Önálló laboratórium dokumentáció

Automatizálási és Alkalmazott Informatikai Tanszék
2023/2024. 2. félév

Készítette:	Landi András
Neptun-kód:	IJK23N
Ágazat:	Számítógép alapú rendszerek
E-mail cím:	landiandras@gmail.com
Konzulens:	Oláh István
Dátum:	2024.05.29.

Téma címe: Siemens S7-1200 PLC programozása

Feladat

A feladat egy 2 aknás és 6 emeletes (földszint + 5 emelet) lift vezérlésének megtervezése és elkészítése a Siemens TIA Portal fejlesztőkörnyezetében egy S7-1214-es PLC-re. A megvalósításhoz az SCL programozási nyelv alkalmazása. A vezérlés mellé továbbá a lift működési folyamatát szimuláló program készítése, szintén SCL nyelven. A kész vezérlést és szimulációt bemutatni képes felhasználói felület kialakítása az operator panelen (KTP 700 basic). Minden emeleten van elhelyezve felfelé és lefelé hívásra alkalmas gomb (kivéve nyilván a földszinten, ahol lefelé nem lehet hívni, illetve az 5. emeleten, ahonnan felfelé menni értelmetlen)

1. Bevezetés, a laboratóriuma munka környezetének ismertetése

1.1 Elméleti alapok

A PLC (Programmable Logic Controller) egy nagy megbízhatóságú, determinisztikus működésű mikroprocesszor alapú vezérlő egység. Gépek, folyamatok automatizálásának céljából lett kifejlesztve, helyettesítve a bonyolultabb logikai hálózatokon, relés, vagy mechanikus reteszelőrendszereket használó vezérléseket.
Jelentős előnye ezekhez képest, hogy sokkal könnyebbé teszi a rendszer módosítását, nem szükséges fizikailag átszerelni alkatrészeket, lehetséges a kódon változtatni, sőt modern rendszerek esetén erre már a folyamat leállítása nélkül is van mód, működés közben lehet frissíteni a vezérlést.

A PLC-k periódikus működésű eszközök, minden periódusban beolvassák bemeneteiket, végrehajtják a bennük található programot, majd a kimeneteket állítják ezek alapján.

A PLC programok úgynevezett blokkokból épülnek fel. Ezek típusai az Organization Block (OB) – főleg programszervezési célokra, Function (FC) – függvények, melyek saját adatterülettel nem rendelkeznek, a globális adatterületet tudják használni és Function Block (FB) – saját adatterülettel rendelkező programrészek. A különböző blokkok képesek egymást hívni, maximum 16 hívás mélységéig. Minden blokkhoz tartozik egy szám, ez nagyrészt tetszőleges, ám van néhány kitüntetett szereppel rendelkező, a feladat megoldása szempontjából lényeges:
OB1 – főprogram, végtelen ciklus, amely adott időközönként (ciklusidő) lefut.
OB100 - indítási blokk, az induláskor egyszer fut le.

Vannak beépített időzítőt, számlálót, komparátort, matematikai függvényeket, bitoperátorokat és még sok más funkciót megvalósító funkcióblokkok, melyből a program írása során tetszőleges mennyiségűt lehet létrehozni, de figyelni kell a megfelelő adatterületek beállításánál.

A globális adatterületen úgynevezett címkéket lehet elhelyezni, melyeknek tetszőleges memóriacímeket lehet adni, ezek aféle globális változóként használhatóak. Fontos, hogy a nyelv engedi, hogy a felhasználó kétszer ugyanazt a memóriacímet felhasználja, esetleg egy figyelmeztetést ad, de a program le fog futni, viszont nyilvánvalóan problémákat okoz, ha egy változót több helyről is felülírunk.

A PLC -ben használt adatterületek: Input(I), Output(Q), Memory(M). Rengeteg féle adattípust támogat, néhány a feladat megoldása szempontjából fontosabb: bitek és bitsorozatok, egész számok, lebegőpontos számok, karakterek, sztringek és különféle tömbök és struktúrák.

Az SCL (Structured Control Language) nyelv, a Siemens által implementált verziója az IEC 61131-3 szabványban leírt ST (Structured Text) Pascal alapú szöveges programozási nyelvnek. Lényeges előnye az iparban többször előforduló grafikus programnyelvekkel szemben (pl. létradiagram, funkcióblokk), hogy lehetővé teszi bonyolultabb, komplexebb logika megvalósítását, átláthatóságát, mivel magasabb szintű nyelvi elemeket és funkciókat (pl. for/while ciklusok, switchek) tartalmaz.

