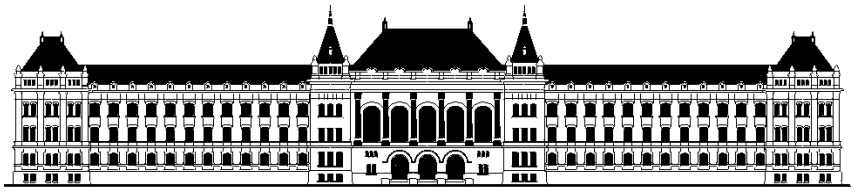


Budapesti Műszaki és Gazdaságtudományi Egyetem

Közlekedésmérnöki és Járműmérnöki Kar

Közlekedés- és Járműirányítási Tanszék

Intelligens jelzőfej alkalmazása a közúti forgalom-irányításban

Jenes Géza

KK5QP1

Konzulensek:
Dr. Tettamanti Tamás
Ludvig Ádám

2013. november 4.

Tartalomjegyzék

1. Bevezetés:

A közlekedés bármely területét vizsgálva elmondhatjuk, hogy a költségek csökkentése mindig a kitűzött célok között szerepel. Természetesen ez nem mehet a minőség és a hatékonyság rovására, hanem egy olyan új állapot létrehozását kell jelentse, amely a régit teljes mértékben helyettesíti, és akár tudásában is túlmutat rajta. A tudomány előrehaladásával néha a meglévő, bár megbízható rendszereket, érdemes újakkal helyettesíteni, amelyek a jelenkor technikája által nyújtott előnyöket jobban kihasználják. A dolgozatom célja, hogy egy napjainkban jól működő rendszert, egy új, korszerű komponens bevonásával egy kicsit átalakítsa, és vele járó előnyöket, hátrányokat a régi rendszerhez viszonyítva megvizsgálja. Témája a jelzőfej „intelligensé" tételét tanulmányozza. A jelzőfej, mely csak a jelzésre szolgáló izzókat hordozza magán, úgy válhat intelligensé, ha az izzók kapcsolási vezérlését átveszi, továbbá a feszültség- és áramellenőrzési funkciókat logikailag továbbítja a vezérlő berendezés irányába. Egy jelzőfejhez kapcsolt számítógép képes ilyen feladatot elvégezni. Így a jelzőfej egy olyan eszközzé válik, melynek konkrét feladata is van a rendszerben és nem csak kimenetként szolgál. Felmerülhet persze a kérdés, hogy minek a jelzőfejbe számítógép, azaz miért szükséges „okos jelzőfej", hiszen eddig is jól működött a forgalomirányítás, és mindez némi többletköltséget eredményez a meglévő rendszerkialakításokhoz képest. Bár az előbbi állítás jogosnak tűnhet, ellenben meg kell vizsgálnunk az új rendszer nyújtotta előnyöket is. Ha ezek az előnyök képesek kiváltani, a régi rendszer hátrányának titulálható problémákat, akkor már érdemes ezt az új kialakítást fejlesztés alá vonni. Az általam készített projekt a forgalomirányító berendezés és a jelzőfejek kábeles kapcsolatában hoz áttörést, ugyanis képes ezt kiváltani egy egyszerűbb, olcsóbb kapcsolatra. Ez természetesen sok munkát von maga után, ugyanis a jelzőfejre egy kis logikai modult kell építeni, és a jelzőfej vezérlését teljesen újra kell tervezni. Egy átalakítás mindig sok problémával jár, melyekkel én is szembesültem a fejlesztés során. A TDK munkám során nem csak elméleti kutatást végeztem, hanem az intelligens jelzőfej fizikai megvalósítását is elvégeztem (Actros típusú forgalomirányító berendezés, PLC-k, és LED-es jelzőfejek segítségével). Dolgozatomban ezen új rendszer, azaz az intelligens jelzőfejekkel történő forgalomirányítás feltételeiről, megvalósításáról, az új rendszerben fellépő problémákról és azok kezeléséről írok. Továbbá foglalkozom még az új rendszer biztonságával és hatékonyságával, valamint a régi rendszerrel való összehasonlításával.

2. A közúti jelzőlámpás forgalomirányítás

A közúti közlekedés irányítása a felgyorsult világban döntő szerepet tölt be. A közlekedés egyik fő feladata, hogy az emberek helyváltoztatási igényeit kielégítse, így egyik legfontosabb hatása, hogy befolyásolja az emberi időfelhasználást. Szükségessége lehetővé teszi a lakóhely és a munkahely elkülönülését, kielégíti a termelés, az elosztás és a fogyasztás szállítási, utazási igényeit, továbbá elősegíti a szabadidő felhasználását. [1]
A helyváltozatási igény egyre nő, így az utakon megjelenő járművek száma is növekszik. A közforgalmú közlekedést nem elegendően veszik igénybe, és nagyon sokan egyedül utaznak egy járműben, ezért a közúti járműforgalom nagysága jelentősen megugrott az utóbbi évtizedekben. Ez magával ránt több problémát is, mint például fokozott balesetveszélyt, megnövekedett környezet terhelést, forgalmi torlódásokat. Az eljutási idő és az átlagsebesség csökken, a megállások száma nő. Az emberekben emiatt sokszor nő a stressz, ami balesetekhez vezethet. Ezeket a problémákat valahogy kezelni kell, hogy a városi közlekedés gördülékenyebben működjön. Ahhoz, hogy elkerüljük az utakon történő baleseteket, csökkentsük a torlódásokat, illetve azok externális hatásait, továbbá az emissziót és a zajt is visszafogjuk, a közúti közlekedést irányítani kell.
A jelzőlámpás szabályozás lokális vagy hálózati szintű alkalmazása hozzájárulhat egy zavartalanabb forgalomlebonyolódáshoz. A különböző forgalmi irányok közötti kapcsolat a csomópontokban jelenik meg, ezért azt mondjuk, hogy ez a leggyengébb láncszem. Itt a szűk keresztmetszet miatt megjelenik a forgalmi akadályoztatás, ami időveszteséget, fokozott balesetveszélyt és jelentős környezetterhelést von maga után.
A jelzőtáblák, jelzőlámpák és az útburkolati jelek azok a szabályozási eszközök, melyeket a közúti közlekedésben kialakított szabályok közül adott időben, helyen, helyzetben, illetve járművel és gyalogosként is be kell tartani a közúti közlekedésben [1]. A közlekedésben résztvevőkre vonatkozó szabályokat, előírásokat, törvényeket és rendeleteket hazánkban a KRESZ foglalja össze.
Jelzőlámpás irányítást csak olyan esetben alkalmazhatunk, ha adott csomópont a jelzésképek hiányában jelzőtáblás irányításra is alkalmas. A jelzőlámpás irányítás alkalmazásának indokoltsága összetett kérdés, de van néhány irányadó alapelv. Első és legfontosabb szempont, ha egy adott csomópontban rendszeresen hasonló típusú balesetek történnek. Ilyenkor javasolt a jelzőlámpa bevezetése. Továbbá főútvonalak csomópontjain, négy és több forgalmi sávú utak találkozásakor, ha hosszú a várakozási idő az alárendelt úton, vagy mindkét irányban villamos közlekedik, illetve veszélyes gyalogátkelőhely

esetén. Összehangolt rendszer kialakításánál egy adott hálózaton is elengedhetetlen a jelzőlámpás irányítás.[1]

A jelzőlámpákat több szempont alapján lehet csoportosítani:

- szabványos lencseátmérő alapján: 100 mm, 200 mm, 300 mm – ezt a láthatóság határozza meg,
- tele zöld vagy maszkos zöld (lásd 1. ábra),

1. ábra: Tele zöld és maszkos zöld lámpák

(forrás: Debreczeni Gábor: Közúti Forgalomtechnika [elektronikus jegyzet] 46.o.)

- három-, kettő- vagy egyfogalmú készülék (lásd 2., 3., és 4. ábrák),

2. ábra: Háromfogalmú jelzőberendezések

(forrás: Debreczeni Gábor: Közúti Forgalomtechnika [elektronikus jegyzet] 47.o.)

3. ábra: Kétfogalmú jelzőkészülékek

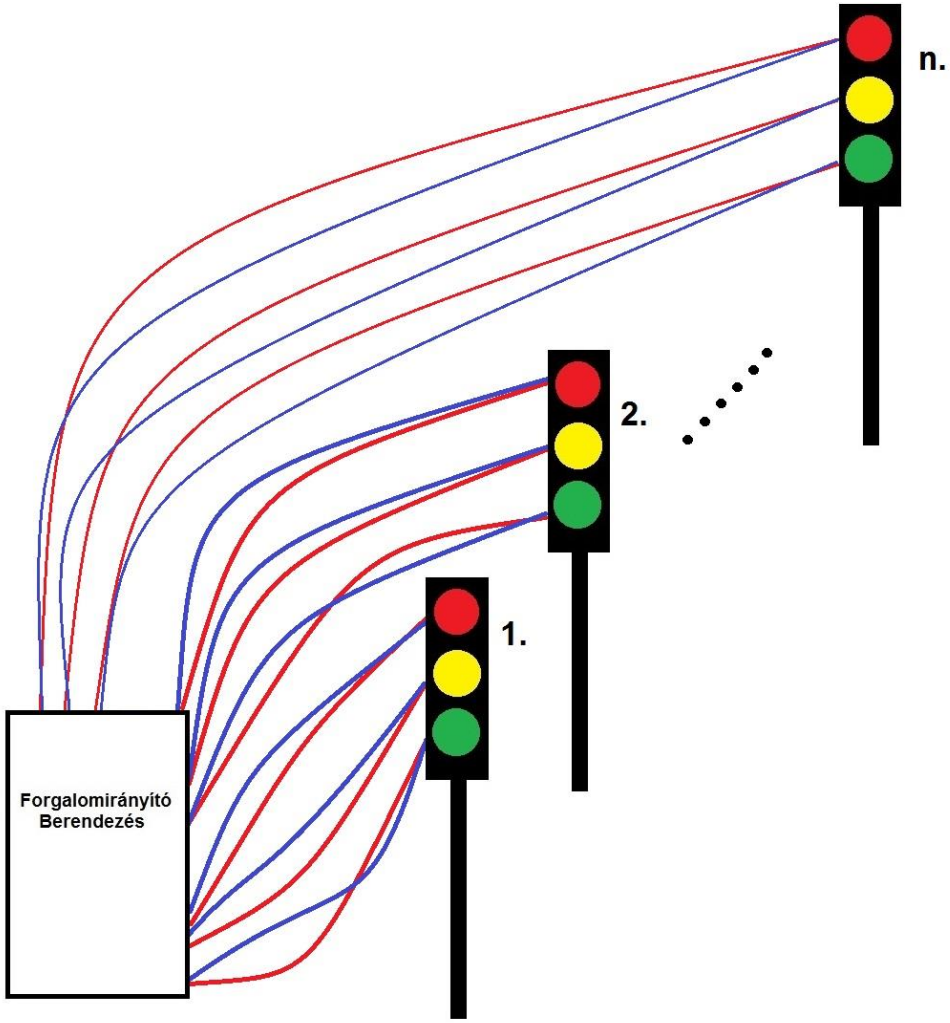
(forrás: Debreczeni Gábor: Közúti Forgalomtechnika [elektronikus jegyzet] 49.o.)

4. ábra: Egyfogalmú jelzőkészülékek

(forrás: Debreczeni Gábor: Közúti Forgalomtechnika [elektronikus jegyzet] 49.o.)

- a jelzőlámpák világítóteste alapján: hagyományos izzó vagy korszerű világító dióda (LED: Light Emitting Diode [2]).

A jelzőlámpás irányítástechnika eszközei a jelzőfejek, valamilyen forgalomirányító berendezés, valamint a köztük lévő kommunikációhoz szükséges átviteli közeg. A dinamikusan vezérelt csomópontoknál szükség van valamilyen járműérzékelő detektorra is. A napjainkban működő rendszereknél a jelzőfejek közvetlen, kábeles kapcsolatban állnak a forgalomirányító berendezéssel. Minden egyes jelzőfej, minden izzójához ki kell húzni egy vezetéket és visszavezetni forgalomirányító berendezésbe, hogy az áramkörön, mint ellenállás működjön az izzó. Ez a fajta kiépítés rengeteg vezeték szükségességét vonja maga után a jelzőlámpával irányított kereszteződésben (lásd 5. ábra).



5. ábra: A jelzőlámpás forgalomirányítás kábelezése sematikusan

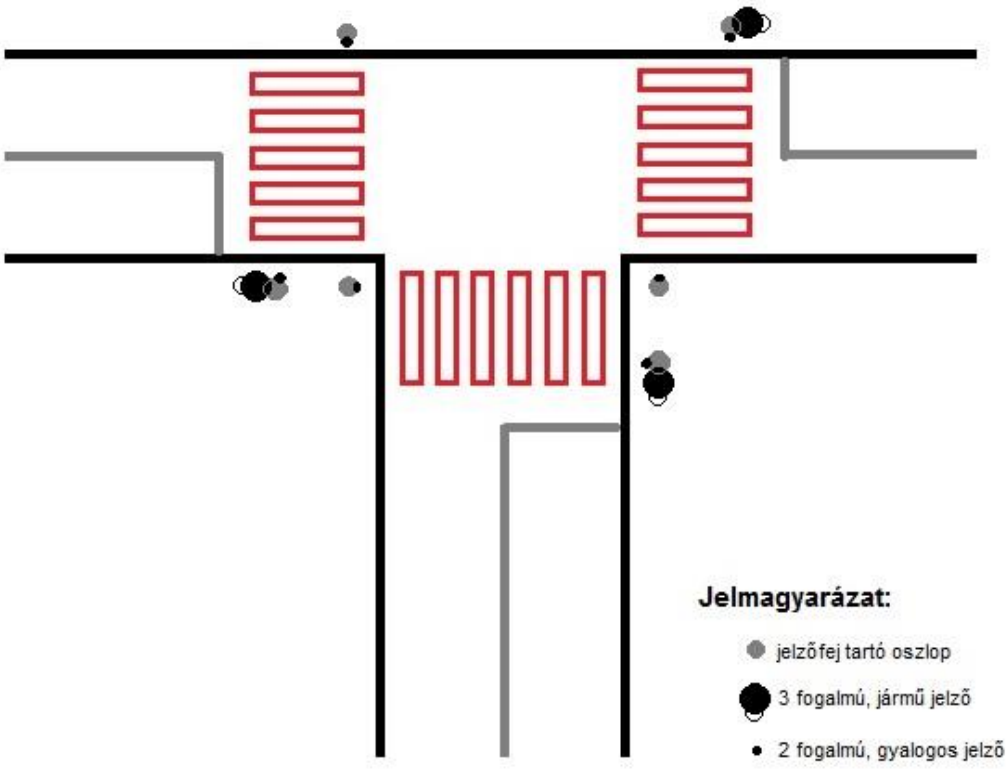
Napjainkban működő jelzőfejek vegyesen hagyományos izzóval illetve LED izzóval felszerelt eszközök. A LED-del szerelt jelzőfejeknek számos előnye van. Nincs betekintési szögből eredő fénytévesztés, hosszabb az élettartam, gazdaságosabb az üzemeltetés. Az izzókkal szemben ezeket a rendszereket általában nem 230V-ról, hanem 40V-ról kell táplálni, ami többször problémát jelentett például a budapesti LEDesítési projektben. Léteznek 230V-os LED-es izzók is, de a LED-es jelzőfejek fő előnye az érintésvédelmi szempontból, hogy kis feszültségről üzemelnek, így ma már leginkább a 40V-os megoldást választják. Egy 40V-ról működő rendszer viszont sokkal érzékenyebb például a felszültség ellenőrzésre, ami a jelzőlámpás ellenőrzés egyik alap művelete. Ezzel kell a zöld égését, árammal pedig a piros izzó megfelelő működését ellenőrizni. Alacsonyabb feszültségen 1-2V-os kilengést már nem hanyagolhatunk el, míg például a 230V-os meghajtásnál ez nem jelentett problémát. Ez azt eredményezi, hogy sokkal jobb minőségű kábelezést igényel ez a rendszer. Például egy felújítást végző projektnél nagy prob-

lémát tud eredményezni, hogyha a megrendelő csak a jelzők cseréjét tervezte, de kiderül, hogy a föld alatt lévő teljes kábelezést cserélni kell. Ezen rendszerek további hátránya, hogy burkolatjavítási munka során gyakran elvágják az alépítményi vezetékeket, amelyekből egy nagyobb csomópont esetén igen sok fut a földben. Ez súlyos károkat tud eredményezni, hiszen minden átvágott kábelt javítani vagy pótolni kell.

