Multidiszciplináris Tudományok, 14. kötet, 1. sz. (2024) pp. 83–93. **https://doi.org/10.35925/j.multi.2024.1.8**

ÉPÜLETVILÁGÍTÁSI VEZÉRLŐALGORITMUS LÉTREHOZÁSA DMX512 PROTOKOLL ALKALMAZÁSÁVAL

Simon Róbert
doktorandusz, tanársegéd, Miskolci Egyetem, Automatizálási és Infokommunikációs Intézet
3515 Miskolc-Egyetemváros, e-mail: robert.simon@uni-miskolc.hu

Trohák Attila
egyetemi docens, Miskolci Egyetem, Automatizálási és Infokommunikációs Intézet
3515 Miskolc-Egyetemváros, e-mail: attila.trohak@uni-miskolc.hu

Absztrakt

Az épületautomatizálás során számos olyan feladattal találkozunk, amelyek már egy meglévő ipari technológiát alkalmaznak a ház intelligens működtetéséhez. Kérdés azonban, hogy ezeknek a bejáratott ipari szabványok alkalmazásának van-e megfizethető alternatívája az okosotthonok létesítésekor. A piaci szereplők több új, kifejezetten létesítményautomatizálásra alkalmas szabványt is sikerrel integráltak az elmúlt években az okosotthonok létrehozására, ám sok esetben ezek alkalmazásának költsége jóval meghaladja egy normál építkezés egyébként is magas költségvetését. Cikkünk alapvetően egy olyan világítástechnikai szabvány feldolgozásával foglalkozik, amely használata rendkívül alacsony költség mellett tudja megfelelő mértékben nyújtani a már elterjedt piaci megoldások kényelmét. A cikkben a szükséges vezérlőalgoritmusok megvalósításával fogunk foglalkozni, azaz egy világításvezérlési rendszer implementálásával egy WAGO PFC200 750-8212 PLC-n a DMX512-es kommunikációs protokollt felhasználva.

Kulcsszavak: *WAGO, DMX512, épületautomatizálás, okosotthon, világításvezérlés*

Abstract

In building automation, we encounter many tasks that already use an existing industrial technology to make the building work intelligently. The question is whether there is an affordable alternative to applying these established industry standards to smart homes. Several new standards specifically for facility automation have been successfully integrated by market players in recent years to create smart homes, but in many cases the cost of implementing these standards is well beyond the already high budget of a normal construction project. Our article is essentially a discussion of a lighting standard that, when used at very low cost, can provide the convenience of existing market solutions at a reasonable level. In our article we deal with the implementation of the necessary control algorithms to implement a lighting control system on a WAGO PFC200 750-8212 PLC using the DMX512 communication protocol.

Keywords: *WAGO, DMX512, building automation, smart home, lighting control*

1. Bevezetés

A DMX512 szabvány alapvetően egy színházi világításvezérlési szabvány volt, amely aztán gyorsan egyeduralkodóvá is vált a teljes szórakoztatóiparban. Persze nem eredeti formájában, mivel az évek

során folyamatosan tökéletesítették azt. Így először a DMX512/1990, majd pedig a DMX512A szabvány váltotta fel és egészítette ki az eredeti leírásokat. Megjelentek újabb csatlakozók a leírásokban, és ezzel együtt újabb és újabb területeken hódított a protokoll – olykor a világításvezérléstől teljesen eltérő felhasználási helyeken.

A DMX512 tagadhatatlan előnye, hogy olcsó technológiáról van szó, ekképpen egy kész rendszer is igen költséghatékony megoldás lehet. A protokoll egy másik előnye, hogy elterjedtségének és széles körű felhasználásának köszönhetően mára igen sokféle dekóder került forgalomba. Ezek közül számunkra a világításvezérléssel kapcsolatos dekóderek érdekesek, hiszen a cikkünk szempontjából jelenleg ezek fontosak.

Világításvezérlés során a DMX512 szabványt jellemzően intelligens lámpákban, színes fényvetőkben alkalmazzák, elsősorban a szórakoztatóiparban, koncerteken és rendezvényeken. Vezérlésüket jellemzően ún. DMX-keverőpultokkal oldják meg, ám tekintettel arra, hogy a DMX512 szabvány kommunikációja csupán egyszerű, csatornaérték típusú üzeneteket használ, így a fényforrások használata általában egyedi beállításokat kíván.

