



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Csábrádi Attila

DEMONSTRÁCIÓS GYÁRTÓSOR VEZÉRLÉSE ÉS FEJLESZTÉSE

KONZULENS

Oláh István

BUDAPEST, 2016

HALLGATÓI NYILATKOZAT

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

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

Kelt: Budapest, 2016. 11. 01.

.....
Csábrádi Attila

Összefoglaló

Eme szakdolgozat egy ipari gyártósor-modell szoftveres hátterének fejlesztését követi végig. A munka a következő területeket fogja érinteni: Először megismerkedünk a gyártósor hardveres felépítésével, elsősorban a vezérlésért felelős Siemens gyártmányú programozható logikai egységgel (PLC), az azt támogató modulokkal (IO-modulok, gyorsszámlálók), a feldolgozóegységek és az azok működését segítő szenzorok működésével. Ezt követően a szakdolgozat ismerteti a választott programozási nyelvet és a választás okát, a PLC működési paradigmáit, a feladathoz választott vezérlési modellt. A szükséges alapok letétele után a dokumentum végigveszi a vezérlő szoftver tervezését és implementálását. Ezután azokat a szoftverblokkokat hozzuk létre, melyek a nem várt működéseket képesek kezelni. Végezetül a gyártósori folyamatokat egy HMI-panelen (human machine interface: ember gép kapcsolat) vizualizáljuk, melyen visszajelzéseket kaphatunk felmerülő hibákról, és a folyamatok egyes jellemzőit is módosíthatjuk. A szakdolgozat érinti azokat a lépéseket, amelyek egy valós gyártóegység üzembe helyezésénél is szükségesek lennének.

Abstract

This final project follows through the development of the software control of a demonstrational factory unit. The work will include the following topics: First we get to know the hardware build-up of the factory unit, such as the main unit, known as programmable logic controller (PLC) manufactured by Siemens, which is responsible for the control signals, the moduls which help the PLC to operate (IO-moduls, high-speed-counters), the operations of the processing-stations and sensors. After that the final project presents the chosen programming language and the reason of that choice, the PLC foundations, the chosen software-model. After the necessary basics the documents follows through the design and implementation of the control-software. Following, we see the creation of software-blocks which are capable to handle non-designed situations. Finally, we visualize these processes with the help of a HMI-modul that can give us feedback about the occuring faults and modify the process-properties. These steps, included by this work, are necessary in a case of installing a real factory unit.

Tartalomjegyzék

1 Bevezetés.....	v
2 A feladatkiírás részletezése	5
2.1 Alkalmazott technológia megismerése	5
2.2 Vezérlés tervezése és fejlesztése	6
2.3 Irányítási szolgáltatások	6
2.4 Tesztelés	7
3 Előzmények	8
3.1 Programozható logikai vezérlők logikája	8
3.2 A PLC fejlesztőkörnyezete	13
3.3 1500-as széria PLC	20
3.4 További hardverek	22
3.5 PROFINET (process field net)	32
3.6 Ring topológia	35
3.7 A gyártósor állomások működése és szenzorjai	38
3.8 Véges állapotú állapotgép (finite state automaton - FSA).....	45
4 A tervezés lépései	49
4.1 A présgép alapprogramja.....	49
4.2 A megmunkáló állomás alapprogramja	53
4.3 A pneumatikus állomás alapprogramja	61
4.4 A robotkar alapprogramja.....	72
4.5 A leállási procedúra	86
4.6 Receptek kezelése.....	88
4.7 Hibakezelés.....	90
4.8 Termékek követése	96
4.9 HMI	97
5 A műszaki alkotás értékelése, fejlesztési lehetőségek.....	104
6 Köszönetnyilvánítás.....	106
7 Irodalomjegyzék	107

1 Bevezetés

A bevezető fejezetben ismertetésre kerül a szakdolgozat-feladat értelmezése, az állapot, amit a feladatkiírás pontjainak teljesítése után a szoftver nyújtani tud, és az érvelés arról, hogy miért indokolt a választott téma a jelenlegi technológiai körülményeket figyelembe véve.

A munka során olyan részfeladatok vannak kifejtve, mely feladatok kötelezően elvégzendőek egy valós ipari alkalmazásban használatos gyártósor működtetésénél.

A munka során szükséges megismerkedni a különböző egységek üzemmódjaival és beépített szenzorjaival, és a feladat során alkalmazott Siemens-termékeket azon célból, hogy azok a munkát nagy hatékonysággal tudják segíteni szolgáltatásaik által.



1-1. ábra: A demonstrációs gyártósor és a vezérlőhardverek

A több felkínált programozási lehetőségek közül szükséges kiválasztani azt, mely a megoldandó problémákhoz illeszkedik, majd egy szoftver-konvenció kerül kiválasztásra, hogy a vezérlés felépítése konzisztens legyen.

A vezérlés fejlesztése során ügyelni kell arra, hogy a munkadarabok átfutása a lehető leggyorsabban történjen, emellett a modulok közti kommunikációt is figyelembe kell tartani.

A vezérlésnek olyan folyamatmonitorozásra is képesnek kell lennie, mellyel egy nem várt működést minél hamarabb detektálni lehessen, ezzel megakadályozva a gyártóegységek sérülését.

Legvégül olyan megjelenítést kell létrehozni a HMI készüléken, melyen átláthatóan minden fontos információ megtalálható.

A feladat sikeres elvégzése esetén egy olyan PLC és HMI vezérlőszoftverek jönnek létre, melyek alkalmazzák az összes funkciót, amit a modell nyújtani tud.

A vezérlésnek egy előre definiált logika alapján determinisztikusnak kell lennie. A biztonsági intézkedéseknek meg kell akadályozniuk az egységek tönkremenetelét. A vezérlőszoftvernek jól átláthatónak kell lennie, hogy egy esetleges funkcióbővítés könnyebben kivitelezhető legyen.

A szoftvernek támogatnia szükséges a procedúrákra jellemző tulajdonságok változtatását, miközben a gyártási konzisztenciát fenntartjuk.

A HMI-modul felületein egy olyan kezelőfelületet szükséges kreálni, melyen könnyen lehet navigálni, valamint folyamat-jellemzőket leolvasni.

A folyamatos igény az egyre jobb és hatékonyabb gyártási folyamatokra megkövetelték a minőségi és megbízható vezérlési technológiákat. A gyártósorok működéséről begyűjtött információk kezelésére, és ezek alapján történő beavatkozásra több technikai megoldás is létezik, melyeknek közös tulajdonságuk a mikrokontroller-alapú vezérlés. Ezen megoldások közül a programozható logikai vezérlő (programmable logic controller - PLC) emelkedik ki rugalmassága miatt, ezért ez a típus felelős az ipari alkalmazások túlnyomó többségének irányításáért.

Ez alapján a feladat témája a villamosmérnöki szaktudás egy fontos területét képviseli, név szerint az ipari automatizálását. A feladat elvégzése során megismerésre került az alkalmazott Siemens-PLC tulajdonságai, valamint az automatizálási szoftverfejlesztés praktikái.

A demonstrációs gyártósor működése betekintést ad a PLC-re épülő rendszerek rugalmasságába és működésébe oly módon, hogy azt a laikus megfigyelők is tudják értékelni, értelmezni. A vállalat, aminek a birtokában van a modell, elsősorban erre alkalmazza az egységet.

Az első rész a vezérlőegységgel fog foglalkozni, bemutatja annak működését, programozási lehetőségeit, valamint a választott nyelvet; a gyártósor-modulok felépítését és funkcióit; a használt kódolási technikát.

Ezután a fejlesztés fázisai következnek. Először prezentálódnak a különálló modulok szoftvereinek a fejlesztése, amikkel az alegységek külön-külön már működőképesek. Ezután implementálódnak a kommunikációs jelek, melyek szükségesek az összehangolt munkavégzéshez. A következő réteg már létező kódsorok refaktorálását szolgálják, ami a robot vezérlésének fejlesztését könnyítette meg. Részletezve lesznek azon funkciók, melyek a folyamatok tulajdonságainak konzisztens változtatását teszik lehetővé. A szofverfejlesztés a hibakezeléssel fog befejeződni.

A munka utolsó részében az HMI-modulon olyan felületek (screen) szerkesztődnek, melyek reprezentálják a gyártósor működését, recepteket és hibaiüzeneteket képes kezelni.

2 A feladatkírás részletezése

A feladat elvégzésére rendelkezésre áll egy gyártósor modell, egy 1500-as Siemens PLC, a jelek fogadását elősegítő kiegészítő modulok, és a TIA Portal fejlesztőszoftver. A következőkben a hardveres és szoftveres sajátosságok, tervezési alapelvek bemutatása olvasható.

2.1 Alkalmazott technológia megismerése

A fejlesztés megkezdése meg kell ismerkedni a programozható logikai vezérlők tulajdonságaival. Logikájukat alkalmazva tervezhető a vezérlés szerkezete.

A munkához a következő témák elsajátítása szükséges:

- A PLC-k ciklikus működése
- Image-memórián történő hozzáférés a folyamat jeleihez
- SCL-nyelv szintaktikája
- Blokk-típusok
- Rendszerblokkok (pl. timer) alkalmazásai
- Adat-típusok és struktúrák

A kihelyezett modulok két típusát alkalmazza a gyártósor: IO-modulokat és gyors-számlálókat. Az IO-modulok a PLC ki/bemeneteinek számát növelik, a gyors-számlálók a robotkar pozíciójának változásakor történő impulzusokat dolgozza fel és továbbítja a vezérlőnek.

A pneumatikus, megmunkáló és prés gyártósor-modulok összes vezérlő jele bináris, így a programblokkok nem igényelnek jelátalakításokat, az állomásokról közvetlenül jövő jeleket dolgozhatjuk fel.

A robotkar vezérlése már összetettebb a fent említett modulokhoz képest. A horizontális- és vertikális mozgásról a gyors-számlálókra érkező impulzusok adnak információt. Az egyéb helyzetváltoztatásokról is impulzusok detektálása ad adatot, de ezeknek szoftveres kezelése is lehetséges.

A Totally Integrated Automation Portal (TIA Portal) egy Siemens által fejlesztett keretrendszer, ami ipari vezérlések fejlesztésére specializálódott. A keretrendszer része a STEP7, mely a PLC-k által támogatott programozási lehetőségeket menedzseli. Rengeteg szolgáltatással rendelkező szoftverek, melyek a szakembereknek sok segítséget nyújthatnak.

2.2 Vezérlés tervezése és fejlesztése

A vezérlőszoftver építőkövei véges állapotú állapotgépek (finite state automaton, FSA) lesznek. Könnyen tervezhető, kódolható, bővíthető számítástechnikai modell. Eme matematikai koncepció átültetése PLC-s környezetbe megkönnyíti a munkát.

A szoftver-struktúra másik építőköve az úgynevezett black-box felépítés. A gyártósor minden egyes része külön blokkokból vezérlődnek, így az egyes függvények „fekete dobozoknak” látják a már elkészített állomás-vezérlőket. Ezzel a felfogással egy összetett feladatot kisebb, könnyebben kezelhető részekre bonthatjuk.

Az állomások vezérlésének implementálása után a kommunikációval szükséges foglalkozni. A kommunikációs jelek adják a moduloknak azon információkat, mikor kell a működésüket elkezdni vagy szüneteltetni.

Hatékony kommunikációs algoritmus fejlesztése kritikus, mivel az a cél, hogy az állomások állási ideje minimális legyen, tehát a folyamatok teljes párhuzamosítására kell törekedni.

2.3 Irányítási szolgáltatások

Az egyik cél az, hogy ne csak egyfajta feldolgozási procedura legyen implementálva a kódban. Az HMI-modul segítségével a folyamat-tulajdonságok változtathatóak egy operátor által, a vezérlésnek ellenben figyelembe kell vennie azt, hogy minden terméknek ugyanolyan kezelésen kell átesnie, tehát a feldolgozás konzisztenciáját fenn kell tartani.

A lehetséges hibás működéseket is kezelni kell. A gyártósor szenzorjai az időintervallum alapú ellenőrzéseket teszi lehetővé. Nem tervezett állapot esetén óvintézkedéseket tehetünk, eme projektben a hibában érintett állomások deaktiválódnak.

Olyan HMI kijelzések elvártak, mely egy, a gyártósor mellett dolgozó operátor munkáját megkönnyíti. Szükséges vizualizálni a modulok működését, a kiválasztható recepteket, manuális robotkar-vezérlőt, és egy hibakezelő felületet.

2.4 Tesztelés

A gyártósor elsősorban a terhelhetőség szempontjából kerül tesztelésre. A hibakezelést a folyamatból kézzel kivett termékekkel és próbaüzemmel lehet majd kipróbálni, valamint fontos lesz annak elemzése, hogy a hibák láttamozása után az állomások milyen állapotba tudnak majd visszatérni.

3 Előzmények

3.1 Programozható logikai vezérlők logikája

3.1.1 Az iparban betöltött szerepe¹

Az automatizálás már régen fellelhető technológiai fogalom. Automatizálás során a gépeket, vezérlő- és kommunikációs rendszereket állítjuk munkába úgy, hogy optimalizáljuk a termékek gyártását vagy a szolgáltatások szolgáltatását. A PLC egy digitális számítógép, melyet elektromechanikai folyamatok automatizálására alkalmazható, ilyen például egy gyári összeszerelő egység. Így egyszerűsödik a mérnöki munka (le lettek váltva a relék), a vezérlés pontossága nő, valamint a költségek is nagymértékben csökkenhetnek (gépek állási ideje eliminálható).

A PLC egy olyan programozható vezérlő, mely képes kezelni a vezérlő logikát, sorrendiséget, időzítést, aritmetikai adatok manipulációját, számlálót. Tekinthető egy ipari környezetben használt számítógépnek, aminek részei a központi feldolgozó egység (CPU, processzor), memória, IO interfész. A CPU felelős a vezérlő intelligenciájáért. Adatokat, státusz-információkat fogad különböző érzékelőktől, például végállás-kapcsolóktól, közelségi-kapcsolóktól; futtatja a memóriában tárolt felhasználói programot; a megfelelő kimeneti értékek beállításával irányítja az eszközöket, például mágneses szelepeket, kapcsolókat.

Az IO interfész a kommunikációs összekötő a vezérlő és az ipari folyamatokhoz közeli eszközök / manipulátorok közt. Általában vezetékes az összeköttetés, de a fejlődési trendek szerint a wireless megoldások elterjedése már nem sokáig fog várni magára. Az IO interfészen keresztül a processzor fogadja a gépeket és folyamatokat jellemző fizikai mennyiségeket, mint például közelség, pozíció, sebesség, magasság, hőmérséklet, nyomás, stb. Az érzékelt állapotok alapján a CPU parancsokat ad az eszközöknek. A PLC gyártója által fejlesztett keretrendszer segítségével elkészíthetjük és menedzselhetjük szoftverünket, melynek elkészítését a keretrendszer felhasználóbarát logikával segíti.

¹ [1] (Role of PLC in automation and its various applications)

A szabvány által definiált PLC-programozó nyelvek közül a LAD (létra-diagramm) könnyen tanulható és átlátható, mivel logikája az egyszerűbb elektromos kapcsolási rajzok felépítését követi. Ezáltal egy alapszintű szoftver megírása nem igényel nagy terjedelmű előtanulmányokat.



3-1. ábra: A legmodernebb Siemens gyártmányú vezérlő, a SIMATIC S7-1500 széria

A PLC-k fizikai kialakítása masszív, a külső ház mechanikai, termikus és elektromágneses védelmet is nyújthat. Robosztusságuk lehetővé teszi, hogy meghibásodás nélkül az ipari folyamatok közelében operálhatnak. Ennek fő előnye a jelvezetékek hosszának csökkenése, így moderálható a kábelezési költség és a vezetékekre szuperponálódó elektromágneses zavarok teljesítménye.

3.1.2 Ciklikus szoftvervégrehajtás²

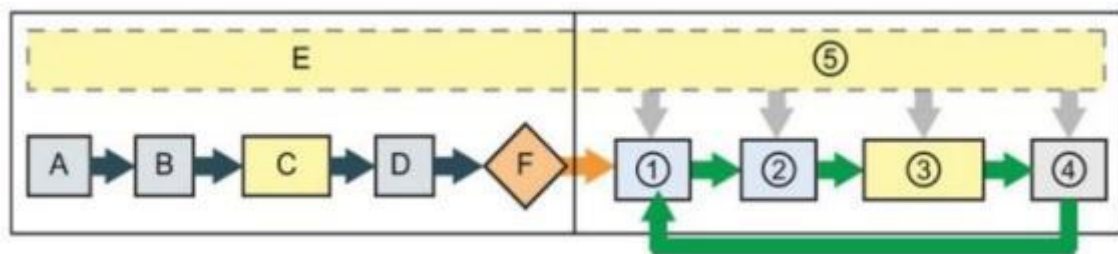
A felhasználói programot ciklikus jelleggel futtatja a CPU. A program végrehajtását a működéshez szükséges diagnosztikai és kommunikációs feladatok közé ütemezi. Tulajdonképp ez egy végtelen ciklus, melyből csak rendkívüli állapot fennállásával lehet kilépni.

A felhasználói szoftver általában nem tud közvetlenül az IO interfésszel kapcsolatot létesíteni. A program csupán a folyamatképnek nevezett belső

² [2] (Automating Manufacturing Systems with PLCs)

memóriaterületen keresztül tud kommunikálni az interfésszel. Ennek az eljárásnak több előnye is van:

- A folyamatképet a ciklus előre definiált pontjain a PLC frissíti, így a szoftver mindig az aktuális környezeti értékek alapján végezheti a vezérlést.
- Így biztosítva van a bemenetei változók konzisztenciája, tehát egy input paraméter az adott ciklus ideje alatt garantáltan változatlan lesz.
- Megakadályozza a kimeneti értékek ugrálását. Egy kimenetet több programrész is állíthat, de a ciklikus kezelés miatt csak meghatározott időpontokban változhatnak a fizikai kimenetek.



3-2. ábra: A Siemens gyártmányú PLC-kre jellemző scan-ciklus

A Siemens-gyártmányú vezérlők ciklikus végrehajtása a következő:

1. Az output-folyamatkép értékeinek kiírása a fizikai kimenetekre.
2. A fizikai bemenetek másolása az input-folyamatképbe.
3. A felhasználói program végrehajtása.
4. Rendszerdiagnosztika futtatása.
5. Megszakításkérések kiszolgálása.

A megszakítás (interrupt - IT) a mikrokontrolleres applikációk fejlesztésénél sűrűn használt eljárás. A felhasználói programot szokás a fő programfolyamnak tekinteni (amit a Siemens-terminológia OB1-nek nevez), amiben az előre tervezett, normál üzemű vezérlés implementációját szokás kódolni. De előfordulhatnak olyan esetek (jellemzően hiba vagy vészhelyzet), aminek a kezelése nem tűr halasztást. Ezeknek a szituációknak a kezelését olyan programfolyamban hozzuk létre, melyeket a CPU nagyobb prioritással kezel, és futását egy jelzőbit aktiválása kezdeményezi (például annak a jelzése, hogy egy környezeti változó veszélyes értéket lépett túl). A

diagram jól szemlélteti, hogy az IT-k ütemezése nem definiált, és bármely ciklusrésznél átvehetik a vezérlést. A kiszolgálás után a CPU folytatja a félbehagyott cikluselem futtatását. Például megszakítást kezdeményező jelenség lehet a maximális ciklusidő átlépése: A gyártó előírása szerint ennyi idő alatt egy scan-ek mindenképp le kell futnia, ha ez nem következett be, akkor hibakezelésre kell indítani.

A diagram mutat nem a ciklust alkotó folyamatokat is. Ezek az úgynevezett STARTUP üzemmódban hajtódnak végre (STARTUP üzemmódba váltás tápfeszültségre kapcsolás és reset után történik). Az üzemmód alatt a következők történnek:

- A folyamatkép formázása, beállításfüggő kezdőértékek beírása.
- Nem megmaradó memória törlése.
- Megszakítások engedélyezése.
- Esetleges STARTUP-programrészek futása.
- A folyamatkép értékeinek írásának engedélyezése az interfészre.

3.1.3 Egy PLC szoftver építőkövei

3.1.3.1 Szervező blokk (organization block - OB)

Ezzel adható meg a programunk struktúrája. Az OB-kben kerülnek meghívásra a már használatra kész egyéb blokkok. Nem lehet szoftveresen hívni, futását csak külső megszakítási esemény indíthatja. A már említett OB1 blokkot indító megszakítási eseménynek a STARTUP vagy a ciklus vége tekinthető.

3.1.3.2 Funkció (FC) és funkcióblokk (FB)

Ezek tartalmazzák magát a felhasználói programot. Rendelkeznek bemeneti és kimeneti paraméterekkel, a hívó blokkal ezeken keresztül osztanak meg adatokat.

A két blokk közti különbség az, hogy az FB képes a saját futását leíró adatokat lementeni a memóriába, mely hívásának végeztével is megmarad. Jellemzően az FB-k állandó monitorozást végeznek, így futásuk során szükségük van az előző ciklus során tárolt adatokra. Úgy is fogalmazhatunk, hogy az FB-k feladata nem végezhető el egyetlen ciklus alatt.

Az FC nem tud megmaradó adatokat eltárolni. Kisebb feladatok egyszeri elvégzését implementálja. Érdemes az FC-re úgy gondolni, mint az egyéb programozási nyelvek szubrutinjaira.

Összességében különböztessük meg a két típusú blokkot úgy, hogy míg az FB-nek szüksége van a múltbeli futások adataira (mint egy sorrendi hálózat), addig az FC az aktuális inputra adott értékekből tud eredményt visszaadni (mint egy kombinációs hálózat).

3.1.3.3 Adatblokk (DB)

Az adatblokkok tárolják a futási információkat. A projekt két fajta adatblokkot használ:

- Hívási adatblokk: Ezeket a keretszoftver akkor hozza létre, mikor egy FB hívódik. A már említett múltbeli lefutások eredményei ide tárolhatóak statikus változóként.
- Globális adatblokk: Ide csak statikus változókat tárolhatóak. Minden szoftver-építőelem olvashatja és írhatja ezeket az adatokat. Tipikusan az egész folyamatot érintő információk kapnak helyet itt, például a gyártósor modulok közti kommunikációs jelek egy globális adatblokkon keresztül érhetőek el.

3.1.3.4 Rendszerblokkok

A szoftver fejlesztése során nem szükséges minden részfeladatot kezelni. A fejlesztőkörnyezet felkínálja a lehetőséget a gyártó által létrehozott blokkok alkalmazására. Ezek az ipari vezérlés során többször felmerülő feladatokat oldják meg. A projekt az éldetektáló és időzítő rendszerblokkokat használja.

3.1.4 A használt programozási nyelv

A projektben a Siemens által támogatott strukturált vezérlő nyelv (structured control language - SCL) került alkalmazásra. Szintaktikája a Pascal-ra épül, tehát egyszerű és könnyen elsajátítható. Megtalálható benne minden, a magas szintű nyelvekben elterjedt programfutás-vezérlőutasítás (FOR ciklus, WHILE ciklus...). A CASE ciklus tökéletes kerete az alkalmazott FSA-nak, ez a döntő érv az SCL mellett.

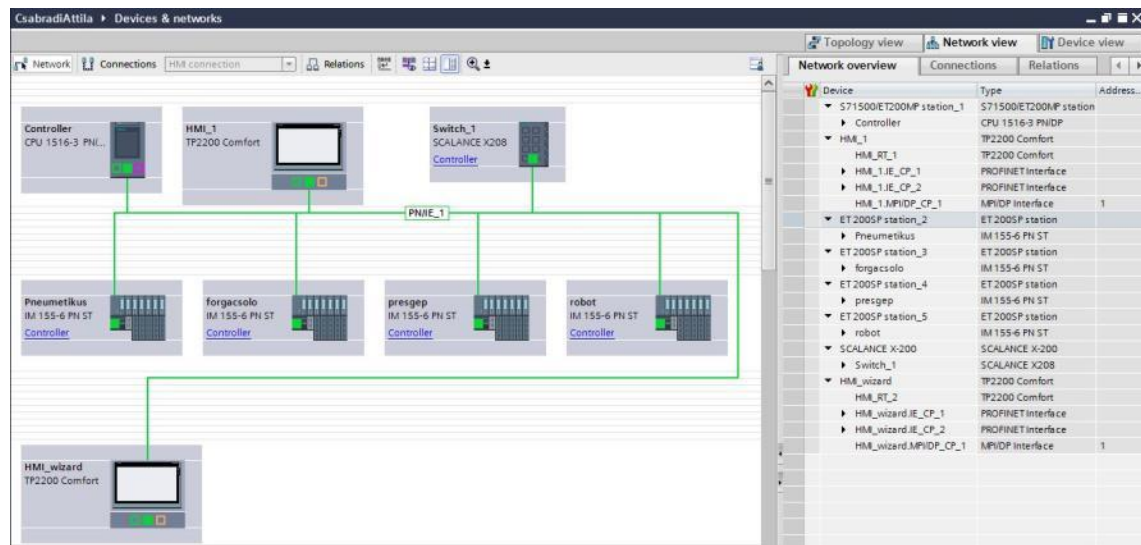
3.2 A PLC fejlesztőkörnyezete

A TIA Portal a Siemens által fejlesztett összetett integrált keretrendszer. A TIA alprogramjainak (Step7: programozás, WinCC: HMI megjelenítés) segítségével az ipari rendszerek üzembe helyezése végigkövethető a kezdetektől fogva egészen a kész vezérlés diagnosztizálásáig. A Siemens víziója az a TIA Portal-al kapcsolatban, hogy akartak alkotni egy olyan átfogó mérnöki rendszert, melyet bármely gyártelep hasznosíthat. Legyen szó tervezésről, tesztelésről, működtetésről, javításról vagy fejlesztésről, a TIA Portal célja az, hogy csökkenteni lehessen a mérnöki munkaórákat és költségeket.

3.2.1 Hálózati kép (devices & network)

Ezen a felületen állíthatóak össze a hardveres egységek, majd azok kommunikációi, összeköttetései. A képen látható a vezérlő PLC, valamint a vele kapcsolatot tartó IO-modulok.

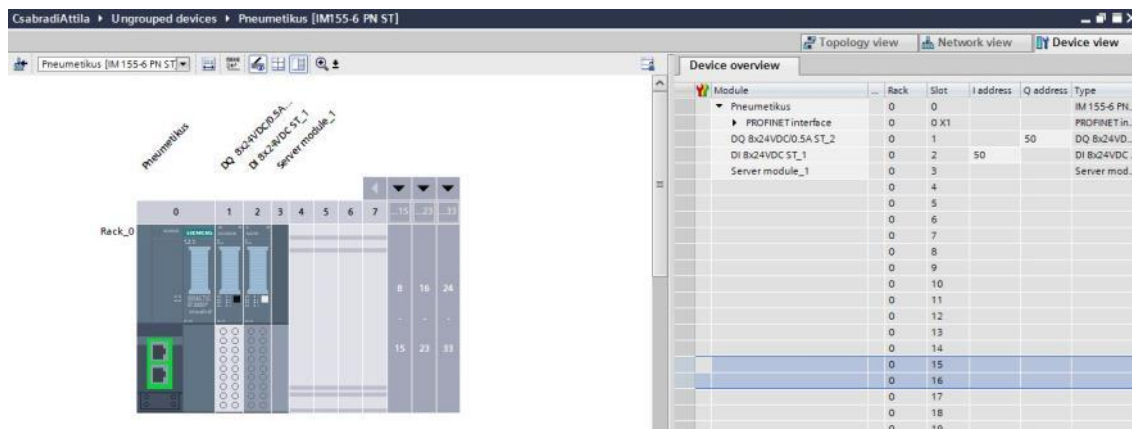
Minden egyes modul egy-egy gyártósor-állomás jeleit gyűjti össze, mert az alkalmazott 1500-as sorozatú CPU nem rendelkezik megfelelő számú csatlakozási ponttal. Az itt látható IO-modulok csupán dokkoló állomásként működnek, beállításaik során be kell tenni a jelek kezeléséért valóban felelős kártyákat.



3-3. ábra: Az elkészült hálózati kép

A fentiekén kívül be kell húzni a HMI-t, valamint egy ipari switch-et is, mely a kiépített ring-topológia irányításáért felel.

Végül be kell állítani, hogy az IO-modulok melyik címtartományra képezzék le a kezelt jeleiket, ügyelve arra, hogy ne legyen átfedés közöttük.

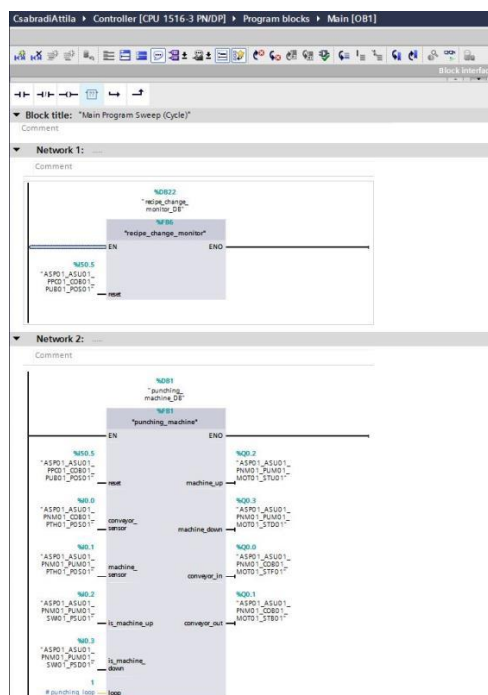


3-4. ábra: Egy dokkoló és a rá csatlakozó input és output modulok

Megjegyzés: A kapcsolódó képen két HMI-modul található, ezekből csak az egyik reprezentálja a valós hardvert, a másik csupán tesztelési feladatot látott el.

3.2.2 Szervező blokk

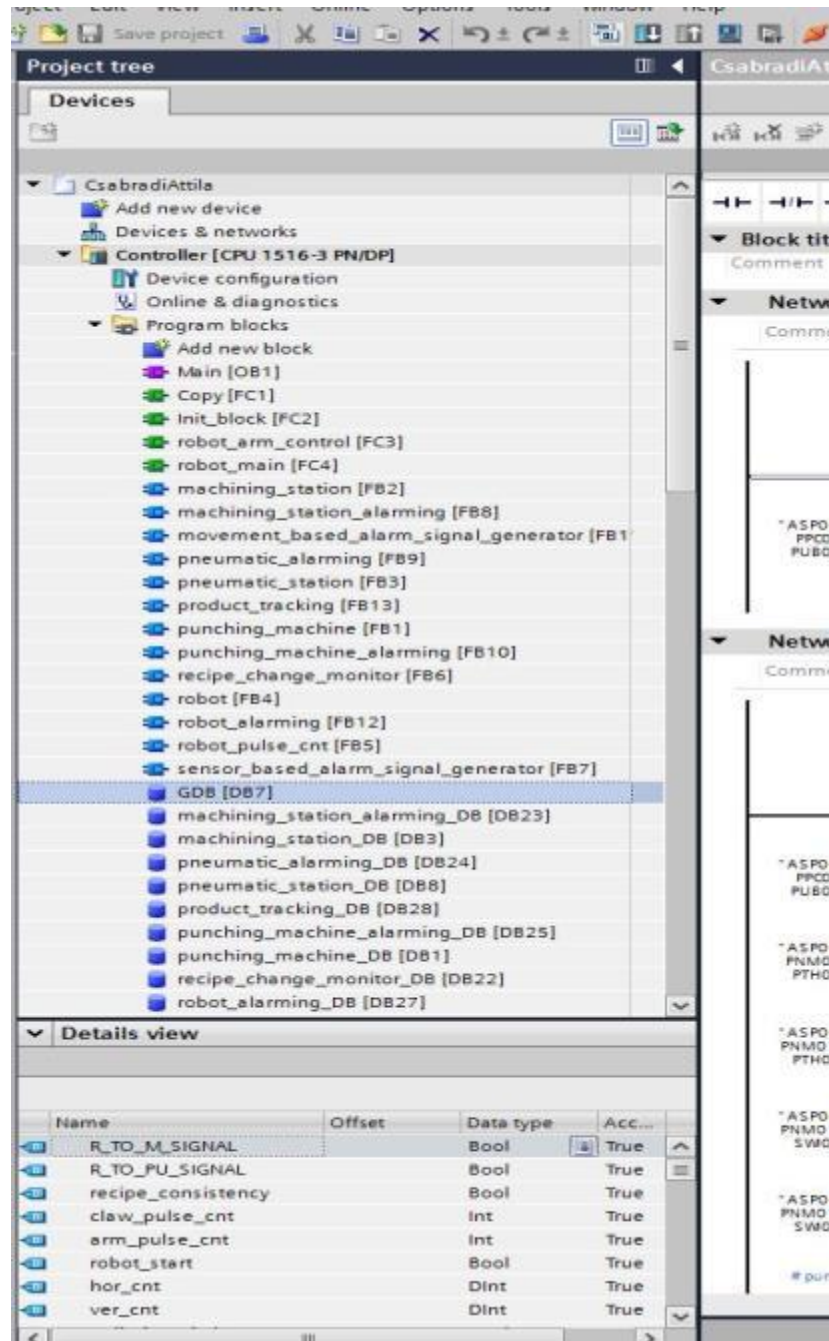
Mivel a vezérlés nem alkalmaz megszakításokat, ezért a projekt egy szervezőblokkot (OB) tartalmaz, melynek elnevezése a szokásos OB1. Itt hívódik meg az összes felhasználói blokk. A PLC futásakor az OB1 fut elsőnek, majd a benne definiált sorrendben a többi blokk.