3. Az intelligens jelzőfej alkalmazásának lehetősége a közúti forgalomirányításban

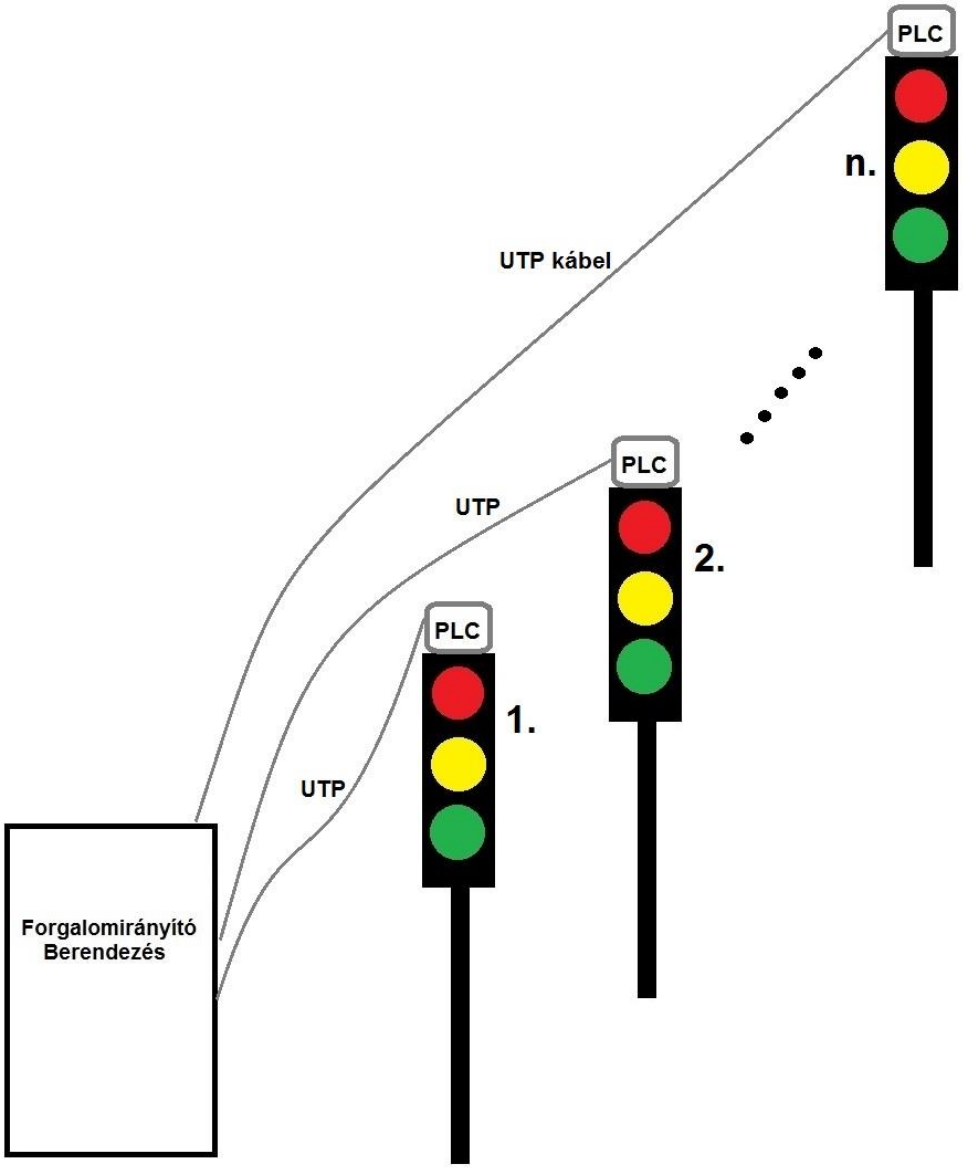
Dolgozatom alap ötlete, hogy a jelzőfejek és a forgalomirányító berendezés közötti kábeles kapcsolatot valamilyen módon egyszerűsítsük, a vezetékek számát és hosszát lerövidítsük. A jelzőlámpás forgalomirányítás általánosságban egy irányító berendezésből és jelzőfejekből tevődik össze. Ebben a struktúrában minden működéshez kapcsolódó feladatot (irányítás, kommunikáció, ellenőrzések) a forgalomirányító berendezés lát el. A jelzőfejek feladatköre kizárólag a jelzésképek megjelenítése. Ez egy logikus kialakítás, hiszen a rendszer biztonsága könnyebben és jobban ellenőrizhető, ha csak egy komponens végez irányítási feladatokat. Ugyanakkor az egyes elemek közötti kapcsolat meglétéhez rengeteg kábelezésre van szükség, amely egy nagyobb csomópont esetén elég nagy költségeket tud maga után vonni. Egy vezeték mérete 4-től akár 30 erűig is terjedhet, mely az árukat nagyban meghatározza. Az érszámot azon ponton lévő jelzőfejek száma határozza meg, ahova a kábelt ki kell húzni.

Vizsgáljunk meg a példa kedvéért egy egyszerű T alakú csomópontot gyalogátkelőhelyekkel! A 6. ábrán látható kereszteződésben minimum 3 közúti háromfogalmú jelzőberendezés megléte szükséges a forgalomirányításhoz (és ez a legegyszerűbb eset, mert nincs ismétlőjelző belógatva az út fölé), továbbá 6 kétfogalmú jelzőberendezés a gyalogos átkelőhelyek irányításához. Ehhez a 9 jelzőhöz mind több erű vezetéket kell kihúzni. Ha csak darabszámra számoljuk, 9 x 3, azaz 27 vezetékre van szükség, és akkor még nem is néztük a vezetékek méretezését és hosszát, amit a forgalomirányító berendezéshez képesti helyük határoz meg.



6. ábra: Egy általános forgalomirányítású, T alakú csomópont

Bevezetem az intelligens jelzőfejeket a rendszerbe. Célom megvalósításához Siemens PLC (Programmable Logic Controller [3]) eszközöket használok fel. Ez a programozható logikai vezérlő átveheti a jelzőfejek izzóinak kapcsolási feladatát a forgalomirányító berendezéstől, hiszen programozható relékimenetekkel rendelkezik. Az általam tervezett rendszerben, minden jelzőfejhez egy ilyen PLC-t csatlakoztattam (azonos jelzésképet adó jelzőfejeknél elég egy darab vezérlő egység is), melyek kapcsolatban állnak a csomópont forgalomirányító berendezésével, de az izzók kapcsolgatását ők maguk végzik. Ezzel a koncepcióval megszüntethetem a fölösleges vezetékezést, hiszen a jelzőfejek izzóit a PLC-kbe kötöm be, mely magán a jelzőfejen kap helyet, így az elektromos vezetékek hossza ott csupán 20 – 30 centiméterre redukálódik. A forgalomirányító berendezés és a PLC kapcsolatához pedig mindösszesen egy darab UTP kábelt kell csak kihúzni (lásd 7. ábra). Természetesen a logikai modulok tápellátását is biztosítani kell, de erre elég egy gerinchálózaton végigvezetett 4 erű kábel.



7. ábra: A jelzőlámpás forgalomirányítás intelligens jelzőfejjel vázlata

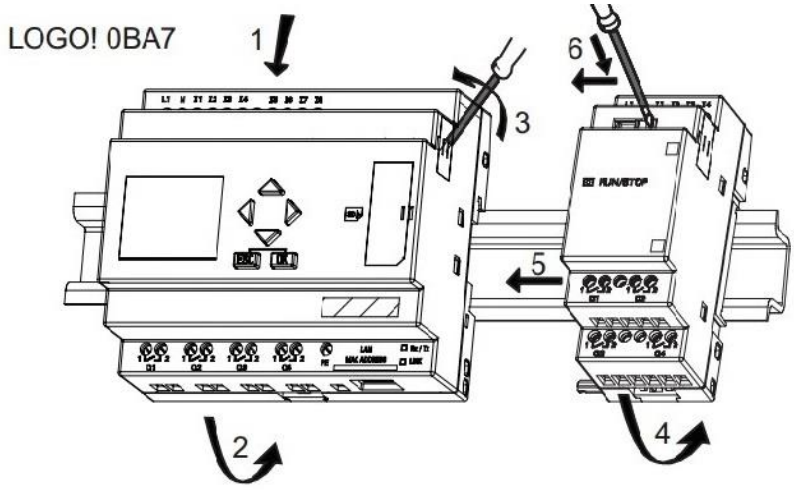
Ezzel jelentős költségmegtakarítást érhető el, mert a több erű kábelek, csak a logikai modul és az izzó közé szükségesek, ami pár tíz centiméteres szakasz. A forgalomirányító berendezés és a logikai modul közé pedig egy jóval olcsóbb UTP kábel kerül. A megoldásnak előnyös lehet az ideiglenesen alkalmazott jelzőlámpás forgalomirányításban is - például forgalmi akadály vagy útépítés miatt kihelyezett jelzőknél. Ilyenkor az ideiglenes rendszer telepítése sokkal gyorsabb és egyszerűbb lehet. Forgalmi terelés esetén akár a vezérlő gép is elhagyható, mert annyira egyszerű a feladat, hogy azt képes a PLC egymagában megoldani.

4. A rendszerhez szükséges hardverek és szoftverek

A projekt megvalósításához több lépésben tudtam csak eljutni. Elsőként meg kellett oldanom a jelzőfej PLC-ről való irányítását, majd meg kellett valósítanom a forgalomirányító berendezés és a PLC közötti kommunikációt. Ahhoz, hogy tisztában legyünk a rendszerben működő komponensekkel először minden felhasznált hardvert bemutatok, melyet a megvalósításnál felhasználtam. Ezek főleg a tanszéki labor eszközei, de van köztük olyan is, amelyet én magam készítettem. Továbbá ismertetem a szükséges szoftvereket is.

4.1. A Siemens LOGO! 12/24 RCE és tápegysége

A LOGO! a Siemens cég univerzális logikai modulja, PLC-je, melyet a 8. ábra szemléltet. Képes bonyolultabb logikai feladatok megoldására és hálózati kommunikáció is kivitelezhető vele, ezért került ez a típus felhasználásra. A közúti laborban összeállított rendszeremben 3 darab LOGO!-t használtam fel. Ebből kettő két közúti jelzőfejet irányított, és a harmadik pedig egy általam készített kis mini jelzőkészüléket.
A LOGO!-t egyenárammal kell táplálni 12 V vagy 24 V feszültségen. A jelzőfejeket irányító LOGO!-k fel vannak helyezve egy sínre, és a melléjük csatolt Siemens tápegység látja el őket. A bekötés menetében a 8. ábrán látható számozott nyilak segítenek. A harmadik PLC-t (ami az általam készített LED-es jelzőfejet irányította) egy univerzális, változtatható egyenfeszültséget szolgáltató tápegység látta el (12 V 2,5 A).

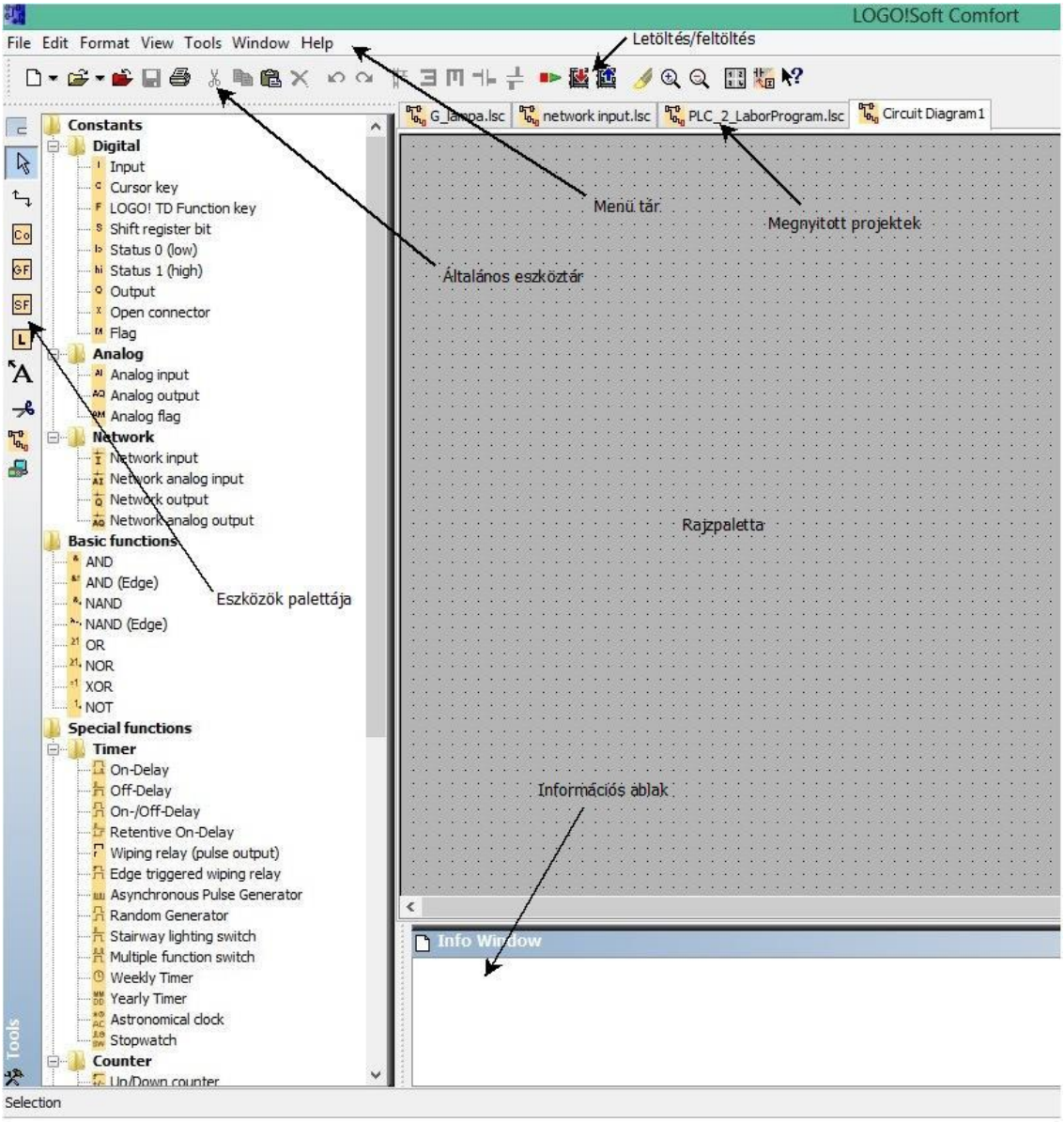


8. ábra: A LOGO! PLC és tápegysége

(forrás: SIEMENS LOGO! V5 manual)

4.2. Siemens Logo Soft Comfort

A Siemens külön szoftvere a PLC-k programozásához, mely egy igencsak felhasználóbarát szoftver (lásd 9. ábra). Ezzel a programmal lehetőség van a vezérlőprogramok PC-n történő megírására. A program a PC-n könnyen kipróbálható, változtatható, tárolható és ki is nyomtatható.



9. ábra: A Logo Soft Comfort általános felépítése

(forrás: Logo Soft Comfort V.7. [program])

A programozáson jelen esetben valamilyen áramköri kapcsolás bevitelét értem. Egy LOGO! program tulajdonképpen nem más, mint az áramkör kapcsolási rajzainak másféle módon történő megjelenítése. A programozása nem kifejezetten nehéz, mivel a

LOGO! programja előre definiált funkciójú blokkokból épül fel. A logikai hálózatot úgy kell felépíteni, hogy a bemenetek és a kimenetek közé a megfelelő blokkokat rakjuk a megfelelő kapcsolatokkal. A be- és kimenetek az alap- és különleges funkciólistában találhatók és az úgynevezett „drag and drop” módszerrel a vezérlőprogramba fűzhetők, illetve tetszés szerint összeköthetők és eltolhatók. [4]
A grafikus felhasználó felület nagyban hozzájárul a felhasználóbarát kezelhetőséghez. „Rajzpalettán” tudjuk létrehozni a programrészleteket és összekapcsolni őket. A be- és kimeneteket ugyanúgy, mint a funkcióblokkokat el lehet látni kommentárral, ami sok esetben megkönnyíti a program átláthatóságát. Az elkészült áramköri kapcsolást szimulálhatjuk a szoftver segítségével, így könnyen ráismerhetünk a hibákra, problémákra.

4.3. Forgalomirányító berendezés

A forgalomirányító berendezés egy Actros VTC 3000 típusú vezérlő, amely laborsegédeszköz (lásd 10. ábra). A berendezés - korszerű technológiájának köszönhetően - a hagyományos értelemben vett jelzőlámpás forgalomirányítási feladatok elvégzésén túl, gyakorlatilag bármilyen közúti forgalomszabályozás vezérlésére alkalmas [5].

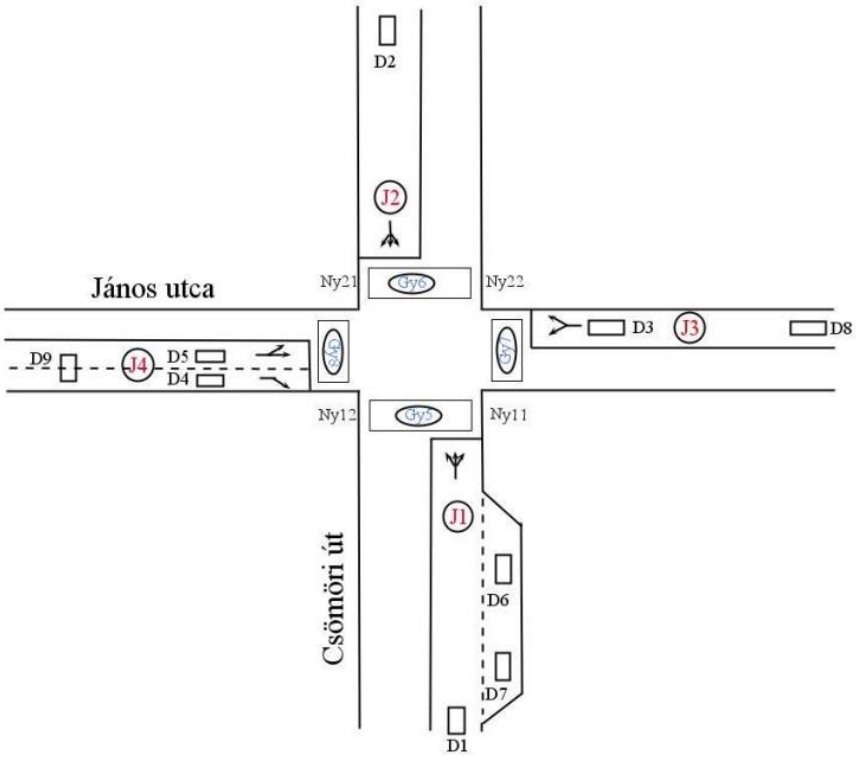
10. ábra: Az Actros VTC 3000

A forgalomirányító berendezés feladata, hogy adott csomópont jelzőfejeinek jelzésképeit vezérelje. A csomóponton működő programokat a berendezés irányítja és váltogatja az aktuális forgalomnak megfelelően.

A gép egy budapesti csomópont forgalomtechnikai tervét tartalmazza (Bp. XVI kerület Csömöri út - János utca csomópont), és ennek megfelelően úgy futtatja a programokat a laborban is, mint a terepi berendezés. A forgalomtechnikai kód 4 programot tartalmaz: 3 darab fixprogramot és 1 darab forgalomfüggő logikát. Az intelligens jelzőfejjel működő rendszer esetén ebből egy darab fix jelzéstervű programot használtam fel, és a gépet minden esetben ebbe vezéreltem.

4.4. Eclipse

Az Eclipse egy olyan fejlesztő szoftver mely hatékonyan támogatja a Java környezetben való programozást. Működéséhez elengedhetetlen, hogy telepítsük mellé a Java Development Kit (JDK)-et is, ami egy fordító a Java SE, Java ME, és Java EE platformokra. A program felépítése nagyban hasonlít az általános programozói felületekhez.
A példaprogramban a Csömöri út és a János utca csomópontjára már megírt kódokat használtam. A csomópont vázlata a 11. ábrán látható. Természetesen a PLC vezérelt rendszer felépítéséhez lényegében bármilyen csomópontot választhattam volna, hogy a rendszert működésbe hozzam.

