is tartalmazhat hibákat, de annak a valószínűsége lényegesen csekélyebb, hiszen az egy régóta használt, jól bevált modul.

A konverter szoftverében azt a hibakeresési elvet alkalmaztam, hogy lépésről lépésre követtem a beérkező adatok mozgását modulszinten, vagyis próbáltam behatárolni, szűkíteni azt, hogy mely modulban keletkezett a hiba. Hangsúlyozom, hogy a hibás adatátvitel nagyon ritkán fordult elő, ami különösen megnehezítette a hibakeresést. Végül sikerült leszűkítenem, hogy a hiba akkor keletkezik, amikor a Reading Inputs modul az adatokat átveszi az RS-232 modultól. A másolófüggvényt, amely az adatátvitelért felelős a két modul között, az RS-232 modul tartalmazza.

A kódolás során azt a megoldási módszert választottam, hogy az eseményvezérelt RS-232 vonalon beérkező adatokat megszakításosan fogadom. Ilyenkor, ha egy adatkeret beérkezik, a periféria automatikusan megszakítást generál. A megszakítás hatására a mikrovezérlő központi adatfeldolgozó egysége a szekvenciális programvégrehajtást felfüggeszti, és elugrik a periféria megszakítás kezelő függvényéhez. Ha ez lefutott, visszatér oda, ahol a szekvenciális programvégrehajtást felfüggesztette.

A beérkező adatokat a megszakítás kezelő függvény a modul bufferébe helyezi. A bufferhez tartozik egy változó is, ami az abban eltárolt adatoknak az aktuális számát tartalmazza. Azaz, minden új adat eltárolását követően, a változót is inkrementálni szükséges.

A Reading Inputs modul, amely a konverter minden bemenő adatának az összegyűjtéséért felelős, ciklikusan megvizsgálja az RS-232 modulnak a bufferét, hogy az tartalmaz-e új adatokat. A vizsgálatot a buffer elemszámát mutató változó lekérdezésével végzi. Ha a változó nem 0 értékű, akkor rendelkezésre állnak új adatok. Ebben az esetben a Reading Inputs modul a buffer tartalmát átmásolja a saját bufferébe, és az elemszámot mutató változót lenullázza. A leírtakat megvalósító függvénynek a definíciója a következő kódrészlet.

```
extern enReturn_t RS232_GetInputData(stInputData_t  *pstInputData)
{
     uint16_t i;

     __HAL_UART_DISABLE_IT(&huart2, UART_IT_RXNE);

     if(LOGIC_IsLogicReadyToReceiveRS232DataBuffer == BTRUE)
     {
          for(i=0; i < RS232_Data_In.RS232_NumberOfData; i++)
          {
                    pstInputData->RS232_Data_In.RS232_Byte[i]=
                    RS232_Data_In.RS232_Byte[i];
          }

          pstInputData->RS232_Data_In.RS232_NumberOfData=
          RS232_Data_In.RS232_NumberOfData;

          pstInputData->RS232_Data_In.RS232Buffer_OverrunError=
          RS232_Data_In.RS232Buffer_OverrunError;

          /* empties RS-232DataBuffer(new data will overwrite the old ones) */
          RS232_Data_In.RS232_NumberOfData = 0;         }
     }

     __HAL_UART_ENABLE_IT(&huart2, UART_IT_RXNE);

     return RET_OK;
}
```

Eredetileg a hibás kódban a félkövér betűkkel kiemelt sorok nem szerepeltek, és ennek hiányában intenzív adatátvitelnél alkalmanként adat byte-ok tűntek el. A függvény meghívását követően, a hibás kódban (félkövér betűvel kiemelt sorok nélkül) az első lépésnél egy feltételvizsgálat következik, ahol a globális LOGIC_IsLogicReadyToReceiveRS232Data flag vizsgálatával megállapításra kerül, hogy a Logic modul készen áll-e adatoknak a fogadására. Ha igen, akkor következik egy for ciklus, ami a függvény argumentumában átadott címre belemásolja az RS-232 bufferban található adatokat. Látható, hogy a for ciklus annyiszor fut le, ahány adat található az RS-232 bufferban. Ha buffer üres, a for ciklus egyszer sem fog lefutni.

A for ciklust követően átmásolásra kerül az RS-232 buffer elemszámlálójának a tartalma is, amiből az adatokat igénylő modul tudni fogja, hogy hány byte adatot vett át. A következő lépésben, ha történt buffer túlcsordulás, az is jelzésre kerül, mert a bufferben lévő adatok ilyenkor inkonzisztensek. Végül az RS-232 bufferhez tartozó adatszámláló kinullázásra kerül, amely sornál a hiba keletkezik. Ha az adatszámláló értéke 0-ra módosul, akkor nincsen szükség a buffer tartalmának az ürítésére, mivel az újonnan beérkező adatok egyszerűen felülírják a régieket. Adat fogadásánál, az elemszámláló aktuális értéke fogja megszabni, hogy a beérkező adat byte a buffernek mely indexű rekeszébe fog kerülni (tömbnél az első rekesz 0 indexű).

A probléma forrása abból adódik, hogy előfordulhat olyan eset, amikor a programvégrehajtás eljutott a for ciklushoz, és eközben adat érkezik az RS-232 porton, ami megszakítást generál. Így a for ciklus végrehajtása felfüggesztésre kerül, és a megszakítás kezelő függvény kezd el futni. A megszakítás kezelő függvény az újonnan beérkező adatot, a modul bufferének abba az indexű rekeszébe helyezi, amilyen értékű az elemszámláló változó (RS232_Data_In.RS232_NumberOfData). Ezt követően inkrementálja az elemszámláló értékét, és a vezérlés visszatér a for ciklushoz.

A C kódból fordított gépi kód, a for ciklust illetően nem egészen úgy működik, mint ahogyan azt a C nyelv érzékelteti. A C nyelv azt sugallja, hogy a for ciklus minden lefutás alkalmával megvizsgálja a feltételben szereplő kifejezést, és akkor fejeződik be, ha a feltétel nem teljesül. Azonban a gépi kódban válik csak igazán láthatóvá, hogy a feltételben szereplő kifejezésben a változók értéke időközben már nem módosulhat.