1.2 A munka környezete

A félév elején néhány a többi a témán dolgozó hallgatóval együtt lehetőség volt a tanszéki laborban megismerkedni az ott található S7-1214 PLC-vel, illetve a KTP 700 basic operator panellel, illetve az ezeket összekötő Profinet hálózat alapjaival.

A bevezetés után az önálló munkát otthonról, a valódi helyett egy szimulált PLC használatával tudtam végezni. A szimulációhoz, illetve a fejlesztés minden lépéséhez a Siemens TIA Portal V17 fejlesztőkörnyezetet használtam, mely lehetőséget nyújt program írására, fordítására, debuggolására, PLC-re töltésére, illetve a PLC és HMI (Human-Machine Interface, másik név az operator panelre) szimulációjára. Egy kis kellemetlenség, hogy a szimuláció csak V4.0 (vagy magasabb) firmware verziótól alkalmazható, viszont a tanszéken V3.0 található. A fejlesztőkörnyezet engedi az eszköz kicserélését, de csak ha a firmware verziószáma azonos vagy nagyobb, visszafelé nem lehetséges, ezért a feladat befejezésével szükséges volt a programot áthelyezni egy másik PLC-re a projekten belül, hogy a tanszéken található PLC-ken ki is lehessen próbálni.

1.3 Kiindulási állapot a félév elején

Az előző félévben az Ipari Irányítástechnika című tárgy keretei közt már megismerkedhettem a PLC-k elméleti alapjaival, illetve az IEC 61131-3 szabványban leírt programozási nyelvekről egy átfogó képet kaptam, de valódi hardverrel és operator panellel nem találkoztam ott, illetve komplexebb vezérlést nem valósítottam meg. Szintén új volt számomra a Siemens TIA Portal fejlesztőkörnyezete (bár megjegyzem a félév során 1-1 alkalommal sor került a használatár Mikrokontroller laboratórium és Laboratórium 2 tárgyakon).

2. Az elvégzett munka ismertetése

2.1 Tervezés

Az első lépés a program felépítésének megtervezése volt. Fontos szempont volt a feladat nyújtotta lehetőségek szerint legáltalánosabb és legkönnyebben módosítható kód felépítése. Ennek érdekében, igyekeztem a programot különválasztható egységekre bontani. Ezek első körben: Egy konkrét lift fülke vezérlése, a központi vezérlőegység, a szimulációt végrehajtó kód, illetve a felhasználói felület megfelelő működését elősegítő segédfüggvények.

Ezek mindegyike egy-egy különálló POU (Program Organization Unit – programszervezési egység, ezeket a Siemens Program Blocksnak nevezi), azaz mindegyiknek van saját feladata és felelőssége, melyek bemeneteket és kimeneteket várnak, ám más értelemben nem függenek egymástól. Ez azt is jelenti, hogy lehetséges például a szimulációs kódrészlet kivétele, de a vezérlés hibátlanul fog továbbra is működni. Ugyanez igaz fordítva is, lehetséges a vezérlést kicserélni bármilyen másikra, amíg a megfelelő kimeneteket valamilyen tetszőleges logika alapján állítja, a szimuláció működni fog.

2.2 Egy lift fülke működésének folyamata

A lift fülkék működéséért felelős vezérlést egy funkcióblokk formájában hoztam létre, mely azt is jelenti, hogy tetszőleges számban lehetséges ezeket példányosítani, tehát lehetne a feladatot bővíteni 3,4… akárhány liftakna esetére, sőt akár redukálni is egyetlen fülkére.

A fülkék általános működése érdekében bemeneteiken nem várnak minden információt a rendszerről, csupán a működésükhöz közvetlenül szükségeseket. Ez első körben a jelenlegi pozíciójuk, melyet közelségérzékelők alapján lehetséges megállapítani, illetve a következő szint, ahova szükséges az adott fülkének mennie. Hogy melyik a következő szint, a központi vezérlőnek

kell eldöntenie, nem a fülkének.
Ezeken kívül még néhány segédbitet is bevezettem, amik segítségével tud kommunikálni, jelezni bizonyos állapotokat/eseményeket a vezérlő felé. Ilyen egy bit mely az egyhelyben álló liftfülkének jelez, hogy nyissa ki az ajtajait, illetve egy bit pár, amin a fülke képes jelezni a vezérlőnek, ha teljesített egy parancsot, illetve egy ehhez tartozó nyugtázó bitet a központi vezérlő felől.