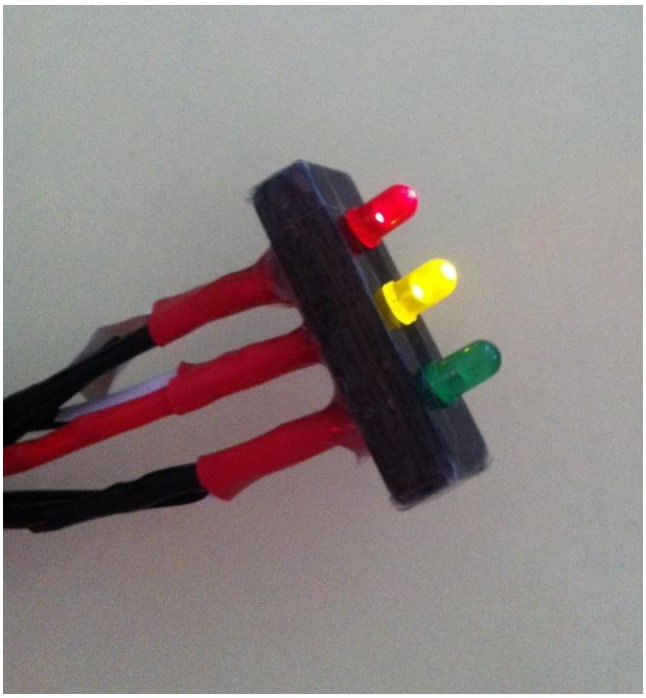


11. ábra: Csömöri út és János utca csomópont sematikus ábrája

(forrás: Polgár János, Tettamanti Tamás: Forgalomtechnikai kód az ACTROS VTC 3000 forgalomirányító berendezésben [elektronikus jegyzet] 4. o.)

4.5. Világító diódákból összeállított jelzőfej

A három diódából (piros, sárga, zöld) összeállított jelzőlámpa egy általam készített mini jelzőfej, amely megjelenítésre, ellenőrzésre alkalmas (lásd 13. ábra). A diódákra felforrasztottam a megfelelő előtét ellenállásokat és egy balsafa tokba ragasztottam őket, ami a lámpatestet adja. Ez az egység így beköthető a LOGO!-ba. Tápegységgel csak a PLC-t kell ellátni, mely meg fogja szakítani az áramkört (a rátöltött logika alapján). Így megkapjuk a kívánt jelzésképet. Ez az egység a labor két szabad jelzőfején kívül további lehetőséget ad a rendszerbe bevonható elemszámokhoz, ezáltal három PLC-t is fel tudtam használni.



13. ábra: A világító diódákból készült jelzőfej

4.6. További hardverek

- Laborban található jelzőfejek: egy darab 230 V-os Elba típusú és egy korszerű 40 V-os Siemens fej (mindkettő LED-es).
- Router és Hub eszközök

5. A LOGO! és a Java együttműködése

Ahhoz, hogy a LOGO!-kat Eclipse alól programozni tudjam, kompatibilissé kellett tennem a LOGO!-t a Java-val. Egy kiegészítést kellett írni az eszközhöz, amit aztán Eclipseben használt csomagjaimhoz hozzá importáltam. Ezt a kiegészítést egyik konzulensem, Ludvig Ádám készítette, aki már foglalkozott korábban ezzel a témával. Ő egy interneten elérhető nyílt forráskódot dolgozott át. Ez az úgynevezett libnodave szoftverkönyvtár. Ennek felhasználásával fel tudtam állítani a kommunikációt az Eclipse és a LOGO! között, bár ennek megléte nem egyértelműen vezetett a megoldáshoz.

„A libnodave könyvtár a hivatalos Siemens PRODAVE szoftverkönyvtár kiváltására készülő, ingyenes, nyílt forrású szoftver komponens, ami a Siemens PLC eszközökkel való hálózati kommunikációt lehetővé teszi. A csomag több PLC típust és többféle kommunikációs módot támogat. A szoftverkönyvtár C nyelven íródott és aktív fejlesztés alatt áll. Többféle platformra lehet lefordítani, az elsődleges platformok Linux és Windows.” [6]

A kapcsolat felállításához Logo Soft Comfortban további beállításokat kellett elvégeznem. A kommunikáció Etherneten keresztül történt, ezért is választottam a LOGO! 0BA7 típusát, mert az képes hálózati kapcsolatok kezelésére és van Ethernet csatlakozó rajta.

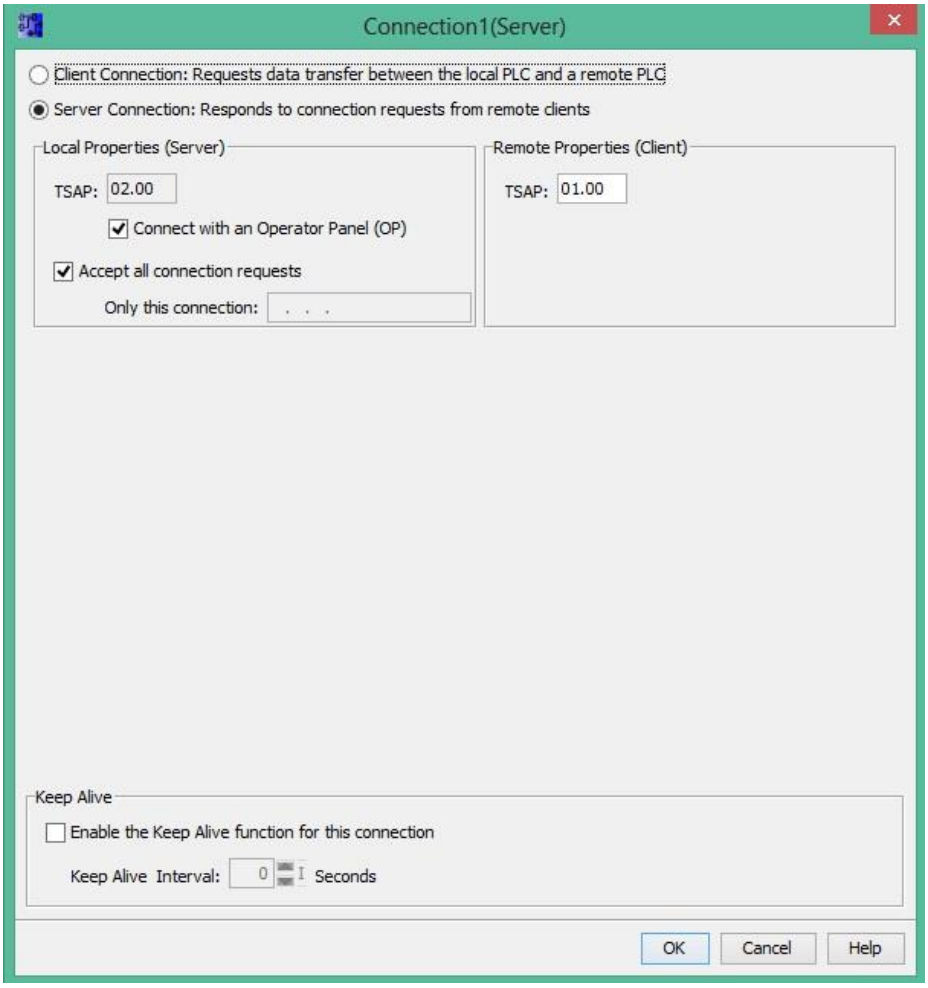
A következőkben a LOGO! kommunikációs beállítását foglalom össze, amely a libnodave könyvtár használatához szükséges. A LOGO! és az Eclipse együttműködéséhez a Logo Soft Comfortban kommunikációs beállításokra van szükség. Ezeket a Tools menü Ethernet Connections… fül alatt találhatóak meg. Elsőként az IP címet kellett megadnom. Ez mindig az a cím, amelyik PLC-vel a kapcsolat fenn áll. Ezek után az alattuk lévő ablakban az „Ethernet connections” szövegre jobb gombbal klikkelve „Add connection”-t választva létrehoztam egy új kapcsolatot, amit a saját feladatomnak megfelelően tudtam beállítani. A Connection1 (Server) –re kettőt kattintva tulajdonságok konfigurálhatóak.

A következő beállítási paramétereket a 12. ábra szemlélteti.

- A „Server Connection” típust választottam a rádió gombok közül. Ekkor a LOGO szerverként viselkedik és fogadja a hálózati csatlakozásokat. Kliens kap-

csolatnál egy másik Siemens eszköz IP-címét és TSAP[1]-ját beállítva, annak a VM[2]-jéből kérdezhetőek le értékek.

- Bekapcsoltam a „Connect with an Operator Panel (OP)” funkciót
- Bekapcsoltam az „Accept all connection request”-et. Így minden IP címről elfogadta a csatlakozási kéréseket a LOGO. Ez opcionális, ha korlátozni szeretném a LOGO elérhetőségét, akkor itt van rá lehetőség.



12. ábra: Szerver kapcsolat tulajdonságai a Logo Soft Comfortban

(forrás: Logo Soft Comfort V.7. [program])

[1]Egy csatorna azonosító szám, amivel egy eszközön egy kommunikációs csatorna kijelölhető, olyan, mint a TCP/IP kommunikációban a port szám, de ez eggyel magasabb OSI rétegben van. Bármire állítható két Siemens eszköz között.

[2] Variable Memory, a PLC-ken kialakítható egy, közös változóknak fenntartott memória rész, amiből lehet lekérdezni adatokat és írni rá adatokat. Ezen adatok jellemzője a cím (eltolás byte-ban) és a hossz (byte-ban).

- A „Remote properties (Client)” keret alatt a csatlakozó, kliens eszköz TSAP címét átállítottam 01.00-ra, mert a libnodave csak ezzel működik.
- A „Keep Alive” (életjel) funkcionalitást kikapcsoltam, mert ezt nem támogatja a libnodave könyvtár még. [6]

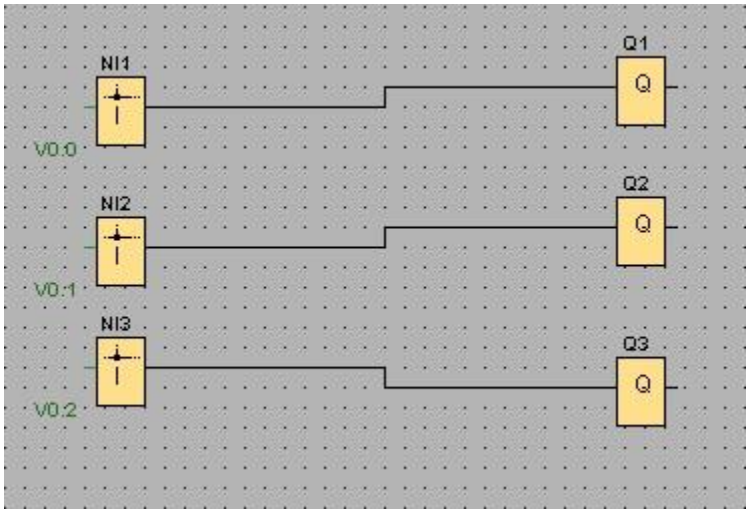
Miután elkészítettem a PLC-khez a progamjukat, a fenti beállítások elvégzésével töltöttem fel a LOGO!-kra, hogy azok működésbe lépjenek. A feltöltést Ctrl+D billentyűkombináció lenyomásával kezdeményeztem. A felugró ablakban megadtam annak a LOGO!-nak az IP címét, amire tölteni akartam a programot.

6. Az intelligens jelzőfejjel működő rendszer felépítése

A következőkben bemutatom a PLC-n futó programtervezési folyamatát, amelyet a Logo Soft Comfortban végeztem. Az alábbiak könnyebb megértéséhez a Mellékletben egy egyszerű példa található.

6.1. Jelzőlámpa váltás tervezése Logo Soft Comfortban, a LOGO! programja

A LOGO! és a Java együttműködéséhez, illetve az Eclipse alól történő programozásához elegendő az alábbi (lásd 13. ábra) egyszerű program is, ami 3 darab Network Input -al és 3 darab Output -al rendelkezik. A LOGO!-ra nem tölthetek fel egy olyan fix programot, ami egy jelzéstervet tartalmaz, mert akkor azt csak úgy tudnám változtatni, hogy a meglévő programot mindig törlöm a LOGO!-ról, és készítek egy más jelzéstervet tartalmazó programot Logo Soft Comfortban és azt a letörölt helyére visszatöltöm. Ez a művelet elég bonyolult és nem enged közvetlen hozzáférési lehetőséget a jelzésterv változtatására. Ezért a LOGO!-ra egy olyan programot készítettem, ami megteremti a közvetlen és folyamatos kapcsolatot az Actrossal, így alkalmas változó jelzéstervek kezelésére. Ezen blokk diagrammal a LOGO! tudásának csak egy szerény részét használtam ki, csak azokat a funkcióit, amivel a feladatokat elvégzi, azaz nyitja és zárja az áramköröket. Ez a program a 13. ábrán látható. A lámpa három színe miatt három bemenetre volt szükségem, azok közül is a hálózati tulajdonságok kezelésére alkalmasakra.



13. ábra: A kapcsolathoz szükséges blokkdiagram

(forrás: Logo Soft Comfort V.7. [program])

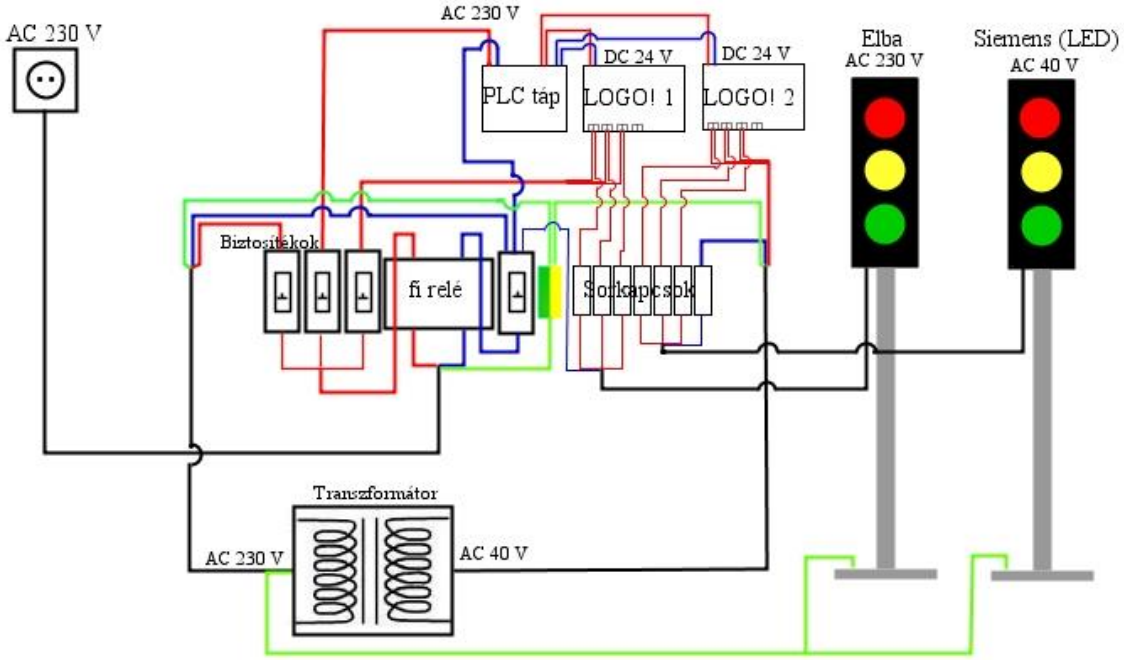
Ezeket a Network Input -okat elhelyeztem a rajzvásznon, majd bal egérgombbal duplát kattintva, eljutva a tulajdonságai közé beállítottam a VB address-ét. Ez azt mondja meg, hogy a LOGO! memóriájának, melyik címén dolgozik illetve, hogy azon belül melyik bitet kezeli. Az adatok a LOGO! VM tárolójába írhatóak, és abból később könnyedén ki is olvashatóak. Ez nagyon fontos, mert az Eclipsből így tudtam a megfelelő izzóra kiadni az utasítást. „A VM egyes byte-jaihoz a PLC program elemeinek tulajdonságait lehet kötni. Ezáltal ezeket a helyeket a VM-ben a PLC minden frissítési ciklusban feltölti az aktuális értékekkel [6].” A programom további elemei a három darab Output. Ezeket egyszerűen csak össze kellett húznom egy-egy ’Network Input’-al, így a program elkészítette köztük a kapcsolatot. A kimenetek a jelzőfej lámpáit szimbolizálják. A Kimenetek Block Properties -énél látható, illetve módosítható, hogy ennek mi a sorszáma. Ez adja meg, hogy melyik kimenet, melyik izzóhoz fog tartozni. Ezeket beállítottam sorban 1-től 3-ig.

A jelzőfej izzóit a PLC- megfelelő fizikai kimeneteire kellett kötöm, illetve az előbb beállított sorrendnek megfelelően, így a helyes képet kaptam a program futtatásakor. Ezzel nem fordulhatott elő olyan hiba, hogyha például a piros színt akartam felkapcsolni, akkor a zöld villant fel. A kész programmal már tudtam működtetni a rendszert Eclipse alól. A blokk diagram később tovább lett fejlesztve, hogy több funkció ellátására is alkalmas legyen. A programot fel kellett tölteni a LOGO!-kra.

