ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

SZAKDOLGOZAT

OE-KVK
2023.

Szabó Dávid
T008843/FI12904/K

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Budapest, 20.23. 12. 12.

Szabó Dávid

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Szabó Dávid
Szakdolgozat száma: SZD2208301116526491[redacted]
Törzskönyvi száma: T008843/FI12904/K **Neptun kódja:**[redacted]

Szak: villamosmérnöki
Specializáció: Műszer-automatika specializáció

A dolgozat címe: Oktatási célra használható, gyártásautomatizálás folyamatainak bemutatására alkalmas rendszer fejlesztése
A dolgozat címe angolul: Development of a system for the presentation of production automation processes that can be used for educational purposes
A feladat részletezése: A feladat egy automatika/mechatronika szakirányú képzésben használt gyártósor alkalmassá tétele PLC programozás oktatására. Annak bemutatása, hogy a PLC programozás milyen kulcs elemeit lehet elsajátítani a gyártósor segítségével, és ezen elemek hogyan kapcsolódnak a tényleges gyártási folyamathoz.

Intézményi konzulens neve: Varga Árpád

A kiadott téma elévülési határideje:2025. 08. 30.
Beadási határidő: 2023. 12. 15.

A szakdolgozat: Nem titkos.

Kiadva: Budapest, 2023. 10. 24.

P.H

Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Elektronikai és Kommunikációs Rendszerek Intézet

.....
Intézetigazgató

A dolgozatot beadásra alkalmasnak találom:

Varga Árpád
.....
belső konzulens

.....
külső konzulens

Tartalomjegyzék

Bevezetés

A szakdolgozatomban a gyártásautomatizálás olyan folyamatait és háttér ismereteit szeretném bemutatni egy oktatási célra használható moduláris gyártósoron, amelyek nem csak az oktatásban, de egy valódi gyártókörnyezetben is megtapasztalhatóak egy mérnök, vagy egy technikus számára. A témát két fő szempont alapján szeretném végig vezetni:

- Melyek azok a területek, amik az oktatás során nem merülnek fel, vagy nincsenek eléggé hangsúlyosan kiemelve?

- Mik azok a kulcsfontosságú elemek, amit egy PLC program megírása során a programozónak lényeges szem előtt tartania, ha olyan programot szeretne létrehozni, amit a későbbiekben effektíven lehet módosítani, vagy akár a gyártómodulokon felmerülő hibák keresésénél alkalmazni?

A PLC programot a Siemens által forgalmazott TIA Portal (Totally Integrated Automation Portal) nevű szoftverben írtam meg. Ennek fő oka az volt, hogy a számításba jövő többi PLC-hez és szoftverhez képest ebben a fejlesztőkörnyezetben van a legtöbb tapasztalatom, modularitás szempontjából is megfelelő, egyszerűen konfigurálható, és ebben a fejlesztőkörnyezetben közvetlenül tudom kezelni a HMI és a PLC programjait, és könnyen tudok végezni akár Online módosításokat a PLC megállítása nélkül is.

Az eszköz, amit a PLC-vel vezérlek a Festo Didactic által oktatási célokra forgalmazott MPS (Modular Production System) állomássorozat egy régebben kiadott verziója, amit a 2010-es évek elején pályázhattak meg a szakközépiskolák és az egyetemek. Választásom fő szempontja itt is az állomássorozat előzetes ismerete volt, mivel a Festo Didactic képzésein ilyen modulokat szereltem össze és programoztam fel, szóval az állomásokat ismerem részletekbe menően, és tudom, hogy mik azok a pontjai, ahol érdemes volt módosítani az alap koncepciót.

A tapasztalatok, amiket az elmúlt néhány évben szereztem, sok olyan dologra világítottak rá, amiket lehetne módosítani a számítógépes folyamatautomatizálás oktatását érintően, illetve olyan kulcsfontosságú elemekre, amiket a külföldi oktatásokon (Siemens, Festo, Beckhoff) mindig kiemelnek, és az Ipar 4.0 programozási sztenderdjeinek alapjaiként tartanak számon, ám a hazai oktatásban még nem fordítanak kellő figyelmet ezkere az elemekre. Bár tapasztaltam kivételt az egyetemen végzett tanulmányaim során, ám itt is akad néhány olyan észrevétel, amit esetleg érdemes lehet fontolóra venni, mivel az egyetemen ilyen irányú végzettséget megszerző hallgatók szakmai pályakezdését jelentősen könnyítené, ha olyan új elemek is bekerülnének az oktatásba, amelyek jelenleg nem részei egy-egy ilyen képzésnek, vagy nincsenek annyira kiemelve, mint amekkora hangsúlyt kapnak egy gyártókörnyezetben.

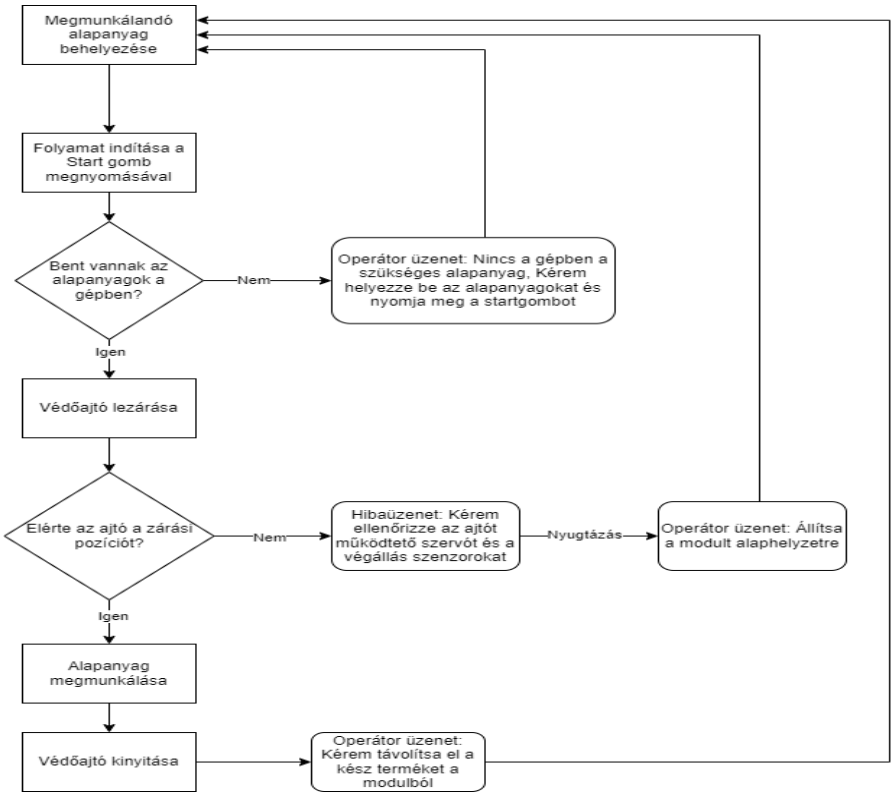
Forráskutatás, piaci alternatívák felmérése

A számítógépes folyamatautomatizálás alapfogalmai

Folyamatirányítás típusai

Vezérlés

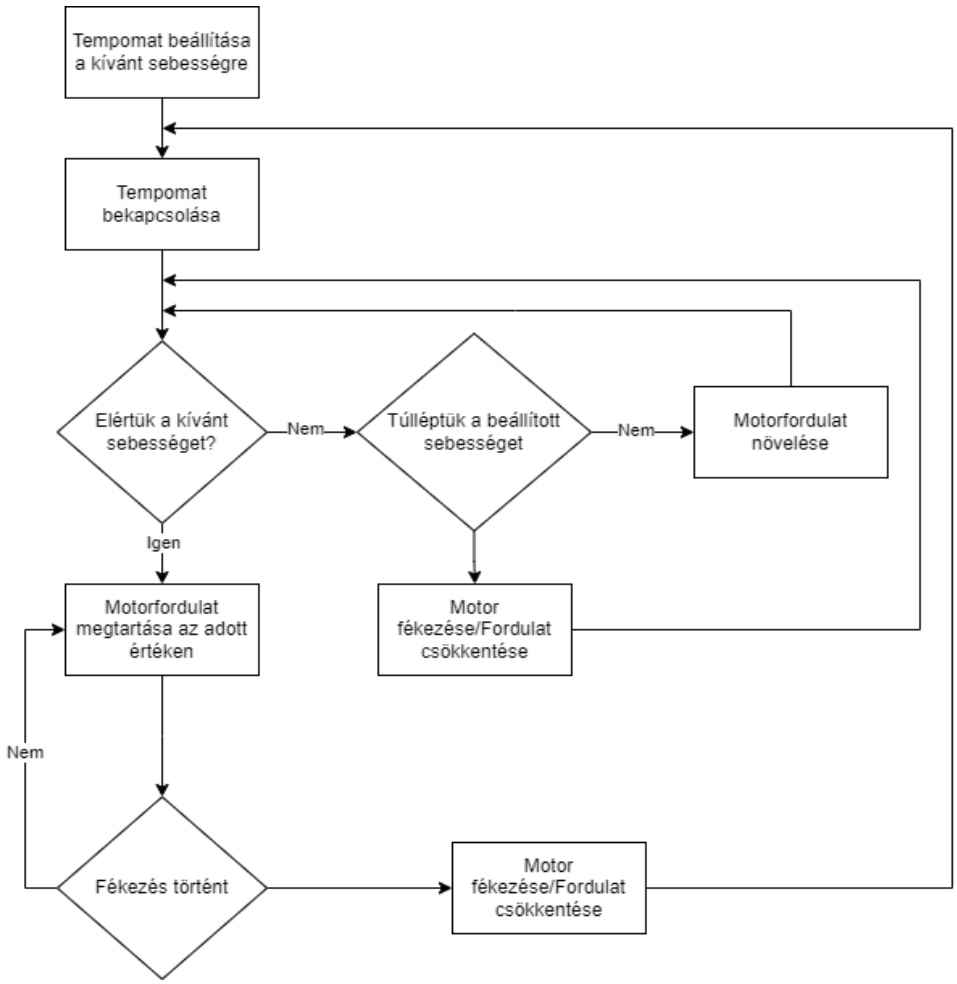
A folyamatirányítási metódusokat a szakirodalom két csoportra szokta tagolni: szabályzásra és vezérlésre. A vezérlés egy olyan folyamat, ahol a program a kiadott utasítások alapján lefut, létrejön egy eredmény és a folyamat a következő lefutás során újra végrehajtásra kerül, amiből ismét létrejön egy eredmény. Ennél a metódusnál a folyamatunkra semmilyen kihatással nincs az, hogy a folyamat végén mi is jön létre, a folyamat nem visszacsatolt. Például vegyük egy présgép működési folyamatát [1. ábra]: Ha az operátor beteszi a megmunkálandó darabot és elindítja a folyamatot, akkor a gép működésbe lép, a beállított végállásig a préselést végző munkahenger lemegy, és miután elérte a végállást visszamegy a kiindulási helyére, hogy ezután az operátor kivegye a kész munkadarabot, és behelyezzen egy újat. A gép nem veszi figyelembe azt, hogy milyen erő hat vissza a préselés közben, csak a kapott utasítási láncot hajtja végre.



1. ábra Példa vezérlésre

Szabályzás

A szabályzás olyan folyamat, ahol a kimeneten megjelenő végeredmény hatással van a folyamatunk lefutására, ezért a folyamatunk visszacsatolt. Ez azt jelenti, hogy az első lefutás után az összes többi lefutása a programnak, ami egymás után következik már valamilyen formában módosulhat, attól függően, hogy a kimeneten milyen tartományon belül szeretnénk várni az eredményeket. A folyamat, ha megfelelő a paraméterezés, akkor egyfajta önbeállást végez, és folyamatosan egyre jobban közelíti a felhasználó által elvárt értéket, míg hibás paraméterezés esetén előfordulhat, hogy az önbeállás során olyan kilengéseket produkálhat a rendszer, amiket fizikailag nem képes a rendszer elviselni, és katasztrofális következményekkel is járhat (például a Csernobili katasztrófa). Példa egy szabályzási folyamatra [2. ábra]: az autókban felhasznált tempomat rendszerek. Ezek a rendszerek úgy működnek, hogy a vezető beállítja a kívánt sebesség értéket, és a beállítást követően, ha a tempomat be van kapcsolva, akkor a sofőrnek nem kell a lábát a gyorsító-/gázpedálon tartani, mivel az autó saját maga



2. ábra Példa szabályzásra

számára központilag szabályozza a motor leadott teljesítményét. A rendszer a megfelelő beállítások mellett stabilan tartja a beállított sebesség értéket, lejtő esetén például kevesebb teljesítményt vesz fel a rendszer a gravitáció miatt bekövetkező gyorsulást kihasználva, míg emelkedő esetén a teljesítményt növeli, hogy tartani tudja a beállított sebességet.

Folyamatirányítási rendszer típusai

Hidraulikus/Pneumatikus rendszerek

A tisztán hidraulikus vagy tisztán pneumatikus irányítási rendszerek lényege a következő: olyan elemekből épülnek fel, hogy a működésük csak egy fajta energiára épüljön. Például egy tisztán pneumatikus felépítésű rendszer csak olyan elemeket tartalmaz, amiket a levegő vezérel, illetve különböző mechanikai hatásokra úgy reagálnak, hogy a levegőforrástól beáramló levegő útját változtatják a rendszeren belül. Főként útváltó szelepekből, valamilyen mechanikai elven alapuló végálláskapcsolókból, munkahengerekből, és hasonló elemekből épülnek fel. A különbség a felhasználási területet érintően jelentkezik, mivel pneumatikus rendszerek olyan környezetbe kerülnek felhasználásra, ahol lényeges szempont a tisztaság, és nem szükséges az a kifejthető erő, amire egy hidraulikus rendszer képes az olaj összenyomhatatlansága miatt.

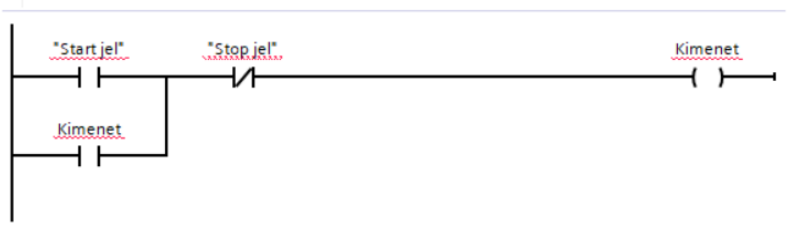
A fizikai kivitelezést megelőzően a tervezéshez itt például a Festo FluidSim nevű programját lehet alkalmazni. A program lehetőséget biztosít, hogy a szabványos rajzjelek alkalmazásával létrehozzuk azt a pneumatikus/hidraulikus kapcsolást, amit majd fizikailag is szeretnénk megvalósítani. A programban lehetőség is van a tesztelésre, így láthatjuk az ábrán azt, hogy a kapcsolás az adott kivitelezés alapján, hogy is működne a valóságban. A program kijelzi a hengerek mozgását, és lehetőség van például a fojtószelepek szimulálására is, így a program működése során képesek vagyunk közel olyan pontos szimulációkat végrehajtani, ami a valóságos körülményeket tükrözi.

Ezek a tisztán pneumatikus/hidraulikus rendszerek már manapság nem sok helyen fordulnak elő egy gyártó környezetben, mivel az adatokat és a működési paramétereket digitálisan kezeljük és továbbítjuk a hálózatokon belül. Viszont egyszerűbb célgépek esetén a mai napig alkalmazzák, vagy akár az olyan munkagépek egységeinek működtetésénél, amiknél szükséges a nagy erő kifejtése (például földmunkagépek).

Ezeket a köröket az elektro-pneumatikus vagy elektro-hidraulikus körök váltották fel, amelyek már olyan elemeket tartalmaztak, amik villamos jeleket is képesek voltak továbbítani, vagy villamos energiával is lehetett vezérelni a kör elemeit.

Relés vezérlésű körök

A relés vezérléssel megvalósított irányítási folyamatok a PLC-vel vezérelt körök elődeinek tekinthetőek. Ezek mögött a körök mögött olyan logika van, ami a PLC programozásban a létra-diagrammnak felel meg. Ezek a vezérlések eléggé hely- és energiaigényesek, emellett a kialakítási és fenntartási költségük sem elhanyagolható. Mivel a folyamatokat öntartásokkal építjük fel, így a memóriák feladatát itt egy-egy relé vagy mágneskapcsoló helyettesíti [3. ábra].



3. ábra Példa öntartásra

Ezek az áramkörök még nem voltak képesek adatok tárolására, és a jelenlegi felhasználásuk csak olyan gépeknél fordul elő, amelyek működése egyszerűen megoldható, nem szükséges, hogy a gép adatokat küldjön vagy fogadjon (különálló modul), és a kivitelezésük olcsóbbra jön ki, mint egy PLC-s megoldás esetében. Erre példa lehet egy présgép, vagy például olyan gép, amiket a nyomdaiparban a papírtekercsek egyenlő méretre történő perforálásánál alkalmaznak.

Mivel a jóval bonyolultabb alkalmazások költséges és helyigényes megoldásokat jelentettek, így a PLC-k és a mikrokontrollerek érkezésével ezek a vezérlési típusok egyre kevéssé használtak az iparban.

PLC vezérlésű körök

A PLC-k és a mikrovezérlők hozták el az automatizálás fellendülését az iparban. Ezek az eszközök a megfelelő programokat végrehajtva rengeteg olyan feladatot képesek ellátni, amit csak egy helyigényes, bonyolult és nehezen karbantartható relés körrel lehetne csak megvalósítani. Emellett a PLC-vel vezérelt eszközök ciklusideje jelentősen jobb lehet, mert a

relés felépítésű köröknél van egy fizikai korlát, ami miatt a valósidejűség elve nem teljesülhet. Mivel a jelenlegi törekvések az ipar irányából olyan jellegűek, hogy az adatok begyűjtése, feldolgozása, kiértékelése és tárolása magas prioritású feladat lett (BigData), így ezekre a feladatokra a PLC-k alkalmazása a jelenkorra már magától értetődővé vált. Ezek az eszközök amellett, hogy rendkívül széles palettán bővíthetőek, rengeteg olyan külső eszközzel is képesek kommunikálni, amelyek már a szenzorika területén is nagy újításnak számítanak. Olyan szenzorokat is tudunk alkalmazni, amelyek analóg jeleket, vagy akár környezeti hatásokat (mint például a páratartalom vagy a fényerősség) is rendkívül pontosan tudnak mérni, így ezeket az adatokat a PLC megfelelően tudja kezelni a folyamatirányítás és az adatok kezelése során. Ezek mellett egy jól strukturált, megfelelően optimalizált PLC program könnyebben használható a hibakeresésre, a folyamatok ciklus idejének optimalizálására, és mindemellett a biztonsági (Safety) körök is jóval pontosabban és érzékenyebben beállíthatóak. Egy PLC-s vezérlés/szabályzás a megfelelően beállított ciklusidők mellett sokkal költséghatékonyabb megoldás, mint a korábbi megoldások bármelyike, ha egy komplex rendszerről beszélünk.

PLC programozási nyelvek

A PLC programozás során a szabvány több dolgot is rögzít, és ezek közé tartoznak azok az egyedi programozási nyelvek, amelyeket a PLC-k programozásánál használunk. Fontos kiemelni, hogy ezek közül a nyelvek közül néhányat csak a PLC-s környezetekben láthatunk, mivel a programozási metódus olyan módon került kialakításra, amelyet más környezetekben nem lenne értelme használni. Ezek a programozási nyelvek a következőek:

- Strukturált szöveges programozási nyelv (SCL) [4. ábra] [5]
- Funkció Blokk Diagram (FBD) [5. ábra] [5]
- Létra Diagram (LD) [6. ábra] [5]
- Utasításlista [7. ábra] [5]
- Lefutó nyelv, állapotgráf, léptetőlánc (GML) [8. ábra] [5]

Strukturált szöveges programozási nyelv

A strukturált programozási nyelv lényege, hogy a programot egy olyan szöveges kialakítás segítségével hozzuk létre, ami hasonló akár a mikrokontrollerek esetében használt programozási nyelvekhez. A programozási nyelv a PASCAL nyelvhez hasonlatosan egy magas szintű, blokk

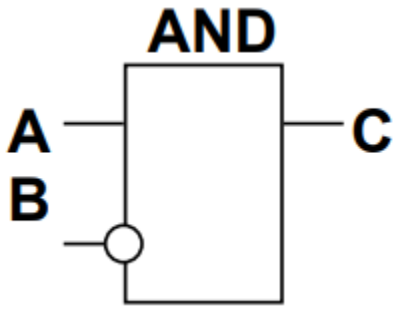
szervezésű programozási nyelv. A programozás során IF, ELSE, OR, AND, CASE, NOT, WHILE és egyéb utasítások segítségével alkotjuk meg a programot [4. ábra]. A PLC-s fejlesztő környezeteknél a változók deklarálása a TIA esetében például egy külön ablakban történik, viszont hasonló metódussal, mint a többi ehhez hasonló szöveges nyelvnél. Deklarálnunk kell a változó nevét, típusát, azt, hogy melyik változócsoportba kerüljön (kimenet, bemenet, belső változó, temporary változó). Kezdeti értéket itt is lehet deklarálni, ami a későbbiekben egy hasznos funkció lehet akár a CASE, akár a BOOL típusú változók esetében, ha nem az alapértelmezett 0 értéket szeretnénk a program indítása/újraindítása esetén.

```
C:= A  AND  NOT B
```

4. ábra Strukturált szöveges nyelv példa [5]

Funkció Blokk Diagram

A Funkció Blokk Diagram lényege, hogy az alap utasításkészletet egy olyan környezetbe ültették át, ami megjelenése és rajzjelei alapján a digitális áramkörök logikai kapus szerkezeti elemeire emlékeztet [5.ábra]. Ezeket az alap kapukat bővítették ki egyéb funkcióblokkokkal, amik például lehetnek időzítők, számlálók, matematikai műveletek, és még rengeteg más funkció (fejlesztőkörnyezetenként más ezeknek a könyvtáraknak az összetétele). A kialakítása miatt egy elektronikai ismeretekkel rendelkező személy számára könnyebben áttekinthető a program, viszont az áttekinthetősége és a programozás folyamata valamivel nehézkesebb, mint az SCL nyelv esetén, ennek következtében a program megírásának ideje is hosszabb.

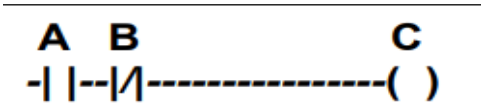


5. ábra Funkció Blokk Diagram példa [5]

Létra Diagram

A Létra Diagram egy olyan grafikus programozási nyelv, ahol a programozásnál felhasznált alap utasításokat ötvözték a relés kapcsolások ábráival és kialakítási logikájával [6. ábra]. A programsorokat úgy hozzuk létre, hogy a bal oldalon lévő „sínt” különböző elemek segítségével egy adott logika mentén

összekötjük a jobb oldalra elhelyezett kimenetekkel. Lényeges, hogy a kimenetek mindig a jobb oldalon, a kapcsolás végén helyezkednek el, és lényeges azt is szem előtt tartani, hogy az adott sorokban a legtöbb programban csak egy „kapcsolás” helyezkedhet el. Az LD nyelv használata egyszerűbb programok esetében célszerű, amiket könnyű áttekinteni. Egy bonyolultabb, összetettebb program írása esetén sokkal nehézkesebb az áttekintés, a hibakeresés, és a program írási ideje is jelentősen hosszabb az SCL nyelvhez képest.



6. ábra Létra Diagram példa [5]

Utasításlista

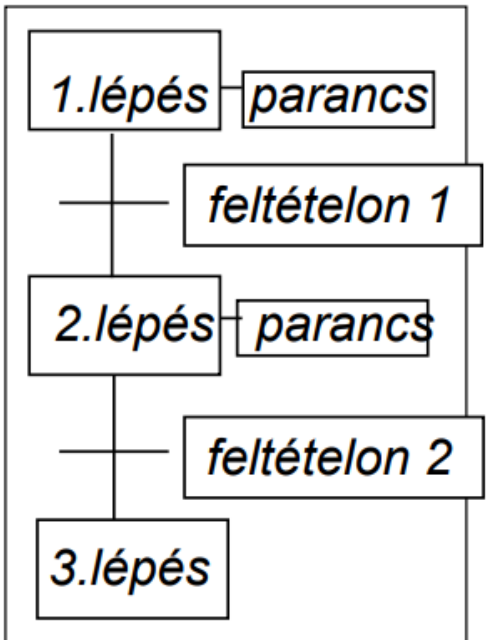
Az utasítás lista programozási nyelv az Assembly nyelvhez hasonlatos. A program kialakításánál lényeges, hogy egy sorba csak egy utasítás kerülhet [7.ábra]. Ezt a programozási nyelvet ritkán alkalmazzák, mivel még ha áttekinthető is, a program írása sokkal időigényesebb, mint az SCL használata esetében.

```
LD      A
ANDN    B
ST      C
```

7. ábra Utasításlista programozási nyelv példa [5]

Állapotgráf

Az állapotgráf nyelv lényege, hogy a program lefutását egy flowchart-hoz hasonlóan alakítjuk ki [8.ábra]. A kialakítása miatt a hibakeresése meglehetősen könnyű, mivel a programban könnyen kereshetőek az aktív lépések, és a folyamatábrás kialakítás miatt könnyen áttekinthető, hogy a lefutás miért akadt meg, vagy milyen feltételek szükségesek, hogy a program kikerüljön egy esetlegesen fellépő hurokból. A lefutás módosítása viszonylag egyszerű, és a későbbi funkciók implementálása is egyszerűen megoldható. Ezt a programozási nyelvet nagyjából azonos arányban használják az FBD programozási nyelvvel.