A világításvezérléssel kapcsolatos DMX512 dekóderek másik és lényegesen kisebb csoportja a statikus fényelemeket vezérli. Idetartoznak a színes LED-szalagok, vagy a hálózati feszültséggel működő normál lámpatestek vezérlései is. Mi ez utóbbi fényforrásokkal foglalkozunk, hiszen egy ház vagy épület élettereinek megvilágítása jellemzően statikus fényerőt kíván meg. A fényforrások napjainkban igen sokféle fajtában érhetők el, közös pontjuk csupán az őket működtető feszültségben van, amely Magyarországon általában háromféle lehet: 12VDC, 24VDC és 230VAC. A törpefeszültséggel működők jellemzően fényerőszabályozhatók (dimmelhetők), ám a hálózati feszültséggel működő fényforrásoknál már nem ilyen egyértelmű a helyzet. A hagyományos izzószálakkal nincsen probléma, mivel egy ohmikus ellenállásról beszélünk, amelynél a feszültség csökkentésével a fényerő is csökken. LED-es fényforrások esetén azonban mindig van a fénykibocsátó diódák előtt valamilyen meghajtó elektronika, amelyeket külön fel kell készíteni, ha a primer oldali feszültség szabályozott. Ezért fényerőszabályozáshoz mindig csak az ehhez megfelelő, dimmelhető LED-fényforrást tudjuk alkalmazni. A DMX-vezérlés szempontjából szerencsére mindhárom feszültséghez érhető el DMX512 dekóder. Ebben a cikkben csak a hálózati feszültséggel (230VAC) működő fényforrást vizsgáljuk, ám működtetésének analógiája bármely más feszültséggel működő vezérlőre is átültethető.

Látható tehát, hogy a DMX512 protokoll használatához alapvetően minden a rendelkezésünkre áll, hogy egy okosotthon világításvezérlését megvalósítsuk. Van PLC-nk, ami képes az üzenetek DMX512 protokoll szerinti kódolására és van dekóderünk, ami képes ezt szabályozott feszültségszintre konvertálni. És vannak végül fényforrásaink is, amelyek ezen szabályozott feszültségeket szabályozott fényerőben reprezentálják. Nézzük, hogyan néz ki ennek a szoftveres megvalósítása!

2. A WAGO PFC200 750-8212 PLC programozásának elméleti háttere

A WAGO PFC200-as szériájú PLC-t kétféle fejlesztőkörnyezetből lehet programozni. Az egyik, a WAGO által fejlesztett e!cockpit, amelybe a Codesys 3-as verzióját integrálták, a másik pedig a Wago-I/O-Pro, amely a Codesys 2.3 verzióját használja. Alapvetően az e!cockpit használata javasolt, mivel modernebb a fejlesztőkörnyezet, folyamatosan fejlesztik és a legújabb WAGO PLC-ket is alapvetően erre a környezetre fejlesztik. A WAGO-I/O-Pro elsősorban a régebbi kiadású, régebbi generációs PLC-k fejlesztőkörnyezete, ezért használata ott indokolt, ahol egy korábbi eszközt vált le az újabb PLC. Így nem szükséges a már meglévő PLC-programot az új e!cockpitbe importálni, hanem használható a régi rendszerrel is. Fontos megjegyezni azonban, hogy ez utóbbi esetben a felhasználható

programmemória 16 MB-ra, az adatmemória mérete pedig 64 MB-ra csökken. E!cockpit használata esetén ezek 32 MB és 128 MB méretűek.

2.1. Az IEC-61131 szabvány szerinti POU-k felépítése

A felhasznált PLC egy ún. *„soft-PLC”*, amely azt jelenti, hogy felépítése és programozhatósága az IEC-61131 szabvány szerint történik. Ez a szabvány a felhasználói program futását nem egy fő organizációs blokk köré szervezi, hanem ún. POU-kat (Program Organization Unit – programszervezési egység) alkalmaz. A POU-k a szabvány szerint négyféle típusba sorolhatók (Ajtonyi, 2007).