6.2. A hardver rész összeállítása

A rendszer működőképessé tételéhez az első feladatom az volt, hogy a hardver oldali eszközök tökéletesen működjenek. A komponenseket össze kellett állítanom, egymáshoz kötnöm őket, melynek menetéről a következőkben fogok írni. A lépéseket 14. ábra szemlélteti. Az Elba típusú jelzőfejet 230 V-on, a Simens típusút 40 V-on kell táplálni. A hálózatról 230 V váltakozó feszültségen kaptam áramot. Ezt életvédelmi szempontból először egy fi-relébe vezettem, onnan pedig három felé vittem tovább, egy-egy biztosítékon keresztül. Elsőnek a transzformátorba, ami a 40 V-os váltóáram előállításához kellett, másodiknak a PLC-k tápegységébe, ami 24 V-os egyenáramot állított elő belőle, és harmadiknak pedig az első LOGO! Output -jára, ahonnan pedig tovább az Elba jelzőfejre vezettem. A vezetékek összekapcsolásakor legtöbbször sorkapcsokat alkalmaztam. Az előbbiekben szétágaztatott 230 V-os első ágat a transzformátor primer ágára vezettem, így a szekunder ágban 40 V váltakozó feszültséget kaptam. Ezt rávezettem a má-

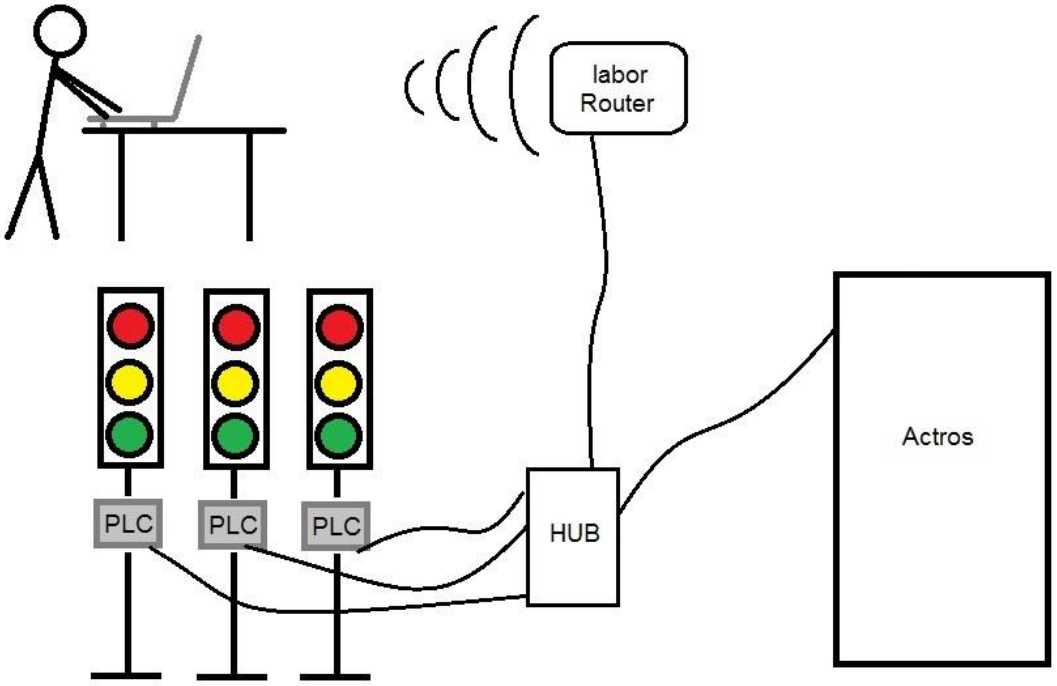
sodik LOGO! kimenetére, ahonnan pedig tovább a Siemens jelzőfej izzóira. A fázist mindig piros színnel jelöltem, a 0-át kékkel, a földet pedig zöld színnel.



14. ábra: A rendszer elektrotechnikai összeállításának sematikus rajza

A LOGO!-kat rákötöttem a tápegységre. A bemeneti oldalon az ’L+’ jelzésre kötöttem a PLC tápból érkező 24 V egyenfeszültséget, az ’M’ jelzésű bemenetre pedig a földpontot. A LOGO! lényegében bemenetei és a kimenetei közötti áramkört szakítja meg és zárja össze, az előre megírt program alapján. A PLC-k kimeneteire rákötöttem a jelzőfejeket. Ennek menete, hogy a kimenetek első csatlakozóira kötöttem a tápláló feszültséget, majd a második csatlakozóról továbbhúztam a kábelt a jelzőfejek izzóihoz. Ezzel a PLC-k és a jelzőfejek kapcsolata létrejött és az áramellátásuk is biztosítva lett.

Továbbá a LOGO!-kat UTP kábel segítségével rákötöttem a labor hálózatára (lásd 15. ábra). Ez nem közvetlenül történt, ugyanis először egy Hub-ba kötöttem őket, majd a Hub-ot a labor Routerére. Az Actrost is be kellett vonni a rendszerbe. Neki saját hálózati kártyája van Ethernet csatlakozóval, így őt szintén egy UTP kábellel be tudtam kötni a Hub-on keresztül a labor hálózatára. Ezzel a teljes rendszer hálózatba került, és az irányítást számítógépről vezérelni tudtam. A 15. ábrán látható egy vázlat a rendszer hálózati kapcsolatairól.



15. ábra: A rendszer hálózati kapcsolatainak sematikus rajza

A LOGO!-k elindításához nem kellett más, minthogy feszültség alá helyezzem őket. Ilyenkor, ha van rá töltve program, akkor az azonnal elindul. Ez a kezelő gombjaival leállítható, és akkor hozzá lehet férni a LOGO! beállítási lehetőségeihez. A menüben a Network opción belül található az IP Address. Itt beállítottam, minden LOGO!-nál egy IP címet, amivel el tudtam érni őket egyesével. [4]
A következő IP címeket használtam: 192.168.0.212; 192.168.0.213; 192.168.0.214.

6.3. Az Eclipse és a program elemei

A Csömöri út és János utca csomópontra megírt VT-packege-t fel tudtam használni a LOGO!-s projektem elkészítésében. A meglévő program java fájljai jó alappal szolgáltak. A PLC bevonása miatti változásokkal kellett kiegészíteni a kódokat, illetve néhány új résszel.
Az Eclipsben végrehajtandó feladatokkal lépésenként haladtam. Elsőként nyitottam egy új projektet. Ezt a File – New – Project fül alatt találtam. A felugró ablakból a Java Project-et választottam, majd be kellett állítanom a mentés helyét. Ezzel létrejött a projektem. Következő lépésben beimportáltam a szükséges Package-eket és fájlokat. Ezek közé tartozott az 5. fejezetben említett libnodave szoftverkönyvtár, illetve a Ludvig Ádám által elkészített logo.java fájl is, ami a LOGO! hálózati kapcsolatának felállításához elengedhetetlen. Az importálási lehetőséget a program bal oldalán elhelyezkedő

Package Explorer részben, a jobb egér gombbal kattintásával értem el. A felugró ablakban kiválasztottam a behozandó fájl típusát, majd végig kellett mennem az importálással kapcsolatos beállításokon. Ezek után ezt a projektet exportáltam egy JAR fájl-ba, ami a kapcsolat felépítéséhez kellett később. Ezt szintén a Package Explorer -ből végeztem.
Ezek után nyitottam egy új projektet, amiben már a tényleges munkámat végeztem. Elsőként az előzőekben kimentet JAR file-t elérési útvonalát kellett beállítanom. Ezt a Projects – Properties fül alatt, majd a felugró ablakban a Java Build Path – Libaries – Add external JAR… menüpont alatt tudtam elvégezni. Ezzel a kapcsolatot az Eclipse oldaláról elkészítettem.
A Logo Soft Comfortban is be kellett állítanom a kapcsolat tulajdonságait, amit már az 5. fejezetben bemutattam. Ennek megléte esetén, a kapcsolatot a LOGO! oldaláról is késznek tekinthettem, így elkezdhettem programozni Eclipseben. A következőkben a projektem java fájljait fogom bemutatni, azok fontosságának részletességével.

6.4. Var.java

A Var, azaz Variable, magyarul változók rövidítése. Ebben a programrészben az általánosan, a program minden részében felhasznált változókat kell definiálni. Ezek a változók public, azaz publikus, mindenki számára elérhető, és static, azaz statikus - értéküket nem változtatják - tulajdonságúak. Itt az összes jelzőcsoportot, ismétlőt, detektort, programot, lámpatípust, mint osztályokat definiálni kell. Nekem is definiálnom kellett a LOGO!-inkat egy új osztályként.

```
public static LogoJelzofej logo;
public static LogoJelzofej logo1;
public static LogoJelzofej logo2;
```

Mindhárom LOGO!-t publikus és statikus osztályként definiáltam LogoJelzofej névvel. A Var osztály az Actros indulásakor egyszer fut le, amikor a deklarálás megtörténik.

6.5. Init.java

Ebben a programrészben kell inicializálni, azaz kezdő értéket adni, minden működéssel kapcsolatos dolognak. Ezek közé tartoznak:

- jelző csoportok (Signal group)
- közbenső idő mátrix (Zwischenzeitmatrix)

- detektorok
- programok
- óra

Mindent a meglévő formájában hagytam, csak a jelző csoport részt egészítettem ki a LOGO!-k inicializálásával. Az előzőekben a változóknál létrehozott LogoJelzofej osztályokat inicializáltam. A LOGO! -khoz, amelyik *logo, logo1, logo2* névvel futnak, hozzárendeltem az IP címeiket. Ettől kezdve, például ha a *logo1*–re hivatkoztam, az az itt megadott IP című PLC-re volt érvényes.

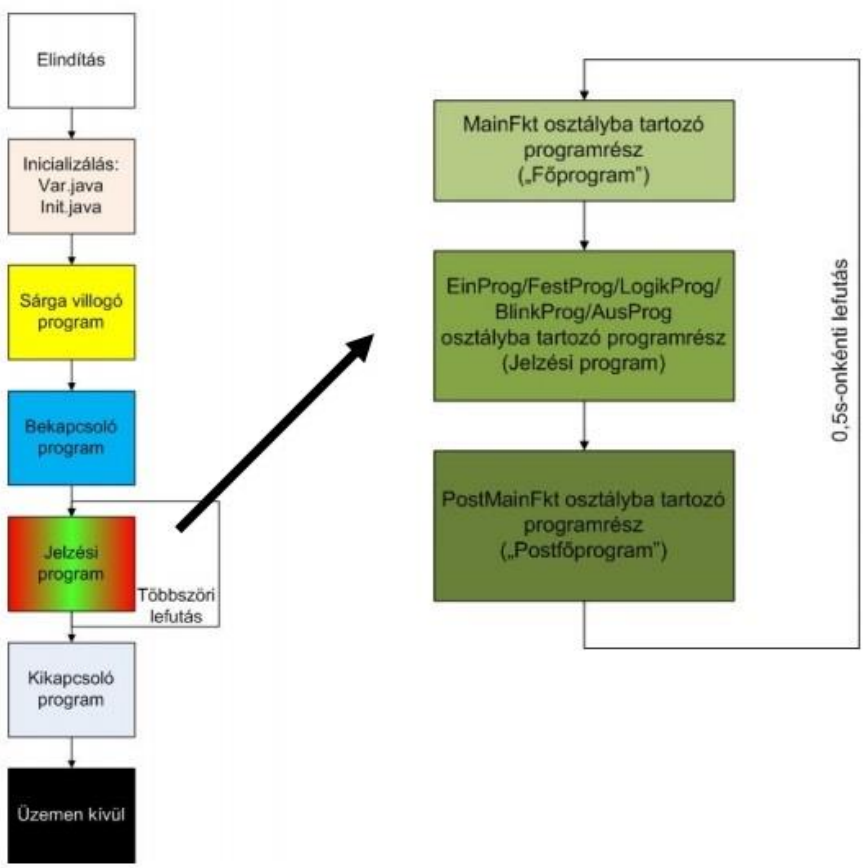
```
Var.logo = new LogoJelzofej("192.168.0.212");
Var.logo1 = new LogoJelzofej("192.168.0.213");
Var.logo2 = new LogoJelzofej("192.168.0.214");
```

6.6. BeProg.java

Feladata a beprogramozás. Nem indulhatnak el a programok azonnal a bekapcsoláskor, így mindig alkalmaznak egy belépő programot. Ez segíti a különböző fázisok megfelelő időben való elindulását, úgymond rávezeti a programokat az elindított idősávra.
A LOGO!-s projektemben ehhez a programrészhez nem kellett nyúlnom, ugyanis én az egyik meglévő fix programba építettem bele a LOGO!-s programozást. Az megfelelő, ha a BeProg elindítja a fix programot.

6.7. AltalanosResz.java

Az Actros elindulása, azaz a változók definiálása, inicializálás és a bekapcsoló program lefutása után belép a program a jelzési programba. A jelzési program az, ami minden fél másodpercben lefut. Ennek része az AltalanosResz.java (a 16. ábrán ez MainFkt néven fut) is. Ebben az osztályban hívtam meg a Var.prog1 –et, amit a FixProg1-re mutat. A működést a 16. ábra szemlélteti. [8]



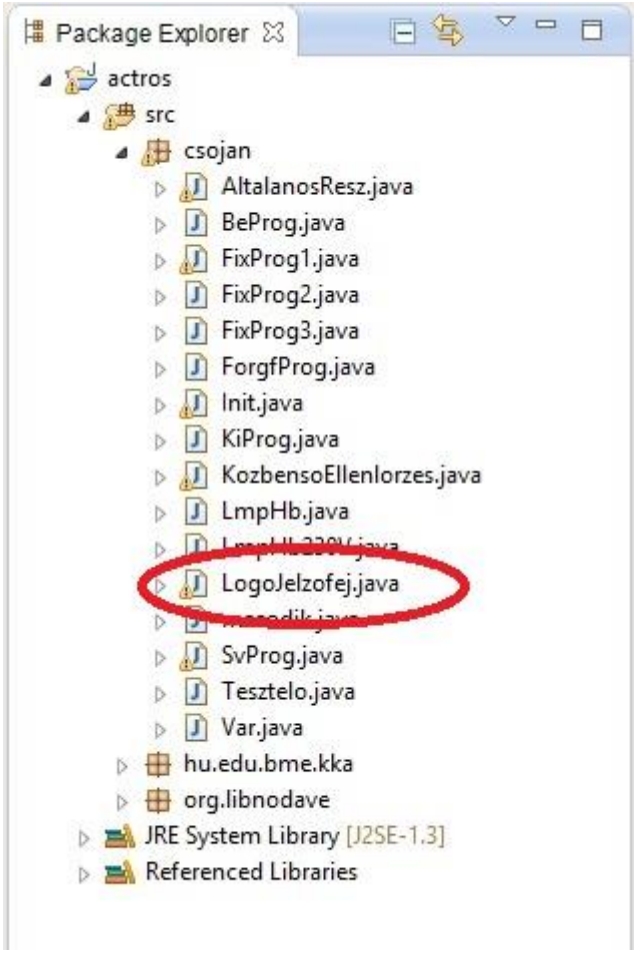
16. ábra: Az Actros működése

(forrás: Polgár János, Tettamanti Tamás: Forgalomtechnikai kód az ACTROS VTC 3000 forgalomirányító berendezésben [elektronikus jegyzet] 3. o.)

6.8. LogoJelzofej.java

Ezt az osztályt én készítettem, hogy a Logo-val kapcsolatos működéseket leprogramozzam. Új osztályt az Eclipseben a File – New – Class menüpont alatt lehet készíteni. A felugró ablakban csak a nevét kell megadni, és a program máris létrehozza az osztályt. Az osztály a ’csojan’ package-be fog tartozni (lásd 17. ábra), azaz a Csömöri út – János utca csomóponthoz. A ’csojan’ ennek a rövidítése.

Az osztály tetején az importált részekben a kivétel kezeléshez kapcsolatos részek, illetve a logo-s szoftverkönyvtárunk van. Ezt mindenképp importálni kell az osztályba, hogy a kapcsolat a PLC-kkel működjön.



17. ábra: Package Explorer

(forrás: Eclipse [program])

Ezek után a `public class LogoJelzofej` utasítással indítottam az osztályt. Elsőként meghívtam a Logo-kat. Ezt lényegében a `logo1 = new Logo (address);` kóddal tettem. Itt a LOGO!-khoz, melyek *`logo, logo1, logo2`* névvel futnak, hozzárendeltem IP cím alapján egy-egy PLC-t.

6.8.1. A `BEKI` függvény

A következőkben egy függvény magyarázata következik, amit ebben az osztályban készítettem el. Ez az úgynevezett `BEKI` függvény, melynek neve a belépő, és kilépő idő rövidítéséből ered.

A `BEKI` a 'csojan' mappában már egy létező függvény volt. Arra szolgált, hogy adott jelzőt vezéreljen. A paraméterében meg lehetett adni a jelző nevét, a zöld kezdetének és végének az idejét. Nagyon praktikus és elegáns függvény, mert nem kódban kellett le-

programozni, hogy mettől meddig tartson a zöld idő, hanem ezt a feladatot egy háttérben futó programkód elvégezte, amit mindig meghívunk. Meghívása így néz ki:

```
K.BEKI(Var.j11, 1, 28);
```

Ez azt jelenti, hogy az 1-es irány első jelzője az első másodperctől a huszonnyolcadik másodpercig legyen zöld.
Egy ilyen függvényt készítettem a LOGO!-hoz. A nevét megtartottam, meghívása teljesen hasonlóan történt, mint az eredeti `BEKI` függvénynek a FixProg.java –ból. A `BEKI` utasítás után a zárójelben lévő (kezdési idő, befejezési idő, logo száma) bármilyen értéket felvehet, mely a ciklusidőn belül van, illetve a harmadik paraméter azt adja vissza, hogy melyik PLC végzi a parancsot.

```
Var.logo1.BEKI(25, 40, 1);
```

Ez azt jelenti, hogy például az logo1-hez tartozó IP című LOGO! a 25. másodperctől a 40. másodpercig legyen zöld. Ezt a függvényt a LogoJelzofej.java-ban készítettem el. Ennek magyarázata következik.
Az egyes lámpa állapotokat definiáltam először. Ezeket külön változókba vettem fel, hogy a kód szebb legyen, és értékül adtam nekik, a megfelelő idő intervallumokat, illetve címeket:

```
private static final int PIROSSARGA_IDO = 2;
private static final int SARGA_IDO = 3;
private static final byte PIROS = 1;
private static final byte SARGA = 2;
private static final byte ZOLD = 4;
private static final byte PIROSSARGA = PIROS|SARGA;
```

Ezek után a kezdő és befejező pillanatokat definiáltam. A getBefejezes() függvény visszatérési értékét a ’bef’ változóba írattam, a getKezdes() függvény visszatérési értékét pedig a ’kezd’ változóba. Ezek a függvények, melyek a ’csojan’ részei, a kezdő és befejező időt adták meg.

```
private int bef;
    public int getBefejezes() {
          return bef;
    }

private int kezd;
    public int getKezdes() {
          return kezd;
    }
```

Majd létrehoztam a `BEKI` függvényt. Public void függvényként definiálva, azaz publikus és nincs visszatérési értéke. Három paramétere a 'kezdes' és 'befejezes' és 'logoszama' változók lettek, melyekből az első kettőt egyenlővé tettem, az előbb létrehozott 'kezd' és 'bef' változókkal. Ezek lesznek a zöld idő kezdő és befejező értéke. A 'logoszama' változó arra szolgált, hogy visszakapjam, hogy épp melyik PLC végezte a feladatot. Ezek után definiáltam a 'sec' változót, mely az aktuális időt adta meg. Ezt egyenlővé tettem a getProgsek() függvénnyel, ami megadja a program aktuális szekundumát. Ezen kívül definiáltam még a ciklusidőt is egy 'ciklusido' változóba ugyan ezzel a metódussal, csak ott a getZyklDauer() függvényt használtam. Ezek a függvények szintén a 'csojan' részei.

```
public void BEKI(int kezdes,int befejezes,final int logoszama) {
      int sec = Var.tk1.getProgSek();
      int ciklusido = Var.tk1.getZyklDauer();
      bef = befejezes;
      kezd = kezdes;
```

A `BEKI` függvény lényegében 'if' állításokból áll. Feltételek felállításával felépítettem a függvényt úgy, hogy teljesen általános legyen. Így bármelyik jelzőnél fel tudtam később használni, hiszen csak a paramétereit kellett megadnom.
A kódsort itt láthatjuk:

```
if (sec == befejezes + SARGA_IDO){
            set(PIROS, logoszama);
            }
if (sec >= kezdes && sec < kezdes + PIROSSARGA_IDO){
            set(PIROSSARGA, logoszama);
            }
if (sec == kezdes + PIROSSARGA_IDO){
            set(ZOLD, logoszama);
            }
if (sec >= befejezes && sec < befejezes + SARGA_IDO){
            set(SARGA, logoszama);
            }
```

A kód magyarázata:

- **Piros szín**

  Ha az aktuális szekundum egyenlő volt a 'befejezes' változó és a sárga idő összegével, akkor kiadtam a LOGO!-nak a piros égést. Ezt úgy értem el, hogy a megfelelő bitre adtam igaz értéket. Ezek egyezését már beállítottam a definiáláskor, illetve a Logo Soft Comfortban. A piros színhez például az 1. bitet kellett

felülírnom (fent definiáltam: `private static final byte ZOLD = 1;`). A `set` egy függvény, mely beállít a LOGO!-nak adott bitjére adott értéket. Erről a függvényről később fogok írni, ezt is én készítettem, abból a célból, hogy a kódolás tisztább és átláthatóbb legyen.