3-5. ábra: Az OB1, az első két meghívandó blokkal

3.2.3 Projekt navigáció (project navigation view - PNV)

A fa-szerkezetű PNV-ben felsorolódik az összes programblokk és egyéb beállítás, valamint az azokhoz tartozó DB-k. Átfogó képet kapunk a projekt felépítéséről. A részletező ablakban további információkat kaphatunk a PNV-ben kijelölt adatblokkokról.

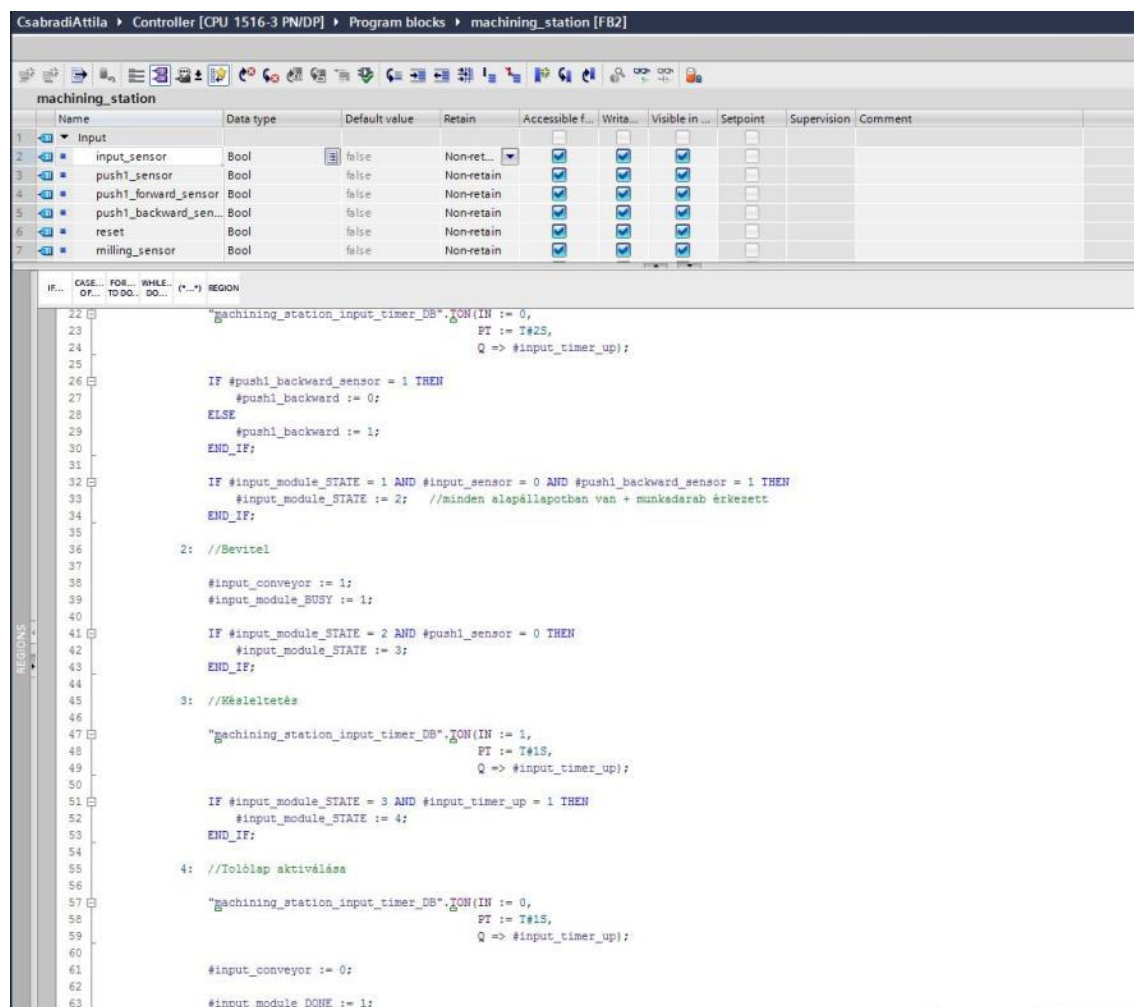


3-6. ábra: Az elkészült projekt PNV-je

Ha online-üzemmódban vagyunk (kapcsolat van kiépítve a PLC-vel, közvetlenül monitorozható az állapota) akkor arról is információt kapunk, hogy a projekt mely blokkoknál tér el a PLC-re töltött projekttől.

3.2.4 SCL szerkesztő

Szövegszerkesztőjében lehet fejleszteni a blokkok logikáját. Az intelligens szövegbevitel segítette a nyelvi elemek, valamint a változók helyes írását.



3-7. ábra: SCL editor

A szerkesztő felső része az interfész, ahol egyrészt megadhatóak azon paramétereket, amiken keresztül kommunikál a blokk (ezek látszódnak az OB1-ben történő meghíváskor is), valamint FB-k esetén a statikus változókat (olyan adatok, melyek nem temporálisak, tehát a ciklus leteltével értéke megmarad, és a blokk belső logikájában van szerepe; az FC-k csak temporális változókat kezelnek) is itt hozhatóak létre.

3.2.5 Adatblokkok

A projektben két típusú DB van használatban:

- Példányosított (instance) DB: Az FB-k adatait tartalmazó adatblokkok. Ezek automatikusan jönnek létre FB vagy rendszerblokk (időzítő, változás-detektor) hívása esetén.
- Globális DB: Az ebben lévő adatokat az összes blokk változtathatja. Jellemzően az egész gyártósort vezérlő információk vannak benne (pl. állomások közti kommunikáció), valamint a HMI-modul által felhasznált adatok.

CsabradiAttila ▶ Controller [CPU 1516-3 PN/DP] ▶ Program blocks ▶ GDB [DB7]

Keep actual values Snapshot Copysnapshots to start values Load start values as actual values

	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Supervision	Comment
1	▼ Static									
2	R_TO_M_SIGNAL	Bool	false							
3	R_TO_PU_SIGNAL	Bool	false							
4	recipe_consistency	Bool	false							
5	claw_pulse_cnt	Int	0							
6	arm_pulse_cnt	Int	0							
7	robot_start	Bool	false							
8	hor_cnt	DInt	0							
9	ver_cnt	DInt	0							
10	tt_limit_switch	Bool	false							
11	height_limit_switch	Bool	false							
12	claw_limit_switch	Bool	false							
13	arm_limit_switch	Bool	false							
14	claw_close_move	Bool	false							
15	claw_open_move	Bool	false							
16	arm_forward_move	Bool	false							
17	arm_backward_move	Bool	false							
18	punching_machine_D...	Bool	false							
19	punching_machine_B...	Bool	false							
20	machining_station_D...	Bool	false							
21	robot_mode	Int	0							
22	punching_machine_...	Int	1							
23	global_stop	Bool	false							
24	product_cnt	Int	0							
25	robot_stop_done	Bool	false							
26	M_PN_ROUTE	Array[0..6] of *robo...								
27	S_M_ROUTE	Array[0..6] of *robo...								
28	PN_M_ROUTE	Array[0..6] of *robo...								
29	PU_PN_ROUTE	Array[0..6] of *robo...								
30	M_PU_ROUTE	Array[0..6] of *robo...								
31	PN_PU_ROUTE	Array[0..6] of *robo...								
32	M_PU_ROUTE	Array[0..6] of *robo...								
33	recipe_change_ack	Bool	false							
34	new_properties	*process_properties								
35	used_properties	*process_properties								
36	M_AL_1	*alarm_signal_gene.								
37	M_AL_2	*alarm_signal_gene.								machining station input alarming
38	M_AL_3	*alarm_signal_gene.								machining station push to milling alarming
39	M_AL_4	*alarm_signal_gene.								machining station drilling alarming
40	M_AL_5	*alarm_signal_gene.								machining station output alarming
41	M_AL_6	*alarm_signal_gene.								
42	M_AL_7	*alarm_signal_gene.								
43	M_AL_8	*alarm_signal_gene.								
44	occured_errors	Array[0..3] of Bool								
45	PN_AL_1	*alarm_signal_gene.								
46	PN_AL_2	*alarm_signal_gene.								
47	PN_AL_3	*alarm_signal_gene.								
48	PN_AL_4	*alarm_signal_gene.								

3-8. ábra: A globális DB

CsabradiAttila ▶ Controller [CPU 1516-3 PN/DP] ▶ Program blocks ▶ robot_DB [DB15]

Keep actual values Snapshot Copy snapshots to start values Load start values as actual values

robot_DB

	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Supervision	Comment
1	▼ Input									
2	reset	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
3	turntable_limit_switch	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
4	vertical_limit_switch	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
5	claw_limit_switch	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
6	arm_limit_switch	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
7	▼ Output									
8	turntable_right	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
9	turntable_left	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
10	vertical_up	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
11	vertical_down	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
12	claw_close	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
13	claw_open	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
14	arm_forward	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
15	arm_backward	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
16	hor_gate	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
17	ver_gate	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			b
18	hor_reset	Byte	16#0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
19	ver_reset	Byte	16#0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
20	InOut									
21	▼ Static									
22	state	Int	0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
23	array_cnt	Int	0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
24	in_pos	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
25	wait	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			

3-9. ábra: A robot példányosított vezérlő DB-je

3.2.6 PLC tag-ek

A gyártósor-állomások műszaki leírásaiból, valamint az IO-modulok által lefoglalt címtartományok alapján létrejön egy memóriatérkép, melynek a megfelelő részeinek olvasásával/írásával megvalósítható a vezérlő logika.

CsabradiAttila ▶ Controller [CPU 1516-3 PN/DP] ▶ PLC tags ▶ Robot_Arm ▶ Digital_Output_2 [8]

Digital_Output_2

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Supervision	Comment
1	ASP01_ASU01_RBA01_GRI01_MOT01_SOP01	Bool	%Q90.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q1) motor gripper open
2	ASP01_ASU01_RBA01_GRI01_MOT01_SCL01	Bool	%Q90.1		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q2) motor gripper close
3	ASP01_ASU01_RBA01_GRA01_MOT01_STF01	Bool	%Q90.2		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q3) motor grip arm befor
4	ASP01_ASU01_RBA01_GRA01_MOT01_STB01	Bool	%Q90.3		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q4) motor grip arm back
5	ASP01_ASU01_RBA01_VEA01_MOT01_STD01	Bool	%Q90.4		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q5) motor vertical axis down
6	ASP01_ASU01_RBA01_VEA01_MOT01_STU01	Bool	%Q90.5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q6) motor vertical axis up
7	ASP01_ASU01_RBA01_TUT01_MOT01_STR01	Bool	%Q90.6		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q7) motor turntable right
8	ASP01_ASU01_RBA01_TUT01_MOT01_STL01	Bool	%Q90.7		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		(Q8) motor turntable left
9	<Add new>				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

3-10. ábra: A robotkar vezérlőjeleinek tag table-je

Annak érdekében, hogy ne kelljen a nem sok információtartalommal rendelkező címekre hivatkozni, létrehozunk tag-eket. Ezek szimbolikus nevet és egyéb megjegyzést adnak a memóriaterületeknek, így könnyebben átlátható a munka.

3.2.7 Watch- és force table

A program-diagnosztizálásban hasznos eszközök a címben szereplő felületek. Ezek segítségével a kívánt memória-szelet értékét figyelhetjük, valamint egy értéket adhatunk meg neki manuálisan.



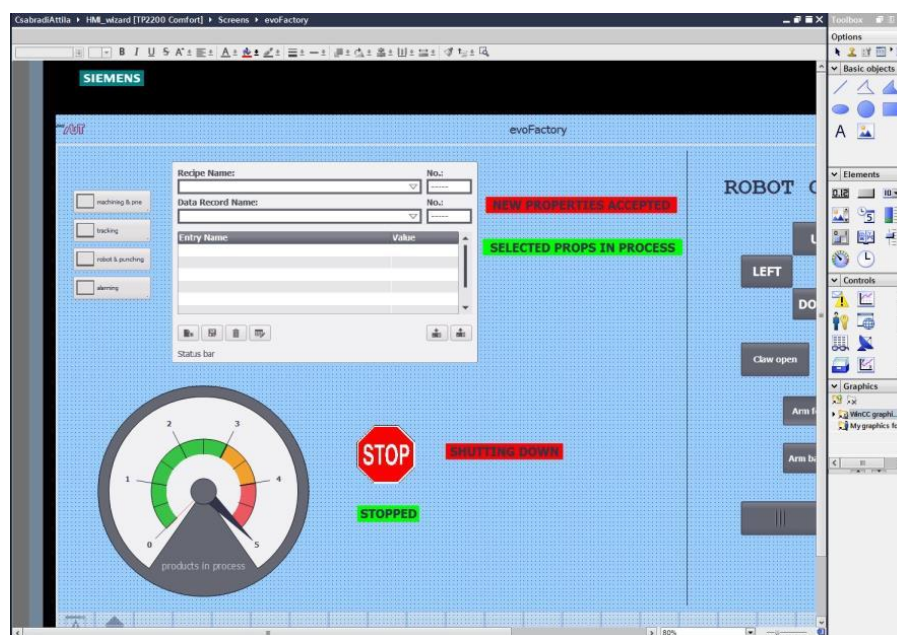
Name	Address	Display format	Monitor value	Monitor with trig...	Modify with trigge	Modify value	Comment
ASP01_ASU01_MAS01_COB01_MOT01_STS01	%Q60.0	Bool		Permanent	Permanent	TRUE	toló1 be
ASP01_ASU01_MAS01_COB02_MOT01_STS01	%Q60.1	Bool		Permanent	Permanent	FALSE	toló1 ki
ASP01_ASU01_MAS01_COB03_MOT01_STS01	%Q60.2	Bool		Permanent	Permanent	FALSE	toló2 be
ASP01_ASU01_MAS01_COB04_MOT01_STS01	%Q60.3	Bool		Permanent	Permanent	FALSE	toló2 ki
ASP01_ASU01_MAS01_SLI01_MOT01_STB01	%Q60.4	Bool		Permanent	Permanent	FALSE	bemeneti szalag
ASP01_ASU01_MAS01_SLI01_MOT01_STF01	%Q60.5	Bool		Permanent	Permanent	FALSE	milling szalag
ASP01_ASU01_MAS01_SLI02_MOT01_STB01	%Q60.6	Bool		Permanent	Permanent	FALSE	milling
ASP01_ASU01_MAS01_SLI02_MOT01_STF01	%Q60.7	Bool		Permanent	Permanent	FALSE	fűrő szalag
ASP01_ASU01_MAS01_MIM01_MOT01_STS01	%Q61.0	Bool		Permanent	Permanent	FALSE	fűrő
ASP01_ASU01_MAS01_DRM01_MOT01_STS01	%Q61.1	Bool		Permanent	Permanent	FALSE	kimeneti szalag

3-11. ábra: A megmunkáló állomás bemeneteit tesztelő watch-table

Érdemes megemlíteni, hogy nem csak a watch table segítségével lehet értékeket monitorozni, hanem akár a funkció- és adatblokkok felületén is (online üzemmódban). A globális adatblokkban például a kommunikációs jelek megfigyelése volt praktikus, a funkcióblokkokban a futó programrészek monitorozás közben ki vannak emelve, megkönnyítve a program futásának megfigyelését.

3.2.8 HMI screens (felületek)

A HMI felületkezelőjének segítségével szerkeszthetők olyan felületeket, melyek hasznos adatokat mutatnak a gyártósorról.



3-12. ábra: Egy HMI-felület szerkesztése

A grafikai elemeket drag & drop elvvel lehet elhelyezni, majd felület-szerkesztővel alakítható, hogy a PLC adott adatainak függvényében hogyan változzon a kinézetük.

A HMI a receptek egyszerű menedzselésére is ad segítséget. A receptnevek megadása után csupán a változtatandó értékeket kell megadni.

A receptkezelő felületen aktivált értékek azonnal beíródnak a megadott adatterületekbe, mely hibás működést okozhat. Ennek kezelése a PLC oldaláról történik.

A HMI képes a PLC bitjeit monitorozni, és ha az adott bit aktiválódik, akkor egy megadott hibaüzenetet tud megjeleníteni a vészjelzéseket jelző ablakban. Szükséges megadni a láttamozó biteket is, melyek aktiválhatóak a kezelő ablakban, és a PLC ebből tudja majd, hogy a felmerült hibás működés nyugtázva lett.

3.3 1500-as széria PLC³

A demonstrációs gyártósort egy 1516-3 PN/DP kódú Siemens gyártmányú PLC vezérli. A szakdolgozat írásakor ez a PLC-család volt a Siemens-vezérlők közt a legkimagaslóbb paraméterekkel bíró, valamint a konkurenciával összevetve is kijelenthető, hogy az alkalmazott hardver a state-of-the-art termékek közé tartozik. Kereskedelmi ára 6000 USD (~ 2 millió forint).

³ [3] (SIMATIC S7-1500 CPU 1516-3 PN/DP Manual)



3-13. ábra: S7-1500 vezérlő

A hardver család tagjai a diszkrét automatizálás bármely területén alkalmazhatóak. A moduláris és mozgó alkatrész nélküli hűtés, az elosztott rendszerek könnyű implementálása és felhasználóbarát kezelése gazdaságos megoldássá teszi rengeteg feladat számára. A fail-safe (hibatűrő) CPU-k alkalmazhatók olyan kritikus rendszerekben, ahol a vezérlés kiesése nem tolerálható. Nagy ellenállással rendelkezik az elektromágneses hullámokkal, ütéssel és vibrációkkal szemben.

Az alkalmazott PLC kommunikációs és programfuttatási lehetőségei:

- PROFIBUS
- PROFINET IO RT/IRT (real-time / izokrón real-time)
- munkamemória: 6.5 MB
- egy bit beállítási ideje: 10 ns

(A használt PLC PROFINET kommunikációt használ az IO-modulokra és a HMI-egységre történő csatlakozáshoz.)

Az 1500-as hardverek rendelkeznek egy egyszerű szöveges kijelzővel. A kijelző alapvető információk megjelenítésére alkalmas, mint például a firmware verziószám, a csatlakozott modulok serial számai. Ezen felül a vezérlő IP címét és egyéb hálózati

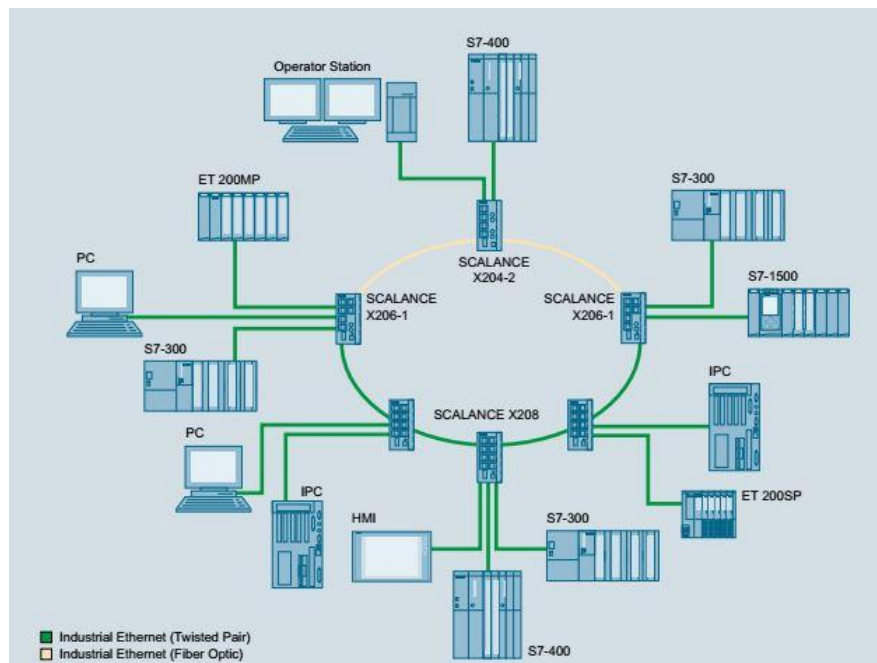
beállítások változtathatóak programozó eszköz nélkül. A hibaüzenetek is megjelennek, segítve az operátorokat a mielőbbi újraindításban.

3.4 További hardverek

3.4.1 Ipari ethernet switch⁴

A gyártósor hardverjeinek összeköttetésében fontos szerepet játszik a SCALANCE X208 ipari ethernet switch (IES). A switch-ek ideálisak az ipari ethernet hálózatok felépítéséhez (10 – 100 Mbit/s sebességig; vonal, csillag vagy ring stuktúra).

Az IES egy aktív hálózati komponens, mely különböző topológiákat támogat. IES-el rendelkező hálózatok kontruálhatóak elektromos vagy optikai összeköttetéssel, vonal / csillag / ring topológiával. A switch megfelelő címekre küldi tovább az adatot.



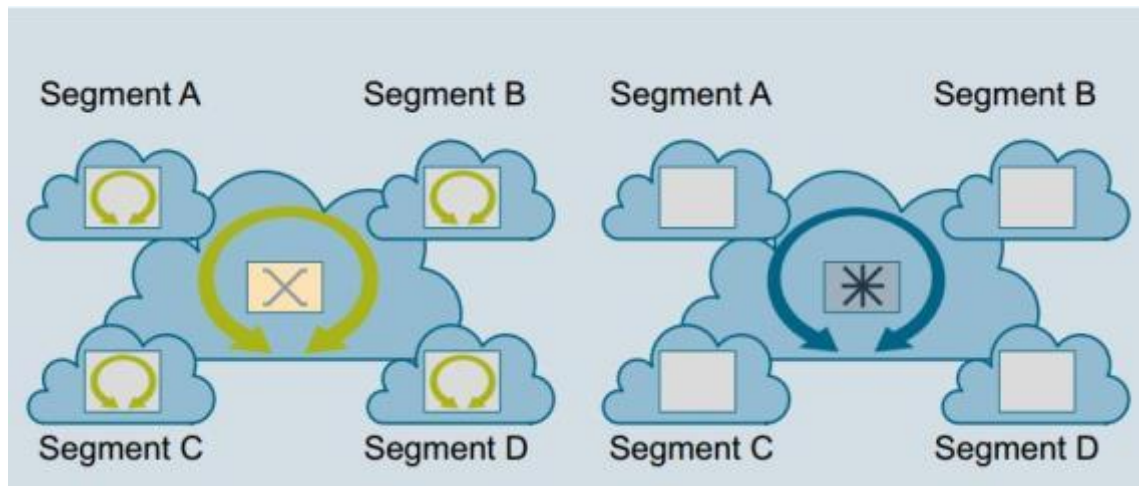
3-14. ábra: Switched hálózatra példa, 6 alhálózattal

A switching technológia lehetővé teszi a párhuzamos kommunikációt úgy, hogy a hálózat felosztódik több szegmensre, így megoszlik a terhelés. A kommunikáció egy szegmensben a többi szegmenstől függetlenül folyhat. Egyszerre több adatcsomag küldése is lehetséges.

További előnyök:

⁴ [4] (SCALANCE X – Industrial Ethernet Switches)

- Switch-ek használhatóak subnet-ek és hálózati szegmensek létrehozására.
- Adatátvitel és hálózati teljesítmény növekszik a kommunikáció strukturálása miatt.
- Ring topológia akár 50 IES-el, akár 150 km kiterjedéssel telepíthető anélkül, hogy a jelkésleltetést figyelembe kelljen venni.



3-15. ábra: Switched és shared hálózat

<i>IES-t tartalmazó hálózat (switched)</i>	<i>IES-t nem tartalmazó hálózat (shared)</i>
Mindegyik különálló szegmens a maximális teljesítményét / bitrátáját képes nyújtani	Az összes végkésző csak a hálózat teljesítményét / bitrátáját képes nyújtani
Párhuzamos adatforgalom több szegmensben, egyszerre több adatkeret	Az összes adatcsomag végigmegy az összes szegmensen
Szűrés: Csak a kiválasztott csomagok léphetik át a szegmenshatárait, a többi lokális kommunikációnak marad meg	Egyszerre csak egy üzenet a hálózatban

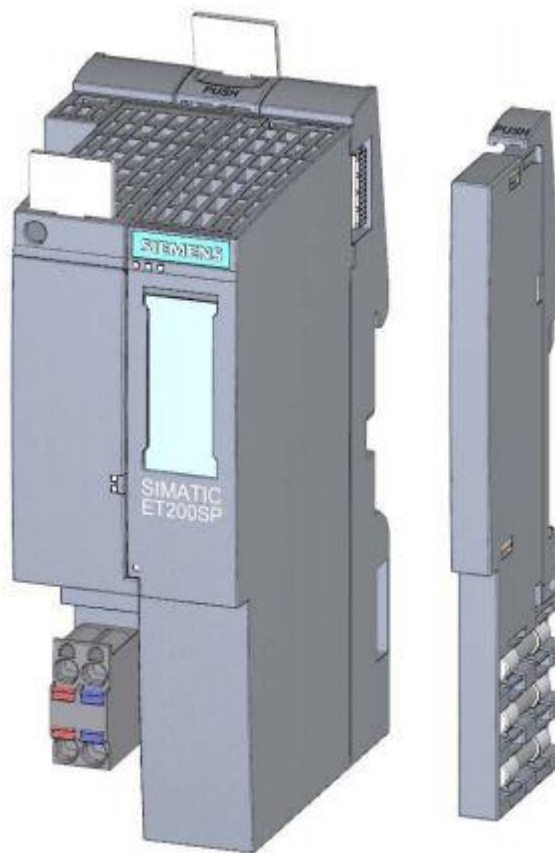
3-1. táblázat: Switchelt és nem-switchelt hálózatok összehasonlítása

3.4.2 IO modulok



3-16. ábra: A gyártósor-állomások interface moduljai és IO modulok

3.4.2.1 Dokkoló állomás⁵



3-17. ábra: IM 155-6 PN ST interface modul

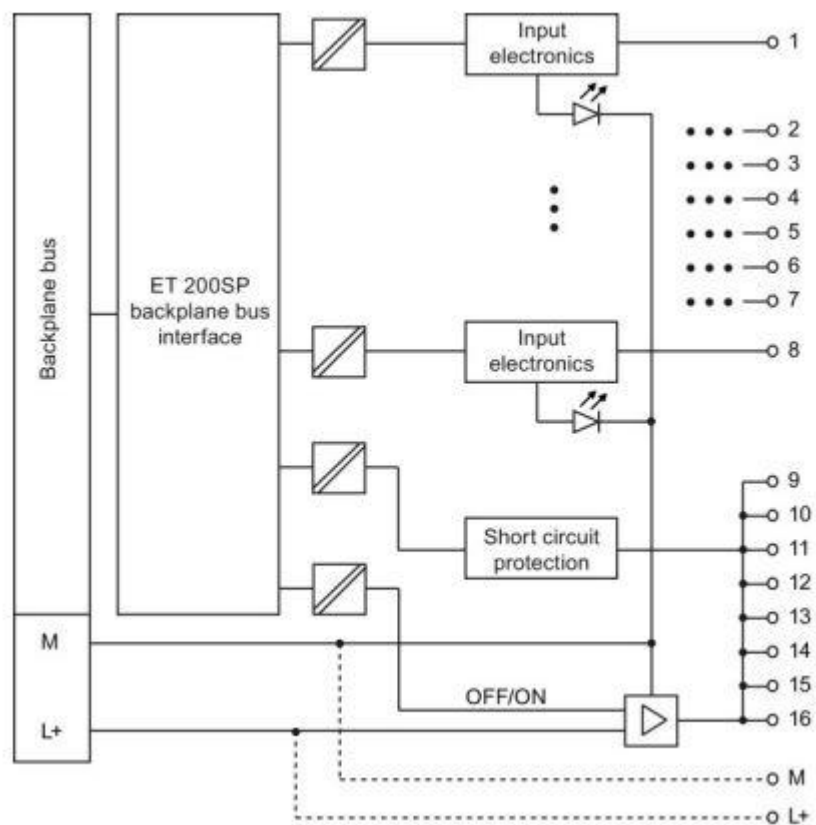
Az interfész modul összekapcsolja a Siemens által támogatott ET 200SP elosztott IO rendszert a PROFINET hálózattal. Mindegyik gyártósor-állomás rendelkezik egy ilyen dokkolóval, mely összegyűjti az adott hardver összes jelét.

⁵ [5] (ET 200SP Interface module IM 155-6 PN ST)

3.4.2.2 Input modul⁶



3-18. ábra: DI 8x24VDC ST input modul



3-19. ábra: A bemeneti modul sematikus rajza

⁶ [6] (ET 200SP Digital input module DI 8x24VDC ST)

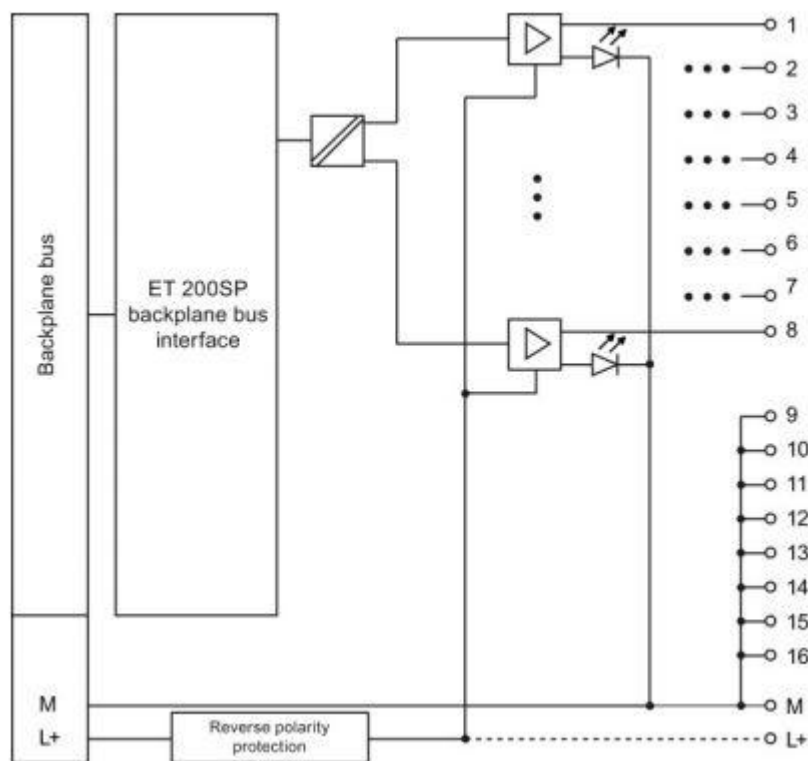
A sematikus rajzon jól látható, hogy egy modul 8 bitet tud kezelni. A 9 – 16 csatlakozókon kapják a szenzorok a működésükhöz szükséges tápot, az 1 – 8 csatlakozásokon pedig a szenzorok válaszjeleit kapjuk, melyeket a feldolgozó elektronika továbbít switcheken keresztül az ET 200SP buszra.

3.4.2.3 Output modul⁷



3-20. ábra: DQ 8x24VDC/0.5A ST output egység

⁷ [7] (ET 200SP Digital output module DQ 8x24VDC/0.5A ST)



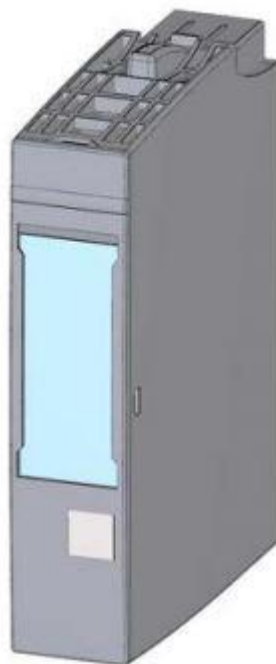
3-21. ábra: Output modul sematikus rajza

Az ET 200SP buszról switch-elt jelet egy vezérelt jelbufferen keresztül az 1 – 8 csatlakozóra küldi. Az input és output esetén is látható, hogy a jelek állapotáról LED-lámpák alapján is kaphatunk információt.

3.4.3 Gyors-számláló modul (high speed counter - HSC)⁸

A robotkar vertikális és horizontális pozíciójáról egy-egy kvadrátúra-enkóder ad információt. A enkóderek kimenete egyszerű impulzusoknak tekinthetőek, de ezek váltásának olyan nagy a frekvenciája, hogy azt a PLC nem tudná üzembiztosan feldolgozni. Ezért ezeket a jeleket két HSC dolgozza fel, majd a robotkar pozícióját leíró integereket továbbítja a PLC-nek.

