

# Strukturált szöveg

Ipari irányítástechnika

KOVÁCS Gábor

[gkovacs@iit.bme.hu](mailto:gkovacs@iit.bme.hu)

# Strukturált szöveg

## *(Structured Text, ST)*

- Magas szintű szöveges nyelv
- Világos felépítés
- Hatékony programszervezési módok
- A gépi kódra fordítás nem tartható kézben közvetlenül
- A magas absztrakciós szint szuboptimális implementációhoz vezethet



**A teljes szöveges kód végrehajtott ciklusonként**

# Kifejezések (*expression*)

- A műveletek a kifejezések eredményét dolgozzák fel
- Egy kifejezés elemei
  - Operandusok (literálisok, változók, akár más kifejezések)
  - Operátorok

# Operandusok

- Literálisok

`17, 'my string', T#3s`

- Változók (elemiek vagy származtatottak)


`Var1, VarArray[12]`

- Függvények visszatérési értékei

`Add(2, 3), sin(1.76)`

- Más kifejezések

`10+20 (=Add(10, 20))`

Operátor	Leírás	Példa → Eredmény	Prioritás
( )	Zárójel: végrehajtási sorrend	$(3+2) * (4+1) \rightarrow 25$	
<fcn name>	Függvényhívás	<code>CONCAT('PL','C') → 'PLC'</code>	
–	Ellentett (aritmetikai)	$-10 \rightarrow -10$	
NOT	Komplement (logikai negálás)	<code>NOT TRUE → FALSE</code>	
**	Hatványozás	$2**7 \rightarrow 128$	
*	Szorzás	$2*7 \rightarrow 14$	
/	Osztás	$30/6 \rightarrow 5$	
MOD	Maradékképzés (modulo)	$32 \text{ MOD } 6 \rightarrow 2$	
+	Összeadás	$32+6 \rightarrow 38$	
–	Kivonás	$32-6 \rightarrow 26$	
<, <=, >, >=	Összehasonlítás	$32<6 \rightarrow \text{FALSE}$	
=	Egyenlőség	<code>T#24h = T#1d → TRUE</code>	
<>	Egyenlőtlenség	$2<>5 \rightarrow \text{TRUE}$	
&, AND	Logikai ÉS	<code>TRUE AND FALSE → FALSE</code>	
XOR	Logikai kizáró vagy (XOR)	<code>TRUE XOR FALSE → TRUE</code>	
OR	Logikai VAGY	<code>TRUE OR FALSE → TRUE</code>	

# Függvényhívások

- A függvényhívás valójában kifejezés: a kifejezés értéke a függvény visszatérési értéke
- Formális hívás
  - Paraméterek azonosítókhoz kötve, tetszőleges sorrendben
  - Elhagyott paraméter esetén a függvény annak kezdeti értékét használja
  - A függvény kimeneti változói más változókhoz rendelhetők
  - `LIMIT (EN:=1, MN:=0, MX:=10, IN:=7, ENO=>VarOut)`
- Informális hívás
  - Paraméterek a függvény deklarációjában meghatározott sorrendben, bemeneti változók nem hagyhatók el
  - A függvény kimeneti változói nem elérhetők
  - `LIMIT (0, 7, 10)`

# Műveletek (*statement*)

Kulcsszó	Művelettípus
<code>:</code>	Értékkadás
<code>&lt;FB name&gt; (parameters)</code>	FB hívás
RETURN	Visszatérés a hívó POU-ba
IF	Kiválasztás
CASE	Kiválasztás
FOR	Iteráció
WHILE	Iteráció
REPEAT	Iteráció
EXIT	Iteráció befejezése

# Értékadás

- `:=` operátor

- Értékadás

- skalár változónak
- tömbnek elemének

- Adattípusok

- a bal és jobb oldal adattípusa kompatibilis
- a típuskonverziós függvények kifejezésként használhatók

```
VAR
```

```
    d: INT;
```

```
    e: ARRAY [0..9] OF INT;
```

```
END_VAR
```

```
d:=4;
```

```
e[3]:=d**2;
```

```
d:=REAL_TO_INT(SIN(2))
```



# Bináris értékadás

- Nem a létradiagramban megszokott logikai függvény (kifejezés), hanem művelet
- Ha az értékadás nem hajtódik végre (pl. `IF`), akkor a változó megőrzi addigi értékét, nem íródik felül

```
IF A  
  THEN B:=1;  
END_IF;
```



# Funkcióblokk-példányok hívása

- Az FB-hívás művelet, kifejezésben nem megengedett
- Formális hívás
  - Paraméterek azonosítókhoz kötve, tetszőleges sorrendben
  - Kimeneti változók más változókhoz rendelhetők
  - Kihagyott paraméterek helyettesítése a hívott FB-ben
    - előző híváskori értékükkel
    - első hívás esetén kezdeti értékükkel
- Informális hívás
  - Közvetlen értékek megfelelő sorrendben
  - Paraméterek nem hagyhatók el

# FB-hívás - példa

```
PROGRAM MyProg
```

```
VAR
```

```
    MyTimer:    TON;
```

```
    A:          BOOL;
```

```
    MyInt :     INT;
```

```
END_VAR
```

```
(*...*)
```

```
MyTimer (PT:=T#1s, IN:=(MyInt=7), Q=>A);
```

```
MyTimer ((MyInt=7), T#1s);
```

```
A:=MyTimer.Q;
```

```
(*...*)
```

```
END_PROGRAM;
```