8. ábra Állapotgráf programozás példa [5]

A PLC és a fejlesztő környezet kiválasztása

A PLC (Programozható logikai vezérlő) a szakdolgozatom egyik fő eleme, így a kiválasztásnál több szempontot is figyelembe vettem. Az egyik szempont az volt, hogy melyek azok a PLC-k, amelyek az itthoni ipari környezetben megtalálhatóak jelentős számban. Az eszköz specifikációkat figyelembe véve három gyártó jött nálam számításba: A Siemens, ami az automatizálás kezdete óta különböző megoldásokkal van jelen a nemzetközi piacon, az Omron, amiről szintén hasonló mondható el, viszont nem annyira elterjedt, mint a Siemens eszközei, és végül a Beckhoff, ami bár az előző kettőhöz képest nem olyan régen van a piacon, de számok terén lassan átveszi a vezető szerepet az automatizálási területen világszerte.

Beckhoff PLC és a TwinCat

A Beckhoff PLC-inek az a sajátossága, hogy egy teljesen egyedi koncepción alapul: van egy Windows operációs rendszerrel ellátott kisméretű PC-nk, ami olyan elemekből épül fel, mint bármelyik átlagos számítógép. A kialakítása közel olyan, mint bármely másik PLC-nek, hogy a szabványoknak megfelelően be lehessen építeni a vezérlőszekrénybe. A felépítése moduláris, az eszközt bővítőkártyákkal lehet az igények szerint módosítani. És a működésében annyiban tér el a többi vezérlőtől, hogy az eszközön kernel szinten egyszerre két folyamat közt van megosztva az idő és a processzor használati jogosultsága: egy adott arányban kap időt a Windows rendszer, amiről tudjuk módosítani a PLC programot a TwinCat nevű szoftvert használva, és a többi

időben a processzort a TwinCat kapja meg, ami kernel szinten fut az eszközön, és a PLC innen kapja meg a programot, ami szerint működnie kell az irányítási folyamatnak. Ez az idő úgy van megosztva, hogy a Windows csak annyi időt kap, ami az ember számára tapasztalható akadásmentes, folyamatos működéshez szükséges, a többi idő pedig a PLC számára áll rendelkezésre. Ezt az arányt lehet módosítani, de az alapbeállítások mellett is képes a Windows olyan módon folyamatokat ellátni, hogy azt a felhasználó megfelelő sebességűnek tapasztalja. Erre főképp azért van lehetősége a Beckhoff eszközeinek, mivel egy teljesen egyedi, saját fejlesztésű kommunikációs protokollt használnak, ami az EtherCat nevet kapta. A jelenleg elérhető ipari kommunikációs metódusok közül leginkább az Ethernet kommunikáció hasonlít erre a protokollra, viszont az adattovábbítás sebessége és a sávszélesség aránya az EtherCat tekintetében sokkal jobb, mint jelenleg bármely más opció, ami a fejlesztők rendelkezésére áll. A Siemens-hez képest viszont az a tulajdonsága nem előnyös, hogy a HMI (Human-Machnie Interface) programozása egy másik szoftverben, a Visual Studio-ban történik, így plusz munkát jelent, hogy a két programot az eszközön szinkronba hozzuk és lebonyolítsuk a megfelelő kommunikációt a programok között, mint például a változók módosítása, vagy a bemenetek kiolvasása.

Omron PLC és a CX-Programmer

Az Omron PLC-i a CX-Programmer nevű szoftverrel programozhatóak, bizonyos területeken előnyösebb lehet a felhasználásuk, mint a másik két gyártó által biztosított alternatíváké. A kezelőfelület jóval letisztultabb, mint a másik két szoftver esetén. Viszont a HMI programozása itt is bonyolultabb, mint az az alternatíva, amit a Siemens szoftvere kínál a programozó számára.

Siemens PLC és a TIA Portal

A harmadik alternatíva a Siemens által forgalmazott PLC-ket jelentette. Itt rendelkezésünkre áll két fejlesztő környezet is: Az egyik a Step7, ami egy régebbi, mára már elavultnak tekinthető fejlesztőkörnyezet, amihez szintén a VisualStudio-t használva lehet HMI programot írni, illetve a TIA Portal, ami egy olyan egységesített megoldás, ami a Siemens összes elérhető termékével kompatibilis, bármilyen Siemens PLC-re és HMI-re írható vele program, és az eszközök programjait egy szoftverben párhuzamosan is lehet kezelni. Emellett a TIA Portal-hoz van egy PLC SIM nevű Siemens kiegészítő szoftver is, ami lehetővé teszi a felhasználó számára, hogy a megírt programot egy virtuális PLC segítségével tesztelni tudja. A TIA Portal emellé tartalmaz egy olyan segédfunkciót is, hogy a szimuláció során a programmal le tudjuk szimulálni a HMI-t

is, ami azért lehet előnyös, mert úgy tudjuk tesztelni akár a PLC programunkat, hogy a hálózatra még nincs bekötve a HMI, így a HMI funkcióját a fejlesztő eszköze látja el. A megjelenítése pedig azonos a kiválasztott fizikai eszközzel, elérhetőek ugyan azok a beviteli források (gombok, érintőképernyő), mint amikkel a fizikai eszköz is rendelkezik, így a tesztelés során úgy tudjuk a folyamatot végig vinni, hogy a fizikai eszközt még nem kell bekössük a hálózatra. Másik előnye a TIA Portal-nak, hogy a program Online módban történő monitorozása közben folyamatosan látjuk az eszközzel kommunikáló, nem a Siemens által gyártott eszközöket is, értesítést kapunk a kommunikációs hibákról, folyamatosan tudjuk monitorozni a folyamatokat és az eszköz állapotát anélkül, hogy az eszköz saját szoftverét elindítanánk a fejlesztésre használt eszközön. Az egyik lényeges funkció, ami még ide sorolható, az a backup-ok kezelése. A TIA biztosít egy Multiuser Server nevű funkciót, ami használható a PLC-k távoli elérésére, lehetőséget ad arra is, hogy például egy programozási folyamatot a programozó Online programozzon és mellette több felhasználó is alacsony késleltetéssel tudja monitorozni a folyamatot. Ha a programban bármilyen módosítás történik, akkor az elmenthető erre a szerverre, és akár a verziók közötti eltérést is könnyen össze lehet hasonlítani a PLC-n jelenleg futó verzióval. Összességében egy kompakt, jól átlátható, könnyen paraméterezhető felületet kapunk, ahol a fejlesztés összes folyamatát el tudjuk végezni egy környezeten belül.

Választás, és annak oka

A választásom a Siemens PLC-ire és a TIA Portal-ra esett, mivel a fejlesztőkörnyezet felhasználóbarát, és sok olyan funkciót lehet vele könnyedén megtanulni, vagy mások számára tanítani, amit egy másik környezetben sokkal körülményesebb megoldani. A Siemens megkéri az árát ezeknek az eszközöknek, így más felhasználásban (ha több idő van a fejlesztésre) érdemes elgondolkodni a Beckhoff által biztosított opciókon, mivel olcsóbb és fizikai értelemben kompaktabb megoldásokat kínálnak. Számomra viszont a fejlesztőkörnyezet nyújtotta előnyök sokkal jobban illettek a szakdolgozat megvalósításához.

A PLC által vezérelt eszközök felmérése és kiválasztása

fischertechnik Training Factory Industry 4.0 rendszer

A fischertechnik által forgalmazott megoldás rengeteg előnnyel bír. Gyakorlatilag olyan komplex rendszert tudunk lemodellezni, ami a jelenlegi Ipar 4.0 témakörrel kapcsolatos területek számos részét lefedi, ez a rendszer kompakt, moduláris, az alkatrészei könnyen és olcsón pótolhatóak, mivel az eszközök jelentős része mondhatni LEGO jellegű, műanyag alkatrészekből

áll [9.ábra], így akár 3D nyomtatással is pótolhatóak az alkatrészek egy meghibásodás vagy egy tesztelés során fellépő mechanikai hiba esetén. Ez az eszköz ugyan úgy vezérelhető egy teljes értékű PLC-ről, mivel rendelkezik a csatlakozáshoz szükséges interfészekkel. Működtetéséhez 24 V-os rendszer elegendő, így akár egy labortápegységről is működtethető. Ez az eszköz a jelenlegi technika és a piaci árak mellett jobb megoldásnak bizonyult volna, és oktatási célokra ajánlatosabb ezeknek az eszközöknek a beszerzése a költséghatékonyság és az átadható tudás tekintetében. Viszont a sérülékenysége miatt érdemes átgondolni a beruházást, mert egy ilyen modul jelenlegi piaci ára megközelítőleg 2.6M forint, ami egy iskolai költségvetésnek nem kis tételt jelent, főleg, hogy a hatékony oktatáshoz legalább 2-3 ilyen eszköz szükséges.

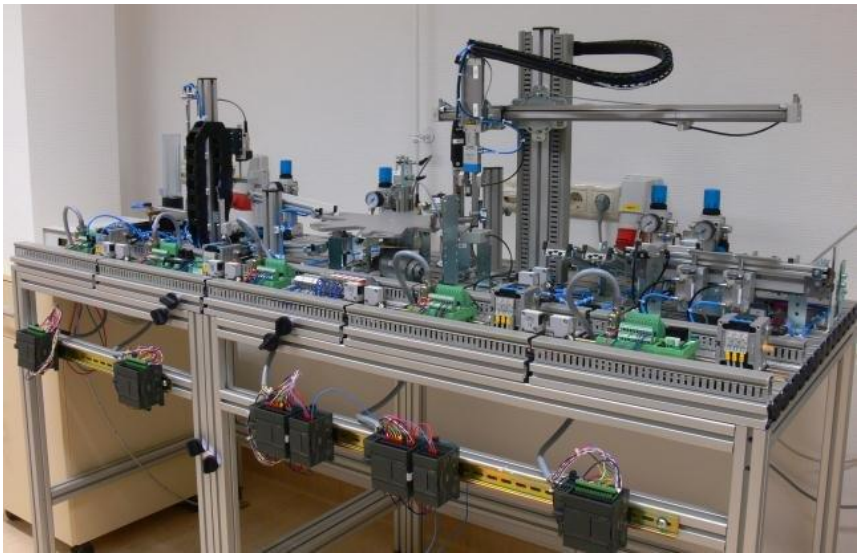
9. ábra fischertechnik Training Factory Indusrry 4.0 állomás [2]

A Festo Didactic MPS rendszere

A Festo Didactic a Festo cég azon területi egysége, ami főképpen az oktatás támogatását és az iparhoz tartozó (főképp mechatronikai irányú) képzéseket támogatja, és ezen képzésekhez létrehozott oktató eszközök gyártásával/tervezésével és forgalmazásával foglalkozik. A cég oldalán elérhető egy katalógus [1], ami összefoglalja azokat a termékeket és lehetőségeket, amiket az MPS modulokat használva ki tudnak használni az oktatók. A képzések lehetőségei alapján több szakirány felől is hasznosítható ez a rendszer, viszont ezeket a folyamatokat főképp a Mechatronika foglalja magába. Az MPS állomások előnye az, hogy olyan anyagokat, eszközöket és egyéb megoldásokat használ, amik az iparban azonos formában megjelennek, azzal a különbséggel, hogy itt a modulok egy egyszerű feladatot szimulálnak. A felépítésüket tekintve két opció van: vagy fix helyre vannak telepítve, és egy alumínium profilokból összerakott szerkezetre vannak feltéve a modul elemei és az azokat tartó, hornyokkal ellátott tábla [11.ábra], vagy a másik megoldás, hogy ezt egy olyan Festo által forgalmazott kocsira

helyezik el, ami kis helyet foglal, így a modul elemei mozgathatóak egymástól függetlenül, könnyen felcserélhetőek, variálhatóak [10.ábra].

10. ábra Festo MPS állomások, kocsira szerelt kivitelben [3]



11. ábra Festo MPS állomások, Bosch-profil állványra szerelt kivitelben [4]

Az oktatásban általában az első [11.ábra] opcióval szembesültem, mivel ezek az állomások a legtöbb esetben egy pályázat keretein belül kerültek ki az iskolákhoz, így ezek a modulok a

laborokban fix helyre kerültek. Számomra azért tűnik jobb opciónak ez a rendszer, mert az eszközök úgy működnek, úgy reagálnak, és úgy hibásodhatnak meg, mint egy tényleges gyártó környezetben: egy rendszer hibakeresését például nagyon hatékonyan lehet megtanítani egy ilyen modulon, mivel olyan szenzorokat, olyan vezetékezést és csatlakozásokat, olyan I/O szigeteket, és olyan villamos és pneumatikus eszközöket alkalmaznak, amik egy gyárhoz telepített gépsornál is megtalálhatóak.

Ennek köszönhetően a technikusok akár az oktatás során olyan valós hibákkal is találkozhatnak egy-egy szimulált esemény során, amikkel a későbbiekben a pályájuk kezdetén szembesülnének először a gyakorlatban.

Választás, és annak oka

A választás során két fő szempontom volt itt is: olyan eszközzel szerettem volna megvalósítani a szakdolgozat gyakorlati részét, amin keresztül rávilágíthatok a technikusi és a mérnöki képzés bizonyos hiányosságaira is, és olyan rendszerben gondolkodtam, ami egyszerű példákon keresztül akár a legtöbb területet lefedheti, ami ezekhez a munkakörökhöz szükséges a sikeres pályakezdés szempontjából. A második szempontom pedig az volt, hogy a rendszer olyan legyen, hogy a programozó, vagy a tanuló érezze a súlyát annak folyamatnak, ami a program utasításai szerint végbemegy. Más hatást kelt az, ha két műanyag elem ütközik, az illesztés mentén szétválik, utána pedig össze lehet rakni, és más hatást kelt az, amikor például a hibás program miatt a munkadarabot egy munkahenger úgy leszorítja, hogy azt kézi erővel nem lehet megmozdítani. Bár, ha lehetőség van az iskolák számára, hogy válasszanak, akkor a fisertechnik eszközeit ajánlom elsőkörben, mivel költséghatékonyabb a működtetés, és kisebb helyen elfér ez a fajta oktató eszköz. Viszont, ha van lehetőség, akkor ajánlom, hogy legalább egy MPS modult is tartsanak fenn az iskolai laborok a fent említett szempontok alapján, vagy ha költséghatékonyabb, akkor egy ezen az elven alapuló, saját készítésű modult állítsanak össze.

Feladat megfogalmazása

A feladatom a következő: egy olyan összetett PLC program létrehozása, amely több modul segítségével képes a számítógépes folyamatautomatizálás alapjait bemutatni, és a program tartalmazza azokat a fő elemeket, amiket egy PLC programozó számára lényeges elsajátítani mind programozási, mind a szemléleti hátteret figyelembe véve. A modulokat vezérlő program

mellett egy olyan rendszer kialakítása a cél, ami magába foglalja több PLC hálózaton keresztül történő használatát és tartalmazza a HMI által történő vezérlés és tesztelés lehetőségét is.

Feladat megvalósítása

A PLC program megírása

A PLC programnak olyannak kell lennie, hogy a későbbiek során bárki képes legyen a kódban eligazodni, képes legyen abban hibát keresni. Ehhez szükség van egy jól felépített, megfelelően felosztott szerkezetre, a program és a változók kommentezésére (lehetőség szerint több nyelven), a változók nevének megfelelő megválasztására (a változó neve egyértelműen utaljon a feladatára, „beszédes" neveket kell alkalmazni). Ha ezek a feltételek teljesülnek, akkor a megalkotott kód sokkal letisztultabb, áttekinthetőbb és olvashatóbb lesz. Ezek mellett javasolt a feladat bonyolultságának megfelelően megválasztani a programozási nyelvet: egy egyszerű folyamatot meg lehet írni akár LD programozási nyelven is és áttekinthető marad a program, viszont, ha a folyamat bonyolultabb, akkor érdemes az ST nyelvet választani. Ennek az az oka, hogy az ST-ben megírt kód könnyebben másolható, és olyanok számára is könnyen olvasható, akik nem rendelkeznek villamosiparral kapcsolatos ismeretekkel, viszont valamilyen programozási nyelvet ismernek. A kód emellett sokkal könnyebben kommentezhető, és sokkal egyszerűbben átlátható lesz.

A kód írásánál érdemes szem előtt tartani azt, hogy elsőre nem minden esetben lehet optimális kódot írni. Először mindig értelmezni kell a feladatot, és utána megoldani úgy, ahogy elsőre meg tudjuk. A kód ilyenkor nem lesz feltétlenül optimális, de azt a végeredményt kapjuk, amit szeretnénk. Ezt követően jöhet az optimalizálás. A programozás során a kódnál egyszerre csak egy-egy részletet módosítsunk, majd lehetőség szerint teszteljünk. Ilyenkor, ha esetleg olyan módosítást hajtottunk végre, ami a vártaktól eltérő eredményt hoz, akkor a módosítást még ilyenkor vissza lehet vonni, és nyomon követhető marad a folyamat. Ellenben, ha egyszerre több dolgot is módosítunk, akkor már jóval nehezebb rájönni, hogy melyik módosítás okozhatja a problémát, így ez nem javasolt.

Ha ezeket az alapelveket szem előtt tartjuk, akkor rövid időn belül tudunk olyan kódot írni, ami megfelelően optimalizált, hibakeresésnél bárki számára használható, és stabil működést tudunk elérni a vezérlés/szabályzás során.

A kód struktúráját érdemes úgy felépíteni, hogy az általunk vezérelni kívánt eszköz programját először egy funkcióblokkba írjuk meg. Ez azért lényeges, mert így meg tudjuk írni az irányításhoz szükséges funkciót úgy, hogy az egy könyvtárba lementhető, sokszorosítható verzióban elérhető lesz.

A gyártósor rövid bemutatása

A gyártósorunk öt önálló modulból épül fel. Ezek a Distribution (Adagoló), Testing (Tesztelő), Processing (Megmunkáló), Handling (Átrakó) és a Sorting (Válogató) modulok. Ezeknek a moduloknak a programban dedikált funkcióblokkjuk van, ezeket hívja meg a fő program. A Safety(biztonsági) programrész minden egyes funkcióblokkban a program része, mivel a különböző állomásoknak saját specifikus feltételei vannak a vészállapot kezelésére. Az állomások alap vezérlését ugyan azok a változók látják el.

A feladat összességében egy olyan gyártási folyamatot modellez le, ahol a gyártás során a megmunkálandó nyersanyagot válogatás nélkül küldjük be a gyártósorra. A munkafolyamat első része arra szolgál, hogy a többi modulra csak egyesével, de a többi folyamat igényeinek megfelelő időben kerüljenek be a megmunkálásra váró munkadarabok a rendszerbe. A második folyamatrész egy vizsgálati folyamat, ahol a termék paramétereit határozzuk meg: méret szín és anyag szerint. Itt a választásnál a nem megfelelő anyag és szín kombinációt tartalmazó munkadarabokat vizsgálat nélkül leválogatjuk, a megfelelő típusú termékek magasságát pedig megvizsgáljuk, és csak a megfelelő paraméterekkel rendelkező termékeket engedjük tovább. A következő rész a megmunkálási folyamatot szimulálja, ahol a nyers munkadarabból létrehozzuk a piacra szánt terméket, és bevizsgáljuk azt, hogy a megmunkálás folyamatai sikeresen zajlottak le, vagy sem. A következő folyamategység a terméket végállomásra juttatja, vagy ha nem volt megfelelő a megmunkálás akkor félre teszi. Az utolsó folyamat pedig egy leválogatási folyamat, ami azért kell, mert a gyártósor kezdetén az alapanyagot válogatás nélkül juttatjuk a termelési környezetbe.

A feladat olyan körülményeket szimulál, amelyek egy gyártási folyamatban is megjelennek: alapanyag kezelés, megmunkálási folyamatok és vizsgálatok, a munkadarabok szállítása különféle technológiákkal (átrakó karokkal, amik kiválthatóak robotokkal, szállítószalag, körasztal), és a termékek kezelése a folyamat során (hibás alapanyag kiszűrése, nem megfelelően elvégzett megmunkálás esetén a termékek vizsgálatra és javításra küldése, elkészült termékek

szelektálása és tárolása). Ezek a folyamatok egy Ipar 4.0-lás gyártási környezetben már másképp vannak kezelve, minden egyes termék egyedi kóddal rendelkezik, amely a termék összes megmunkálási folyamatáról, összetevőiről és paramétereiről tartalmazza az információkat.

A PLC konfiguráció kiválasztása

A PLC által vezérelt modulok számából adódik, hogy a feladathoz egyetlen PLC modul alkalmazása nem lenne elegendő, mivel az általam használt PLC nem rendelkezik elegendő mennyiségű be-és kimenettel, illetve nem áll rendelkezésre annyi bővítő modul, hogy a folyamatot egy PLC használatával el tudjam látni. Így a modulok két részre lesznek osztva, az első PLC-n fut a fő program, és ennek a PLC-nek a ki és bemeneteit az első modul fogja használni. A többi modult a második PLC fogja ellátni ki-és bemenetekkel, és ez a PLC a Master-Slave rendszeren belül egy Slave funkciót fog betölteni, az első PLC-hez a hálózat segítségével csak a kimeneteit és a bemeneteit biztosítja, így ennek a PLC-nek csak bővítő szerepe lesz. A folyamathoz egy HMI is fog tartozni, amiről a modulok önállóan monitorozhatóak, kiválaszthatóak az üzemmódok, és ezen a panelen lesz elérhető a manuális tesztelés lehetősége is.

A PLC program szerkezete

A PLC programom felépítése a következő: Az állomások programjaihoz létrehoztam egy-egy funkcióblokkot, amelyeket meghívok a fő programban. Ennek az oka az, hogy a program sokkal áttekinthetőbb, ha a modulokat külön-külön kezelem, illetve ezeket a létrehozott funkcióblokkokat egy könyvtárhoz adva bármikor újra felhasználhatom a későbbi projektek során, ha szükség lenne rájuk.

A vészállapotokat minden egyes funkcióblokkban a modulra specifikusan kezelem le. Ennek az oka, hogy a különféle eszközök vészállapotát más vezérlési logika szerint kell megírni. Például a második (Testing) modulnál lévő liftet vész állapot esetén féküzemben kell megállítani, hogy a termék és az eszköz véletlenül se sérüljön meg, és biztosítva legyen az a fix pozíció, amit a vészállapot előtt elért a lift. Az utolsó (Sorting) modulnál viszont más a helyzet. Ott a vészgomb megnyomását követően minden megáll: a futószalag vezérlése, a terelőkarok vezérlése (amelyek így visszaállnak az alaphelyzetre), a stopper munkahenger is elveszti a vezérlést és alaphelyzetre áll. Mivel a modul működtetésénél olyan körülményeket vettem figyelembe, amik egy ipari környezetben is lényegesek, így ezt a vészállapotok kialakításánál is figyelembe kellett vennem.

A HMI programjánál néhány fő szempontom volt: Az alapképernyőn az operátor vagy a technikus ki tudja választani a modulcsoportra vonatkozó működési üzemmódokat (automata/lépés üzemmód), rendelkezzen az alap kezelőfelülettel (vészgomb, start, nyugtázó gomb, alaphelyzetre állítás), illetve egy szimulált lámpaoszlop segítségével kijelzésre kerüljön, hogy a modul jelenleg milyen üzemmódban van, és mit vár a következő lépésben. A fő képernyő második fele egy menüként szolgál, ahonnan el lehet navigálni azokra a képernyőkre, ahol a modulokat egyesével lehet monitorozni, és a tesztüzem képernyőjén a modulok ki és bemeneteit egyesével tesztelni is lehet direkt (kézi) vezérléssel.

A program írása során ST (Strukturált Szöveges) programozási nyelvet használtam. A program lefutását egy állapotgéppel valósítottam meg. A Case szerkezet, egy olyan átfogó keretet ad a programnak, ami több szempontból is segítséget nyújthat. Az egyik ilyen szempont, hogy a léptetésre használt Step változó visszajelzést ad nekünk, hogy jelenleg a program melyik lépésnél jár. Ez a hibakeresésnél hasznos, mivel így gyorsan ki lehet deríteni, hogy esetleg a program lefutása mi miatt akadhatott meg, programhiba van, vagy esetleg valamelyik szenzortól nem kaptuk meg a megfelelő jelet. Ezen a Case-en belül a Step változó csak fix értékeket vehet fel, csak olyan értékeket, ami egy valós lépéshez tartozik. A program első részletében látható, hogy a léptetést a Step_1_modul nevű változó értékének változtatásával oldottam meg. A változó alap esetben 0 értékű, így a program minden újraindítás/feltöltés során a 0. lépésről indul. Ebben a lépésben a szimulált lámpaoszlop sárga lámpáját villogtatom, ezzel jelezve, hogy a modul nyugtázásra vár az operátor/technikus oldaláról. Az 1. lépésben a program üzemmódját tudjuk kiválasztani, hogy automata vagy lépés üzemmódban akarjuk a programot végrehajtani. Ha a lépés üzemmódot választjuk, akkor minden egyes program béli lépés során a start gomb megnyomásával kell kezdenünk a folyamatot. Ezt akkor szokták használni a gyártás során, ha valamilyen paraméterezés történt, és például a módosított pozíciókat akarják visszaellenőrizni. Az első lépésnél a modul kap egy ellenőrzést, hogy a nyugtázás megtörtént-e, ha nem akkor visszalép a program az előző lépéshez, ahol a nyugtázást kértük a felhasználótól. A program minden egyes lépését úgy írtam meg, hogy a legelső feltétel, amit nézünk, az a vészgomb megnyomásának állapota. Ha az be van nyomva, akkor automatikusan a 66. lépésbe kerül a program. A 0-1-66-69-es számú lépések az összes általam írt funkcióblokknál standardizált lépések. A 0. az alap kiindulási állapothoz tartozik, az 1. a folyamat lefutásának módkiválasztása, a 66. lépés a vészállapot kezelése, a 69. lépés pedig az alaphelyzetre futás és a 0. lépésbe

visszatérést kezeli le. A programrészletek között a lépéseket a Step változó 10-zel történő növelésével oltottam meg. Így a léptetés, és a folyamat közbeni funkcióváltás is egy másolható programrészletté vált, amit a programozás során minden esetben csak át kellett másoljak a következő lépbe. A program úgy lett megírva, hogy ha a folyamat közben bármikor átváltunk lépésüzemmódba, akkor a folyamat újrakezdéséig ebben a módban marad. A program csak akkor léphet át a következő lépésre, ha a szükséges folyamatok megtörténtek. Például, ha a fenti programrészletet nézzük, akkor a program csak akkor léphet át a következő lépésre, ha az adagoló ki van tolva. Ha ez nem teljesül, akkor a program leragad ennél a lépésnél. A 66. lépésben a program vészállapotát kezelem le. A program lépés szerkezete azért lett kialakítva 10-es lépésközökkel, mert így, ha szükséges lenne részfolyamatokkal bővíteni a programot, akkor minden lépés között rendelkezésemre áll további 9 lépés, amit egyesével fel lehet használni. Viszont, ha köztes lépéseket iktatunk be, mindig figyelni kell az előző lépésnél, hogy hogyan módosítjuk a Step változó értékét. Mivel az alapértelmezett lépésköz 10 egység, így, ha nem módosítjuk a szükséges helyen a lépésköz értékét, akkor a program nem is fog belépni az általunk beiktatott köztes lépésekre.