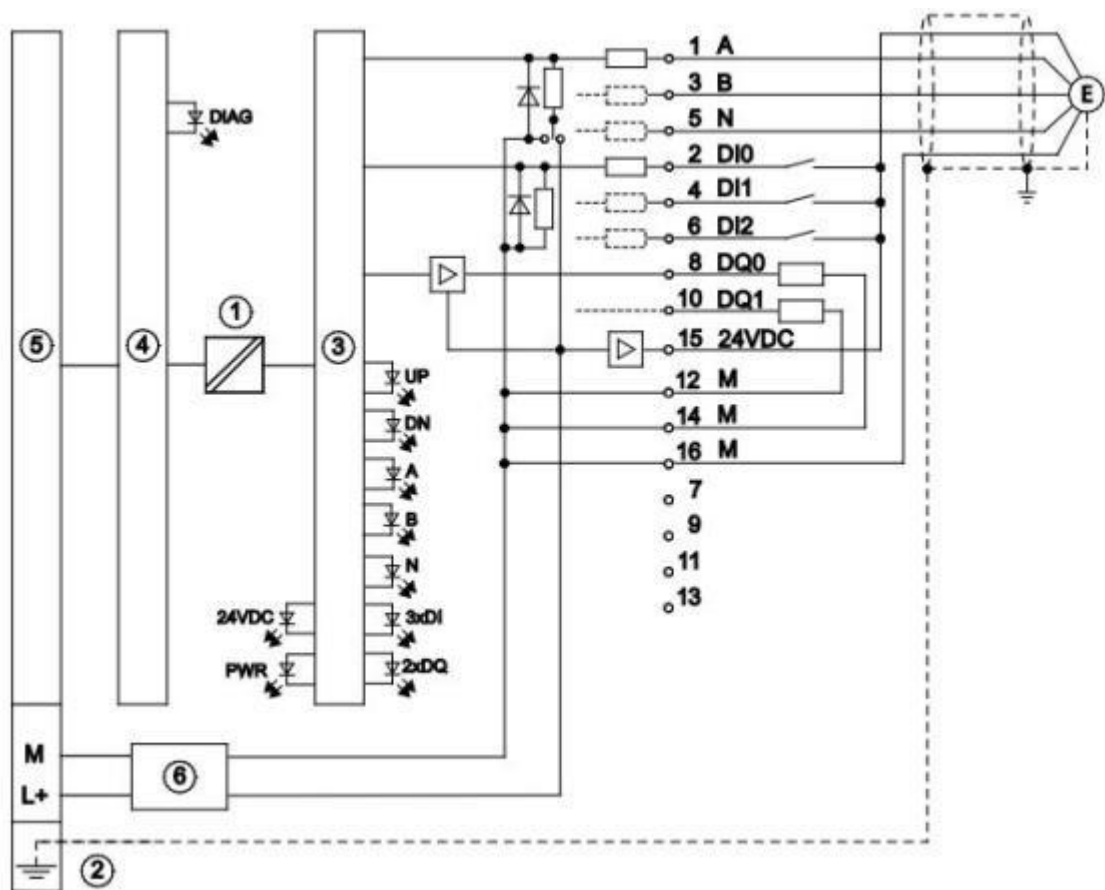
⁸ [8] (ET 200SP Technology Module TM Count 1x24V)



3-22. ábra TM Count 1x24V gyors-számláló modul

A robotkar tesztelése közben kiderül, hogy a maximális horizontális kitérés (start pozíciótól a tölcsérig) során a HSC kb. 10000 impulzust kapott, mialatt az út megtétele kb. 3 másodpercig tartott, tehát ez másodpercenként ~3300 impulzus kiadását jelenti. Az 1500-as PLC leggyorsabb scan-je 1ms^9 , tehát a bemenetek olvasása legjobb esetben is 1000-szer történik meg másodpercenként. Az impulzusok szoftveres megoldásával azok jelentős hányada nem kerülne regisztrálásra. A HSC ezt a problémát oldja meg.

⁹ [9] (SIMATIC S7-1500, ET 200SP, ET 200pro Cycle and response times)

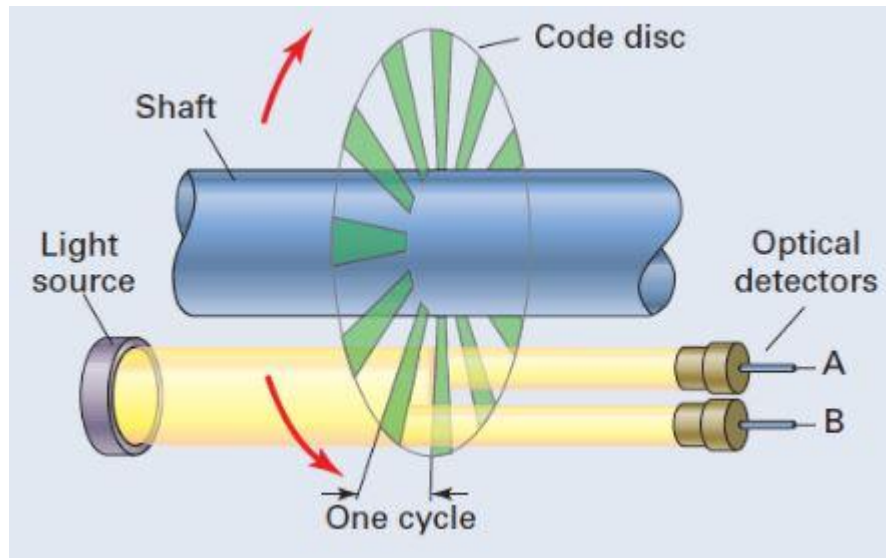


3-23. ábra: TM Count és egy monitorozandó technológiai egység

- L+ / M tápegység: 24V egyenáram szolgáltató. Belső védőáramkör használatával kivédhető az esetleges polaritáscsere.
- Enkóder jelei: Három enkóder jel beolvasására képes az egység, a két tárcsajel, valamint a zérus-jel, mely fordulatonként ad egy impulzust. A robotkar nem alkalmaz zérus-jelet, mivel nincs szükség teljes fordulatra, de a kábelezése miatt amúgy se lenne kivitelezhető. Az inputjelek elektronikus izolálva vannak a busz interfésztől.
- Digitális input (DI): Egy digitális inputot felhasználunk a számláló engedélyezésére.
- Digitális output (DQ): Digitális output aktiválható, ha dekóder egy bizonyos értéket elér. Ezt a funkciót a kész projekt nem használja.

3.4.4 Kvadrátúra-enkóder (QU-EN)¹⁰

A QU-EN az inkrementális enkóderek családjába tartozik. Az inkrementális enkóderek a lineáris mozgás során fordulatokként egyenlő számú pulzust tudnak generálni.

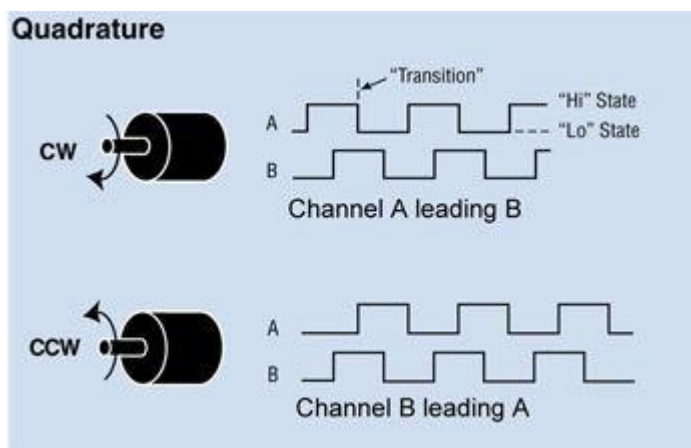


3-24. ábra: A kvadrátúra-enkóder felépítése

A QU-EN a következő részekből áll: Egy kód-diszkből, mely együtt forog a monitorozandó tengellyel, egy fényforrásból, ami a kód-diszk fűrt sávját világítja át, és optikai szenzorokból (a fényforrás és érzékelő statikus).

A kód-diszk két sávot tartalmaz, amiket nevezhetünk A és B csatornáknak. Ezek a sávok úgy vannak fűrva, hogy a rajtuk átjutó fény által generált jelalakok 90°-os eltolásban legyenek egymáshoz képest.

¹⁰ [10] (http://www.dynapar.com/Technology/Encoder_Basics/Quadrature_Encoder/)



3-25. ábra: Forgásirány meghatározása a csatornák jelváltozásának sorrendjéből

Olyan alkalmazásoknál, ahol a mozgás irányának meghatározása is szükséges, ott egy vezérlő (jelen esetben a HSC) figyeli a csatornák szint-változásait és ebből következtethető a forgásirány. Az ábrán bemutatott példa esetén az óramutató forgásával megegyező irányban az A csatorna 90°-al megelőzi a B csatornát, így például a monitorozott 0-1 jelváltások is ilyen sorrendben következnek be. Ellentétes irányú forgás esetén ez a sorrend felcserélődik.

Ebből belátható, hogy a HSC a jelek folytonos monitorozásából megállapítja, hogy a pozíciót jellemző számot növelje vagy csökkentse.

3.5 PROFINET¹¹ (process field net¹²)

A PROFINET a világ elsősorú ipari hálózati megoldása, mely képes kezelni a gyártáshoz közeli egységeket (pl. szenzorok, aktuátorok), vezérlőegységeket, valamint a vállalati informatikai rendszereket.

A PROFINET teljes mértékben kompatibilis az irodai/kereskedelmi ethernet rendszerekkel, és ennek összes szolgáltatásával rendelkezik. A két hálózat közti különbség annyiban merül ki, hogy az irodai hálózatok nem képesek valós idejű teljesítményre (ami elengedhetetlen az ipari automatizáláshoz), és ipari környezettel szembeni tűrőképességük sem ideális.

¹¹ [11] (<http://us.profinet.com/technology/profinet/>)

¹² munkafolyamat közeli hálózat

A PROFINET-nek meg kell felelnie mind az üzemi és a vállalati informatikai hálózatok által támasztott igényeknek. Eme igények sokszor egymásnak ellentmondóak, ezért a PROFINET három különböző szolgáltatás nyújtására képes:

- Szabványos TCP/IP: Nem-determinisztikus funkciókra használják, mint például parametrizálásra, videó/audió továbbításra, adatok küldésére a menedzsmenti szintek felé.
- PROFINET RT (valós idejű): Az RT esetén a TCP/IP szolgáltatási réteg meg van kerülve, hogy determinisztikus teljesítményt lehessen nyújtani az 1 – 10 ms reakcióidejű tartományban. Szoftver-alapú alapú megoldást nyújt, mely alkalmazható az általános IO feladatoknál.
- PROFINET IRT (izokrón valós idejű): Itt a jel-prioritizálás és ütemezett kapcsolások adnak nagy pontosságú szinkronizációt mozgás-vezérlésekhez. Egy milliszekundum alatti ciklusidő elérhető, aminek bizonytalansága (jitter) mikroszekundum nagyságrendű. Ez a szolgáltatás alkalmazásorientált kiegészítő integrált áramkörök használatát igényli.

Mind a három szolgáltatásra egyszerre is lehet igényt tartani. Sáv szélesség megosztás garantálja, hogy egy IO ciklus legalább fele rendelkezésre áll TCP/IP kommunikációra, teljesen függetlenül a felhasználás módjától.

A PROFINET egyik legnagyobb előnye, hogy támogatja a terepi buszok (field bus) használatát. A terepi busz egy digitális ipari hálózati technológia, melyen megvalósítható adatok átvitele vezérlők és gyártásközelbeni eszközök között. Egy fejlett kommunikációs eljárás megkönnyíti a vezérléssel, monitorozással kapcsolatos fejlesztéseket.

A terepi busz előnye, hogy képes több eszközt összekötni egy kábellel. Ez magában (ahogy a tapasztalat mutatja) csökkenti a tervezési, mérnöki, üzembehelyezési és karbantartási költségeket akár 40%-al.

Egy mérnöki tervező eszköz (a szakdolgozatban ez a Totally Integrated Automation Portal) összekapcsolódik a vezérlővel, majd annak információkat ad át az IO eszközökről egy GSD fájlból. A projekt konfigurálása után letöltődik a vezérlőre, ezután tud a PLC kapcsolatot felvenni az IO eszközökkel.

Az IO eszközök hierarchikusan vannak strukturálva:

- Eszköz (device): A szakdolgozatban ezek a Devices & networks felületen látható dokkolók.
- Modul: A ténylegesen IO jeleket feldolgozó kártyák, melyek a dokkolókban foglalnak helyet.
- Csatorna: Egy modul több csatornát is képes kezelni.

Ki- és bemenetek cserélődnek a vezérlő és IO eszközök között mint ciklikus adat. Ezen felüli információk (pl. diagnosztika), amiknek előfordulása kevésbé gyakori, azok aciklikus adatként kezelődnek.

A PROFINET és PROFIBUS¹³ szolgáltatásokat egy szervezet (PI North America) hozta létre és fejleszti, ezért több mérnöki koncepcióban mutatnak hasonlóságot. De például az eszközök hardveres leírását szolgáló GSD fájlok esetén a PROFIBUS ASCII, a PROFINET xml fájlokat alkalmaz.

A PROFIBUS R-485 fizika rétegről a PROFINET Ethernet-jére váltás egy modernebb technológiára váltás. Az ethernet mindenhol fellelhető. Az ethernet használatával a PROFINET biztosította jövőjét; a kereskedelmi ethernet fejlődésével együtt a PROFINET is fejlődik. A PROFINET indulásakor a 100Mbit/s sebesség volt gyakori, de a mai gigabites ethernet-el is kompatibilis.

Az ethernet-alap nagyobb sávszélességet és üzenetméretet, nagy címzési tartományt adott. Habár a struktúra nem korlátozza a címzési tartományt, egyes vezérlőknek lehetnek korlátjaik, mely CPU és memóriafüggő.

A PROFINET gyorsaságának egyik oka az adó / nyelő modell használata. Az összes csatlakozott végeselem bármikor kommunikálhat. Mióta az ethernet hálózatokat teljesen switch-vezéreltek (full-duplex), azóta az esetleges ütközések is eliminálva vannak.

A PROFIBUS ezt a master / slave modellel érte el: A master mindig kézben tartotta a hálózatot, a végállomások csak felkérés esetén kommunikálhattak.

¹³ [12] (<http://us.profinet.com/the-difference-between-profibus-and-profinet/>)

3.6 Ring topológia¹⁴

A PLC, IO modulok és a HMI-k összeköttetésénél az úgynevezett ring topológia lett alkalmazva. A fő cél az volt, hogy egy esetleges hardveres meghibásodás által okozott kár minimalizálva legyen. A munka szempontjából legfőbb előnye az, hogy egy összeköttetés kiesése esetén (pl. csatlakozó, kábel vagy modul meghibásodás) a kommunikáció fennmarad.

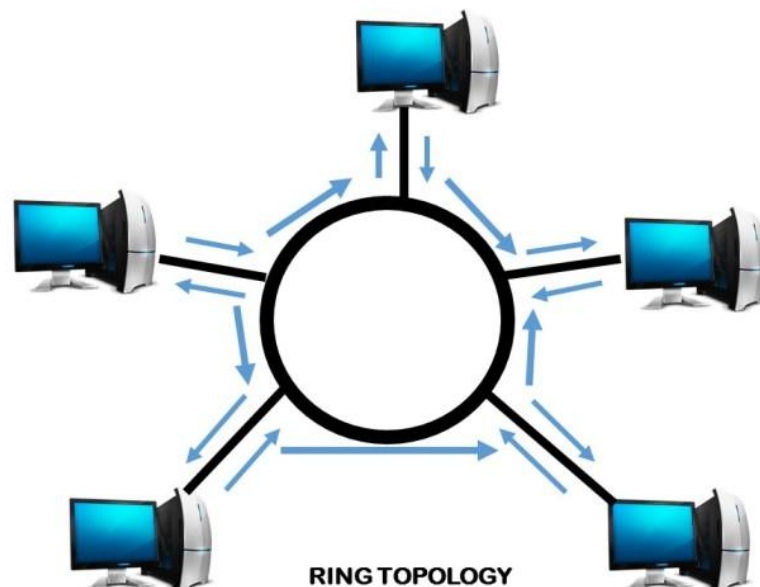
A ring topológiában az eszközök kör alakban vannak csatlakoztatva. Minden adatcsomag körbemegy a hálózaton amíg el nem éri célját. Az illusztráción jól látható, hogy egy összeköttetés megszűnésének esetén az információk folyása fenntartható (ennek kezeléséért az ipari switch felelős az STP protokoll¹⁵ betartatásával).

Egyéb előnyök:

- Az adat egy irányú folyása, így az ütközés lehetősége minimális.
- Hálózati szerver nem szükséges ahhoz, hogy kommunikáció legyen két állomás közt.
- Nagy sebességű adattovábbítás szomszédos állomások közt.
- Jól skálázható (további állomásokkal történő kiegészítés alig befolyásolja a hálózati teljesítményt).

¹⁴ [13] (<https://fossbytes.com/what-is-ring-topology-advantages-and-disadvantages-of-ring-topology/>)

¹⁵ Az STP-nek az a célja a hálózatban, hogy az ne tartalmazzon hurkot. Ezt kapcsolatok megszakításával éri el. Meghibásodás esetén a sérült összeköttetést pótolhatja egy blokkolttal.



3-26. ábra: Ring topológia felépítése

Legnagyobb hátránya, hogy habár két szomszédos eszköz között gyors az adatközlés, de egy tokenfoglás-felszabadítás (ami tekinthető egy kommunikációs ciklusnak) már jóval lassabb lehet az egyéb alternatívákhoz viszonyítva (pl. csillag topológia).

A ring topológia alapelve a tokenizálás¹⁶. Ez a protokoll annyit jelent, hogy egy eszköz csak akkor használhatja a hálózatot, ha birtokolja az ehhez való jogot, vagyis a token. Mivel a hálózaton egyszerre csak egy állomás küldhet adatot, így nem fordulhat elő ütközés.

Alapállapotban egy szabad token „kering” a ringben, ami felfogható egy hasznos információval feltölthető keretnek. Adáskéréskor az adó állomás megszerzi a token-t és feltölti adattal.

Aztán az adatot továbbküldi a dedikált irányba a szomszéd állomásnak. Az adatcsomagban szerepel állomásazonosító is, ha ez nem egyezik meg a szomszéd címével, akkor az szintén továbbküldi az ő szomszédjának. Ez addig ismétlődik, míg a célzott állomáshoz nem ér. Az adatot kiolvassa a célállomás.

¹⁶ [14] (<http://www.datacottage.com/nch/troperation.htm#.WDCQL9UrK00>)

Ekkor még nem szabadul fel a token. A célállomás a keretet visszaküldi láttamozással és az üzenő állomás címével. Ez eljut az üzenő állomáshoz a már részletezett módon, majd az a láttamozást kiolvasva felszabadítja a tokent.

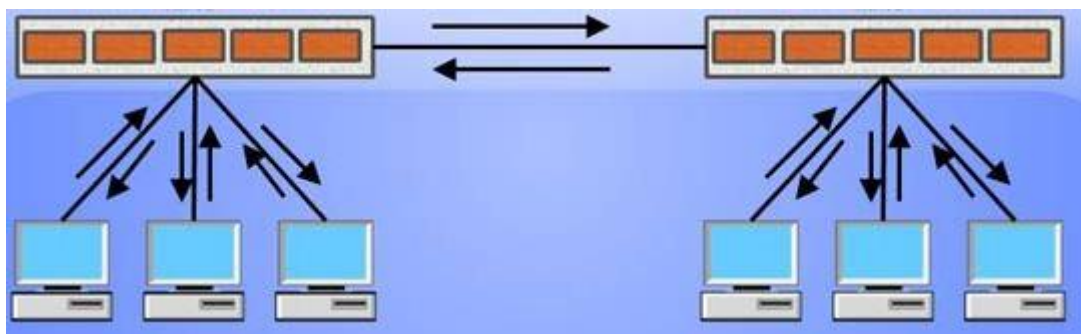
A ring hálózat indulásakor a legnagyobb MAC címmel rendelkező egység megkapja a monitorozó szerepkört. Feladatai:

- Figyelni, hogy eszköz ne okozhasson problémát.
- Hiba után a kommunikáció újrafelvétele.
- Megszakítás (a ring resetelése) kezdeményezése hiba esetén.
- Polling ütemezése.

Polling során az eszközök lementik a megállapodás szerinti ring-iránnyal ellentétes irányba nézve legközelebbi aktív szomszéd címét (nearest active upstream neighbor - NAUN). Ez fontos információ a hálózat újraindítása szempontjából.

Egy új állomás beléptetésénél az adott egység öntesztet végez, és ha saját csatlakozását megfelelően működőnek találja, akkor jelez a relének, ami áramkörileg bekapcsolja a ringbe.

Hiba fellépésének esetén annak helye onnan deríthető ki, hogy valószínűleg az az egység romlott el, mely annak még aktív állomásnak a NAUN-ja, mely először tapasztalta a kommunikáció megszűnését. Ekkor a leállást okozó egység leválasztódik a ringről, de ehhez egy központi egység is szükséges.

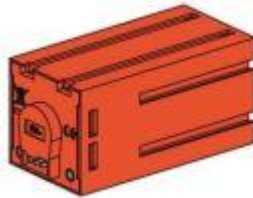


3-27. ábra: HUB-vezérelt ring

A központi vezérlővel rendelkező ring ötvözi a ring- és csillag topológiák előnyös tulajdonságait. A tokenizálás folyamata változatlan marad, de megnövekedett a megbízhatóság, mivel a hibás egységet egy ipari switch automatikusan lekapcsolhatja a hálózatról.

3.7 A gyártósor állomások működése és szenzorjai

3.7.1 Az alkalmazott szenzorok és aktuátorok

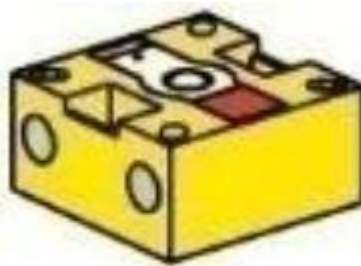


3-28. ábra A robotkaron használt DC motor

A robotkar horizontális és vertikális helyzetváltoztatásáért egy-egy enkóderes villanymotor felel. Ezek állandómágneses DC motorok, melyek képesek a szögváltoztatásuk követésére (QU-EN jel, a tárcsamozgást Hall-effektus szenzor érzékeli).

24V a működési feszültsége, a maximális teljesítmény leadás 2.03W 214fordulat/perc forgási frekvencián, ekkor az áramfelvétel 320mA. A motortengely/kimeneti-tengely áttételezése 25:1-hez, tehát az enkóder egy motorfordulat során 3 impulzust generál (vagy 75 impulzust egy kimenet-tengely fordulat alatt).

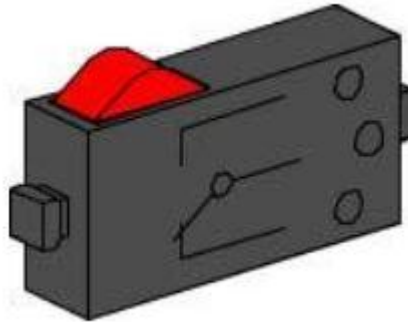
Mivel a motor két egymáshoz képest eltolt jelet bocsát ki (QU-EN), ezért képesek vagyunk megkülönböztetni a forgási irányokat.



3-29. ábra Fototranzisztor

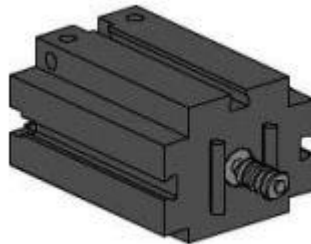
Fénykapu több helyen is található a gyártósorban. Működése közben a fototranzisztor elektromos áramot bocsát ki bizonyos erősségű fény hatására. Egy bizonyos fényerősség felett a fototranzisztor elveszítheti vezetőképességét. Ha a

fototranzisztorral szemben egy kis szórású fényforrást helyezünk el, akkor az megbízhatóan szolgáltat áramot, így felhasználható fénykapuként.



3-30. ábra: Kapcsoló

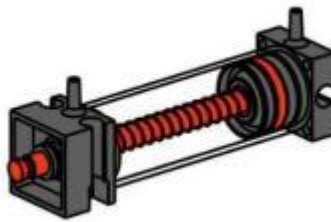
A kapcsolók referencia- (pl. forgótálca pozíció) és végálláskapcsolóként¹⁷ (pl. tolólapok) is felhasználásra kerültek.



3-31. ábra: DC motor

Állandómágneses egyenáramú motor, melyre kiegészítő áttételezés is könnyedén szerelhető. 24V-on üzemel, maximális áramfelvétele 300mA, maximális forgatónyomatéke 5mNm. A szállítószalagok és a forgótábla mozgására van használva.

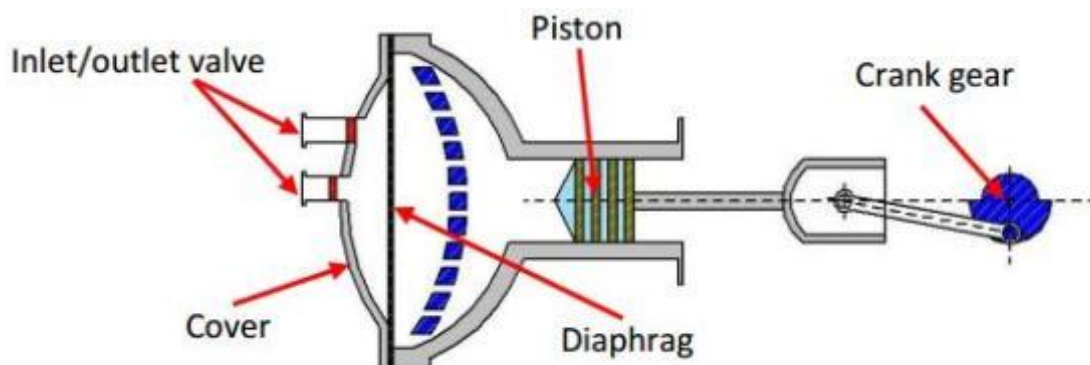
¹⁷ később használt rövidítése: VÁK



3-32. ábra: Pneumatikus henger

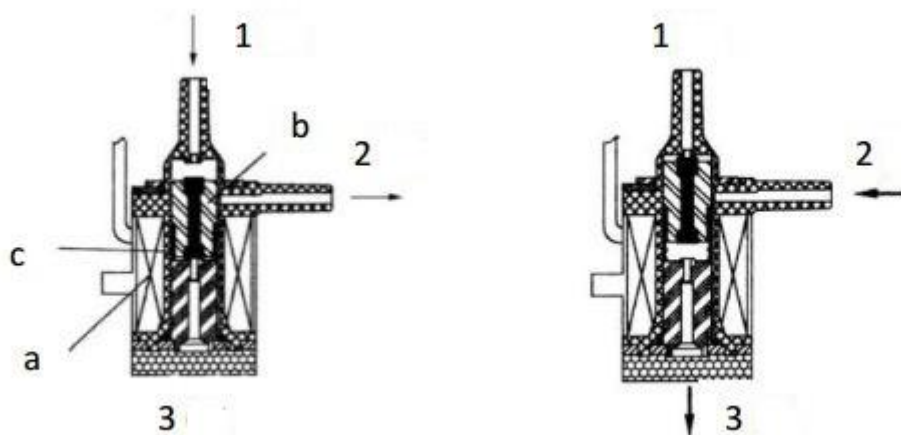
Ezt az eszközt a pneumatikus állomás belökője és présgépje használja. Egy dugattyú kettéosztja a henger térfogatát két kamrára. A két kamra nyomásának különbsége erőt fejt ki a hengerre, ami ezáltal elmozdul.

Ha a hengerre csökkenő nyomást helyezünk (kiszivattyúzzuk a levegőt), akkor a dugattyú rúdja a rúgóerőt leküzdve elmozdul a hengerben úgy, hogy a lezárt kamra térfogata ezzel együtt nőni fog. A kiszivattyúzás megszűnésével a rúgó a kezdeti állapotba löki vissza a dugattyút.



3-33. ábra: A kompresszor sematikus felépítése

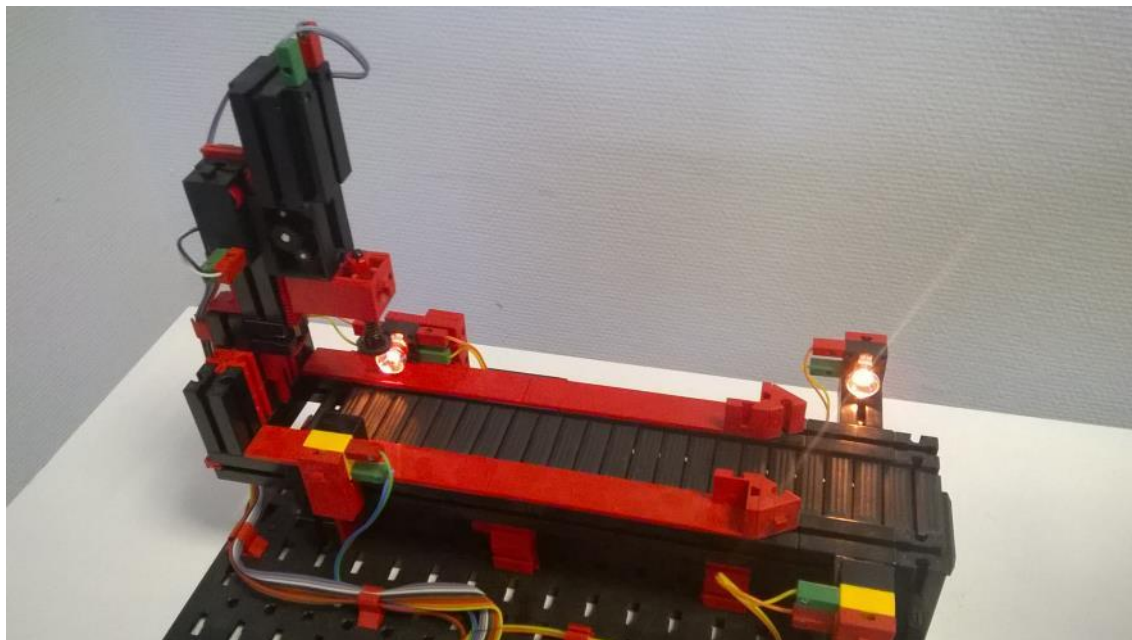
Kompresszor látja el a levegősűrités feladatát a pneumatikus állomásban. A kamráját egy membrán választja ketté. Egy vezérmű-tengely mozgat egy dugattyút az egyik kamrában, ami a levegőt beszippantja vagy kinyomja a másik kamrában. A dugattyú jobbra mozgása esetén a membrán követi azt, és levegő áramlik be a második kamrába a bemeneti szelepen keresztül. Ellenkező irányú elmozdulása esetén levegő hagyja el a kamrát a kimeneti szelepen keresztül. Maximális létrehozható nyomás 0.7bar.



3-34. ábra: A mágneses szelep működése

Mágneses szelepekkel történik a pneumatikus hengerek vezérlése. A kapcsolásokért egy tekercs (a) felel, mely egy rúgó (c) ellen fejt ki hatását. Mikor feszültség jut a tekercsre, akkor a tekercs mozgó magja (b) a Lorentz-erő hatására a rúgóerővel szemben elmozdul, és a szelep így kinyílt. A feszültség esésekor a Lorentz-erő megszűnik, és a rúgó visszalöki a tekercsmagot a kiindulási helyére.

3.7.2 Présgép (punching)

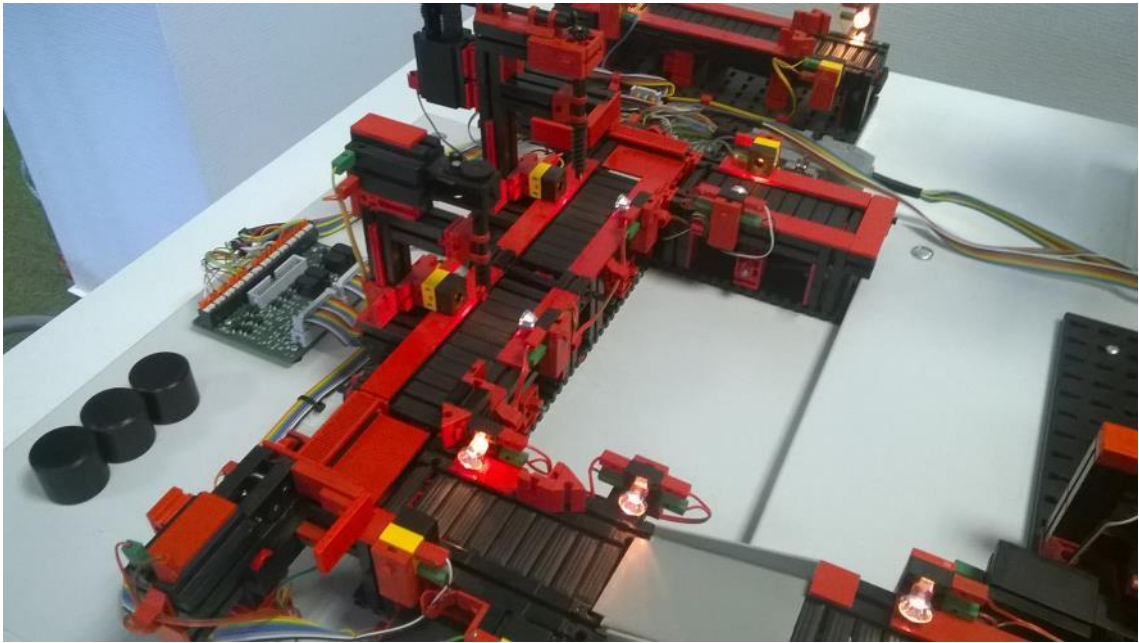


3-35. ábra: Présgép állomás

A bemenetére a robotkar fogja letenni a terméket, amit a bemeneti fénykapu jelez. A szállítószalag beviszi a terméket a préshez, majd az elvégzi a préselést. A gép

végállásainál kapcsolók vannak. Végeztével a szállítószalag visszaviszi a terméket a bemenethez, majd onnan a robotkar továbbítja.