A vezérlő adja át továbbá a különböző érzékelők jeleit, ezek az ajtó nyitott, illetve csukott állapotát jelző közelségérzékelők, egy az ajtónál elhelyezett érzékelő mely képes jelezni, ha valaki az ajtót megfogja és persze az ajtó nyitó/csukó gombok jeleit.

Ezek alapján a lift fülke előállítja kimeneteit, melyek segítségével tudja mozgatni magát és az ajtókat működtetni. A felfelé és lefelé mozgáshoz is tartozik egy-egy jel, illetve az ajtó nyitásához és csukásához is.

A fülke vezérlését egy egészen egyszerű állapotgép valósítja meg. Habár a PLC rendelkezik nem felejtő memóriával és lehetséges lenne az állapotok eltárolása, biztonsági okokból úgy döntöttem, hogy a rendszer indításakor mindig a földszintre megy minden fülke, ahol csukott ajtóval várja a következő utasítást, avagy a nyomógombok segítségével természetesen nyitható az ajtaja. Egyszerű időzítők segítik a működést, ha parancs nélkül várakozik nyitott ajtóval valahol, akkor 20 másodperc elteltével bezárja ajtaját. Indításkor az ajtó a parancs megérkezése után 5 másodperccel zárja automatikusan az ajtaját (ezt az időzítőt az ajtó megfogása nullázza) vagy az ajtózáró gomb segítségével manuálisan is kiváltható az ajtó zárása. Ha az ajtó bezárult, a parancstól függően elindul a lift fel- vagy lefelé. Megérkezéskor egy olyan állapotba kerül, melynek során jelzi, hogy az adott parancs teljesült, és nem mozdul tovább, amíg a vezérlő ezt le nem nyugtázza.

A lift ajtaját megfoghatja egy ember, vagy belóghat valami az ajtóba. Ennek érzékelésére a valóságban talán egy nyomatékmérő lenne a legalkalmasabb, de az egyszerűség kedvéért a feladatban én egy egyszerű lézeres érzékelővel modelleztem, mely folyamatosan aktív jelet produkál, amíg meg nem szakítják a lézert, ekkor pedig alacsony logikai szintet mutat. Ha valaki megfogta a lift ajtaját, akkor az újra kinyílik, és újra kezdi az 5mp várakozást, hogy záródhasson, hacsak valaki meg nem nyomja az ajtózáró gombot, mely esetben azonnal újra próbálja a zárást.

2.3 Központi vezérlőegység

A liftfülke egyszerűsített állapotgráfja, néhány segédállapot elhagyásával

Mivel egy többaknás liftrendszerről beszélünk, a hívógombok lenyomásakor szükséges eldönteni, hogy melyik lift fülke szolgálja ki a hívást. Pontosan erre a célra van létrehozva egy központi vezérlőegység, mely a liftek helyzete, állapota és mozgási iránya alapján képes eldönteni minden beérkező hívásról, hogy melyik lift feladata legyen.

Itt egy kis kitérőt tennék, az alapvető gyűjtőalgoritmusról, mely 1 aknás esetben használatos. Egyszerűen annyi az algoritmus lényege, hogy a lift egészen addig halad egy irányba, amíg van további hívása/parancsa abba az irányba, avagy el nem éri az utolsó szintet az adott irányba, mely esetben pedig megáll, illetve irányt vált, ha az ellenkező irányban van hívás/parancs. Ez az algoritmus alapvetően használható több aknás esetben is, a liftek belsejében megnyomott gombok működtetésére tökéletesen megfelel. A probléma a szinteken elhelyezett hívógombokból adódik, melyekből rögtön kettő is van a legtöbb emeleten, felfelé és lefelé.