Azaz, miután visszatér a vezérlés a for ciklushoz, a feltételben szereplő elemszámláló értéke időközben megváltozott (RS232_Data_In.RS232_NumberOfData), ami hibás működést eredményez. Ilyenkor hiába nőtt a változó értéke, a for ciklus az elindulásának pillanatában szereplő értékkel dolgozik tovább, vagyis adat veszik el. A kódban látható, hogy a for ciklust követően, az elemszámláló kinullázásra kerül, vagyis a beérkezett adat végérvényesen elveszik.

A jelenség megmagyarázza, hogy a hibák miért csak véletlenszerűen fordultak elő. A probléma a közös erőforrás hozzáférésen alapul, ezért szinkronizációra volt szükség. A kódban látható félkövér betűkkel kiemelt sorok, olyan könyvtári függvények, amelyek a megszakításokat tiltják, ill. engedélyezik. A for ciklust megelőzően a megszakítások érvényre jutását letiltottam, majd a buffer számláló értékének 0-ra módosítását követően, azt ismét engedélyeztem. Ezzel a megoldással sikerült kiküszöbölnöm a hibajelenséget, mivel a bufferhez, és az elemszámot tartalmazó változóhoz, a hozzáférést nem megszakítható, atomi műveletté alakítottam.

### 2.4.3 Reading Inputs modul

A Reading Inputs modult tartalmazta már ugyan a sablon, de az csak utalásokat tartalmazott arra vonatkozólag, hogy a modulnak a szoftverarchitektúrában milyen jellegű szerepkört kell betöltenie. Fő feladata, hogy az eszköz mindenféle mérendő paraméterét begyűjtse, hogy azokat már egy rendezett formában elérhetővé tegye a Logic modul részére, ahol azok végül feldolgozásra kerülnek.

A modul feladatkörében a kulcs fogalom az egy struktúrába való rendezés. Létrehoztam típusdefinícióval egy stInputData_t nevű struktúrát, amelynek a definícióját a lenti kódrészlet szemlélteti. A struktúrába rendezett bemeneti adatok a Logic modulban már kényelmesen elérhetőek, és a kód könnyű átláthatóságát eredményezi.

```
typedef struct
{
    /* Data from RS-232 port */
    stRS-232DataIn_t RS232_Data_In;  /* RS-232 Process Data In from RS-232
                                        port */
                                    /* (IOL-Device -> IOL-Master) */
    /* Data from PLC */
    stProcessDataOut_t PDO_Buffer;  /* IO-Link process Data Out */
                                    /* (IOL-Master -> IOL-Device) */
    /* Data from ADC */
    uint16_t u16Voltage24V;         /* Supply monitoring */
    uint16_t u16Temperature;        /* Temperature sensing */
    uint16_t u16External24V;        /* Short circuit monitoring */

    /* Data from DIN */
    stDigitalInput_t Digital_Input; /* Digital input 1,2 */

} stInputData_t;
```

Látható, hogy a struktúra tartalmaz egy RS232_Data_In nevű elemet, amely az RS-232 vonalon beérkező adatoknak a tárolója.

A PDO_Buffer nevű elem az IO-Link vonalon beérkező, kimenő folyamat adatok részére fenntartott buffer.

A kódban látható következő blokkban, a struktúrának olyan elemei következnek, amelyek a mikrovezérlő AD konverter perifériájától származó információkat tárolják. Az u16Voltage24V nevű elembe a tápfeszültség aktuális értékével kapcsolatos információ kerül elmentésre. A mikrovezérlő hőmérsékletérzékelő perifériájától lekérdezett érték, az u16Temperature nevű elembe kerül. A konverter 24V-os tápfeszültség kimenetén, a rövidzár észlelése céljából mért adatnak, az u16External24V nevű elem van fenntartva.

Végül a két digitális bemenetnek az értékei, a Digital_Input nevű elembe kerülnek elmentésre.

Az Operating System modulban található ütemező, a Reading Inputs modulnak az IN_ReadingInputs() nevű függvényét hívja meg ciklikusan. A függvény definíciója lentebb látható.

```
extern enReturn_t IN_ReadingInputs(enCompletionFlags_t enCompletionFlag_p)
{
	enReturn_t ret = RET_OK;
	/* Clear module's input data buffer */
	in_clear_input_data_buffer(&IN_InputData);

	if (RET_OK == ret)
	{
		ret = RS232_GetInputData(&IN_InputData);
	}

	if (RET_OK == ret)
	{
		/* Trigger receive and answer of acyclic IOLink data */
		ret = FBS_CheckForAcyclicFieldBusData();
	}

	if (RET_OK == ret)
	{
		/* Get latest Process Data Out (PD-OUT) from IO-Link Master */
		ret = FBS_GetFieldbusOutputData(&IN_InputData.PDO_Buffer);
	}

	if (RET_OK == ret)
	{
		ret = in_supply_monitoring();
	}

	if (RET_OK == ret)
	{
		ret = in_temperature_sensing();
	}

	if (RET_OK == ret)
	{
		ret = in_short_circuit_monitoring();
	}

	if (RET_OK == ret)
	{
		ret = in_get_digital_inputs_data();
	}

	return ret;
}
```

A függvény a meghívását követően, a biztonságos kódolás jegyében, először törli a bemeneti adatokat tároló struktúra elemeinek a tartalmát, mielőtt azokba bármi új adatot helyezne. Ezt a műveletet az in_clear_input_data_buffer() nevű függvény látja el.

A következő lépésben meghívásra kerül az RS232_GetInputData() függvény, amelynek a kódja az RS-232 Driver modul fejezetnél már bemutatásra került. A függvény meghívásával az RS-232 vonalon beérkező aciklikus adatok kerülnek begyűjtésre a bemeneti adatokat tartalmazó struktúra megfelelő elemébe (RS232_Data_In).