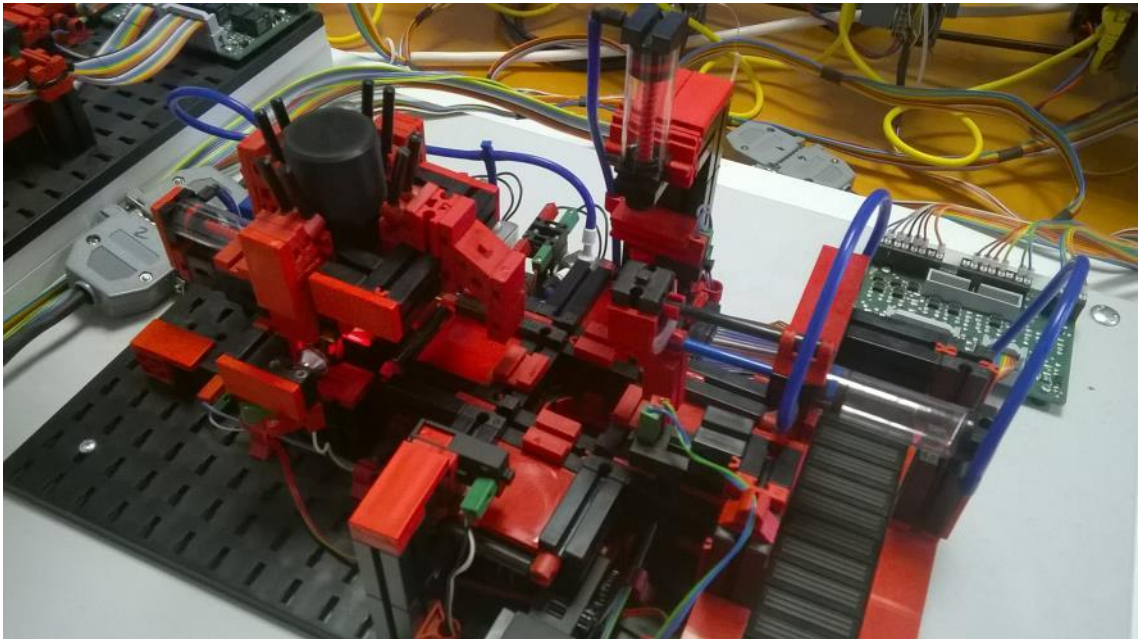
3.7.3 Megmunkáló állomás (machining)



3-36. ábra: Megmunkáló állomás

A kép alján látható rámpán lecsúszó terméket a bemeneti fénykapu érzékeli, majd a szalag elviszi az első tolólap elé. A tolólap rátolja a terméket a marófej szalagjára. A marófejnél lévő fénykapu jelzésénél megáll a szalag és a marás elkezdődik. Végeztével továbbindul a szintúgy fénykapuval ellátott fűrőfejhez. A fűrés befejeztével a szalag a második tolólaphoz viszi a terméket, majd a kiviteli szalag elviszi a tolólaptól. Innen a robotkar viszi tovább a vezérléstől függően.

3.7.4 Pneumatikus állomás



3-37. ábra: Pneumatikus állomás

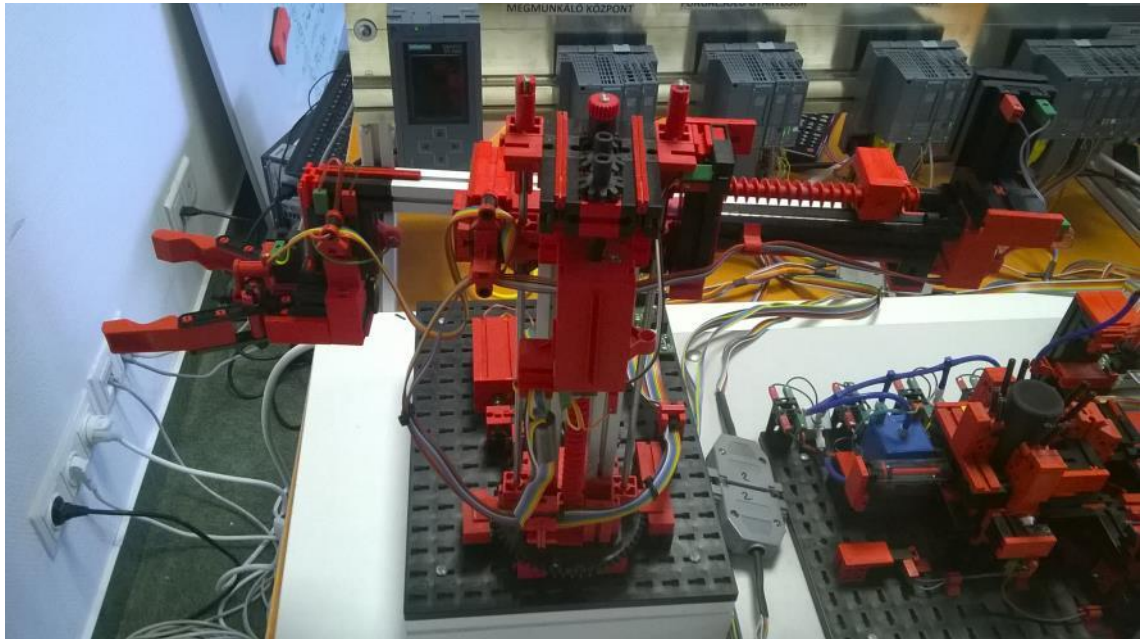
A pneumatikus állomás bemenete (ami egyben az egész gyártási folyamat kezdete is lesz) egy tárolótölcsér, amiben 3-4 termék tud elhelyezkedni egymáson. Egy fényszenzor van a tölcsér alján, mely jelzi, hogy van-e termék benne.

Az oszlopban lévő legalsó terméket egy pneumatikus henger a forgótálca egy szabad tálcájára lök. Arról, hogy a forgótálca pozícióban van az összes állomásnál, arról egy referenciakapcsoló ad információt.

A forgó tálca a terméket a préshez viszi, ahol egy másik referenciakapcsoló jeléből tudható, hogy a présnél tartó tálca üres vagy sem. A préselést egy újabb pneumatikus henger végzi.

Ezután a forgótálca a kilököhöz viszi a terméket, ahol egy harmadik referenciakapcsoló segítségével tudhatunk arról, hogy a termék ide érkezett. A pneumatikus tolólapát rátolja a kimeneti szállítószalagra a terméket, majd az a megmunkáló-állomás rámpájához viszi azt.

3.7.5 Robotkar



3-38. ábra: Robotkar kiindulási állapotban

A balra-jobbra fordulás és a magasságváltoztatás mikéntjéről már volt szó. Ezeken kívül még két helyzetváltoztatás lehetséges, a forgási tengelyre merőleges irányú karmozgatás (betolás), valamint a markoló nyitása-zárása.

Ez a kettő mozgás csigahajtással történik. Tengelyükre egy referenciakapcsoló úgy van felszerelve, hogy azt minden egyes csigamenet bekapcsolja. Így a QU-EN-hez hasonló impulzussorozatot kap a vezérlő, amiből a pozícióra lehet következtetni, a különbség csupán annyi, hogy itt csak egy „tárcsa” produkál jelet.

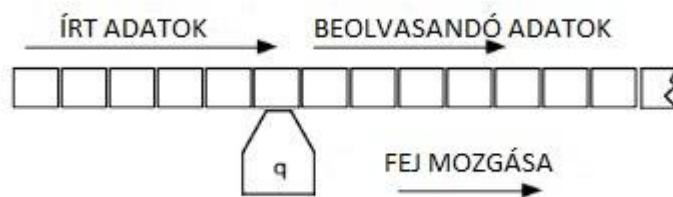
A csigamenet impulzusait nem HSC dolgozza fel, a váltási frekvenciájuk megengedi, hogy jeleiket a PLC szoftveres úton kezelje.

Mind a négy helyzeti változó (magasság, elfordulás, kar előre-hátra, markoló) rendelkezik egy-egy végálláskapcsolóval. Minden munkavégzés első lépésének annak kell lenni, hogy a négy változót a végálláskapcsolókig kell vezérelni, majd az őket jellemző szám nullázódik.

3.8 Véges állapotú állapotgép (finite state automaton - FSA)¹⁸

A vezérlés tervezése során egyértelművé vált, hogy az elkészült szoftver ezer nagyságrendű kódsort fog tartalmazni. A cél az, hogy a szoftvert könnyen át lehessen látni, bővíteni (például a kommunikáció implementációjánál többször változtatni kellett a már elkészült blokkokat). A fejlesztés kézbentartásának egyik hatékony módja, hogy egy program-modellt követ a tervezés. Ebben a projektben a véges állapotú állapotgép modellje került alkalmazásra.

A matematikai leírások során úgy szokás jellemezni az FSA-kat, mint csökkentett képességű Turing-gépeket. A Turing-automata képletes leírása egy olyan véges állapotú vezérlő, mely rendelkezik egy bármely irányba mozgatható író/olvasófejjel, ami egy végtelen mágnes-szalagon tárolt adatokat olvassa és írja. Ha meg van szabva, hogy a fej csak egy irányba mozoghat, akkor az FSA képletes definíciója az eredmény.



3-39. ábra: Az FSA modellje, egy korlátozott Turing-gép

Tehát az FSA beolvassa az adatfolyamot a mágnes-szalagról, a belső logikája egy adott állapotba viszi az FSA-t, majd a mágnes-szalagra kimásolja kimenetét. Matematikailag nem szükséges a szalag-metafóra támaszkodni, nézhető úgy is, hogy a ki- és bemeneten időben változó események történnek. Figyelembe véve, hogy a fej csak egy irányba mozoghat, akkor a modellt átalakíthatjuk egy fekete dobozzá, amely bemeneti sorozat hatására egy kimeneti sorozatot generál.

Az így definiált modellnek két típusa van:

- Mealy-automata: Az adott bemeneti szekvenciát leképezi egy kimeneti szekvenciára. Ez az általános modell.

¹⁸ [15] (Introduction to the theory of Computation)

- Moore-automata: A bemeneti szekvenciája megváltoztatja belső állapotot, de a kimeneti értékek csupán ettől a belső állapottól függ. Ez egy egyszerűbb modell.

Az implementáció törekszik a Moore-automata kereteinek betartására, legfőképp mert a Mealy-hez képest lényegesen könnyebben szervezhető a kimenetek beállítási.

A következő program egy bináris adatfolyamot olvas, és egy bináris adatfolyamot ír. A kimenetnek 0-nak kell lennie, kivéve ha a bemeneten változott az érték (0-1 vagy 1-0 váltás).

Az alábbi ciklus ismétlődik:

1. Karakter beolvasása
2. Következő állapot kiválasztása
3. Megfelelő karakter írása a kimenetre
4. Következő állapot kiválasztása

Itt az állapotokat szubrutinokkal írják le. A kezelhető karakterfolyam variációk számának és az alkalmazott aritmetikának végesnek kell lennie.

A program az **f()** szubrutin hívásával kezdődik. Tegyük fel, hogy a **read()** szubrutin a következő karaktert adja vissza a bemeneti karakterfolyamból, a **write(c)** pedig a 'c' karaktert teszi ki a kimeneti adatfolyamra.

```
void f() //Az első hívandó szubrutin
{
    switch( read() )
    {
        case '0': write('0'); g(); break;
        case '1': write('0'); h(); break;
    }
}

void g() //meghívódik, ha az előző beolvasott karakter '0' volt
{
    switch( read() )
    {
        //változatlan bemenet esetén megint ez a fgv hívódik
        case '0': write('0'); g(); break;

        //változott a bemenet, 1-et írunk kimenetre
        case '1': write('1'); h(); break;
    }
}

void h() //meghívódik, ha az előző beolvasott karakter '1' volt
```

```

{
    switch( read() )
    {
        //változott a bemenet, 1-et írunk kimenetre
        case '0': write('1'); g(); break;

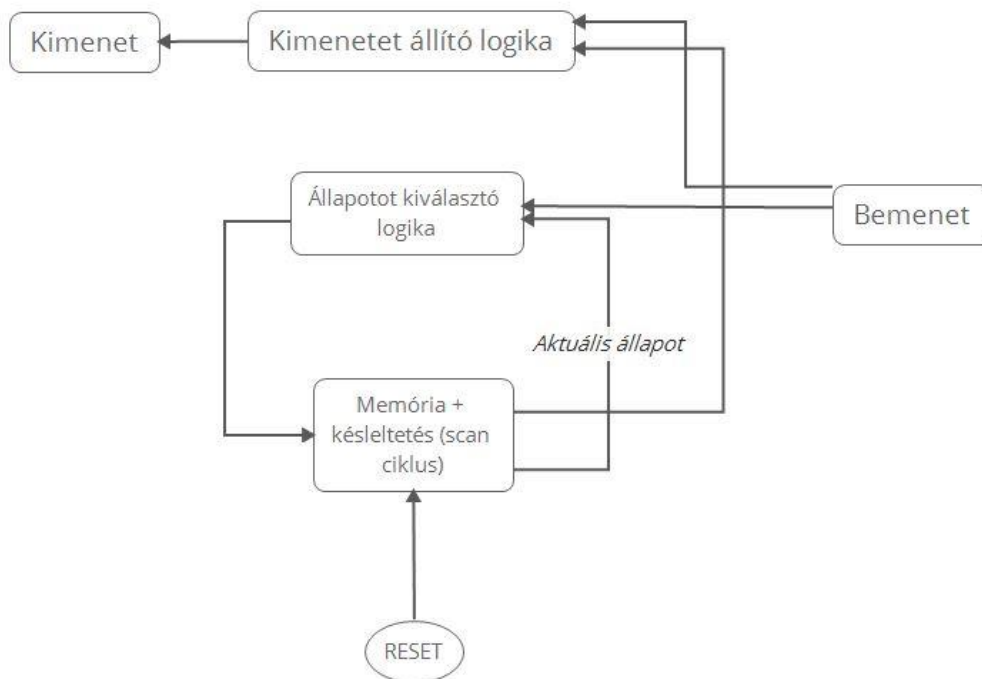
        //változatlan bemenet esetén megint ez a fgv hívódik
        case '1': write('0'); h(); break;
    }
}

```

A példában jól látszik, hogy az FSA egyszerű switch-case szerkezettel megvalósítható. Az SCL is támogatja a switch-case-t, ez volt a legerősebb indok az SCL mellett.

A példából két tanulság vonható le, amire feltétlenül ügyelni kell a hardverközeli SCL használata közben:

- Szükség van egy előre definiált kiindulási állapotra, ahová külső jel hatására bármikor tud a vezérlés lépni. A PLC-s környezetben ez azt jelenti, hogy a kiindulási állapotban minden kimenetet ki kell kapcsolni, minden változót inicializálni kell, és egy start-jelre hagyható el ez az állapot. A vezérlés kiindulási állapotba állítását egy egyszerű nyomógomb aktív jele szolgálta.
- Az állapotváltásokat teljes mértékben kézben kell tartani. Az állapotváltásokért felelős logikának minden előforduló változókombinációt kezelnie kell, elkerülendő egy nem-tervezett állapot fellépését.



3-40. ábra: Visszacsatolt hálózat

Az előbbieken részletezett FSA alkalmazását PLC-s környezetben a visszacsatolt modell jellemzi.

A már említett okok miatt az állapot- és kimenetváltó logika a reset megjelenéséig nem lehet aktív (a memóriaterület, ahol az állapotadatok tárolva vannak, nincs inicializálva). Reset hatására az inicializáló értékek betöltődnek a memóriába. Az előző scan-ben felvett állapot és a bemenet alapján lép a következő állapotokra. Az általános blokkvázlaton látható, hogy a bemenet közvetlenül hatással van a kimeneti logikára (tehát egy Mealy-automata).

4 A tervezés lépései

4.1 A présgép alapprogramja

A következő táblázat a blokk interfészét mutatja be:

BEMENETEK	típus	komment
reset	bool	Kezdeti állapot felvétele
conveyor_sensor	bool	Bemeneti fénykapu
machine_sensor	bool	Présgép fénykapu
is_machine_up	bool	Présgép felső végálláskapcsoló
is_machine_down	bool	Présgép alsó végálláskapcsoló

4-1. táblázat: Présgép bemenetei

KIMENETEK	típus	komment
machine_up	bool	Présgép fel
machine_down	bool	Présgép le
conveyor_in	bool	Szállítószalag a présgép felé
conveyor_out	bool	Szállítószalag a bemenet felé

4-2. táblázat: Présgép kimenetei

LOKÁLIS VÁLTOZÓK	típus	komment
state	integer	A vezérlés állapota
x	integer	Préselés számlálója
conv_wait	bool	TON időzítő kimenete

4-3. táblázat: Présgép statikus változói

Reset érkezésénél a kimenetek kikapcsolódnak, a lokális változók inicializálódnak. Induláskor minden géprésznek alaphelyzetben kell lennie. A szállítószalagnak csak a kimenetét kell 0-re állítani, a présgépnél pedig felső állásban kell tartani a prést. Ezt a következőképpen valósítható meg:

```
IF #is_machine_up = 1 THEN
    #machine_up := 0;
    #machine_down := 0;
ELSE
    #machine_up := 1;
    #machine_down := 0;
END_IF;
```

Tehát amíg a prés nem éri el a felső végállaskapcsolót, addig felfelé irányítódik, egyébként pedig leáll a mozgása.

Ha a bemeneti fénykapu terméket érzékel, akkor megindul a szállítószalag a prés felé. Megvalósítása:

```
IF #state = 1 AND NOT #conveyor_sensor THEN
    state := 2;
END_IF;
```

Észrevehető, hogy a bemeneti fényszenzor nulla értékére lép a következő állapotba. Ennek oka az, hogy a fototranzisztor működési elve alapján amikor érzékel fényt, akkor aktív, egyébként pedig nullát ad (tehát negatív logikát követ).

A szállítószalag leáll, mikor a prés fénykapuja jelez, ez a fenti kódrészlethez hasonlóan van kezelve, ügyelve a negatív logikára.

A prés leereszkedik, amit az alaphelyzetnél használt kódhoz hasonlóan oldható meg:

```
IF #is_machine_down THEN  
    #machine_down := 0;  
ELSE  
    #machine_down := 1;  
END_IF;
```

A prés felemelkedik, ami a fenti kóddal megoldható, csupán az ellenkező irány változóját kell állítani. A prés felvitele után ellenőrizve van, hogy kell-e még a prést leereszteni. Ha kell, akkor ahhoz az állapothoz tér vissza, mely leengedte a prést.

Az ellenőrzés úgy történik, hogy a program elején letárolt préselés számlálója ennél a pontnál csökken 1-el. Ha az eredmény nulla, akkor mehet tovább a vezérlés, egyébként még préselni kell:

```
IF #state = 4 AND #is_machine_up THEN  
    #x := #x - 1;  
    IF #x = 0 THEN  
        #state := 5; //Végeztünk a préseléssel, köv. állapot  
    ELSE  
        state := 3; //Különbén visszatérünk a préseléshez  
    END_IF;  
END_IF;
```

A préselés végeztével kifelé indul a szalag. A bemeneti fénykapu érzékenysége miatt az már akkor jelzi a kifelé tartó terméket, mikor még nem érte el a szalag szélét. Ez a robot szempontjából fontos, mivel a szalag szélén jóval több helye lesz a manőverekre. Ezért a fénykapu jelzésénél még nem áll le a szalag, hanem egy időzítő indul:

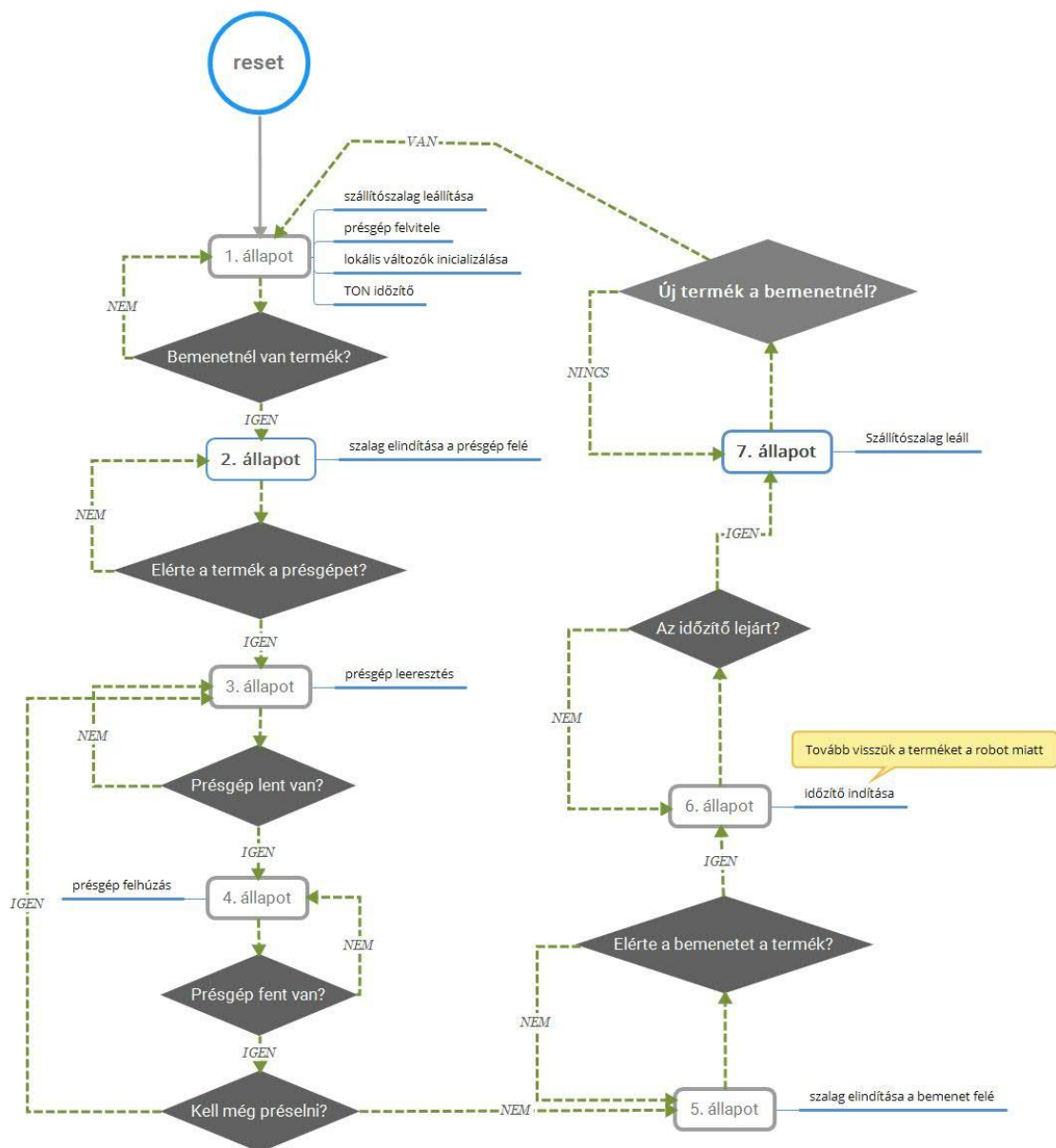
```
''punching_machine_timer_DB''.TON(IN := 1; PT := T#350MS,  
    Q => #conv_wait);
```

A TON timer a bemenetének (IN) aktiválása után PT^{19} idővel (ami most 350ms, kb. ennyi idő kell a fénykapu jelzésétől a szalag szélének eléréséig) ad a kimenetére (Q) jelet, amit a conv_wait lokális változóban tárolódik. Az időzítő lejártával a szalag leáll, majd egy új termék esetén újra kezdőállapotba lép.

Figyelmes olvasással rá lehet jönni, hogy van egy logikai csúsztatás a leírásban. Az utolsó állapotból akkor lép az elsőbe, ha új termék kerül a bemenetre, amit szintén a fénykapu fog jelezni. De honnan tudható, hogy a fénykapu nem az előbb kivitt terméket jelzi, hanem egy másikat? Ezt már a kommunikáció fejezete fogja tárgyalni.

A következő ábrán a vezérlés gráfos (flow-chart) leírását mutatja be. Ezen akár nagyobb terjedelmű szoftverek működése is könnyedén átlátható.

¹⁹ preset time



4-1 ábra: A présgép állapotai

4.2 A megmunkáló állomás alapprogramja

A megmunkáló állomás jóval több funkcióval rendelkezik a présgéphez képest, ezért az állomást 4 logikai alegységre van bontva, amit egy-egy FSA irányít.

4.2.1 Bemenet alegység

<i>BEMENET</i>	<i>típus</i>	<i>komment</i>
<i>input_sensor</i>	bool	Bemenet fénykapu
<i>push1_sensor</i>	bool	Bemeneti tolólap

		fénykapuja
<i>push1_forward_sensor</i>	bool	Bemeneti tolólap hátsó kapcsoló
<i>push1_backward_sensor</i>	bool	Bemeneti tolólap első kapcsoló
<i>reset</i>	bool	Kezdeti állapot felvétele
KIMENET		
<i>input_conveyor</i>	bool	Bemeneti szállítószalag
<i>push1_forward</i>	bool	Tolólap előre
<i>push1_backward</i>	bool	Tolólap hátra
LOKÁLIS		
<i>input_module_STATE</i>	integer	Állapot
<i>input_module_DONE</i>	bool	A bemenet végzett a termékkel
<i>milling_module_BUSY</i>	bool	Marófejnél termék van
<i>input_timer_up</i>	bool	TON időzítő kimenete

4-4. táblázat: Bemenet alegység interfésze

A reset jel utáni eljárás minden modulnál meg fog egyezni, tehát itt is a változók inicializálódnak és a gépeket alaphelyzetbe kell vinni. Itt ez annyit jelent, hogy a tolólapot hátravitt helyzetbe kell állítani. Kezelése megegyezik a prés kezelésével:

```

IF #push1_backward_sensor THEN
    #push1_backward := 0;
ELSE
    #push1_backward := 1;
END_IF;

```

Amikor a bemenetnél lévő rámpán lecsúszik a következő termék, azt a bemeneti fénykapu érzékeli, és indul a szalag.

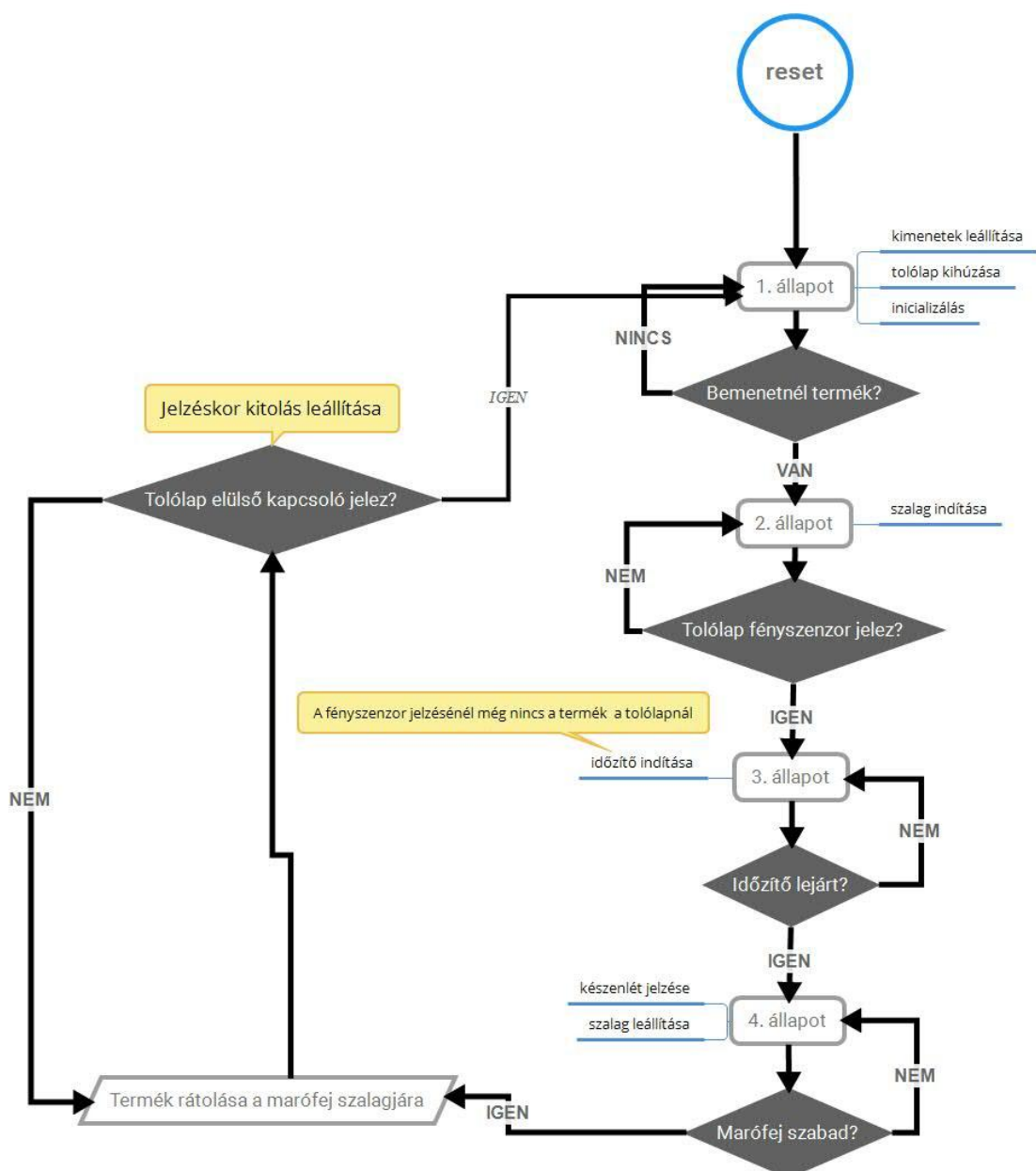
A tolólaponnál lévő fényszenzor jelzésénél még nem áll le a szalag, mivel ekkor még nincs teljes terjedelmében a tolólap előtt. Ekkor egy időzítő indul, hogy biztosan a tolólap elé érjen a termék, lejártakor leáll a szalag. Az inicializálás gondoskodott arról, hogy a tolólap ekkor biztosan ne legyen útban.

Az időzítő lejártakor figyeli a következő alegység, a marófej állapotát. Csak az érdemi információ a vezérlésnek, hogy a marófejnél van-e jelenleg termék. A szoftver szempontjából lényegtelen, hogy figyel-e a marófejre, de a szállítószalag mechanikája miatt már lényeges.

Ha a marófejnél termék van, akkor nagy valószínűséggel az megmunkálás alatt áll, tehát a marófej szállítószalagja nem mozog. Álló szalagra nem tanácsos tolólappal terméket tolni, mert előfordulhat, hogy a tolólap előtt épp két szalag-elem közti hézag van (a tengelyeken történő átfordulásakor jönnek létre). Ekkor a termék beleakad ebbe a hézagba, ami a szalag és a tolólap tönkremenetelét eredményezheti.

Az időzítő lejártakor a készenlétet jelző `input_module_DONE` értéke is 1-re állítódik, aminek hatására a szabad maróegység elindítja a szalagját, így elkerülhető a fent leírt meghibásodás.

Ha a `milling_module_BUSY` nulla értéket vesz fel, akkor a tolólap előretolódik. Az elülső végálláskapcsoló jelzésénél megáll, majd visszatér a kezdeti állapotba.