Formális hívás

Informális hívás

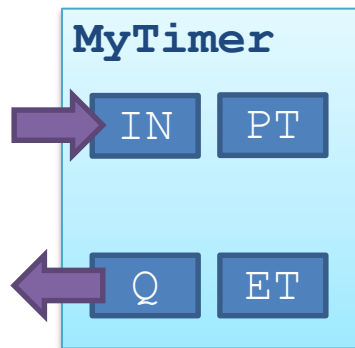
Kimeneti változó értékének más változóhoz  
rendelése informális hívás esetén



# FB-hívás

```
MyTimer.IN:=TRUE;
```

```
A:=MyTimer.Q;
```

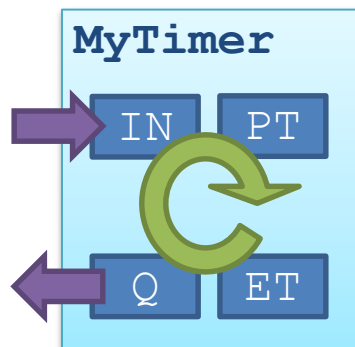


- Az FB-példány változóinak írása és olvasása csak memóriáhozáférést jelent
- Az FB-példány kódrésze nem fut le
- Az FB-példány kimeneti változói nem frissülnek

```
MyTimer.IN:=TRUE;
```

```
MyTimer() ;
```

```
A:=MyTimer.Q;
```



- **Az FB-példány kimeneti változói csak akkor frissülnek, ha az FB-példányt meg is hívjuk!**

*formális hívással (adatmozgatás és FB-hívás egy sorban):*

```
MyTimer (IN:=TRUE, Q=>A) ;
```

Mivel `MyTimer.PT`-nek nem adunk értéket, ezért annak előző hívás során beállított értéke marad érvényes (az FB-példány bemeneti változója megőrzi értékét)

# Kiválasztás

- Kiválasztás logikai (BOOL) értékű kifejezés alapján
- Minden ágban tetszőleges számú műveletből álló blokk állhat
- ELSIF és ELSE ágak elhagyhatók
- Lezárásként END\_IF; használata kötelező

```
IF <BOOL expression> THEN
    <statement block>
ELSIF <BOOL expression> THEN
    <statement block>
ELSIF <BOOL expression> THEN
    <statement block>
ELSE
    <statement block>
END_IF;
```

# Kiválasztás - példa

```
IF  (A=1)  THEN
    X:=1;
    Y:=1;
ELSIF (A=2 OR A=3) THEN
    X:=1;
    Y:=0;
ELSE
    X:=0;
    Y:=0;
END_IF;
```

# Eset-kiválasztás

- Kiválasztás egész (ANY\_INT) vagy felsorolás típusú kifejezés alapján
- Az esetekhez több érték is megadható
- Alapértelmezett eset: ELSE (elhagyható)
- Lezárás: END\_CASE; (nem hagyható el)

```
CASE <INT expression> OF
<value1>:                <statement block>
<value2>,<value3>:      <statement block>
ELSE  <statement block>
END_CASE;
```

# Eset-kiválasztás - példa

```
CASE A OF
    1:      X:=1;
           Y:=1;
    2, 3:   X:=1;
           Y:=0;
ELSE
    X:=0;
    Y:=0;
END_CASE;
```



# Iteráció

**A teljes iteráció egyetlen PLC-cikluson belül hajtódik végre**



- Ha óvatlanul használjuk, akkor rontja a determinizmust és watchdog-hibát is okozhat
- Ne használjuk eseményre való várakozásra
- Használhatjuk
  - tömb vagy adatmező elemeinek vizsgálata
  - egy művelet előre definiált számú ismétlésére

# While hurok

- A `BOOL` típusra kiértékelődő feltételes kifejezést a műveletek végrehajtása előtt vizsgálja
- Akkor hajtja végre a műveleteket, ha a feltételes kifejezés értéke `TRUE`

```
WHILE <BOOL expression> DO  
    <statement block>  
END_WHILE;
```

# While hurok - példa

VAR

MyArray: 1..10 OF INT;

i: INT;

MaxVal: INT:=0;

END\_VAR

(\* ... \*)

i:=1;

WHILE (i<=10) DO

IF (MyArray[i]>MaxVal)

THEN MaxVal:=MyArray[i];

END\_IF;

i:=i+1;

END\_WHILE;

(\* ... \*)

# Repeat – Until hurok

- A `BOOL` típusra kiértékelődő feltételes kifejezést a műveletek végrehajtása után vizsgálja
- A műveleti blokk legalább egyszer végrehajtódik
- Az iterációt a feltételes kifejezés `TRUE` értéke esetén fejezi be

```
REPEAT
```

```
    <statement block>
```

```
UNTIL <BOOL expression>
```

```
END_REPEAT;
```

# Repeat hurok - példa

VAR

MyArray: 1..10 OF INT;

i: INT;

MaxVal: INT:=0;

END\_VAR

(\* ... \*)

i:=0;

REPEAT

i:=i+1;

IF (MyArray[i]>MaxVal)

THEN MaxVal:=MyArray[i];