Ezt követően az FBS_CheckForAcyclicFieldBusData() függvény kerül meghívásra. A függvény feladata, hogy az aciklikus ISDU logikai csatornán beérkező kéréseket kiszolgálja. Mivel az ezen a logikai csatornán beérkező kérések a konverter paramétereivel, konfigurációjával kapcsolatosak (pl. RS-232 vonal soros kommunikációs paraméterei), ezért a logikának ez nem releváns információ, így nem is kerülnek továbbításra a Logic modulba. Minden beérkező kéréshez, a címzési metódusnak megfelelően, tartozik egy index és subindex páros, amely címen a konverter egy paramétere érhető el. Például a 0x40-es index és 0x01-es subindex címen, az RS-232 vonalon a soros kommunikáció adatátviteli sebesség paramétere érhető el. Egy paramétert olvasni és írni is lehetséges. Olvasás esetén a paraméter aktuális értékét küldi vissza az ISDU csatornán. Írás esetén pedig a kiválasztott paraméter értékét állítja. A függvény az IO-Link stack és a Configuration modul interfészfüggvényeinek a felhasználásával, kiszolgálja a beérkező kéréseket. A konverternél elérhető paraméterek, amelyek a szoftverspecifikációban bemutatásra kerültek, a Configuration modulban kerültek implementálásra.

Az FBS_GetFieldbusOutputData() függvény az IO-Link vonalról a ciklikus kimenő folyamat adatokat gyűjti be, és helyezi el azokat a bementi adatokat tartalmazó struktúrába (a PDO_Buffer elembe).

Az in_supply_monitoring(), in_temperature_sensing() és in_short_circuit_monitoring() függvények, a mikrovezérlő AD konverter perifériának a különböző csatornáiról olvasnak ki egy digitalizált értéket, és mentik azokat a bemeneti adatokat tartalmazó struktúra megfelelő elemeibe (u16Voltage24V, u16Temperature, u16External24V).

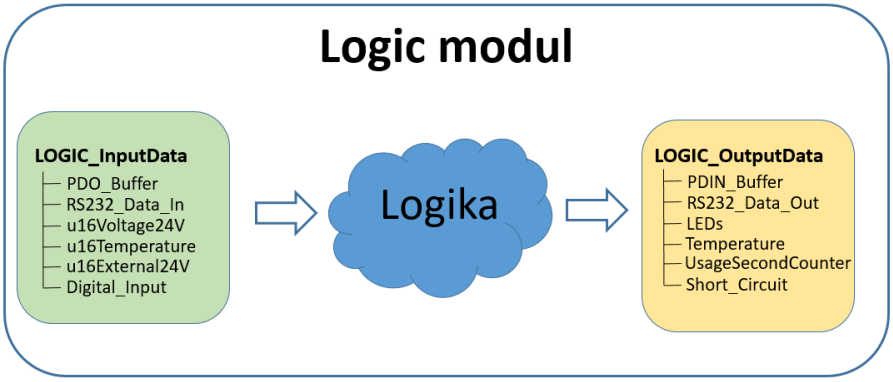
Végül az in_get_digital_inputs_data() függvény a digitális bemenetek aktuális értékét kiolvassa, és menti el azokat a bemeneti struktúrába (Digital_Input elembe).

### 2.4.4 Logic modul

A Logic modul, ahogyan azt a neve is sugallja, a konverter logikájáért felelős kódrészleteket tartalmazza. Az előző fejezetben bemutatásra került, hogy a Reading Inputs modul a konverter bemeneteiről adatokat olvas be, és azokat már rendezett formában kínálja a Logic modulnak. A bejövő adatok alapján a logika elkészíti a modul (tulajdonképpen magának a konverternek) kimenő adatait, amiket egy kimenő struktúrába rendez. Ezután ezeket a kimenő adatokat a Writing Outputs modul a megfelelő csatornákon továbbítja a „külvilág” részére.

#### 2.4.4.1 A modul kimeneti adatstruktúrája

A bevezetőben leírt módon a Logic modul átvesz egy bemeneti adathalmazt, a Reading Inputs modultól, és azokat felhasználva előállít a többi modul számára egy kimeneti adatstruktúrát. Ezt a jellegű működést a 2.13. ábra szemlélteti.




**2.13. ábra: A Logic modul bemeneti és kimeneti adathalmazai**

A 2.13. ábrán látható, hogy a bemeneti adathalmaz a LOGIC_InputData nevű változóból érhető el, a logika által előállított adatok pedig a LOGIC_OutputData nevű változóba kerülnek elmentésre. Mindkét változó típusa egy struktúra. A bemeneti adathalmaz típusa természetesen megegyezik a Reading Inputs modulnál bemutatott stInputData_t típussal. A kimeneti adathalmaz, vagyis a LOGIC_OutputData változó típusa stOutputData_t, amelynek a definíciója lentebb olvasható.

```
typedef struct
{
	/* Data to PLC */
	stProcessDataIn_t PDIN_Buffer;

	/* Data to RS-232 port */
	stRS-232DataOut_t RS-232_Data_Out;

	/* LED */
	stOutputSignals_t LEDs;

	/* Temperature values */
	stTemperaureData_t Temperature;

	/* Usage time measurement in second*/
	uint32_t UsageSecondCounter;

	/* Indication of short circiut */
	bool_t Short_Circuit;

}stOutputData_t;
```

A kódból látható, hogy a modul kimeneti adathalmaza már tartalmazza a bemenő folyamat adatokat, amelyet a PDIN_Buffer nevű elem tárol. Ezeket az adatokat fogja a vezérlés, ami általában egy PLC, ciklikus jelleggel megkapni. A szoftver specifikációban már említésre került, hogy a bemenő folyamat adatok hosszát 32 byte-ra állítottam. Így a Logic modul által előállított bemenő folyamat adatoknak a PDIN_Buffer tárolóját is ehhez kellett igazítanom, vagyis a stProcessDataIn_t típus mögött egy 32 byte méretű tömb rejlik.

Az stOutputData_t típusú struktúrának az RS232_Data_Out a következő eleme. Ebbe a tárolóba kerülnek azok az adatok, amelyek a konverter RS-232 vonalára csatlakozó eszköznek kerülnek aciklikusan elküldésre.