4-2. ábra Megmunkálóállomás bemenetének állapotai

4.2.2 Marófej alegység

<i>BEMENET</i>	<i>típus</i>	<i>komment</i>
milling_sensor	bool	Marófej fénykapu
reset	bool	Kiindulási állapot
<i>KIMENET</i>		
milling_conveyor	bool	Marófej szállítószalag

milling	bool	Marófej forgás
<i>LOKÁLIS</i>		
milling_module_BUSY	bool	Marófejnél termék van
milling_module_STATE	integer	Marófej állapota
milling_module_DONE	bool	Marófej végzett a termékkel
milling_timer_up	bool	TON időzítő kimenete
input_module_DONE	bool	A bemenet végzett a termékkel
drilling_module_BUSY	bool	Fúrófejnél termék van

4-5. táblázat: Marófej interfésze

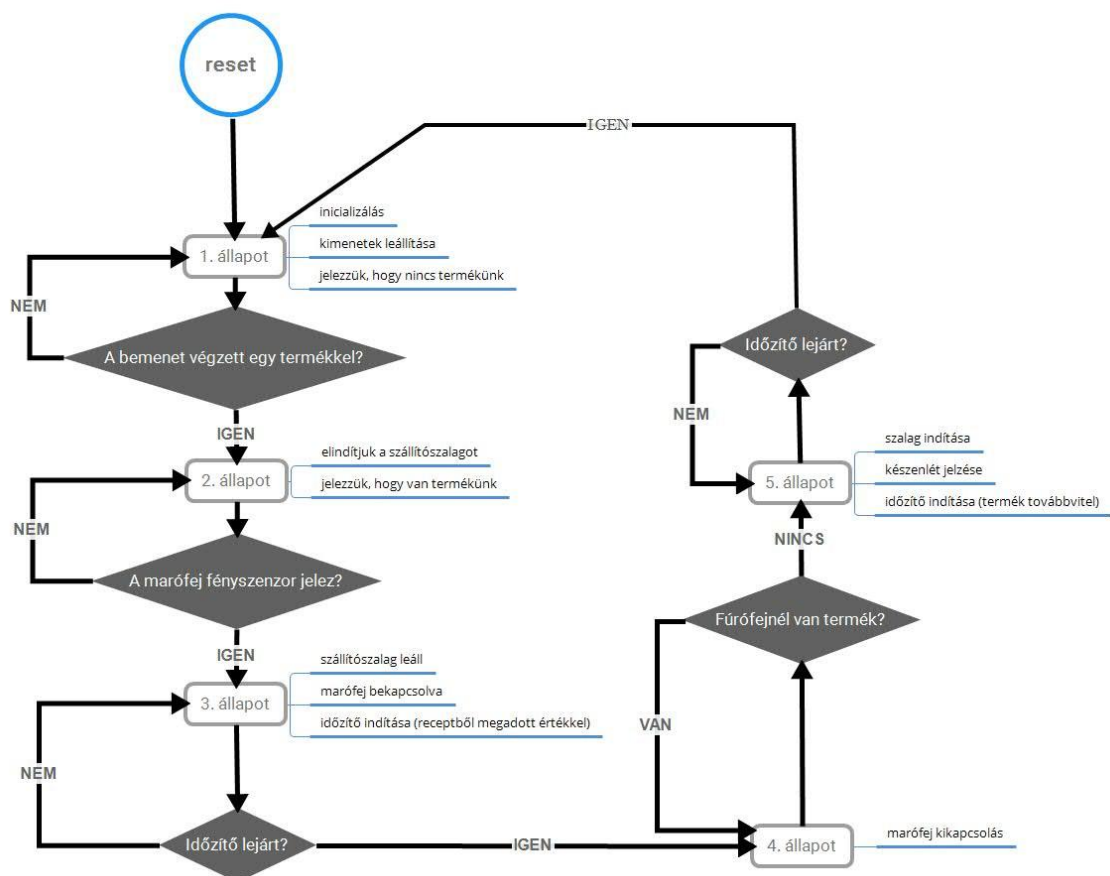
Reset hatására inicializálódnak a változók, leáll a kimenet. Az előző esetekben előforduló alapállapotba teendő gép itt nincs: a marófej és a szállítószalag áll le, valamint jelez, hogy nincs termék.

Amint jelet kapu a bemenet végeztéről, akkor elindul a szállítószalag, ami leáll a marófej fényszenzorának jelzésére, valamint jelez, hogy termék van az alegységnél.

Elindul a marás, és egy TON időzítő (ekkor a PT-t egy receptből kapja). Az időzítő lejártával leáll a marás, és újrainicializálódik az időzítő, mivel ebben a ciklusban még egyszer szükség lesz rá. Várakozik addig, amíg a fúrófejnél nem lesz termék.

Ha bekövetkezik, akkor egy új PT-vel újraindul az időzítő, és elindul a szalag, amivel átküldi a terméket a fúró szalagjára. Az időzítő lejártakor visszalép a kiindulási állapotba.

Azért van szükség a továbbítás ilyen megoldására, mert nem lehet szenzorral érzékelni, ha átért a fúró szalagjára a termék, ezért olyan időintervallumot használ a vezérlés, ami alatt nagy valószínűséggel sikeresen átjut. A bemenet tolólapjánál ugyanez a logika volt követve az ottani időzítővel.



4-3. ábra: Marófej alegység állapotai

4.2.3 Fúrófej alegység

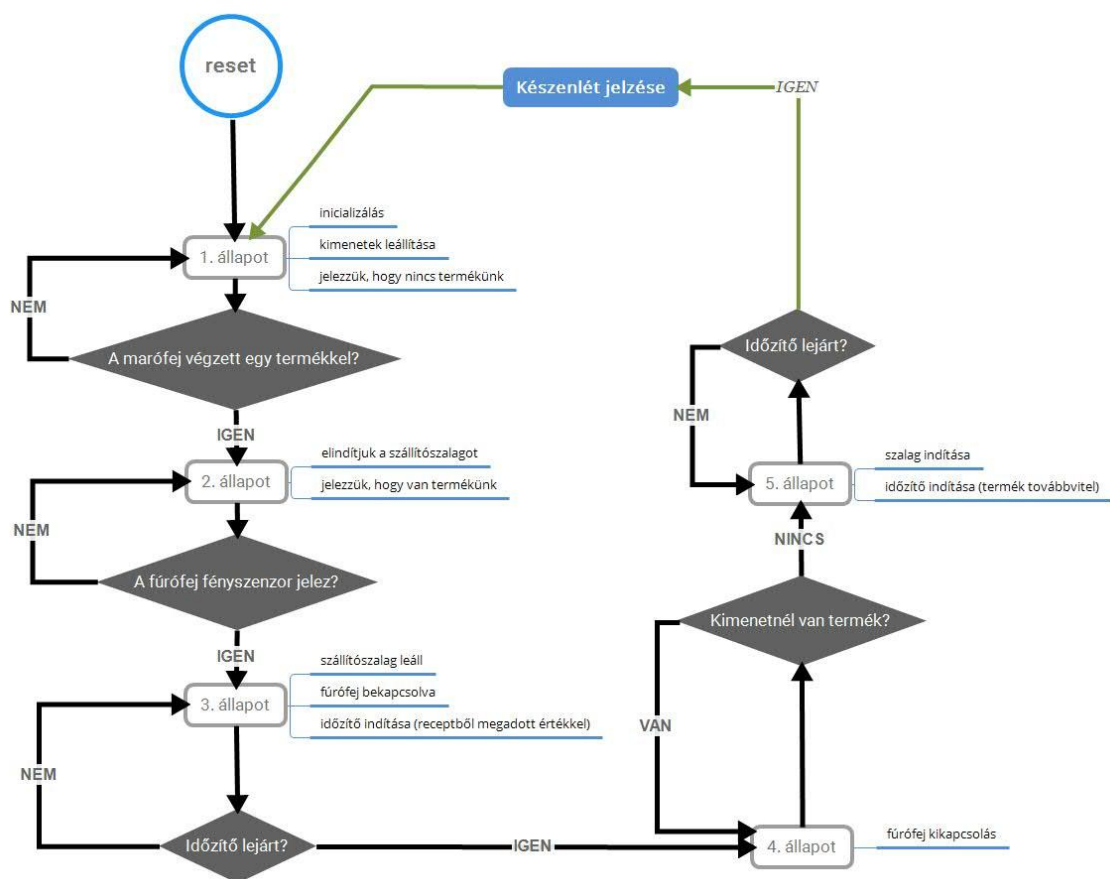
BEMENET	típus	komment
reset	bool	Kezdőállapotba váltás
drilling_sensor	bool	Fúró fényszensor
KIMENET		
drilling	bool	Fúrófej forgás
drilling_conveyor	bool	Fúró szállítószalag
LOKÁLIS		
milling_module_DONE	bool	A maró végzett egy termékkel
drilling_module_STATE	integer	Fúróegység állapota
drilling_module_DONE	bool	Fúróegység végzett egy

		termékkel
drilling_module_BUSY	bool	Fúróegységnél termék van
output_module_BUSY	bool	Kimenetnél termék van
drilling_timer_up	bool	Időzítő kimenete

4-6. táblázat: Fúró interfésze

A fúró alegység logikája teljesen megegyezik a marófej alegység logikájával, különbség csak a fenti interfész deklarálásában van.

Hardver miatt adódó különbség: Mivel a fúró szalagja nem egy másik szalagra viszi a terméket, hanem a kimeneti tolólap elé, ezért készenlétet csak az átvitel végeztével jelezhet.



4-4. ábra: Fúrófej alegység állapotai

4.2.4 Kimenet alegység

<i>BEMENET</i>	<i>típus</i>	<i>komment</i>
<i>push2_sensor</i>	bool	Kimenet fénykapu
<i>push2_forward_sensor</i>	bool	Kimeneti tolólap első kapcsoló
<i>push2_backward_sensor</i>	bool	Kimeneti tolólap hátsó kapcsoló
<i>reset</i>	bool	Kezdeti állapot felvétele
KIMENET		
<i>output_conveyor</i>	bool	Kimeneti szállítószalag
<i>push2_forward</i>	bool	Tolólap előre
<i>push2_backward</i>	bool	Tolólap hátra
LOKÁLIS		
<i>output_module_STATE</i>	integer	Állapot
<i>drilling_module_DONE</i>	bool	Fúrófej végzett a termékkel
<i>output_module_BUSY</i>	bool	Kimenetnél termék van
<i>output_timer_up</i>	bool	TON időzítő kimenete

4-7. táblázat: Kimenet alegység interfésze

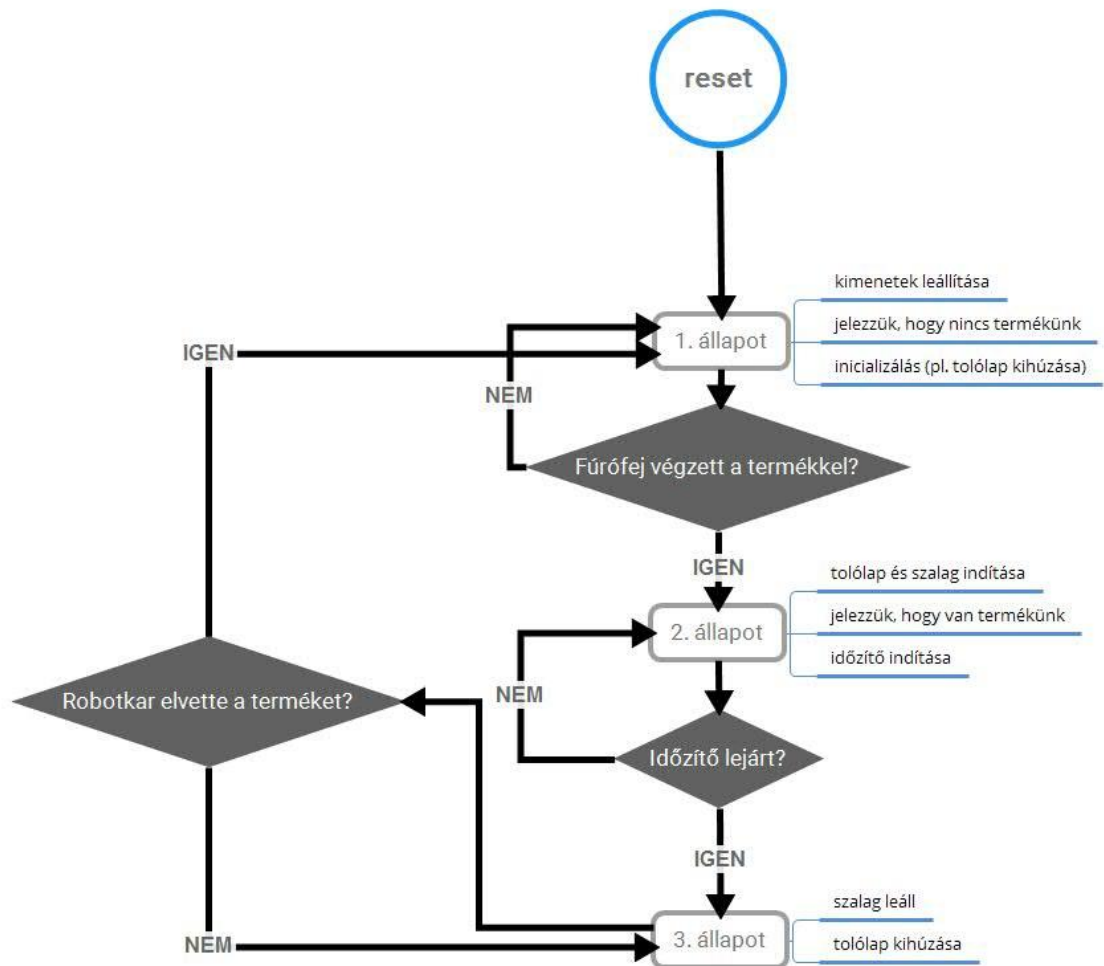
Alapállapotban inicializálás történik, valamint a tolólapot kezdeti állapotba viszi a bemeneti alegységgel megegyező módon. A fúrófej vezérlésénél gondoskodni kell arról, hogy a készenléti jel adásakor már a tolólap előtt legyen a termék.

Ekkor a tolólap előreindul a végállaskapcsolóig, ezzel együtt a kimeneti szalag is indul (ennek oka már részletezve volt). Itt már a robotnak adódik át a termék, így ütközésig kell kivinni a terméket, hogy a robotnak minél több helye legyen a mozgásra.

Mivel a szalag végénél nincs szenzor, ezért megint időzítő alapján vezérlődik a szalag. Az időzítőt a tolólap előreindulásával együtt indítja. (Van egy fényszenzor a

tolólapnál, de ez ebben az esetben nem sok érdemi információt tud nyújtani, de az alarming fejezetnél már ki lesz használva.)

Az időzítő lejártával leáll a szalag, majd mikor a robot elvette a terméket, akkor visszatér a kezdeti állapotba (tehát ennek a működéséhez állomások közti kommunikáció is szükséges).



4-5 ábra: Kimenet alegység állapotai

4.3 A pneumatikus állomás alapprogramja

A pneumatikus állomást is 5 alegységként kezelődik, melyek a belökő, prés, kilökő, forgótálca és szállítószalag egységek.

4.3.1 Belökő alegység

BEMENETEK	típus	komment
input_sensor	bool	Tárolótölcsér fényszenzor
reset	bool	Kezdeti állapot felvétele
position_sensor	bool	Forgótálca referenciakapcsoló
KIMENETEK		
push_to_turntable	bool	Belökés a forgótálcára
LOKÁLIS		
input_wait	bool	Időzítő kimenet
product_cnt	integer	Az állomásban lévő termékek száma
input_state	integer	Bemenet állapota

4-8. táblázat: Belökő egység interfésze

Inicializálás során a termékek száma nullázódik, jelzi a készenlétet (nem lők be terméket), a belökő visszahúzódik. A tölcser-fényszenzor jelzésekor indul az időzítő.

Ha a forgótábla referenciakapcsolója jelez és az időzítő lejárt, akkor belöködik a termék, valamint a termékszámoló növekszik 1-el. A számláló növelésének kódolásánál különösen oda kell figyelni a PLC működésére. A switch-case szerkezet azon része, amit az állapotleíró kijelöl, addig fut, míg az állapot nem változik. Ha tehát a növelés parancsát az adott állapot parancsai között helyeznénk el, akkor annak értéke nem lenne determinisztikus (minden scan-nél lefutna). Tehát ott kell növelni az értéket, ahol garantáltan csak egyszer fog lefutni, és lehetőleg elhelyezkedése közel legyen logikailag a termék behelyezéséhez. Ideális megoldás a kiindulási állapot állapotvezérlője:

```

IF #input_state = 1 AND #position_sensor AND NOT #input_sensor AND
#input_wait THEN

    #input_state := 2; //arra az állapotra váltás ahol a belökés történik

    #product_cnt := #product_cnt + 1;

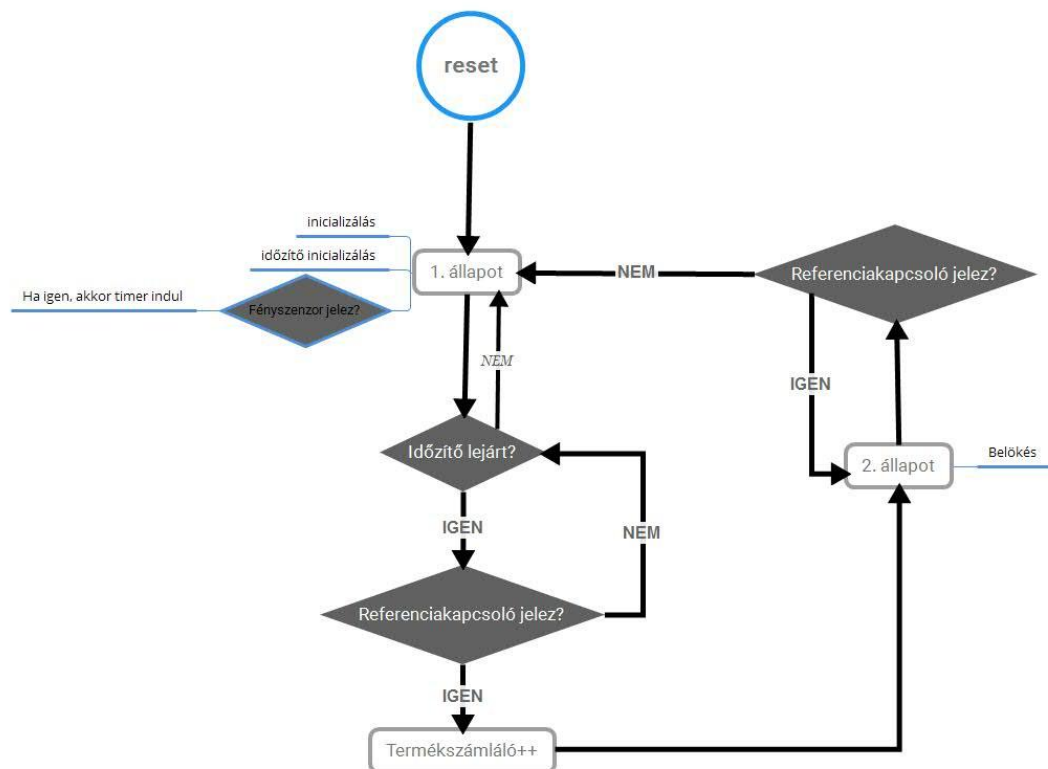
END_IF;

```

Időzítőt azért kell nézni, mert előfordulhat, hogy egy termék egy üres tölcsérbe esne be. Időzítő nélkül a fényszenzor jelzésekor azonnal belökné a terméket, ami nem szerencsés, mivel a terméknek és a tölcsérnek van egy minimális elasztikussága, ezért meg kell várni, hogy a termék felfeküdjön a tölcsér aljára. Nem felfeküdt termék meglökésénél előfordulhat, hogy ferdén érkezik meg a tálcára. A referenciakapcsoló jelzésével tudható, hogy van szabad tálca a belökő, prés és a kilökő előtt.

Jelzi, hogy elvégezte a belökést, majd a referenciakapcsoló nulla értékére a kiinduló állapotba lép.

Ideális esetben a belökés után a tálca azonnal továbbforogna, és ilyenkor a referenciakapcsoló mindig nulla értéket adna. De ha a már előbb belökött termék a forgás miatt elérte a prést, és az dolgozik rajta, akkor a tálca nem foroghat. Ezért lép csak a továbbforgás jelzésére a kiindulási állapotba, mivel csak továbbforgás után lökhető be a következő termék. Így biztosítva van, hogy a következő referenciakapcsoló jelzés egy újabb üres tálcat jelentsen.



4-6 ábra: Belökő állapotai

4.3.2 Prés alegység

BEMENET	típus	komment
punch_sensor	bool	Prés kapcsoló
reset	bool	Kiindulási állapot
KIMENET		
punch	bool	Présfej leeresztése
LOKÁLIS		
punch_OK	bool	Présnél nincs termék
punch_STATE	integer	Prés állapota
punch_wait	bool	Marófej végzett a termékkel
loop	integer	Préselések száma

4-9. táblázat: Prés interfésze

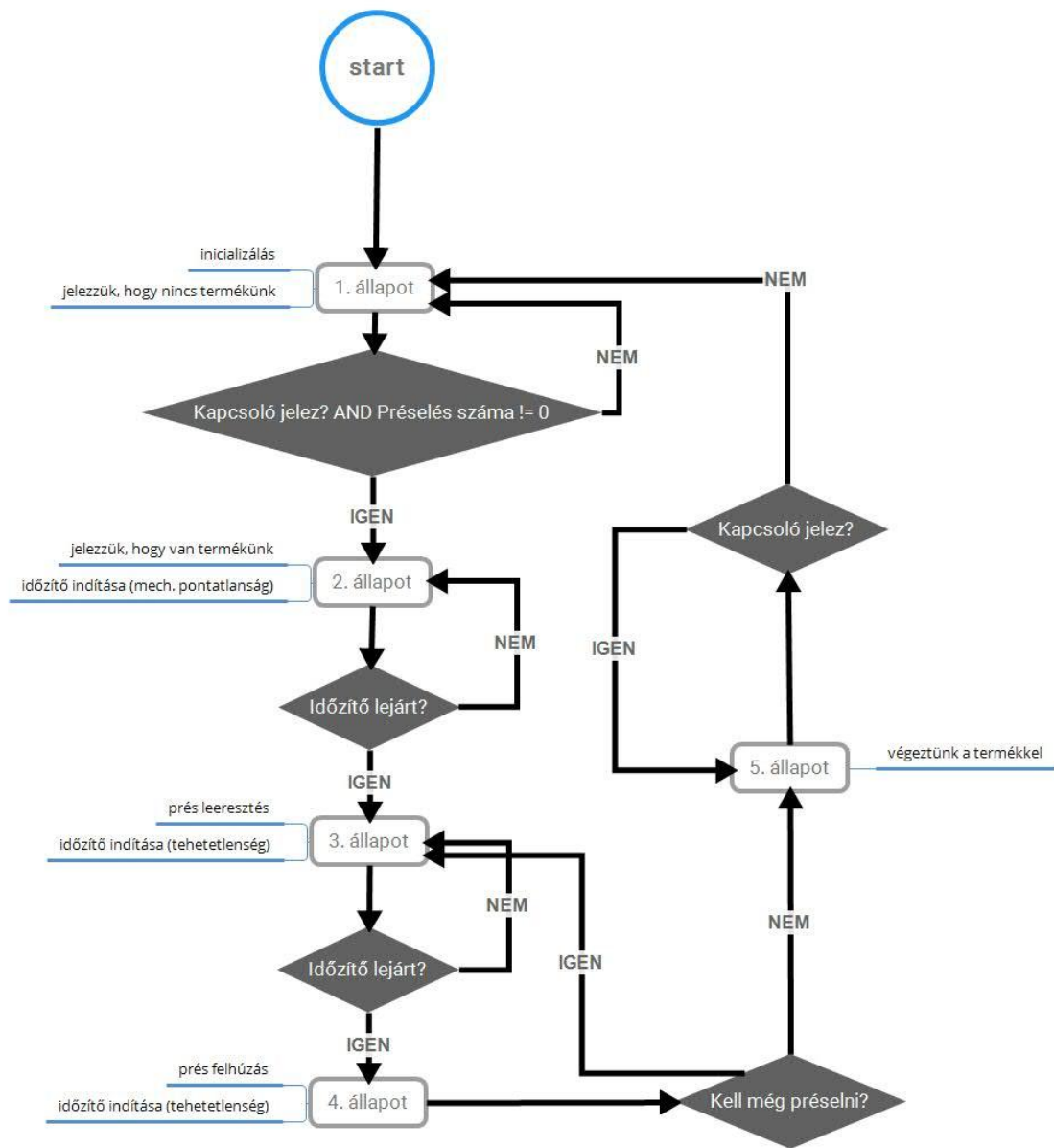
Inicializáláskor a változók inicializálódnak, figyelve arra, hogy a présfejet fel kell emelni, valamint a globális DB-ből (recept) lementődik az elvárt préselések száma.

A présnél lévő referenciakapcsolót figyeli, amit a tálcán lévő termék megnyom, mikor az elérte a prést. Jelzi, hogy van terméke. A referenciakapcsoló jelzésekor azt is ellenőrzi, hogy a beállított préselés-szám nulla-e. Ha nulla, akkor nem engedi elindulni a prést, így azt se jelzi, hogy van termék, tehát nulla beállított préselés esetén nem tartóztatja fel fölöslegesen a forgótáblát.

Ezután időzítő indul, aminek a végén leereszti a prést. Időzítő azért szükséges, mert a referenciakapcsoló jelzésekor még nincs a termék pontosan a présfej alatt. Ezzel a mechanikai pontatlanság ki lett küszöbölve.

Leeresztéskor/felhúzáskor az időzítő újraindul, és lejártakor lehet az ellenkező irányba vezérelni azt. Ennek oka a pneumatikus henger tehetetlensége, tehát idő kell ahhoz, hogy a nyomáskülönbség leküzdje a rúgóerőt (egy scan ideje biztosan nem elég). Felhúzás után ellenőrzi, hogy kell-e még préselni, ha nem, akkor továbbmegy.

Jelzi, hogy végzett, majd a referenciakapcsoló nulla jelzésénél visszatér a kiindulási állapotba (ennek oka megegyezik a belököével).



4-7. ábra: A prégép állapotai

4.3.3 A kilőkő alegység

<i>BEMENET</i>	<i>típus</i>	<i>komment</i>
<i>push_sensor</i>	bool	Kilőkő referenciakapcsoló
<i>reset</i>	bool	Kezdeti állapot felvétele
<i>KIMENET</i>		
<i>push_to_conv</i>	bool	Kilőkés a szállítószalagra
<i>push_to_conv_back</i>	bool	A kilőkő rúd visszahúzása
<i>LOKÁLIS</i>		
<i>output_STATE</i>	integer	Állapot
<i>new_prod_on_conv</i>	bool	A kilőkő terméket tett a szalagra
<i>output_wait</i>	bool	Időzítő kimenete
<i>output_OK</i>	bool	Kilőkőnél nincs termék

4-10. táblázat: Kilőkő interfésze

A resetelés folyamata itt egy kicsit eltérő a többi géphez képest. Itt is az az elvárt kezdő állapot, hogy a kilőkő alaphelyzetben legyen, vagyis ne legyen a termék útjában.

Habár ez a tolólap is a pneumatika törvényei szerint működik, mechanikus kialakítása miatt ennek vezérlése valamivel bonyolultabb. Az eddig használt pneumatikus hengerek dugattyúinak, valamint a dugattyúkra szerelt rudak mozgását az adta, hogy a kompresszorral nyomáskülönbséget hoztunk létre a dugattyú által kettéosztott kamrákban, és az ebből adódó nyomóerő leküzdötte a rúdon lévő rúgó rúgóerejét. A nyomás kiegyenlítésekor a rúgóerő kezdeti állapotba lökte a dugattyút.

Az itt alkalmazott henger abban más, hogy a dugattyú mindkét oldalán fellépő nyomást lehet szabályozni. Erre lehet következtetni a modul által felhasznált jelekből, mivel látható, hogy a lökő mozgását most két bittel vezérli, nem eggyel, mint az előző esetekben.

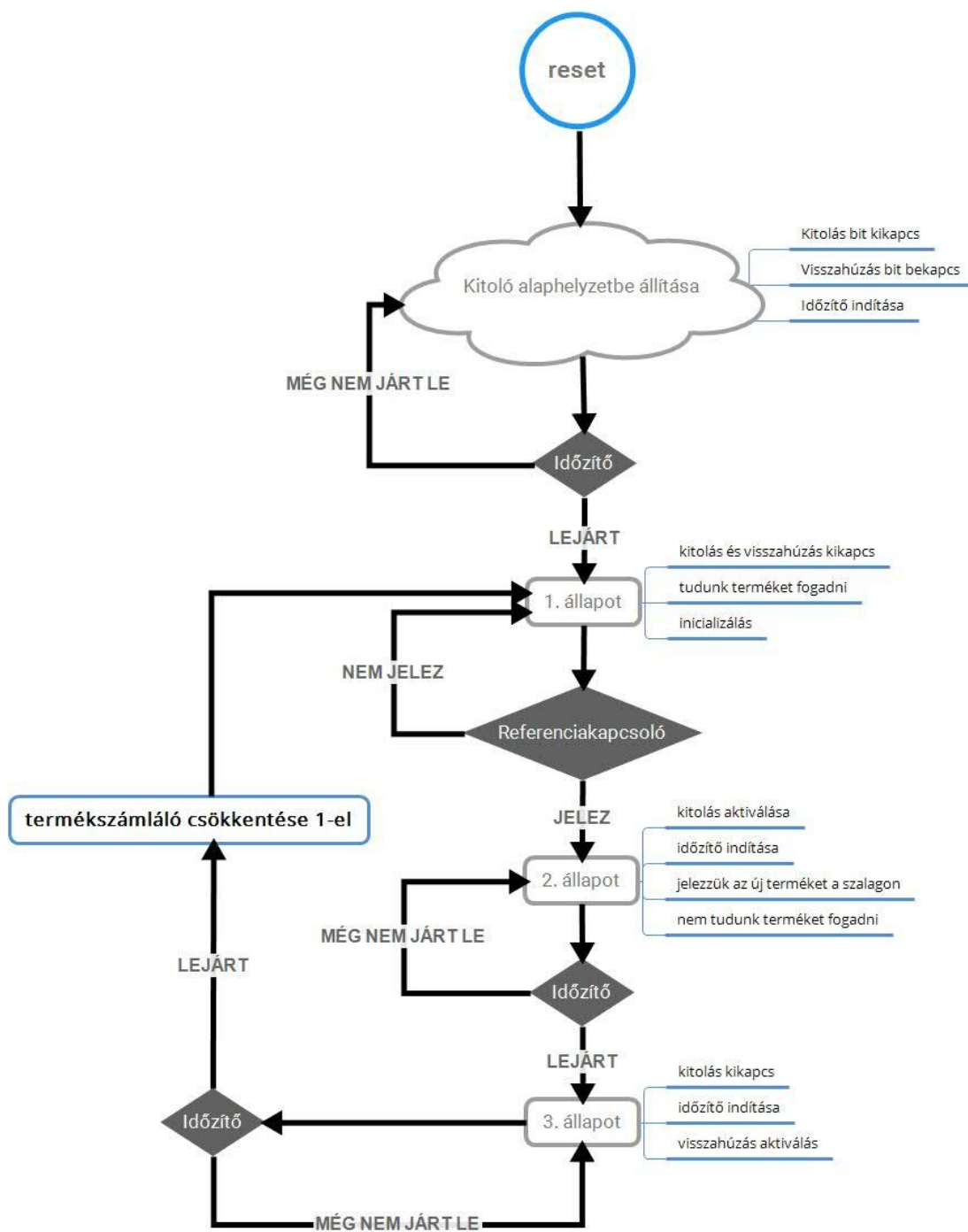
Amire itt még ügyelni kell, hogy nem inicializálhat úgy, hogy ráerőlteti a vezérlésre a kilőkő visszahúzását, amit majd elvesz a következő állapotban. Ennek oka,

hogyan a kompresszor a dugattyúkamarákhoz a már említett mágneses szelepeken keresztül csatlakozik, és a program a pneumatikus technikák esetén ezeket a szelepeket aktiválja. A mágneses szelepeket nem hosszantartó nyitásokra tervezték, a feszültség alatt álló tekercsnek a hőmérséklete gyorsan növekszik, amit aztán a szelep fém felépítése is hamar átvesz, ami meghibásodást okoz. Ebből következik, hogy egyszerre a két irányra nem ad jelet, mert nem fog a dugattyú megmozdulni (nincs nyomáskülönbség), de a szelepeket felesleges hőterhelés alatt tartjuk.

Tehát inicializáláskor azt teszi, hogy kiadja a kilökö visszahúzását okozó jelet, valamint egy időzítőt indít. Lejárával elveszi a vezérlést és a kiinduló állapotba lép. Ekkor biztosítva van, hogy a kilökö rúd alaphelyzetben van anélkül, hogy a szelepeket feleslegesen gerjesztené.

Kiinduló állapotban jelzi, hogy nincs termék, és monitorozza a kilökőnél lévő referenciakapcsolót (ami egyben végálláskapcsoló is, mert egy megrakodott tálca nem tud körbefordulni).

Ezután kilöki a terméket a szállítószalagra, időzítőt indít, jelzi a kilökést a szalag blokkjának, valamint azt is, hogy most nem tud terméket fogadni. Az időzítő lejártakor visszahúzza a rudat, új időzítőt indít, majd annak leteltekor a kiindulási állapotba tér vissza, és csökkenti az állomásban levő termék számlálót 1-el. Ennek okáról a kompresszor vezérlésénél lesz szó.



4-8. ábra: A kilókó állapotai

4.3.4 Forgótálca alegység

BEMENETEK	típus	komment
position_sensor	bool	Forgótálca referenciakapcsoló
reset	bool	Kezdeti állapot felvétele

KIMENETEK		
turntable	bool	Forgatás
LOKÁLIS		
tt_wait	bool	Időzítő kimenet
product_cnt	integer	Az állomásban lévő termékek száma
turntable_state	integer	Forgótálca állapota

4-11. táblázat: Forgótálca interfésze

A forgótálca vezérlése egyedi olyan szempontból, hogy állapotainak váltása attól a három alegységtől (be- és kilökő, prés) függ, amelyeket kiszolgál forgásával.