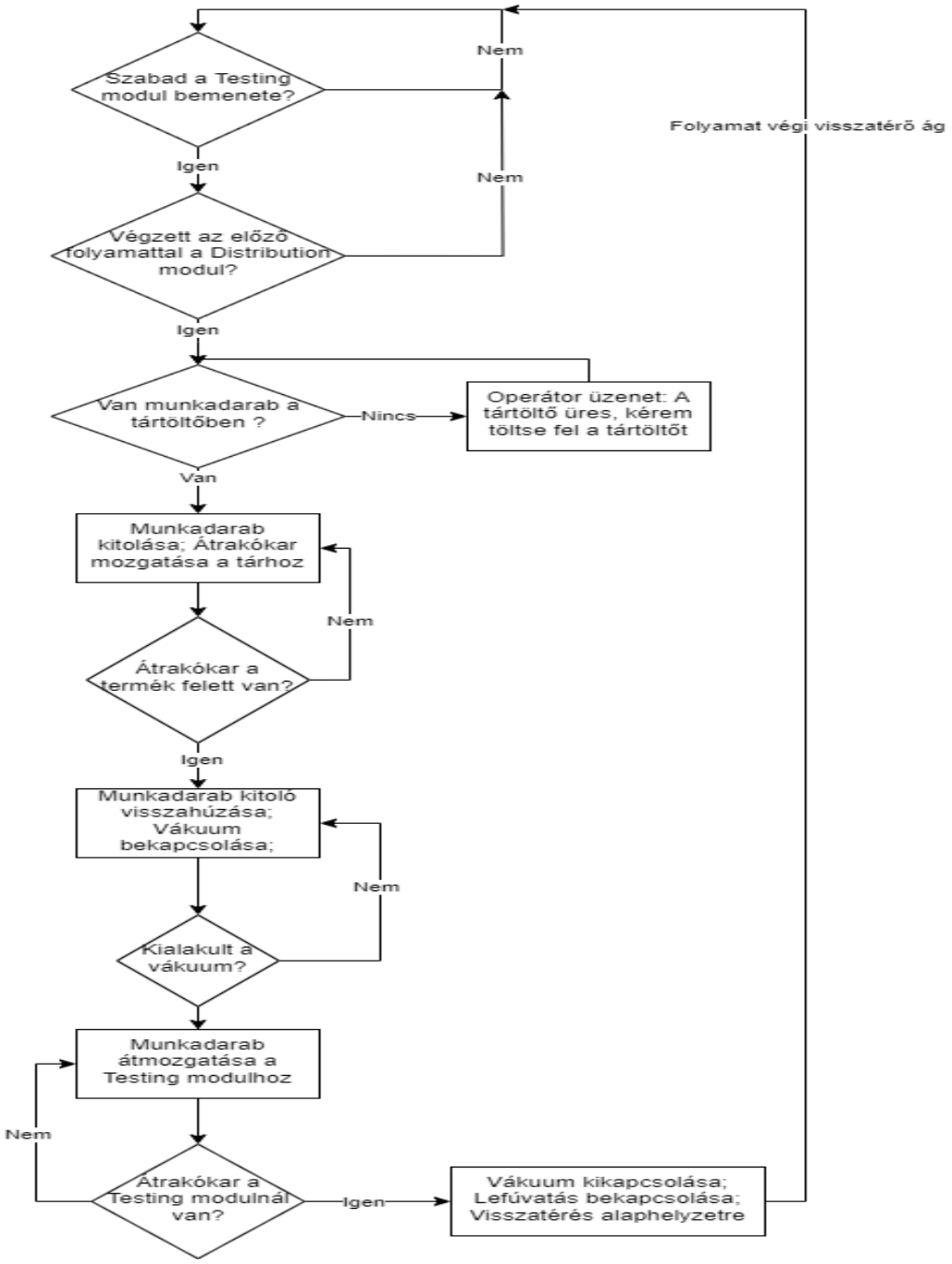
Első modul (Distribution) feladata

Az első modul feladata a következő: Ha a tártöltőben van munkadarab, akkor a beérkező start jel után az adagoló tolja ki a munkadarabot [14.ábra]. Egy 180°-os elfordulásra képes átrakó kar a munkadarab felé mozog, és az átrakó végére szerelt vákuumos tapadókorong segítségével megfogja a munkadarabot [13.ábra]. Ezt követően, ha a vákuum megfelelően kialakult, akkor az adagoló visszaáll alapállapotba. A munkadarabbal az átrakókar átmozog a lerakási pozícióba, és ott lerakja a terméket, majd ezt követően visszaáll alaphelyzetre [12.ábra].

Vészállapot kezelése

A vészkör kialakításának egyik fő szempontja a következő: A termék lehetőleg egyetlen pillanatig se maradjon szabadon, rögzítés nélkül. Ezért, ha a vészgombot aktiválják, akkor a következő lehetőségek adódnak: ha a termék az adagolónál van, akkor a terméket az adagoló munkahengere rögzítse. Mivel a kitolást végző munkahengert egy monostabil útváltószeleppel vezéreljük, ezért ez a szelep állandó vezérlésen marad, ha elérte a végállást. Ha nem érte el a végállást, akkor viszont a vezérlést elvesszük róla, hogy visszaálljon az alaphelyzetbe. Ez azért lényeges, mert ha az átrakókar valamilyen oknál fogva a felvételi pozíciónál van, mielőtt a termék ott lenne, akkor a terméket kitolva az átrakóra szerelt vákuumos megfogó rögzítése

sérülhet, így ezt az esetet ki kell zárni. A másik lehetőség akkor áll fent, amikor az átrakókar a termékkel mozog a lerakási pozíció felé: Ha a terméket felvette, akkor a vákuumot ebben az esetben is meg kell tartani, mivel a terméket nem ejthetjük le. A másik feladat pedig az, hogy az átrakó kart féküzembe helyezzük. Ez azt jelenti, hogy az átrakó kar munkahengere kétoldali vezérlést kap, így sem előre, sem hátra nem lehet kitéríteni. A nyugtázás hatására ezek a rögzítések feloldanak, és a termék manuális vezérléssel eltávolítható lesz a modulból, ha a terméket a vákuumos megfogó fogta, akkor a lefúvatáshoz tartozó szelepet kell vezérelni ehhez, mivel a termék megragadhat a vákuumos megfogón, ha az jó pozícióban lett felvéve. Ha nem lett eltávolítva a termék, akkor az alaphelyzetre futáskor a program ezt figyelembe veszi, és a terméket átrakja a lerakási pozícióra.

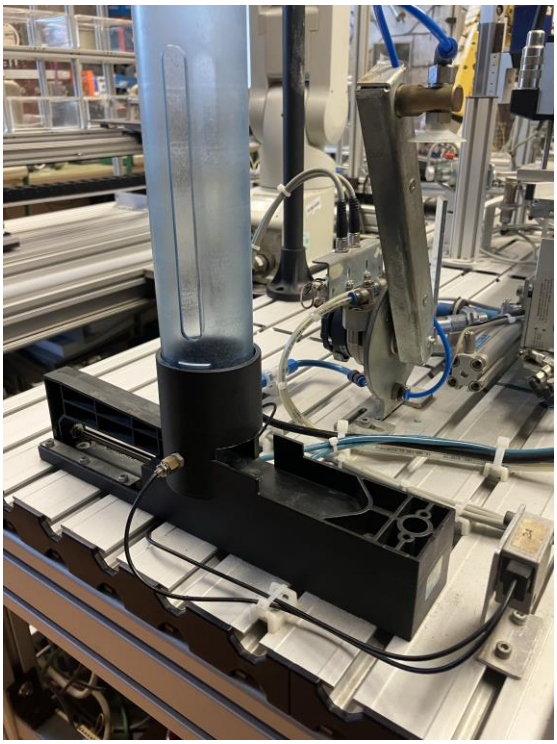


12. ábra Distribution modul - folyamatábra

Első modul (Distribution) oktatási funkciói

Az első modul oktatási és programozási szempontból ugyan rendkívül egyszerű feladatot ad a tanulónak, ám az alapismeretek közül jópárat el lehet vele sajátítani. Ezek közül például a legelső, és legalapvetőbb, a léptetőlánc kialakítása. A feladat során a folyamatok egymás után következnek, egy lefutó vezérlést kell írni a feladat megvalósításához. Ez a feladat megtanítja a hallgatónak, hogy hogyan kell megírni egy program léptetőláncát, megtanítja, hogy egy folyamat

ki- és belépési feltételeit hogyan kezelje le. Emellett a feladat további részeként a tanuló megtanulja, hogy a léptető üzemmód és az automata üzemmód hogyan implementálható a programba, mik azok a szabályok, amiket ilyenkor figyelembe kell venni. A feladat végén következik az a részlet, ami a legfontosabb a tervezés során: a vészállapot kezelése. Ebben a feladatrészben a tanuló megtanulja, hogy a különféle beavatkozókat a szituációktól függően hogyan kell kezelni, illetve megtanulja azt a szabályt, hogy a gyártás során a termék nem lehet szabadon, valamilyen formában mindig stabilan rögzíteni kell a terméket, mivel balesetveszélyt jelenthet, és a termék esetleges károsodása is veszteséget jelent a gyártás során. A folyamatot kiegészíti a nyugtázott állapot, és az alaphelyzetre futás, amit mindig nagy körültekintéssel kell megírni, mivel, ha a modul akármilyen helyzetben kerül vészállapotba, képesnek kell lennie visszaállni az alapállapotra károsodás nélkül.



14. ábra Distiribution modul - tártöltő és adagoló egysége

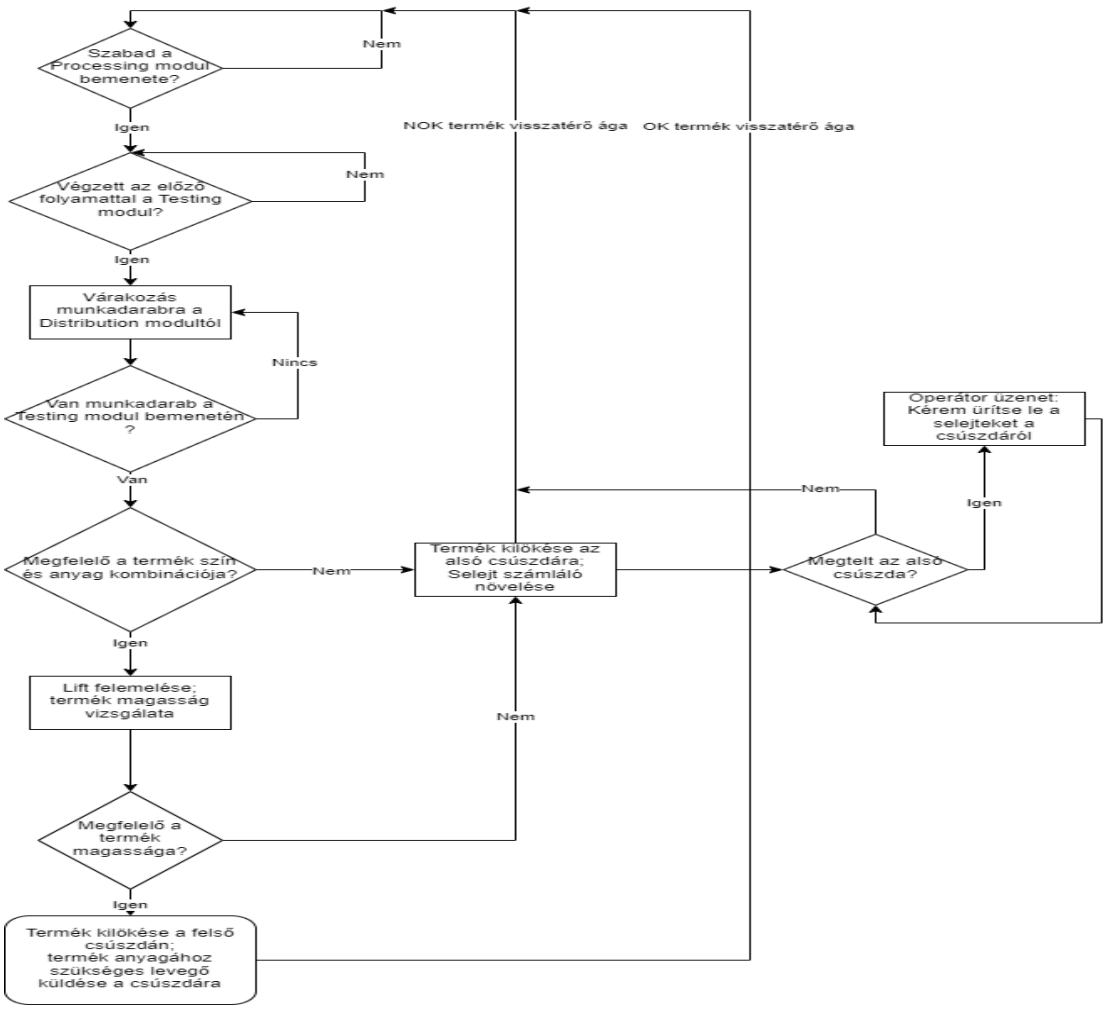
13. ábra Distribution modul - vákuumos megfogóval ellátott átrakókar

Második modul (Testing) funkcióblokkja

A Testing modul néhány lényeges funkcióban tér el az első modultól: ennél a modulnál a termék magasságának vizsgálatát végezzük el. A terméket egy lift segítségével emeljük fel a magasságmérést végző egységhez [17.ábra], a magasságmérő egység egy komparátorra kötve ad

nekünk egy jelet [19.ábra], ezzel tudjuk, hogy a termék megfelelő magasságú [18.ábra]. Ha a termék megfelelő magasságú, akkor a felső csúszdán lökjük ki, ha nem megfelelő, akkor az alsó csúszdára kerül. Mivel a termék lehet fém műanyag is, így a súlyeltérés miatt a csúszdán két különböző forrásból tudjuk levegővel segíteni a termék lecsúszását [16.ábra]: a fémhez magasabb nyomású, a műanyaghoz pedig egy alacsonyabb nyomású levegő kerül a csúszda felületére. Emellett a termék anyagának és színének teljes beazonosítása is lehetséges a modulra felszerelt szenzorok segítségével [15.ábra].

A feladat programozásánál fontos szempont, hogy a liftet egy mágnes mozgatásával emeljük fel, így, ha az egyik oldalt túlvezéreljük, akkor a liftet el tudja engedni a vezérelt kimenethez tartozó mágnes, így a lift lezuhan. Ezért kiemelten fontos, hogy a vezérlésnél és vészállapot esetén a féküzem a stabil üzemeltetéshez szükséges szabályok szerint történjen meg. A termék beazonosítása a Sorting modullal azonos módon itt is megtörténik, amit arra tudunk felhasználni, hogy a hibás anyag/szín kombinációjú termékeket is le tudjuk válogatni, így azokat már nem kell feleslegesen felküldeni a Processing modulra. Ezek a leválogatási és mérési funkcióka azért lényegesek, mert egy gyártás során fölösleges selejtek keletkezhetnek abból, ha olyan termékeket próbálunk megmunkálni, ami a megmunkálás előtt sem megfelelő. Ezzel felesleges költségeket lehet megspórolni, mivel a termelésbe sem kerül felesleges energia, és az alapanyagok vagy belsőleg kezelhetőek, hogy utána alkalmasak legyenek a megmunkálásra, vagy a beszállítóval egyeztetve cserélhetőek. Mivel a kapott alapanyag nem megfelelő, így ez abban is segítség lehet a beszállító felé, hogy vissza tudják követni, hogy milyen hiba okán került kiszállításra olyan termék, ami a megrendelő számára nem megfelelő.



15. ábra Testing modul - folyamatábra

Második modul (Testing) oktatási funkciói

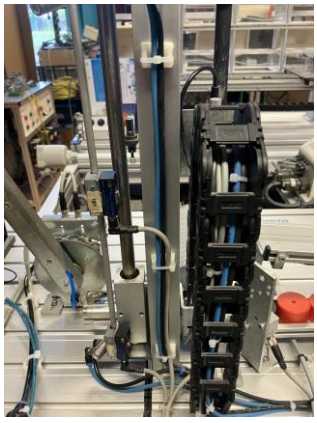
A második modul programozása során a hallgató megtanulja, hogy a különféle termékeket milyen szempontok szerint érdemes vizsgálni, milyen prioritási sorrendet kell felállítani a vizsgálat során, és megtanítja, hogy egy vezérlési szempontból érzékeny eszközt, hogy kell a programozás során kezelni.

A termék vizsgálatánál a prioritási sorrend a következő: elsődleges szempont, hogy a termék megfelel-e az anyagot és a színt érintő kritériumokkal szemben. Ez a termékek optikai vizsgálatát szimulálja, amit például a termelésben már önálló programmal rendelkező kamerákkal szokták vizsgálni. Ezeket a kamerákat a PLC csak 4 érték szintjén kezeli: vizsgálat kezdete, vizsgálat vége, vizsgálat eredménye megfelelt, vizsgálat eredménye nem megfelelő. A feladat ehhez hasonló folyamatot hajt végre: amint belépünk a folyamat vizsgálati ciklusába, elvégezzük a

termék elsődleges vizsgálatát. Az induktív szenzor egy bool típusú változóban ad értéket, hogy az alapanyag fém, vagy sem. Az optikai szenzor arról ad jelet, hogy a termék fekete színű, vagy sem. Mivel az alapanyag szempontjából csak 3 kombináció elfogadható (nem fekete és fém, fekete és nem fém, nem fekete és nem fém) így le tudjuk válogatni azokat az alapanyagokat, amik nem tesznek eleget ennek a kritériumnak, anélkül, hogy elkezdenénk a második vizsgálati folyamatot. A harmadik szenzor ezen a területen egy kapacitív szenzor, aminek az a feladata, hogy csak akkor adjon jelet, ha ott egy megmunkálandó alapanyag van. Ezt a jelet arra használjuk, hogy a második állomás folyamatait elindítsuk, és arra, hogy amíg itt van egy termék, addig ne induljon el az első modul.

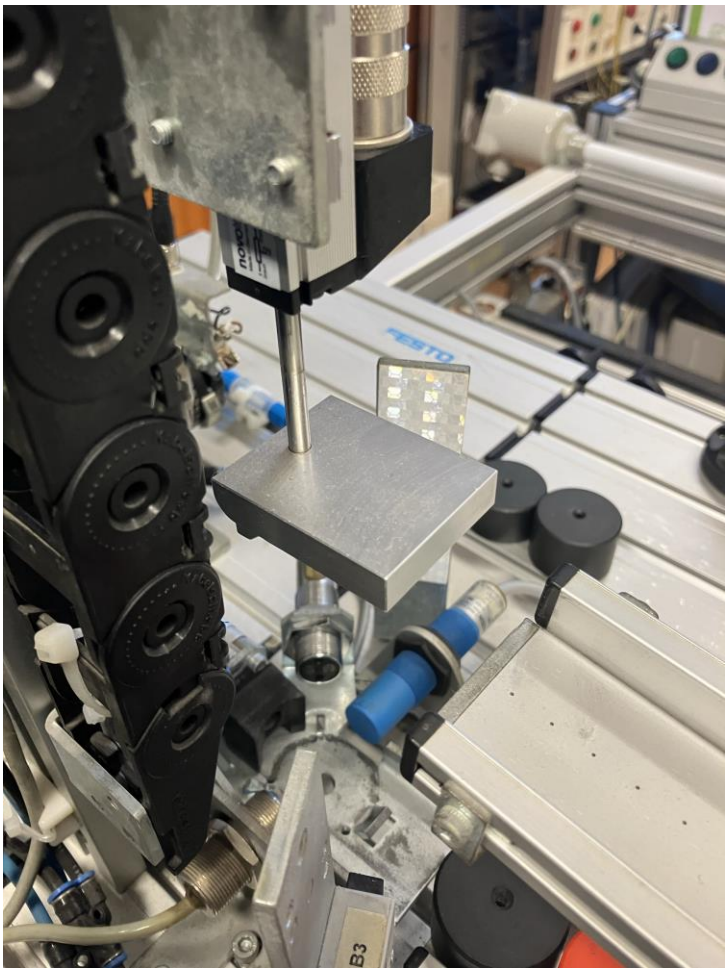
A második vizsgálati szempont a termék méretével kapcsolatos paraméterek felmérését jelenti. Ebben a szakaszban a terméket felemeljük a mérési pozícióba, és ha elértük a pozíciót kettő lehetőségünk van: ha a termék mérete megfelelő, akkor a komparátor jelet ad a PLC-nek, és tudjuk, hogy a termék megfelelő, ha nem kapunk jelet, akkor a termék mérete nagyobb, vagy kisebb, így az alsó csúszdára kell kilökni azt. Ez a típusú mérés például olyan folyamatot szimulál egy gyártásban, amikor egy felparaméterezett eszközzel hézagot mérünk. Elsőnek elvégzünk egy mérést, kiválasztjuk a hézagoló típusát, belehelyezzük a termékbe a megfelelő helyre, és elvégzünk még egy mérést. Ha a várt eredményt kapjuk, akkor a hézagoló megfelelő, ha nem, akkor ki kell cserélni.

Fontos, hogy a terméket csak akkor vizsgáljuk részletesen, ha egy előzetes teszteléseken megfelelt. Egy folyamatnál lényeges szempont, hogy a termék csak akkor kerüljön sorra bizonyos megmunkálási folyamatokra, ha az előző modulnál vagy vizsgálatnál megfelelő értékeket kapott.

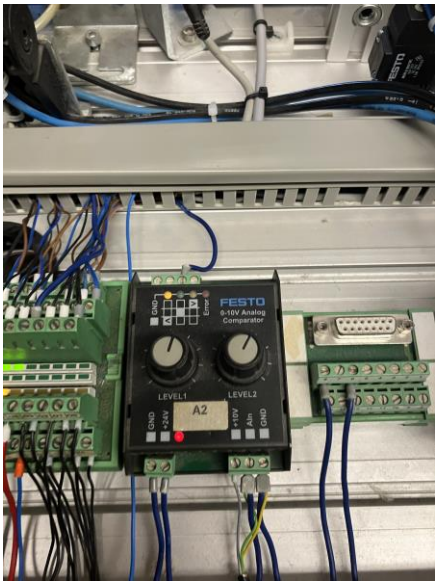


16. ábra Testing modul - Dugattyú nélküli munkahenger

17. Testing modul - Levegő rásegítéses csúszda



18. ábra Testing modul - vizsgáló szenzorok és magasság mérő



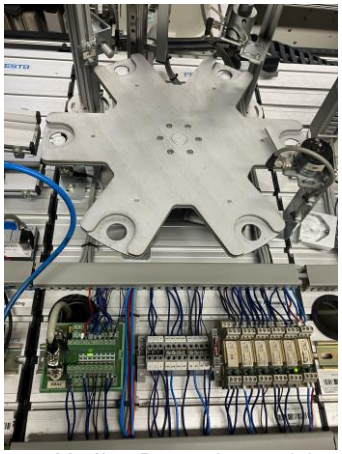
19.ábra Testing modul - Analóg jeleit kezelő komparátor

Harmadik modul (Processing) funkcióblokkja

A harmadik modul az előző feladatokhoz képest annyiban más, hogy itt egyszerre több munkadarabot kell kezelnünk. A folyamattal egy termék megmunkálását szimuláljuk [20.ábra], illetve egy egyedi szállítórendszer is része a folyamatnak. A feladat lényege, hogy a modul 100%-os kihasználtsága mellett úgy alakítsuk ki a lépésláncot, hogy a modul véletlen sem sérüljön meg. A termékeket egy körasztalon juttatjuk el a Testing modultól a Handling modulig. Ezen a körasztalon történik a termékek megmunkálása is. Minden számunkra lényeges pozícióhoz tartozik egy helyzet érzékelő, ami a furatokkal ellátott körasztalnál a termékek jelenlétérő ad jelet. Az asztal pontos pozícionálását egy induktív szenzor segítségével végezzük el. A szenzor úgy van elhelyezve, hogy csak a körasztal alján lévő csavarokat tudja érzékelni, így azokkal a csavarokkal és az induktív szenzor jelével tudjuk az asztalt forgató motort a megfelelő helyen megállítani.

A folyamat következő fő részei a megmunkálás, a vizsgálat [21.ábra] és a termék eltávolítása a körasztalról [22.ábra]. A termék 4 pozícióban lehet: az első az a pozíció, ahova az előző állomásról érkezik, a második pozícióban a termék megmunkálását szimuláljuk egy kis fúróegység segítségével, a harmadik pozícióban a furatot ellenőrizzük (mivel valós fúrás nem történik, így a termék egyik oldala be van marva, így a termék orientációja adja meg, hogy a folyamat sikeres volt, vagy sem), a negyedik pozícióban pedig egy kar segítségével a terméket ki lökjük a Handling állomás felvételi pozíciójára.

A folyamat egyik legfontosabb eleme az, hogy mindig tudnunk kell, hogy melyik pozíciókon van termék, és melyeken nincs, mivel, ha a terméket rossz időben forgatjuk át a másik pozícióra, azzal kár tudunk okozni az állomásban.