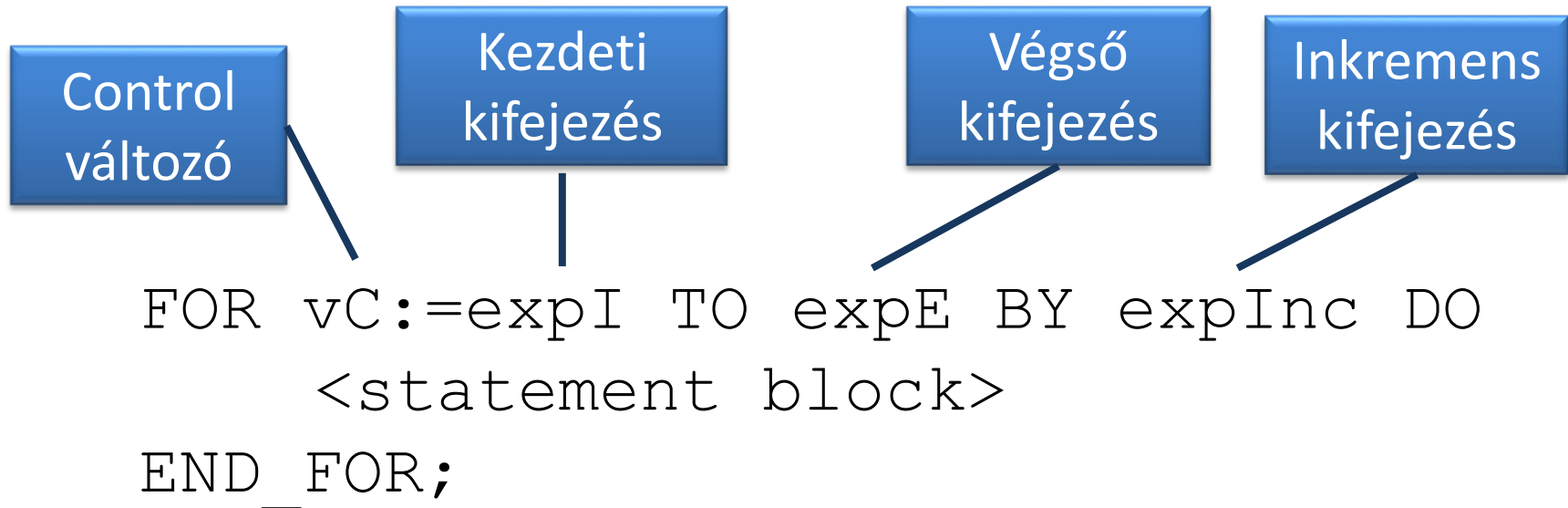
END\_IF;

UNTIL (i=10)

END\_REPEAT;

(\* ... \*)

# For hurok



- A négy változó/kifejezés azonos adattípusú kell, hogy legyen (SINT, INT, DINT)
- A control változónak, valamint a kezdeti és végkifejezésben szereplő változóknak nem adható érték a hurkon belül
- Az inkremens kifejezésben szereplő változónak a hurkon belül is adható érték (nem ajánlott)

# For hurok - példa

```
VAR
```

```
    MyArray:    1..10 OF INT;
```

```
    i:          INT;
```

```
    MaxVal:     INT:=0;
```

```
END_VAR
```

```
( *   ...   * )
```

```
FOR i:=10 TO 1 BY -1 DO
```

```
    IF (MyArray[i]>MaxVal)
```

```
        THEN MaxVal:=MyArray[i];
```

```
    END_IF;
```

```
END_FOR;
```

```
( *   ...   * )
```

# Kilépés hurkokból

- A hurkokból az EXIT művelettel lehet kilépni
- Csak abból a hurokból lép ki, amelyben végrehajtjuk, külsőbb szinten nem hat

```
j := 0;  
WHILE (j < 10) DO  
    i := 0;  
    WHILE (i < 10) DO  
        IF (i = j) THEN EXIT;  
        ELSE i := i + 1;  
        END_IF;  
    END_WHILE;  
    j := j + 1;  
END_WHILE
```

i	j
0	0
1	0
1	1
2	0
2	1
2	2
3	0
...	

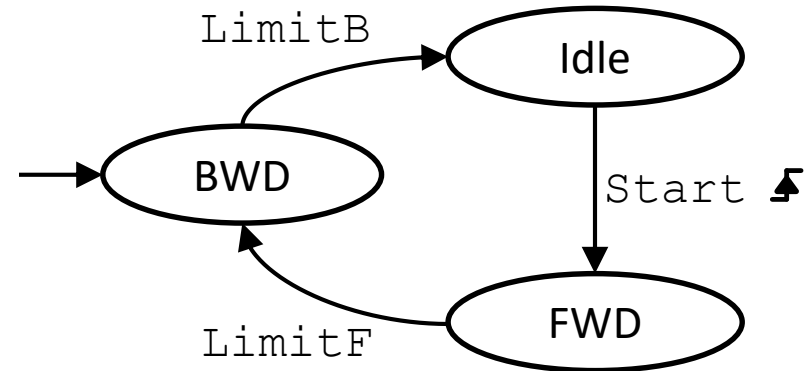


# Visszatérés a hívó POU-ba

- RETURN kulcsszó
- Kiválasztás művelet tetszőleges ágában használható feltételes visszatérésre
- Függvények esetén a visszatérési értéket előbb be kell állítani (függvénynévvel egyező nevű változó)
- Ha hiányzik, az utolsó sor végrehajtása után történik meg a visszatérés