A struktúra további elemei a LEDs, Temperature, UsageSecondCounter, Short_Circuit, amelyek nincsenek kapcsolatban az IO-Link RS-232 adatátvitellel. A LEDs elembe a LED-ek vezérlésével kapcsolatos információk kerülnek. A Temperature elem a hőmérsékleti adatok tárolója. A UsageSecondCounter elembe a konverter első bekapcsolástól eltelt ideje kerül elmentésre. A Short_Circuit elem értéke mutatja, hogy az eszköz 24V-os kimenete rövidre van e zárva. A struktúrának tehát ezen elemeibe a diagnosztikai célú adatok kerülnek tárolásra.

#### 2.4.4.2 A beérkező adatok feldolgozásának a folyamata

Az Operating System modulban található ütemező, ciklikusan meghívja a Logic modul LOGIC_ComputeData() nevű függvényét. A LOGIC_ComputeData() függvény feladata, hogy a konverter a szoftverspecifikációban leírt, IO-Link rendszer fölé rendelt kommunikációs logikát megvalósítsa. A függvény definíciója lentebb olvasható.

```
extern enReturn_t LOGIC_ComputeData(enCompletionFlags_t enCompletionFlag_p)
{
    enReturn_t    enRetVal = RET_OK;    /* return value */

    /* Clean the Output buffers before calculation */
    if (RET_OK == enRetVal)
    {
        enRetVal = logic_reset();
    }

    /* Incoming data */
    if (RET_OK == enRetVal)
    {
        enRetVal = IN_GetReadingInputStructures(&LOGIC_InputData);
    }

    if (RET_OK == enRetVal)
    {
        enRetVal = get_PDOUT_header_flags();
    }

    if (RET_OK == enRetVal)
    {
        enRetVal = set_PDIN_header_flags();
    }

    if (RET_OK == enRetVal)
    {
        enRetVal = determine_state();
    }

    if (RET_OK == enRetVal)
    {
        enRetVal = call_statemachine();
    }

    /*****************************************************/
    /********* Handlers **********************************/
    /*****************************************************/

    if (RET_OK == enRetVal)
    {
        enRetVal = logic_temperature_handler();
    }

    if (RET_OK == enRetVal)
    {
        enRetVal = logic_usageSecondCounter_handler();
    }

    if (RET_OK == enRetVal)
    {
        enRetVal = logic_short_circuit_handler();
    }

    if (RET_OK == enRetVal)
    {
        enRetVal = logic_led_handler();
    }
```

```
    if (RET_OK == enRetVal)
    {
        enRetVal = logic_event_handler();
    }

    return enRetVal;
}
```

Látható, hogy a LOGIC_ComputeData() függvény első lépésben meghívja a logic_reset() nevű függvényt, amely törli a Logic modul minden belső változójának a tartalmát, mielőtt az bármi kalkulációba kezdene. Erre a biztonságos programozás jegyében van szükség.

A következő lépésben az IN_GetReadingInputStructures() függvény kerül meghívásra, amely átmásolja a Reading Inputs modul által összegyűjtött bementi adatokat a Logic modul saját bufferébe.

Miután a bemeneti adatok a modul rendelkezésére állnak, a get_PDOUT_header_flags() függvény kerül meghívásra. A függvény feladata, hogy a kimenő folyamat adatok alapján, beállítsa a PLC header biteket reprezentáló változók értékét. Ezekre a modul szintű, bool típusú változókra azért van szükség, hogy a kódot átláthatóbbá, könnyebben értelmezhetőbbé tegyék.

A set_PDIN_header_flags() nevű függvény, a PLC header bitek alapján, a konverter header biteket állítja be.

Miután a konverternek a header bitjei értéket kaptak, a determine_state() nevű függvény kerül meghívásra. A függvény felhasználva a PLC és konverter header biteket, megállapítja, hogy a konverter aktuálisan milyen állapotban tartózkodik.

Ezt követően az IO-link rendszer fölé rendelt kommunikációs logika utolsó függvénye is meghívásra kerül, amely a call_statemachine() nevet viseli. A függvény a konverter állapotgépét tartalmazza, amelynek definíciója a lentebb látható kódrészlet. A kódból látható, hogy a konverter aktuális állapota a globális state nevű, enum típusú változóba kerül elmentésre. Ahogyan az megfigyelhető, 7 állapotot különböztettem meg, amelyek az újraindítás, tétlen, olvasás, írás, várakozás olvasásra, feladat befejezése és hibakezelés állapotok. Az állapotok fő feladata, hogy előállítsák a bemenő folyamat adatokat, ill. az írás állapotban még a kimenő RS-232 adatokat is.

```
enReturn_t call_statemachine(void)
{
	enReturn_t enRetVal = RET_OK;

	switch(state)
	{
		case Reset_State:
			enRetVal = Reset();
			break;

		case Idle_State:
			enRetVal = Idle();
			break;

		case Read_State:
			enRetVal = Read();
			break;

		case Write_State:
			enRetVal = Write();
			break;

		case Waiting_For_Read_Request_State:
			enRetVal = Waiting_for_read_request();
			break;

		case Job_Is_Done_State:
			enRetVal = Job_is_done();
			break;

		case Error_State:
			enRetVal = Error();
			break;
	}

	return enRetVal;
}
```

Visszakanyarodva az ütemező által hívott LOGIC_ComputeData() függvényhez, a call_statemachine() függvényt követően már nem található az IO-Link rendszerre implementált protokollhoz kapcsolódó tartalom. Azonban a konverter rendelkezik olyan diagnosztikai és hibafelismerő funkciókkal is, amelyek nem az IO-link rendszer fölé rendelt kommunikációs logikába vannak beágyazva, vagyis nem a bemenő folyamat adatokban jelennek meg hibakódként.

A mikrovezérlő rendelkezik egy belső hőmérsékletmérő perifériával, amely által szolgáltatott adatokat a logic_temperature_handler() nevű függvény dolgozza fel. A függvény feladata nem csupán a hőmérséklet kiszámítása, hanem a bekapcsolás pillanatától, ill. a valaha mért legmagasabb, legalacsonyabb hőmérsékletértékeknek az eltárolása is, diagnosztikai célokból.

A logic_usageSecondCounter_handler() nevű függvény, a konverter első bekapcsolásától számítva, méri a működés közben eltelt másodperceket. A funkció szintén diagnosztikai célokra használatos.