Az első gondolat a hívások kezelésére talán a két lift fülke távolságának kiszámolása, majd a közelebbi választása. Ez volt az első stratégia, amelyet megpróbáltam megvalósítani, ám egészen gyorsan nyilvánvalóvá vált, hogy ez több ponton is nehézkesen végrehajtható. Az első probléma, hogy figyelembe kell venni a liftek haladási irányát, az összes az liftnek bufferelt parancsot és a hívás irányát is, amely már önmagában is igencsak sok ‘if’ ágat igényel, például, ha egy lift a 3. emeleten megy felfelé, és a 2. emeleten felfelé hívnak liftet, akkor előfordulhat olyan eset, hogy ez a lift még felmegy az 5. emeletre, majd vissza le a földszintre, majd fel a 3. emeletre, csak hogy kiszolgálhassa azt a hívást. Sok másik ehhez hasonló eset fordulhat elő, és mindegyiknél a távolság kiszámolása nagyon bonyolult és követhetetlen folyamat lenne. A másik probléma ezzel a megközelítéssel, hogy egy gomb megnyomásakor rögtön kiosztaná valamelyik liftnek a lekezelését. Ez alacsony kihasználtság esetén nem gond, de ha mindkét lift folyamatosan használva van több szintről egyszerre, akkor előfordulhatnak olyan hívások, amelyeket menet közben kezelni tud egy lift, ezáltal nem dönthető el előre, hogy pontosan mennyi idő lesz mire odaér, lehet, hogy a gomb megnyomásakor kiszámolt távolság módosul mire odaérne a lift.

A probléma megoldására egy felépítés szempontjából sokkal egyszerűbb utat választottam. Az 1 aknás algoritmusból kiindulva, nem rendel minden hívást hozzá egy lifthez amint az a hívás beérkezik, hanem a hívásokat eltárolja mindaddig amíg egy lift ki nem szolgálja azokat. Ennek megvalósítása kód szinten 3 részből áll.

Az első eset, hogy ha mindkét lift egyhelyben várakozva áll, ekkor továbbra is szükséges egy távolságszámolás, de mivel álló liftekről van szó, ez is leegyszerűsödik egy szimpla különbségképzésre (lift helyzete – hívás szintje), pontosabban annak abszolút értékére, elvégre nem tudjuk, hogy a hívás egy adott lift alatt vagy felett van, de csak a távolság abszolút értéke számít.

A második lépésben az első lift állapotát kell ellenőrizni, hogy melyik irányba halad (vagy esetleg egyhelyben áll), és a legközelebbi hívás irányába kell küldeni. Amennyiben nincs hívás és nincs hova menni, az állapotát frissíteni kell, jelezve, hogy várakozik a lift. Amennyiben van hívás, az állapotát úgy kell állítani, hogy jelezze melyik irányba indult el. Ez nem változhat egészen addig amíg van az adott irányba feladata, ha ez elfogyott, akkor várakozó állapotba kerül, melyből természetesen lehet, hogy azonnal tovább is lép, mivel az eddigihez képest ellenkező irányban volt feladat.

Harmadik lépésben pontosan ugyanazt vizsgáljuk meg, mint a másodiknál, csupán az első lift helyett a másodikra. A kódrészletek sorrendje határozza meg a prioritási szinteket, az SCL nyelvben ugyanis a switch-ek mindegyike csak és kizárólag egy esetet értékel ki, utána automatikusan kilép a switch szerkezetből (tehát olyan, mintha C nyelvben minden case-hez írnánk egy breaket).
Ebből az is következik, hogy ha teljesen azonos mindkét liftnek az állapota és helyzete, az első fogja kapni a prioritást.

Ezzel a megoldással azt a viselkedést értem el, hogy egy már mozgó liftnek ha az útjába esik egy újonnan érkezett hívás, azt le fogja tudni kezelni, majd halad tovább az eredeti irányba. Továbbá egynél több hívás esetén minden esetben mozogni fog mindkét lift, akkor is, ha azonos irányba esik mindkettő, és le tudná kezelni igazából egyetlen fülke is, várakozó állapotban csak akkor lehet lift, ha nincs egyetlen parancs/hívás sem, esetleg egy van, de azt a másik fülke már kezeli. Kijelenthetem tehát, hogy ez a megoldás lényegesen leegyszerűsítette a vezérlő felépítését, elvégre csupán néhány switch szerkezet és feltételellenőrzés okosan megválasztott sorrendje számít. Hátránya sajnos szintén ez, a kódban bőven szerepel ismétlés, ezért egy kicsi módosításnál is könnyen előfordulhat, hogy sok helyen szükséges a kódot változtatni.

Fontos, hogy az operátor panelen elhelyezett gombok SET jellegű viselkedést mutatnak a megnyomás hatására, ezért a gombok megnyomása automatikusan eltárolódik. Más esetben szükség lenne egy külön bufferre, ahova el tudja tárolni a beérkezett kéréseket/hívásokat, de így közvetlenül az bemeneti jeleket használhatja fel.

Ezen felül a vezérlő feladata lekezelni, hogy ha egy lift teljesített egy parancsot. Ezt a lift egy jelzőbit segítségével tudja közölni, melyre a vezérlő minden végrehajtásakor ráellenőriz, jelzés esetén pedig az adott emeletet kiveszi a bufferelt hívások/feladatok listájából.