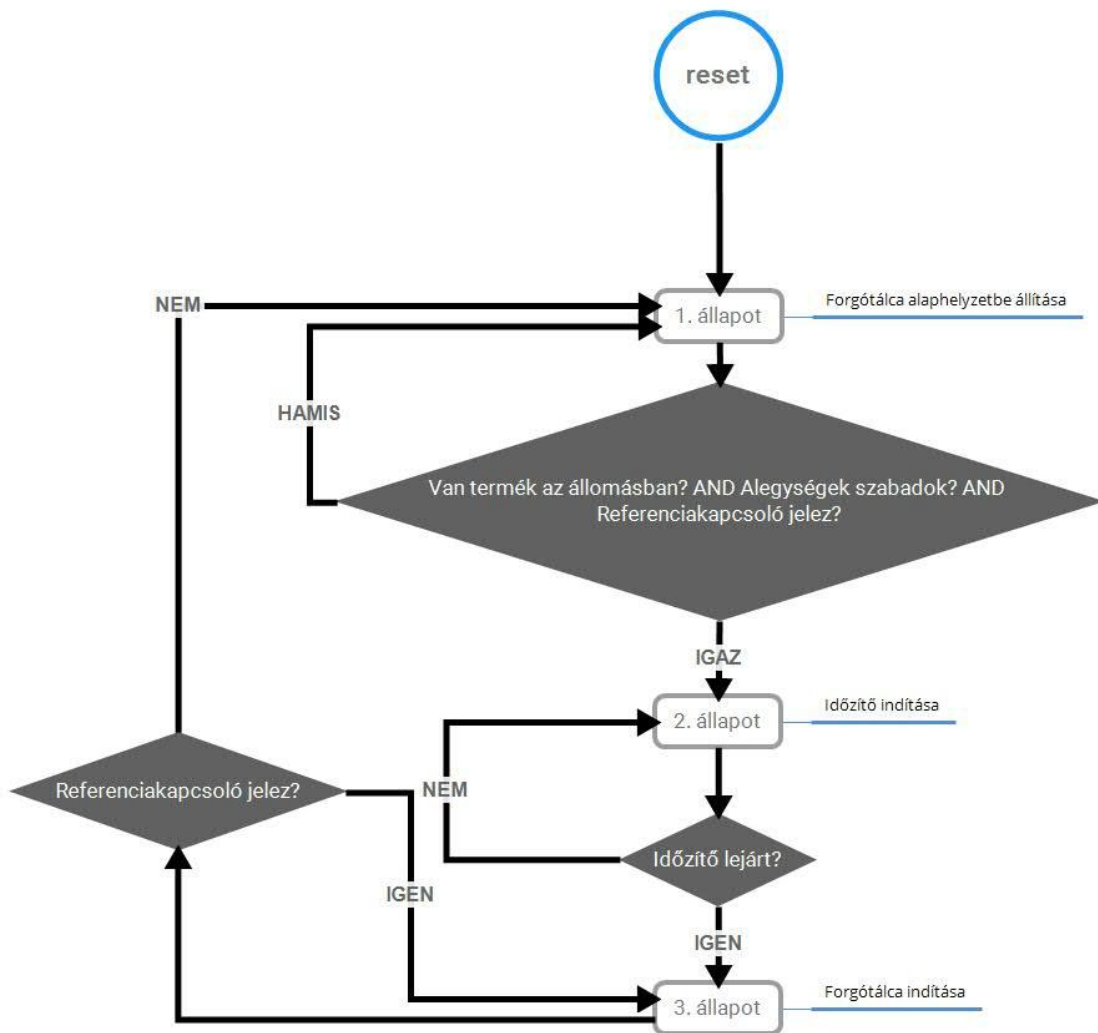
Inicializálás során a tálcát alaphelyzetbe állítja, vagyis a következő olyan pozícióba, ahol a referenciakapcsolója jelez. Ahogy már volt róla szó, a forgótálca kapcsolója akkor jelez, mikor olyan helyzetben van, hogy mindegyik kiszolgált alállomás előtt van egy tálca.

Ebből az állapotból való kilépéshez több feltételt is vizsgálni kell. Ezek a következők:

- Az állomásban lévő termékek száma ne legyen nulla. Ezt a számlálót a belökő növelte 1-el, minden munkaciklusa során, valamint a kilökő csökkentette 1-el ugyanilyen módon. Tehát a forgótálca nem fog működni, ha már nincs termék a tálcákkal kapcsolatban lévő alegységeknél.
- A forgótálca referenciakapcsolója jelezzon. Tehát csak a forgótálca alapállapotából léphet tovább.
- Az alegységek nem lehetnek foglaltak. A prés és a kilökő indulásakor jelzik, hogy ezek most terméket dolgoznak fel (punch_OK és output_OK biteket 0-ra állítja a vezérlésük). A kilökő esetében a kitolt állapotú tolólapban egy forduló tálca kárt okozna, a présnél pedig nem lehet tovább vinni a terméket, míg a préselés nem fejeződött be. A belökőt nem kell ilyen szempontokból vizsgálni, mert annak mozgó része semelyik állapotában nem akadályozza a forgótálcát.

A következő állapotban egy időzítő indul, melynek lejártakor léphet a következő állapotba. Ennek az oka megegyezik a belökő tölcésérzsenzoránál használt késleltetés okával. A belökőrúd nagy gyorsulás ad át a terméknek, és mikor azt megállítja a tálca korlátja, egy rövid idő szükséges, míg a termék felfekszik a tálcára.

Ezután elindul a tálca, majd a referenciakapcsoló kikapcsolására a kiindulóállapotba lép.



4-9. ábra: A forgótálca állapotai

4.3.5 Kimeneti szállítószalag alegység

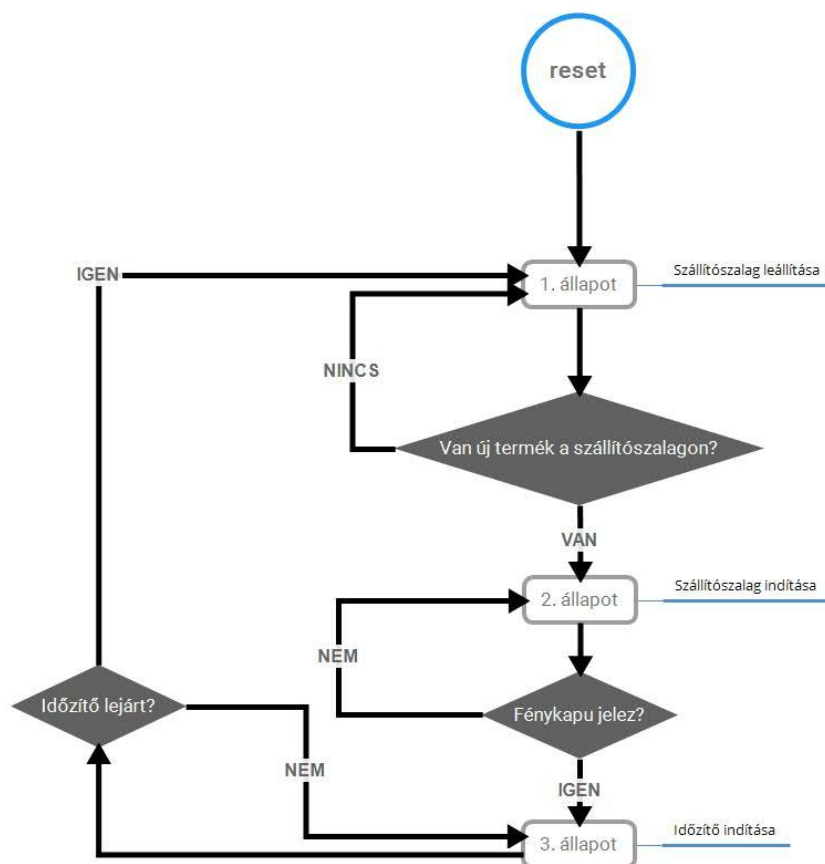
BEMENET	típus	komment
conv_sensor	bool	A szalag végén lévő fénykapu

reset	bool	Alaphelyzetbe állítás
KIMENET		
conveyor	bool	A szállítószalag indítása
LOKÁLIS		
new_prod_on_conv	bool	Új termék van a szalagon
conv_state	integer	Állapotleíró
conv_wait	bool	Időzítő kimenete

4-12. táblázat: Kimeneti szállítószalag interfésze

Reset hatására leállítja a szállítószalagot. Akkor léphet tovább, ha a kilökő jelzi, hogy egy új terméket lökött a szalag elejére.

Ekkor indul a szalag, majd a végén lévő fénykapu jelzésére időzítőt indít, aminek lejártakor alapállapotba lép. A fénykapu jelzésekor még nincs rátolva a termék a megmunkáló állomás rámpájára, ezért kell a szalagot tovább működtetni.



4-10. ábra: Kimenet szállítószalag állapotai

4.3.6 Kompresszor

A kompresszor vezérlése egyszerű. A pneumatikán alapuló manipulátorok működéséhez nyomáskülönbséget kell létrehozni, és a kompresszor bekapcsolt állapotban ezt teszi. A vezérléssel a kompresszor motorja kímélve van (minimalizálja azt az időt, amikor úgy működik, hogy az összes mágnes-szelep zárva van).

Ennek kezelését szolgálja a termékszámológó. Ha ez nem nulla, akkor a kompresszor működik, egyébként ki van kapcsolva.

A belökő azelőtt növeli a számlálót, mielőtt a belökést beállítaná, tehát (kiindulási állapotot feltételezve) amikor a pneumatikus hengernek működnie kellene, akkor a kompresszor már üzemel, és nyomást épített fel.

A kilökő akkor csökkenti ezt a számlálót, miután a tolólapját visszahúzta és a kiinduló állapotba lépne. Tehát (feltételezve, hogy ez volt az utolsó termék) a visszahúzás után azonnal kikapcsol a kompresszor.

4.4 A robotkar alapprogramja

A robotkar vezérlésének felépítése több szempontból is különböző lesz a többi állomásától. A legfontosabb ezek közül az, hogy a kezelt ki- és bemenetek már nem csak bináris értékek lehetnek. A következő pontokban a robotkar mozgásának feladata kezelhető részfeladatokra bontódik.

4.4.1 UDT-k alkalmazása

A robotkar vezérlése során rengeteg jelet kell feldolgozni, tervezés közben kézben tartani. Alapvető eljárás, hogy a leíró tag-eknek olyan elnevezéseket adjunk, melyek következtetések, tömörek, valamint a legtöbb információt tartalmazzák az adott tagról.

Ebben a tervezési problémában ad segítséget az UDT (user defined data types – felhasználó által létrehozott adattípusok). Eme feature segítségével létrehozható olyan változótípus, melyek az adott feladathoz ideálisak. Ilyenkor a típus alá rendelhető olyan változók, amelyek logikailag összetartoznak.

Az UDT-eket definiálásuk után globális DB-kben példányosítódnak, így az információk egy előre tervezett rendszerben érhető el.

4.4.1.1 A robotkar bemeneti információit kezelő UDT

A robotkar helyzetéről a következő információkat kapja a PLC:

- A négy szabadságfok (horizontális, vertikális, betolás, markoló) végálláskapcsolóinak jelei
- Négy számérték, amik a szabadságfokok helyzetét írja le (ebből kettőt közvetlenül a gyors-számláló kártyák szolgáltatják, a másik kettő értékei szoftveresen kezelődnek a későbbiekben leírtak szerint)

Ennek megfelelően a bemeneti jeleket magába foglaló UDT felépítése:

robotInputUDT	típus	komment
verLS	bool	magassági VÁK
horLS	bool	oldalirányú VÁK
rotLS	bool	betolás VÁK
gripLS	bool	markoló VÁK
verPos	double integer (32 bit)	magasság helyzet
horPos	double integer	oldalirány helyzet
rotPos	integer (16 bit)	betolás helyzet
gripPos	integer	markoló helyzet

4-13. táblázat: robotInputUDT (VÁK: végállás-kapcsoló, LS: limit switch)

Az elkészült UDT példányosítódik egy olyan globális DB-ben, ahol az összes vezérlő jel össze van gyűjtve. A példányosított UDT értékeit egy rutin minden ciklusban frissíti.

4.4.1.2 A robotkar bemeneti információit kezelő UDT

A kimeneti jelek a szabadságfokok kétirányú mozgását vezérlik, tehát 8 jel csomagolódik ebben az UDT-ben. Annak érdekében, hogy konzekvens elnevezés legyen, az adott szabadságfok két irányát aszerint lehet megkülönböztetni, hogy a VÁK felé vagy sem történik az elmozdulás.

robotOutputUDT	típus	komment
horToLS	bool	oldalirány VÁKFM
horFromLS	bool	oldalirány VÁKTM
verToLS	bool	magassági VÁKFM
verFromLS	bool	magassági VÁKTM
rotToLS	bool	betolás VÁKFM
rotFromLS	bool	betolás VÁKTM
gripToLS	bool	markoló VÁKFM
gripFromLS	bool	markoló VÁKTM

4-14. táblázat: robotOutputUDT (VÁKFM: VÁK felé mozgás, VÁKTM: VÁK-tól elfelé mozgás)

Ezt az UDT-t is példányosítani kell a vezérlő jeleket gyűjtő DB-ben. Ennek értékeit a szoftver állítja, egy rutin minden ciklusban az értékeket kiteszi a megfelelő kimeneti tag-ekre.

4.4.1.3 Robot koordináta UDT

Ebben a típusban a robot egy pozíciójának adatai tárolódnak:

robotCoordinateUDT	típus	komment
hor	double integer	oldalirányú poz.
ver	double integer	magassági poz.
rot	integer	betolás poz.
grip	bool	markoló poz.

4-15. táblázat: Koordinátát leíró UDT

Ebből tömböt alkotva egy több pontot érintő útvonalat is meg lehet adni.

4.4.1.4 Egyéb adatok a gyűjtő DB-ben

A markoló kezelése el fog térni a többi szabadságfok vezérlésétől, ezért ennek két jelét letárolja a DB-ben:

- expGripper (bool): A markoló esetén két állapotot fog megkülönböztetni: egy teljesen kinyitott állapotot, a másik állapotban pedig rászorít a termékre. Értékével jelezi, hogy milyen állapotban szeretné a markolót.
- gripperOK (bool): A markolót vezérlő blokk ezzel a bittel jelzi, ha sikerült felvenni a várt pozíciót.

4.4.2 A betolás és markoló pozíciójának meghatározása

A címben szereplő két szabadságfok pozíciójáról csupán annyi információt kap, hogy mindkét esetben egy-egy kapcsoló van szerelve a csigatengelyre, aminek menetei forgás közben benyomják azt. Ebből adódóan egy impulzus jut a bemenetre. Az a számlálási konvenció, hogy a VÁK-tól való eltávolodás esetén növekszik az érték, egyébként csökken.

Ez a blokk minden ciklusban le fog futni, tehát nem elég az impulzusok szintjét vizsgálni. Ehelyett a szintváltásokat figyeli, pontosabban fogalmazva az 1-0 váltásokat.

Bemenetek:

- rotWormGearPulse: A betolás csigatengely-kapcsolójának jele
- gripperWormGearPulse: A markoló csigatengely-kapcsolójának jele

Minden más olvasandó és írandó adatot a gyűjtő DB-ben talál.

4.4.2.1 A szintváltás-érzékelő (edge detector) működése

Ez egy rendszerblokk, ami típusától függően képes egy bool változó szintváltását jelezni. Mivel jelen esetben a 0-1 váltást monitorozza, ezért az úgynevezett negatív-él detektort. Alkalmazása:

```
''rotNegEdgeDet''(CLK := #rotWormGearPulse);
```

A nevet a felhasználó adhatja a rendszerblokk hívásakor. Működése annyiból áll, hogy a CLK bemenetére adott jel (ami a példában a betolás csigatengely-kapcsoló) 1-ről 0-ra történő változására a rendszerblokkhoz tartozó példányosítási DB-ben lévő Q bit egy ciklus idejéig 1-re vált.

4.4.2.2 Az értékek frissítése

Ha már monitorozza a szintváltásokat is, akkor már csak IF szerkezetekkel ki kell fejezni a logikát. Példának nézhető a betolás mozgása:

```

IF #rotNegEdgeDet.Q THEN

    IF '''robotPositionData'''.robotOutput.rotFromLS THEN

        '''robotPositionData'''.robotInput.rotPos :=
            '''robotPositionData'''.robotInput.rotPos + 1;

    END_IF;

    IF '''robotPositionData'''.robotOutput.rotToLS THEN

        '''robotPositionData'''.robotInput.rotPos :=
            '''robotPositionData'''.robotInput.rotPos - 1;

    END_IF;

END_IF;

```

A legfelső IF feltétele az éldetektor kimenetének (Q) 1 értéke, tehát ez 0-1 váltáskor teljesül. A következő IF feltételben látható, hogy hivatkozik egy DB-re. A robotPositionData a gyűjtő DB, a benne lévő robotOutput egy példányosított UDT. Az eddigiek alapján látható, hogy most az a feltétel, hogy van-e a betolásnál VÁK-tól elfelé mozgás. Ha igen, akkor a megfelelő adatot növeli. A harmadik IF ugyanezzel a logikával a VÁK felé mozgást nézi.

4.4.3 Az általános szabadságfok kezelő blokk (robot_position)

A vertikális és horizontális mozgást leíró számok az ezres nagyságrendben mozognak, a bemozgást leíró pedig a tízes nagyságrendben. A robotkar útvonalának tervezése során egy előre definiált értékekre kell ráállni a szabadságfokokkal. De a horizontális és vertikális mozgás során semmiképp se lehet pontosan felvenni a kívánt értéket. Ezért definiálni kell a vezérlésnek egy sávot, aminek középpontjában a célérték van, és elégségesnek található a mozgás során a sáv elérése.

Input	típus	komment
mode	bool	számítási mód
targetPos	double integer	cél érték
actualPos	double integer	pillanati érték
limitSwitch	bool	VÁK érték
negativeLogic	bool	Negatív logika van?

KIMENET		
inTargetPosition	bool	Megérkeztünk a pozícióba
moveFromLimitSwitch	bool	VÁKTM
moveToLimitSwitch	bool	VÁKFM
LOKÁLIS		
upperBandLimit	double integer	sáv felső határ
lowerBandLimit	double integer	sáv alsó határ
tempPos	double integer	lementett aktuális pozíció

4-16. táblázat: Az általános mozgási blokk interfész

A tapasztalatok szerint a 25 sugarú sáv megfelelő pontosságot ad²⁰, a blokk ezzel kezd:

```
#upperBandLimit := #targetPos + 25;
#lowerBandLimit := #targetPos - 25;
```

Valamint lementődik az aktuális pozícióérték.

4.4.3.1 Start mód (mode = 0)

Ekkor az adott szabadságfokot a referenciakapcsolójáig vezérli. Ez fontos, mivel minden pozícióérték relatív, így kell egy fix kiindulási pont, ahol ezeket az értékeket lehet nullázni. Ennek kódolása a már ismert módon történik:

```
IF NOT #limitSwitch THEN

    #moveToLimitSwitch := 1;
    #moveFromLimitSwitch := 0;
    #inTargetPosition := 0;

ELSE

    #moveToLimitSwitch := 0;
    #moveFromLimitSwitch := 0;
    #inTargetPosition := 1;
```

²⁰ A sáv sugarát a PLC reakcióideje és a robotkar tehetetlensége határozza meg. Ha ezek nőnek, akkor a sugarat is növelni kell. Túl kicsire állított sugár által okozott jellegzetes hiba a célkoordinátánál történő „rezgés”, aminek magyarázata az, hogy a szoftver próbálkozik felvenni a tartomány értékét, de mindig túlsúsúszik.

```
END_IF;
```

Jelzi, ha elérte a célt, ami most a VÁK.

4.4.3.2 Horizontális és vertikális mód (mode = 1)

A vertikális mozgásért felelős DC motor inkrementer adója úgy számlál, hogy a lefelé mozgás csökkenti a számlálót. De a VÁK a legfelső pozíciónál van, így tehát ez ellentmond a konvenciónak. Ezt úgy lehet megoldani, hogy az aktuális értéket kivonja 0-ból (abszolút értékét veszi), ezután a logika konzisztens lesz:

```
IF #negativeLogic THEN  
    #tempPos := 0 - #tempPos;  
END_IF;
```

Ezután azt ellenőrzi, hogy jelenlegi pozícióból merre induljon:

```
IF #targetPos > #tempPos THEN  
    #moveFromLimitSwitch := 1;  
    #moveToLimitSwitch := 0;  
    #inTargetPosition := 0;  
  
    (...)
```

A konvenciónak megfelelően, ha a kívánt pozíció nagyobb a jelenleginél, akkor a VÁK-tól elfelé indul. A másik lehetőség lekezelésének logikája megegyezik ezzel.

Ezután ellenőrzi, hogy a sávot elérte-e:

```
IF #tempPos < #upperBandLimit AND #tempPos > lowerBandLimit THEN  
    #moveFromLimitSwitch := 0;  
    #moveToLimitSwitch := 0;  
    #inTargetPosition := 1;  
  
END_IF;
```

Tehát leáll a mozgással és jelzi a sikeres elérést.

4.4.3.3 Betolás mód (mode = 2)

A betolásnak azért kell szánni külön számítási módot, mivel annak helyzete tízes nagyságrendű számmal van jellemezve (jóval lassabb a változás sebessége), és így a sávok figyelembe vétele nem szükséges, mert fel tudja pontosan a célt venni:

```
IF #tempPos = #targetPos THEN
```

```

#moveFromLimitSwitch := 0;
#moveToLimitSwitch := 0;
#inTargetPosition := 1;

END_IF;

```

4.4.4 A markoló vezérlése

Ezt külön blokkban érdemes kezelni, mivel az eddigiektől különbözik a logikája. Elsősorban csak két állapotot fog állítani (nyit/zár), valamint a mechanikai pontatlanságot is ki akarja küszöbölni. A markoló csigatengelyén kb. 15 menet van, így a többihez képest durva felbontást szolgáltat a pályájáról. Ezt úgy oldja meg, hogy minden nyitáskor VÁK-ig nyitja a markolót és nullázza a számlálóját.

```

IF NOT ''robotPositionData''.expGripper THEN

  IF NOT ''robotPositionData''.robotInput.gripLS THEN

    ''robotPositionData''.robotOutput.gripToLS := 1;
    ''robotPositionData''.robotOutput.gripFromLS := 0;
    ''robotPositionData''.gripperOK := 0;

  ELSE

    ''robotPositionData''.robotOutput.gripToLS := 0;
    ''robotPositionData''.robotOutput.gripFromLS := 0;
    ''robotPositionData''.gripperOK := 1;
    ''robotPositionData''.robotInput.gripPos := 0;

  END_IF;

END_IF;

```

Ekkor a 0 elvárt markoló állapotot nézi, a VÁK-ig megy a már ismert módon. A VÁK elérésekor a már adott ok miatt a számlálóját nullázza.

```

IF ''robotPositionData''.expGripper THEN

  IF ''robotPositionData''.robotInput.gripPos <> 13 THEN

    ''robotPositionData''.robotOutput.gripToLS := 0;
    ''robotPositionData''.robotOutput.gripFromLS := 1;
    ''robotPositionData''.gripperOK := 0;

  ELSE

    ''robotPositionData''.robotOutput.gripToLS := 0;
    ''robotPositionData''.robotOutput.gripFromLS := 0;
    ''robotPositionData''.gripperOK := 1;

  END_IF;

END_IF;

```

Ekkor az 1 elvárt állapotot kezeli. Látható, hogy addig csukja össze a markolót, amíg a pozíció nem 13. Ennél az értéknél a markoló stabilan rászorít a termékre.

4.4.5 Általános vezérlő blokk (robot_coordinate_handler)

Ez a blokk egyszerre kezeli mind a négy mozgásformát. Egy olyan UDT-t fogad, amiben az összes elvárt pozíció szerepel, és a blokk ezeket az értékeket egyszerre (párhuzamosan) állítja be.

Input	típus	komment
toStartPosition	bool	VÁK-okhoz vezérlés
routeArrayElement	robotCoordinateUDT	cél értékek
KIMENET		
movementDone	bool	minden a célnál van
LOKÁLIS		
verOK	bool	vertikális mozgás kész
horOK	bool	horizontális mozgás kész
rotOK	bool	betolás kész

4-17. táblázat: Összefogó mozgási blokk interfész

Lekezele, ha a VÁK-hoz kell vezérelni mindent:

```
IF #toStartPosition THEN
    ''robot_position''
    (mode := 0,
    targetPos := routeArrayElement.hor,
    actualPos:= ''robotPositionData''.robotInput.horPos,
    limitSwitch:= ''robotPositionData''.robotInput.horLS,
    negativeLogic := 0,
    inTargetPosition => horOK,
    moveFromLimitSwitch => ''robotPositionData''.robotOutput.horFromLS,
    moveToLimitSwitch => ''robotPositionData''.robotOutput.horToLS);

    ''robotPositionData''.expGripper := 0;
    (...)
```

A már létrehozott blokknak átadja a horizontális mozgás adatait, valamint a markolót is kiindulási helyzetbe parancsolja. A többi mozgást is lekezele a robot_position hívásaival.

```

(...)
ELSE

    ''robot_position''
    (mode := 1,
    targetPos := routeArrayElement.hor,
    actualPos:=''robotPositionData''.robotInput.horPos,
    limitSwitch:=''robotPositionData''.robotInput.horLS,
    negativeLogic := 0,
    inTargetPosition => horOK,
    moveFromLimitSwitch => ''robotPositionData''.robotOutput.horFromLS,
    moveToLimitSwitch => ''robotPositionData''.robotOutput.horToLS);

    ''robotPositionData''.expGripper := routeArrayElement.gripper;

    (...))

END_IF;

```

Normál működésnél pedig a mode bemenetet kell átírni (hor. és ver. mozgás=1, betolás=2).

```

IF #verOK AND #horOK AND #rotOK AND ''robotPositionData''.gripperOK THEN

    #movementDone := 1;

ELSE

    #movementDone := 0;

END_IF;

```

Ha a párhuzamosan futó mozgások végeztek, akkor jelzi a kimeneten.

4.4.6 Az útvonalakat kezelő blokk

Az előző pontban létrejött egy blokkot, mely képes teljes mértékben irányítani a robotkart. A mozgások párhuzamosan történnek, aminek hátránya, hogy így állomások között történő ingázások esetén beleütközne akadályokba. Ezért érdemes több pontos útvonalakban gondolkodni. Két pont között történhet a mozgás egymással párhuzamosan. Az útvonalakat a már előző pontban használt koordináta UDT-ből alkotott tömbökkel leírható. Tehát ennél a pontnál az előző pontban megadott blokk hívódik meg és adódik át a megfelelő tömb. A tömbök kezeléséhez szükség volt egy statikus változóra: routeArrayCounter (integer).

```

''robot_movement_handler''
(toStartPosition := 0,
routeArrayElement := ''robotRoutes''.exampleRoute[#routeArrayCounter],
movementDone => #atCoordinate);

```

```

IF #state = 2          AND
   #routeArrayCounter = 6 AND
   #atCoordinate THEN

    #state := 3;

    #routeArrayCounter := 0;

END_IF;

IF #state = 2          AND
   #routeArrayCounter < 6 AND
   #atCoordinate THEN

    #routeArrayCounter := #routeArrayCounter + 1;

END_IF;

```

Feltételezzük, hogy a példában a mozgás 2-es állapotánál tart. Meghívódik az előző pontban tárgyalt blokk, aminek átad egy 7 elemű tömböt, 0..6 számozással, ami egy koordináta típusú tömb.

Az első IF azt vizsgálja, hogy a tömbszámláló a végére ért-e, és ez mellett a mozgás befejeződött-e. Ha igen, akkor az útvonalat befejezte, mehet a következő állapotra, és nullázza a tömbszámlálót.

A második IF azt vizsgálja, hogy növelheti-e a tömbszámlálót. Ha az 6-nál kisebb és az előző mozgáspontot elérte, akkor igen.

Tehát egy kezelő blokk felbontható három részre: az útvonal-tömböt kezelő függvényre, útvonal befejezésének ellenőrzésére, valamint a tömbszámláló növelésére.

A vezérlés első pontja az, hogy a robotkar a szabadságfokokat VÁK-ig vezérli. Ennek elvégzésekor a pozíciókat nullába szükséges váltani, mivel ehhez a ponthoz van viszonyítva az összes koordináta. A markoló és betolás nullázása egyszerű, mert szoftveresen vannak kezelve, és egy globális DB-ben találhatóak. De a másik két pozíciót olyan modulok adják, melyek egy reset hatására törlik kiadott értékeiket. A tag-táblában már fel vannak véve azok a címek, melyek a gyorszámlálók reset-bemeneteit jelentik. Erre adott 0-1 váltás idéz elő törlést. Ez úgy történik, hogy a startpozíció felvételekor ezekre a címekre 0-át ad a PLC, majd a mozgás indulásakor 1-re váltanak, tehát a mozgás kezdeténél kapnak 0 értéket. A DB-ben lévő pozíciókat is ekkor törli a szoftver.

4.4.7 A robotkar útvonalai és az állomásokkal történő kommunikáció

Az előző pontban ki lett fejtve, hogyan valósul meg egy több pontot tartalmazó útvonal kezelése. Ezután annyit kell tenni, hogy watch table segítségével manuálisan irányítjuk a robotkart a kérdéses pontba, majd leolvashatóak a koordináták. Ezeket aztán bevezethetjük egy olyan globális DB-be (robotRoutes), ahol a robot útvonalait tároljuk. A fentieknek megfelelően átadódnak ezek az útvonalakat, majd annak végeztével a megfelelő állapotba lép.

4.4.7.1 A robotkar jelei

Feltételezve azt, hogy a gyártás a tölcser-tárolóban felhalmozott termékekkel kezdődik, akkor a robotnak a feldolgozó- és a prés állomások felé kell jeleket adnia.

Gondolatban végig is követhető a folyamat:

1. A pneumatikus állomáson átmegy a termék, majd ráteszi a rámpára.
2. A megmunkáló állomás végén a termék megáll. Az állomás működésének részletezésénél már volt róla szó, hogy arról, hogy a termék elhagyta a kimeneti alegységet, már a robottól kell információt kapnunk.
3. Ekkor a prés gépben még biztosan nincs termék, tehát a robot ide teszi a terméket.
4. A megmunkáló állomásban egyszerre akár 4 termék is lehet, tehát termékiadása jóval gyorsabb lesz, mint a prés gépnek. Ezért annak érdekében, hogy a termelés ne álljon egy lassabb állomás miatt, azért ha a megmunkáló végez egy termékkel, de a prés nem szabad, akkor a terméket a kiindulópontra teszi, ami a pneumatikus állomás tölcser tározója.
5. A prés gépből kijövő terméket szintén a tölcserbe teszi.

A fentieket végiggondolva belátható, hogy a robotkarnak 2 szituációban kell jelet adnia:

- Amikor a megmunkáló állomástól vesz el terméket, akkor egy bittel jelzi, hogy ez megtörtént.
- A prés gépnek is ad jelet, amikor letesz egy terméket.

Ennek implementálásának megértéséhez nézzünk egy példát. Tételezzük fel, hogy a robotvezérlés 3-as állapotánál tart. Egy olyan útvonalat tesz meg, aminek kiindulópontja a megmunkáló állomás kimenete zárt markolóval, végpontja a présgép bemenete fölött van, miután letette a terméket. Tehát az útvonal végeztével jelezni kell a présnek, hogy kezdheti a munkát:

```
''communication''.robotToMachiningSignal := 0;

(...)

IF #state = 3          AND
   #routeArrayCounter = 6 AND
   #atCoordinate THEN

   ''communication''.robotToMachiningSignal := 1;

   (*állapotválasztó logika*)
```

Minden állapotot azzal kell kezdeni, hogy ezeket a kommunikációs jeleket nullázza (a példában csak a megmunkálóállomás jelzőbitje említődik). Az útvonal végén 1-be állítja ezt a bitet, és ott is marad a robot következő állapotáig.

Ezeket a jeleket menedzselni kell minden állapotban, ahol a végpont az, hogy terméket vesz el a megmunkálótól, vagy terméket tesz le a préshez.

4.4.7.2 A robotkar jeleinek fogadása

Vegyük példának a présgép módszerét arra, hogyan használja a robotkar jelét. A prés állapotgráfja azt mutatja, hogy az utolsó állapotból akkor lép a kiindulóba, ha egy új terméket kapott. Természetesen ezt a szenzorokból nem tudható meg, ezt az elnevezést a prés alapműködésének minél könnyebb megértésének érdekében adtam ennek az átmenetnek.