- **Piros-sárga szín**

  Ha az aktuális szekundum nagyobb volt vagy egyenlő, mint a 'kezdes' változó, valamint kisebb, mint a 'kezdes' változó és a piros-sárga idő összege (ezt 2 másodpercre definiáltam: `private static final int PIROSSARGA_IDO = 2;` mert Magyarországon 2 másodpercig ég együtt a piros a sárgával zöldre váltás előtt), akkor kiadtam a LOGO!-nak a piros-sárga együttégést a `set` függvény meghívásával.

- **Zöld szín**

  Ha az aktuális szekundum egyenlő volt a 'kezdes' változó és a piros-sárga idő (2s) összegével, akkor beállítottam a LOGO!-nak a zöld égést.

- **Sárga szín**

  Ha az aktuális szekundum nagyobb vagy egyenlő volt, mint a 'befejezes' változó valamint kisebb, mint a 'befejezes' változó és a sárga idő összege (ezt 3 másodpercre definiáltam: `private static final int SARGA_IDO = 3;` mert Magyarországon 3 másodpercet ég a sárga izzó, pirosra váltás előtt), akkor kiadtam a LOGO!-nak a sárga égést.

6.8.2. A `set` függvény

A `set` függvény szolgált arra, hogy a LOGO!-nak ki lehessen adni egy adott utasítást. A LOGO! VM-ére (Variable Memory) írni az alábbi függvényekkel lehet:

```
Logo.setByte(int pos, int value)
Logo.setWord(int pos, int value)
Logo.setDWord(int pos, int value)
```

A zárójelben két paramétert kell megadni vesszővel elválasztva, az első a pozíció, a második az érték, amit a helyre szeretnénk írni. Lehet bájtot (Byte), szót (Word) és dupla szót (Dword) írni a VM-be. A függvény visszatérési értéke az aktuális Logo példány, ha halmozni szeretnénk az írást.

Problémát jelentett a kommunikáció sebessége. Az Actros úgy működik, hogy minden másodpercen kétszer lefuttatja a kódokat, tehát 0,5 s-ként lefutott a LogoJelzofej.java, ami használta a `set` függvényemet. Ebbe a fél másodperbe bele kellett volna férjen, hogy mind a három PLC-hez eljusson az utasítás és azok végrehajtsák. Az utasítások egymás után lettek kiküldve, amire a rendelkezésre álló idő nem volt elegendő. Így az Actros megállt mikor a programnak el kellett volna indulnia, mert az adott sebesség alatt nem zajlott le mind a három LOGO! kommunikációja. A probléma megoldására a szálakra bontás vezettem be a rendszerbe. A main szálon kívül további két szálat indítottam el, melyek már ellátták a szükséges feladatokat. A `set` függvényt a LogoJelzofej osztályban írtam meg, mert itt használtam a függvényt.

6.9. FixProg1.java

A FixProg1.java a ’csojan’ (Csömöri út – János utca kereszteződés) egy alap fix jelzéstervű programját tartalmazza. A rendszeremnek lényegében mindegy, hogy melyik programban hívtam meg a Logo-s `BEKI` függvényem. Én az első fix programból, azaz a ’FixProg1’-ből tettem ezt meg. A ’csojan’-nak még van ’FixProg2’, ’FixProg3’ fix jelzéstervű programjai és egy ’ForgfProg’ nevű forgalomfüggő programja. Az AltalanosResz.java –ban azt állítottam be, hogy a ’FixProg1.java’ program induljon el. Ezek után már csak meg kellett hívnom a FixProg1.java –ban az előzőekben a LogoJelzofej.java –ban megírt `BEKI` függvényem, amit a következő kódsorral tettem:

```
Var.logo.BEKI(0, 20, 1);
Var.logo1.BEKI(25, 40, 2);
Var.logo2.BEKI(45, 55, 3);.
```

A `BEKI`-t publikus függvényként definiáltam, így tudtam másik osztályban is használni. A Var.logo.BEKI (x, y, z) meghívta a függvényemet. A ’FixProg1.java’ is minden fél másodpercben lefut, így 0,5 másodpercenként a program átugrott a LogoJelzofej.java –ban megírt kódrészre és lefutott az ott megírt logikai feltétel. A zöldidő hosszát teljesen egyszerűen az ’x’ és ’y’ helyére behelyettesített értékek közötti idő adta.

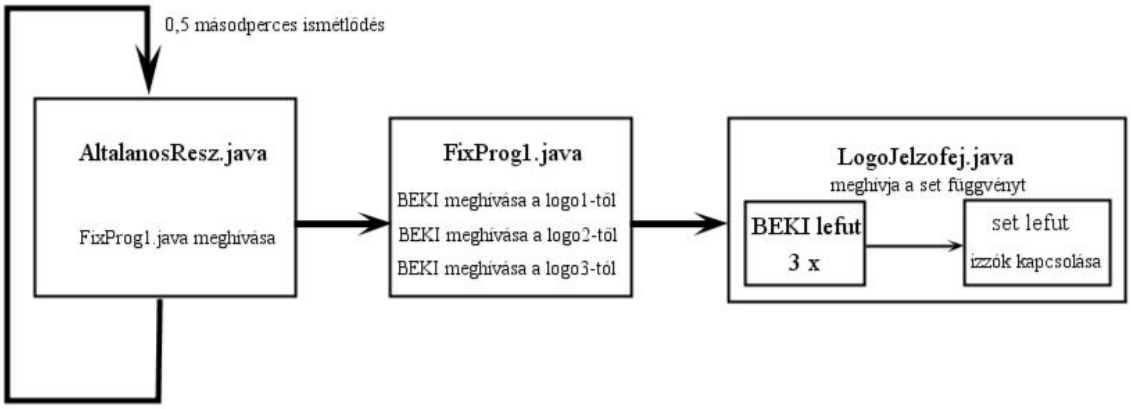
6.10. A rendszer működése

Ebben a fejezetben a rendszer működésének összefoglalója következik. A működéshez először minden eszközt feszültség alá kellett helyezni. Bekapcsoltam a labor routerét, hogy legyen hálózat, majd a 14. ábrán látható biztosítékokat felkapcsoltam. Ezzel elin-

dultak a PLC-k, áram alá került az Elba jelzőfej, továbbá elindult a transzformátor, ami áram alá helyezte a Siemens jelzőfejet. A harmadik PLC (a világító diódás általam készített jelzőt irányító) elindításához a tápegységét bedugtam a hálózatba. A PLC-ket az Actrossal és hálózattal összekötő hub-ot is feszültség alá helyeztem. Majd elindítottam az Actrost is. Hálózaton keresztül feltöltöttem rá a 'vt.jar' fájlt, ami az Eclipsből kiexportált 'src' mappa fájljait tartalmazta (lásd 17. ábra). Ezek után az Actrost újra kellett indítanom, hogy a friss fájllal induljon el. A kódok hibamentessége esetén az Actros el tudott indul és elkezdte kiküldeni a LOGO!-knak az utasításokat, melyek vezérelni kezdték a jelzők izzóit.

Az AltalanosResz.java minden fél másodpercben lefut. Ez meghívja a FixProg1.java-t. A FixProg1.java-ban lévő kódsor az IP cím alapján azonosítva, PLC-nként sorban meghívja a LogoJelzofej.java `BEKI` függvényét, mely eldönti az időpillanat alapján, hogy melyik jelzésképet kell használni. Ezután szintén PLC-nként sorban, kiadja azt a `set` függvénynek, mely a main szálon kívül elindít egy új szálat és elküldi a LOGO!-nak az utasítást. Így fél másodperc alatt három új szál indul el, mert három logikai modulunk van. Ekkor fizikailag megtörténik az lámpa izzójának kapcsolása. Ezek a szálak a feladatuk elvégzésével leállnak és megszűnnek. A következő fél másodpercben ugyanez a műveletsor játszódik le.

A folyamatot a 18. ábrán követhetjük le.



18. ábra: A működés folyamatábrája

Eközben feszültség ellenőrzés céljából további feladatokat végeztetek a LogoJelzofej.java –ban. Erről a 7. fejezetben írok részletesen.

7. A rendszer biztonsága

Napjainkban a biztonságot úgy definiáljuk, hogy egy olyan zavartalan állapot, amely veszélyektől, vagy bántódástól mentes [9]. Közös cél, hogy a közúti közlekedésben is elérjünk egy ilyen biztonságos állapotot, melyhez elengedhetetlen, hogy a forgalomirányító berendezések hibátlanul végezzék a feladatukat. A projektemmel egy olyan új rendszert hoztam létre, mely egy meglévő átalakításával jött létre. Az átalakítás során a biztonsággal kapcsolatos feladatok ugyan megmaradtak, de ezeknek a feladatoknak a megoldását újra kellett terveznem. Egy forgalomirányító berendezés biztonsági funkciói a következők kell, hogy legyenek:

- a tiltott (ellentétes) jelzésképek felismerése, megjelenésük esetén a berendezés sárgavillogó üzemmódba való kapcsolása,
- belső hiba esetén hibabiztos állapotba kerüljön (hibától függően sárgavillogó vagy sötét üzem),
- zöld együttégés tiltása – közbenső idők meglétének ellenőrzése
- és a fénypontokon lévő izzók kiégésének ellenőrzése.

Ezen ellenőrzések és funkciók hiányában, súlyos balesetek következhetnek be, ezért meglétük nagyon fontos. A hibák előfordulását nehéz kiküszöbölni. Míg például a közbenső időből eredő hiba tervezési probléma következménye lehet, addig egy izzó kiégés főleg műszaki probléma eredménye. Célom, hogy a rendszerem bármelyik eset bekövetkezését azonnal érzékelje, és az irányítást egy biztonságos állapotba vezérelje.
Az ellentétes jelzéskép kiadása nem fordulhat elő, hogyha a kódban, ahol a jelzőfejek izzóit vezéreljük nincs hiba. A belső hiba esetén a PLC-k sárga villogó funkcióba kapcsolnak, erről a 7.1 fejezetben írok. Zöld együttégés tiltását a közbenső idők meglétével garantáljuk. Ezek ellenőrzését a 7.2 fejezet mutatja be. Az izzók kiégése feszültség és áram ellenőrzéssel vizsgálható. Erről a 7.3 fejezetben lesz szó. Először bemutatom, hogy milyen módon működik a rendszeremben a sárga villogó funkció, majd az egyes hibaforrások felismerésére és kezelésére kialakított metódusokat ismertetem.

7.1. Sárga villogó üzemmód

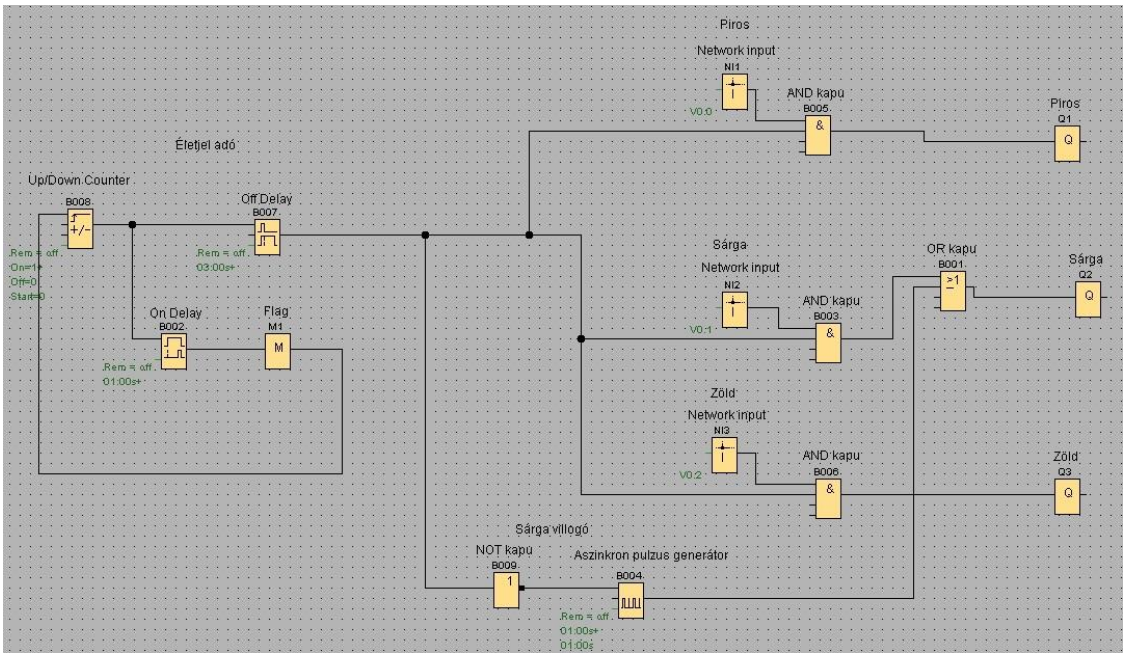
A közúti közlekedés forgalomirányításában, a sárgán villogó jelző a rendszer üzemen kívüli állapotát jelenti. Ez lehet a meghibásodás, de olykor a jelzőlámpás irányítás szükségtelensége (például éjszaka lekapcsolt lámpák) miatt is. Ilyen esetben a jelzőfejek

mellett elhelyezett közúti jelzőtáblák lépnek érvénybe, melyek általában a stop tábla és az elsőbbség adást kötelező tábla. Egy rendszerben nagyon fontos, hogy a felhasználó mindig értesüljön a hibáról. A sárga szín villogása figyelemfelkeltő. Továbbá teljesen elszeparálható a többi jelzésképtől, így a közlekedésben résztvevő azonnal tudja, hogy kereszteződésben fokozott figyelemre van szükség.
A jelzőket sárga villogó üzemmódba (az éjszakai szándékos sárgavillogó üzemet kivéve) zavar esetén kell vezérelni, ami fizikai oldalról lehet például izzó kiégés, vezeték szakadás és bármilyen irányítási hiba fellépése is, mint például zöld együttégés, vagy a forgalomirányító berendezés belső hibája.
A kidolgozott rendszerben az egyik legfontosabb kapcsolat a LOGO!-k és az Actros között áll fenn. Ha ez a kapcsolat megsérül, akkor elveszíthetjük az irányítást a jelzőfejek fölött, így ennek biztonsága kulcsfontosságú. Ezért is választottam egy olyan megoldást, ami teljes mértékben garantálja, a kapcsolat meghibásodásával is biztonságos a kimenetet. Sárga villogó állapotot idézünk elő, ha a PLC és az Actros között nincs kapcsolat, így nem történhet meg olyan, hogy egy kiadott jelzéskép a jelzőkön marad, mert megszakadt a kapcsolat és a következő utasítás már nem ért oda. Ha egy jelzéskép ki van adva, de valami miatt a kapcsolat megszakad, akkor a LOGO! azonnal sárga villogó üzembe vezérel. Ezt úgy tudtam megvalósítani, hogy a LOGO!-ra feltöltött programon módosítottam, azaz lényegében az Actros ki van hagyva ebből a biztonsági funkcióból.

7.1.1. A sárga villogó program tervezése Logo Soft Comfortban

A sárga villogó funkció kialakítása a következő módon történt. Az Actros és a PLC közötti kapcsolat meglétét vizsgáltam. Ha a kapcsolat fenn áll, akkor az Actros által kiadott jelzésképek futhatnak. Ha nincs kapcsolat, akkor a PLC átveszi az uralmat és a rajta futó program sárga villogóra állítja a jelzőfejeket.
A kapcsolat meglétét folyamatosan vizsgálnom kellett. Mivel az Actros fél másodpercenként lefuttatja a kódjait, ezért ezt ki tudtam használni. Egy jelet fogok vizsgálni, amit az Actrossal küldetek el, hogy megérkezik-e. Ennek menete az, hogy az Actrosban fél másodpercenként lefutó ’FixProg1’ program mindig kiküldte ezt a vezérlőjelet a PLC felé, ami ezt mindig lenullázta. A kapcsolatot vizsgáló programban van egy ellenőrző visszacsatolás, az Actros által küldött jel nullázásának utolsó idejéről. Ha ez meghaladt egy bizonyos határértéket, amit én határoztam meg a biztonság függvényében - azaz például 2 másodperc óta nem érkezett jel az Actros felől - akkor kiadta a PLC-nek a sárga villogó üzemmódba való kapcsolás utasítását. Mindehhez át kellett alakítani az

eddig meglévő LOGO!-ra feltöltött programomat, amit a 6. fejezetben ismertettem. Az újratervezett program blokk diagramja a 19. ábrán látható.