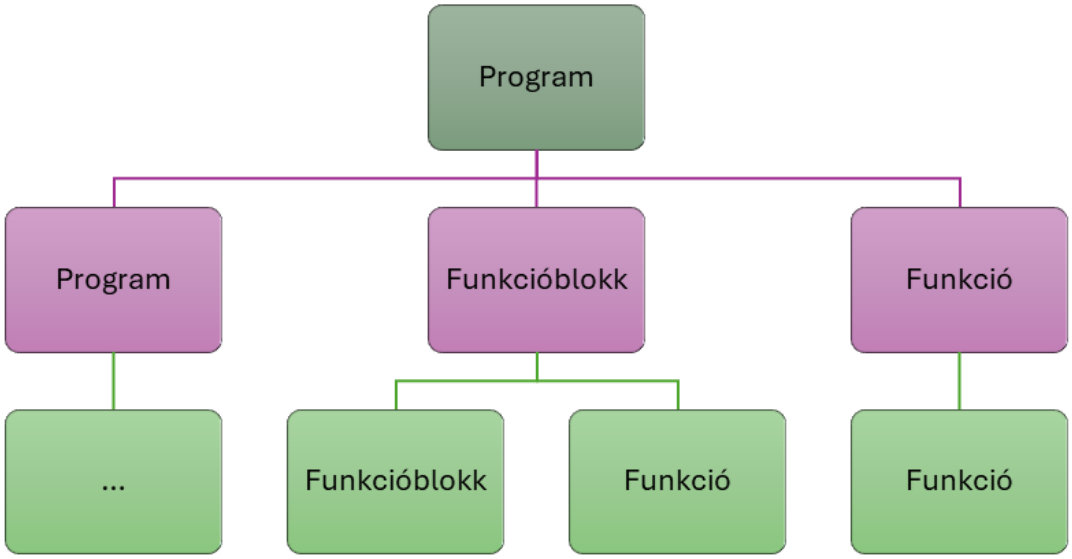
2.1.1. Taszk

Logikai szervezési egység, amely alapvetően a *programok* futtatását kezeli. Ez a kezelés történhet eseményvezérelten, vagy időzítve. Ez utóbbi a gyakrabban használt megoldás, hiszen alapvetően a PLC-programok ciklikus futású programok, bár bizonyos esetekben az eseményvezérelt futtatás megkerülhetetlen. Ilyen esemény pl. a PLC indulása, amelyhez inicializálási feladatokat köthetünk. Ciklikus futtatás esetén igen fontos a *taszkok* priorizálása, mivel végső soron ez fogja a *taszk* által futtatott *programok* számára a valósidejűséget biztosítani. A WAGO PFC200 750-8212 PLC preemptív taszk ütemezést alkalmaz, ami azt jelenti, hogy a magasabb prioritású *taszkok* képesek megszakítani az alacsonyabb prioritású *taszkok* futását, ezáltal biztosítva a pontos ütemezést a valósidejűséget megkívánó alkalmazások számára. Nem preemptív ütemezés esetén a *taszkok* futásának megszakíthatatlansága a fő szempont, így ilyen típusú ütemezés esetén a feladatok végrehajtásának ideje az, ami változatlan marad.

2.1.2. Program

A *programok* a hívási struktúrában legfelül helyezkednek el *(1. ábra)*. Érdemes megjegyezni, hogy a szabvány – és így a fejlesztőkörnyezetek is – támogatják a *programok* hívhatóságát a *funkcióblokkok* felől is. Tekintettel azonban arra, hogy a POU-k hívási struktúrája felülről lefelé irányban halad, e hívás ezzel az iránnyal éppen ellentétes lenne. Ezzel együtt érdemes tudni, hogy van erre lehetőség, ám használata – az *alkalmazás* egészére nézve – nem logikus. A *programok* tehát olyan programszervezési egységek, amelyek általában egy *alkalmazás* komplett szegmensét kezelik. Ha pl. az *alkalmazás* egy okosotthon üzemeltetése, akkor annak egy lehetséges *programja* lehet a fűtés, vagy a világítás üzemeltetése. Fűtés esetén kapcsolni kell a kazánt, szabályozni kell az előremenő vízhőmérsékletet, indítani kell a szivattyúkat stb., amelyek mind olyan részfeladatok, amelyeket majd a *funkcióblokkok* fognak lekezelni. A világításvezérlés esetében kezelni kell a lámpakapcsolókat – amelyek természetesen lehetnek intelligens kapcsolók is – a fényforrások direkt kapcsolását, vagy terepi buszon keresztüli szabályozását és amennyiben a ház rendelkezik fénymérő szenzorokkal, úgy adott esetben a konstans fényáramok biztosítását a megjelölt helyiségekben. Ezek megint olyan alfeladatok, amelyeket jellemzően a következő hívási szinten, a *funkcióblokkokkal* fogunk lekezelni.

Ez tehát a *programok* logikai szerepe a hívási struktúrában, azonban szerepük nem csupán itt merül ki, hiszen a szabvány az ő részükre biztosítja *globális változók* definiálását. Ám ettől a fejlesztőkörnyezetek gyakran eltérnek, mégpedig úgy, hogy ún. *globális változólistákat* alkalmaznak. Ezekre hivatkozva aztán a bennük definiált változók a teljes *alkalmazás struktúrában* elérhetővé válnak. Ekképpen nélkülözhetővé válik az *external* változó típus definiálása is.



1. ábra. *IEC 61131 szabvány szerinti POU hívási struktúra*

2.1.3. Funkcióblokk

A *funkcióblokkok* az IEC-61131 szabvány programszervezési egységei közül a leggyakrabban használt POU-k. Ezek az egységek valósítják meg általában a tényleges munkavégzést önállóan, vagy még további alegységekre bontva. Ezek a további alegységek is általában szintén *funkcióblokkok.* Közös jellemzőjük, hogy futtatásuk önmagában nem lehetséges, mivel használatuk előtt példányosítani kell őket. A példányosítást általában a hívó POU-ban végezzük el, jellemzően lokális változóként definiálva azt, méghozzá úgy, hogy a változó adattípusának a *funkcióblokk* azonosítóját adjuk meg. Ezzel létrehozzuk a *funkcióblokk* ún. *adatblokkját.* Sajátossága ezen *adatblokkoknak,* hogy jelenlétük a memóriában, a PLC-program indulása után statikus, vagyis a *funkcióblokk* meghívása után az *adatblokk* által elfoglalt memóriaterület nem szabadul fel. Ekképpen a benne tárolt információk, a PLC-program futása alatt végig megmaradnak, így a példányosított *funkcióblokk* a hozzá tartozó *adatblokk* tartalmát a következő futás alkalmával úgy használhatja, hogy az előző futás eredményei abban még benne maradtak. Ekképpen státuszállapotokat és az algoritmus működését befolyásoló információkat is tárolhatunk benne a *funkcióblokk* számára. A PLC-k fizikai felépítése általában lehetővé teszi ún. reteszelt memóriaterület alkalmazását is, amelynek tulajdonsága, hogy a PLC nem várt újraindulása esetén – pl. hálózati feszültség kimaradás – is megmarad a változó tartalma. Amennyiben az *adatblokk* tartalmaz reteszelt változókat is, úgy a változók értékei nem csak a következő futás alkalmával lesznek olvashatók, hanem újraindulás után is.