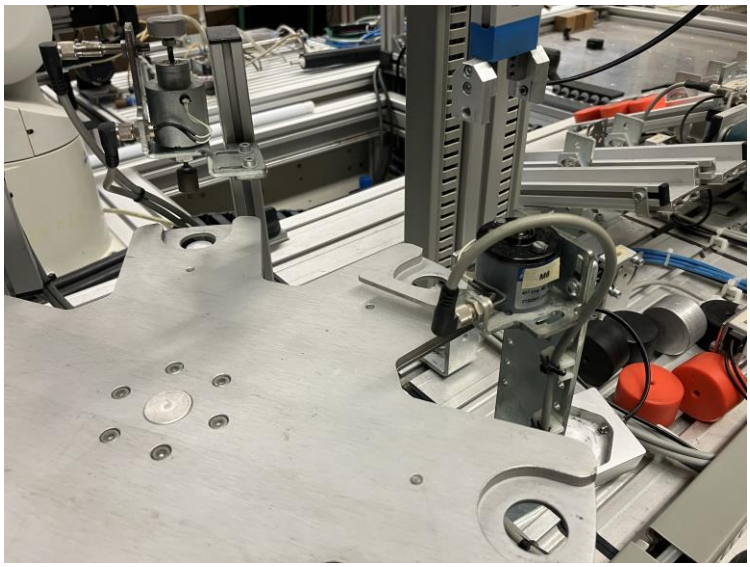


20. ábra Processing modul

Harmadik modul (Processing) oktatási funkciói

A harmadik modul egy összetett folyamatot ad ki feladatnak, ahol lényeges az, hogy nyomon kövessük a termékek pozícióit és fenntartsuk a folyamatos termék-áramlást. Ennek az a jelentősége, hogy egy gyártósor akkor tud hatékony lenni, ha a lehetőségekhez képest mindig a legjobb hatásfokon van működtetve. Az eddig tanult elemek ennél a modulnál is megtalálhatóak, viszont itt a modul biztonsága, az időzítések megfelelő használata, és a megfelelő hatékonyság fenntartása a cél. Mivel a modulnak várnia kell a Handling állomásra, így az lesz az, ami a modul ciklusidejét meghatározza. Ezt követően pedig az a meghatározó, hogy az előző modulról milyen időközönként érkezik alapanyag.

21. ábra Processing modul - megmunkáló és tesztelő egység



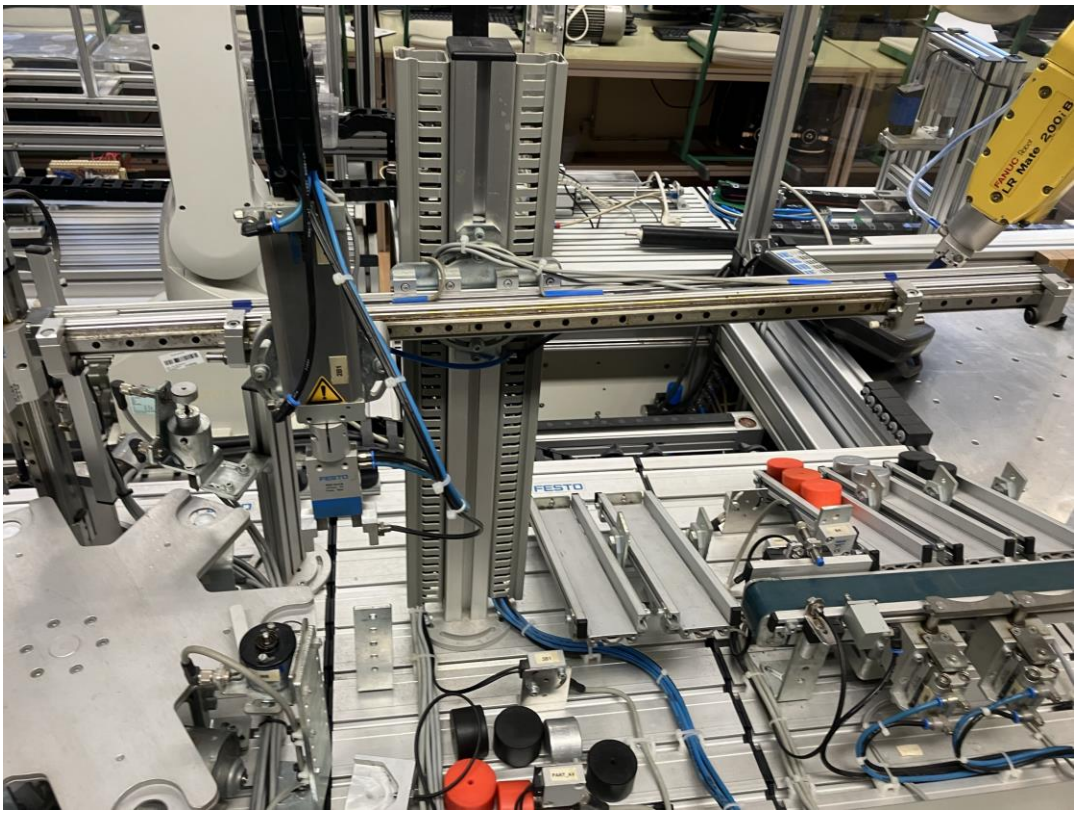
22. ábra Processing modul - Munkadarab továbbító egység

Negyedik (Handling) modul funkcióblokkja

A negyedik modul ismét egy egyszerű folyamat: A felvételi pozícióról felvesszük a terméket, és ha az nem felelt meg az előző modulnál, akkor lerakjuk a két csúszda egyikére. Ha megfelelt, akkor pedig továbbítjuk az utolsó modulra, ami a válogatást végzi. A modul programozásánál az a lényeges, hogy a beérkező termék adatok alapján kell kiválasztani a lerakási pozíciót.

Negyedik (Handling) modul oktatási funkciói

A negyedik modulnál a feladat ismét egyszerű, akárcsak az első modul esetében, annyi eltéréssel, hogy itt már három olyan pozíció van, ahova a terméket le lehet tenni [23.ábra]. Mivel a terméket már az előző moduloknál bizonyos szempontok alapján válogattuk, így ennél a modulnál csak az a válogatási szempont maradt, hogy a termék sikeresen meg lett munkálva, vagy sem. A modul önmagában nem képes arra, hogy komplex vizsgálatot végezzen, így csak az alapján dől el, hogy hova kerül a termék, hogy milyen utasítás érkezik a modul bemenetére. Ebből a tanuló megtanulhatja, hogy a különféle modulok között az adattovábbítás milyen szempontok szerint történik, és hogyan lehet egyszerre nyomon követni több termék adatait is, mivel az előző modulnál már egyszerre 4 termék van jelen, aminek a kiértékelését még a második modul végezte el.



23. ábra Handling modul

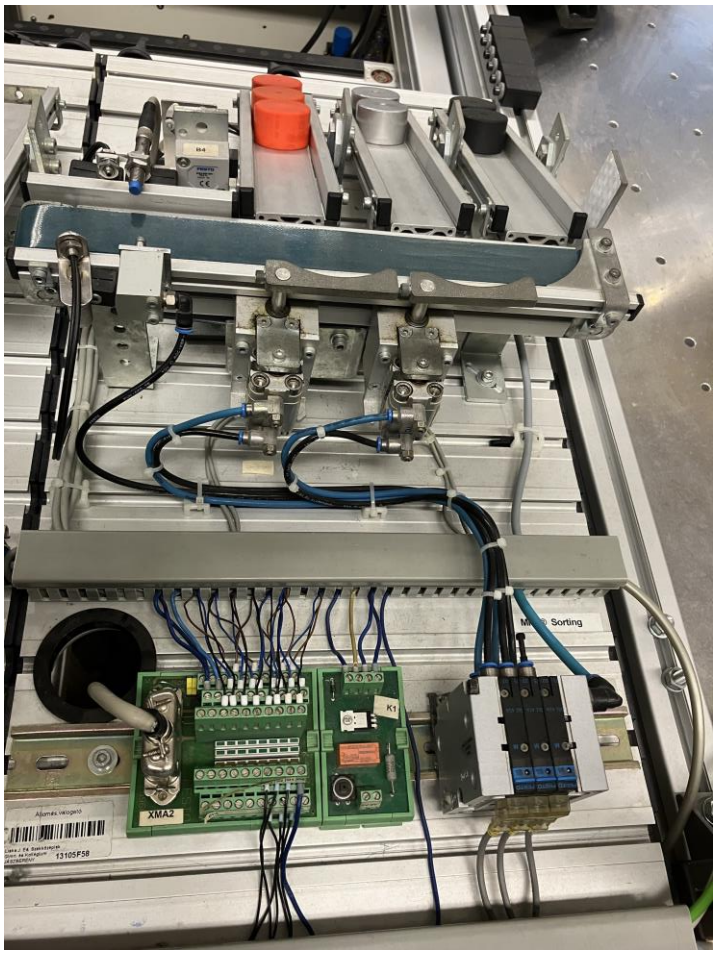
Ötödik (Sorting) modul funkcióblokkja

Az ötödik modul feladata az, hogy a futószalagra helyezett termékek anyagát és színét újra felülvizsgálja, majd azt három felé szortírozza: az egyik csúszdára a fekete műanyag termékek, egy másik csúszdára a vörös színű műanyag termékek, és a megmaradt fém termékek is egy külön csúszdára kerülnek [24.ábra].

A modul felprogramozására két lehetőség is van: vagy egy recept jellegű rendszerrel választja ki a felhasználó, hogy mi legyen a válogatási sorrend, vagy a programozó előre definiálja ezt. A saját programom esetében a második megoldásnál maradtam, mivel ezt a megoldást egyszerűbb volt kivitelezni, kevesebb a hibalehetőség, és nem szerettem volna tovább bonyolítani a programot.

Ötödik (Sorting) modul oktatási funkciói

A modul programozása során itt is a vizsgálatok elvégzése az egyik leglényegesebb szempont, a másik az energia menedzsment (csak akkor menjen a futószalag, ha termék is van rajta).



24. ábra Sorting modul

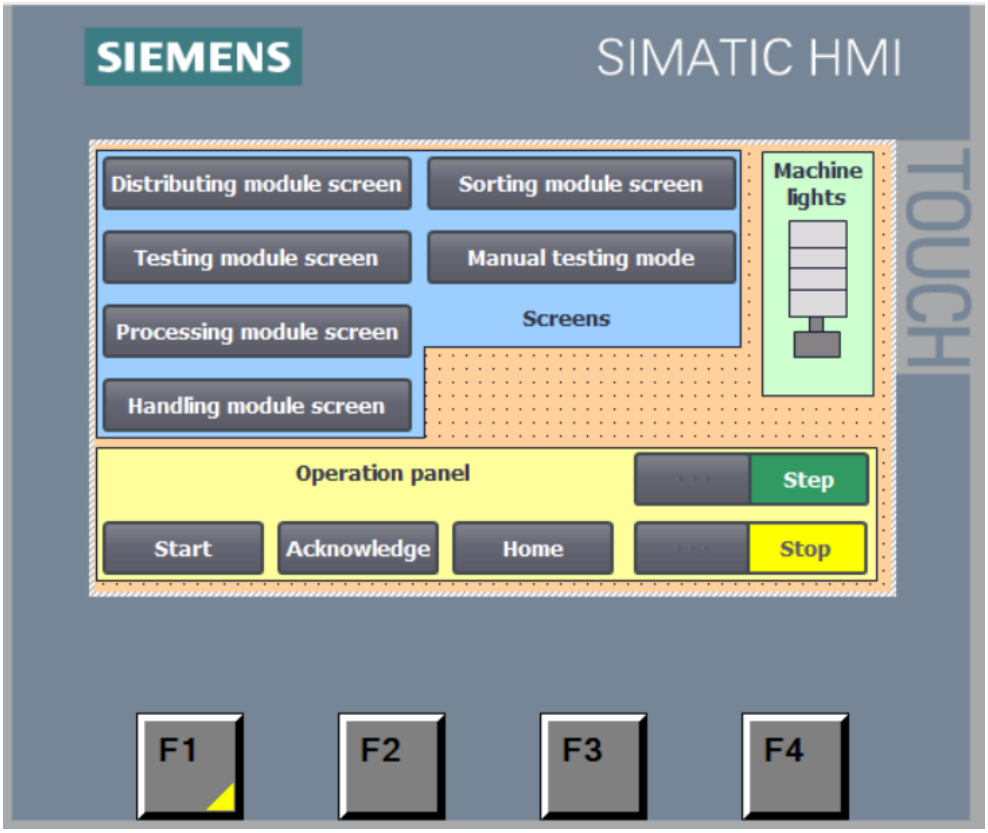
HMI képernyők kialakítása

A modulok vezérlésére az ipari megoldásokhoz hasonlóan a fizikai gombok mellett HMI-t is alkalmaztam a feladat megoldása során. A HMI felhasználásánál több szempontot is figyelembe vettem: lényegesnek tartom, hogy a HMI képernyők olyan kialakítást kapjanak, ami tartalmazza az összes lényeges funkciót és információt, legyen letisztult a kialakítása, egyszerű és egyértelmű legyen a felület kezelése, és egyértelmű visszajelzéseket adjon a felhasználó számára. Az ipari felhasználás során a HMI képernyők tartalmaznak még több olyan funkciót is, amit a feladat megoldásánál nem használtam fel. A képernyők kialakításánál törekedtem arra, hogy olyan megoldásokat használjak, amik bevezető jellegűek, könnyen elsajátíthatóak, és kellőképpen szemléletesek. A gyártásban szerzett tapasztalataim alapján a HMI felület kialakítása rengeteg időt képes megspórolni egy-egy modul kezelése során. A kialakítás során törekedni kell az egyszerűségre, ne legyenek feleslegesen hosszú, bonyolult szövegek, inkább csak hivatkozások, és olyan elnevezések, amelyek beszédesek a felhasználó számára. Emellett lehet alkalmazni egyszerű animációkat is, például a kimenetek állapotának visszajelzésére, illetve olyan

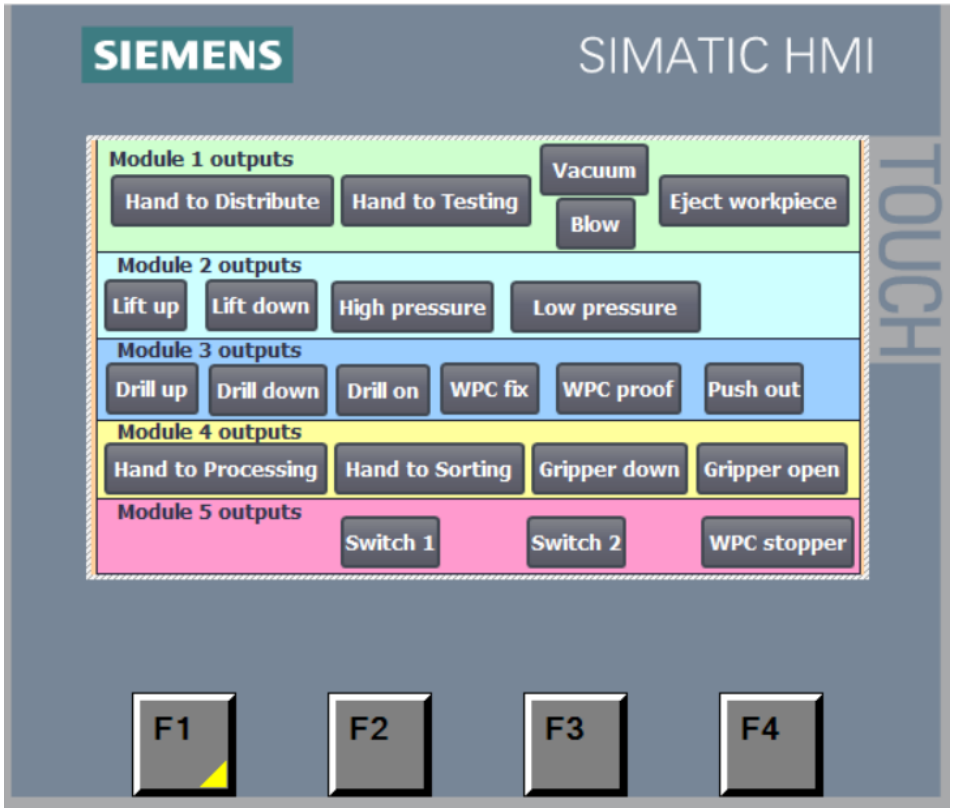
objektumokat alkalmazni, amik a felhasználás során valamilyen változást mutatnak a kezelő számára (például olyan gombokat alkalmazni, aminél a megnyomás esetén változik az animáció, így a felhasználó tudja, hogy a gombnyomás ténylegesen megtörtént a felület szoftvere szerint is).

Fontos a rendelkezésre álló felület megfelelő kihasználása: a felület ne legyen lehetőség szerint túlzsúfolt, viszont tartalmazza a szükséges elemeket. Lényeges, hogy a kijelző rendelkezésre álló felületét kihasználjuk: Nincs értelme például egy-egy olyan screen-nek, ami csak néhány gombot tartalmaz, ha a felületet akár egy másik rendelkezésre álló screen-nel is össze lehet vonni.

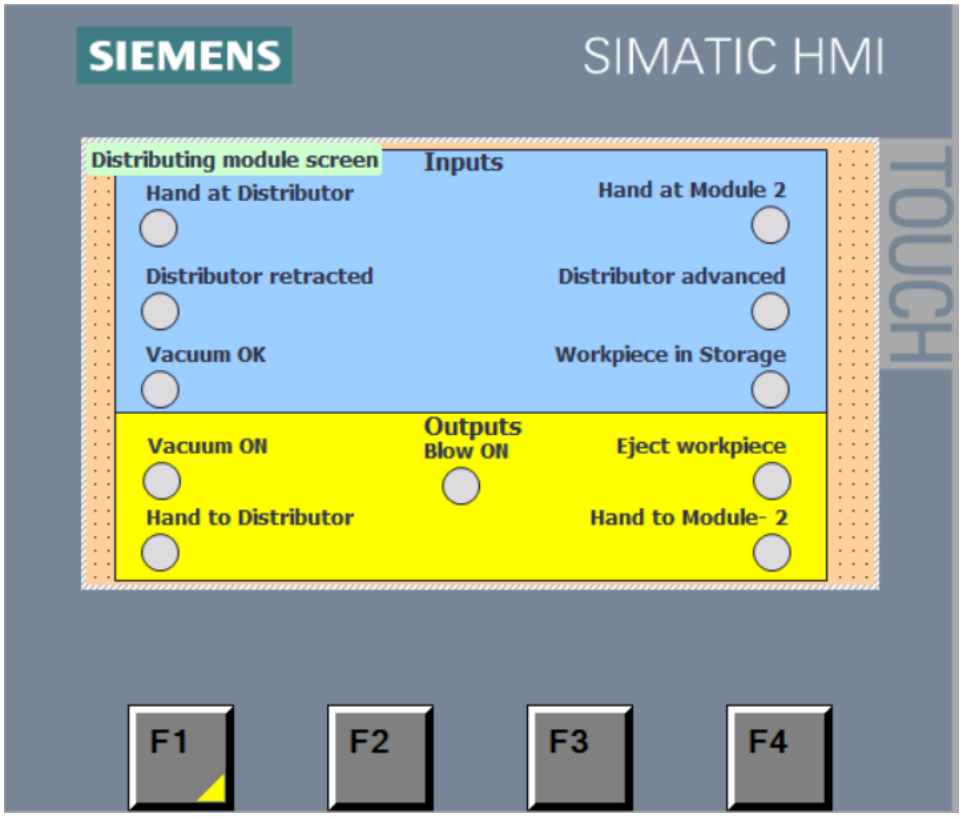
Én a programozás során az alábbi felosztásokat használtam:



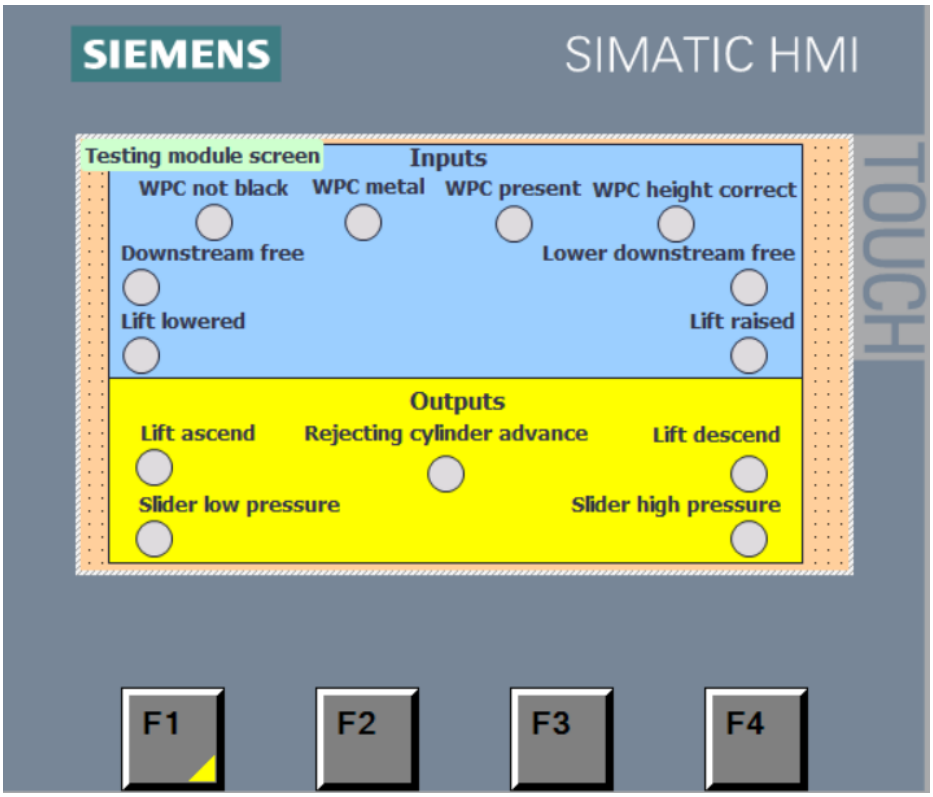
25. ábra Operation screen



26. ábra Manual testing screen



27. ábra Distribution modul screen



28. ábra Testing modul screen

A [25. ábrán] látható felület az alapértelmezett screen, ez jelenik meg a program indítását követően. Erről a képernyőről lehet navigálni az összes modulhoz tartozó saját felületre, a manuális teszteléshez tartozó felületre és a gyártósor alap állapotait is innen lehet kezelni, mint például az üzemmód váltást, a vészkört, vagy az indítási és nyugtázási folyamatokat. Ezek mellett a kijelzőn a gyártósorhoz tartozó lámpaoszlop animációja is megtalálható itt, ami visszajelzést ad a gép aktuális állapotáról. Az F1 funkciógomb ezen a kijelzőn azt az utasítást hajtja végre, hogy lenyomás esetén vissza navigál a HMI-n az előzőleg megnyitott felületre.

A [26. ábrán] a manuális teszteléshez tartozó felület található. Ha ez a felület aktív, akkor a gyártósor először vészállapotba lép, és a nyugtázást követően kerülünk át a teszt módba. A funkcióblokkok engedélyező bemenete inaktívvá válik, és a modulok kimenetei a gombokkal szabadon vezérelhetővé válnak. Ennek az a jelentősége, hogy bizonyos funkciókat így ki lehet próbálni, le lehet tesztelni, vagy szükség esetén olyan mozgásokat végezni, amik a program lefutásától eltérő sorrendben vagy variációban történnek. Az F1 funkciógomb itt és az összes többi kijelzőnél már azzal a funkcióval bír, hogy visszaállítja a kijelzőre az alap felületet. Az F2 gomb aktiválja és az F3 gomb deaktiválja a teszt módot.

A [27.ábra]. és a [28.ábra] pedig két olyan felületet mutat az öt modulhoz dedikált felület közül, amelyen a modul saját ki és bemeneteinek vezéreltségét követhetjük nyomon.

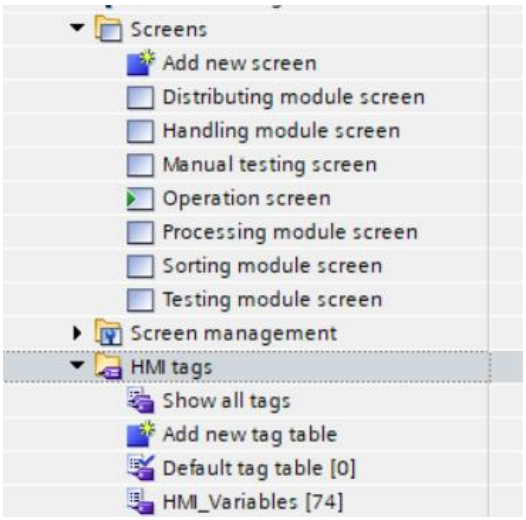
HMI változótábla

A HMI számára azért szükséges egy teljesen saját, egyedi változótáblát létrehozni [29.ábra], mert ennek segítségével le lehet csökkenteni a HMI és a PLC közötti felesleges adatforgalmat. A lényege, hogy egy HMI változótáblába létrehozzuk azokat a változókat, amiket a HMI használatánál szeretnénk használni: a gombok bemeneti változói, kimeneti változók, belső változókat. Ezt követően ezt a változótáblát össze linkeljük a PLC változótáblájával. A megfelelő változók összekapcsolását követően már a HMI és a PLC változói azonos értékeket vesznek fel és azonos időben változnak, ennek következtében a PLC reagál a HMI-n történő utasításokra, és a HMI pedig reagál a PLC felől érkező jelekre. Ha ez úgymond direkt vezérléssel történne a PLC táblájából, akkor a HMI és a PLC között felesleges adatforgalom lenne minden egyes PLC ciklus során. Így viszont csak akkor történik adatmozgás, ha a két tábla között eltérés keletkezik.