19. ábra: A sárga villogással ellátott blokkdiagram

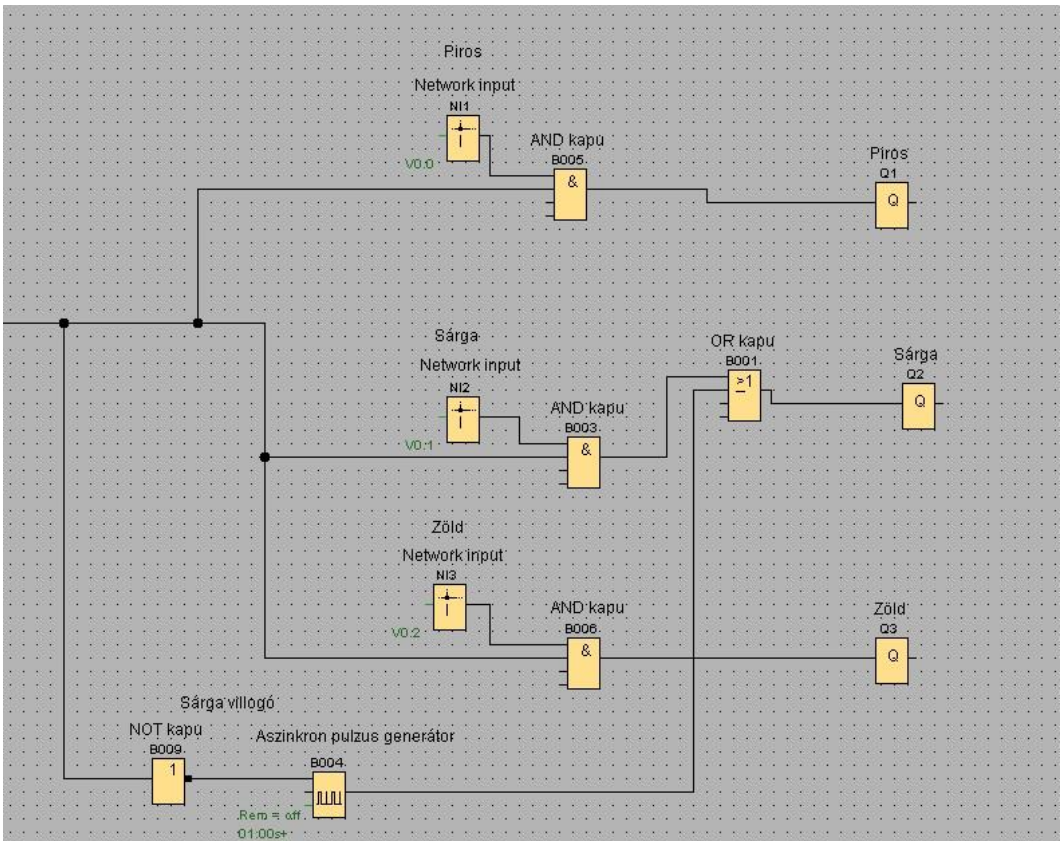
(forrás: Logo Soft Comfort V.7. [program])

A következőkben a program egyes részleteinek magyarázata következik, a kimenetek oldaláról kezdve. Elsőként a 20. ábrán látható részletről írok.

- Ha érkezett ellenőrző vezérlőjel az Actros felől ÉS kiadtam a piros jelzést (Network Inputon keresztül az Actrosból), akkor kiengedem a jelet a Q1 kimenetre (ami fizikailag a piros izzónak felel meg).
- Ha érkezett a jel az Actros felől ÉS kiadtam a sárga jelzést (Network Inputon keresztül az Actrosból) VAGY, ha sárga villogó utasítás érkezett, akkor továbbengedem a jeleket a Q2 kimenetre (ami fizikailag a sárga izzónak felel meg).
- Ha érkezett jel az Actros felől ÉS kiadtam a zöld jelzést (Network Inputon keresztül az Actrosból), akkor kiengedem a jelet a Q3 kimenetre (ami fizikailag a zöld izzónak felel meg).

A sárga villogó funkciónál, magáért a villogásért egy aszinkron pulzus generátor (Asynchronous Pulse Generator néven találjuk meg a Logo Soft Comfortban) felelt. Ezt a 20. ábra alján láthatjuk. A pulzus generátor a tulajdonságai között meghatározott időközönként megszakítja a jelünket, azaz pulzálást idéz elő. Ha a blokkon jobb egér

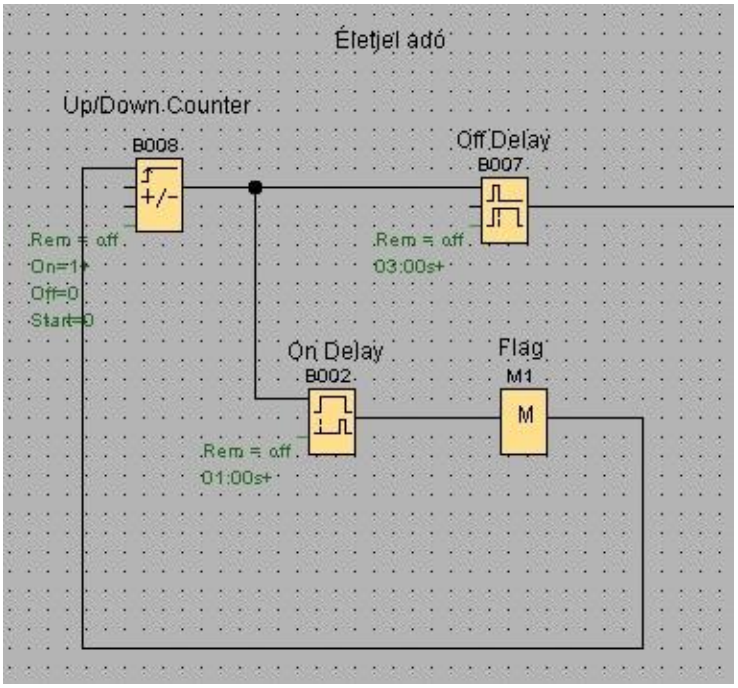
gombbal kattintásával, elérhető a Block Propertiest, ahol be lehet állítani az impulzus szélességét. Ezt 1 másodpercre állítottam.



20. ábra: Részlet a blokkdiagramból I.

(forrás: Logo Soft Comfort V.7. [program])

A 19. ábra bal oldalán elhelyezkedő rész felel az Actros felőli kommunikációért. Ezt kinagyítva a 21. ábrán láthatjuk. Életjelet kellett generálnom az Actrosból, majd azt a PLC-n futó programmal ellenőriznem. Ehhez a Logo Soft Comfort VM Mapping funkcióját tudtam kihasználni. A magyarázatom értelmezéséhez a következőkben a Parameter VM Mapping-ról írok röviden.

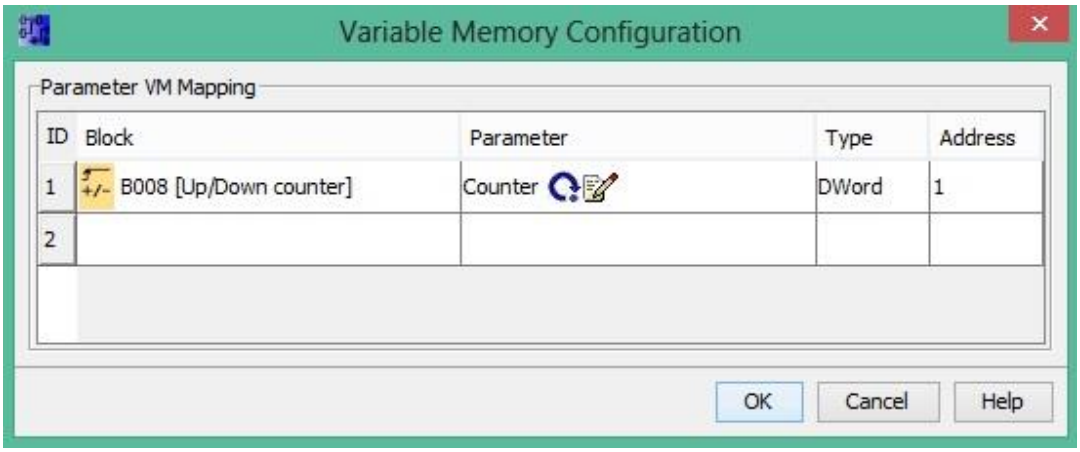


21. ábra: Részlet a blokkdiagramból II.

(forrás: Logo Soft Comfort V.7. [program])

Parameter VM Mapping

A VM (Variable Memory - lásd. 2) egyes byte-jaihoz a PLC –n futó program elemeinek tulajdonságait lehet kötni. Ezáltal ezeket a helyeket a VM-ben a PLC minden frissítési ciklusban feltölti az aktuális értékekkel. A Logo Soft Comfortban a Tools menü alatt található „Parameter VM Mapping" ablakban lehet beállítani, hogy melyik blokk melyik tulajdonságát a VM-en belül hova írja ciklusonként a LOGO! (lásd 22. ábra). [6] Én az Up/Down Counter (Fel/Le Számláló) blokk VM-én állítottam.



22. ábra: A VM konfigurációja

(forrás: Logo Soft Comfort V.7. [program])

A Block és a Parameter oszlopokban lehet kiválasztani a kívánt függvényblokkot és annak kívánt paraméterét. A típusa adott a Type oszlopban és az Address alatt beállítható a VM-en belüli címe [6]. A megvalósítás során Dword típust alkalmaztam - mert a program csak ezt kínálja fel - mely 4 byte-ot foglal el, a címének pedig 1-et adtam meg. Ez lényegében bármi lehet, csak figyelni kell, hogy az Eclipsből majd a beállított címre küldjem jelet.

Térjünk vissza a 21. ábrára. Az Up/Down Counteren keresztül fog érkezni a jel az Actros felől, amit az Eclipseben adtam ki utasításként. Mögötte elhelyeztem egy Off Delayt, ami úgy működik, hogy ha ráengedem a jelet, az átmegy rajta. Ha megszűnik a jel az inputján, az outputon még marad jel (lényegében húzva marad), elindul benne egy visszaszámláló (aminek az értékét a blokk tulajdonságaiban lehet beállítani). Ha lejár a visszaszámláló, akkor megszűnik a jel az outputon is. Az Off Delay-re 3 másodperces visszaszámlálást állítottam be. Így az Actrosból fél másodpercenként érkező jel folyton elindítja, de visszaszámlálás csak fél másodpercig jut el a háromból, mert érkezik a következő jel (feltéve, ha a kapcsolat még fenn áll, de pont ezt vizsgálom).

A Counter mögé közvetlenül beraktam egy visszacsatolást, melyre egy On Delayt kötöttem. Ez a visszacsatolás fogja belökni az Off Delayt a sárga villogó funkcióba, hogyha nem érkezik több jel az Actros felől. Az On Delay úgy működik, hogy mikor rámegy a jel, akkor elindul benne egy visszaszámláló. Ha letelik a beállított idő, akkor a blokk tovább engedi a jelet. 2 másodpercre állítottam az On Delay blokk visszaszámlálási idejét. Így a fél másodpercenként érkező Actros jel mindig újraindítja benne a visszaszámlálást. Amíg az Actrossal van kapcsolat, addig a visszacsatoláson nem megy jel, mert az On Delay nem engedi át. Ha megszűnik a jel az Actros felől és letelik az On-Delay-ben a visszaszámlálás, akkor továbbengedi az utolsó hozzá érkezett jelet, ami eljut Up/Down Counter-re és lekapcsolja. A megszűnő jel lekapcsolja az Off-Delayt is, és elindítja a sárga villogó funkciót.

A VM írására az alábbi függvényekkel van lehetőség:

- Logo setByte(intpos, intvalue)
- Logo setWord(intpos, intvalue)
- Logo setDWord(intpos, intvalue) [6]

Lehet byte-ot, szót (Word) és dupla szót (Dword) írni a VM-be. Én az alábbi paranccsal írtam a VM-re:

```
logo_i.setDWord(1, 10);
```

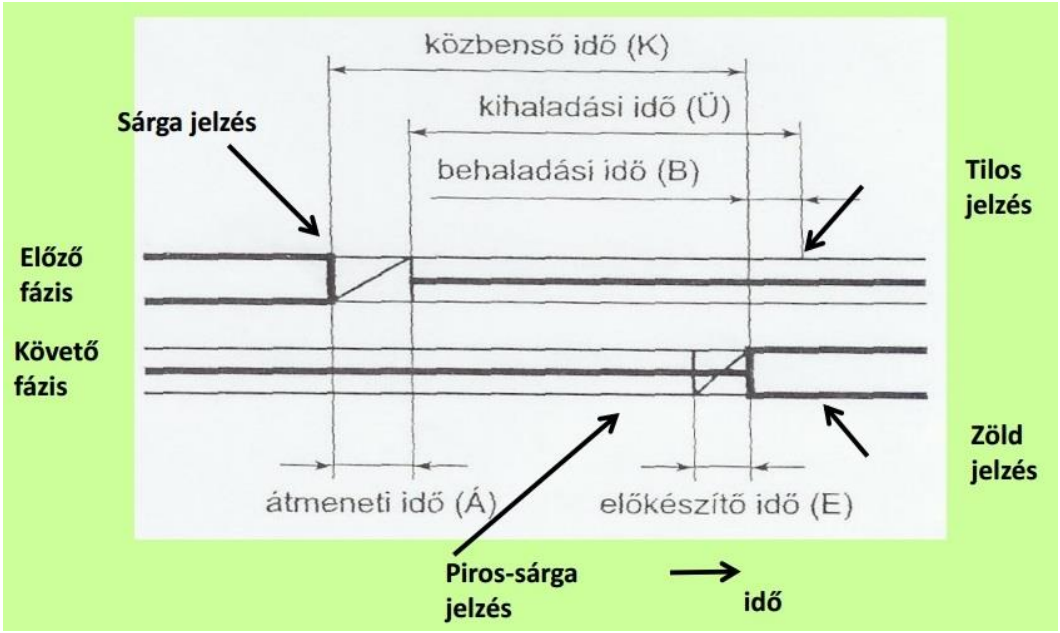
Itt látható, hogy a Logo Softban a VM Configuration-ban beállított első címre írtam 10-es értéket. Az Eclipsben egy olyan helyre kellett raknom a vezérjelet generáló kódot, ami minden fél másodpercben lefut. A `BEKI` függvényem ilyen, mely a LogoJelzofej.java-ban van megírva. A `BEKI`-ben indítottam egy új szálat a main szál és a `set` függvényben indított szál mellé. Ez a szál minden fél másodpercben elküldi a `setDWord(1, 10);` paranccsal a vezérlőjelet a LOGO!-knak.

7.2. Közbenső idők meglétének ellenőrzése

A csomóponton áthaladó forgalmi áramlatok forgalombiztonsági okból közvetlenül nem követhetik egymást. Az egymást keresztező, vagy nyomvonal szempontjából kapcsolódó (fonódó) mozgások szabad jelzései között biztosítandó legkisebb időt közbenső időnek nevezzük. A közbenső idő három tényezőből áll, mely:

- átmeneti – sárga jelzés – idő
- kihaladási – ürítési – idő
- behaladási idő [1]

Értékét úgy számítjuk, hogy az átmeneti időhöz hozzáadjuk a kihaladási időt, majd kivonjuk a behaladási időt (lásd 23. ábra).



23. ábra: A közbenső idő részei

(forrás: Debreczeni Gábor: Közúti Forgalomtechnika [elektronikus jegyzet] 59.o.)

Az általam felállított rendszer minden fél másodpercben ellenőrzi a közbenső idők meglétét. Ha ebben hiba van, akkor a rendszert sárga villogó üzemmódba vezérli. A LogoJelzofej.java-ban írtam egy kódsort, amely a `BEKI` függvényünkben megadott belépési és kilépési időből számítja a közbenső időt. Ezeket manuálisan számítottam, így ha további LOGO! kapcsolnék a rendszerbe, ezt a programrészt ki kellene egészíteni. A rendszeremben a kilépési és a belépési idő különbségeként meghatározható a közbenső idő (ezeket az időket függvénnyel kértem le). Ez nem minden esetben eredményezett pozitív számot, ezért vettem a művelet utáni érték abszolút értékét. Ezt minden variációra elvégeztem és külön változókba mentettem őket. 3 PLC esetén 6 variáció, 6 változó.

```
kozb01 = Math.abs((Var.logo.getKezdes()-
          Var.logo1.getBefejezes()));
kozb02 = Math.abs((Var.logo.getKezdes()-
          Var.logo2.getBefejezes()));
kozb10 = Math.abs((Var.logo1.getKezdes()-
          Var.logo.getBefejezes()));
kozb12 = Math.abs((Var.logo1.getKezdes()-
          Var.logo2.getBefejezes()));
kozb20 = Math.abs((Var.logo2.getKezdes()-
          Var.logo.getBefejezes()));
kozb21 = Math.abs((Var.logo2.getKezdes()-
          Var.logo1.getBefejezes()));
```

Ezek után minden közbenső idő megvan egy változóban, melyeket az előre megadott (amit a csomópont tervezője által előírt) közbenső idő értékekhez hasonlítottam relációval. Ha ezek közül egy is rossz volt, azaz kisebb az általam számított közbenső idő, mint a megadott, akkor a rendszert sárga villog üzemmódba kapcsoltam. Ennek metódusa során az Actrosnak a LOGO!-kkal való kapcsolat bontása utasítást adtam.

```
logo1.closeConnection();
```

A kapcsolat bontása sárga villogó üzemmódot eredményez, mert nem küld az Actros vezérlőjelet a PLC-nek. Ezt már a 7.1 fejezetben részleteztem.