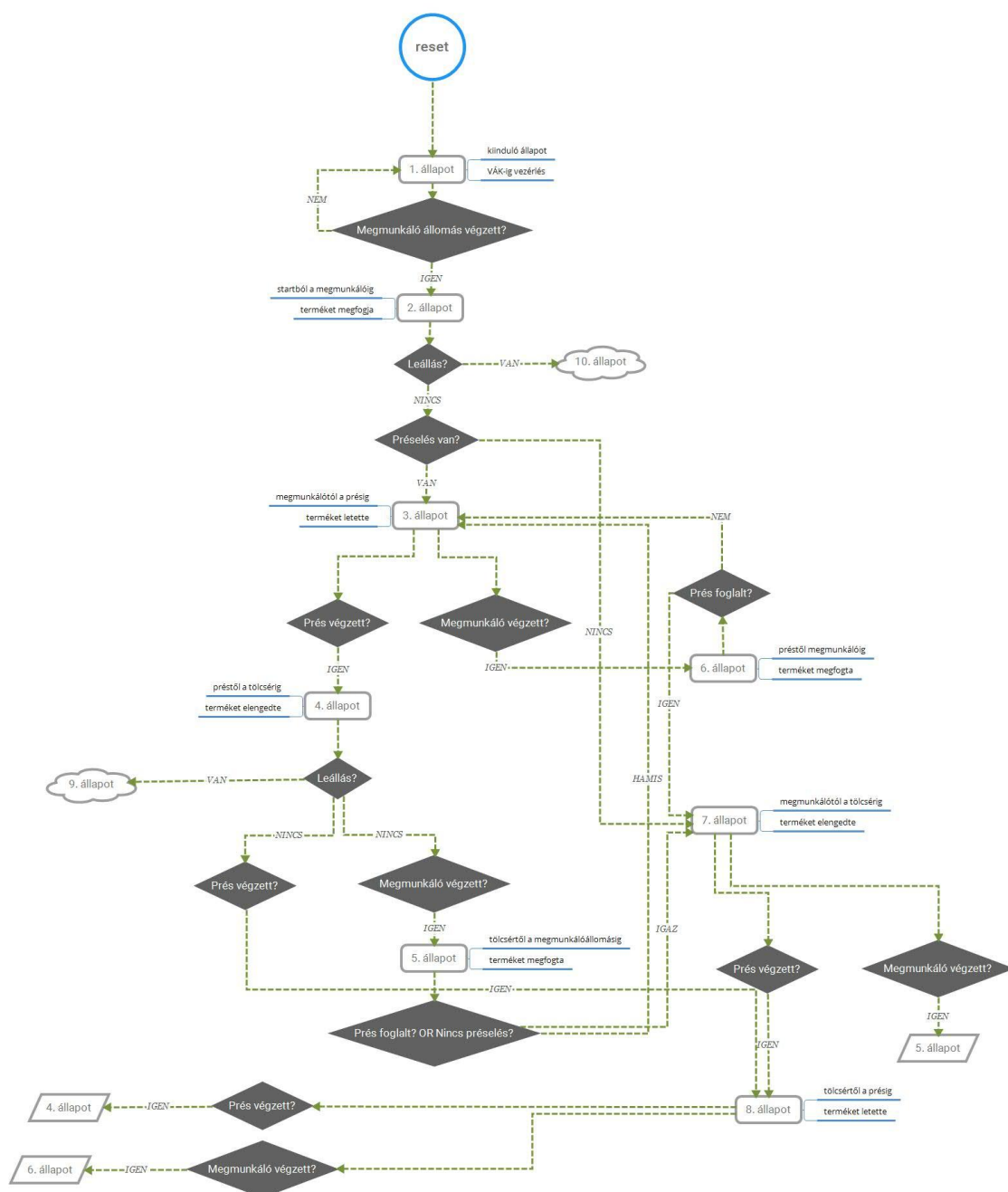
A valódi implementációban akkor lép a kiinduló állapotba, ha a bemeneti szenzor jelez. De ez hibás működést okoz, mivel a kiinduló állapotban azonnal teljesülnek a továbblépés feltételei. Ezért a termék visszaérkezésekor egy újabb munkaciklust indít el.

Ezt elkerülendő úgy, hogy a robotkar kommunikációs jelét is bekerül az indulási feltételek közé. Bár ez nem lesz teljesen elég, mivel a robotkar akár hosszabb ideig is adhatja ezt a jelet (fals indítást adhat). Ezt megoldandó, kommunikációs jel 0-1 felfutó élét monitorozza a már megismert éldetektorral:

```
''signalPosEdgeDet''(CLK := ''communication''.robotToPuncherSignal);
```


$$(\dots)$$

4.4.8 A robotkar mozgási gráfja



4-11. ábra: A robot útvonalai

Az ábrán végigkövethető az eddig ismertetett termékkezelési stratégia megvalósítása. Látható két felhő-szimbólummal megkülönböztetett állapot is. Ezek nem tartoznak szorosan a robot alapl működéséhez, hanem a leállás opció során lépnek érvénybe.

A folyamatban kétszer is meg van vizsgálva, hogy van-e préselés. Ennek hátterében a receptkezelés van, ahol beállítható, hogy 0 préselést kér az operátor. Ilyenkor úgy fut a vezérlés, hogy a prés nincs figyelembe véve. Le is ellenőrizhető: ha nincs préselés, akkor csak azon állapotokhoz érhetünk el a gráfban, amikkel a termékeket a megmunkálóból a tölcsérbe rakodjuk.

Több állapotnál előfordult, hogy utánuk két feltétel is következett. Ez azt jelenti, hogy mindkét feltétel egyszerre kerül vizsgálat alá, és azon a vonalon megy tovább, ahol a feltétel először teljesült.

A paralelogrammában jelzett állapotok jelentése az, hogy az adott állapot következik a vezérlésben, de a bejártott összekötés már zavaró bonyolultságúvá tenné az ábrát.

Természetesen az is előfordulhat, hogy a továbbmenetel feltételei közül egyik sem teljesül az útvonal végén. Ekkor a robotnak várakoznia kell egy kommunikációs jelre. Ilyenkor beállítódik egy bit (wait), aminek értéke bekerül a tömbszámlálót növelő logika feltételei közé, így nem tud kilépni a definiált tömb határaiból (ami rendszerhibát okozna és a PLC azonnali leállításához vezetne).

A fentiek megvalósításához még három kommunikációs bit állítására van szükség:

- `''communication''.machiningDone`: Ezt a bitet a végzett megmunkáló állítja.
- `''communication''.puncherDone`: Ezt a bitet a végzett présgép állítja.
- `''communication''.puncherBusy`: Ennek értéke akkor 1, ha a prés dolgozik, és van nála termék.

4.5 A leállási procedúra

Ez a funkció a valós gyártósoroknál is fellelhető. Lényege az, hogy a gyártósorból ki akarjuk üríteni a termékeket egy raktárba, majd ha a kiürített

gyártósoron elvégződött a szükséges karbantartás, akkor a termékek visszahelyezése után újraindítható a termelés.

Külön raktárral nem rendelkezünk, de a présgép egyenes szállítoszalagja használható raktárként. Tehát olyan állapotot kell neki definiálni, ami a bemeneti szenzor jelzésekor a terméket a szalag végéig beviszi. Ezt elegendő egy időzítővel vezérelni.

A robotkar működése annyiban tér el, hogy a leállás parancsára csak a megmunkáló állomásból a présgépbe tesz terméket. Az előbbi vezérlési gráfját megvizsgálva látható, hogy az alapprogramból két helyen lehet belépni a leállási procedúrába. Az egyik lehetőség akkor van, amikor először vesz fel a megmunkálótól terméket, a másik, amikor a présből a tölcsérbe tesz terméket. Azért választottam ezeket az állapotokat a kilépés lehetőségére, mert a további vezérlések implementálása egyszerűsödik, mivel mindkét állapotban garantálva van, hogy a prés szabad.

Az eljárás része az is, hogy a robotkar a procedura végén a kezdeti állapotba menjen. Ezt a gyártásban lévő termékek számlálásával lehet megoldani, amit a már létrehozott blokkokkal könnyű kivitelezni. A pneumatikus állomás belökője már regisztrálja a belökött termékek számát, itt egy másik változót is növel, amit egy globális DB-ben tárol. Ezt persze már nem csökken a kilökönél, hanem csak akkor, mikor a termék a gyártás végére ér (vagyis mikor megint bekerül a tölcsérbe), vagy leállás esetén akkor, mikor a présre került a termék. A termékszámológát akkor kezeli a szoftver, mikor már az adott útvonallal végzett:

```
IF #state = 2          AND
   #routeArrayCounter < 6 AND
   #atCoordinate THEN

   #routeArrayCounter := #routeArrayCounter + 1;

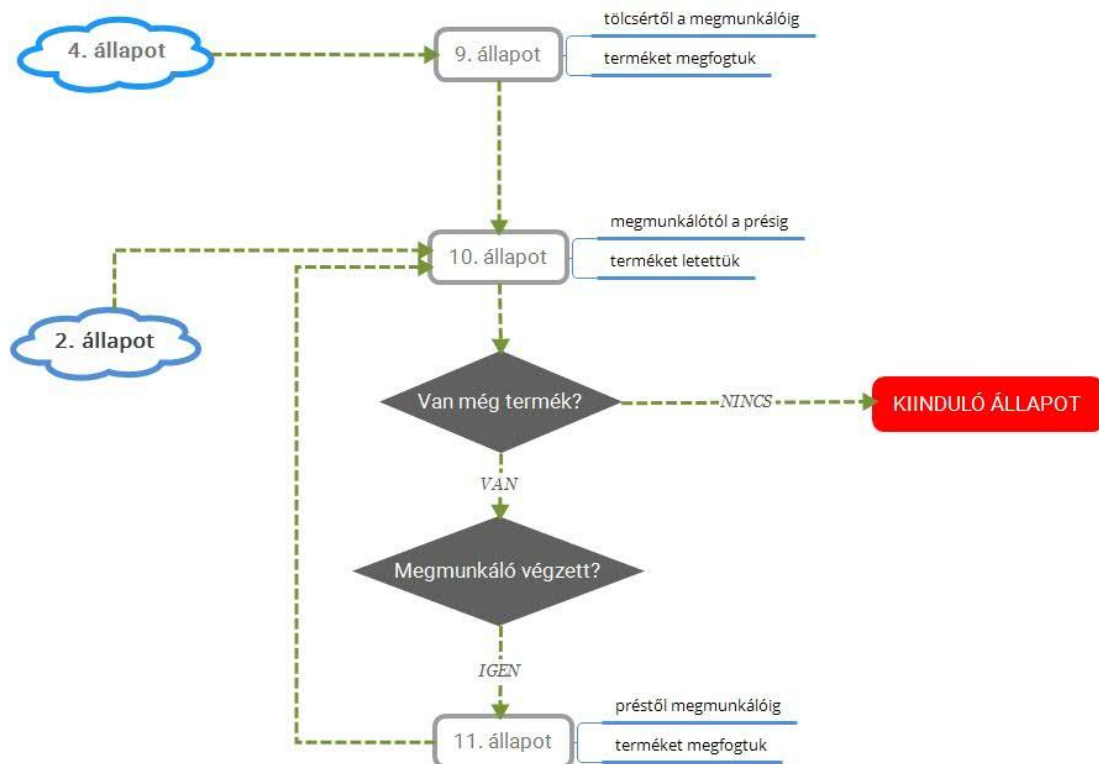
   IF #routeArrayCounter = 6 THEN

       ''GDB''.product_cnt := ''GDB''.product_cnt - 1;

   END_IF;

END_IF;
```

Tehát amikor az útvonal utolsó pontja következik, akkor már a számláló értéke aktualizálva van, és az állapotváltó logika észleli, ha nincs több termék.



4-12. ábra: Robot leállás kezelése

Az ábra a már bemutatott robot-vezérlési gráf kiterjesztésének tekinthető. A felhő szimbólumok mutatják, hogy a normál üzemmódból melyik állapotból léphet át leállásba. Csak a présre tesz terméket, majd mikor végezett, akkor a robotkar a start-pozíciót veszi fel (VÁK-ig vezérel).

4.6 Receptek kezelése

Már többször volt róla szó, hogy egyes gyártási beállításokat az úgynevezett receptekből kapják a blokkok. Ennek háttere az, hogy egy HMI felületen be lehet majd állítani a használandó értékeket, ezzel a gyártás menetét befolyásolhatjuk, gyorsíthatjuk:

- Pneumatikus állomás préselés: Receptváltozóból kértük le a vezérlés a préselés számát. Ha ez nulla, akkor nincs préselés.
- Megmunkáló állomás marás és fúrás ideje: Ezt is receptből kapja. Ha az idő 0 másodperc, akkor nem áll meg a gép alatt a termék.
- Présgép: Ha a préselések száma nulla, akkor a robotkar kihagyja a présgépet.

A recepteknek van egy hátránya, mégpedig az, hogy alapesetben a receptváltás nincs egy időrészhez rendelve. Ha van termék megmunkálás alatt, akkor arra már az új recept beállításai lesznek érvényben, tehát a termékek nem ugyanabban a megmunkálásban részesültek. További intézkedéseket kell tennünk, ha a recept konzisztenciáját fenn akarjuk tartani.

Ennek kezelési stratégiája a következő: Ha receptváltás történik, akkor nem enged terméket vissza a gyártásba, tehát a tölcsérben fognak tárolódni a megmunkálásuk után, ami még a régi recept alapján történt. Ha minden termék a tölcsérbe került, akkor újra engedélyeződik a gyártás az új recept paramétereivel.

Először kell egy UDT, amiben a gyártási tulajdonságok szerepelnek:

- milling_time: Marás ideje
- drilling_time: Fúrás ideje
- pne_punching: Pneumatikus állomás préseles szám
- punching_machine: Prés gép préseles szám

Ezt az UDT-t példányosítjuk kétszer: Az egyik értékeit a HMI közvetlenül állítja, a másik értékeit pedig felhasználja a gyártósor.

Létrehozunk egy szubrutint, aminek az a dolga, hogy jelzi, ha új receptérték(ek) kerültek beállításra. Ha ez történt, akkor jelzi, hogy sérült a gyártás konzisztenciája:

```
IF ''GDB''.usedProperties.milling_time<>''GDB''.newProperties.milling_time
OR
''GDB''.usedProperties.drilling_time<>''GDB''.newProperties.drilling_time
OR (...)
THEN

''GDB''.recipeConsistency := 0;

ELSE

''GDB''.recipeConsistency := 1;

END-IF;
```

A newProperties értékeit a HMI állítja, a usedProperties értékeit kapják meg az állomások. Ha a két változó bármely tagja nem egyenlő akkor sérül a konzisztencia. Szükséges minden változót ellenőrizni, mivel előfordulhat, hogy két recept csak egy paraméterben különbözik, valamint a HMI receptkezelőjében akár egyesével is megadhat az operátor értékeket az előre definiált receptektől függetlenül.

Ha visszakap a pneumatikus állomástól egy láttamozó jelet, akkor a használatban lévő változóba betöltődik az új:

```
IF ''GDB''.recipeChangeAcknowledge THEN  
  
''GDB''.usedProperties.milling_time:=''GDB''.newProperties.milling_time;  
''GDB''.usedProperties.drilling_time:=''GDB''.newProperties.drilling_time;  
(...)  
  
END_IF;
```

A pneumatikus állomásnak nem szabad új terméket fogadnia, mikor receptváltás történt. Tehát a belökő vezérlésénél, a kiindulás állapotváltó logikájánál bekerül a recept-konzisztencia is a feltételek közé.

A kiinduló állapotban ha a termékszámoló nulla és nincs recept-konzisztencia, akkor adja a láttamozójelet, betöltődnek az új értékek, a konzisztencia visszaáll, majd a belökő el tud indulni.

4.7 Hibakezelés

A gyártósor működése során nem tervezett állapotokba juthat. Ennek oka az állomások kialakításának mechanikai tökéletlensége. Ebből fakadó hibák lehetnek például az elakadt termékek. Ezek olyan hibakezeléssel oldható meg, ami időintervallum alapú. Tehát ha egy termék elhalad egy szenzor előtt, akkor normál működés esetén eléri a következő szenzort X másodperc alatt. Ha ez mégsem történik meg X-től több másodperc alatt, akkor hibát jelez, annak okát kiírja a HMI, valamint a hibás állomás kikapcsolódik. Ha a hibaelhárítás megtörtént, akkor a HMI-n láttamozzuk a hibát és az állomás a félbehagyott állapotától folytatja.

Az időtartam hosszának, aminek meghaladásával hibás működés jelenthető ki, egyértelműen hosszabbak kell lennie a normál működés idejétől. Hogy mennyivel hosszabb, az függ a szakasz megbízhatóságától. A gyártósor mechanikai gyengepontjai a két szállítószalag csatlakozása (megmunkáló), valamint a robotkar pontatlansága, ami többször okozott hibát a présnél. Nem pontosan teszi rá a terméket a bemenetére, ezért a szalagnak időre van szüksége, hogy a terméket lehúzza a peremről.

A másik ellenőrizendő pont a robotkar. Ezt úgy oldható meg, hogy rövid intervallumokként (250 ms) beolvassa a szubrutin a 4 pozíció jelét, és ha a két egymás utáni beolvasás különbsége kisebb a normál működés közben elvártnál, akkor a robot megakadt és azonnal le kell állni.

A hibák könnyebb követésének érdekében létrehozunk egy UDT-t, ami a következőképp épül fel:

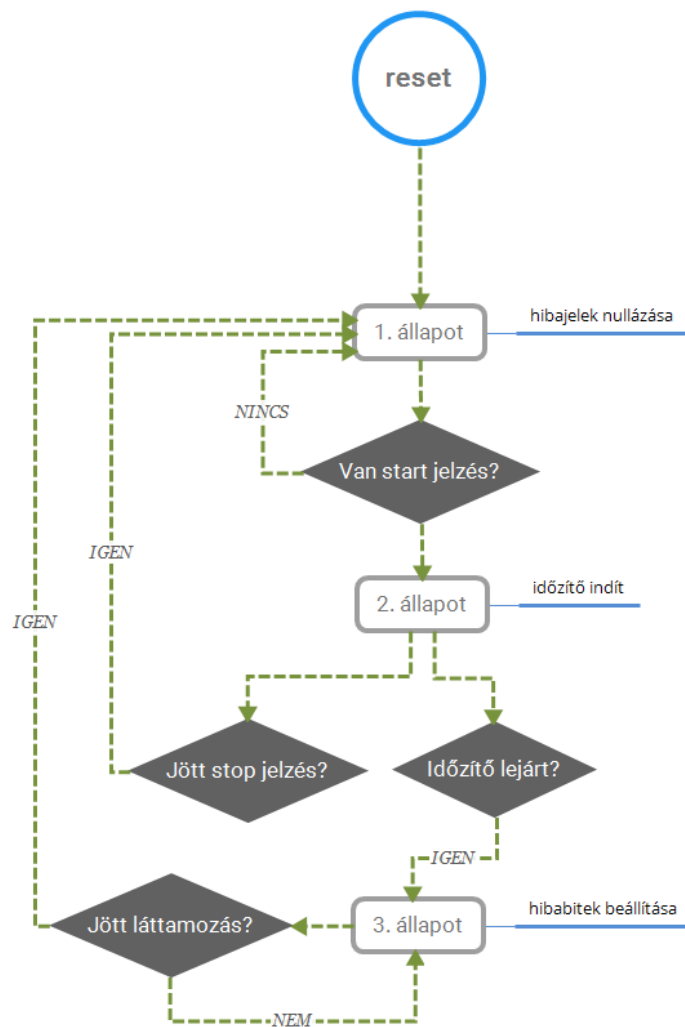
- `alarm_time`: Időintervallum típus. Ez adja meg az adott hibalehetőség felső időkorlátját.
- `delta`: Mozgásmonitorozás közben a minimum lemozdulás értéke.
- `error`: Hibabit, amit 1-et vesz fel észlelt hiba esetén.
- `hmi_error`: Hibajelzés (integer), amit a HMI olvas.
- `hmi_ack`: Láttamozó bit a HMI-től, ha 1 az értéke, akkor a HMI-n láttamoztuk a hibát.

Ezt az UDT-t annyiszor példányosítjuk, ahány hibalehetőségünk van.

4.7.1 Időtartam alapú hibajel

Létrehozunk egy függvényt, melynek példányosításai egy-egy hibalehetőséget tudnak kezelni. A bemenet egy start és stop jel lesz, amik a vizsgált szakasz elején és végén lévő szenzorjeleket kapják meg. A függvény megkapja ki- és bemenetként az előbb definiált UDT egy példányát (az időintervallumot, a deltát, valamint a láttamozást bemenetnek vesszük, a többi kimenetnek).

Reset után a kimeneti értékeket nullázza. Start jel esetén indít egy időzítőt, aminek idejét az UDT adja. Ezután várakozik, és ha az időzítő lejártáig nem kap stopjelet, akkor a hibajelzéseket állít be. Láttamozásra vár, hatására a kiindulásba lép.



4-13. ábra: Idő alapú hibagenerálás

4.7.2 Mozgás alapú hibajel

Ennél a függvénynél a bemenetek a monitorozandó szabadságfok irányjelei, valamint pozíciója. Ezen kívül ez is megkap egy UDT, csak itt egyértelműen a delta lesz releváns.

A kezdeti inicializálás után a következő állapotba lép, ha bármely irányba mozgást érzékel. Ekkor egy timert indít rövid időtartamra (250 ms), és egy lokális változóba lementi az akkor beolvasott pozíciót. Ha lejárt előtt leáll a mozgás, akkor a kiindulás állapotába lép. Ha nem, akkor ellenőrizheti az értékeket.

Ehhez van három érték: az időzítő előtti és utáni pozíció, valamint a delta, mely értékének megfelelőt mindenképpen változnia kell a pozíciónak, ha nem akadályozott a mozgás. Az ellenőrzéshez el kell különíteni két esetet, a VÁK felé mozgást és az

ellenkező irányú mozgást. Előre lefektetett logika alapján tudható, hogy az első esetben a pozíció nő, a másodikban csökken. Tehát:

```
IF #movementAwayFromLimitSwitch THEN

    (*Ha a késleltetés végén beolvasott poz. nagyobb, mint az elején
    beolvasott + a minimum elvárt mozgás, akkor nincs hiba*)
    IF #decoder > #tempDecoder + #delta THEN
        #state := 1;
    ELSE
        #state := 5;
    END_IF;

END_IF;
```

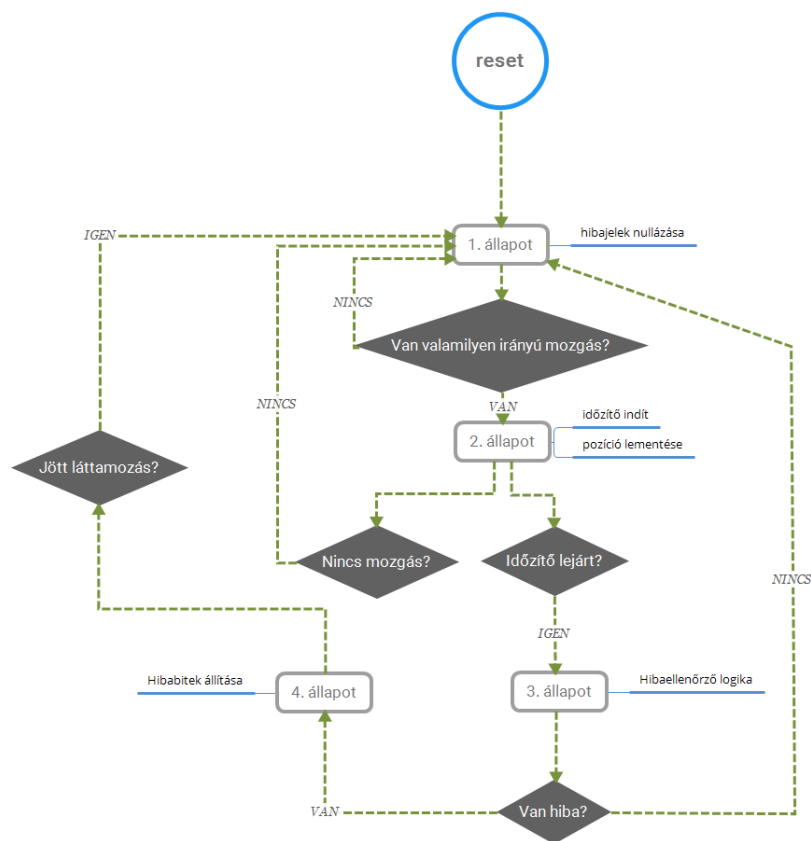
A másik irányba történő mozgás monitorozásának logikája megegyezik, csupán arra kell figyelni, hogy ott a pozíció csökkenni fog:

```
IF #movementToFromLimitSwitch THEN

    IF #decoder < #tempDecoder - #delta THEN
        #state := 1;
    ELSE
        #state := 5;
    END_IF;

END_IF;
```

Ezután hiba esetén az időalapú megfigyeléssel megegyezően tér vissza a kiindulásba.



4-14. ábra: Mozgás alapú hibajelzés

4.7.3 Időalapú vezérlés implementálása

A függvények megírása után azokat példányosítani kell, hozzájuk a megfelelő UDT-ket rendelni (majd a hibának megfelelő üzenetet adunk a HMI-n). A robotmozgás esetén ezzel végeztünk is, de az időalapú megfigyelésnél több dologra kell figyelni. Ennek végigkövetéséhez a megmunkáló alarming-ját veszem példának.

Az ellenőrzés szempontjából a szenzorok szintje nem lényeges, hanem inkább a fel- és lefutó élek adhatnak információt. Vegyük az a szituációt, amikor a megmunkáló a robotkarra várakozik! Ekkor van egy termék a szalag végén, amit a robot majd legközelebb elvesz. Mivel a kimeneti alegység ezáltal még nincs kész, ezért a következő termék a fűrófejnél várakozik a továbbvitelre. Az ezután következő a marófejnél, az azután az első tolólap előtt, a legutolsó pedig a rámpa alján várakozik, ahol már jelez a bemeneti fénykapu. Ez a szituáció könnyen előfordulhat, ha tele tölcserrel (5 termék) kezdjük a gyártást, mivel a robot + prés állomások áteresztése jóval lassabb, mint a pneumatikus + megmunkáló állomásoké.

Tehát ezen az állomáson vizsgáljuk:

- bemeneti és az első tolólapi szenzor közti szakaszt
- „első tolólapi szenzor” és fúró közti szakaszt
- fúró és maró közti szakaszt
- valamint a fúró és kimeneti szenzor közti szakaszt.

A felsorolásban az idézőjellel jelölt szakasz különleges eljárást igényel. Nem érdemes a második felsorolt szakaszt az első tolólapi szenzorral kijelölni, mert várakozás esetén a termék a tolólap előtt fog lenni, miután elhaladt a szenzor előtt. Ezért a második szakasz startjelének az első tolólap első végálláskapcsolóját állítom²¹, így a termék továbbállásakor indul a monitorozás is.

Szeretnénk elkerülni, hogy a várakozás miatt lévő állás ne indítson időzítést. Ez megoldható éldetektálókkal. Fényszenzorok esetén ez azt jelenti, hogy beérkező termék esetén 1-0 élt ad, elhagyó termékénél 0-1 élt. Például a maró és fúró közti szakasz jeleinek detektálása:

```
//marófel elhagyásának detektálása pozitív él detektorral
#leaving_milling_det(CLK:=#millingSensor);

//fúrófejhez megérkezés detektálása negatív él detektorral
#arriving_drilling_det(CLK:=drillingSensor);
```

Ennek a metodikának másik előnye az, hogy az alarming kezelő rutin pozitív logikát feltételez, míg a fényszenzorok nem ezt követik. A detektorokkal tulajdonképp átkonvertáltuk a negatív logikát pozitívba.

A megfelelő jelek előállítása után már csak a függvényt kell meghívni:

```
//marófej -> fúrófej szakasz ellenőrzése
#drilling(prop:='GDB'.M_AL_3,
          start:=#leaving_milling_det.Q,
          stop:=#arriving_drilling_det.Q);
```

A prop paraméternél kell megadni a szakaszhoz tartozó UDT-t.

Megvalósítottam a tolólapok ellenőrzését is (előforduló hiba az, hogy a tolólap leesik a sínjéről). Ekkor a startjel az előre- vagy hátratólás, míg a stopjel a megfelelő végálláskapcsoló:

```
//Az első tolólap előremenetelének ellenőrzése
```

²¹ A hátsó kapcsoló 1-0 váltását is ki lehetne jelölni kezdésnek.

```
#p1F(prop:='GDB'.M_AL_5,  
      start:=#pusherForward,  
      stop:=#pusher1ForwardLimitSwitch);
```

Ezenkívül a hátramenet, majd ugyanígy a második tolólap mozgása is ellenőrizve van.

Hátramaradt teendők az, hogy a hibás állomás működését letiltuk. Ehhez készítünk egy 4 elemű tömböt, ahol egy elem 1 értéke egy állomás hibás állapotát jelenti. Ez egyszerűen megtehető, hiszen csak az implementációnk végén egy IF szerkezetben megvizsgáljuk minden az állomáshoz köthető alarm-UDT hibabitjét és ha valamelyik jelez, akkor a tömb megfelelő tagját 1-be állítjuk.

Ilyenkor is nagy hasznunkra válik az állapotgépes felépítés. Hiszen hibás esetén könnyedén leállíthatjuk az állomás működését úgy, hogy a vezérlő CASE szerkezetet egy IF szerkezetbe tesszük, melynek feltétele az állomás hibabitjének 0 értéke. Ellenkező esetben kikapcsolunk minden kimenetet, így hiba esetén az állomás leáll. Mivel az állapotleíró nem változtattuk, ezért nyugtázás után a megszakadt állapotot folytatja az állapotgép.

A présgépnél a bemeneti és prés szenzor jeleit nézzük a már közölt logika szerint (ezt a szakaszt két irányba is vizsgálni kell). Ha hiba lép fel, akkor leáll az állomás és készenléti jelet sem tud adni, tehát a robotkar kihagyja a mozgásából a prés bemenetét.

A pneumatikus egység ellenőrzése ugyanígy történik, csupán a szakaszokat egyesével jelölik ki fénykapuk és referenciakapcsolók.

4.8 Termékek követése

Ezzel a feature-el szeretnénk regisztrálni, hogy a termékeink mely állomásoknál tartanak a gyártás folyamán. Az egyszerűség kedvéért feltételezzük, hogy 4 termékkel dolgozunk. Létrehozunk egy 4 elemű integer tömböt, mely a termékeket képviseli. Az integer 0 értéke jelzi, hogy a termék a tölcserben van, 1 a pneumatikus állomást, 2 a megmunkálót, 3 a présgépet jelenti.

A bemeneteink olyan jelek, amelyek aktív értéke jelzi, hogy egy állomásra termék érkezett:

- Pneumatikus belökő

- Rámpa alján lévő szenzor
- Robot jelzése a megmunkálónak
- Prés foglaltságjelzője

Nézzük a pneumatikus állomásba belépés kezelését! A belökő jelére megkeressük a tömbben az első olyan elemet ami nulla, tehát a tölcserben van. Majd ezt az elemet 1-be állítjuk, mert a pneumatikus állomásnál van. Implementációja:

```
//a tömb végétől kezdünk
#array_cnt := 3;

//ha az érték 0, akkor frissítjük a beállítandó indexet
WHILE #array_cnt > -1 DO
    IF ''GDB''.tracking[#array_cnt] = 0 THEN
        #trackingArrayElementToSet := #array_cnt;
    END_IF;
END_WHILE;

''GDB''.tracking[#trackingArrayElementToSet] := 1;
```

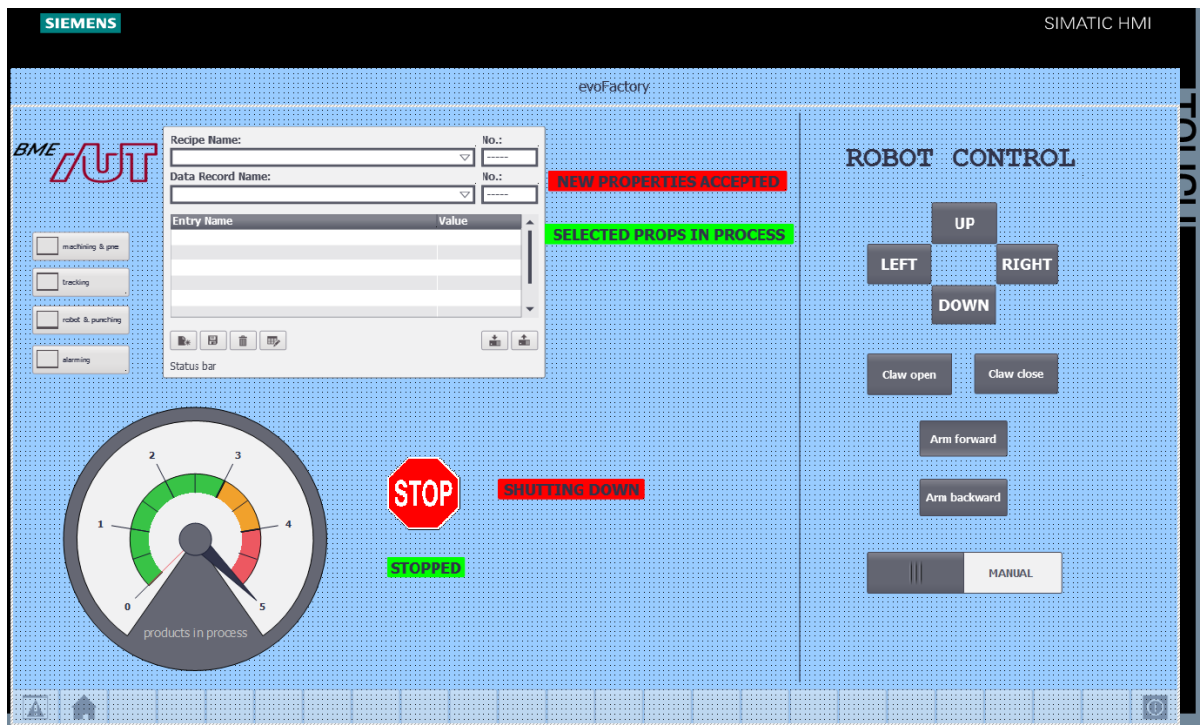
A többi átmenet ellenőrzésének logikája megegyezik a fentivel, csupán a megmunkálóállomás elhagyásánál kell figyelni a prés foglaltságára, mert ettől függ a termék célja.

Természetesen ez nem egy abszolút követő rendszer, mivel a gyártósornak nincs apparátusa a termékek egyediségét meghatározni. A tölcsernél felálló sorrend is változni fog gyártás közben, mert például az első terméket a második előzni fogja a prés miatt. De arról pontos képet fogunk kapni, hogy a termékek közül melyik hol található.