Ha ezeket a lépéseket végrehajtotta, egy-egy változóban el van tárolva a liftek következő parancsa, melyet átad a lifteknek a vezérlő ciklusának végén.

A vezérlő dolgát segíti továbbá egy segédfüggvény, melynek célja, hogy a liftek helyzetét jelző érzékelők bool típusú bemeneteiből egy az algoritmus számára feldolgozható egész számot alkosson, ami jelzi a liftek utolsó ismert helyzetét.

A központi egység felelős az egyes lift fülke modulok és az azokhoz tartozó ki és bemenetek közötti kapcsolat megteremtéséért.

2.4 Szimuláció

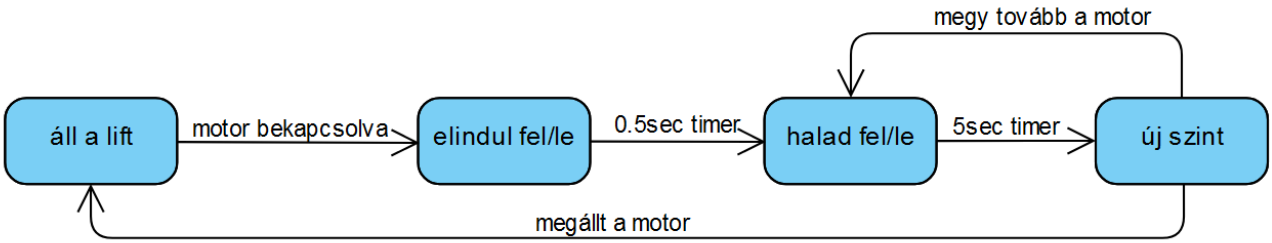
A vezérlés bemutatásához szimulálni szükséges bemeneteket. Ezt egy külön függvény blokk hajtja végre, mely a vezérlő és a lift fülkék kimenetei alapján generál bemeneti jeleket. Ezeket a jeleket úgy alakítottam ki, hogy egy valóságos modell lehetséges bemeneteire hasonlítsanak. Például könnyű megoldás lett volna szimplán egész számként eltárolni a liftek helyzetét, ez azonban a valóságban nem valószínű, hogy rögtön elérhető lenne, mint egy érzékelő kimenete. Sokkal hihetőbb, ha minden szinten elhelyezünk egy-egy helyzet érzékelőt (pl. kapacitív vagy induktív vagy más elven működő közelség érzékelő). A szimuláció ilyen szintenkénti közelség jelet állít elő a liftek fel/le motorjainak jeleiből, különböző időzítők segítségével. Két féle időzítőről beszélhetünk liftenként, melyeknek sebessége egyesével állítható. (tehát akár eltérő is lehet a két lift esetén) Az egyik azt hivatott szimbolizálni, hogy a lift áthaladása egy szinten véges sebességgel történik, tehát a közelségérzékelő egy bizonyos ideig folyamatosan jelezni fog, amig a lift áthalad. Ennek az értékét én 0.5 másodpercre állítottam mindkét lift esetén. A másik időzítő az a lift két emelet közti távolság megtételét jelzi, ennek értékét 5 másodpercre választottam mindkét lift esetén. Fontos lehet megjegyezni, hogy a tényleges idő mire a lift feljebb kerül egy emelettel ezen két időzítés **összege**.

A szimuláció továbbá az ajtók mozgását is kezeli, szintén időzítők segítségével, ezeket egységesen 1 másodpercre állítottam nyitásnál is és csukásnál is, mindkét lift esetére, de természetesen külön-külön állítható mind a 4 idő.

Az egész szimuláció egyetlen függvénybe van zárva, ami azt jelenti, hogy a szimuláció kikapcsolható egyetlen függvényhívás megszüntetésével.

A szimuláció a generált bemeneteket a PLC memóriájában tárolja el, de a megfelelő Tag Table-ben a memóriacímet átírva tényleges ki és bemeneti címekre, a szimuláció könnyedén lecserélhető valóságos ki és bemeneti jelekre.

A szintjelzők szimulációjáért liftenként egy-egy állapotgép felelős, melyet a lifteket fel és lefelé mozgató motorok és a korábban említett időzítők működtetnek.
Ennek az állapotgráfja alább látható. Az elindulási állapotban még egy rövid ideig (0.5sec) jelez az adott szint érzékelője, majd a haladási állapotban egyik közelségérzékelő sem jelez, amíg el nem éri a lift a következő szintet.



