Szegedi Tudományegyetem

Informatikai Tanszékcsoport

**Ipari folyamat szimulációja és irányítása programozható logikai vezérlővel**

Diplomamunka

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Miklós Árpád |  | Dr. Kincses Zoltán |  |
|  | mérnök informatikus szakos hallgató |  | egyetemi adjunktus |  |

Szeged

2017

Ipari folyamat szimulációja és irányítása programozható logikai vezérlővel.



**Diplomamunka mérnök-informatikus MSc szakos hallgató számára**

**Témavezető: Dr. Kincses Zoltán**

**Témakör: ipari informatika, rendszer szimuláció, irányítás**

**Műszaki Informatika Tanszék**



**A feladat leírása, a munka célja**

A hallgató feladata egy ipari folyamat emulációjának és irányításának elkészítése. A munka célja, egyrészt az Országos Ajtonyi István Irányítástechnikai Programozó Versenyen szereplő technológia emulációjának elkészítése egy HIL szimulátor eszköz segítségével. A munka másik célja az így elkészített rendszer irányításának megvalósítása egy programozható logikai vezérlővel. Ez elkészült munka később jól alkalmazható a későbbi PLC versenyekre történő felkészítésben

**A munkavégzés fontosabb lépései:**

* A HIL szimulátor eszköz és a hozzá tartozó szoftverek megismerése, ismertetése
* Az emulálni kívánt technológia megismerése
* A programozható logikai vezérlő és a hozzá tartozó szoftver megismerése, ismertetése
* A HIL emuláció elkészítése
* Az elkészült emuláció irányításának megvalósítása programozható logikai vezérlővel
* Hibakezelés
* Tesztelés
* A dolgozat megírása

**A fejlesztéshez rendelkezésre álló erőforrások:**

* OMRON CJ2M PLC, I/O egységek és a programozásához szükséges szoftver
* Lucas Nülle I/O interfész PRO/TRAIN-hez
* Lucas Nülle BORIS szoftver csomag
* Lucas Nülle PRO/TRAIN



**A jelentkezés feltételei:**

* Érdeklődés a PLC alapú vezérlések és vizualizációjuk iránt
* Angol nyelvtudás

# Tartalmi összefoglaló

* **A téma megnevezése:**

Ipari folyamat emulálása Hardware-In-the-Loop (HIL) eszközzel és ennek irányítása programozható logikai vezérlővel.

* **A megadott feladat megfogalmazása:**

Emulálnom és irányítanom kell egy ipari folyamatot, amelyik egy korábbi Országos Ajtonyi István Irányítástechnikai Programozó Versenyen szerepelt. Az irányításhoz Programozható Logikai Vezérlőt (PLC) kell alkalmaznom, az emulációhoz pedig a WinFACT szoftvercsomagot. Az alap feladatkiíráson túl plusz feladatként meg kell terveznem egy olcsóbb alternatívát a Lucas Nülle I/O interfész helyettesítésére.

* **A megoldási mód:**

A feladatkiírásban szereplő szoftvercsomagok és a hozzájuk tartozó technológiák megismerése és ismertetése. A kiválasztott ipari folyamat emulációjának és a hozzá tartozó irányításnak a megtervezése és megvalósítása, majd ezek tesztelése. Az I/O interfész és a hozzá tartozó szoftver közötti kommunikáció feltárása, tanulmányozása és az eredmények alapján egy saját eszköz megtervezése és tesztelése.

* **Alkalmazott eszközök, módszerek:**
* Alkalmazott eszközök: Lucas Nülle I/O interfész, Omron CJ2M-CPU32 PLC, CX-Programmer 9.1, mikroC PRO for PIC 6.6.2 és Microsoft Visual Studio 15 fejlesztőkörnyezetek, WinFACT 7 BORIS tervezőszoftver, Altium Designer 16.1.7 NYÁK-tervező szoftver, MikroElektronika EasyPIC v7 fejlesztőlap.
* Alkalmazott módszerek: kommunikáció, fejlesztőkörnyezet és fejlesztőeszközök tanulmányozása, fejlesztés, tesztelés, áramkör-szimuláció, mérés és kiértékelés.
* **Elért eredmények:**

A WinFACT 7 szoftvercsomag BORIS (Block Oriented Simulation System) szoftverét kiismertem és dokumentáltam. Megterveztem és teszteltem a kiválasztott ipari folyamatot – karosszéria gyártósor –, illetve elkészítettem a hozzá tartozó irányítást. Megvizsgáltam az I/O interfész és a BORIS közötti kommunikációt, az I/O interfész felnyitása nélkül. Az eredmények alapján megterveztem egy alternatív áramkört, amelyik képes az említett eszközt helyettesíteni. A terveket szimulációval és egy fejlesztőlap segítségével teszteltem, majd az elkészíttetett eszközzel sikeresen működtettem az emulációt.