4.9 HMI

A vezérlés végeztével el kell készítenünk a folyamatot támogató HMI-eszköz felprogramozását. A HMI tekinthető egy olyan kijelzőnek, melynek felépítése ipari környezet tűrő, valamint az érintőképernyő technológia segítségével a kezelőtől parancsokat tud fogadni. A HMI felületeinek szerkesztésénél azt a tervezési elképzelést kell figyelembe venni, hogy a modell akár egy valódi gyártósor is lehetne, tehát olyan kezelőfelületek szükségesek, melyek az esetleges gyártósori operátorok munkáját megkönnyítik.

4.9.1 A HMI felületei



4-15 ábra: A HMI kezdőképernyője

A kezdőképernyőn jelennek meg a folyamat legfontosabb jellemzői. A jobb oldalon a robotkar manuálisan irányítható, miután a kezelőfelület alatt lévő csúszkával kikapcsoljuk a robotot vezérlő blokkot. Ez az opció leginkább a vezérlés fejlesztése során volt hasznos, mikor meg kellett találni az egyes célpontok koordinátáit.

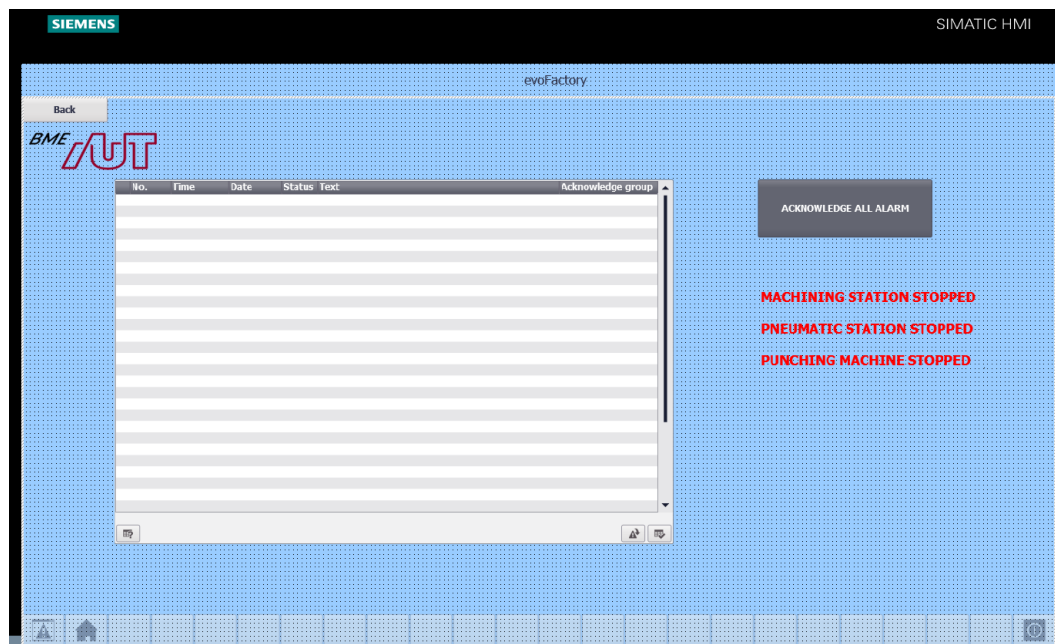
A táblázatban a receptek kiválasztása lehetséges. A NEW PROPERTIES ACCEPTED szöveg akkor jelenik meg, mikor a recept konzisztencia sérült, de a kért receptet sikeresen beolvastuk, és még nem jutott érvényre. A SELECTED PROPS IN PROCESS felirat akkor aktív, mikor már a kijelölt recepttel kezdődött újra a gyártás.

Az analóg mérőeszköz a folyamatban lévő termékek számát jelzi. A STOP táblára kétszer bökve aktiváljuk a leállási procedúrát, amit a SHUTTING DOWN szöveg is jelez. A leállítás végeztével jelenik meg a STOPPED felirat.

A bal oldali gombokkal a többi képernyőre navigálhatunk.

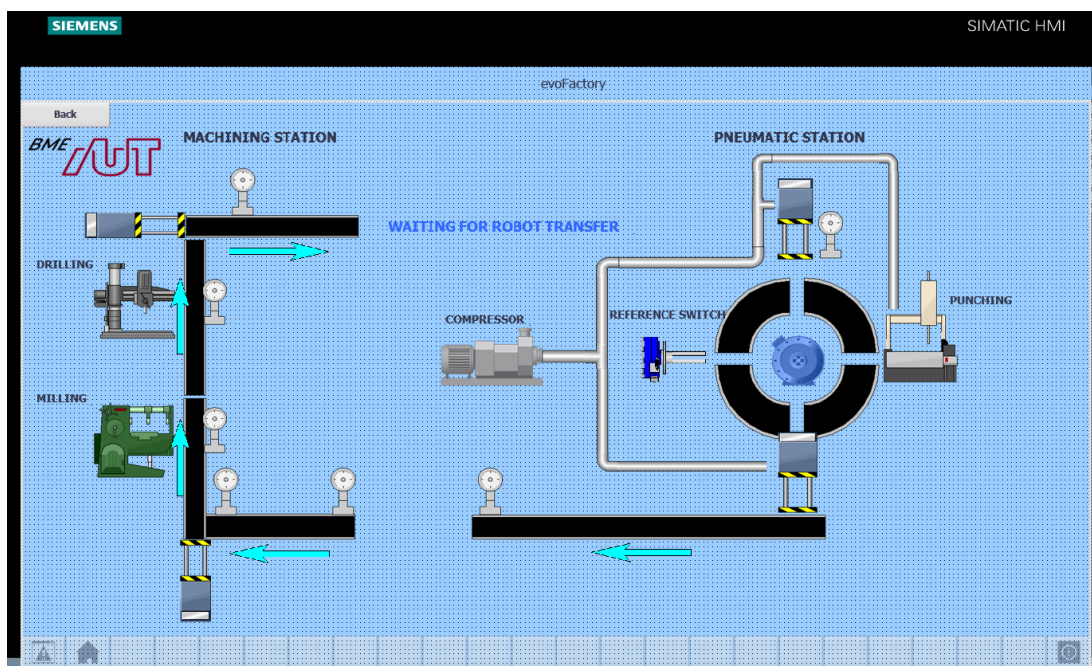
A TIA Portal felület-szerkesztője megjelenít minden grafikai elemet, ami megjelenhet a működés során. A működésükből adódóan például a recepttel kapcsolatos szövegek egyszerre nem jeleníthetők meg. Helytakarékoság szempontjából a két

szöveget lehetne egymásra helyezni, így egy területen jelennének meg, de az alkalmazott elrendezés a dokumentáció miatt előnyösebb.



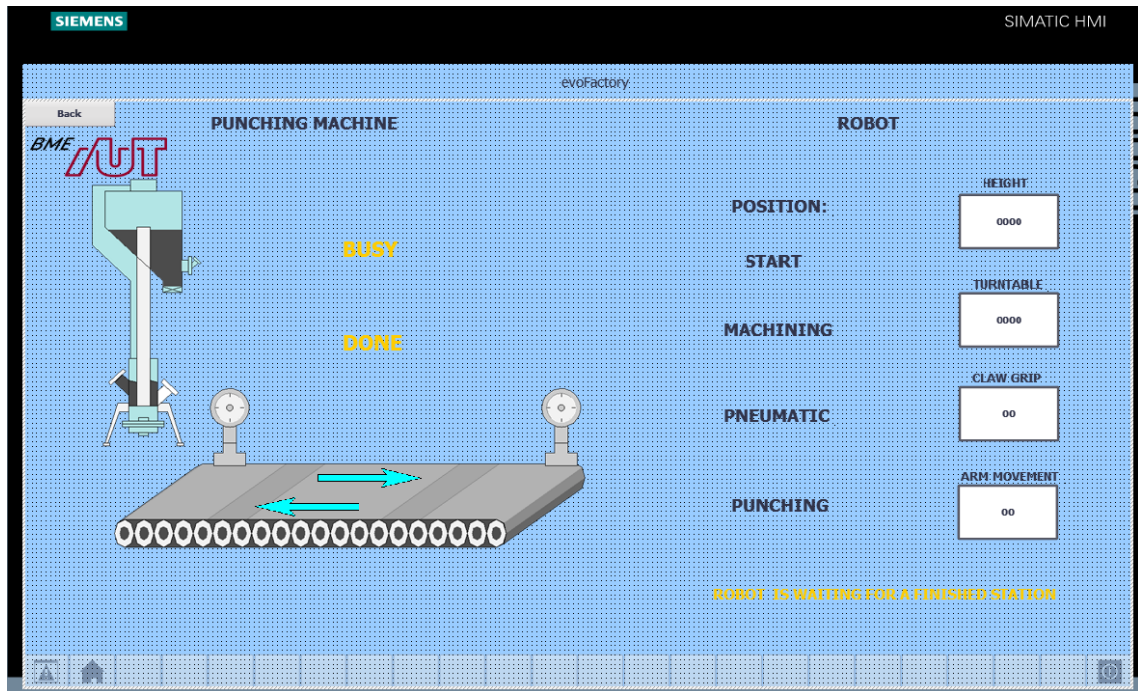
4-16 ábra: HMI hibakezelő

A hibakezelő felületen jelenítődnek meg a hibaüzenetek, amikből az operátor tájékozódhat, hogy mi okozhatta a hibát a gyártásban. A hiba láttamozásával az adott állomás újraindul. A felugró szövegek arról adnak információt, hogy mely állomások álltak le. A kihelyezett gombbal az összes hiba egyszerre láttamozható.



4-17 ábra: Megmunkáló és pneumatikus állomások

A TIA Portal grafikus elemeit gyűjtő könyvtárból a folyamatokat legjobban jellemzőket használva szerkeszthetők a folyamatképek. A működés nyomonkövetésének érdekében a működésben lévő elemek háttérszínét megváltoztatjuk.



4-18 ábra: Prés és robot

A présgépnél ugyanígy jártunk el. A robotkar lemodellezésének feladata túlmutat a szakdolgozat keretein, de a legfontosabb információk leolvashatóak: A robot elhelyezkedése, valamint a szabadságfokainak pozíciói.

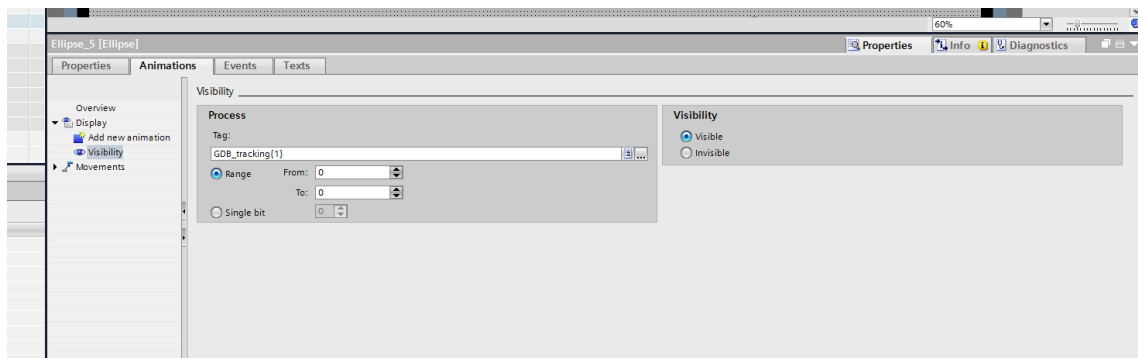


4-19 ábra: Termék nyomonkövetése

A nyomkövetés megjelenítése úgy zajlik, hogy a termék pozíciójának megfelelően egy kör jelzi a táblázatban az elhelyezkedését.

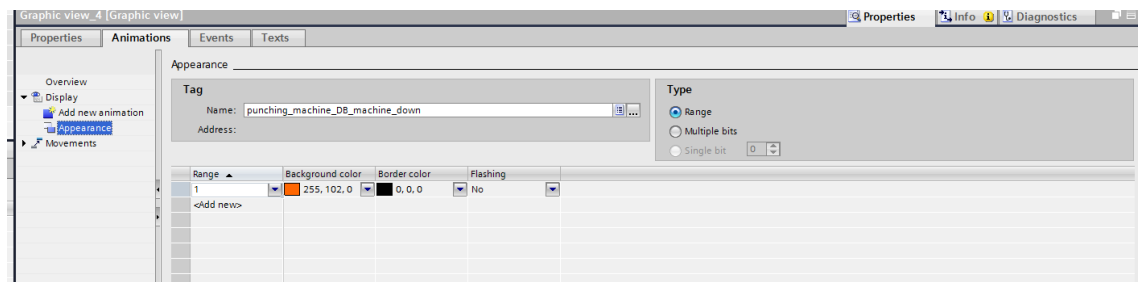
4.9.2 A grafikus elemek dinamikus megjelenítése

A TIA Portal kezelőfelületével könnyedén beállíthatjuk az adott grafika viselkedését.



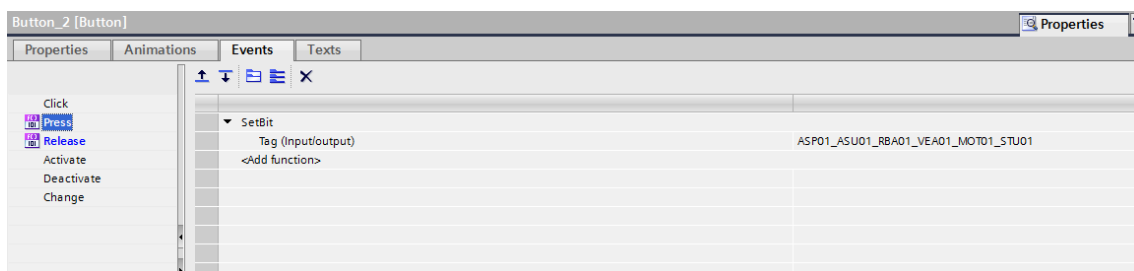
4-20 ábra: Láthatóság beállítása

A fenti képen látható beállítással annak a körnek a láthatóságát vezéreljük, amely akkor jelenik meg a nyomkövető képernyőn, mikor a kinevezett második termék a tölcserben van. A beállításhoz a regisztráló tömb második elemét rendeltük, és ha ennek az elemnek értéke 0, akkor látható.



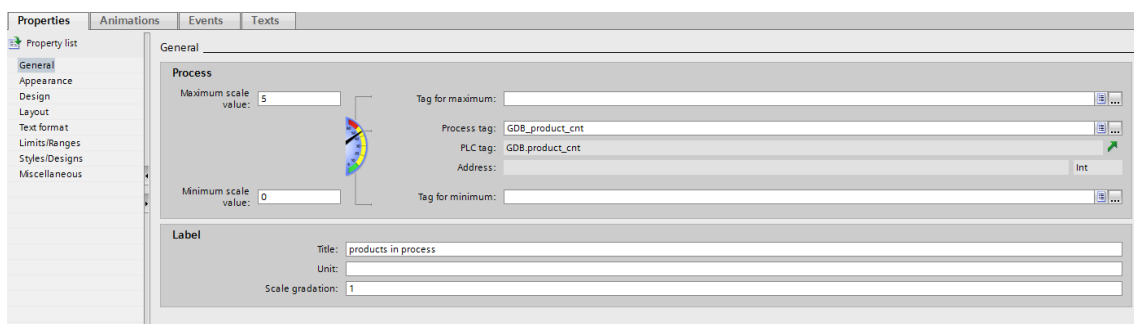
4-21 ábra: Megjelenítés beállítása

Itt a présgépet szimbolizáló objektum háttérszínét állítjuk. A beállításhoz azt a bitet rendeljük, amely a prés lemenetelésénél 1 értékű. Beállítjuk, hogy ennek 1 értéke esetén a háttér színe narancssárga legyen. Így a működés a kijelzőn jól megfigyelhető.



4-22 ábra: Eseménykezelés

A rendelkezésre álló függvényekkel a nyomógombokat is könnyen tudjuk paraméterezni. Az eseményeknél kiválasztjuk a lenyomást, majd a rendelkezésre álló bitet 1-be állító függvénynek átadjuk például robotkar egyik kimenetét a manuális vezérléshez. Látható, hogy az elengedés esemény is be van állítva, ott a bitet 0-ba állító függvényt alkalmazzuk.



4-23 ábra: Analóg kijelző beállítása

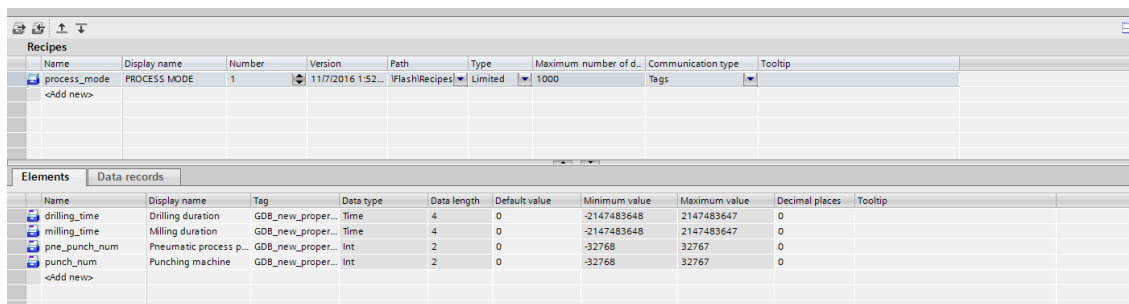
Az analóg kijelzőhöz rendelődik a termékszámoló változó. Címkrét, maximális/minimális értéket, valamint felosztás finomságát is beállítjuk.

4.9.3 Recept- és hibakezelés beállítása

Discrete alarms										
ID	Name	Alarm text	Alarm class	Trigger tag	Trigger bit	Trigger address	HMI acknowledgment tag	HMI acknowledgment bit	HMI acknowledgment address	Report
1	MA_ALARM_01	Machining/input module missing product	Errors	GDB_M_AL_01_hmi_error	0	GDB_M_AL_1_hmi_error.x0	GDB_M_AL_01_hmi_ack	0	GDB_M_AL_1_hmi_ack.x0	<input type="checkbox"/>
2	MA_ALARM_02	Machining/milling module missing product	Errors	GDB_M_AL_2_hmi_error	0	GDB_M_AL_2_hmi_error.x0	GDB_M_AL_2_hmi_ack	0	GDB_M_AL_2_hmi_ack.x0	<input type="checkbox"/>
3	MA_ALARM_03	Machining/drilling module missing product	Errors	GDB_M_AL_3_hmi_error	0	GDB_M_AL_3_hmi_error.x0	GDB_M_AL_3_hmi_ack	0	GDB_M_AL_3_hmi_ack.x0	<input type="checkbox"/>
4	MA_ALARM_04	Machining/output module missing product	Errors	GDB_M_AL_4_hmi_error	0	GDB_M_AL_4_hmi_error.x0	GDB_M_AL_4_hmi_ack	0	GDB_M_AL_4_hmi_ack.x0	<input type="checkbox"/>
5	PN_ALARM_01	Pneumatic/input module missing product	Errors	GDB_PN_AL_1_hmi_error	0	GDB_PN_AL_1_hmi_error.x0	GDB_PN_AL_1_hmi_ack	0	GDB_PN_AL_1_hmi_ack.x0	<input type="checkbox"/>
6	PN_ALARM_02	Pneumatic/puncher missing product	Errors	GDB_PN_AL_2_hmi_error	0	GDB_PN_AL_2_hmi_error.x0	GDB_PN_AL_2_hmi_ack	0	GDB_PN_AL_2_hmi_ack.x0	<input type="checkbox"/>
7	PN_ALARM_03	Pneumatic/output module missing product	Errors	GDB_PN_AL_3_hmi_error	0	GDB_PN_AL_3_hmi_error.x0	GDB_PN_AL_3_hmi_ack	0	GDB_PN_AL_3_hmi_ack.x0	<input type="checkbox"/>
8	PU_ALARM_01	Punching machine/input missing product	Errors	GDB_PU_AL_1_hmi_error	0	GDB_PU_AL_1_hmi_error.x0	GDB_PU_AL_1_hmi_ack	0	GDB_PU_AL_1_hmi_ack.x0	<input type="checkbox"/>
9	PU_ALARM_02	Punching machine/output missing product	Errors	GDB_PU_AL_2_hmi_error	0	GDB_PU_AL_2_hmi_error.x0	GDB_PU_AL_2_hmi_ack	0	GDB_PU_AL_2_hmi_ack.x0	<input type="checkbox"/>
10	PN_ALARM_04	Pneumatic to machining transfer error	Errors	GDB_PN_AL_4_hmi_error	0	GDB_PN_AL_4_hmi_error.x0	GDB_PN_AL_4_hmi_ack	0	GDB_PN_AL_4_hmi_ack.x0	<input type="checkbox"/>
11	MA_ALARM_05	Machining/pusher1 forward movement failed	Errors	GDB_M_AL_5_hmi_error	0	GDB_M_AL_5_hmi_error.x0	GDB_M_AL_5_hmi_ack	0	GDB_M_AL_5_hmi_ack.x0	<input type="checkbox"/>
12	MA_ALARM_06	Machining/pusher1 backward movement failed	Errors	GDB_M_AL_6_hmi_error	0	GDB_M_AL_6_hmi_error.x0	GDB_M_AL_6_hmi_ack	0	GDB_M_AL_6_hmi_ack.x0	<input type="checkbox"/>
13	MA_ALARM_07	Machining/pusher2 forward movement failed	Errors	GDB_M_AL_7_hmi_error	0	GDB_M_AL_7_hmi_error.x0	GDB_M_AL_7_hmi_ack	0	GDB_M_AL_7_hmi_ack.x0	<input type="checkbox"/>
14	MA_ALARM_08	Machining/pusher2 backward movement failed	Errors	GDB_M_AL_8_hmi_error	0	GDB_M_AL_8_hmi_error.x0	GDB_M_AL_8_hmi_ack	0	GDB_M_AL_8_hmi_ack.x0	<input type="checkbox"/>
15	R_ALARM_01	Robot: Error during vertical movement	Errors	GDB_R_AL_1_hmi_error	0	GDB_R_AL_1_hmi_error.x0	GDB_R_AL_1_hmi_ack	0	GDB_R_AL_1_hmi_ack.x0	<input type="checkbox"/>
16	R_ALARM_02	Robot: Error during horizontal movement	Errors	GDB_R_AL_2_hmi_error	0	GDB_R_AL_2_hmi_error.x0	GDB_R_AL_2_hmi_ack	0	GDB_R_AL_2_hmi_ack.x0	<input type="checkbox"/>
17	R_ALARM_03	Robot: Error during claw movement	Errors	GDB_R_AL_3_hmi_error	0	GDB_R_AL_3_hmi_error.x0	GDB_R_AL_3_hmi_ack	0	GDB_R_AL_3_hmi_ack.x0	<input type="checkbox"/>
18	R_ALARM_04	Robot: Error during arm movement	Errors	GDB_R_AL_4_hmi_error	0	GDB_R_AL_4_hmi_error.x0	GDB_R_AL_4_hmi_ack	0	GDB_R_AL_4_hmi_ack.x0	<input type="checkbox"/>

4-24 ábra: Hibakezelés

A felmerülő hibák kezelése is könnyen menedzselhető a TIA Portalban. A fenti táblázat kitöltése után tulajdonképp a konfigurálással végeztünk is. Megadjuk a hibához tartozó üzenetet, a hibát aktiváló (triggerelő) bitet, valamint a láttamozó bitet. Ezeket a biteket az alarminghoz kapcsolódó példányosított UDT-k tartalmazzák, és a megfelelő példányra kell hivatkozni.

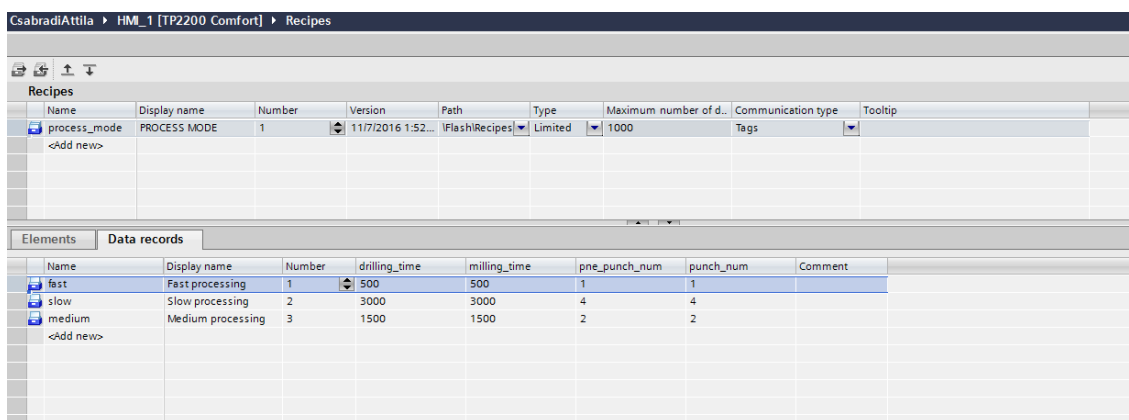


Name	Display name	Number	Version	Path	Type	Maximum number of d...	Communication type	Tooltip
process_mode	PROCESS MODE	1	11/7/2016 1:52...	FlashRecipes	Limited	1000	Tags	
<Add new>								

Name	Display name	Tag	Data type	Data length	Default value	Minimum value	Maximum value	Decimal places	Tooltip
drilling_time	Drilling duration	GDB_new_proper...	Time	4	0	-2147483648	2147483647	0	
milling_time	Milling duration	GDB_new_proper...	Time	4	0	-2147483648	2147483647	0	
pne_punch_num	Pneumatic process p...	GDB_new_proper...	Int	2	0	-32768	32767	0	
punch_num	Punching machine	GDB_new_proper...	Int	2	0	-32768	32767	0	
<Add new>									

4-25 ábra: Receptkezelés

A receptelemek megadásával megadjuk az egyes „összetevők” neveit, és hogy ezek mely változókat állítsák (már volt róla szó, hogy a new_properties változóhoz kell a receptkezelőt kapcsolni).



Name	Display name	Number	Version	Path	Type	Maximum number of d...	Communication type	Tooltip
process_mode	PROCESS MODE	1	11/7/2016 1:52...	FlashRecipes	Limited	1000	Tags	
<Add new>								

Name	Display name	Number	drilling_time	milling_time	pne_punch_num	punch_num	Comment
fast	Fast processing	1	500	500	1	1	
slow	Slow processing	2	3000	3000	4	4	
medium	Medium processing	3	1500	1500	2	2	
<Add new>							

4-26 ábra: Recept rekordok

A receptelemek deklarálása után megadjuk, hogy az egyes recepteknél a receptelemek milyen értéket vesznek fel.

5 A műszaki alkotás értékelése, fejlesztési lehetőségek

A feladatkiírásban felállított követelményeknek a vezérlőszoftver eleget tesz. A gyártósor modell minden funkciója működésre van bírva. A receptekkel megadott paramétereket a megfelelő blokkok fogadják, és annak megfelelően adaptálják vezérlésüket. A feltérképezett hibalehetőségeket monitorozzuk, és a hibát generáló állomás automatikusan leáll. A definiált indulási pont (tölcsérben lévő termékekkel induló vezérlés) megkövetelt egy definiált leállási pontot is, ezt implementálta a leállási procedúra. A HMI felületei minden releváns információt megjelenítenek.

Fejlesztés közben és után a gyártósor több órás próbaüzemet teljesített vezérlési hiba nélkül, a mechanikából adódó hibák mindig regisztrálásra kerültek. Tesztelés során először egy termék kezelése került kipróbálásra, majd miután több munkaciklus az elvárt viselkedéssel lefutott, ezután következett a több termékkel tesztelés. A pneumatikus állomás forgótálcája és azzal kapcsolatban lévő alegységei képesek voltak a folyamatos termékellátást kezelni. A megmunkáló alegységei csak 1-1 terméket képesek feldolgozni egyszerre, de ez kielégítő, mivel a tölcsérben ennyi fér el. A présgép el tudta látni mind a préselési, mind a raktározási feladatokat. A robotkar mozgási pályájának pontossága megfelelő volt, képes volt a normál- és leállási procedúra termékmozgatásait elvégezni. Az esetek kb. 5%-ban nem pontosan tette le a terméket a prés bemenetéhez, tehát azt a szállítószalag nem tudta mozgatni, de a hibakezelés jelezte a problémát. Minden szakasz hibaellenőrzése ki lett próbálva úgy, hogy kézzel az adott szakaszból elvettem a terméket. A hibát produkáló állomások leálltak, majd a HMI-n információ volt olvasható a leállítás okáról. A termék visszatétele után a láttamozás hatására az állomás újraindult. A termékmegmunkáló alegységek fogadták a receptek értékeit, úgy, hogy a konzisztencia mindig érvényben maradt. A recept 0 értéke a pneumatikus állomásnál azt jelentette, hogy nem állt le a tálca a présnél, és a prés nem indult; a megmunkálónál a termékek elhaladtak a maró- és fűrőfej alatt; a présgépre nem tett terméket a robot.

További fejlesztési lehetőségek:

- A robotkar egy munkadarab letétele után az adott állomás rakodója fölött vár a következő kommunikációs jelre. Az útvonalának elvégzése

után egy semleges, az állomásoktól egyenlő távolságra lévő koordinátánál várakozhatna, így reakcióideje csökkenne.

- Egy vonalkód-olvasó készülék beszerzésével és integrálásával lehetséges lenne a termékek abszolút követése.
- A leállási procedura a HMI-n történő stop jelzés hatására azonnal érvényre jut. Ennek az aktiválódását is lehetne késleltetni a receptváltáshoz hasonlóan annak érdekében, hogy a leállítás előtt minden termék egyező megmunkálásban vegyen részt.
- Egy vákuumos tappanccsal felszerelt robotkar manipulátora kisebb helyen is elférne, mint a mostani robot markolója, így a navigálás egyszerűbb lenne, valamint a termékek felszedése is gyorsabb lenne.
- A robot enkódereinek minimális szabálytalanságát ki lehetne úgy küszöbölni, hogy egy előre definiált számú útvonal megtétele után a kiindulópontba megy resetelni.
- A vezérlések szekvenciális részeit refaktorálni lehetne a Siemens-eszközök által támogatott GRAPH-nyelvvel. Mivel a blokkok tetemes része gráfokkal leírható, egy gráf alapú nyelv használata még átláthatóbbá tenné a szoftvert.

Az elvégzett feladatok hatására jelentős tapasztalatra tettem szert a sorrendi folyamatok vezérlésében, a hardverek mechanikai tulajdonságainak szoftver oldali kezelésében, a robotkar mozgásának implementációjában, az átlátható HMI-felületek szerkesztésében.

A robotkar részletezett szoftver-paradigmája újrafelhasználásra került a tulajdonos vállalat által kezdeményezett modellbővítésnél.

6 Köszönetnyilvánítás

Szeretném megköszönni az ipari- és tanszéki konzulenseimnek, Kiss Tamásnak és Oláh Istvánnak a tanácsokat, útmutatásokat, szakmai és egyéb kérdésekben.

7 Irodalomjegyzék

- [1] N. Kaushal: *Role of PLC in automation and its various applications*, International Journal of Enhanced Research in Science, Technology & Engineering, Vol. 4 Issue 7, 2015 július, 38-41. oldal
- [2] Hugh Jack: *Automating Manufacturing Systems with PLCs*, Version 5.0, 2007. május 4., 2. és 8. fejezet
- [3] Siemens: *SIMATIC S7-1500 CPU 1516-3 PN/DP*, <https://support.industry.siemens.com/>, A5E03461313-AD, 11-24. oldal
- [4] Siemens: *SCALANCE X – Industrial Ethernet Switches*, <https://support.industry.siemens.com/>, brosúra, 6-7. oldal
- [5] Siemens: *ET 200SP Interface module IM 155-6 PN ST*, <https://support.industry.siemens.com/>, A5E03576904-AD, 8-10. oldal
- [6] Siemens: *ET 200SP DI 8x24VDC ST digital input module*, <https://support.industry.siemens.com/>, A5E03574157-AC, 8-11. oldal
- [7] Siemens: *ET 200SP Digital output module DQ 8x24VDC/0.5A ST*, <https://support.industry.siemens.com/>, A5E03574575-AC, 8-11. oldal
- [8] Siemens: *ET 200SP Technology Module TM Count 1x24V*, <https://support.industry.siemens.com/>, A5E33002339-AA, 9-29. oldal
- [9] Siemens: *S7-1500, ET 200SP, ET 200pro Cycle and response time*, <https://support.industry.siemens.com/>, A5E03461504-AC, 20. oldal
- [10] Dynapar: *Quadrature Encoder Overview*, http://www.dynapar.com/Technology/Encoder_Basics/Quadrature_Encoder/, 2016. dec. 2.
- [11] PI North America: *PROFINET, Industrial Ethernet for advanced manufacturing*, <http://us.profinet.com/technology/profinet/>, 2016. nov. 23.
- [12] PI North America: *The difference between PROFIBUS and PROFINET*, <http://us.profinet.com/the-difference-between-profibus-and-profinet/>, 2016. nov. 23.
- [13] Fossbytes: *What is ring topology? Advantages and Disadvantages of ring topology*, <https://fossbytes.com/what-is-ring-topology-advantages-and-disadvantages-of-ring-topology/>, 2016. nov. 25.
- [14] Network Cabling Help: *Token Ring*, <http://www.datacottage.com/nch/troperation.htm#.WEhs59UrK01>, 2016. nov. 25.

- [15] Michael Sipser: *Introduction to the Theory of Computation*, 2nd edition, ISBN 0-534-95097-3, Thomson Course Technology, 1.1 fejezet