Érdemes azonban megkülönböztetett figyelmet fordítani erre a memóriaterületre, ugyanis mérete – a normál használatú memóriához képest – elenyésző. Az általunk használt és itt is bemutatott PLC reteszelt memóriaterülete mindösszesen 128 KB.

2.1.4. Funkció

Az IEC-61131 szabvány legkisebb, legegyszerűbb programszervezési egysége, amely célirányos felhasználásra készült. Alapvetően komplex kifejezések vagy összetett algoritmusok egyszerű hivatkozhatóságát teszi lehetővé a többi POU számára. Csak bemeneti paraméterei lehetnek, kimenetiek nem, mivel a *funkció* – vagy másnéven *függvény* – önmagában hordozza az értéket, vagyis az algoritmus visszatérési érték a híváskor, valós időben kerül meghatározásra.

Fontos különbsége továbbá a *funkcióblokkhoz* képest, hogy nem rendelkezik *adatblokkal,* így példányosítása sem szükséges.

2.1.5. Alkalmazás

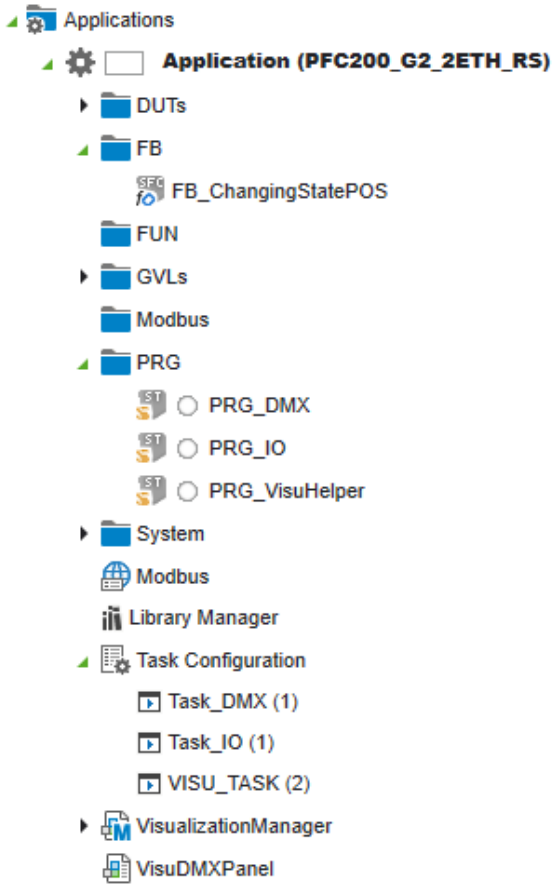
A PLC-n futó *programok* jellemzően egy-egy *alkalmazás* programszervezési egységei. A *programok* futtatásáért a *taszkok* felelősek, amelyek végső soron egy *erőforráson* fognak lefutni. Az *erőforrás* egy olyan fizikailag is létező egység, amely rendelkezik CPU-val, memóriával, háttértárral és képes a logikai algoritmusok futtatására. A PLC is egy ilyen *erőforrás*. Az *alkalmazást* nem szabályozza egyébiránt az IEC 61131 szabvány, ám logikai szerepe és tényleges mivolta nem kérdéses, hiszen végső soron az *alkalmazás* fogja meghatározni a *programok* számát és tulajdonságait. Az *alkalmazás* tulajdonképpen a *programok* felett álló, legfelsőbb szervezőelv.

3. A világításvezérlő szoftver

Az elméleti háttér áttekintése után nézzük meg, hogy miként valósítható meg a vezérlőszoftver a konkrét PLC-n. Az eszközünk programozását az IEC-61131 szabvány szerint tudjuk elvégezni, tehát taszkszervezésű *programokat* kell létrehoznunk, amelyek az *alkalmazás* jól definiált feladatait fogják elvégezni. A *programok* futtatásához definiálnunk kell az őket ciklikusan meghívó *taszkokat,* és egyúttal végig kell gondolni azok prioritását is.