# Állapotgép megvalósítása ST nyelven

- Állapotok reprezentációja: felsorolás típusú változó
- Az állapotok a hozzájuk tartozó címkével kényelmesen kezelhetők
- Kezdeti állapot
  - Alapértelmezésben a felsorolás első (0 értékű) esete
  - Deklaráció során explicit megadható



```
VAR  
    State : (Idle, FWD, BWD) := BWD;  
END_VAR
```

Kezdeti érték explicit beállítása nélkül az Idle lenne a kezdeti állapot

# Állapotgép megvalósítása ST nyelven

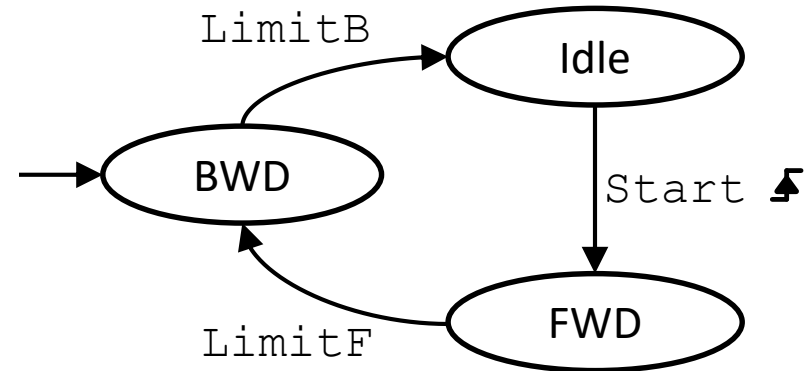
- Átmenetek kiindulási állapotuk szerint egy CASE struktúrában kezelhetők
- Az egyes átmenetek feltételei a megfelelő esetekben vizsgálhatók
- Az állapotváltás során az állapotváltozó értékét frissítjük
- A feltételekhez használt funkcióblokk-példányokat ajánlott a CASE struktúrán kívül minden futáskor meghívni

```
VAR
    State : (Idle, FWD, BWD) := BWD;
    rStart: R_TRIG;
END_VAR

rStart(CLK:=Start);
CASE State OF
    BWD: IF LimitB THEN
        State:=Idle;
    END_IF;
    Idle: IF rStart.Q THEN
        State:=FWD;
    END_IF
    FWD: IF LimitF THEN
        State:=BWD;
    END_IF;
END_CASE;
```