A HMI változó tábla a következő fő szekciókból tevődik össze [30.ábra]:

- Name: itt lehet megadni a változók nevét (Szabvány szerint kerülendő az ékezetes betűk és a szóköz használata, mivel a program lefutásánál gondot okozhat)
- Data type: itt lehet megadni a változó típusát, ezek között megtalálható az összes standard típus, amit a mikrokontrollereknél is szoktak alkalmazni
- Connection: itt adható meg, hogy a kommunikáció során melyik szálon végezze el a kommunikációt a rendszer
- PLC name: A PLC nevét adjuk itt meg, amivel a változótáblánk kapcsolatban áll
- PLC tag: itt jelöljük ki, hogy a HMI táblájának kijelölt változója a PLC melyik változójával legyen összerendelve
- Address: ebben a mezőben láthatjuk, hogy a változónk melyik címet tudja írni és/vagy olvasni
- Access mode: itt a Symbolic és az Absolute access között választhatunk. Ennek az a jelentősége, hogy a HMI változója közvetlen ráhatással legyen az adott címre, vagy csak szimbolikus hatása legyen, azaz a PLC változót írhatja át, ami így módosítja a címen lévő értéket.
- Aquisition cycle: Itt adható meg változónként az ciklusidő, amin belül a két tábla összekapcsolt értékeit összehasonlítja a rendszer. Ez azért lényeges, mert így testre lehet

szabni, hogy az adott változó milyen gyakorisággal legyen ellenőrizve. Például egy gomb lenyomása az emberi reakcióidő miatt nem ugyan azt a lekérdezési gyakoriságot igényli, mint amikor egy analóg értéket akarunk módosítani



29. ábra HMI táblák helye a könyvtár rendszerben

HMI_Variables

Name	Data type	Connection	PLC name	PLC tag	Address	Access mode	Acquisition cycle
Hand_at_Module2_HMI	Bool	HMI_Connectio...	PLC_1	Kar_mod2nel		<symbolic access>	1 s
Turntable_WPC_at_drilling_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Turntable_WPC_at_checking_H...	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Turntable_in_position_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Turntable_drill_ascended_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Handling_gripper_descended_...	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Handling_slide_free_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Handling_WPC_black_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Handling_gripper_descend_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Handling_gripper_open_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	100 ms
Sorting_WPC_metal_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	1 s
Sorting_WPC_not_black_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	1 s
Sorting_WPC_present_HMI	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	1 s
Sorting_First_switch_retracted_...	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	1 s
Sorting_Second_switch_retract...	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	1 s
Sorting_Second_switch_advan...	Bool	HMI_Connectio...	PLC_1	<Enter PLC tag>		<symbolic access>	1 s

30. ábra Példa HMI tábla kialakítására, táblázat elemeinek bemutatása

PLC programozáshoz és hibakereséshez szükséges ismeretek és módszerek

A szisztematikus hibakeresés módszere

Az általam használt hibakeresési módszer a pneumatikus/hidraulikus rendszerek tervezése és felépítése során használt egyik módszer logikáján alapszik. Az itt felhasznált módszerek a Kaszkád-módszer és a csoportbontás módszere. A Hibakeresési metódus a csoportbontásos rendszert veszi alapul. Az a rendszer, amiben egy hibát keresünk mindig részekre bontható. A hibakeresés során ismerjük a hibát, például azt, hogy melyik lépésnél állt meg a folyamat, vagy milyen „tüneteket” produkál a rendszer a hibás működés közben. Először mindig egy-egy nagyobb részegységre bontjuk fel a folyamatunkat, és a hozzá tartozó rendszereket. Ilyenek lehetnek például a hajtás rendszer, a pneumatikus/hidraulikus körök, az ezekhez kapcsolódó szenzorok, a kézi scannerek, csavarozóegységek és így tovább. A hiba jellemzőiből tudunk következtetni, hogy milyen csoportok kerülhetnek szóba a hiba keresése során. Ezt követően meghatározhatjuk, hogy a hiba szoftveres vagy hardveres jellegű. Sok esetben a hardveres hibákat is szoftveres segítséggel egyszerűbb megtalálni. Például, ha van egy szenzorunk, amit sejtünk, hogy rendellenesen működik, akkor arra tudunk beállítani egy scope-ot (a TIA Portal környezetében Trace-nek nevezzük ezeket), így a szenzor jelét időtartományban vizsgálva megláthatjuk azt, ha a szenzorjel nem konzisztens, szakadozott, vagy bármiféle fizikai változás nélkül triggerel. A másik szoftveres hibakeresési lehetőség például a változólista aktív vizsgálata a lefutás közben. Ha egy kimenet/bemenet értéke megváltozik, akkor az a monitorozott változótáblán rövid időn belül látható (max 1 teljes PLC ciklus ideje alatt jelet vált a táblán kijelzett érték is). Így ha például ha a táblában egy kimeneti jelet látunk a PLC által módosulni, de az eszköz nem reagál, akkor annak a következő okai lehetnek: Kommunikációs hiba miatt nem jut ki a jel a végrehajtó egységhez, aminek az oka vagy egy sérült kábel, vagy egy hibásan működő I/O sziget lehet, lehetséges még, hogy a működésnek fizikai korlátjai vannak, mint például hogy nem kap elegendő levegőt egy munkahenger a mozgáshoz, vagy túlzott mértékű fojtás lett hozzá beállítva, így nem tud mozdulni, illetve lehet olyan hiba is, ami nem látható, de a gép mozgásából ered, mint például egy hibásan elvezetett gégecső a robotra erősítve, ami feszül, és súrolja a csőben elvezetett kábeleket és/vagy levegő csöveket. A megfelelő felbontásokkal és vizsgálatokkal hatékonyan lehet a hibát keresni és a felesleges tényezőket kizárni, és sokkal effektívebb, mintha egy koncepció nélkül keresnénk hibát a rendszerben. Emellett ezzel a módszerrel vissza kereshetők a meghibásodás okai is, ami hosszú távon segíthet az újabb

meghibásodás megelőzésében. Ez a módszer nem összekeverendő a Méréstechnika során tanult szisztematikus hiba jelentésével, mivel ezzel a módszerrel, olyan hibákat is fel lehet deríteni, amik nem a szisztematikus hibák csoportjába sorolhatóak.

A módszer lényege összefoglalva a következő: az adott hibajelenségeket listázzuk ki, vegyük számításba, hogy mik okozhatják a hibát, csoportbontással szűkítsük le a hibakeresések területeit, és kizárásos alapon vizsgáljuk ki a hiba okozóját, majd állapítsuk meg a hiba gyökérokát.

A hibajavítás esetén mindig törekedni kell a hibát megkerülő módszerek kialakításának elkerülésére. Átmeneti megoldásként jó lehet, ha meg tudjuk kerülni a hibát, hogy a gép működjön, viszont azt az alapvető tényt figyelembe kell venni, hogy a folyamatnak úgy is működnie kellett, hogy ezek a funkciók nem voltak kiiktatva/megkerülve. Minden egyes ilyen megkerüléssel a gép működését befolyásoló tényezőket iktatunk be, amik növelhetik a gép ciklusidejét, vagy kivehetnek olyan funkciókat, amik a minőségi paramétereket és a keletkező selejtek számát jelentősen befolyásolják.

PLC program kialakítása az IEC-1131-3 szabvány szerint

Ahogy az Jancskárné Anweiler Ildikó kéziratában is olvasható [5], szabvány a PLC program legkisebb egységének a POU-t (Program Organisation Unit – Programszervezési egység) jelöli ki. Ez az egység lehet függvény, funkcióblokk és program. A legmagasabb szintet a program adja meg, amin belül meghívhatunk függvényeket és funkcióblokkokat egyaránt. A program kialakítása két részből tevődik össze: a deklarálási szint és a programtörzs.

A program kialakítása során azért szükséges a szabvány szerint eljárni, mivel ezzel a metódussal hatékonyan ki lehet aknázni a PLC-k hardveres és szoftveres lehetőségeit, és egy olyan programot tudunk létrehozni, amely mások számára is könnyebben értelmezhető.

A szabvány ezek mellett kitér a felhasználható változók típusaira, és azok deklarálására, a Funkcióblokkok és függvények kialakítására és paraméterezési lehetőségeire is.

Eszközök közötti kommunikáció

A Master-PLC a Slave-PLC és a HMI közötti kapcsolat lebonyolítására a Profinet kommunikációs metódust alkalmaztam. Ez az egyik lehetőség, de lehetett volna alkalmazni Profibus vagy más az iparban használt kommunikációs metódust is. A választásom azért erre a típusra esett, mivel az adatátviteli sebesség, az egyszerű kiépíthetőség és az iparban való gyakori alkalmazása miatt egy olyan kommunikációs típus, amit szinte már alapvető hálózati

ismeretekkel rendelkező mérnök vagy technikus is könnyen tud alkalmazni egy-egy projekt megvalósítása során. A kommunikáció egy switch-en keresztül történik, amihez a szabványosított profinet kábelt használtam. A PLC-ken két-két darab port van, így a további bővítés még lehetséges, ha olyan eszközöket akarunk csatlakoztatni a PLC-kre, amiknek külön kommunikációs port szükséges, de lényeges az adat közlés sebessége is (például scannerek, kamerák).

A feladat megoldása közben felmerült hibák és megoldások

A legelső hibámat a Distribution modul alaphelyzeti pozíciójának meghatározásánál követtem el. Az átrakó kar alaphelyzeti pozíciójának a lerakási pozíciót választottam meg. Ez azért lehet problémás, mert az átrakó kar ebben az esetben megakadhat a liftben, ha valamilyen oknál fogva a kar előbb érne arra a pozícióra, mint hogy a Testing modul liftje elérné az alsó alaphelyzetét. Ezt úgy küszöböltem ki, hogy az alaphelyzeti pozíciója az átrakókarnak innentől egy tetszőleges pozíció lett a két végállás között. Mivel alap esetben a karhoz csak 2 végállás kapcsoló tartozik, így a köztes pozíciót nem tudtam pontosan beállítani, így a kart egy TON típusú időzítő segítségével állítom meg a két végállás között. Ezt a módosítást elvégeztem a program alap lefutása közben is.

A következő hibám a vészállapot kezelésben történt, és az összes modult érintette. Alapból úgy alakítottam ki a programot, hogy minden modult azonos funkciógombokkal állítok vészállapotba, nyugtázok ki és állítok alaphelyzetre. Ez az első két modul esetében mechanikai károsodáshoz vezethetne. Ezért a módosítás alapján csak a vészállapotot kezelem egy funkciógombbal, és az összestöbbi funkciót, ami az automata/lépés üzemmódváltást, a nyugtázást és az alaphelyzetre futást hajtja végre mind egyesével példányosítottam. Ezt követően úgy módosítottam a funkcióblokkokat, hogy egy alaphelyzetre futási szekvencia alapján a modulok egyesével, a Sorting-tól visszafelé haladva külön-külön nyugtázást kérnek és alaphelyzetre térnek a Distribution modullal befejezve a sort. A folyamat, során mindig feltétele lesz a moduloknak, hogy a soron következő modul előtte elérje az alaphelyzetet, addig nem fogja engedni a vészállapotról történő nyugtázást.

Az üzemmódváltást azért kellett példányosítanom, mert egy valódi környezetben ezek a beavatkozások mindig csak egy modulra/modulcsoportra vonatkoznak, globális hatása csak a

vészállapot kapcsolónak lehet. Mivel a tesztelés során elegendő mindig a vizsgált modult lépés üzemmódba helyezni, így az összes modulra kihatással lévő beavatkozás szükségtelen.

Ezek a módosítások az alapértelmezett HMI screen módosítását jelentették, így a módosított screen a következőképpen lett összeállítva.

A modulok alapfelszereltsége közben annyiban módosult, hogy sikerült egy megfelelő lámpaoszlopot is felszerelni a modulcsoportra, így az állapot kijelzése az első modul esetében fizikailag is meg tudott valósulni. Viszont mivel a példányosítás miatt a modulok funkcióit külön kezelem, így létre kellett hozzak több szimulált lámpát is, mivel, ha egy lámpát vezértelt volna az összes modul, akkor értelmezhetetlen lett volna a lámpa által adott visszajelzés.

A modulok programozásának oktatásához felhasznált alapismeretek

A programozás a következő alap szempontok szerint épült fel. Az első, hogy a feladat leírása alapján létre kell hozni egy lefutási diagramot. Ennek az a jelentősége, hogy a szövegesen megfogalmazott feladatot megfelelően értelmezve elemeire tudjuk bontani. Egy flowchart sokat tud segíteni abban, hogy a programnak egy egyszerűbb verzióját írjuk meg, mert egy ilyen ábrából egyszerűen ki lehet szűrni a felesleges programrészleteket, illetve könnyen észre lehet venni azokat a pontokat, ahová be lehet építeni egy plusz funkciót, ha szükség lenne rá.

Ezt követi az állapot diagram, ami a kimenetek és a bementetek változását ábrázolja egy azonos skálázású diagramsorozaton. Ez annyiban tud segítséget nyújtani, hogy a lefutási szekvenciát könnyebben meg lehet írni ennek segítségével, mivel a diagramok alapján egyértelmű ok-okozati összefüggéseket levonni, és ennek segítségével meg lehet írni a működő alap szekvenciát. Ennek egyszerűsítésére használható az úgynevezett Kaszkád-módszer, de ez már csak akkor javasolt, ha van egy működőképes alapszekvenciánk, amit szeretnénk optimalizálni.

Ezt követően létrehozzuk a főprogramot, és a funkcióblokkokat. Az alaphelyzet mindig az, hogy a különféle modulok programjait külön blokkokban kezeljük le, és ezeket meghívjuk a fő programban. Ha a fő programban írnánk meg a modulok programjait, azzal két fő probléma lenne: A modulok funkciónak másolása bonyolultabbá válna, és a másik probléma az, hogy így sokkal több erőforrást kell igénybe vennie a rendszernek, hogy végig léptesse a teljes programot. A funkcióblokkok felhasználásával megnyílik a példányosítás lehetősége, így a blokkokat tetszőleges darabszámban lehet sokszorosítani. A másik előnye az, hogy az így létrehozott

funkcióblokkokat könyvtárhoz tudjuk adni, amit így a fejlesztőkörnyezetből bármikor elő lehet venni és felhasználni.

A program megírása során a funkcióblokkokat jelen esetben úgy hoztam létre, hogy a blokkok egyben tartalmaznak egy-egy modult. Ezt tetszőlegesen lehetne még tovább bontani. Példa lehet erre, hogy a különféle munkahenger vezérléseket le lehet külön funkciókba vagy funkcióblokkokba menteni, és így a későbbi felhasználásnál csak meg kellene őket hívni a modulhoz tartozó blokkon belül. Ezt azért nem így írtam meg a feladat során, mert attól tartottam, hogy ha nagyon „felaprózom” a programot, akkor azzal több hibalehetőséget iktatok be a funkcióblokkomba.

Ha megírtuk az alap szekvenciát, akkor ezt utána kiegészítjük az automata/lépés üzemmóddal. Ezt nem bonyolult beiktatni, és mivel a program szekvenciát egyszerűbb állapotgépes rendszerben, CASE utasítás használatával kialakítani, így a lépésközöket is egyszerűen meg lehet határozni.

A következő funkció a vészállapot és a nyugtázás kialakítása. Itt a már korábban ismertetett elmélet szerint kell eljárni: mivel a különféle aktuátorok más-más vészállapoti protokoll szerint kell működjenek, így ezeket mindig a szituációra szabottan kell kialakítani. A vészállapot szekvencia és a nyugtázás kialakításánál szintén érdemes egy flowchart elkészítése, mivel sokat könnyít a szekvencia átlátható kialakításában és a hibák észrevételében.

Ha a vészkör megírása megtörtént, akkor az alaphelyzetre futás szekvenciája következik. Ezt pedig mindig úgy kell megírni, hogy a lehető legtöbb opcióval számoljunk: Milyen pozícióból indulunk alaphelyzetre, milyen folyamatok akadtak meg a vészállapottal, és ha van termék a modulban, akkor azt milyen módon kell kezelnünk. Ezek mellett egy modulcsoport esetében mindig figyelembe kell venni azt is, hogy az adott modul alaphelyzetre mozgása közben ne léphessen fel olyan állapot, ami a modulok károsodását okozhatja.

Ezt követően, ha szükséges a programot ki lehet egyéb biztonsági funkciókkal bővíteni, mint például a kétkezes indítás.

Ha a programblokkok összeálltak, akkor ezeket meghívjuk a fő programba. Ezt követően létrehozzuk a változó listákat, amik már azokat a kimeneti és bemeneti címeket tartalmazzák, amiket a funkcióblokkok kimeneteihez és bemeneteihez rendelünk.

Ezt követi a HMI felületek kialakítása. A HMI-nél mindig először létre kell hozni a HMI változóit, és létre kell hozni a változók összerendelését a PLC változótáblájával. Ez azért szükséges, mert ha direkt módon használnánk a PLC változótábláját, akkor a kijelzőnk minden egyes PLC ciklus alatt megkapná a ki és bemenetek értékeit, ami azért szükségtelen, mert a HMI-k általános képfrissítési frekvenciája 60 Hz körül van, ami 60 képkockát jelenít másodpercenként. Mivel a PLC ciklus ideje ettől jóval magasabb, így hiába kapna ettől magasabb frekvenciával jelváltásokat a HMI, a kijelző nem lenne képes megjeleníteni azt. A két tábla alkalmazásával viszont a hálózaton csak akkor történik adatcsere, ha változás következik be, így a szükségtelen adatforgalmat el tudjuk kerülni.

Oktatási metódusok, és az iparban szükséges tapasztalat összehasonlítása

Az oktatások első szakaszában a hallgatók megismerkednek az utasításlistával és a Létra Diagrammal. Ezt követően az oktatás nagyrészében már csak az LD programnyelvet használják a hallgatók. Az egyetemi oktatás során foglalkoztunk SCL nyelvvel is, de arra már sajnos nem jutott megfelelő mennyiségű idő az oktatási óraszámok kerete miatt. Ezzel az a probléma, hogy az LD nagyon jó arra, hogy egy villamos ismeretekkel rendelkező tanuló meg tudja tanulni a PLC programozás mögött álló alap logikát, viszont a programozási nyelv használata az iskolai példákon kívül más környezetben már nem elég hatékony. Ezért a hallgatóknak több időt kellene adni az SCL és az FBD programnyelvek megismerésére és elsajátítására. Ennek az az oka, hogy a programozók, akik az ipari eszközöket programozzák már a kész, saját maguk által elkészített funkcióblokkokat alkalmazzák. Ha a hallgató nem sajátítja el ezeknek a nyelveknek a használatát, akkor a hibakeresési ideje sokkal nagyobb lesz, még akkor is, ha egy áttekinthető kódot lát maga előtt, mivel nincs meg a kellő tapasztalata, hogy megfelelő hatékonysággal tudja áttekinteni a kódot.

Az oktatások során sajnos a hibakeresésre csak annyi idő van fektetve, amennyi a gyakorlatok során keletkező hibák keresése során adódik. Ezzel az a baj, hogy ha egy hallgató hatékonyan tud programot írni, az még nem biztos, hogy könnyen is tud egy más által megalkotott kódban hibát keresni. Mivel a hallgatók nem látnak sokkal összetettebb, mások által írt programokat, így nem is minden esetben tudják elsajátítani a szisztematikus hibakeresés módszerét. És ha ezt nem sajátítják el, akkor a hibakeresés menete is hosszabb, mivel akkor csak az a metódus marad, hogy a hallgató szép sorban végig néz mindent a kódban, és mondhatni tűt keres a szénakazalban.

Ez a helyzet hatványozottan rosszabb, ha az ismeretlen kód nincs megfelelően kommentezve, illetve, ha a változó nevek nem elég „beszédesek”. Például, ha csak annyi van egy kódrészletben, hogy: „ha A és B aktív és a C nem aktív, akkor legyen aktív a K kimenet”, akkor ebből nem igazán lehet rájönni, hogy az adott változóknak mi a funkciója, ezért következhet a címek kikeresése az IO listából, amit aztán össze kell vetni a modulhoz tartozó elektronikus dokumentációval. Viszont, ha a változó nevek megfelelőek, akkor ez elkerülhető: „ha A_munkahenger_felso_vegallas aktív és B_munkahenger_felso_vegallas aktív és C_megfogo_nyitva nem aktív, akkor a K_kiloko_munkahenger_kitolas legyen aktív. Már csak a változó nevek módosításával elérhető, hogy a hibakeresés folyamata könnyebb legyen. Erre a felsőoktatásban, mérnöki szinten figyeltek, és megértették a hallgatókkal ennek a fontosságát, viszont a technikusi szinten ez nem rögzül a hallgatókban. így a programok kialakításánál ragaszkodnak az I1, I2, O1, O2, M1, M2… változó nevekhez. Ezzel egy oktatási környezetben még nincs akkora probléma, mivel a legbonyolultabb feladatoknál sem használják ki a rendelkezésre álló ki és bemeneteket. Viszont egy komolyabb programnál, ahol a PLC több ki-és bemeneti bővítővel van felszerelve, ott már annyi lehet a változó, hogy az indexek és a számok használata már meg tudja kavarni a programozót a folyamat közben.

A következő probléma a funkcióblokkok használatának oktatása során történik. A technikusi szinten ez csak említésre kerül, de a programokat mindig a fő rutinban írják meg, és legtöbbször egy egyszerű lefutó szekvenciát használnak. Ez azért nem elegendő, mert egy komolyabb program már rengeteg funkcióblokkot használ, amiknek a működésében, a blokkok meghívásának módjában ki kell igazodni, ha a program lefutásának hibáit keresi a programozó.

A következő szint már az, amikor a hallgatók nem csak modulokban, hanem rendszerek szintjén képesek átlátni a folyamatokat. Ez azért fontos, mert így a hallgató már rengeteg időt tud spórolni a programozás során, ha nem csak az adott modulokra koncentrál, hanem összességében nézi a folyamatot. Így sokkal könnyebben tud ciklusidőt optimalizálni, és úgy tudja megírni a modulok programjait, hogy a folyamatok módosítás nélkül is egymásra tudnak épülni, és nem kell plusz tesztelési időt szánni a külön szempontok szerint programozott modulok összehangolására.

Fejlesztési javaslatok, amik még implementálhatóak a programba

HMI felülethez köthető fejlesztések

Az egyik fontos rész, az a hibaüzenetek létrehozása, kezelése és kijelzése. A hibakeresési folyamatokat sokban tudja könnyíteni, ha bizonyos funkciók Error ágához egy-egy hibaüzenetet társítunk. A TIA Portal-ban erre van egy külön pont, amit ProDiag-nak neveznek. Ide létre lehet hozni Operator üzeneteket, Warning üzeneteket és Error üzeneteket. Az Operator és a Warning üzenetek nem állítják meg a folyamatot, míg az Error esetében az automata mód és a szekvencia lefutása megáll. Ezeket az üzeneteket valamilyen nyugtázási folyamattal resetelni kell, ha a hiba megoldódott. Az Error üzeneteket nem csak a hibák kijelzésére szokták használni, hanem akkor is, ha valamilyen lényeges folyamatrész miatt meg kell állítani a folyamatot, mint például zsírozási folyamatok és ragasztási folyamatok (ha egy adott időnél tovább marad a zsír vagy a ragasztó a fejben, amivel a termékre kerül az adott anyag, akkor az be tud száradni, ezzel eltömítve a rendszert). Ezek az üzenetek a megfelelő beállítások alapján több nyelven is kiírathatóak a HMI felületére, és akár egy log-ban tárolni is lehet őket. A log, vagy jegyzék alkalmazása amiatt lényeges, mert előfordulhat olyan szituáció, mikor a hiba csak egy tűimpulzus idejére jelenik meg, viszont a folyamatot megállítja. Ha a hiba megszűnik, akkor a kijelzése is megszűnik a HMI-n, viszont, ha már egyszer is felvillan akármilyen kis időpillanatra, akkor azt a rendszer automatikusan a log-hoz adja.

A következő hozzáadható funkció a több nyelvű kijelzés alkalmazása. Ha a programban nem is, de a HMI-n biztosítani kell. hogy legalább három nyelvi opció közül lehessen választani a kezelőnek: alap esetben ezek az angol, a német, és a munkakörnyezet anyanyelve szoktak lenni. Ez amiatt lényeges, mert a német és az angol világnyelvek, így akár a beszállítók, akár külsős programozók is képesek olyan nyelven használni a HMI-t, amit meg is értenek. A saját anyanyelv viszont amiatt lényeges, mert például egy hazánkban élő és itt dolgozó operátor nem biztos, hogy képes a saját anyanyelvén kívül bármilyen másik nyelven olvasni, így a felület kialakításánál kell ez a lehetőség is, hogy az operátoroknak szánt üzeneteket és utasításokat el tudják olvasni.

A fejlesztési lehetőségek közé tartozik a receptkezelés. A receptkezelések segítségével határozhatóak meg a gyártás különféle paraméter csoportjai, amik termékspecifikusak, ezáltal lehet köztük eltérés. Ennek a funkciónak az a jelentése, hogy a recept szerint lementett paramétercsoportok könnyen módosíthatóak, vissza lehet keresni a verziószámokat és az előzetes verzióhoz képest az eltéréseket, valamint így egy egyszerű metódussal az átállási folyamat során

a paramétereket nem szükséges manuálisan módosítani, hanem központilag egy lista elem kijelölésével meg lehet adni, hogy milyen terméket, és ezáltal milyen előre meghatározott paramétereket vegyenek fel a modulok/modulcsoportok.

Az utolsó lényeges HMI-hez köthető opció, amivel ki lehet egészíteni a programot az a jogosultságok kezelése. Lehetőséget ad a rendszer arra, hogy különféle jogosultsági szinteket adjunk meg, külön felhasználóval és jelszóval védve. Ez amiatt lényeges, mert így különféle szinteket, hierarchiát lehet felállítani a kezelők között. Mivel az operátoroknak nem szabad a különféle beállításokat és a módváltásokat kezelni, így nekik lehet adni a legalsó, korlátozott szintet. A technikusoknak lehet adni azt a szintet, amikor már a kezelőfelületek elérhetővé váljanak, és az is, hogy különféle paramétereket tudjanak módosítani. Efelett állhat a mérnöki szint, akiknek már egy kicsivel több paraméterhez lehet hozzáférése. Efelett a Quality osztály áll, akik olyan paramétereket tudnak már állítani, ami a termék minőségi jellemzőit is befolyásolhatja. A legfelső szint pedig a Developer vagy Admin szint, amivel minden HMI-hez köthető beállítást elérhetünk a felületen, amit lehetséges módosítani, azok közül bármit tudunk módosítani.