3.1. Programok

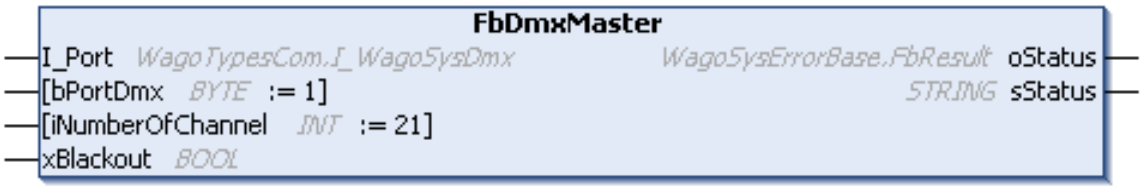
Az *alkalmazás* világításvezérlési feladatokat fog ellátni DMX512 protokollt felhasználva. Ezért mindenekelőtt szükség lesz egy olyan *programra*, amelyik a csatornákon megjeleníteni kívánt fényerősségek értékét elküldi a terepi buszon keresztül a DMX512 dekódereknek – ennek neve *PRG_DMX* lesz. Az egyes csatornák értékeinek módosíthatóságához szükséges továbbá definiálnunk egy olyan HMI-t is, ami lehetővé teszi ezt nekünk, egyúttal pedig vizuálisan is láthatóvá teszi a csatornák pillanatnyi állapotát. Ezért szükség lesz egy másik *programra* is, amelyik a HMI-t vezérli majd. Ennek neve a *VisuElems.Visu_Prg* lesz, amit a fejlesztőkörnyezet saját maga definiál, amikor létrehozunk egy új vizualizációt. Emiatt ez nem is látható a *2. ábra* által bemutatott struktúrában, hiszen ez nem egy különálló *program*, hanem a *VisuElems* objektum egy *programja*. A működéshez szükséges továbbá egy olyan technikai *program* is, ami az IO-csatornák – jelen esetben tehát a manuális kapcsolók állapotát – fogja kezelni. Ez lesz a *PRG_IO*. Azért definiáltunk ehhez egy külön *programot,* mivel egy okosotthon vezérlésben számos be- és kimenet van, amelyeket majd egyszerre kezelnünk kell és sok esetben teljesen eltérő módon. Egy világításkapcsoló – a klasszikus le/fel kapcsolás mellett – az okosotthonok esetében jellemzően dimmelési feladatokat is ellát, így figyelnünk kell a kapcsolók felől érkező magas jelszintek időtartamát és különbséget kell tennünk az impulzus jellegű rövid kapcsolások, illetve a jól elkülöníthetően hosszabb, dimmelési parancsok között. Az utolsó *programunk* a *PRG_VisuHelper*, ez állít elő a *HMI* számára olyan származtatott értékeket, amelyek a megjelenítést, illetve az egyes paraméterek limitálását segítik.



2. ábra. *POU-k struktúrája az e!cocpiten belül definiálva*

3.1.1. PRG_DMX

Ennek a *programnak* a feladata a jelek küldése a dekóderek felé. Ehhez szükséges a *WagoAppDMX* könyvtár telepítése és azon belül használni kell az *FbDmxMaster funkcióblokkot.* Ez a könyvtár csak a 750-652 modul soros portjával képes együttműködni, a PLC beépített soros portját nem tudja használni. A beépített csatolófelületet a PLC elsődlegesen a Modbus RTU kommunikációra használja (wago.com).



3. ábra. *Az FbDmxMaster funkcióblokk bemenő és kimenő paraméterei*

Az *I_Port* annak a modulnak a nevét jelöli, amelyiken keresztül a kommunikáció zajlik. A mi esetünkben ez az *RS232_485_Interface*. A *bPortDmx* a DMX-univerzumot jelöli, amin belül értelmezhetjük az 512 csatornát. Ez alapértelmezetten 1, mivel általában egyetlen univerzumot használunk csupán. Abban az esetben, ha több univerzumunk lenne, akkor azt egy újabb 750-652-es modul használatával tudnánk csak elérni. Annak természetesen a neve is más lenne és a *bPortDmx* változónak is más értéket kellene adjunk. Az *iNumberOfChannel* jelöli az 512-ből ténylegesen felhasznált csatornák számát. Az *xBlackout* pedig egy olyan bináris változó, amelyik TRUE értéke a teljes DMX-univerzumot elsötétíti. Ez utóbbi változót használjuk a HMI-ben a szoftveres elsötétítéshez. Ebben az esetben ugyanis mindegy, hogy az egyes csatornák felé milyen értéket küldünk, az *xBlackout* TRUE értéke esetén mindegyik érték felülíródik. Az *FbDmxMaster funkcióblokk* kimenő paraméterei elsődlegesen a *funkcióblokk* hívásakor bekövetkező hibás vagy normál lefutásról ad visszajelzést objektum és szöveges formátumú üzeneten keresztül. Ilyen hibás lefutást okozhat a rossz paraméterezés, mint pl. nem létező modulra vagy csatornaszámra történő hivatkozás.

4. ábra. *Az FuDmxSetChannel bemenő és kimenő paraméterei*

Miután beállítottuk az univerzumot a Master *funkcióblokkal*, a csatornák értékét az *FuDmxSetChannel függvénnyel* fogjuk tudni állítani. Itt kap újra szerepet a *bPortDmx* változó, amit megismertünk a Master *funkcióblokknál*, hiszen az az érték, amit ott hozzárendeltünk a fizikai modulhoz, az fogja definiálni ennek a csatornának az univerzumát. Az *iChannel* és *bValue* változók a csatornaérték-párosokat jelentik. Egy csatorna 256-féle értéket képes felvenni (ezért a byte adattípus), míg a csatornák száma 512 lehet (ezért az int adattípus). A *függvény FALSE* értéket ad vissza, ha a paraméterekkel az univerzumban nem létező csatornát címzünk.