A logic_short_circuit_handler() elnevezésű függvény az RS-232 oldalon a kapcsolható külső 24V-os tápfeszültség rövidre zárását hivatott észlelni. Ha a felhasználó hibás installációból eredően, ezt a tápfeszültséget rövidre zárná, akkor a függvény ezt érzékeli és jelzi egy globális flag bebillentésével.

A LED-ek vezérléséért a logic_led_handler() függvény felelős. Feladata, hogy a konverter állapotáról közvetlenül a felhasználót tájékoztassa, LED-eken definiált jelzésekkel. A LED-eken a következő jelzések értelmezettek:

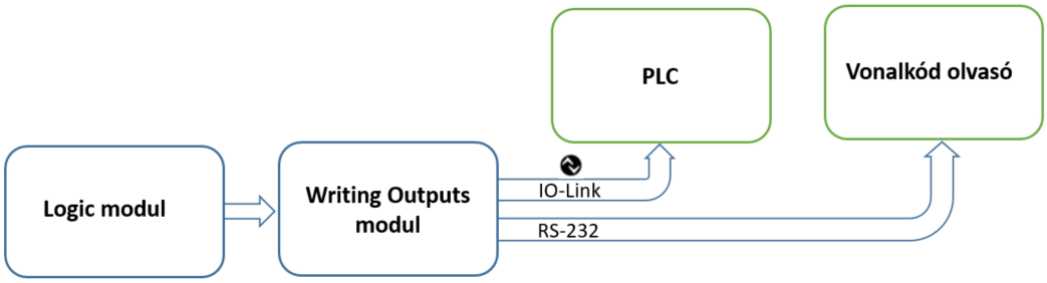
- A modul feszültség alatt áll.
- A tápfeszültség értéke túl alacsony.
- Az IO-Link kommunikáció aktív.
- A kapcsolható külső 24V-os tápfeszültség rövidre van zárva.

A LED-ekhez kapcsolódó mikrovezérlő pin-eket, ugyan a HMI modul vezérli a megfelelő időzítésekkel, de magát a kijelzésre kerülő állapotot ez a függvény állítja elő a HMI részére.

Végül a LOGIC_ComputeData() utolsó függvénye a logic_event_handler(). Feladata, hogy az IO-Link kommunikáció aciklikus diagnosztikai csatornájának a felhasználásával, a vezérlés részére diagnosztikai eseményeket generáljon. Ha a konverter a -5 $^0$C-tól a +55 $^0$C-ig terjedő működési tartományon kívülre esik, vagy a tápfeszültség értéke túl alacsony, a függvény figyelmeztető eseményeket (Warning event) generál. Továbbá a függvény hibaeseményt (Error event) hoz létre, ha a kapcsolható külső 24V-os tápfeszültséget a felhasználó rövidre zárja.

### 2.4.5 Writing Outputs modul

A Writing Outputs modul feladata, hogy a logika által előállított adatokat átvegye, és azokat a megfelelő csatornákon elküldje a konverter kimeneteire. A modul által betöltött funkciót a 2.14. ábra szemlélteti.




**2.14. ábra: A Writing Outputs modul funkciója**

Az Operating System modulban található ütemező, a Writing Outputs modulnak az OUT_WritingOutputs() nevű függvényét ciklikusan meghívja. A függvény definíciója lentebb olvasható.

```
extern enReturn_t OUT_WritingOutputs(enCompletionFlags_t enCompletionFlag_p)
{
    enReturn_t ret = RET_OK;

    /* Get a copy of the Output Data from LOGIC */
    if (RET_OK == ret)
    {
        ret = LOGIC_GetOutputData(&OUT_stOutputData);
    }

    /* Send PD-IN data to IO-Link Master */
    if (RET_OK == ret)
    {
        ret =  FBS_SetFieldbusInputData(&OUT_stOutputData);
    }

    /* Send out RS-232 data */
    if (RET_OK == ret)
    {
        ret = RS232_WriteRS232OutputData(&OUT_stOutputData);
    }

    return ret;
}
```

A kódban látható, hogy a függvény első lépésben meghívja a LOGIC_GetOutputData() nevű függvényt, amely átmásolja a Logic modul kimeneti adatait a modul saját tárolójába.

Ezt követően meghívásra kerül az FBS_SetFieldbusInputData() nevű függvény, ami az IO-Link vonalon a ciklikus bemeneti folyamat adatokat elküldi a PLC irányába. A függvény a kommunikáció lebonyolításához az IO-Link stack interfészfüggvényeit használja.

Végül az RS232_WriteRS232OutputData() függvény hívódik meg, amely az eseményvezérelt RS-232 vonalon küldi el az adatok, ha azok éppen rendelkezésre állnak.

### 2.4.6 HMI modul

A HMI modult a sablon már tartalmazta, de azt plusz funkciókkal kellett kiegészítenem. A feladata, hogy a logika eredményei alapján, a LED-eket villogtassa a megfelelő időzítésekkel, a szoftverspecifikációban leírtak szerint.

Az Operating System modulban található ütemező ciklikusan meghívja a modul HMI_LedHandler() függvényét, amely a Logic modul által előállított LED-eken kijelzendő konverter állapotokat átmásolja a modul saját tárolójába, majd azok alapján jelzéseket generál a LED-eken a felhasználónak. A HMI_LedHandler() függvény definícióját a következő kódrészlet szemlélteti.

```
extern enReturn_t HMI_LedHandler(void)
{
    enReturn_t     enRetVal = RET_ERROR; /* return vale */
    stLEDs_t       stLedLogicState;      /* LEDs states calculated in LOGIC */
    bool_t          is_50ms_expired = BFALSE;

    enRetVal = HAL_Timer_GetLEDTimerState(&is_50ms_expired);

    if (RET_OK == enRetVal)
    {
          if (BTRUE == is_50ms_expired)
          {
               /* retrieve the calculated LED states from LOGIC */
               enRetVal = LOGIC_GetLED_Data(&stLedLogicState);

               /* check the blinking generator */
               if (RET_OK == enRetVal)
               {
                    enRetVal = HMI_led_blinking_generator();
               }

               /* refresh the new LED states */
               if(RET_OK == enRetVal)
               {
                    enRetVal = refreshLEDs(&stLedLogicState);
               }
          } /* End of timer triggered tasks - if(is_50ms_expired)*/
          else
          {
               /* Do nothing here. Wait for timer event. */
          }
    }
    return enRetVal;
}
```