* **Kulcsszavak:**

ipari informatika, rendszer szimuláció, irányítás, HIL, PLC, technológia-visszafejtés

# Tartalomjegyzék

[Tartalmi összefoglaló 3](#_Toc478226792)

[Tartalomjegyzék 4](#_Toc478226793)

[Bevezetés 5](#_Toc478226794)

[1. A HIL szimulációhoz rendelkezése álló eszközök 7](#_Toc478226795)

[1.1. Az ipari folyamat emulációját lehetővé tevő programok 7](#_Toc478226796)

[1.2. Az emulált technológia és az irányítást végző eszköz közötti kapcsolat 7](#_Toc478226797)

[1.3. Az irányítást megvalósító programozható logikai vezérlő 7](#_Toc478226798)

[2. Az emuláció bemutatására kiválasztott ipari folyamat 8](#_Toc478226799)

[3. A BORIS használatának gyakorlati bemutatása 9](#_Toc478226800)

[3.1. A példaprogram ismertetése 9](#_Toc478226801)

[3.2. A példaprogram megvalósításának bemutatása 9](#_Toc478226802)

[3.3. A példaprogramhoz tartozó vizualizáció megvalósításának bemutatása 9](#_Toc478226803)

# Bevezetés

Az iparban használt rendszerek üzembe helyezése rendkívül magas költségekkel jár. A költségek egyik forrása a ráfordított idő, amit az ilyen rendszerek tervezésének és fejlesztésének körülményessége okoz. Sok esetben az automatizálást végző beágyazott rendszerek fejlesztése az irányítani kívánt rendszerre kapcsolva történik, ami a körülményektől függően akár azt is megkövetelheti, hogy az ilyen munkálatok a beüzemelés helyszínén történjenek. Az ilyen nehézségek csökkentése érdekében ma már HIL (Hardware-In-the-Loop) szimulátorok is alkalmazhatók, amik nem csak az irányítani kívánt rendszerek helyettesítését képesek ellátni a fejlesztési munkálatok alatt, de a napjainkban egyre kiemeltebb szempontot képviselő minőség biztosításában is jelentős szerepet kaphatnak. A végtelenítve futó automata-teszt rendszerek információt adnak a rendszer megbízhatóságáról, amik a HIL szimulátorok alkalmazásával költséghatékonyan megvalósíthatók. Mindezek mellett a HIL szimulátorok jól alkalmazhatók az oktatásban és a különböző képzési programokban is.

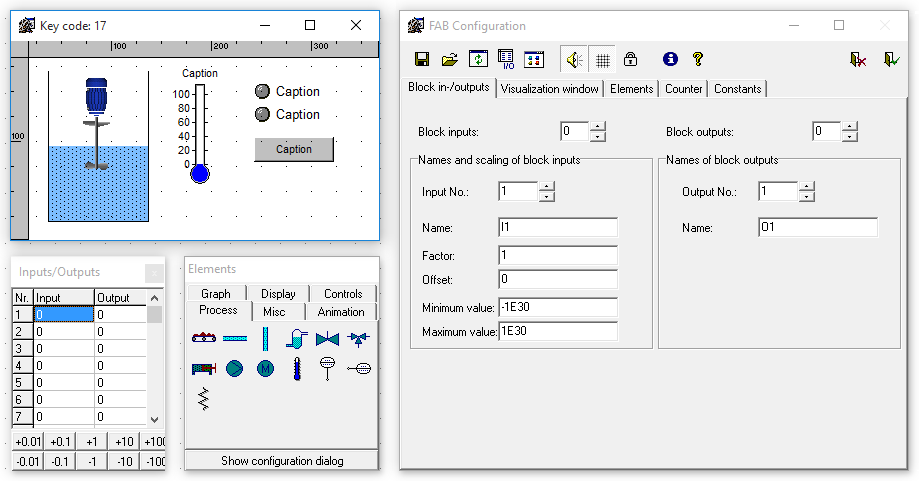
A diplomamunkám elsődleges célja egy létező HIL szimulátor tanulmányozása, a használatának a bemutatása egy konkrét példán keresztül és egy lehetséges helyettesítő eszköz megtervezése. A bemutatásra szánt példa egy karosszéria gyártósor lesz, aminek az irányítását az ipari automatizálásban elterjedt módon, egy programozható logikai vezérlővel fogom megvalósítani. A helyettesítő eszköz megtervezéséhez a helyettesíteni kívánt I/O interfészt annak megbontása nélkül fogom tanulmányozni, ellenkező esetben ugyanis az I/O interfész elvesztené a garanciális javítás lehetőségét. A másik célom a helyettesítő eszköz megvalósítása és összehasonlítása a helyettesíteni kívánt I/O interfésszel alkalmazhatósági és ráfordítási szempontokból, illetve a bemutatásra szánt példa futtatásával mindkét eszközön.

Az 1. fejezetben azokat az eszközöket mutatom be, melyeket a Műszaki Informatika Tanszék biztosított a HIL szimuláció megvalósítására. Ezeknek az eszközöknek a tanulmányozását Görbedi Ákos kollégámmal együtt végeztük. A 2. fejezetben ismertetem az emulálni kívánt ipari folyamatot, a karosszéria gyártósort, majd a 3. fejezetben a vizualizációra fókuszálva egy gyakorlati példán keresztül mutatom be a BORIS használatát. A 2. fejezetben ismertetett technológia és a hozzá tartozó irányítás megvalósítását a 4. fejezetben foglalom össze. Az 5. fejezetben a BORIS projektek hordozhatóságának problémájáról és az általam megvalósított megoldásról írok. Ezeket követően a 6. fejezetben rátérek az alkalmazott I/O interfész és az általam tervezett helyettesítő eszköz műszaki jellemzőire, majd a 7. fejezetben összegzem a diplomamunka készítése során szerzett tapasztalatokat és az elért eredményeket.