5. ábra. *Az FbDmxMaster funkcióblokk Job metódusa*

Az *FbDmxMaster* meghívása csupán inicializáláskor, illetve a blackout változásakor meghívandó funkcióblokk. Ha a csatornák értékét módosítottuk, akkor az egyes csatornák értékeinek beállítása után kell meghívni az *FbDmxMaster funkcióblokk Job metódusát.* Ez lesz tulajdonképpen az az eljárás, ami ténylegesen vezérli a 750-652-es fizikai modult és kiküldi a terepi buszra a csatornaérték-párosokat.

3.1.2. Megjegyzések a DMX512 protokoll használatához

Fontos megjegyezni, hogy az egyes dekódereken beállított címérték nem magának az eszköznek a címét jelenti, hanem annak a csatornának a számát, ahová a dekóder a saját, első csatornáját értelmezi. Ha tehát egy kétcsatornás eszközünk van, és mondjuk a 13-as csatornát állítjuk be, mint kezdőcsatornát, akkor a dekóderünk a 13-as és 14-es DMX-csatornák értékeit fogja reprezentálni a rákötött eszközökön.

Ezért a világítás tervezésekor már célszerű előre átgondolkodva meghatározni, hogy az egyes helyiségek hány fényforrást fognak tartalmazni. Egyúttal minimális ráhagyással is célszerű számolni, hogy egy esetleges átépítés, bővítés miatt ne kelljen a többi csatornát áthangolni. Azért minimálissal célszerű csupán számolni, mivel a csatornák számának növekedésével egyúttal a küldési idő is növekszik. Sok csatorna esetén pedig már lassulás tapasztalható, ami főleg folyamatos fényerőszabályozás esetén fog lépcsős fényerőugrásokat eredményezni, illetve a fel-le kapcsolások reakcióideje nő meg. Mennyiben lehet probléma ez egy családi ház esetén? Vegyünk egy kellően nagy ingatlant az alábbiak szerint:

- Hálószoba 12 m^2 4 db (5 fényforrás, 2 LED-szalag)
- Hálószoba 16 m^2 1 db (7 fényforrás, 3 LED-szalag)
- Gardrób 6 m^2 2 db (2 fényforrás, 5 LED-szalag)
- Nappali 40 m^2 1 db (8 fényforrás, 5 LED-szalag)
- Konyha 10 m^2 1 db (3 fényforrás, 2 LED-szalag)
- Mosókonyha 6 m^2 1 db (2 fényforrás, 0 LED-szalag)
- Előszoba 3 m^2 1 db (2 fényforrás, 1 LED-szalag)
- Fürdőszoba 8 m^2 2 db (3 fényforrás, 2 LED-szalag)
- Toilette 4 m^2 2 db (2 fényforrás, 1 LED-szalag)
- Közlekedő 8 m^2 2 db (2 fényforrás, 1 LED-szalag)
- Lépcsőház 10 m^2 1 db (2 fényforrás, 1 LED-szalag)

A fényforrások alatt egyetlen csatornát értek, pl. különálló lámpaként, míg LED-szalagok esetén szalagonként 4-et (RGBW). Ebben az esetben az alábbi eredményt kapjuk:

$$N_{Channel} = (5 + 7 + 2 + 8 + 3 + 2 + 2 + 3 + 2 + 2 + 2) * 1$$
$$+ (2 + 3 + 5 + 5 + 2 + 0 + 1 + 2 + 1 + 1 + 1) * 4 = 38 + 23 * 4 = 130$$

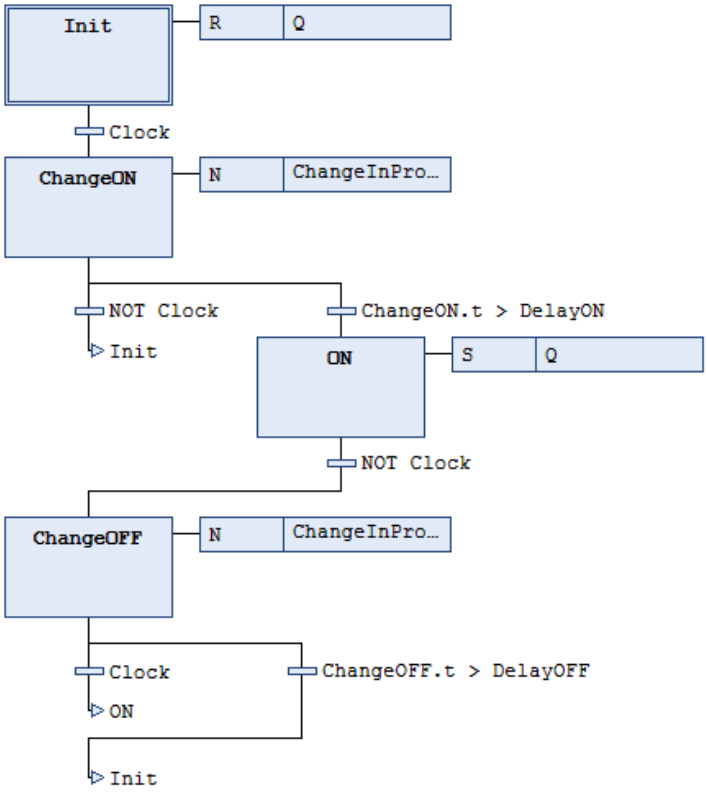
Tehát egy ekkora (nettó 185 m^2) ház esetén 130 csatornával tudunk számolni egy nagyon színes, nagyon gazdag világítási rendszert alkalmazva. Tapasztalatom szerint a 200 csatorna körül lassul már érezhetően a rendszer, tehát a 130 bőven ezalatt van. Ha egy ettől is nagyobb házat, vagy egy nagyobb létesítményt (pl. társasházat) szeretnénk kezelni, abban az esetben érdemes egy újabb 750-652-es modult beszerezni és megosztani a fényforrásokat külön DMX-univerzumokra. Ha lehet, próbáljuk meg a csatornák számát 24-gyel oszthatóvá tenni, mivel a 750-652-es modul vételi és átviteli puffere is 24 byte, így 96 csatorna esetén csak 4 blokkot kell küldjön a modul, de 97 esetén már 5-öt kell.