A szimuláció állapotgépe egy liftre

A szimuláció az szint értékét alapvetően egész számként hozza létre és tárolja, de egy segédfüggvény segítségével ez visszaalakítható egyszerű bool jellegű szintérzékelők jeleire

2.5 Felhasználói felület

A felhasználói felületnek jelenleg 3 féle nézete van. Egy külső nézet, melynél egy legördülő menü segítségével választhatjuk, hogy melyik szint szemszögéből szeretnénk látni, minden emeleten elhelyezve fel és lefelé hívó gomb. Két belső nézet is van a két liftnek, ahol azokat lehet küldeni bármelyik szintre, illetve az ajtót működtetni a nyitó és záró gombokkal.

A legördülő menü működéséhez szükséges volt egy HMI Tag létrehozása, mely tárolja, hogy éppen melyik szint szemszöge az aktív. Ez abból a szempontból is egy előnyös megoldás, hogy ha a felhasználó másik nézetre vált, majd vissza a külső nézethez, az ugyan azon szintet fogja mutatni, mint a váltás előtt.
A legördülő menü elemeit a „Text and graphic lists” menüponton belül lehet megadni, illetve azokhoz egy-egy egész számú értéket rendelni.

Minden nézet esetén lehet látni az ajtó állapotát, külső nézet esetén mindkettőt, az adott szinten. Az ajtó képes nyitott és zárt állapotot jelezni. Az ajtó felett el van helyezve egy szintjelző, mely az adott lift haladási irányát is jelzi egy egyszerű kis fel vagy lefelé mutató nyíllal. A nézetek közt az alul elhelyezett 3db „LIFT1”, „LIFT2”, „KÜLSŐ” gombokkal lehet váltani. Ezen gombok működéséhez minden nézethez rendelni kellett egy nevet, és a gomb elengedésének hatására elsül egy esemény, ami aktiválja a megnyomott gombnak megfelelő nézetet, annak a neve alapján.

Az első lift belső nézete

külső nézet

A felhasználói felület ténylegesen 3 nézetből áll, azokon a nézeteken több kép is van elhelyezve, amiknek különböző láthatósági beállításai vannak, így „animálva” például az ajtó nyitást és csukást. Ezekhez néhány számítást végeznie kell a PLC-nek, például az ajtó nyitottságát el kell döntenie, a megfigyelt szint és a lift helyzete alapján. Ezen számításokat egy külön függvénybe gyűjtöttem ki, melyet a fő ciklus első lépésben hajt végre mindig.

Habár a vezérlésben egy lézeres érzékelővel megvalósítottam a lehetőségét, hogy az ajtót valaki megfogja, de végül úgy döntöttem a felhasználói felület kialakításakor, hogy erre nem adok lehetőséget. Ennek fő oka, hogy már így is eléggé zsúfolt volt a kijelző és az ajtók nem rendelkeznek folyamatos nyitási animációval, csupán egy nyitott és egy csukott állapotban lehetnek megjelenítve, ezért akárhogyan is raktam volna rá egy gombot, ami az ajtó megfogását illetett demonstrálni, nem segítette volna a felhasználói élményt.

3. Eredmények, következtetések

A vezérlés működését a tanszéken kipróbáltam a QB125 laborban egy valódi PLC-n és operátor panelen. Az algoritmus hatékonyságával meg vagyok elégedve.
A központi vezérlő program kialakítása miatt sajnos nehezen alakítható át más akna és szint esetére, de a lift fülke elvileg akárhány emelet és akna esetében használható, a megfelelő vezérlés implementálása mellett. Megjegyezném, hogy nagyobb akna/szint szám esetén mindenféleképpen szükséges lenne egy másik vezérlő algoritmus is, mivel ez a feladatban leírt környezetre van optimalizálva, egyáltalán nem garantálja, hogy ha bővítve lenne, akkor a hatékonysága továbbra is elfogadható lenne.

A felhasználói felületet és az animációkat lehetne bővíteni, javítani, de funkcionalitás tekintetében teljesen működőképes.

A munka során megismerkedtem a Siemens TIA Portal fejlesztőkörnyezetével és az ahhoz tartozó rengeteg felhasználói segédlettel.

Betekintést nyerhettem egy PLC program fejlesztésének folyamatába is, a tervezéstől kezdve, programírás, debuggolás, teszt szimulált környezetben és végül élesben is kipróbálni valódi eszközökön a kész megoldást.