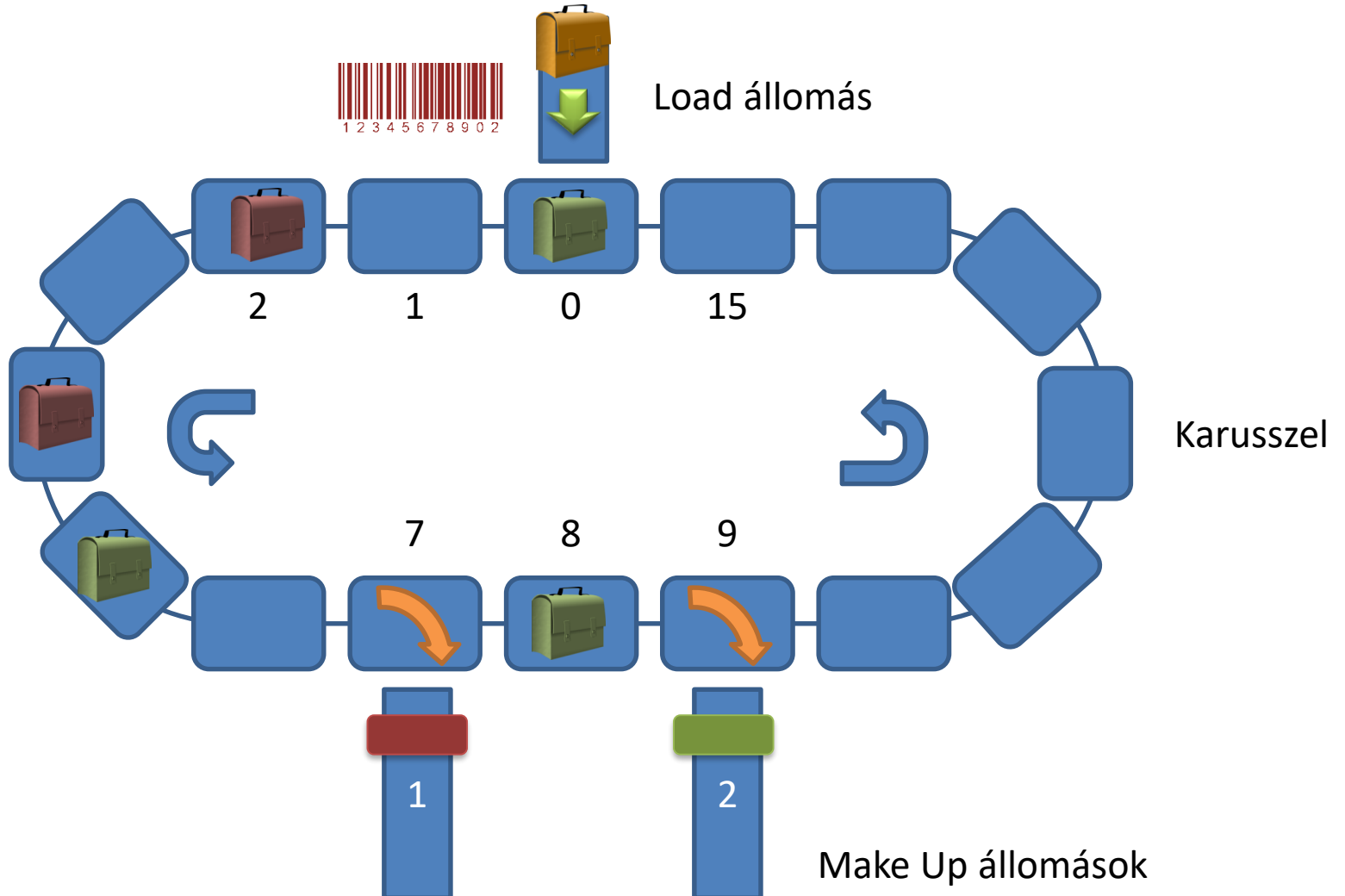
# Állapotgép megvalósítása ST nyelven

- A kimeneti leképezés logikai függvényei egyszerűen megvalósíthatók
- A kimeneti változókhoz egy logikai kifejezés eredményét rendeljük minden ciklusban
- A kimenetek az állapotátmenetek végrehajtása során is beállíthatók, ilyenkor a program módosítása során különös figyelemmel kell eljárni



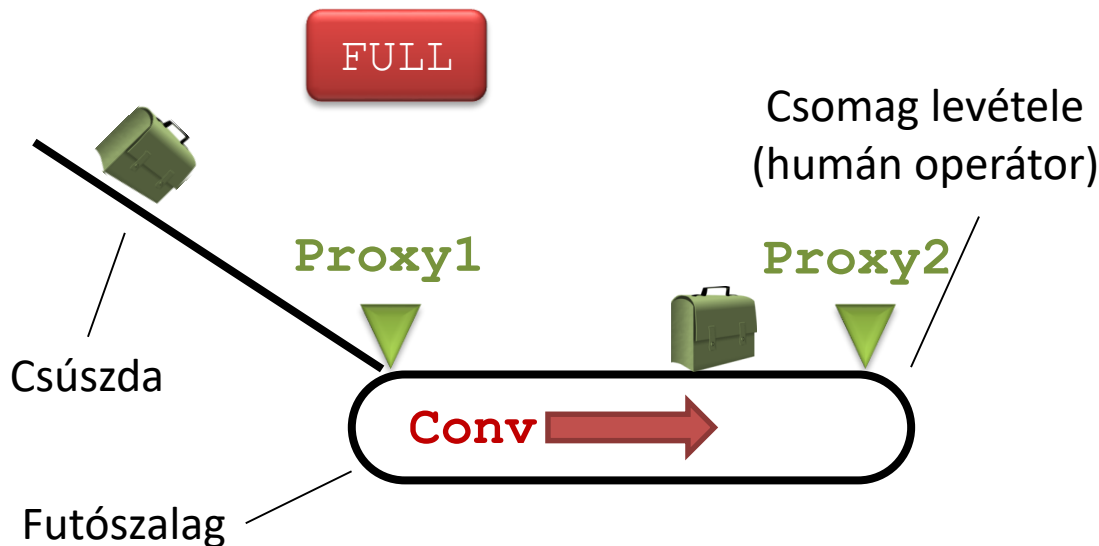
```
Motor:=NOT (State=Idle) ;  
Dir:=(State=FWD) ;
```

# Csomagszállító rendszer



# Make Up állomás

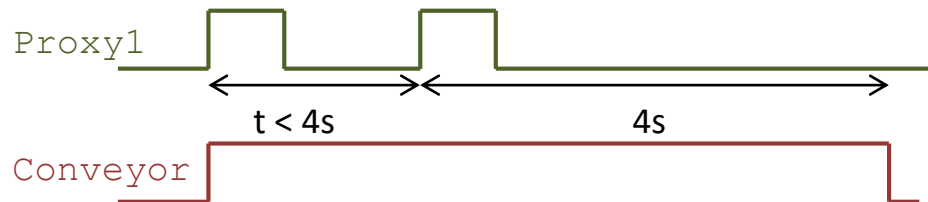
A csomagok a csúszdán érkeznek a szalagra. Az érkező csomagokat a futószalag az állomás másik végére továbbítja, ahol egy humán operátor rakodja át őket a csomagszállító targoncára. A szalag két végén egy-egy érzékelő (Proxy1 és Proxy2) jelzi, ha éppen van ott csomag.