PLC programhoz köthető fejlesztések

Az egyik lényeges pont, amivel ki lehet egészíteni a programot, az a WDT-k használata. A WatchDogTimer-ek feladata az, hogy a különféle folyamatok felett egy afféle biztonsági felügyeletet kapjon a rendszer. Például ha tudjuk, hogy a modul egyik elemének mozgási ideje 10 másodperc, akkor a végállás szenzorokat egy-egy ilyen időzítővel összekapcsolva tudjuk a folyamat idejét ellenőrizni, és ha a mozgás nem éri el a beállított biztonsági időn belül a a folyamat idejét ellenőrizni, és ha a mozgás nem éri el a beállított biztonsági időn belül a végállást, akkor egy hibát tudunk ezáltal aktiválni, ami már specifikus, konkrét információt ad a kezelő számára, mint pl: „Hiba: A préselő munkahenger végállás hiba! Kérem ellenőrizze az IX2.2-es végállás érzékelőt, és a MB15.2-es útváltó szelepet, és a hozzá tartozó fojtószelepeket!” Az ilyen módon szerkesztett üzenetekkel és ezekkel az időzítőkkel sok olyan hibakeresést le lehet szűkíteni, ami visszatérő hiba lehet a mechanikai elhasználódás miatt, és le lehet szűkíteni a hiba okozóinak sorát is.

A következő fejlesztési lehetőség a modul funkcióblokkjaihoz tartozó Error ágak kialakítása, és azok nyugtázásának kialakítása. Egyszerűbb úgy hibát észlelni a program lefutásánál, ha a funkcióblokk alap esetben vissza jelez nekünk egy Error ágon, hogy valami probléma történt a

blokkhoz tartozó szekvencia lefutása közben. Ezt szintén össze lehet kapcsolni egy hibaüzenettel, így, ha például egy kézi scanner-hez tartozó funkcióblokk sorozatosan hibát dob fel a folyamat során, akkor kizárásos alapon lehet tudni, hogy vagy a kézi scanner, vagy a scanner-hez csatlakozó adatkábel a hibás.

Fejlesztési lehetőséget jelenthet még az, hogy a projekthez társítjuk a vállalati szintek kialakítását. Egy PLC projektnél lényeges lehet, hogy ne csak lokálisan legyen eltárolva a projekt file, ami a lefutás kódját és a specifikációkat tartalmazza, hanem egy külső, hálózatról elérhető tárhelyen is. Az efféle tárolási megoldás amiatt lényeges, mert a PLC projektet adott esetben nem csak egy PC-ről kell elérni, mivel a programozók általában a számukra kiadott, a cég tulajdonában lévő gépről dolgoznak. És ha bármilyen módosítás történik, a szerveren tárolt verziót minden felhasználó ugyan abban a formában képes megnyitni, és update-elni. Ezt a megoldást a TIA Portal környezetben a MultiuserServer beállításával- lehet használni. Ennek az a lényege, hogy a projekteket egy szerveren tároljuk, amihez a megfelelő beállítások mellett a TIA Portal-ból meg tudunk nyitni. Így, ha csatlakozunk a hálózatra, amelyen a szerver fut, akkor a világ bármely pontjáról meg tudjuk nyitni a szükséges projekteket. Szükség esetén a projektek több különféle verzióját is tárolhatjuk a szerveren, így akár össze is lehet venni a különféle módosításokat, és a működő back-up rendszer segítségével bármikor visszaállítható a PLC programja egy korábbi mentett verzióra. Így akár megoldható az is, hogy egyetlen PC-t használunk, ami kezeli a gyártósor összes programját (kamera programok, adatbázis kezelő, szervók programjai, PLC fejlesztő környezet), és ennek a PC-nek biztosítunk hozzáférést a programozó szoftver Licenszeihez és a MultiuserServer-hez, amit így akár a belső hálózatra csatlakozva bármilyen PC segítségével el tudunk érni, ha az a gép benne van a cég által engedélyezett gépek listájában, vagy VPN segítségével biztosított a hozzáférése a belső hálózathoz.

Egy plusz funkciót jelenthet a FlyingChangeover nevű típusváltási metódus. Ennek az az előnye, hogy egy adott típusváltás esetén nem szükséges megvárni, míg a gyártósorunk leürül. A folyamatot úgy kell kialakítani, hogy amikor felkerül az utolsó termék a leváltásra kívánt típusból, akkor a modulokat és a modulcsoportokat egyesével haladva át lehet állítani a kívánt új típusra, és így jóval kisebbre lehet állítani

Új elméleti és gyakorlati ismeretek átadásához köthető javaslatok

Az egyik ilyen téma az iparban használt kamerákhoz köthető. Egy PLC programozó a munkája során szinte biztosan fog olyan képfeldolgozó technológiát használni, ahol vagy alakzatfelismerést, vagy szín felismerést, vagy bármi ezekhez hasonlatos termék ellenőrző vagy válogató folyamat során ipari kamerákat és a hozzá tartozó szoftvereket megismerje. Ez persze nem azt jelenti, hogy minden egyes kameraszoftvert teljes részletességgel kellene oktatni, de legalább egy egyszerűbb szoftvert ismertetni kellene a hallgatókkal, ahol megismerhetik azt, hogy egy kép kiértékelését milyen beállítások alkalmazásával végzi el a program, milyen módon lehet egy terméken lévő DMC-ből, vagy bármilyen más vonalkódból a PLC számára továbbítani az adatot.

Ezek mellett szükség lenne arra, hogy a PLC programozást végző tanulók legalább alap szinten megismerkedjenek az ipari robotokkal. Ezen belül a robotok programjának alap szerkezetére, és a robotok biztonságos kézi mozgatásának elsajátítását kellene megismerni a hallgatóknak. Erre azért van szükség, mert PLC szintjén a robot csak úgy van kezelve, mint egy egyszerű aktuátor: kiadjuk az adott kimenetre az utasítást, ami a robot programja alapján meghív egy job-ot, ami egy előre beállított mozgássorozatot és szerszámkezelést jelent. Mivel a hibák egy része adódhat a robot mozgásából, így a hallgatónak ismernie kell, hogy a robot milyen módon tud mozogni, és hogy milyen esetekben lehet a robot hibás működését okozó hiba PLC oldalról, és mikor lehet a robot programja által.

Tapasztalatok összefoglalása

Az oktatásból a munkaerő piacra kikerülő diákok tudását az jellemzi, hogy a PLC programozási ismeretek bizonyos alapjait elsajátítják, egyszerűbb programokat meg tudnak írni, viszont a komplexebb feladatok, hibakeresések és külső perifériák kezelését érintő területeken az alapok hiányoznak. A programozás alap szabályait csak néhányan értik meg, mivel nem minden esetben van erre megfelelő hangsúly fektetve a technikusi szinten az elméleti oktatás és a gyakorlati tudás átadása közben. A mérnöki oktatásban a tapasztalataim szerint a problémák viszont főképp az óraszámokból adódnak. Az oktatóknak nem áll rendelkezésére megfelelő mennyiségű idő arra, hogy a szükséges ismereteket megfelelően átadják, és vannak olyan szituációk, amikor a különféle ismeretek plusz tantárgyak beiktatását igényelnék az oktatási rendszerbe. A tanórák során például az SCL programozási nyelv megismerésére csak kevés idő áll rendelkezésre, és így a hallgatók nem minden esetben tudják kitapasztalni annak előnyeit.

A másik, amit be kellene iktatni a gyakorlat orientált elméleti ismeretek közé, az a hibakeresés. Ez amiatt lényeges, mert a diákok javarészt csak olyan hibákkal ismerkednek meg a tanulási folyamat során, ami szintaktikából ered, vagy mivel a saját maguk által szerkesztett kódot kell javítani, így könnyen megtalálják a hibát. Viszont egy más logikájú, más gondolkodású programozó által írt kódban már nem ilyen egyszerű hibát keresni. És néhány ilyen hibakeresési folyamat után tudatosul a diákokban az is, hogy miért érdemes a kommentezést használni, miért érdemes a változóneveket céltudatosan megválasztani.

Fejlődést jelenthetne ezen a téren, ha a duális képzés nem egy szűk kör számára elérhető opció lenne, hanem egy általános lehetőség. Jelen helyzetben az egyetemekről csak azok a diákok juthatnak duális képzésre, akik jó vagy kiemelkedő eredményekkel rendelkeznek. Ezzel az a probléma kerül előtérbe, hogy pont azok a hallgatók nem kerülnek be egy részletes gyakorlati képzési formába, akiknek fokozottan szüksége lenne egy olyan környezetre, ahol az iparban aktívan résztvevő „mentoroktól” tudnák megtanulni a szakma azon részeit, ami a képzési időbe nem fér bele. A jelenlegi rendszer helyett egy olyan rendszer érhetne el hatásosabb eredményeket, ahol az oktatási intézmény szakirányonként legalább 2-3 olyan céggel állna szerződésben, akik képesek befogadni azt a tanulókapacitást, amit egy-egy szakirány nyújtani tud. Ez előnyös lehet a cégek számára is, mivel úgy kapnának munkaerőt a piacról, hogy a frissen végzett hallgató már valamennyire ismerné a céges struktúrát, lenne olyan jellegű tapasztalata, amit a cégnél aktívan használnia kell, és az egyetem is olyan hallgatókat tudna így a piacra engedni, akiknek a szakmai tudása és tapasztalata már a junior szint felső határához közelít, ezáltal a hallgatók kezdő piaci értéke is magasabb. Ez még ha magasabb kezdő fizetést is jelentene a hallgatóknak, a felvevő piac akkor is gyorsabban alkalmazná őket, mert az aktív szakmai tapasztalat miatt a cégeknek kevesebbet kellene költeni a belső képzésekre, amelyeknek darabja többszáz vagy ezer eurós tétel is lehet.

Az előzőekben tárgyalt fejlesztési lehetőségekkel, és azokkal az alapismeretekkel, amiket az általam kidolgozott programok során bemutattam egy olyan szemléletet és olyan tulajdonságokat lehet a hallgatók számára átadni, amikkel a munkaerőpiacon kellő magabiztossággal tudják a tudásukat tovább fejleszteni, könnyebben érnek el sikerélményeket és nagyobb ütemben tudnak fejlődni a PLC programozással kapcsolatos szakmai pályájukon. Az oktatás általános álláspontja az, hogy az alapokat kell megfelelően elsajátítani, és a többi tudást megszerzik majd a szakmai pályájuk során a diákok. Viszont ezeket az alapokat érdemes tovább fejleszteni az oktatás terén

is, mivel a technológia fejlődése, és az ipar új vívmányai már másabb szemléletet igényelnek, mint amire például néhány évvel ezelőtt még könnyen tudtak támaszkodni a hallgatók. Egy megfelelően kialakított, célirányos tantárgystruktúrával ez hatékonyabbá tehető, mivel a tanulók a szakma/szakirányválasztás után jelentős százalékban a saját szakterületükkel fognak foglalkozni, és az ehhez szükséges ismereteket próbálják gyarapítani, elmélyíteni. Szükséges, hogy a diákok rendelkezzenek egy a pályát jellemző alapvető ismeretrendszerrel, mint például, amit a szakmai alapozó tárgyak adnak, viszont érdemes lenne akár a képzési idő bővítésével olyan új ismereteket átadni, amelyek már a gyakorlat orientáltság egy olyan szintjét mutatják meg, ami megfelelő bevezetés lehet számukra az iparba.

Abstract

The topic of my thesis was about the education of PLC programmers in the technicians level and the engineer level. In my thesis I mostly focused on the differences between the knowledge basis that the students have when they go for the labor market and what informations or knowledge the do not get which would be necessary when the go for work to a company.

I used my experiences from the technician level education and the engineer level too, and as a countermeasure I compaered it with the knowledge which I from my company and other sources such as the training provided by Festo Didactic.

For support my theory I created a PLC program for a production system which has five modules in it. I used the TIA Portal for creating the PLC program and the function blocks and I used that software for the HMI screens develoment as well. In hardware level I used the MPS system of Festo Didactic, and for the controlling I used two Siemens PLC. While I created my function blocks I was focused on a few topics which should be the part of the basis knowledge of a PLC programmer, such as the use of the proper segmentation of the PLC programs, the creation of a usable and clean HMI screens, the proper commenting of a PLC code, show the difference of using SCL instead of LAD and so on. Some of these principles are provided fo the students from their teachers, but for some topics there are not enough time or some of the topics do not got the proper focus. For example in different programming courses the students learn how to give proper names for the variables, so anyone who knows the language which was the program written in, they can easily read the code, and they learn on those courses how to use the comment system properly.

At the and I had a few idea how I could improve my program to show some more important topics and features about a production system, and some assumptions about changing the education system. One of this assumptions was about finding the failures, and teaching some methods for it. For example the teachers should show a more difficult program, and a working system which has failures, and the students should search for the rootcause, or if they should try to find the programming mistakes in each others program. Another main topic is the dual training system. Most of the stundents only have the possibility to take part in a dual training system if they have very good marks, so the students with less talent can only be part of a real production system when they finish their studies. This should be changed, because the most development in a students basis knowledge can come from the excercises and the experience they can collect from the industry. I realised that if we would like to achive the Industry 4.0 in a worldwide level then we should develop or educational systems as well. Some of the universities are on a way to do this, but we need to achive this in our technician level educations as well.

At the end of this thesis I could find some topics too that I should change while I create or analyse a program or a production system, and I found some topics which I should train myself too. I refreshed some of the base knowledge wich we used in school projects, and found some more possibilities, that the new generation can learn by programming to these systems.

Ábrajegyzék

Forrásjegyzék

[1.] MPS állomás általános ismertető: https://www.festo.com/net/supportportal/files/10142/mps.pdf
Elérés:2023.11.24.

[2.] fisertechnik Ipar 4.0 gyakorló modul ismertető: https://www.fischertechnik.de/en/products/industry-and-universities/training-models/554868-training-factory-industry-4-0-24v
Elérés:2023.11.24.

[3.] MPS állomás kocsira szerelt kivitel fotója:
https://twitter.com/reletec/status/31365785126243232/photo/1
Elérés:2023.11.24.

[4.] MPS állomás profilra szerelt kivitel fotója: https://www.researchgate.net/figure/Festo-MPS-workstation_fig8_282003522
Elérés:2023.11.24.

[5.] PLC programozás az IEC1131-3 szabvány szerint, Jancskárné Anweiler Ildikó (főiskolai docens PTE – PMMFK Műszaki Informatika Tanszék) kézirata, https://rievtech.info/konyvek/Jancskarne-Anweiler-Ildiko-PLC-programozas-az-IEC1131-3-szabvany-szerint.pdf
Elérés:2023.11.28.

[6.] MPS állomás: Electric_Circuit_Diagram_Distribution, Festo Didactic GmBH.
Elérés:2003.08.28.

[7.] MPS állomás: Pneumatic_Circuit_Diagram_Distribution, Festo Didactic GmBH.
Elérés:2003.08.28.

[8.] MPS állomás: Electric_Circuit_Diagram_Testing, Festo Didactic GmBH.
Elérés:2003.08.28.

[9.] MPS állomás: Pneumatic_Circuit_Diagram_Testing, Festo Didactic GmBH.
Elérés:2003.08.28.

[10.] MPS állomás: Electric_Circuit_Diagram_Processing, Festo Didactic GmBH.
Elérés:2003.08.28.

[11.] MPS állomás: Electric_Circuit_Diagram_Handling, Festo Didactic GmBH.
Elérés:2003.08.28.

[12.] MPS állomás: Pneumatic_Circuit_Diagram_Handling, Festo Didactic GmBH.
Elérés:2003.08.28.

[13.] MPS állomás: Electric_Circuit_Diagram_Sorting, Festo Didactic GmBH.
Elérés:2003.08.28.

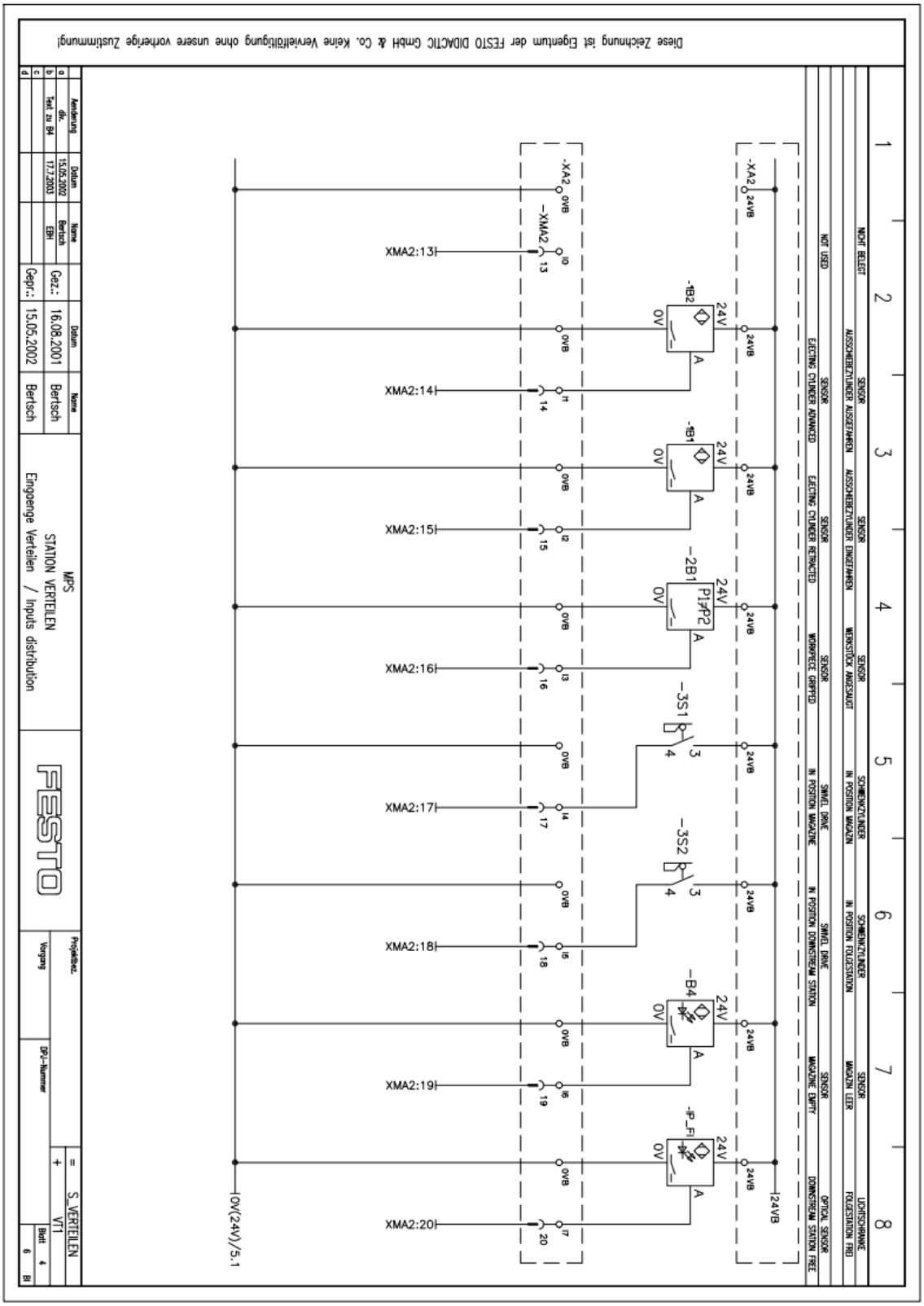
[14.] MPS állomás: Pneumatic_Circuit_Diagram_Sorting, Festo Didactic GmBH.
Elérés:2003.08.28.

Mellékletek

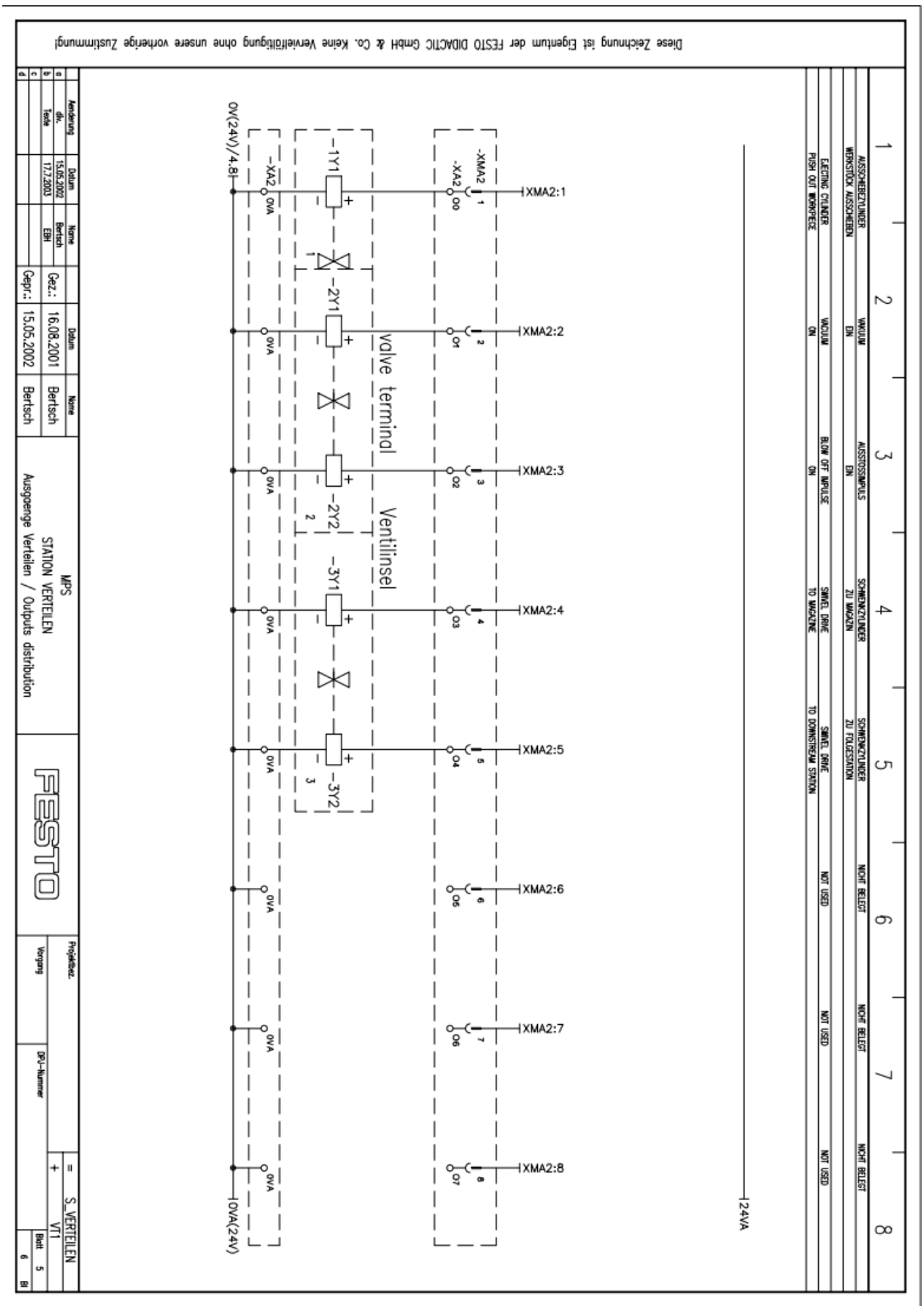
Az alábbi kapcsolási rajzok az MPS állomásokhoz biztosított dokumentáció részét képzik.

A dokumentációhoz tartozó források a forrásjegyzék végén találhatóak [6] - számmal kezdődően.