A kódban látható, hogy a mikrovezérlőnek egy időzítő perifériája felhasználásával, 50 ms időközönként, frissítésre kerülnek a LED-ek állapotai (ha szükséges). Látható, hogy a HAL_Timer_GetLEDTimerState() függvény, az is_50ms_expired nevű változónak igaz értéket ad 50 ms-onként. A frissítési művelet során, először a Logic modul által előállított LED-eken

kijelzendő konverter állapotokat másolja át a modul a saját bufferébe, amelyet a LOGIC_GetLED_Data() függvénnyel végez. Következő lépésben a HMI_led_blinking_generator() függvény hívódik meg, amely meghatározza, hogy a LED-eknek aktuálisan be- vagy kikapcsolt állapotban kell lenniük. A szoftverspecifikációban megadott LED jelzések generálásához, a függvény egy számlálóval számolja az eltelt időt ($x * 50\,ms$), így különböző frekvenciájú jelzéseket képes előállítani. Végül a refreshLEDs() függvény már ezen információk alapján, közvetlenül a mikrovezérlőnek azokat a pinjeit vezérli (GPIO[1] pinek), amik a LED-ekhez csatlakoznak.

### 2.4.7 Configuration modul

A Configuration modul feladata a konverter paramétereinek a kezelése. A konverter paraméterei, az IO-Link kommunikációnak megfelelően, az aciklikus ISDU csatornán kerülnek átvitelre. A paraméterek ennek megfelelően, az IO-Link kommunikáció bemutatásában leírtak szerint, egy index és subindex címzési metódussal érhetőek el. Azaz, minden paraméterhez tartozik egy index és subindex érték.

Már bemutatásra került, hogy az ISDU csatornának a kiszolgálását a Reading Inputs modul végzi, ahol említettem, hogy a paraméterek olvasására és írására a Configuration modul, az ISDU csatorna kezeléséhez pedig az IO-Link stack kínál interfészfüggvényeket.

Ahhoz, hogy a beállított paraméterek a konverter újraindulását követően is megőrizzék az értéküket, a mikrovezérlő belső EEPROM memóriáját használtam az értékek elmentésére.

Egy index és subindex párossal kiválasztott paramétert olvasni és írni is lehetséges, ennek megfelelően a modul két függvényt biztosít a paraméterek kezeléséhez. A paraméterek olvasására és írására leimplementált függvények prototípusait a következő kódrészlet szemlélteti.

```
extern enReturn_t CONFIG_Get_ISDUConfigData( uint16_t index, uint8_t subindex,
uint8_t *buflen, uint8_t aBuffer[], uint8_t *additionalCode );

extern enReturn_t CONFIG_Evaluate_ISDU_telegram( uint16_t index, uint8_t
subindex, uint8_t buflen, uint8_t aBuffer[], uint8_t *additionalCode );
```

Látható, hogy a paraméterek olvasását megvalósító CONFIG_Get_ISDUConfigData() függvény, az argumentumában átvesz egy index és subindex értéket, ami a konverter egy

[1] General Purpose Input Output

paraméterét jelöli ki. A kijelölt paraméterhez tartozó EEPROM címről kiolvassa a paraméter aktuális értékét, és azt az argumentumában megadott címre belemásolja (aBuffer[]), mellékelve az adat méretét is (*buflen). Ha az olvasás során valamilyen hiba merülne fel (például nem létezik paraméter a megadott index, subindex címen), akkor egy hibakódot ír az argumentumában az erre a célra átvett címre (*additionalCode).

A paraméterek állításáért a CONFIG_Evaluate_ISDU_telegram() függvény felelős. A függvény az argumentumában átvesz egy index, subindex párost, ami a konverter egy paraméterét jelöli ki, egy adattömböt (aBuffer[]), amiben a paraméter új értéke található, és egy adathossz értéket (buflen), amely megmondja, hogy hány byte méretű az adat a tömbben. A függvény ezeket az információkat felhasználva, a kijelölt paraméterhez tartozó EEPROM címre beleírja az új értéket, és annak érvényt is szerez, vagyis módosítja a konverter konfigurációját. Ha valamilyen hiba adódna a művelet végrehajtása közben (például a paraméterérték nem értelmezett), akkor egy hibakódot helyez az argumentumában átvett címre (*additionalCode).

## 2.5 A fejlesztés során felhasznált eszközök

A konverter kifejlesztését a kapcsolási rajz, majd a nyomtatott áramkör megtervezésével kezdtem, amit a cégnél használatos Zuken CR-8000 szoftvercsomaggal végeztem [11]. A kapcsolási rajz elkészítéséhez a Design Gateway Tool-t, a nyomtatott áramkör megtervezéséhez a Board Designer Tool-t használtam.

Miután az elkészült hardver a rendelkezésemre állt, a működtető beágyazott szoftver implementációjához láttam hozzá. A szoftvert a Keil µVision5 fejlesztőkörnyezetben C nyelven írtam [12].

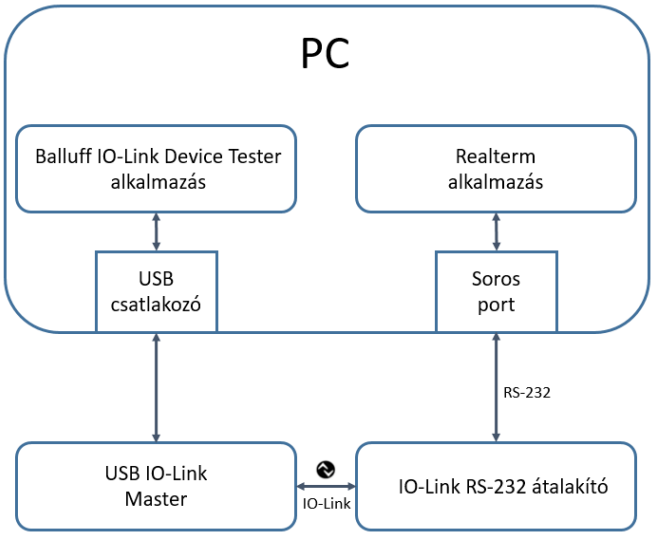
A mikrovezérlő felprogramozásához, és annak a belső folyamataiban a hibakeresés (debugging) lehetőségéhez (breakpoint-ok alkalmazása, regiszterek és változók értékeinek a monitorozása, stb.) szükségem volt egy programozó/debugger eszközre, amely célból az ST-LINK/V2 eszközt alkalmaztam [13]. Az ST-LINK/V2 programozó/debugger a mikrovezérlőhöz egy serial wire periférián keresztül csatlakozik.