7.3. Feszültség ellenőrzése

A feszültséget a zöld izzók esetében kell ellenőrizni. A LOGO! 12/24 RCE a bemenetein (inputok) tud feszültséget ellenőrizni, mégpedig úgy, hogy az inputon lévő feszültséget a földponthoz hasonlítja. A specifikáció szerint, ha az inputon megjelenő feszültség

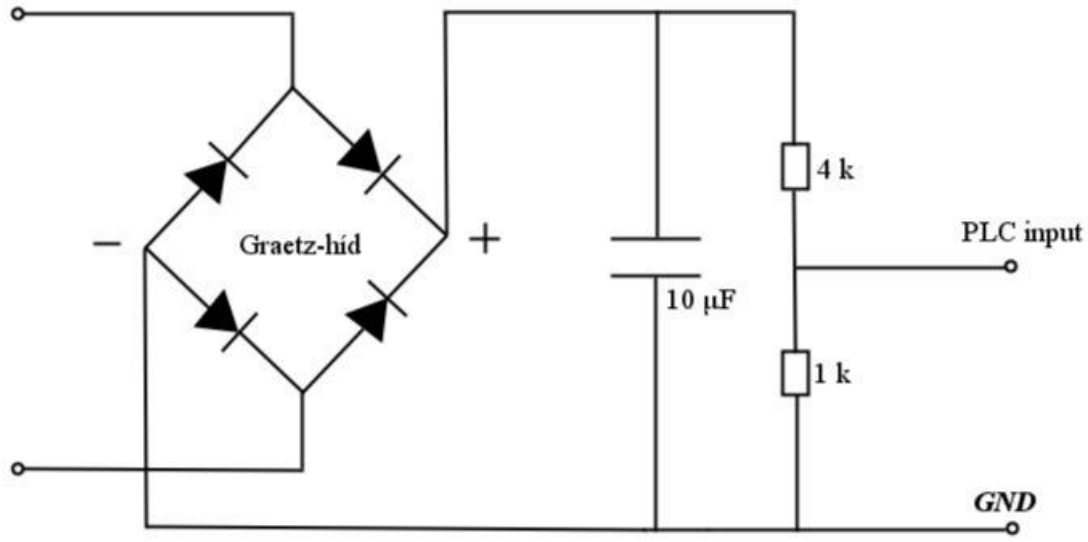
kisebb, mint 5 V, akkor a válasza ’0’ lesz, ha az inputon megjelenő feszültség minimum 8,5 V, akkor a válasza ’1’ lesz. Így a PLC-től kapott válasz boolean típusú, az az a válasz ’true’ ha a zöld izzó ég és ’false’ ha a zöld izzó nem ég.

A feszültség lekérdezését a programomban a `BEKI` függvénybe írtam bele. Több függvény is rendelkezésre áll a Logo olvasásához, én a ’boolean getInput (int i)’ függvényt használtam. Az inputok sorszámozása 0-tól indul és mivel a zöld izzót akartam vizsgálni, ezért a 2. inputra kellett hivatkoznom, illetve ide kellett visszakötnöm a zöld izzót. Ahhoz, hogy az ellenőrzést nyomon tudjam követni kiírattam a parancsot a képernyőre a következő kóddal:

```
System.out.println(logoszama + " PLC " + "zoldeges " + logo_i.getInput(2)+" " + Var.tk1.getProgZeit());.
```

A ’logoszama’ változó megmondja, hogy melyik PLC-ről való ellenőrzését írattam ki. Ezt a lekérdezést szintén az előbb `BEKI`-ben elindított újabb szálba tettem bele. Erre azért van szükség, mert itt is a LOGO!-tól kértem egy választ, ami a 3 PLC esetén nem férne bele a fél másodperces futási időbe.

A jelzőfejek zöld izzóinak visszakötésénél ügyelnem kellett arra, hogy az izzókba 40 V-os illetve 230 V-os váltakozó áram megy ki. Ha ezt visszakötöm a PLC-be a váltakozó áramot fogja mérni. Így előfordulhat, hogy feszültség esetén is hamis választ kapunk, mert a szinusz görbe egy rossz pontjára mért rá a PLC. Emiatt egyenirányítanom kellett a váltakozó feszültséget. Ezt egy Graetz-híd és hozzá kapcsolódó kondenzátor és ellenállások segítségével tettem meg. A kapcsolási rajzot a 24. ábra szemlélteti.

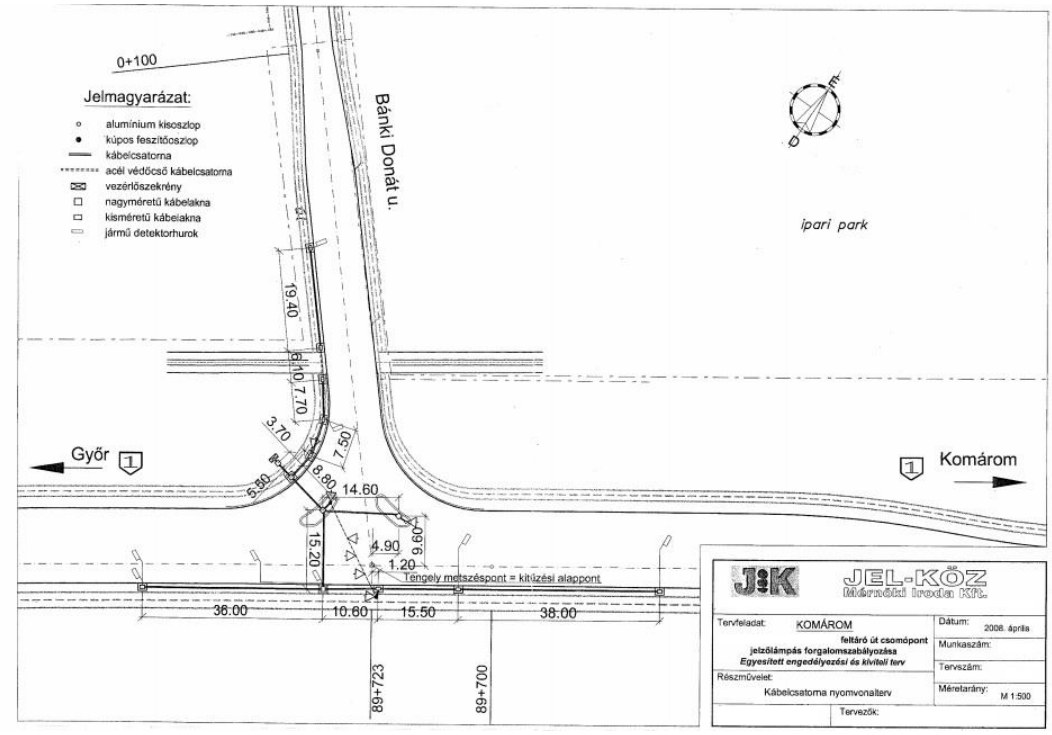


24. ábra: Egyenirányító kapcsolási rajza

8. A fejlesztett rendszer értékelése

Ahhoz, hogy egy új rendszer hasznosságáról, illetve hatékonyságáról megbizonyosodjunk, össze kell hasonlítani egy már létező rendszerrel. Az én általam fejlesztett rendszert a napjainkban működő, általános forgalomirányító rendszerrel tudjuk legkönnyebben összevetni. A két rendszer előnyeit és hátrányait vizsgáltam meg. Az intelligens jelzőfej funkcionalitásban nem lépett előrébb a korábbi rendszernél, hiszen nem tud többet, mint az elődje. Ellenben gazdasági előnyei vannak. Legfőképpen a telepítési költsége, de emellett a károkból eredő javítási költsége is kevesebb.

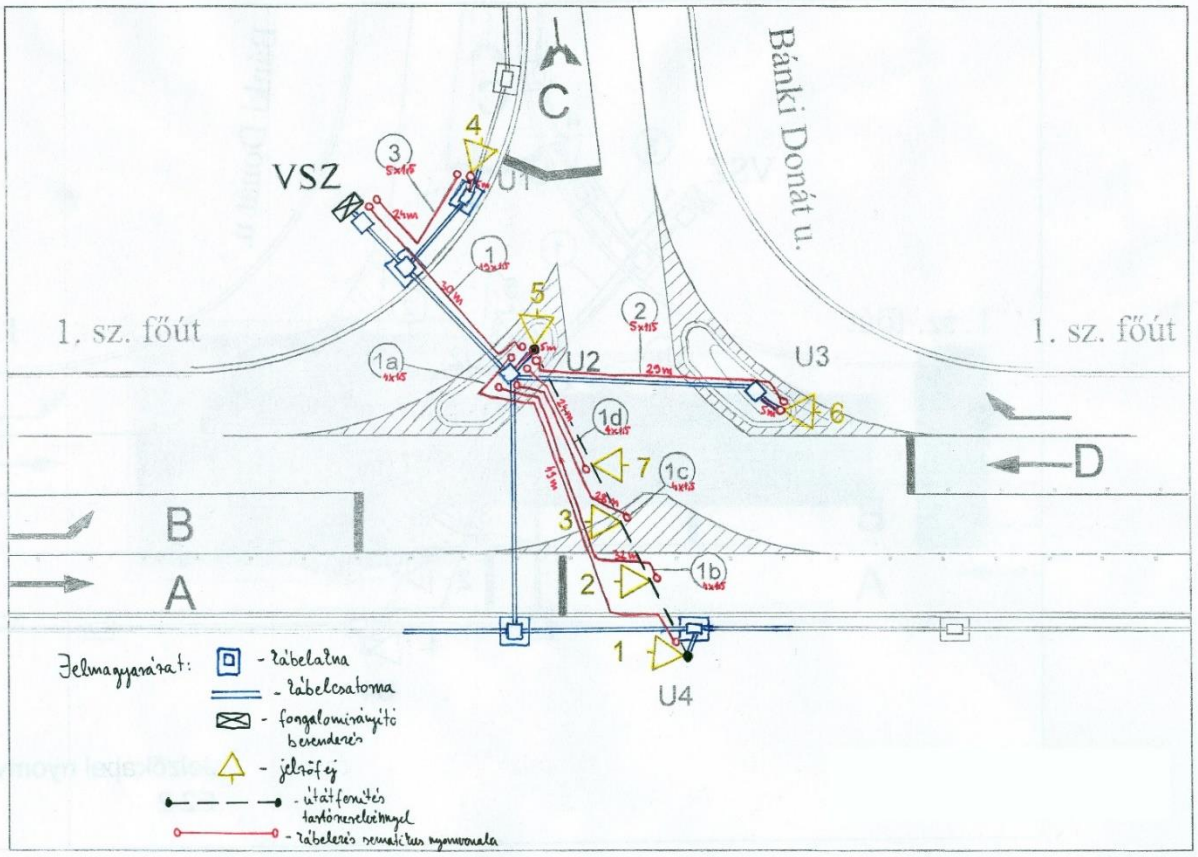
Összehasonlításként egy T-alakú csomópont kialakításának költségét vizsgáltam meg a jelenlegi rendszerre, aztán az intelligens jelzőfejjel működő rendszer esetére. A csomópont egy valóságos kereszteződé, mely Komárom mellett található. A csomópont kialakítását és forgalomirányító eszköz szükségletét a Swarco Traffic Hungary Kft. tervezte még 2008-ban. Ennek teveit és költségvetését elkértem a cégtől, hogy egy olyan konkrét kereszteződést tudjak az összehasonlítás alapjává tenni, ahol ma is az általános forgalomirányítás zajlik. [10] A csomópont a 25. ábrán látható.



25. ábra: T alakú csomópont tervrajza

(forrás: Bodó Lajos: A jelzőlámpás irányítás villamos munkáinak terve [elektronikus dokumentum] 8. o.)

Az építési költségek vizsgálatánál csak azokat a pontokat szedtem össze, amiben a két rendszer eltér. Összegyűjtöttem a tervek alapján a kiépítéshez szükséges kábelek típusait és mennyiségüket. Ebben a rendelkezésemre álló nyomvonalterv segített, melyet a 26. ábra szemléltet. Az 26. ábrán látható egy VSZ felirat. Ez a forgalomirányító berendezés helye. Az elektromos művek ide ad tápellátást. Innen kell eljuttatni az áramot a jelzőfejekhez. A kábeleket nem lehet bárhol vezetni és bárhogyan. Erre kábelcsatornákat kell készíteni a föld alatt. Előfordulhat, hogy az utat többször is át kell vágni, hogy ezek a csatornák a megfelelő helyen húzódjanak. A csatornákba vonóvezetékkel húzzák be a kábeleket. A csatornák kábelaknákba vezetnek bele. Ezekben az aknákban végzik a kábelek behúzását, összekapcsolását, le- és szétágaztatását. Minden olyan helyen, ahol a kábelek elágaznak, egymásba futnak, egy-egy ilyen aknát kell építeni. A 26. ábrán ezeket az aknákat kék színű téglalap jelzi, a kábelcsatornákat meg szintén kék színű két párhuzamos vonal. Ahol kábelcsatornára nincs lehetőség, ott átfeszítést alkalmaznak. Átfeszítés kell az út fölé belógatott jelzőfejeknek. Ehhez külön tartószerelvényt kell építeni. A kábeleket a sodronykötélen vezetik el a jelzőfejekig. Ebben az esetben különleges UV álló kábeleket alkalmaznak. A 26. ábrán az útátfeszítést fekete szaggatott vonal jelzi. Az 1. sz. főút fölé kellett jelzőfejeket belógatni. A 26. ábrán számmal ellátott sárga háromszög jelzi a jelzőfejeket. A kábelezést piros vonal jelöli, a vonalon lévő szám a hosszúságukat. Ezek a vonalak nem ott helyezkednek el, ahol a valóságban futnak a kábelek – csatornában vannak -, de a jobb átláthatóság érdekében ezeket máshol ábrázolták.



26. ábra: Jelzőkábelek nyomvonalrajza

(forrás: Bodó Lajos: A jelzőlámpás irányítás villamos munkáinak terve [elektronikus dokumentum] 9. o.)

A Swarco-tól kapott műszaki dokumentáció tartalmazza a kábelek pontos típusát és hosszát. Ez az 1. táblázatban látható. Továbbá kaptam a cégtől egy költségvetés kivonatot is, mely tartalmazza a kábelek árait. Ezek itt láthatóak:

- SZRMKVM – J 4x1.5 mm^2 – 323 Ft (nettó ár)
- SZRMKVM – J 5x1.5 mm^2 – 385 Ft (nettó ár)
- SZRMKVM – J 19x1.5 mm^2 – 1056 Ft (nettó ár)

A J 4x1.5 mm^2 például azt jelenti, hogy ez egy 4 erű kábel. A csomópontban 4, 5 és 19 erű kábeleket használtak fel. Nagyon sok erű kábelt általában akkor használnak, ha egy távolabbi pontra viszik a kábeleket. A VSZ és az U2 pont között vezet 19 erű kábel, mert az U2-től sok felé ágazik tovább. (lásd 26. ábra)
A kábelezés költsége ezek után kiszámolható.

$$C = 148\ m * 323\ Ft + 53\ m * 385\ Ft + 30\ m * 1056\ Ft = 47804\ Ft + 20405\ Ft + 31680\ Ft = 99889\ Ft\ (nettó\ ár)$$

Kijelenthető, hogy az általános forgalomirányítás kábelezésének költsége ennél a T-alakú csomópontnál körülbelül nettó százezer forint.

Honnan	Hová	Kábel szám	Kábel típus		Hossz	
VSZ	U2 oszlop	1	SZRMKVM-J	19 x1,5 mm2	30	m
U2 oszlop	1 fej (átfeszítésen át)	1a	SZRMKVM-J	4x1.5 mm2	49	m
U2 oszlop	2 fej	1b	SZRMKVM-J	4x1.5 mm2	32	m
U2 oszlop	3 fej	1c	SZRMKVM-J	4x1.5 mm2	28	m
U2 oszlop	7 fej	1d	SZRMKVM-J	4x1.5 mm2	24	m
U2 oszlop	5 fej	1e	SZRMKVM-J	4x1.5 mm2	5	m
U2 oszlop	U3 oszlop	2	SZRMKVM-J	5x1.5 mm2	29	m
U3 oszlop	6 fej	2a	SZRMKVM-J	4x1.5 mm2	5	m
VSZ	U1 oszlop	3	SZRMKVM-J	5x1.5 mm2	24	m
U1 oszlop	4 fej	3a	SZRMKVM-J	4x1.5 mm2	5	m
	földháló		alu sodrony	50 mm2	90	m
			Összesítés:			
			SZRMKVM-J	4x1.5 mm2	148	m
			SZRMKVM-J	5x1.5 mm2	53	m
			SZRMKVM-J	19x1.5 mm2	30	m
			Alu. Sodrony	50 mm2	90	m

1. táblázat: Kábelek típusai és mennyiségük

(forrás: Bodó Lajos: A jelzőlámpás irányítás villamos munkáinak terve [elektronikus dokumentum] 12. o.)

Ezek után kiszámoltam, hogy mennyi lenne ennek a csomópontnak kábelezési költsége, ha intelligens jelzőfejet használnak. A kábelek legnagyobb részétől meg lehet válni. A jelzőfejek között ki kell alakítani egy gerinchálózatot egy SZRMKVM-J 4x1.5 mm^2 –es kábellel. Ez látja el a logikai modulok tápegységeit árammal. Ezen kívül már csak UTP kábelre van szükség a jelzőfejek és a forgalomirányító berendezés közé.