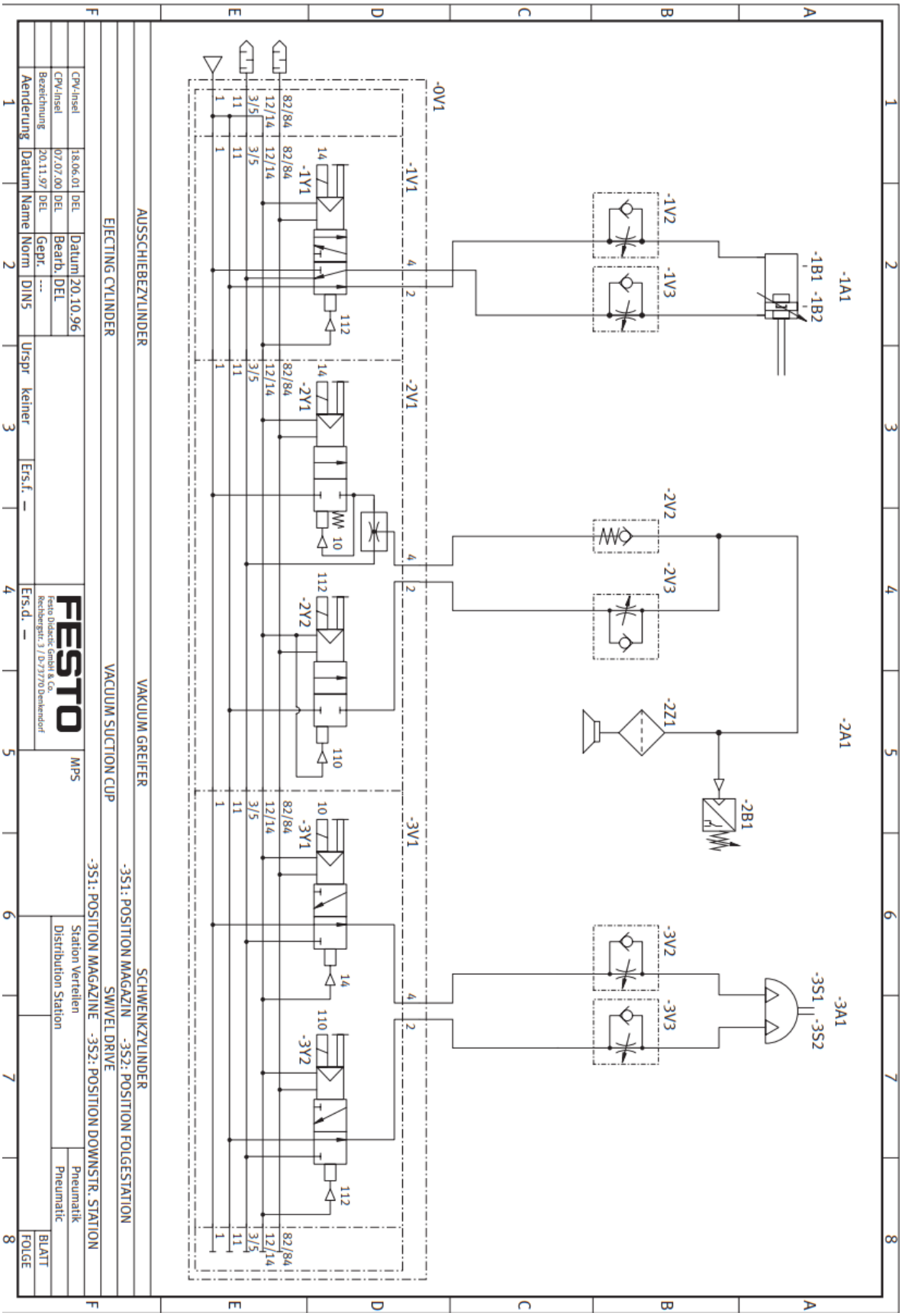
A modulok elektronikus és pneumatikus kapcsolási rajzai



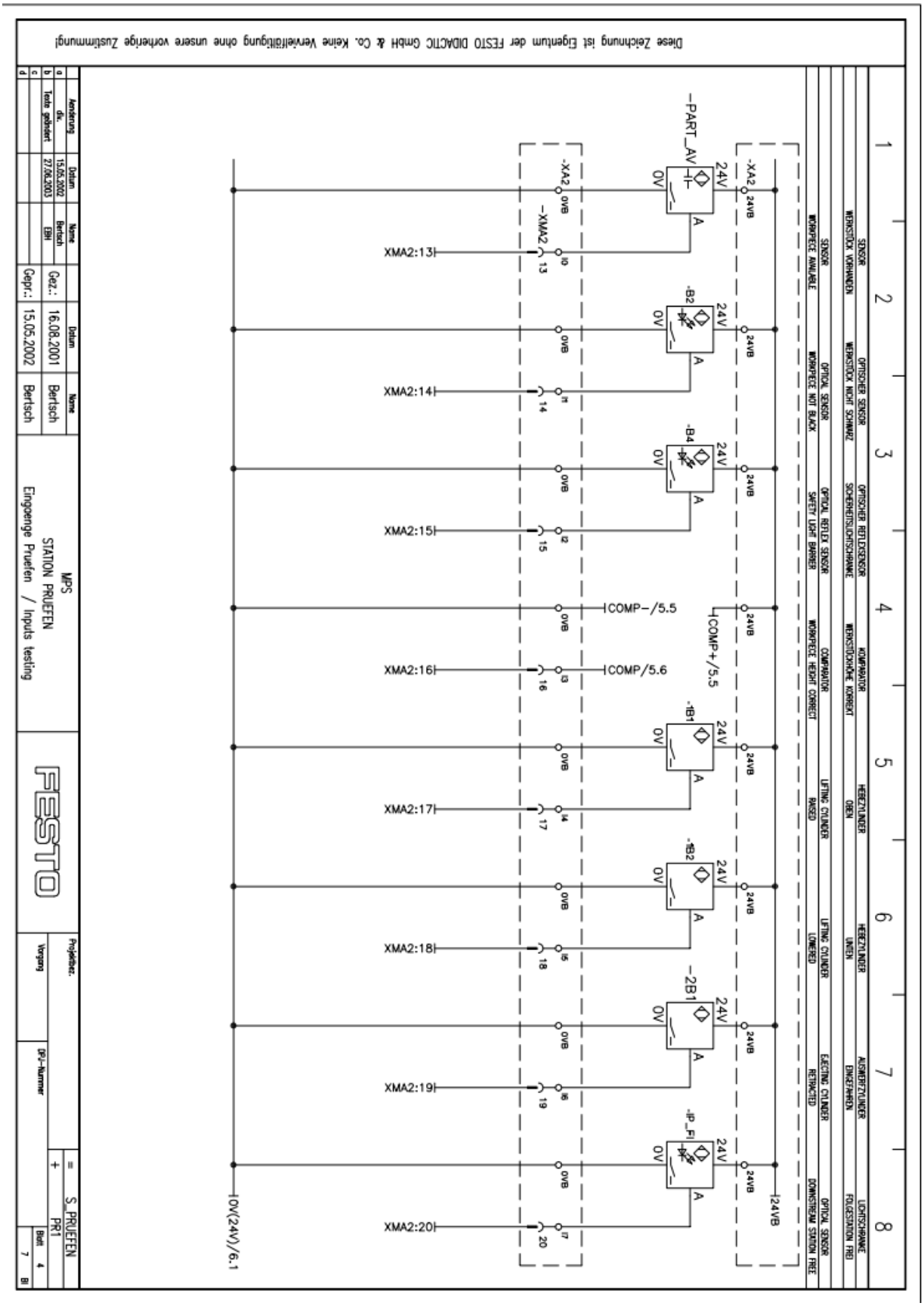
31. ábra Distribution modul - Bemenetek elektromos kapcsolási rajza [6.]



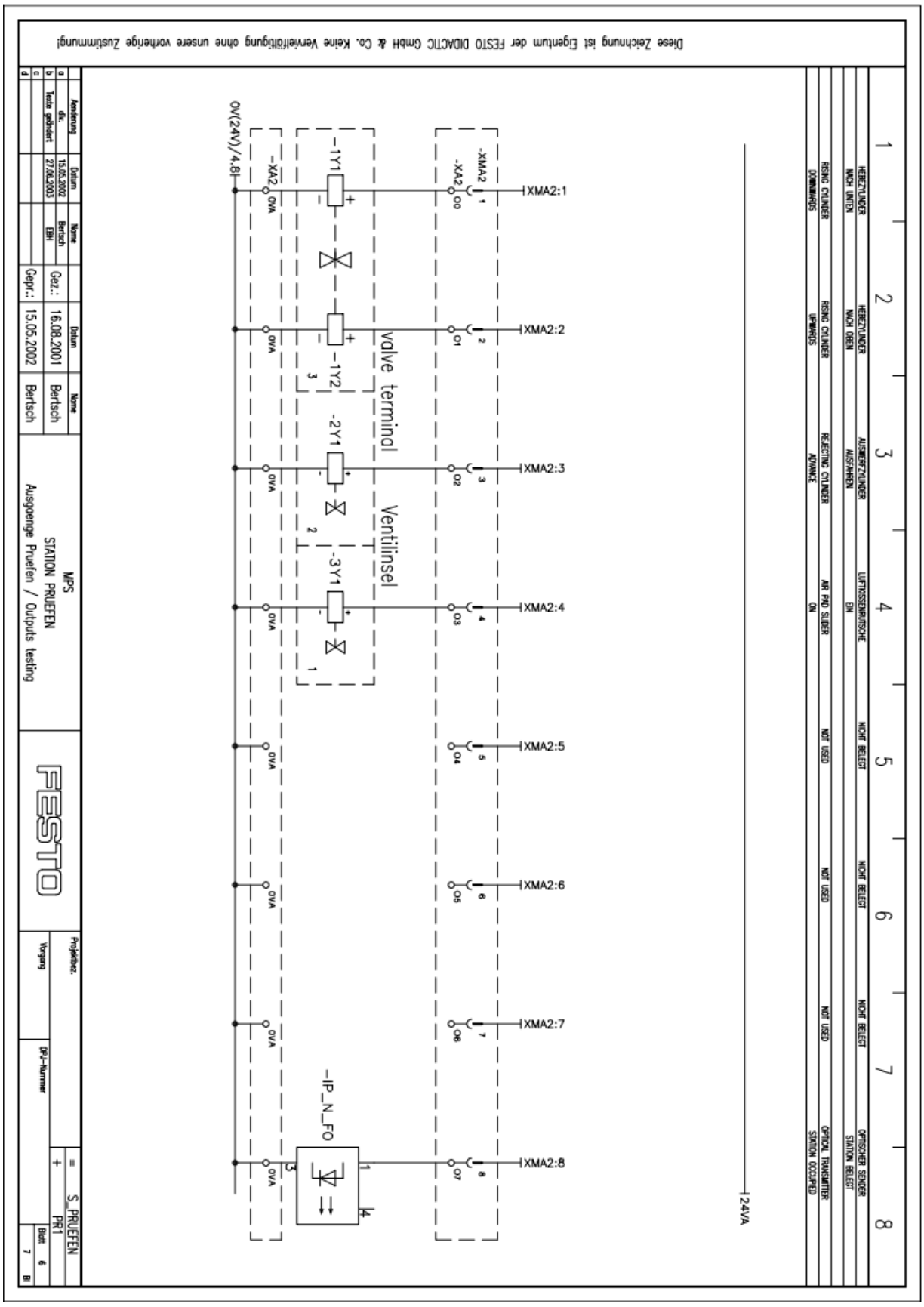
32. ábra Distribution modul - Kimenetek elekrtomos kapcsolási rajza [6.]



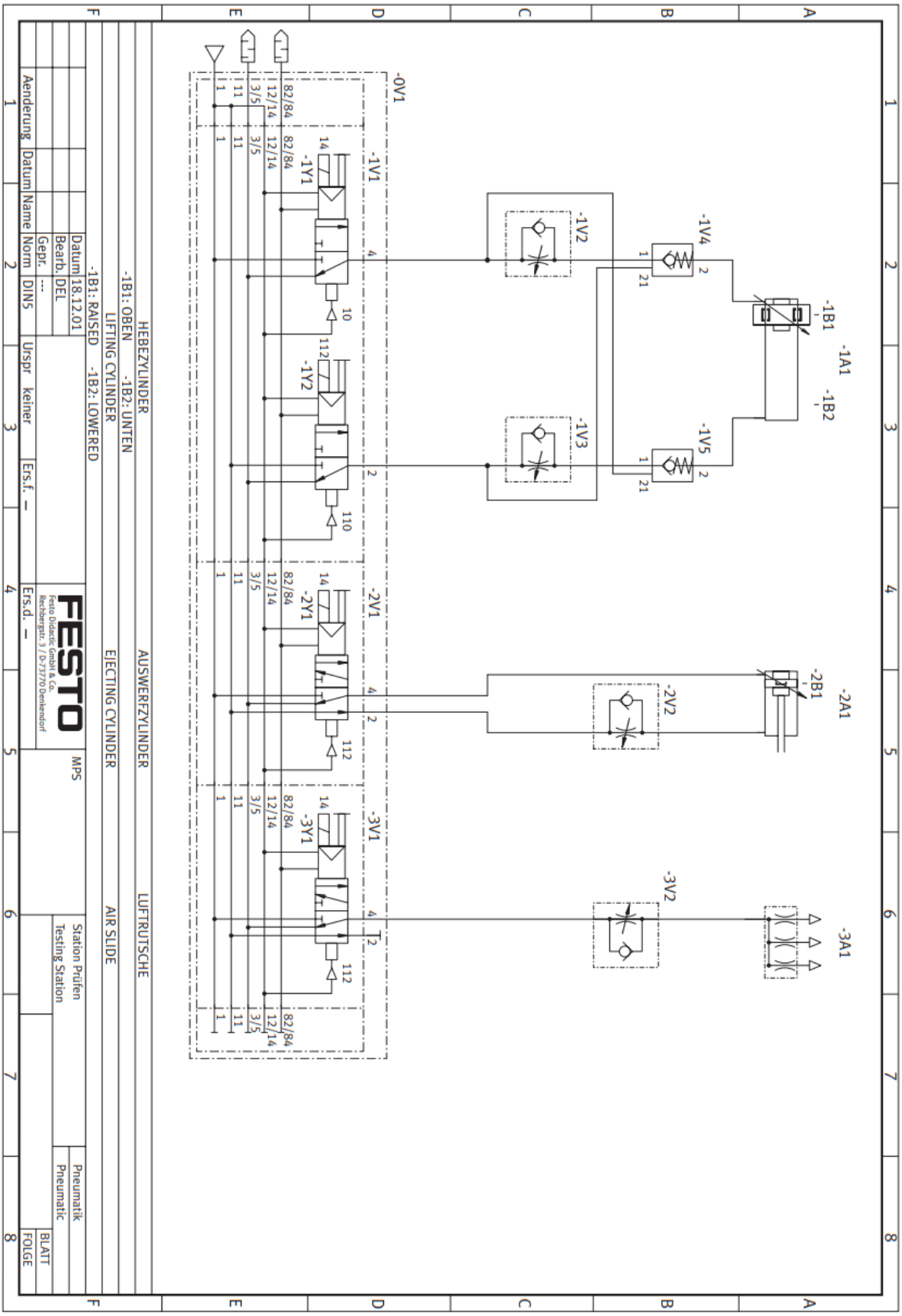
33. ábra Distribution modul - Pneumatikus kapcsolási rajz [7.]



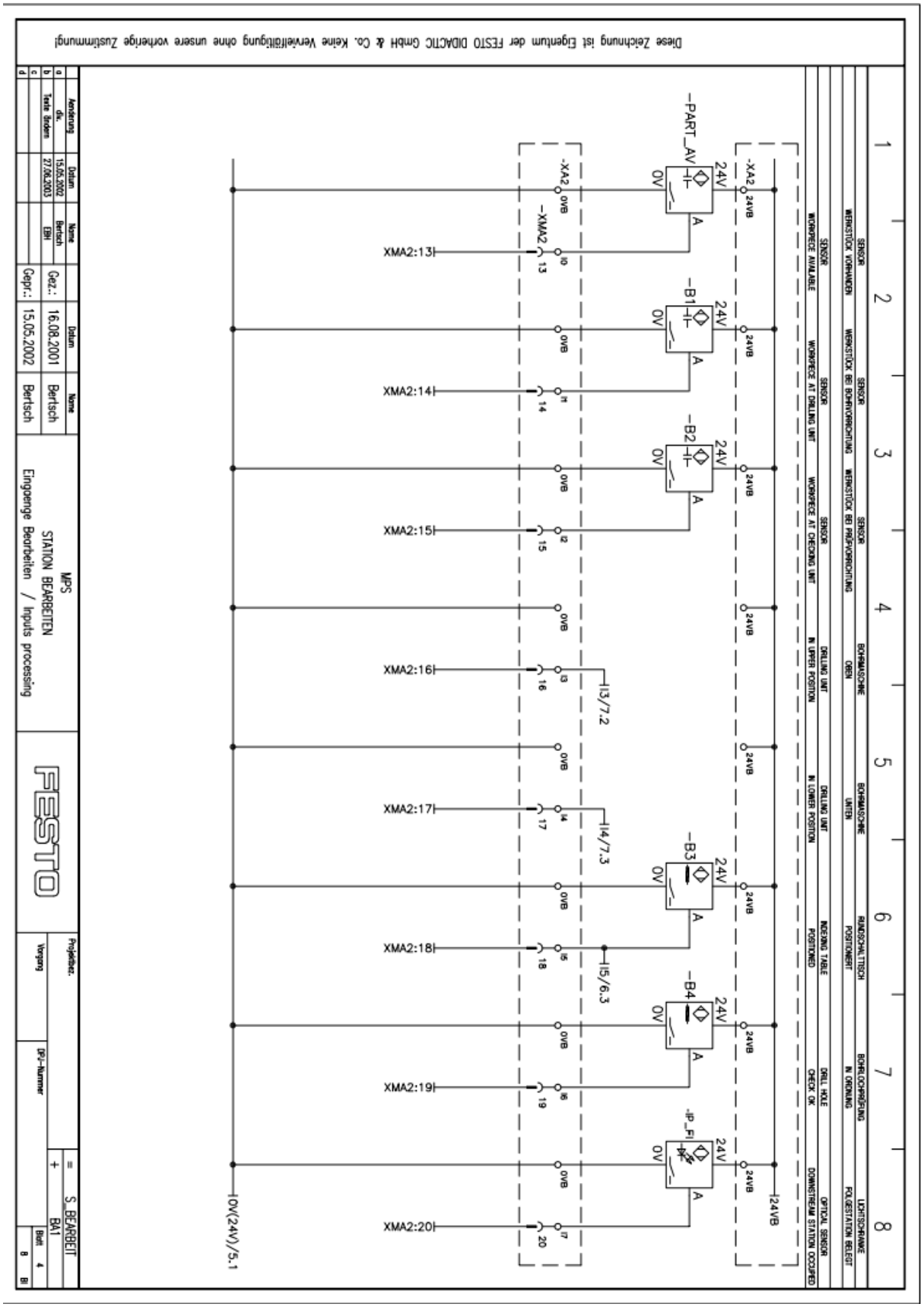
34. ábra Testing modul - Bemenetek elektromos kapcsolási rajza [8.]



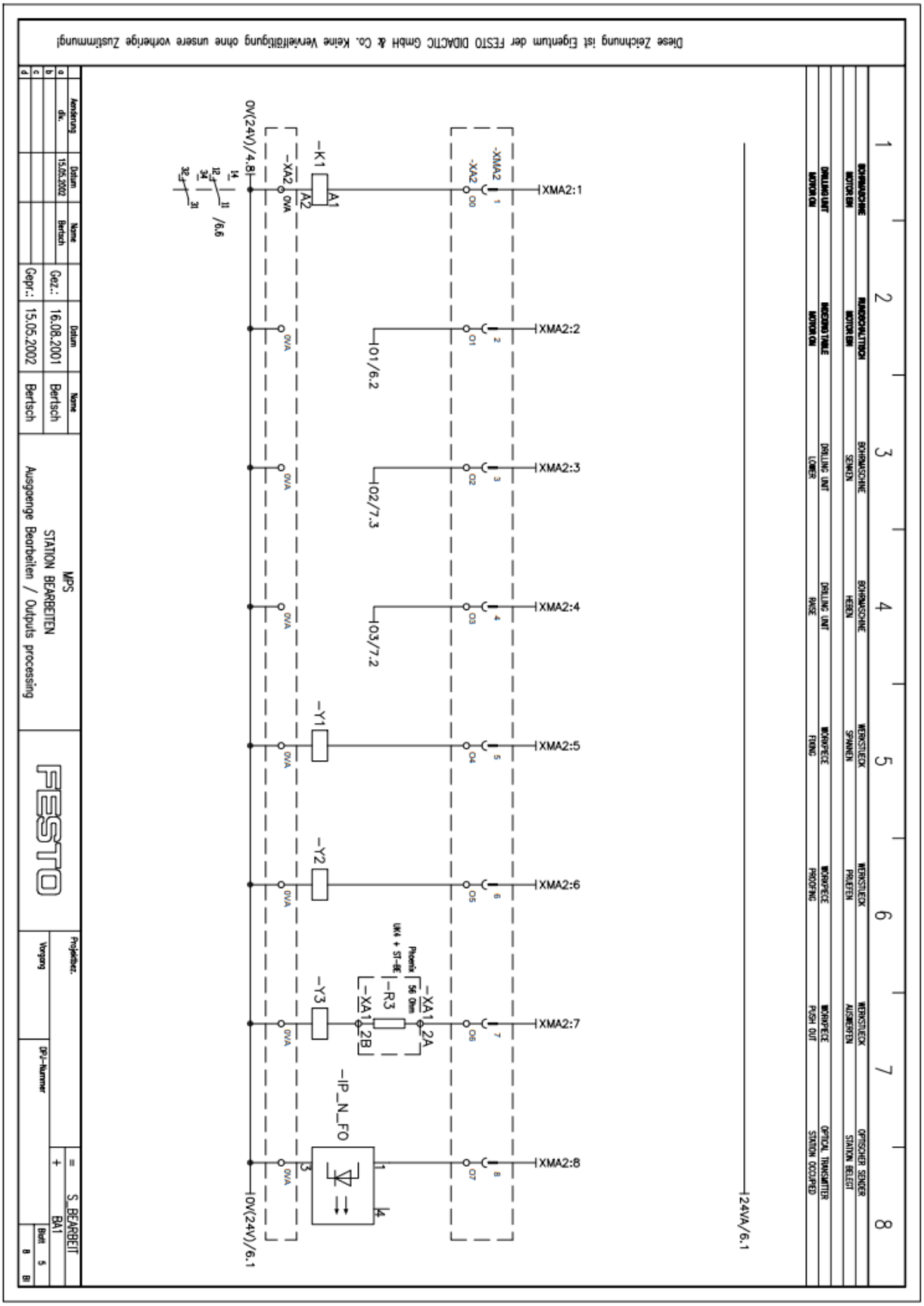
35. ábra Testing modul - Kimenetek elektromos kapcsolási rajza [8.]



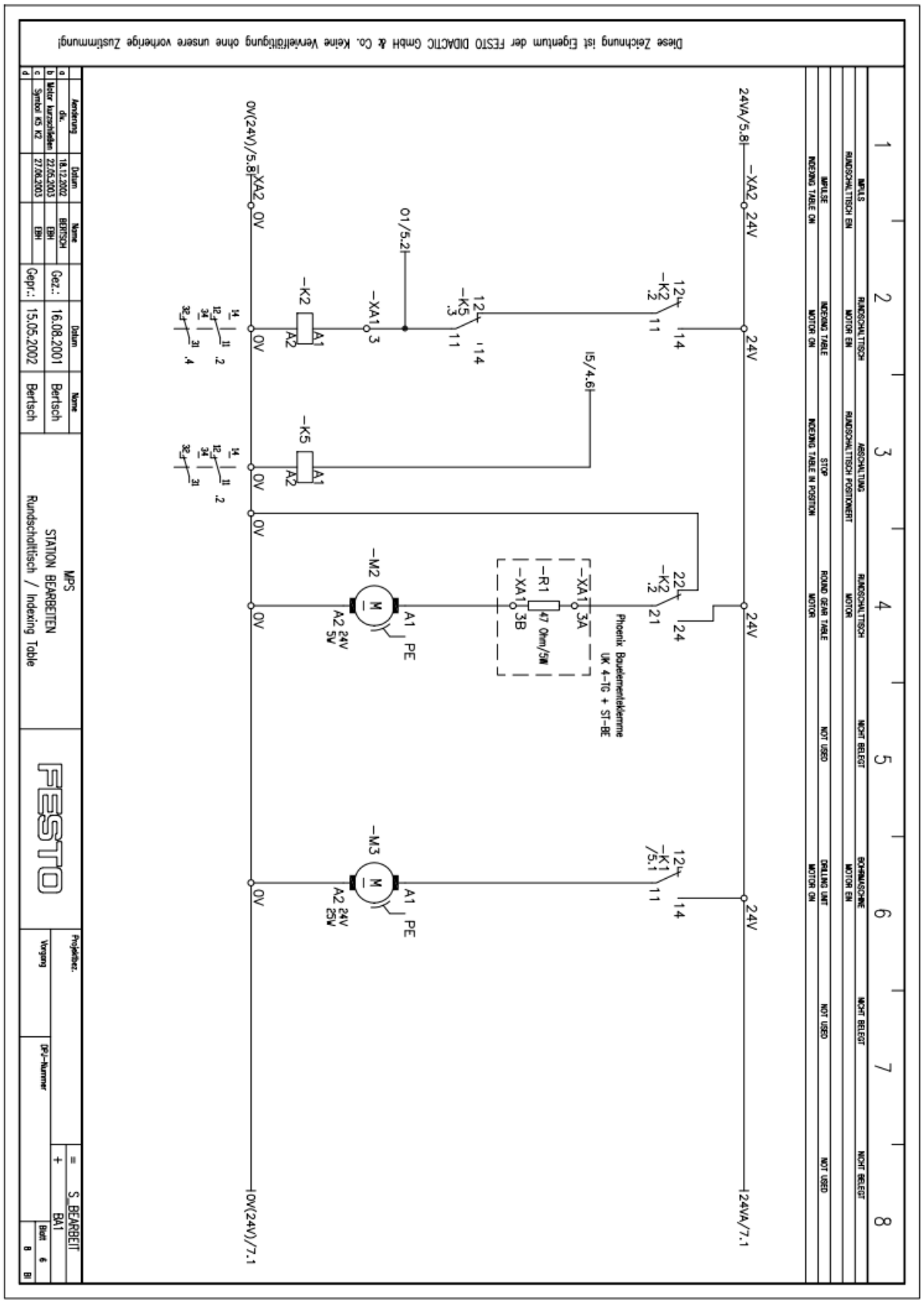
36. ábra Testing modul - Pneumatikus kapcsolási rajz [9.]