A szoftverfejlesztés során szükségem volt arra, hogy a konverter működését valós körülmények között is tesztelhessem, ezért készítenem kellett egy olyan rendszert, amely szimulálja a PLC-t és az RS-232 interfésszel rendelkező eszközt.

A PLC-én futó szoftver elkészítése nem tartozik a feladatkörömbe, azonban a konverter szoftverének a fejlesztésénél, tesztelésénél szükségem volt arra, hogy szimulálni tudjam a PLC működését. E célból a cégnek a belső fejlesztésű Balluff IO-Link Device Tester nevű PC-én futó alkalmazását használtam, amely egy USB master modullal képes az IO-Link logikai csatornáin adatot küldeni és fogadni. Ez számomra elegendő az implementált szoftver ellenőrzésére, hiszen manuálisan írhatom a kimenő folyamat adatokat, és olvashatom az arra válaszként érkező bemenő folyamat adatokat a konvertertől.

A konverter RS-232 vonalának a szimulálására az internetről ingyenesen letölthető Realterm nevű alkalmazást használtam [14]. A Realterm egy soros kommunikációs terminál alkalmazás, aminek a segítségével küldhetek, fogadhatok adatokat, és bonyolultabb adatfolyamok esetén hibakeresésre is jól alkalmazható eszköz. A Windows operációs rendszerek beépítetten is tartalmaznak egy HyperTerminal nevű soros kommunikációs terminál alkalmazást, de ennek az eszközkészlete kisebb.

Így a szoftverfejlesztés során, az egyes funkciók tesztelésére kialakított rendszert a 2.15. ábra szemlélteti.




**2.15. ábra: A konverter fejlesztése során alkalmazott rendszer**

A 2.15. ábrán kialakított rendszerrel a konverter mindkét irányú adatátvitel esetén tesztelhető (olvasás, írás műveletek).

Olvasás művelet esetében, adatot küldök a Realterm alkalmazással, amivel azt szimulálom, hogy a csatlakoztatott RS-232 interfésszel rendelkező eszköz adatot küld. Miután a konverter a Realterm által elküldött adatokat fogadta, a Balluff IO-Link Device Tester alkalmazás, ami a PLC-t szimulálja, a bemutatott, IO-Link rendszer fölé rendelt

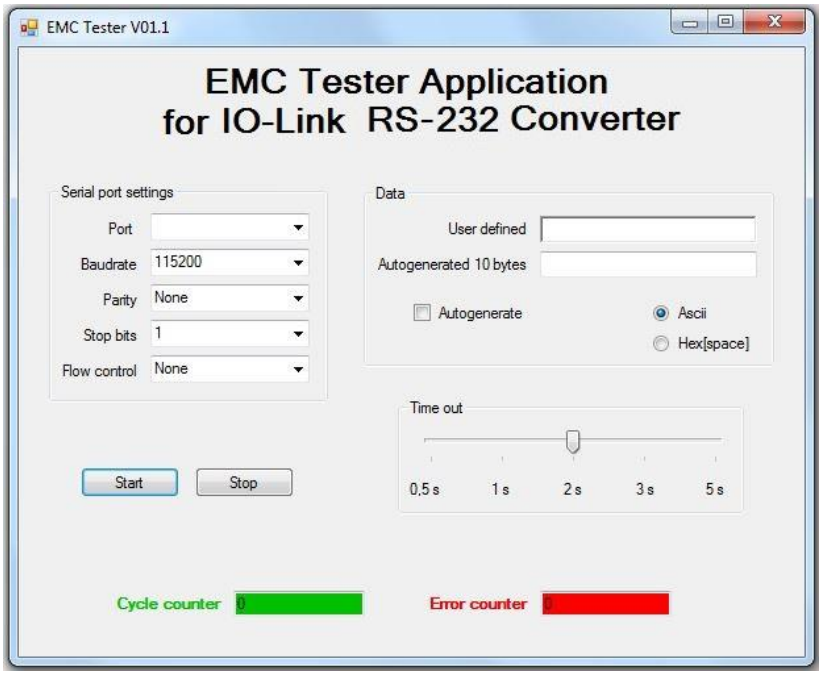
kommunikációs logika alkalmazásával, átveszi az adatokat. Az alkalmazás biztosít egy olyan felületet, ahol manuálisan írhatom a kimenő folyamat adatokat, ill. kijelzi, hogy a konverter milyen bemenő folyamat adatokat küld válaszként vissza.

Az írás műveletnél az ellentétes irányban zajlik az adatátvitel. Itt a Balluff IO-Link Device Tester alkalmazás küldi az adatot, amit a konverteren keresztül a Realterm terminál fogad.

A 2.15. ábrán kialakított rendszerrel a fejlesztés során, lépésről lépésre tesztelni tudtam, hogy az implementált szoftver valóban helyesen működik-e. Ellenőrizhető, hogy a Realterm-ből elküldött adatok megegyeznek-e, a Balluff IO-Link Device Tester alkalmazásban fogadott adatokkal, ill. fordítva, valamint az átvitel során az egyes lépések, a szoftver specifikációban meghatározottak szerint zajlik-e.