# A HIL szimulációhoz rendelkezése álló eszközök

## Az ipari folyamat emulációját lehetővé tevő programok

FAB…



1.1. ábra: Flexible Animation Builder

## Az emulált technológia és az irányítást végző eszköz közötti kapcsolat

## Az irányítást megvalósító programozható logikai vezérlő

# Az emuláció bemutatására kiválasztott ipari folyamat

# A BORIS használatának gyakorlati bemutatása

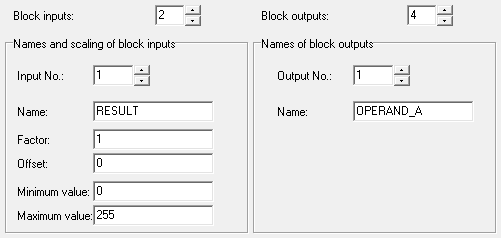
## A példaprogram ismertetése

## A példaprogram megvalósításának bemutatása

## A példaprogramhoz tartozó vizualizáció megvalósításának bemutatása

A 3.2. fejezet jól példázza, hogy a BORIS-ban készült szimulációk nem feltétlenül igényelnek vizualizációt, az eszköztára számos olyan blokkot biztosít, amikkel a felhasználó kölcsönhatásba kerülhet a futó szimulációkkal. Ennek ellenére a vizualizációk sokkal áttekinthetőbb interfészeket biztosíthatnak a szimulációk számára, illetve lehetőségeket adnak a rendszerek könnyebb megértését biztosító animációk elkészítésére is.

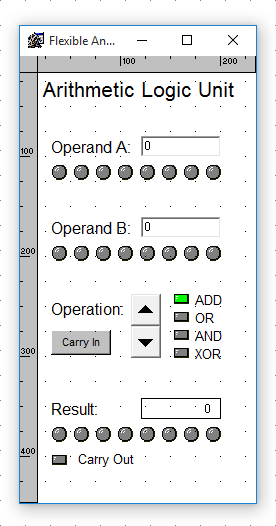
Egy új vizualizáció létrehozásához a blokkdiagramon a *Blocks 🡪 User-DLL-Block 🡪 Flexible Animation Builder* menüponton keresztül elérhető blokkot kell elhelyezni. Ez a blokk az eszköztár *User* fülén is megtalálható ugyanilyen néven. A vizualizáció szerkesztéshez elő kell hívni a FAB szerkesztőfelületét (1.1. ábra), ami az imént említett blokk szerkesztőablakából a *Dialog…* gombra kattintva érhető el. Itt legjobb első lépésként beállítani a be- és kimeneteket, ami a *FAB Configuration* ablak *Block in-/outputs* fülén végezhető el. A példában a vizualizáció legyen egyben beviteli és megjelenítő felület is. Ehhez kettő bemenetre és négy kimenetre lesz szükség (3.1. ábra).



3.1. ábra: a vizualizáció be- és kimeneteinek beállító felülete

Az első bemenet fogadja majd az ALU eredményét, a másik bemenet pedig az átvitel jelzőbitjét. A négy kimenet közül az első kettő szolgáltatja az operandusokat, a harmadik a művelet sorszámát, a negyedik pedig a kezdeti átvitelt jelző bitet.

A munkafelületre új elemeket az *Elements* ablakban megjelenő eszköztárról lehet behelyezni. Amennyiben ez az ablak nem jelenne meg, a munkafelületen előhívható helyi menüből előtérbe hozható az *Elementfenster* menüpontra kattintva. Az operandusok beviteléhez két beviteli mezőre lesz szükség, melyek a *Controls* fülön *Edit field* néven találhatók meg. Emellett a logikai műveletek jobb szemléltetése érdekében az operandusok és az eredmény binárisan is meg lesznek jelenítve, amikhez LED elemekre is szükség lesz (3.2. ábra). A LED-ek a *Display* fülön *LED* néven találhatók meg.



3.2. ábra: az ALU példaprogram vizualizációja

A műveletek közötti váltást egy léptető nyomógombpár fogja megvalósítani (3.2 ábra), ami a *Controls* fülön található *Spin element* néven. Ugyanezen a fülön található *Switch* elem egy kapcsoló, amelyik majd a kezdeti átvitel jelzőbitjét fogja jelenteni. Az eredmény megjelenítése a *Display* fülön levő *Numerical field* elemmel valósítható meg. A címkézés vagy a különböző szerepet ellátó LED-ek eltérő elemekkel való jelölése egyéni döntés kérdése, a vizualizáció működésében nem játszanak szerepet. A címkék a *Graph* fülön *Static text* néven szerepelnek, a négyszögű LED-ek pedig a *Display* fülön *Rectangular LED* néven találhatók meg. A vizualizáció működésének beállítása előtt célszerű az említett elemeket először a 3.2 ábra szerint vagy egyéni elrendezés alapján behelyezni.