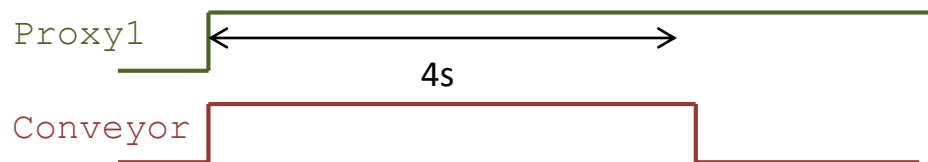
## Specifikáció

- A Proxy1 közelítésérzékelő jelzésére (csúszdáról érkező csomag) működtessük a szalagot 4 másodpercig
- A Proxy2 közelítésérzékelő lefutó élére (szalagról levett csomag) működtessük a szalagot 1 másodpercig
- Ha tele van a szalag, jelezzük a Full kimeneten

# Időzítések a bemeneti oldalon



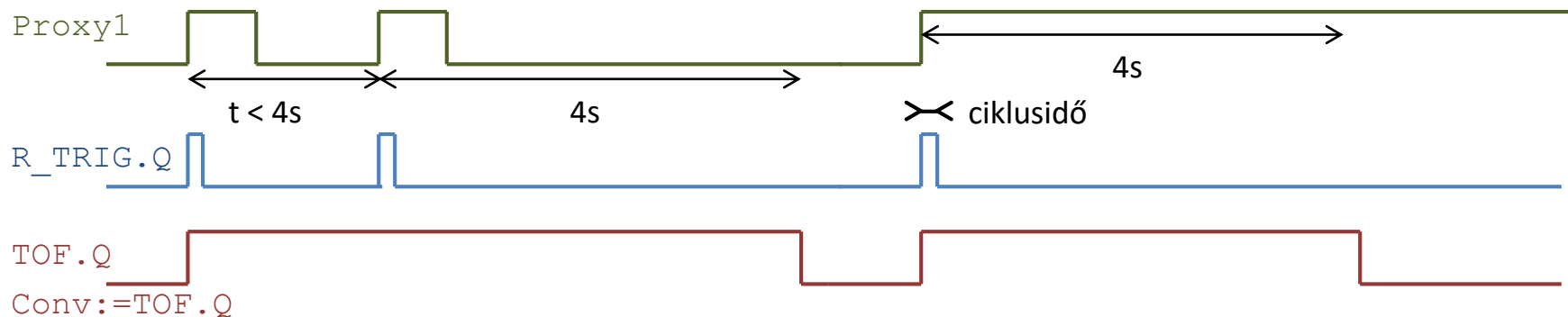
Ha az első csomag érkezése után elindított mozgás közben új csomag, érkezik, akkor a mozgás annak érkezése után 4 másodperccel álljon le



Ha a szalag megtelt, az érkezett csomag nem jut tovább az érzékelőtől. A mozgás ekkor is csak 4 másodpercig tartson.

**Megoldás: R\_TRIG + TOF**

- Az R\_TRIG funkcióblokk a bemenetére kötött közelítésérzékelő felfutó élére 1 PLC-ciklus idejéig aktív kimenetet (R\_TRIG.Q) ad
- R\_TRIG.Q lefutó élét egy TOF időzítővel 4 másodperccel késleltetjük

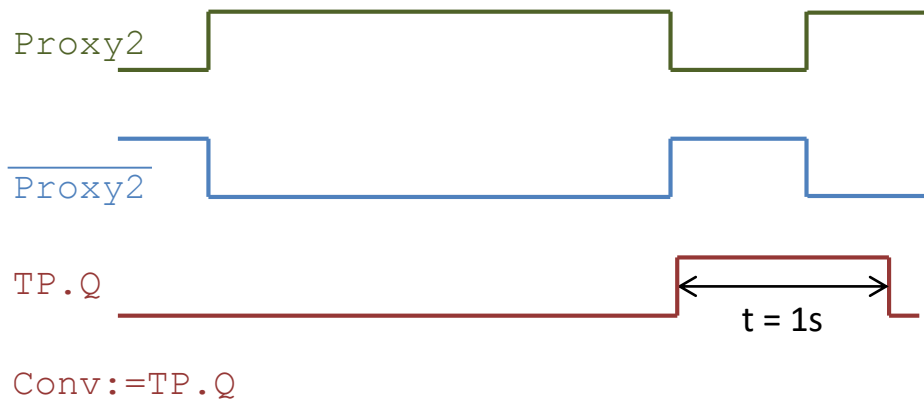


# Időzítések a kimeneti oldalon



Ha elvesznek egy csomagot a kimeneti oldalról (lefutó él), akkor 1 másodpercig működtetni kell a szalagot. Ezalatt az esetleges további jelváltásokat nem kell figyelni.

Megoldás: TP időzítő  $\overline{\text{Proxy2}}$  bemenettel

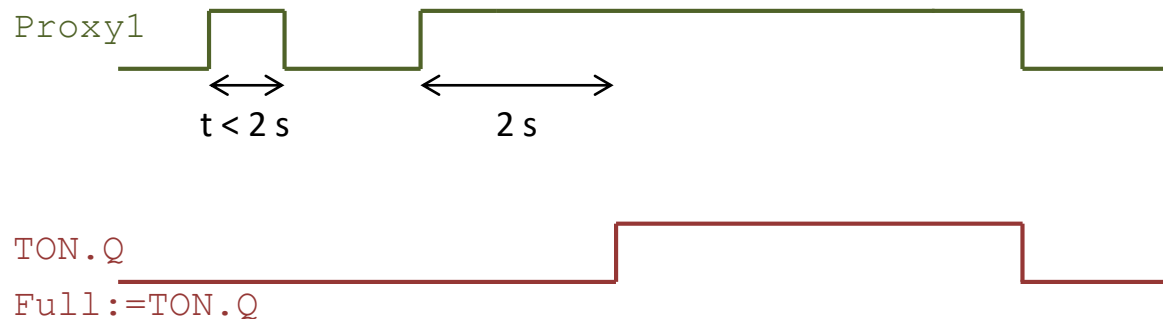


A szalagot mozgatni kell, ha akár a be-, akár a kimeneti időzítő kimenete aktív!