A konverternek egy bootloader alkalmazást is tartalmaznia kell. A bootloader-t nem én készítettem, a cégnél minden IO-Link-es eszköznél egységesen, ugyanazt a bootloader programot használják (amelyek azonos típusú mikrovezérlővel rendelkeznek). A bootloader lehetővé teszi, hogy az IO-Link csatlakozón keresztül soros kommunikációval az összeszerelést követően töltsük fel az eszköre a firmware-t. Ez azért fontos, mert a termék az összeszerelés után egy fém házzal, gyantával kitöltve le van zárva, és a programozó csatlakozóhoz így már nem lehet hozzáférni. Ezért, ha rendelkezésre áll egy új szoftver verzió azt már csak a bootloader-rel lehet letölteni az eszközre.

Az RS-232 driver modul című fejezetben már említésre került, hogy az EMC tesztelés céljából egy PC-én futó alkalmazást kellett készítenem. Az alkalmazást C# nyelven implementáltam, ami a 2.16. ábrán látható. A Serial port settings nevű blokkban a soros kommunikációs paraméterek állíthatóak be. A Data nevű blokkban a felhasználó manuálisan megadhatja az elküldendő adatot, vagy automatikusan generált 10 byte méretű adatokat is választhat. A Time out blokkban 5 előre definiált timeout érték közül lehet választani. A zölddel jelölt Cycle counter szövegdobozban a sikeres ciklusok (az elküldött adat timeout-on belül visszaérkezett) száma, a pirossal jelölt Error counter a sikertelen (timeout értéken belül nem érkezett üzenet, vagy nem az elküldött adat érkezett vissza) ciklusok számát jelzi ki.




**2.16. ábra: A konverter EMC tesztelése céljából készített alkalmazás**

# 3 Összegzés

Az IO-Link RS-232 átalakító kifejlesztésének az igényét konkrét vevői megrendelés ihlette (General Motors), és az általam létrehozott eszköz tömeggyártásba fog kerülni. Pillanatnyilag terméklistán még nem szerepel a konverter, azonban kész prototípus már rendelkezésre áll. A gyártást megelőző fázisban még számos tesztnek vetik alá a konvertert (EMC, Szoftver tesztek, hőmérséklet tesztek, stb.), hogy biztosak lehessünk a termék hibamentességében, ill. abban, hogy valóban megállja-e a helyét ipari környezetekben is (CE, UL, stb. tanúsítványoknak megfeleltetés [15], [16]).

A diplomamunkám során alaposan megismertem az IO-Link kommunikációt. Ezt követően elkészítettem a hardver kapcsolási rajzát, majd a nyomtatott áramköri rajzot. Miután rendelkezésemre állt a hardver, az azt működtető beágyazott szoftver fejlesztésének láttam hozzá. Elkészítettem továbbá a konverternek az IODD leíró fájlját, amivel minden IO-Link slave eszköznek rendelkeznie szükséges. Miután az eszköz elkészült, azt EMC tesztelésnek vetettem alá, amin sikeresen megfelelt. Az EMC tesztelés elvégzéséhez egy külön PC-én futó grafikus felhasználói felületű tesztszoftvert is implementálnom kellett.

A konverternél továbbfejlesztési lehetőségként már felmerült az igény egy olyan variáns létrehozására, amely RS-232 helyett RS-485 jelszintet használ az eseményvezérelt soros kommunikációs oldalon. A variáns apróbb szoftver és hardver módosításokat igényelne. A termelés igényeit, és a költséghatékonyságot figyelembe véve, a két variáns közös PCB-t használna. Azaz a layout terv úgy kerülne kialakításra, hogy mindkét variánshoz alkalmas legyen, és a gyártás során csak a beültetéskor dől el, hogy melyik variánst fogja megvalósítani az adott PCB. A szoftveres részről a variánsok kezelése úgy történne, hogy a PCB tartalmazna egy ellenállás-alapú feszültségosztót (ellenállás létrát), és az azon beállított feszültség érték visszamérésével, a firmware képes lenne megállapítani, hogy mely variánsról van szó.

# Irodalomjegyzék


[1] International Electrotechnical Commission (2013. szept.01), Document Number: iec 61131-9, http://standards.globalspec.com/std/1629607/iec-61131-9

[2] IO-Link Consortium (2011. okt.), IO-Link Interface and System V1.1 Specification, http://www.io-link.com

[3] IO-Link Consortium (2011. aug.), IO-Link Device Description Specification Version 1.1, http://www.io-link.com/en/Download/Download.php

[4] STMicroelectronics (2010. júl.02), STM32L151x6/8/B STM32L152x6/8/B datasheet, http://www.st.com/content/ccc/resource/technical/document/datasheet/66/71/4b/23/94/c3/42/c8/CD00277537.pdf/files/CD00277537.pdf/jcr:content/translations/en.CD00277537.pdf

[5] ELMOS Semiconductor AG (2010. jún.16), E981.10 datasheet, http://www.elmos.com/fileadmin/2013/02_products/01_interface/01_io-link/e981-10_elmos_ds.pdf

[6] Maxim Integrated Products, MAX14824 datasheet, https://datasheets.maximintegrated.com/en/ds/MAX14824.pdf

[7] intersil (2015. szept.01), ICL3232IVZ datasheet, http://www.intersil.com/content/dam/Intersil/documents/icl3/icl3221-22-23-32-41-43.pdf

[8] Maxim Integrated Products, MAX3250 datasheet, https://datasheets.maximintegrated.com/en/ds/MAX3250.pdf

[9] Texas Instruments (1998. szept.), LM2674 datasheet, http://www.ti.com/lit/ds/symlink/lm2674.pdf

[10] infineon (2012. dec.01), Smart High-SidePower Switch ISP762T datasheet, http://www.infineon.com/dgdl/Infineon-ISP762T_20121201-DS-v01_04-en.pdf?fileId=db3a304412b407950112b428ef533ebd

[11] Zuken, CR-8000, http://www.zuken.com/en/products/pcb-design/cr-8000

[12] ARM Keil, µVision5, http://www2.keil.com/mdk5/uvision/

[13] STMicroelectronics, ST-LINK/V2, http://www.st.com/en/development-tools/st-link-v2.html

[14] Realterm: Serial Capture Program, https://realterm.sourceforge.io/index.html#downloads_Download

[15] https://ec.europa.eu/growth/single-market/ce-marking_en

[16] http://www.ul.com/marks/ul-listing-and-classification-marks/promotion-and-advertising-guidelines/specific-guidelines-and-rules/