Az elemek beállításai a méreteiken és a színeiken túl kiterjednek arra is, hogy egy-egy elem valamelyik paramétere melyik be- vagy kimenettel áll kapcsolatban. A munkafelületre behelyezett elemek listája a *FAB Configuration* ablak *Elements* fülén található meg. A listában szereplő elemek tulajdonságai a lista melletti *Element properties* táblázatban módosíthatók. Ez a táblázat mindig a listában kijelölt elem tulajdonságait jeleníti meg. Itt a működtetés szempontjából a legfontosabb az operandusokat biztosítók beviteli mezők, a műveletet meghatározó nyomógombpár és az eredményt kijelző számmező összekötése a megfelelő ki- vagy bemenetekkel. A beviteli mezők és a nyomógombpár esetén a kimenetet az *Output* tulajdonság határozza meg. Ebbe a mezőbe a megfelelő kimenet sorszámát kell beírni. A beviteli mezőknél az engedélyezett értékek tartományát a *MinValue* és a *MaxValue* tulajdonságok szabályozzák, a nyomógombpár esetén ugyanezt a *Min* és *Max* tulajdonságok látják el. Mivel az ALU csak négy műveletet tud elvégezni a nyomógombpár *Min* értéke az 1 lesz, a *Max* pedig a 4, a léptetés nagysága – *Increment* – pedig szintén 1 lesz. A számmező és az átvitelt jelző LED kapcsolatai hasonlóképpen állíthatók be, az *Input* tulajdonság segítségével. Az operandusokat és a kiválasztott művelet eredményét binárisan megjelenítő LED-ek esetében a kapcsolatot nem lehet ilyen egyértelmű módon megadni, mivel a FAB nem biztosít külön lehetőséget a bemenetek bitjeinek a közvetlen elérésére. Azonban a legtöbb tulajdonság kifejezéssel is megadható [1]. Egy egész szám n-edik bitje valójában a számnak és a 2 n‑edik hatványának osztásából származó érték 2-vel képzett maradékának az egész része (3.1).

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

A FAB egyik hiányossága, hogy nem rendelkezik a maradékképzéshez szükséges függvénnyel, ezért a maradékot a vele ekvivalens kifejezéssel [2] kell kiszámíttatni (3.3).

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |
|  |  | (3.3) |

A (3.3) kifejezés eredménye 0 vagy 1, ennek megfelelően a LED-ek *OnValue* tulajdonságainak 1-es értékűeknek kell lenniük. Ennek a tulajdonságának a nagyobb vagy egyenlő kiértékelése miatt a (3.3) kifejezésből elhanyagolható a maradékképzés eredményének tört részét levágó művelet. A LED-ek *Input* mezőibe az elfoglalt helyüktől függően a 3.1. táblázat szerinti kifejezéseket kell beírni, ahol „O1” az első kimenetet jelenti. A kimeneteket a FAB kifejezésekben az „O” betűjel vezeti be, a bemeneteket pedig az „I” betűjel.

|  |  |
| --- | --- |
| Megjelenítendő bit | Kifejezés |
| 0 | O1 - (2 \* INT(O1 / 2)) |
| 1 | (O1 / 2) - (2 \* INT((O1 / 2) / 2)) |
| 2 | (O1 / 4) - (2 \* INT((O1 / 4) / 2)) |
| 3 | (O1 / 8) - (2 \* INT((O1 / 8) / 2)) |
| 4 | (O1 / 16) - (2 \* INT((O1 / 16) / 2)) |
| 5 | (O1 / 32) - (2 \* INT((O1 / 32) / 2)) |
| 6 | (O1 / 64) - (2 \* INT((O1 / 64) / 2)) |
| 7 | (O1 / 128) - (2 \* INT((O1 / 128) / 2)) |

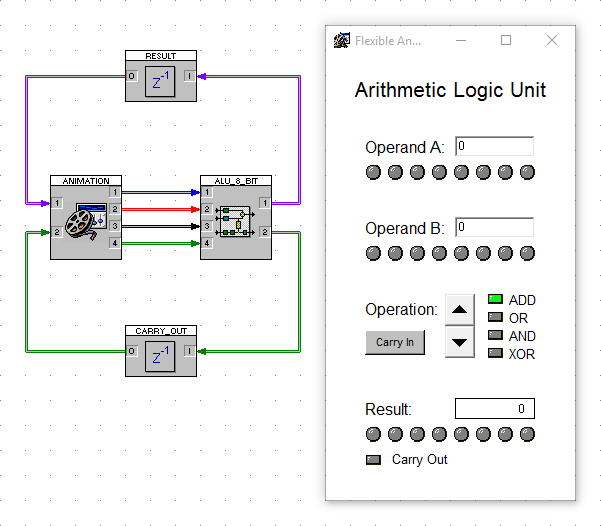
3.1. táblázat: a LED-ek ki- és bekapcsolását végző utasítások a FAB-ban.

A műveletek kijelzésénél nem a művelet sorszámának bináris reprezentációjára van szükség, hanem a sorszámnak megfelelő LED felgyújtására. A LED-ek *OnValue* tulajdonságainak nagyobb vagy egyenlő kiértékelése miatt minden ilyen LED *Input* mezőjében olyan kifejezésre van szükség, amelyiknek a hozzá tartozó *OnValue* érték a maximuma. A (3.4) kifejezés egy lehetséges megoldása az említett problémának, ahol „x” a be- vagy kimenetnek a szimbóluma, az „n” pedig az az érték, amelyikre az adott LED-nek fel kell gyulladnia.

|  |  |  |
| --- | --- | --- |
|  |  | (3.4) |

A (3.4) kifejezésben, ha az „x” értéke kisebb, mint „n”, az *Input* értéke az *OnValue* értékénél kisebb pozitív valós szám lesz. Amennyiben az „x” és az „n” megegyeznek az *Input* is meg fog egyezni az *OnValue* értékével. Abban az esetben pedig, ha az „x” nagyobb az „n” értékénél, az *Input* értéke negatív valós szám lesz.