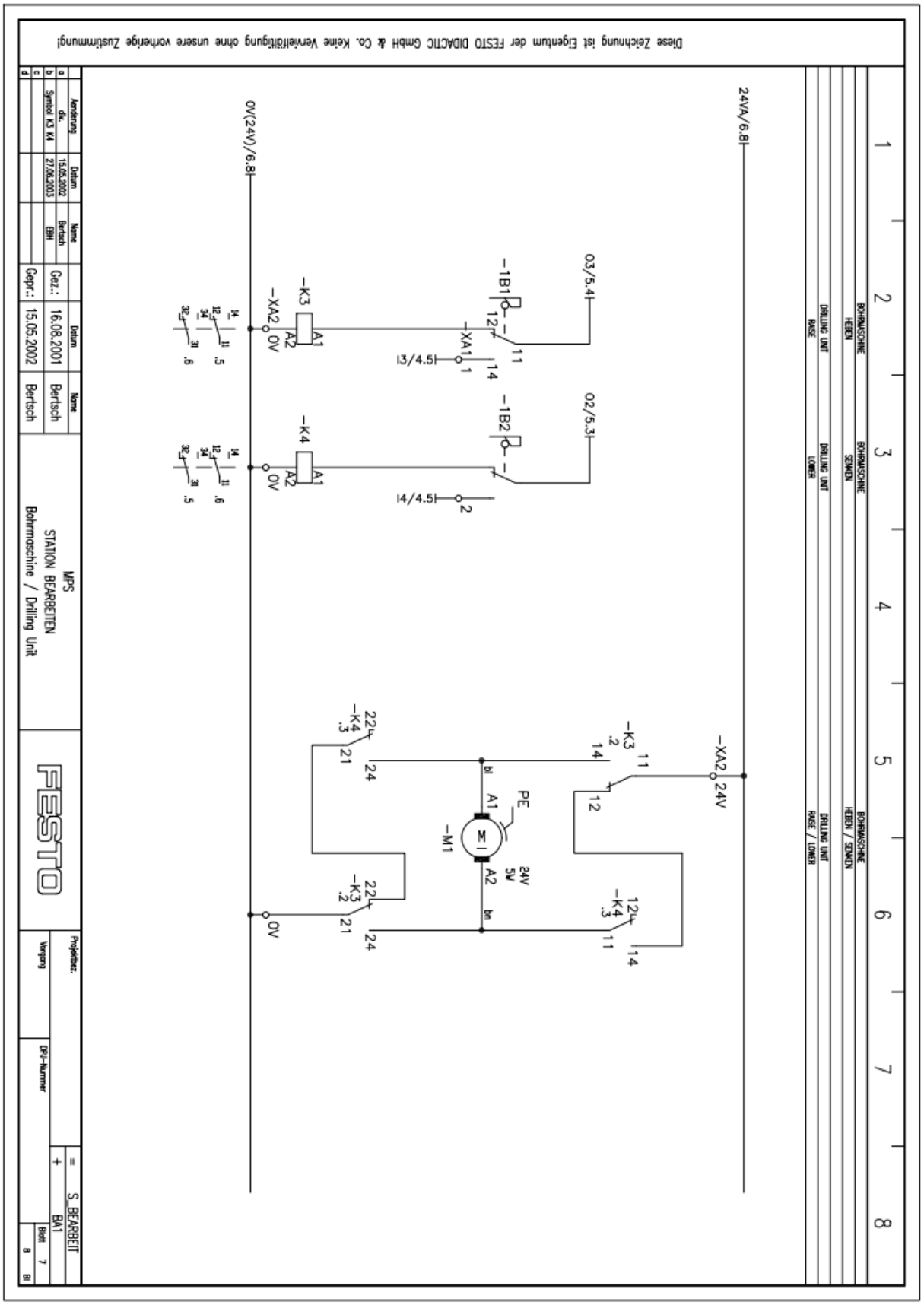
37. ábra Processing modul - Bemenetek elektromos kapcsolási rajza [10.]



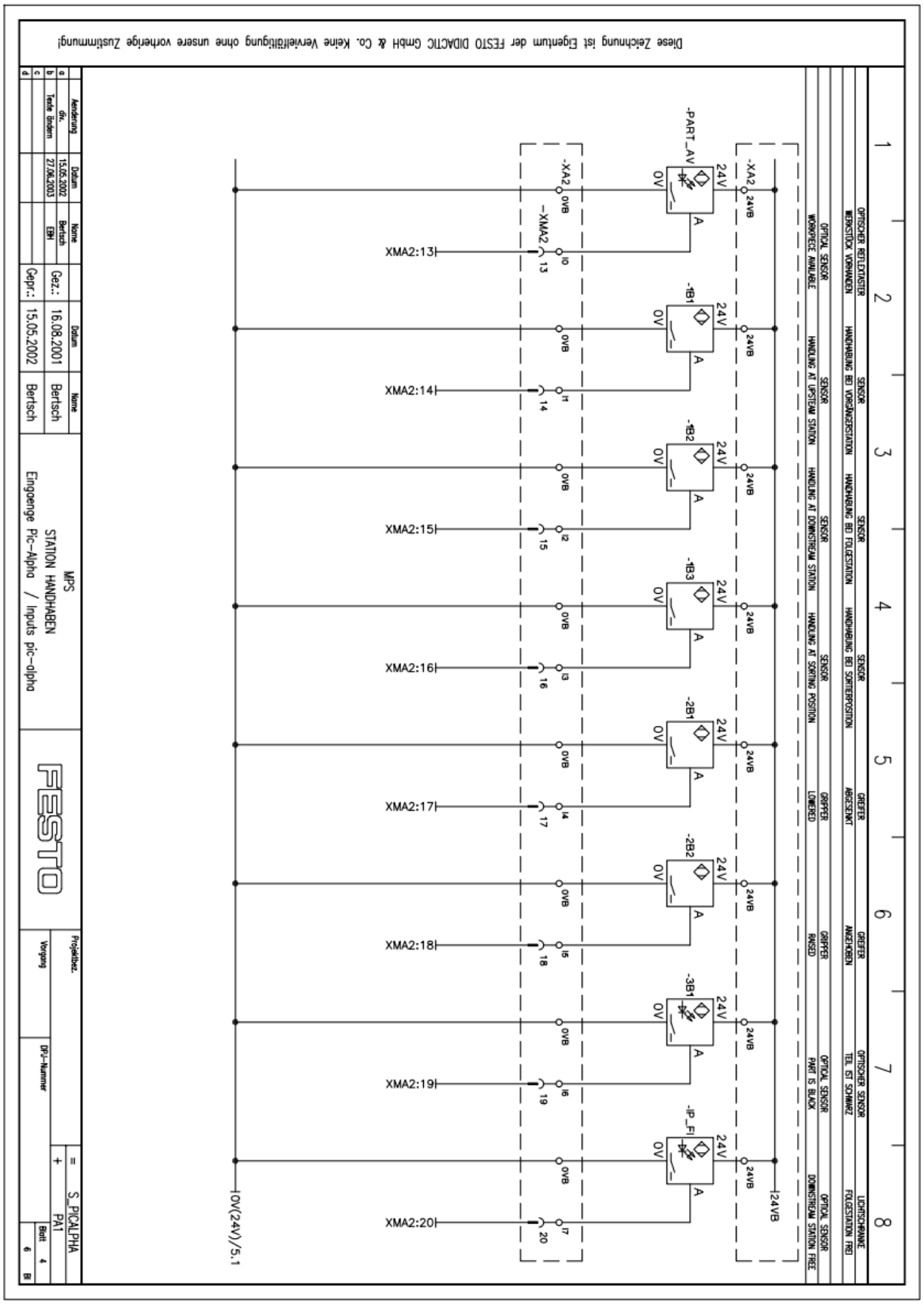
38. ábra Processing modul - Kimenetek elektromos kapcsolási rajza [10.]



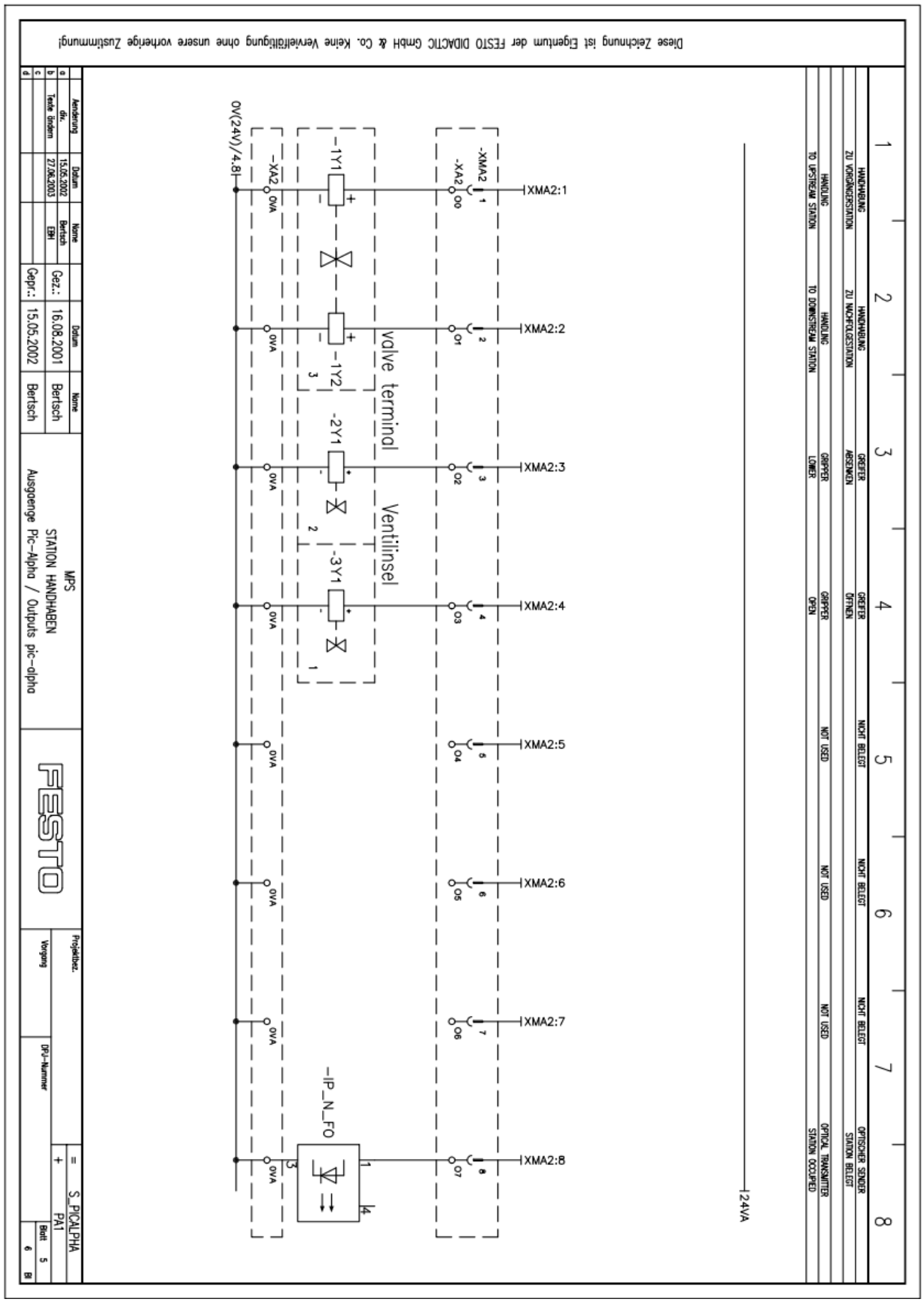
39. ábra Processing modul - Fúró egység kapcsolási rajza [10.]



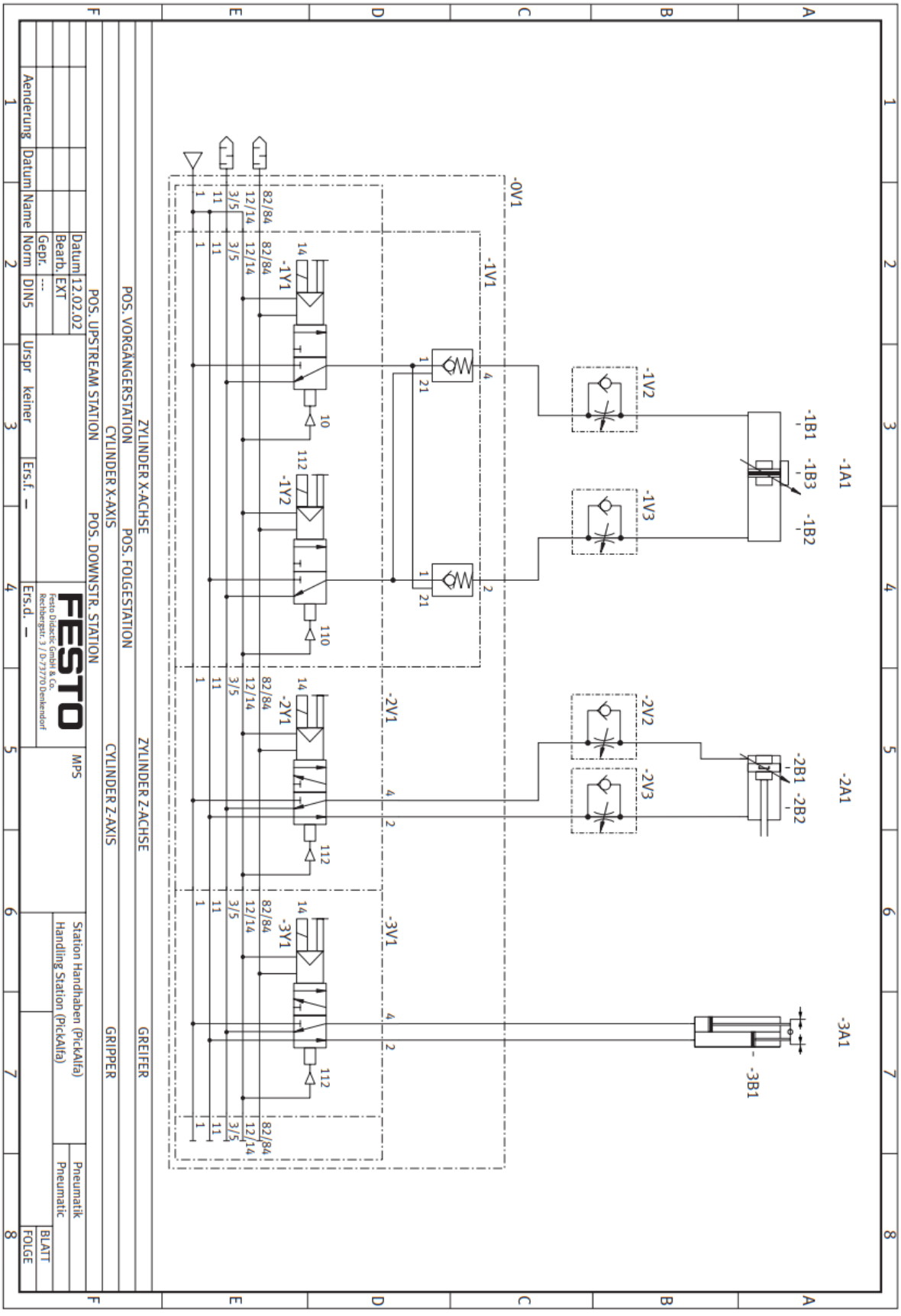
40. ábra Processing modul - Körasztal vezérlésének kapcsolási rajza [10.]



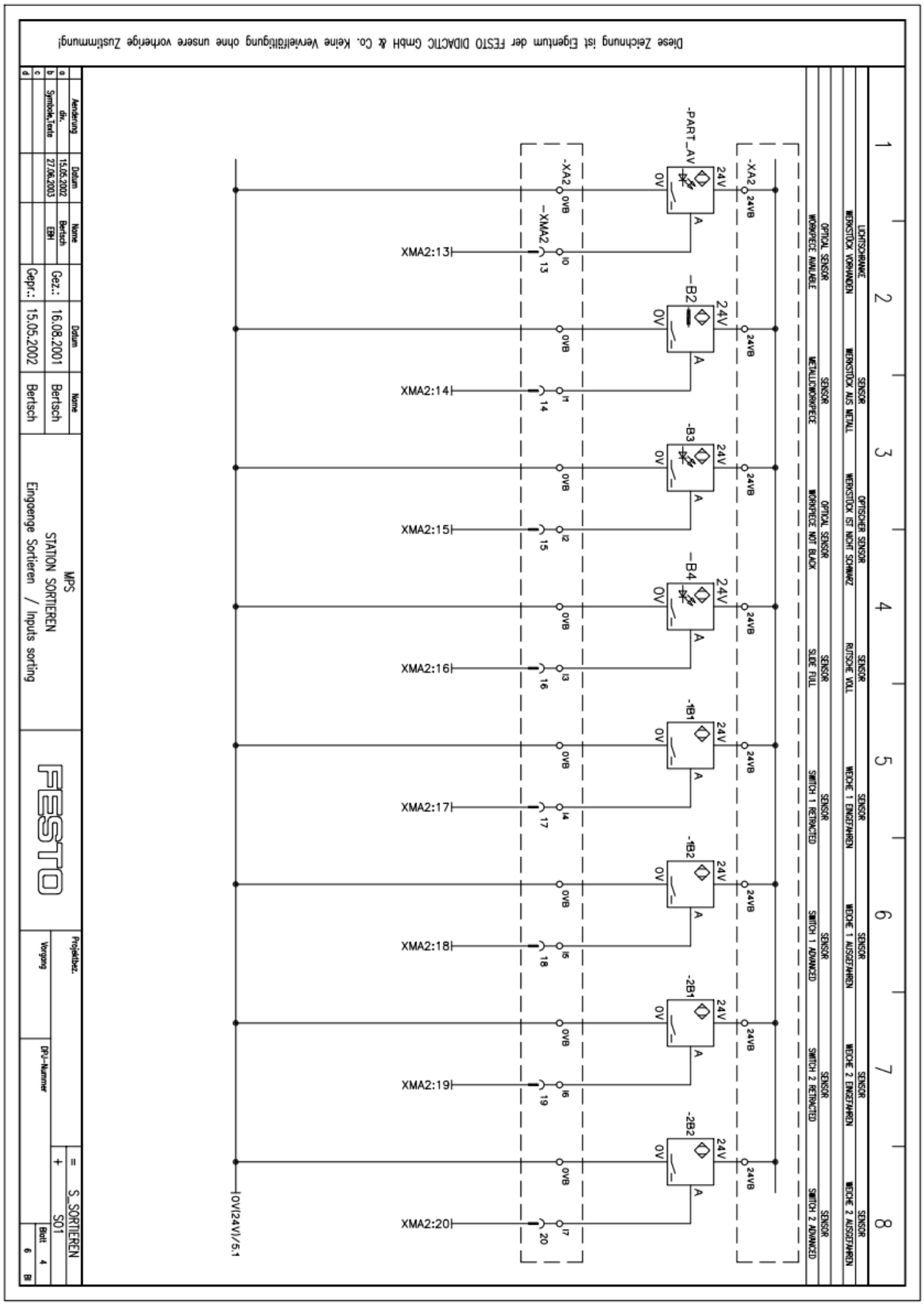
41. ábra Handling modul - Bemenetek elektromos kapcsolási rajza [11.]



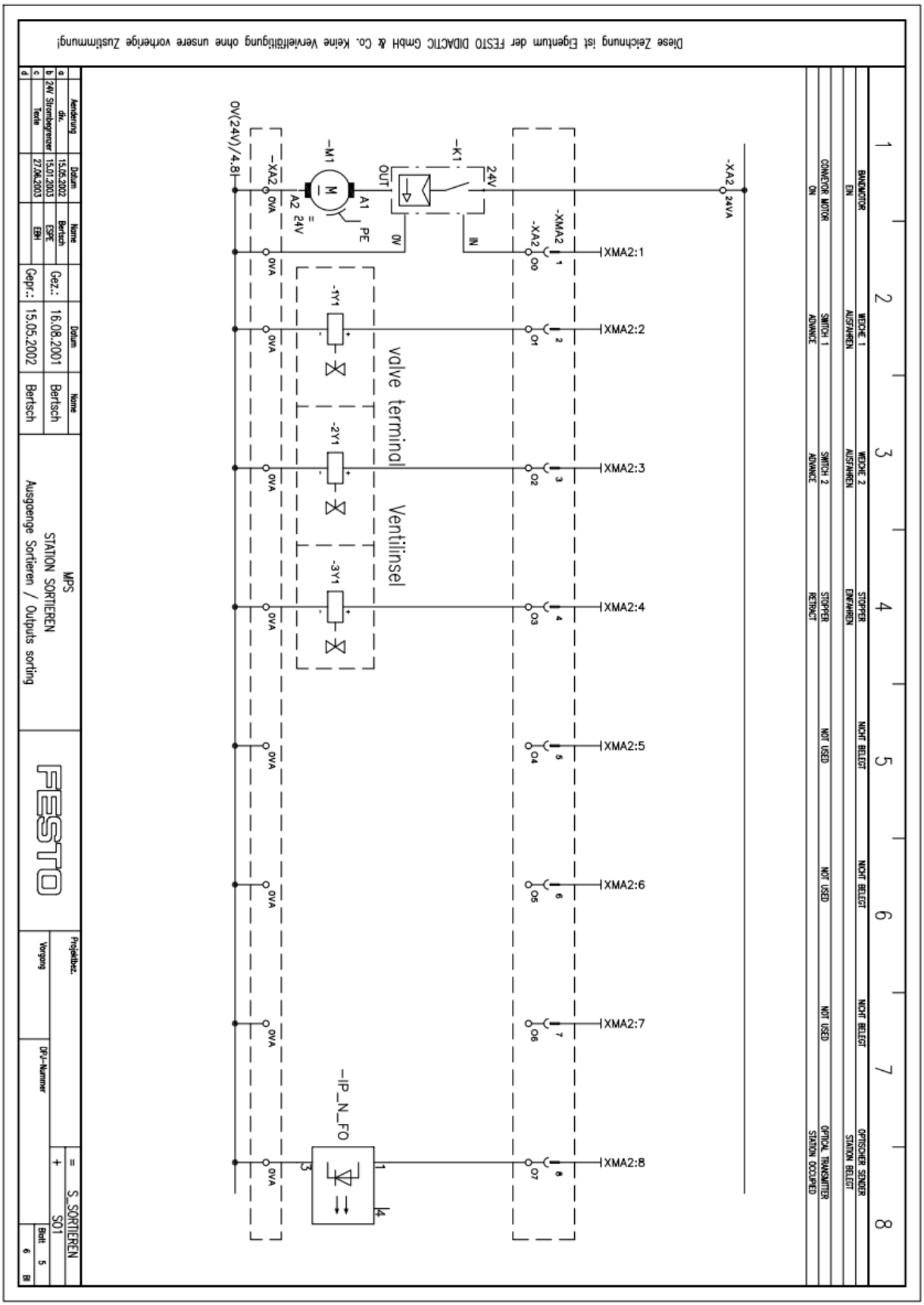
42. ábra Handling modul - Kimenetek elektromos kapcsolási rajza [11.]



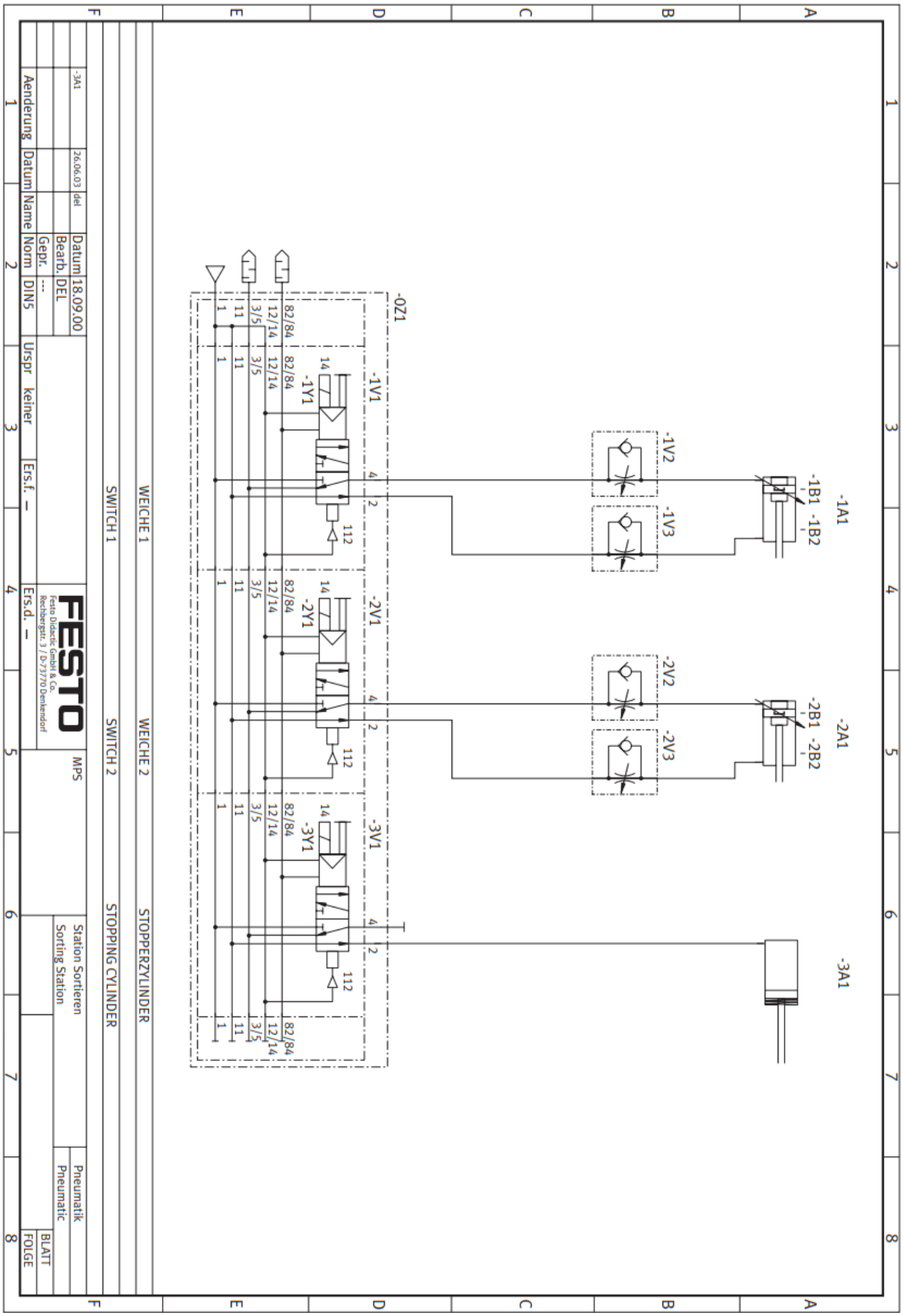
43. ábra Handling modul - Pneumatikus kapcsolási rajz [12.]



44. ábra Sorting modul - Bementek elektromos kapcsolási rajza [13.]



45. ábra Sorting modul - Kimenetek elektronikus kapcsolási rajza [13.]



46. ábra Sorting modul - Pneumatikus kapcsolási rajz [14.]