Ha egy, a fenti példától kisebb, átlagos családi házat veszünk, kevesebb helyiséggel, kevesebb LED-szalaggal és fényforrással, akkor kényelmesen el fogja tudni kezelni egyetlen modul is a világításvezérlést.

3.1.3. PRG_IO

Az általunk létrehozott tesztkörnyezet tartalmaz egy manuális kapcsolót is, amelyik kapcsoló engedélyezi vagy blokkolja a teljes világítást. Ez igaz mind a szoftveres, mind a hardveres oldalra is, tehát blokkolás esetén a szoftver *blackout = TRUE* értéket küld, hardveresen pedig tiltjuk azt a kimenetet, amelyik a dekóderek tápellátását biztosítja egy relén keresztül. A relét kímélendő, célszerű hiszterézist alkalmazni a kapcsolások között, így a kapcsolók pergésmentesítéséről is tudunk gondoskodni, illetve a véletlen kapcsolások ellen is tudunk védekezni. A Wago PFC200-as sorozata egy sor, felhasználó által programozható LED-et is biztosít számunkra a PLC előlapján, így némiképp

kiegészítve a hiszterézist kezelő algoritmust, a meghatározott LED-en keresztül figyelmeztetni is tudjuk a felhasználót, hogy a világításvezérlés hardveres engedélyezésében változás fog bekövetkezni. Mi a 7-es számú LED-et használjuk erre. A hiszterézist kezelő SFC-diagram a *6. ábra* követhető:



6. ábra. *Az FB_ChangingStatePOS funkcióblokk SFC ütemdiagramja*

Maga a *PRG_IO* az *FB_ChangingStatePOS* funkcióblokkunkon keresztül kezeli a meghatározott digitális bemenetet, amelyet a *Clock* bemenetre kell kötnünk. További két bemenet a *DelayON* és a *DelayOFF*, amelyek a ki- és bekapcsolások közti időintervallumokat adják meg. A *funkcióblokk* két stabil állapotot kezel, ezek az *Init* és *ON* lépések. A kettő között átmeneti lépések vannak, amelyek aktiválódásakor a *funkcióblokk ChangeInProgress* kimeneti változója kapcsol magas jelszintre. Ez a kimeneti változó teszi lehetővé, hogy a változás alatt valamilyen figyelmeztetést adjunk a felhasználó számára, vagy valamilyen előkészítő állapotba tegyük valamely hardver elemet. Mi a 7-es számú LED-en keresztül küldünk figyelmeztetést gyors, narancssárga/zöld alternáló fénysorozattal. Ha ennek hatására a digitális bemenet állapota visszaáll az eredeti alacsony vagy magas jelszintre, úgy a villogás abbamarad, és a *funkcióblokk* visszalépteti az előző stabil állapotra a vezérlést (Ajtonyi és Gyuricza, 2002).

3.1.4. PRG_VisuHelper

A következő fejezetben bemutatásra kerül majd a HMI, viszont a HMI adatait feldolgozó *program* strukturálisan még ebbe a fejezetbe tartozik, így működését itt mutatjuk be. A LED-es izzók dimmelése nem szabályozható a teljes tartományon, mivel egy bizonyos tartomány alatt egyszerűen kikapcsol. A határteljesítmény keresése közben pedig vibrálást tapasztalhatunk, ami fárasztó és kellemetlen is a szemnek. Ezért a teljes szabályozási tartományt csatornánként célszerű rögzíteni. A *standard függvények* között találjuk a *Limit* függvényt is, amely funkciója éppen ezt a szerepet tölti be. Így a *PRG_VisuHelper program* egyik funkciója ezen limitek biztosítása, így elkerülhető, hogy akaratunk ellenére valamely csatorna kikapcsoljon.