A FAB külön fájlokba is képes az elkészített vizualizációt kimenteni. Ez a funkció a vizualizációk újrahasznosításakor lehet hasznos, azonban ettől függetlenül a vizualizáció a tartalmazó BORIS fájl mentésekor szintén mentésre kerül a tartalmazó fájlba. Az elkészült vizualizáció és a 3.2. fejezetben bemutatott szuperblokk közvetlenül nem összeköthető, ugyanis a BORIS algebrai huroknak fogja venni, ahol a kimenetek és a bemenetek körkörösen függenek egymástól. Ez könnyen feloldható a *Unit delay* blokk segítségével, amelyik a Blocks 🡪 Dynamics 🡪 Unit delay menüpont segítéségével helyezhető be a blokkdiagramba (3.3. ábra). Ez egy mintavételező elem, amelyik alapértelmezetten 1 másodperces időközönként vesz mintát a bemenetéről, de ez csökkenthető.



3.3. ábra: az ALU példaprogram az elkészült vizualizációval.

A bemutatott vizualizáció megtalálható a mellékelt CD *Symulation* könyvtárának *examples* alkönyvtárában *alu\_8bit\_visualization.fab* néven.

# A karosszéria gyártósor emulációja és irányítása

## Az emuláció megvalósításának bemutatása

### A robotok és a futószalagok vizualizációja FAB segítségével

### A robotok mozgatása

### A karosszériák mozgatása

### A cellák ellenőrzőlogikái

### Különbségek az eredeti feladathoz képest

### A karosszéria gyártósor teljes vizualizációja

## Az irányítás megvalósításának bemutatása

### A cellák irányítása

### Az operátor beavatkozásaira reagáló irányítás

# A BORIS projektek hordozhatósági problémája és megoldása

A fejlesztői munkákban rendkívül fontos a verziókezelés [3], aminek a megvalósítására ma már több ingyenesen igénybe vehető eszköz is létezik. (A diplomamunka verziókezelése a Git szoftverrel történt, a GitHub szolgáltatásainak igénybevételével.) A verziókezelés nem csak a végtermék kiadott verzióinak menedzselése, biztonsági mentések előállítása és elágazások (branches) esetleges létrehozása miatt kiemelt fontosságú, de a csapatban végzett, hatékony fejlesztői munka gyakorlatilag elképzelhetetlen nélküle.

Egy BORIS projekt verziókezelése ugyanúgy megvalósítható, mint bármely más projekté, az állományai ugyanis rendkívül kisméretűek és karakteres alapúak. Ezek alapján még a csapatban végzett fejlesztés is elképzelhető, ezzel szemben azonban a BORIS egy igen kellemetlen akadályt emel: a több fájlból álló projekteket más útvonalra mozgatva az állományokon belül kereszthivatkozások célt tévesztenek. Amennyiben a fejlesztői csapat minden egyes tagja ugyanazon az útvonalon tárolja a projekt állományait, az említett hiba nem jelentkezik, azonban így a végtermék is csak a fejlesztés során alkalmazott útvonalon futtatható.

A BORIS által létrehozott állományok abszolút útvonalakat tárolnak, ugyanakkor a BORIS nem képes a relatív útvonalak feloldására sem. Ennek a megoldására a szoftver látszólag nem biztosít egyértelmű lehetőséget, azonban az általa létrehozott állományok karakteres mivolta miatt viszonylag egyszerűen írható olyan segédprogram, amelyik korrigálja az állományokban tárolt abszolút útvonalakat.

A diplomamunka keretein belül a BORIS projektek hordozhatósági problémáját korrigáló segédprogram C++ nyelven lett megvalósítva Visual Studio 2015 fejlesztőkörnyezetben, a forrása pedig megtalálható a mellékelt CD *BORIS Tools* könyvtárának *BORISTeleporter* alkönyvtárában. Ez a program egy BORIS projekt gyökérkönyvtárában futtatva bejárja az alkönyvtárakat, összegyűjti a projekt állományainak abszolút útvonalait, majd mindegyiket megvizsgálva felülírja a bennük tárolt hivatkozásokat.

A BORIS által létrehozott állományok tartalmai elsősorban a szoftver számára szolgáltatnak adatokat, melyek alapján fel tudja építeni a blokkdiagramokat, azonban szövegszerkesztőben megnyitva az emberi szem számára is értelmezhető tartalommal rendelkeznek, ahogy azt az 5.1. táblázat is mutatja.

|  |
| --- |
| [Block #5]  383 SUPERBLOCK  ALU  175 575 246 659 100  4  2  0 0 0 0  E:\SZTE\Diplomamunka\Emulated System\Emulation\common\alu\_8bit.SBL  0  [Block #6]  430 BLOCKBUTTON  OP\_A\_B7  25 25 96 64 100  0  1  0 0 0 0  0  0 |

5.1. táblázat: Részlet egy a BORIS által létrehozott állományból.