# Teli szalag jelzése

- Mikor van tele a szalag?
- Ha egy csomag érkezik, elindítjuk a szalagot, de a csomag 2 másodpercen belül nem távozik a közelítésérzékelő elől
- Ha a csomag később távozik, akkor már nincs tele a szalag
- Megoldás: TON időzítő



FUNCTION\_BLOCK FBMakeUp

VAR\_INPUT

Proxy1: BOOL;

Proxy2: BOOL;

END\_VAR

VAR\_OUTPUT

Conveyor: BOOL;

Full: BOOL;

END\_VAR

VAR

InEdge: R\_TRIG;

TimerIn: TOF;

TimerOut: TP;

TimerFull: TON;

END\_VAR

InEdge (CLK:=Proxy1, Q=>TimerIn.IN);

TimerIn (PT:=T#4s);

TimerOut (IN:=NOT (Proxy2), PT:=T#1s);

TimerFull (IN:=Proxy1, PT:=T#2s, Q=>Full);

Conveyor:=TimerIn.Q OR TimerOut.Q;

Bemeneti változók: a két közelítésérzékelő

Kimeneti változók: futószalag működtetés és  
tele szalag jelzés

Bemeneti érzékelő felfutó élét érzékelő FB

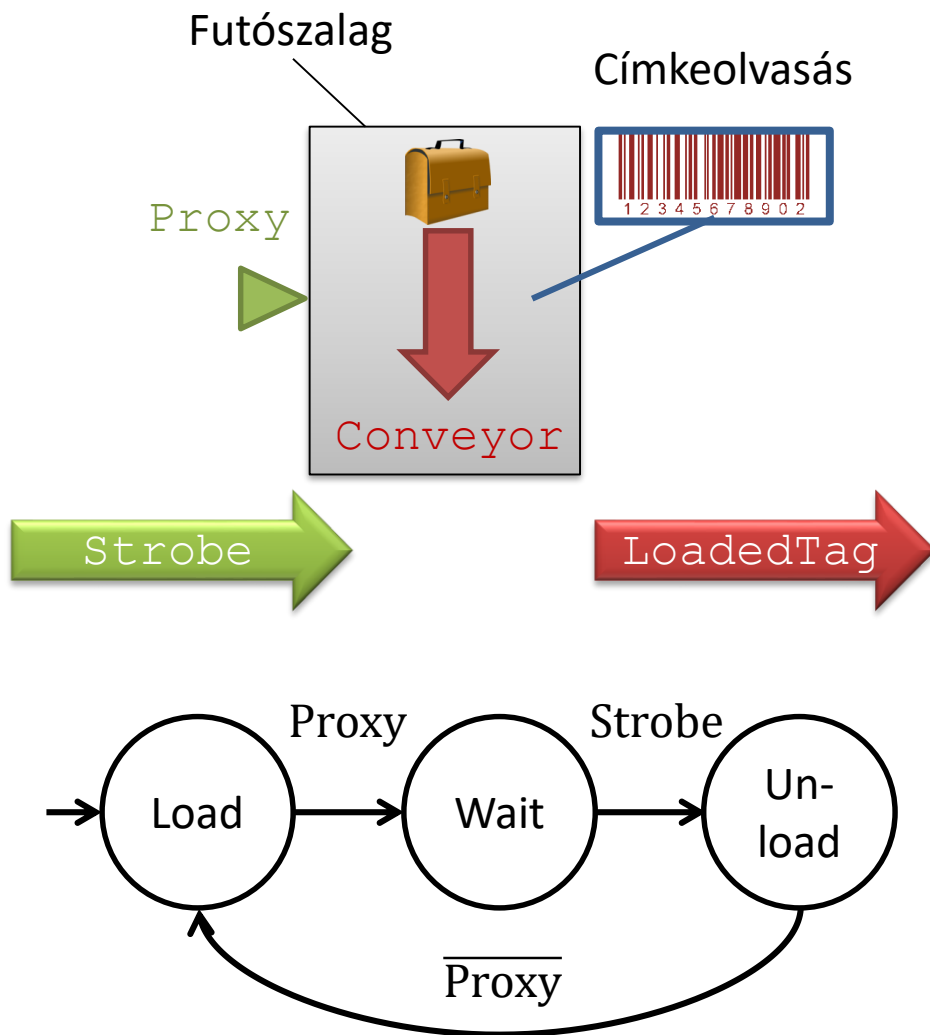
Időzítók példányosítása

Időzítések bejövő és kimenő  
csomag esetén

Teli szalag jelzése

Szalag-kimenet beállítása

# Load állomás



- Indításkor járjon a futószalag
- Ha csomag érkezik, akkor állítsuk meg és olvassuk be a címkét (TagRead() függvény)
- Strobe jel (hívás Strobe=TRUE értékkel): az állomáson tartózkodó csomag továbbítható
  - Ha van csomag, akkor indítsuk el a szalagot és a címkét adjuk ki a LoadedTag kimeneten
  - Ha nincs csomag, akkor a LoadedTag kimenet értéke legyen 0
  - Strobe jel nélküli hívás esetén is 0-ba állítsuk a LoadedTag kimenetet

```

FUNCTION_BLOCK FBLoad
VAR_INPUT
    Strobe: BOOL;
    Proxy: BOOL;
END_VAR
VAR_OUTPUT
    Conveyor: BOOL;
    LoadedTag: UINT;
END_VAR
VAR
    Tag: UINT;
    State: (Load, Wait,
            Unload);
END_VAR