Az intelligens jelzőfejjel irányított rendszer kábelszükségletei az egyes pontok között sorban (értelmezésben a 26. ábra segít):

Honnan	Hova	Kábel típus	Hossz [m]
VSZ	U2	UTP	30
U2	U4	UTP	36
U2	U3	UTP	29
U3	6. jelzőfej	UTP	5
U2	5. jelzőfej	UTP	5
VSZ	U1	UTP	14
U1	4. jelzőfej	UTP	5
Összesen:			**124**

2. táblázat: UTP kábel szü*kséglet*

Az U2 –től U4-ig való vezetés magába foglalja a 7., 3., 2., és 1. jelzőfej érintését is! A logikai modulok tápellátására bőven elegendő a 4 erű kábel. Ennek szükséglete körülbelül megegyezik az UTP kábelek hosszával. Az UTP kábel átlagára 150 Ft/méter.
Intelligens jelzőfej esetén a kábelezés költsége így:

$$C = 124\,m * 150\,Ft + 124\,m * 323\,Ft = 18600\,Ft + 40052\,Ft$$
$$= 58652\,Ft\,(nettó\,ár)$$

Ebből a számításból láthatjuk, hogy az intelligens jelzőfejjel működő rendszer nettó kábelezési költsége valamivel több, mint ötvenezer forint. Ez 58,71 %-a az általános rendszer kábelezési költségének. Ez igen meggyőző lehet egy befektető számára a napjaink gazdasági helyzete mellett. Természetesen a PLC-k árát sem felejtethetjük el, és ez nincs benne ebben a számításban, mégpedig a következők miatt.
Az intelligens jelzőfejjel működő rendszer megalkotásának kezdetekor több variáció is volt arra, hogy milyen logikai modult használjak majd fel a projektben. A tanszéki labor két lehetőséget kínált ebből az egyik volt a Siemens PLC-je, a másik pedig egy BeagleBoard[3]. A relékimenetek könnyű programozhatósága miatt a PLC-t választottam, és a rendszert sikerült működőképesre megalkotnom. Ezzel igazoltam, hogy az intelligens jelzőfej további korszerűbb és akár költséghatékonyabb rendszert nyújt a közúti forgalomirányításban. Az általam felhasznált LOGO! 12/24 RCE nem tartozik az olcsó PLC modulok közé. Tudásának csak egy nagyon kis részét használtam ki. Egy hasonló

[3] A BeagleBoard egy kis teljesítményű, nyílt forráskódú, hitelkártya méretű miniszámítógép. http://beagleboard.org/

paraméterekkel rendelkező kis modult - mely akár lehet egy alaplapra forrasztott áramkör is - sorozatgyártásban nagyon alacsony költséggel elő lehetne állítani. Ennek megtervezése az én dolgozatomnak nem része. Legjobb megoldásnak valamilyen mikrokontrolleres egységet látok. Megvalósításához egy 8 bites mikrokontrollerre, Ethernet portra, relékre, illesztőkártyára és további egységekre a feszültség és áram ellenőrzéshez lenne szükség. Ezek költsége körülbelül 5000 Ft-ot tesz ki. Kiemelném ugyanakkor, hogy ez az összeg egy kiskereskedelmi forgalomban beszerzendő egységek esetén igaz. Sorozatgyártás esetén ez a költség a felére, vagy akár a harmadára lecsökkenhet. Ugyanakkor az alábbi, közelítő költségtervvel kívánom alátámasztani az intelligens jelzőfej létjogosultságát (lásd 3. táblázat).

Szükséges eszközök és kellékek megnevezése	**Hagyományos jelzőlámpás irányítás költsége [Ft]**	**Intelligens jelzőfej-jel megvalósított irányítás költsége [Ft]**	**Pénzbeli megtakarítás [Ft]**
SZRMKVM – J 4x1.5 mm^2 kábel	47804	40052	7752
SZRMKVM – J 5x1.5 mm^2 kábel	20405	-	20405
SZRMKVM – J 19x1.5 mm^2 kábel	31680	-	31680
UTP kábel	-	18600	-18600
Logikai egység az irányításra (mikrokontrolleres)	-	5000	-5000
Összesen:	99889	63652	36237

3. táblázat: Költségvetési terv táblázata

Százalékban kifejezve ez **36,28 %**-os végső **megtakarítási költséget** jelent, az általam felhasznált és becsült adatok alapján. Mindemellett, azt vegyük figyelembe, hogy ez a számítás egy átlagos T-csomópontra lett elvégezve. Tehát a csomópont méretének növekedésével a megtakarítás abszolút értéke természetesen nő, és emellett várhatóan a relatív értéke is növekedni fog.

Ezen felül a költségvetés számítás nem tartalmazza az élő munkaerőben megspórolható költségeket. Egy általános forgalomirányításra épített rendszernél ugyanis a kábelek fektetése, behúzása a csatornákba sok kábel esetén sok időt, illetve emberi munkaerőt igényel. Továbbá több időt és energiát kíván az elektromos kábelek beazonosítása is, hogy még véletlenül se forduljon elő az, hogy egy adott lámpát rossz végre kötnek. A gyakorlatban ezért szokták az izzókat egyenként „kivillogtatni", mielőtt elindítják a csomópontot. Az intelligens jelzőfejekkel vezérelt rendszernél minderre nincs szükség, mert egy elektromos kábel vezet a tápellátáshoz, és a logikai egységekbe kötött UTP kábeleket mindegy, hogy milyen sorrendbe kötjük be az irányító egységekbe. Ezek beazonosítása egyszerűen a kiosztott IP címek alapján megvalósítható.

9. A rendszer továbbfejlesztésének lehetőségei

Az intelligens jelzőfejjel működtetett rendszer előnye tehát, hogy a kábelezés nagy részét el lehet hagyni, ezzel a beruházások költségein csökkenteni lehet. Továbbá a kiépítése gyorsabb és egyszerűbb, és alkalmas egymagában egyszerűbb terelések vezérlésénél a forgalom irányítására.

A kidolgozott rendszerben további fejlesztést kíván az árammal való ellenőrzés (piros izzó kiégését mindig ezzel kell ellenőrizni). Ez a rész a biztonsági funkciók közül még nem készült el, de közeljövőben egy árammérő modul beépítésével tervezem megvalósítani. Hasonló kialakítással, mint a feszültség ellenőrzésnél, a modulról kapott jelet fogom a PLC bemenetére visszavezetni, és azt ellenőrizni.

Az általam fejlesztett rendszerben alkalmazott PLC nem a konkrét feladathoz fejlesztett egység, hanem egy jóval nagyobb tudású vezérlő számítógép. A PLC helyett olyan egyszerűbb cél hardvert lehet érdemes alkalmazni, amely alkalmas a szükséges logikai műveletek elvégzésére, és megvalósítható vele az IP alapú hálózati kommunikáció.

A rendszer továbbfejlesztésének másik része az alkalmazott hálózati kommunikáció revíziója. A kialakított rendszerben TCP/IP protokollt alkalmaztam, amely sebessége (lévén nem real-time protokoll) nagyobb csomópontok esetén már okozhat problémát. A kommunikációs sebesség felgyorsítása érdekében az UDP (User Data Protocol) protokoll alkalmazása tűnik kézenfekvőnek (megfelelő ellenőrzéssel).

Érdemes lenne az intelligens jelzőfejjel működő rendszerrel elérhető megtakarításokra nézve, különböző nagyságú csomópontok esetére egy pontosabb gazdasági számítást végezni.

10. Összefoglaló

Az intelligens jelzőfejjel való forgalomirányítás rendszerét labor körülmények között sikerült megvalósítanom, így azt mondhatom, hogy a projekt kimenete mindenképp pozitív.

A forgalomirányító berendezés (Actros) és a PLC-k között a kapcsolatot felépítettem, így képes a két eszköz egymásnak feladatokat adni és kommunikálni. Az irányítást az Actros oldaláról újra kellett építeni. A meglévő program jó alappal szolgált, de ki kellett egészíteni számos új osztállyal, és függvénnyel. Emellett Java-ban való szálkezeléssel is meg kellett ismerkednem, hogy a rövid idő alatt elvégzendő feladatokat is maradéktalanul teljesíteni tudjam.

A jelzőfejek izzóinak kapcsolgatása a PLC-ről megvalósult, a logikai modulra tervezett programmal. Ennek tervezésébe bele kellett kalkulálni a forgalomirányító berendezéssel való kapcsolat ellenőrzésére szolgáló funkciót. Ezek mellett sikerült a rendszer biztonságáról is gondoskodni. Az eszközök kapcsolatának megszakadásával a rendszer sárga villogó üzemmódba kerül. Feszültség ellenőrzésével vizsgálni tudjuk a zöld izzók működését. Az ellenőrzést mindenképp ki kellene terjeszteni áram ellenőrzésre is, mert a piros izzók kiégését ezzel kell vizsgálni. Ennek kivitelezése jövőbeli terveim között van.

A kivitelezett rendszert összehasonlítottam egy meglévővel. Az összehasonlítás eredményeképp megállapítottam, hogy az intelligens jelzőfejjel való forgalomirányítás lényegesen kevesebb kábelezést igényel és ezzel jelentős költségmegtakarítást érhető el.

Irodalomjegyzék

[1] DEBRECZENI Gábor Béla (2013) *Közúti Forgalomtechnika* [elektronikus jegyzet]. Budapest, KUKG, URL: http://www.kukg.bme.hu utolsó letöltés: 2013.10.20.

[2] HIROU Sakai, TOSHIHIDE Kawamura (1987) *Light – Emitting Diode*, Tokyo, US Patent, Patent number: 4698730

[3] SIEMENS (2009) *LOGO! V5 manual*. München, Siemens

[4] SIEMENS (1999) *LOGO! Soft Comfort* [elektronikus dokumentum]. München, Siemens, utolsó letöltés: 2013.10.20.

[5] TETTAMANTI Tamás (2010) *ACTROS VTC 3000* [elektronikus jegyzet]. Budapest, KJIT, URL: http://kjit.bme.hu utolsó letöltés: 2013.10.20.

[6] LUDVIG Ádám (2013) *Siemens Logo 0BA7 kommunikációs beállítása a libnodave szoftverkönyvtár alkalmazásához* [elektronikus dokumentum].

[7] TARNAI Géza, BOKOR József, SÁGHI Balázs, BARANYI Edit, BÉCSI Tamás (2011) *Irányítástechnika I.* Budapest, Typotex, ISBN 978-963-279-602-4

[8] POLGÁR János, TETTAMANTI Tamás (2010) *Forgalomtechnikai kód az ACTROS VTC 3000 forgalomirányító berendezésben* [elektronikus jegyzet]. Budapest, KJIT, URL: http://kjit.bme.hu utolsó letöltés: 2013.10.20.

[9] JUHÁSZ József (1972) *Magyar Értelmező Kéziszótár.* Budapest, Akadémia, ISBN 963056212X p. 139

[10] BODÓ Lajos (2008) *A jelzőlámpás irányítás villamos munkáinak terve* [elektronikus dokumentum]. Budapest, Swarco Hungary

Ábrajegyzék

Táblázatjegyzék

Melléklet

Mintapélda a jelzőlámpa-kapcsolás megvalósítására Logo Soft Comfortban

Első lépésként, megoldottam a jelzőlámpa váltás, izzók működtetésének feladatát a PLC-vel. Ez később az Eclipse-ből lett leprogramozva, azaz az Eclipse-ben kiadott utasításokat a LOGO! hajtja végre a köztük fennálló kapcsolat megléte esetén. Ennek ellenére nem árt, ha tisztában vagyunk a Logo Soft Comfort kezelésével és egy rövid, ide illő példával betekintést nyerhetünk a szoftver olyan tulajdonságaiba, melyeket később is használtam.

A tervezést és programozást a LOGO!-n is el lehetett volna végezni, de a kis kijelzője és a szerény vezérlőgombjai miatt elég körülményes, így inkább a Siemens által erre a célra fejlesztett Logo Soft Comfort szoftvert vettem igénybe. Négy állapotot kellett megjelenítenem, ezek sorban: Piros, Piros-Sárga, Zöld, Sárga. Az állapotokat felvettem egy táblázatba (lásd 4. táblázat). A négy darab állapotot három darab izzó hajtotta végre. Nekünk az kellett, hogy az izzók közül, megfelelő időben a várt jelző kapcsoljon be. A problémát legegyszerűbben Karnaugh - tábla felrajzolásával tudtam megoldani (lásd 5. táblázat).

	A	B	C	Piros	Sárga	Zöld
0.	0	0	0	0	1	0
1.	0	0	1	0	1	0
2.	0	1	0	0	0	1
3.	0	1	1	0	0	1
4.	1	0	0	1	1	0
5.	1	0	1	1	1	0
6.	1	1	0	1	0	0
7.	1	1	1	1	0	0

4. táblázat: Az állapotokat mutató táblázat

Piros

A \ B	0	0	0	0
1	1	1	1	1

C

Sárga

A \ B	1	1	0	0
1	1	1	0	0

C

Zöld

A \ B	0	0	1	1
0	0	0	0	0

C

5. táblázat: Karnaugh táblák

Az eredmények: $F_P = A$

$F_S = \overline{B}$

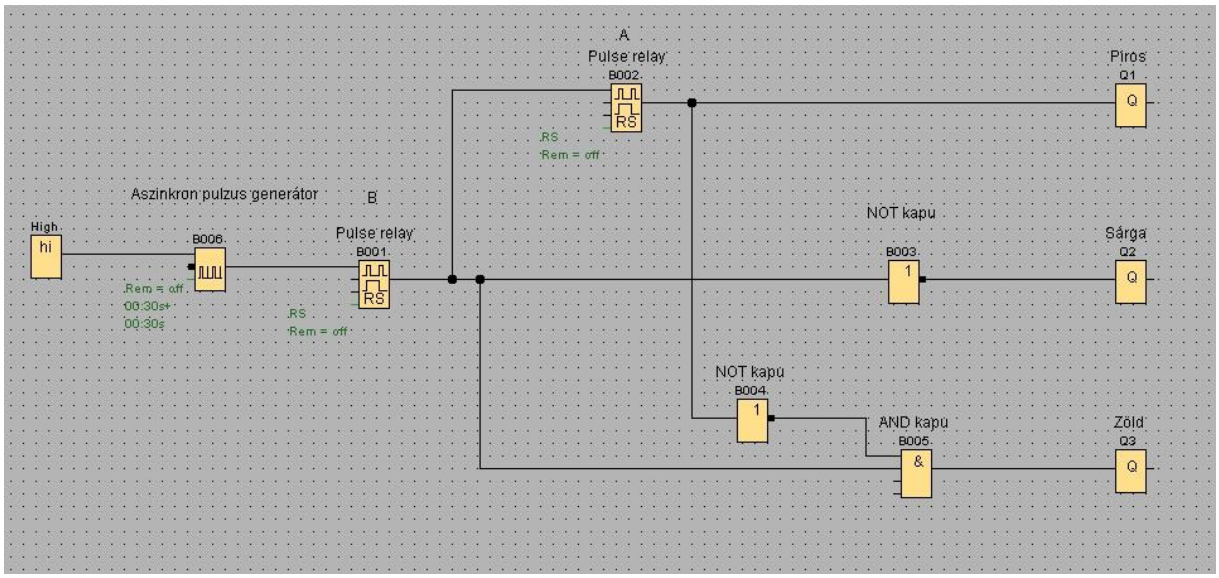
$F_Z = \overline{A}B$ [7]

Az eredmények ismeretében könnyen meg tudtam rajzolni ezeket az állapotokat a Logo Soft Comfortban. Azt volt a cél, hogy időben változva a jelzők állapotai váltogassanak, és ne nekem kelljen az állapotátmeneteket manuálisan végrehajtani, ezért bevezettem a rendszerbe egy aszinkron pulzus generátort. Ez a kimentén ugráltatja a jelet ’0’ és ’1’ között, a tulajdonságaiban megadott időintervallum szerint. Majd felhelyeztem a rajzpalettára, egy High blokkot, amely folyamatosan ’1’-et adott a kimenetére, így meghajtotta a mögé kötött aszinkron pulzus generátort (lásd 27. ábra). A generátoron kettőt kattintva megjelenik egy ablak, ami a blokk tulajdonságait tartalmazza. Itt beállítottam a Pulse Width -re 30 Seconds [s: 1/100 s] -ot, és az Interpulse Width -re is 30 Seconds [s: 1/100 s] -ot. Ezzel azt az eredményt kaptam, hogy a generátor 30 századmásodpercenként kapcsolgatta a belőle kijövő jelet, ami nem más, mint a 4. táblázat ’C’ oszlopa. A ’B’, és az ’A’ oszlopot is elő kellett állítanom, ezért szükségem volt két ’Pulse relay’ (Impulzus relé)-re. Az impulzus relé a következőképp működik. Ha ’1’-et adok a bemenetére, akkor a kimentén megjelenik az ’1’. Ha ráadom a ’0’-át, akkor húzva marad, és tartja az ’1’-et a kimenetén, majd ha újra ’1’-et adok rá, akkor megtörténik a reset funkció és a kimenete ’0’-ra vált. Lényegében úgy működik, mint egy SR tároló.

Egy ilyen Pulse relay-t beraktam az aszinkron pulzus generátor mögé, így a ’0’,’1’,’0’,’1’,’0’,’1’,’0’,’1’ jelsorozatot át tudtam alakítani ’0’,’0’,’1’,’1’, ’0’,’0’,’1’,’1’ jelsorozattá, amivel megkaptam a 4. táblázat ’B’ oszlopát. Majd e mögé egy újabb Pulse relay-t helyeztem és így eljutottam a ’0’,’0’, ’0’,’0’,’1’,’1’, ’1’,’1’ jelsorozatig, ami az 4. táblázat ’A’ oszlopának felel meg (lásd 27. ábra). Ezek után már csak a Karnaugh – táblákból (lásd 5. táblázat) megkapott eredményeket kellett feldolgoznom.

- A piros égést megkaptam, amikor egy Output (Kimenet) blokkot felhelyeztem a rajztáblára és bekötöttem az 4. táblázat után ’A’-nak elnevezett utolsó Pulse relay -be.
- A sárga égést megkaptam, mikor elhelyeztem egy Kimenet blokkot, és ezt egy NOT (negáció) kapuval sorba kötve az 4. táblázat után ’B’-nek elnevezett első Pulse relay kimenetére kötöttem.
- Végül a zöld égést úgy állítottam elő, hogy egy újabb Kimenet blokkot helyeztem a palettára. Ennek bemenetét egy AND (és) kapuval kötöttem össze. Az AND kapu egyik bementére rákötöttem, az ’A’ kimenetét egy NOT-al közbeiktatva, a másik bemenetére pedig a ’B’ kimenetét kötöttem.

Az egyes blokkoknak lehet nevet adni, amit érdemes is megtenni, hiszen így átláthatóbbá válik a rendszer. A blokkra duplán kattintva előugrik egy ablak, melynek mindig a legutolsó füle a Comment fül. Ez alatt tetszés szerinti nevet lehet változtatni, ami a blokk fölött fog megjelenni. A megjelenő felirat mozgatható, ha épp útban lenne. Az elkészült programot a 27. ábra szemlélteti.



27. ábra: Blokkdiagram a jelzőlámpás váltásról Logo Soft Comfortban

(forrás: Logo Soft Comfort V.7. [program])

Az elkészült áramkör a Logo Soft Comfortban is futtatható, egy szimuláció keretében. Ezt az F3 gomb megnyomásával lehet elindítani. Ha tervezés jó, akkor a kimeneteken a lámpák a megfelelő sorrendben villannak fel. Természetesen a program LOGO!-ra is feltölthető, és ha a fizikai kapcsolat fenn áll a jelzőfejekkel, akkor az izzók megfelelő időben és sorrendben villannak majd fel.