Az 5.1. táblázat két blokk leírását mutatja be, melyek közül a *Block #5* jelzésű az *alu\_8bit.SBL* állományban tárolt blokkdiagramot foglalja magába. Jól látható, hogy az *alu\_8bit.SBL* állományra való hivatkozás abszolút útvonal formájában van jelen. Ahhoz, hogy az útvonalak korrigálhatók legyenek, a BORIS Teleporter először minden BORIS specifikus kiterjesztéssel rendelkező állományt megkeres a projekt könyvtárán belül és egy kulcs-érték tárolóba gyűjt, ahol a kulcs a fájl neve, az érték pedig az abszolút útvonala (5.2. táblázat).

|  |
| --- |
| **namespace** *fs* = boost::filesystem;  **const** fs::path applicationPath = fs::current\_path();  **const** std::set<std::wstring> extensions = {  *L*".bmp", *L*".BMP", *L*".bsy", *L*".BSY", *L*".fab", *L*".FAB",  *L*".sbl", *L*".SBL"  };  std::map<std::wstring, std::wstring> filenamesWithPath;  **for**(**auto** entry : recursive\_directory\_range(applicationPath)) {  **if**(extensions.find(entry.path().extension().c\_str()) !=  extensions.end())  filenamesWithPath[entry.path().filename().c\_str()] =  entry.path().c\_str();  } |

5.2. táblázat: A BORIS Teleporter forrásának egy részlete, amelyik megkeresi a BORIS specifikus fájlokat (beleértve a FAB fájlokat is és a képeket is).

A projekt könyvtárában levő állományok útvonalainak begyűjtése után a BORIS Teleporter soronként átvizsgál minden egyes állományt – a képfájlok kivételével – és felülír minden olyan útvonalat, amelyik valamelyik megtalált állományra hivatkozik (5.3. táblázat). Azok az útvonalak, amelyek a projekt könyvtárán kívülre hivatkoznak, változatlanok maradnak.

|  |
| --- |
| std::wstring::size\_type  *findPathBeginning*(**const** std::wstring& fullString,  std::wstring::size\_type offset = 0);  std::wstring::size\_type  *findPathEnd*(**const** std::set<std::wstring>& extensionSet,  **const** std::wstring& fullString,  std::wstring::size\_type offset = 0);  *// A megtalált állományok feldolgozása a képek kivételével.*  **for**(**auto** fileEntry = filenamesWithPath.begin();  fileEntry != filenamesWithPath.end(); ++fileEntry) {  **if**(fs::path(fileEntry->first).extension() == *L*".bmp" ||  fs::path(fileEntry->first).extension() == *L*".BMP")  **continue**;  *// Az eredeti állomány átnevezése és új állomány nyitása a*  *// a módosított tartalomnak.*  fs::rename(fs::path(fileEntry->second),  fs::path(fileEntry->second + *L*".orig"));  std::wifstream borisOrigFile(fileEntry->second + *L*".orig");  std::wofstream borisFile(fileEntry->second);  *// Az aktuális állomány soronkénti átvizsgálása.*  std::wstring line;  **while**(std::getline(borisOrigFile, line)) {  *// Az aktuális sorban minden egyes útvonal megkeresése*  *// és korrigálása, ha szükséges.*  **auto** pathBegin = findPathBeginning(line);  **auto** pathEnd = findPathEnd(extensions, line);  **while**(pathBegin != std::wstring::npos &&  pathEnd != std::wstring::npos) {  **const** **auto** previousSize = line.size();  fs::path reference(line.substr(pathBegin,  pathEnd - pathBegin));  *// A megtalált útvonal felülírása, ha szükséges.*  **const** **auto** it = filenamesWithPath  .find(reference.filename().c\_str());  **if**(it != filenamesWithPath.end())  line.replace(pathBegin,  pathEnd - pathBegin, it->second);  *// A keresés eltolásának korrigálása a megváltozott*  *// sorhossznak megfelelően.*  **if**(previousSize < line.size())  pathEnd += line.size() - previousSize;  **else** **if**(previousSize > line.size())  pathEnd -= previousSize - line.size();    *// Új útvonal keresése az aktuális sorban.*  pathBegin = findPathBeginning(line, pathEnd);  pathEnd = findPathEnd(extensions, line, pathEnd);  }    *// A korrigált sor beleírása az új állományba.*  borisFile << line << std::endl;  }  } |

5.3. táblázat: A BORIS Teleporter forrásának egy részlete, amelyik minden megtalált állományban módosítja az útvonalakat, hogy azok megfeleljenek a hivatkozott állományok valódi útvonalainak.

A FAB által létrehozott állományok szerkezete eltér attól, mint amilyent a BORIS használ (5.1. táblázat), éppen ezért a BORIS Teleporter egy soron belül több útvonal meglétét feltételezi, ahogy az a FAB által létrehozott állományokra jellemző.

A BORIS Teleporter ugyan megoldja a BORIS projektek hordozhatósági problémáját, azonban a hatékony verziókezeléshez érdemes a szükségtelen módosítások elkerülésére törekedni, ennek a programnak a használata viszont több fős munka esetén minden módosításhoz hozzáfűzheti az útvonalak módosítását is. A BORIS Teleporter tovább fejlesztésével ez a probléma is megoldható, azonban ez a funkció nem része a diplomamunkának.

# Az I/O interfész helyettesítési lehetőségei

A BORIS fejlesztőkörnyezet és a külvilág közötti kapcsolat megvalósításhoz a Műszaki Informatika Tanszék egy Lucas Nülle CO3715-1H típusú I/O interfészt biztosított. Ez az eszköz elfogadhatónak bizonyult a fejlesztések során, azonban a magas költségei miatt megfontolandóvá vált az esetleges helyettesítési lehetőségek felkutatása.