A másik funkciója a 0–255 értelmezési tartomány 0–100-ig tartó, százalékos tartományra konvertálása, így hiába egyetlen potmétert tekerünk, a csatornára kiküldött teljesítmény százalékosan is megjelenik. Fontos, hogy a HMI- és a DMX-fényerőszabályozás is lineáris szabályozást alkalmaz. Ha logaritmikus szabályozást szeretnénk használni, azt a PLC-ben kell majd lekezelnünk.

3.2. Ütemező taszkok

Az előző pontban bemutatott *programok* ütemezését a *taszkok* látják el. A *2. ábra* jól mutatja, hogy a négy különálló *programunkat* 3 db *taszk* ütemezi. Az IO-k kezelését a *Task_IO,* a DMX kezelését a *Task_DMX,* és a vizualizációt pedig a *VISU_TASK*. Ez utóbbi kezeli mind a vizualizációt, mind pedig a *PRG_VisuHelper programot.* Ütemezésüket tekintve a leggyorsabb az IO-kezelés 10 ms ciklusidővel, és 10-es prioritással. Őt követi 50 ms ciklusidővel a DMX üzenetek küldése 14-es prioritással, legvégül pedig a vizualizáció van, szintén 50 ms ciklusidővel, de már csak 15-ös prioritással.

4. HMI

Az egyes csatornák értékét legegyszerűbben egy vizuális felületen keresztül tudjuk megjeleníteni és állítani is. Az általunk létrehozott felület a *7. ábra* tekinthető meg. Mint látható, 4 db diszkrét csatornát tudunk kezelni vele, hiszen a tesztrendszerben 2 db, kétcsatornás DMX-dimmer dolgozik. Felül helyezkedik el a csatorna száma, alatta pedig a csatornán kiküldött aktuális teljesítmény százalékosan megjelenítve.

Az első beavatkozónk egy potméter, amellyel állítani tudjuk a kimenet fényerejét. Hogy milyen tartományban, azt a potméter két oldalán található *MIN* és *MAX* feliratú szövegboxokban tudjuk limitálni. Így a minimumteljesítmény rögzíthető, de akár a maximális tartomány is korlátozható.

A következő beavatkozónk a csatorna ki-be kapcsolását lehetővé tevő kétállású kapcsoló. Kikapcsolt állapotában a csatorna inaktív, a potméter bármilyen állítása sincs hatással a fényerőre. Amennyiben a csatorna engedélyezett, úgy a fényerő és a teljesítményszint is változik valós időben a potméter mozdításának hatására.

Az alsó sorban a hardveres és a szoftveres *blackout* állapotáról kapunk információt, illetve tudjuk állítani azt. Az alul, bal oldalt elhelyezkedő, zöld színű lámpa világít, ha a hardveres kapcsoló nem tiltja a világítás működését, és a dekóderek áram alatt vannak. Ha nem világít, abban az esetben nincs áram alatt a rendszer, így szerelést is lehet végezni rajta – bár biztonsági relék és biztonsági PLC-k híján ez inkább csak elméleti biztonságot jelent, célszerűen egy manuális beavatkozót is be kell iktatni, pl. egy manuális kismegszakítót, a szabályos villamos bekötéshez.

Tőle jobbra található a szoftveres *blackout* kapcsolója. Ez a kapcsoló áram alatt tartja a dekódereket, viszont mindegyik lámpa fényét leoltja. Ebben az esetben a 7-es számú LED lassan, zöld színnel villog – jelezvén, hogy a rendszer működik, csak a csatornák blokkolt állapotban vannak (codesys.com).

7. ábra. *A HMI-felület*

5. Összefoglalás

A cikkünkben bemutattuk, hogy a tesztrendszerünk 4 csatornáját a két dekóderen hogyan és milyen módszerekkel tudjuk a PLC-n keresztül vezérelni. Bemutattuk azt a könyvtárt, amelyik a DMX-kommunikációt megvalósítja, megismerkedtünk az ezekhez szükséges *funkcióblokkokkal* is, és be is mutattuk működését egy saját készítésű HMI-n keresztül. A HMI azonban csak egyetlen módszer, amivel a kimenetek manipulálhatók, hiszen innentől kezdve egy impulzuskapcsolóval, egy hálózati üzenettel, egy mikrovezérlő által küldött paranccsal, ha már meg tudjuk szólítani a PLC-t, hogy módosítsa valamely csatornát, akkor a PLC ezt végre fogja tudni hajtani. Okosotthonbéli alkalmazhatóságát ekképpen igazoltnak véljük.

Irodalom

[1] Ajtonyi, I. (2007). *PLC és SCADA-HMI rendszerek.* AUT-Info Kft. Miskolc.

[2] *WAGO-I/O-SYSTEM 750 – WagoAppDMX & WagoAppColorConverter RGB Control via DMX* (2021). wago.com

[3] Ajtonyi I., Gyuricza I. (2002). *Programozható irányítóberendezések, hálózatok és rendszerek*. Műszaki Könyvkiadó Kft., Budapest. ISBN 963-16-1897-8

[4] *CODESYS Visualization –Development of HMI Screens directly in the IEC 61131-3 Development Environment*. (2018). codesys.com