Mivel a készülék a Műszaki Informatika Tanszék tulajdona, ugyanakkor érvényes garancia alatt áll, a működésének a megismerése csak a megbontása nélkül valósítható meg. Ez azt jelenti, hogy csak az I/O interfész és a számítógép közötti kommunikációra lehet hagyatkozni, ugyanis ez az a réteg, amelyik megbontás nélkül is információt szolgáltathat a két eszköz közötti információcseréről.

## BORIS és az I/O interfész közötti kommunikáció

Az I/O interfész a számítógéppel való kommunikációra soros adatátvitelt használ, RS-232 [4] adatátviteli szabvánnyal. Ennek a ténye leolvasható a készülék előlapjáról, illetve az adatvezetékének DE-9 [5] típusú csatlakozója is csak a számítógép soros portjába illeszthető. Ennek a fajta adatátvitelnek a lehallgatására több kiváló módszer is létezik. Mára már az igényesebb oszcilloszkópok is rendelkeznek a szabványos kommunikációs protokollok dekódolására alkalmas funkciókkal, azonban a költséges eszközök mellett ez a feladat szoftveresen is megoldható.

A diplomamunka keretein belül a BORIS és az I/O interfész közötti kommunikáció lehallgatására az Eltima Software által fejlesztett Serial Port Monitor program került felhasználásra. Ez a program 15 napig ingyenesen is használható, és alkalmas a soros portot kezelő meghajtóprogramnak elküldött beállítások (pl. baudráta) megjelenítésére is.

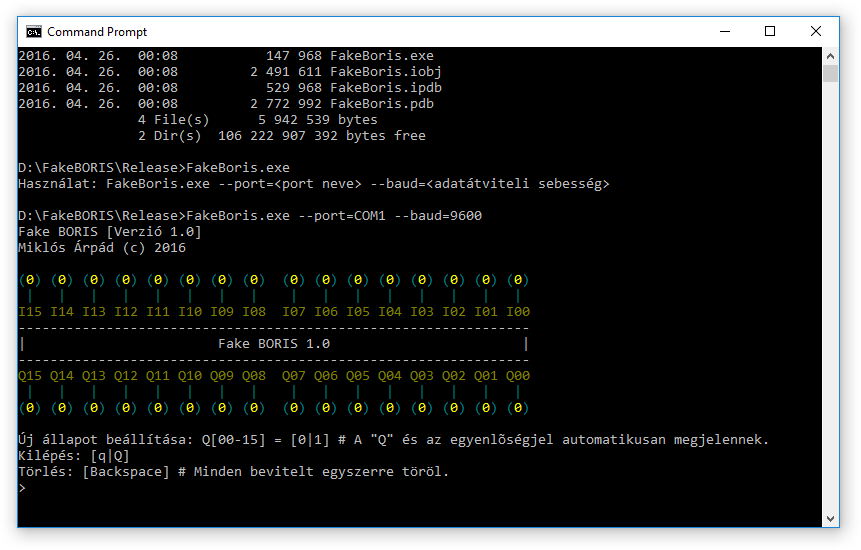
A kommunikáció lehallgatása során a következő információk kerültek megállapításra:

* A BORIS és az I/O interfész 9600 baudos kapcsolatot épít ki, ahol a karakterszélesség 8-bit, nincs paritás és 2 stop bit jelzi minden adat végét.
* A számítógép és az I/O interfész mester-szolga viszonyban állnak, ahol mindig a számítógép kezdeményezi az adatcserét.
* Másodpercenként átlagosan 50 adatcsere történik, ami azt jelenti, hogy ideális esetben a BORIS felületén történő változások 20 milliszekundumos mintavételezéssel jelennek meg az I/O interfész kimenetén és fordítva.
* Az első utasítás, amit a számítógép kiküld a 0x02, amire az I/O interfész visszaadja a típusazonosítóját, mint nullával zárt karakterlánc: SO3715-1H.110
* Az I/O interfész azonosítását követően a számítógép folyamatosan ugyanazt a két utasítást küldi az eszköznek:
  + 0xb9 – paraméter nélküli utasítás, hatására az I/O interfész két byte-on visszaadja a bemeneteiről leolvasható logikai szinteket.
  + 0xba – két byte-os paraméterrel rendelkező utasítás, hatására az I/O interfész a paraméterben levő biteknek megfelelően módosítja a kimeneteinek a logikai szintjeit.
* Az I/O interfész kimenetein beállítandó állapotokat és a bemeneteinek az aktuális állapotait reprezentáló byte-ok közül az első a 15-8 sorszámú ki- vagy bemeneteket jelentik, a második a 7-0 sorszámúakat.

### A BORIS kommunikációs jeleit utánzó tesztprogram

A BORIS és az I/O interfész közötti kommunikáció lehallgatásából származó eredményeket mindkét oldalról tesztelni kell a levont következtetések igazolásának érekében. A tesztelés első lépéseként egy olyan számítógépes program került megvalósításra, amelyik képes az I/O interfész számára azt a látszatot kelteni, hogy a BORIS kommunikál vele.

A tesztprogram első verziója C++ nyelven íródott a Boost függvénykönyvtár felhasználásával (6.1. ábra). Ennek a szerepe kizárólag a korábban említett következtetések igazolása, a kényelmes használat pusztán másodlagos szerepet kapott. A futtatásához ismerni kell annak a soros portnak a nevét, amelyikre az I/O interfész csatlakoztatva lett, illetve meg kell határozni az alkalmazni kívánt baudrátát. A kimenetek működtetése utasítások bevitelével lehetséges, ugyanakkor a bemeneteken történt változások automatikusan megjelennek a karakteres grafikát alkalmazó felületen.



6.1. ábra: a FakeBORIS 1.0 futás közben.

A program egyik kulcsfontosságú, ám a BORIS és az I/O interfész között feltárt kommunikáció igazolását nem befolyásoló tényező, a soros port aszinkron kezelése, egy harmadik féltől származó forráskóddal [6] lett megvalósítva. Ez a forráskód a Boost.Asio függvénykönyvtárra épül és megkönnyíti a Boost.Asio által biztosított soros port kezelés használatát. Az aszinkron kommunikáció elkülöníti az I/O interfész kimeneteit működtető byte-ok előállítását és kiküldését a bemenetek állapotait jelző byte-ok fogadásától és a programban történő megjelenítésétől. Ez a megoldás jelentősen leegyszerűsíti a fejlesztést, azonban arról nem szolgáltat információt, hogy a kiküldött utasításra érkezett-e válasz. Ez a probléma ebben a programban nem kerül megoldásra.

A Fake BORIS 1.0 forráskódja megtalálható a mellékelt CD *BORIS Tools* könyvtárának *FakeBORIS* alkönyvtárában. A projekt könyvtárstruktúrája a következő:

* **app** – a tesztalkalmazást létrehozását és futtatását implementáló fejléc- és forrásfájlok gyökérkönyvtára.
  + **src** – a tesztalkalmazás forrásfájljainak könyvtára. Ebben található az a forrásfájl is, amelyik megvalósítja a **main** függvényt.
* **lib** – az üzleti logikát és a használatát biztosító interfészt implementáló fejléc- és forrásfájlok gyökérkönyvtára.
  + **inc** – az üzleti logika fejlécfájljainak könyvtára. Ebben találhatóak azok a fájlok, melyek interfészként szolgálnak a Fake BORIS beágyazásához egy tetszőleges alkalmazásba.
  + **src** – az üzleti logika forrásfájljainak könyvtára, melyek megvalósítják az interfész mögötti logikát.
* **thirdparty** – a harmadik féltől származó források gyökérkönyvtára. Az ebben található könyvtárak érintetlen formában tartalmazzák mindazokat, melyeket a harmadik fél fejlesztett és tett közzé.

Az üzleti logikát ebben a projektben a **FakeBoris** osztály valósítja meg, a publikus interfészét a 6.1. táblázat szemlélteti. A **start** függvénye után megjelenik a konzolképernyőn a karakteres szemléltető grafika és az I/O interfész bemenetein történő változások folyamatosan kirajzolódnak rajta a **stop** függvényének a meghívásáig. Annak érdekében, hogy a felhasználó logikai szinteket tudjon beállítani az I/O interfész kimenetein, a **readUntil** függvényt kell meghívni. Ez a függvény addig blokkolja a hívó szálat, amíg a felhasználó le nem nyomja a paramétereként megadott billentyűt. A megvalósított tesztprogramban ezt a szerepet a *Q* billentyű tölti be.

|  |
| --- |
| **class** FakeBoris: **public** AbstractScheduler {  **public**:  */\*\* Példányosítja az FakeBoris osztályt. \*/*  FakeBoris(**const** std::string& device,  **unsigned** **int** baud\_rate);  */\*\* Felszabadítja a memóriát a létrehozott példány*  *megszűnésekor. \*/*  **virtual** ~*FakeBoris*();  */\*\* Kirajzolja a konzolképernyőre az I/O interfészt ábrázoló*  *karakterképet, inicializálja a soros portot és elindítja az*  *I/O interfésszel történő ismétlődő adatcserét. \*/*  **void** start();  */\*\* Leállítja a soros ponton keresztüli ismétlődő*  *adatcserét és lezárja a portot. \*/*  **void** stop();  */\*\* Folyamatosan bekéri a felhasználótól a kimenetek*  *állapotai megváltoztató paramétereket a megadott megállító*  *karakterig. \*/*  **void** readUntil(**char** stop\_char);  }; |

6.1. táblázat: a FakeBoris osztály publikus interfésze.

Az **AbstractScheduler** őszosztály az ismétlődő műveletek végrehajtását szervezi külön osztályba. Ennek az osztálynak nincs publikus interfésze és önmagában nem is példányosítható.

A Fake BORIS sikeresen megvalósította a kitűzött célokat, azonban az aszinkron kommunikációból fakadó hiányosságai és a kényelmetlen a használata miatt szükségessé vált egy grafikus kezelői felülettel (GUI) rendelkező verzió elkészítése is.

### Az I/O interfész működését utánzó PIC mikrovezérlő

A BORIS és az I/O interfész között feltárt kommunikációnak a számítógép oldaláról történő sikeres igazolását követően…

## Az I/O interfészt helyettesítő elektronika

### A helyettesítő elektronika kapcsolási rajza

### A be- és kimenetek áramköreinek szimulációja

### A helyettesítő elektronika nyomtatott áramköre

## A megvalósított helyettesítő eszköz bemutatása és tesztelése

### A helyettesítő eszköz előállítási költségei

### Jelkésleltetések és logikai szintek mérése PicoScope segítségével

### Az emulált ipari folyamat működtetése a helyettesítő eszközzel