```

Felsorolásként származtatott  
állapotváltozó (adattípusa INT)

Szalag-kimenet  
beállítása

```

CASE State OF
Load:
    Tag:=0;
    IF Proxy THEN
        State:=Wait;
        Tag:=TagRead();
    END_IF;
Wait:
    IF Strobe THEN
        State:=Unload;
    END_IF;
Unload:
    IF NOT Proxy THEN
        State:=Load;
    END_IF;
END_CASE;

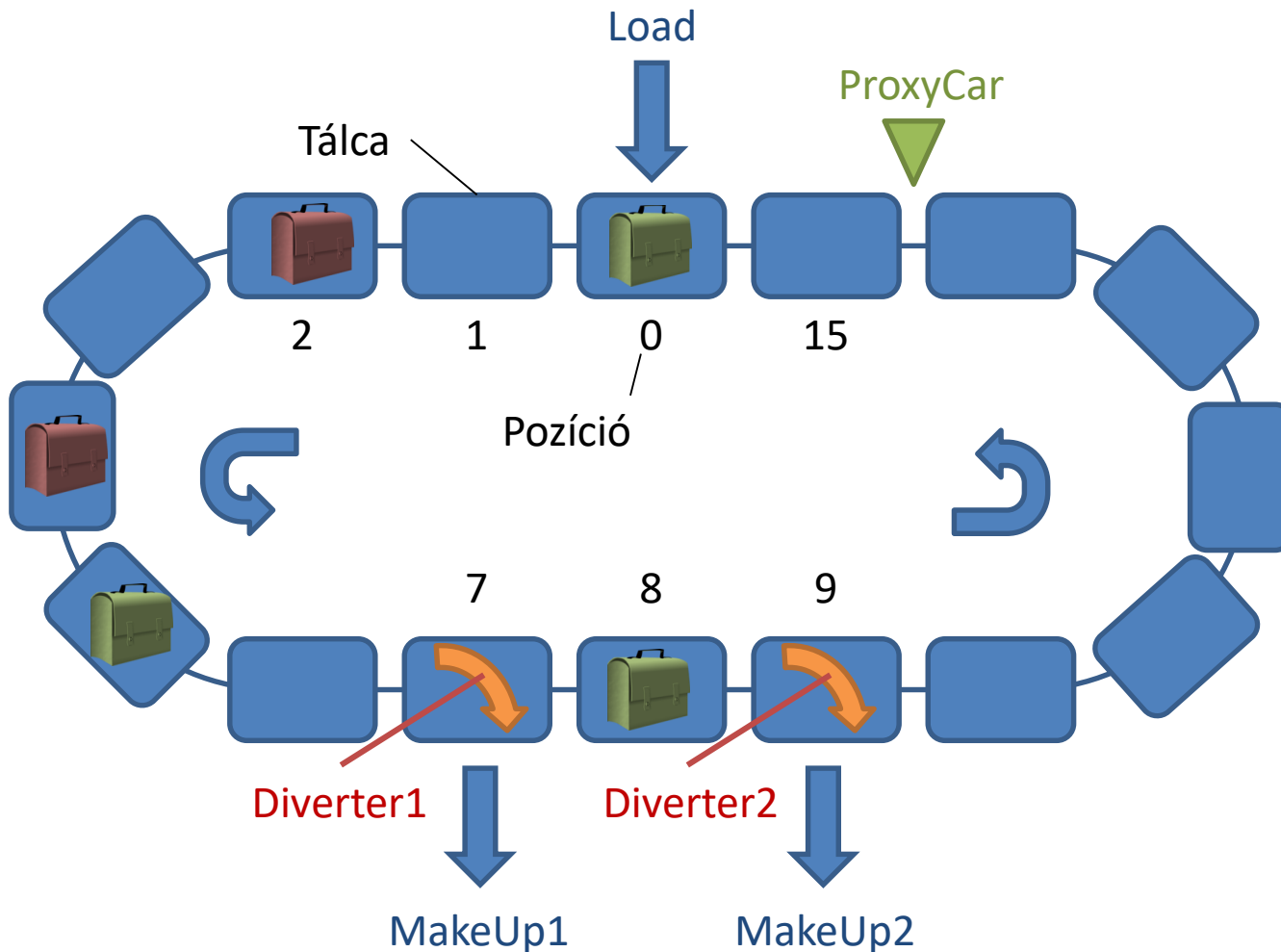
IF Strobe THEN
    LoadedTag:=Tag;
ELSE
    LoadedTag:=0;
END_IF;

Conveyor:=NOT (State=Wait);

```

Állapotgép CASE  
struktúrával

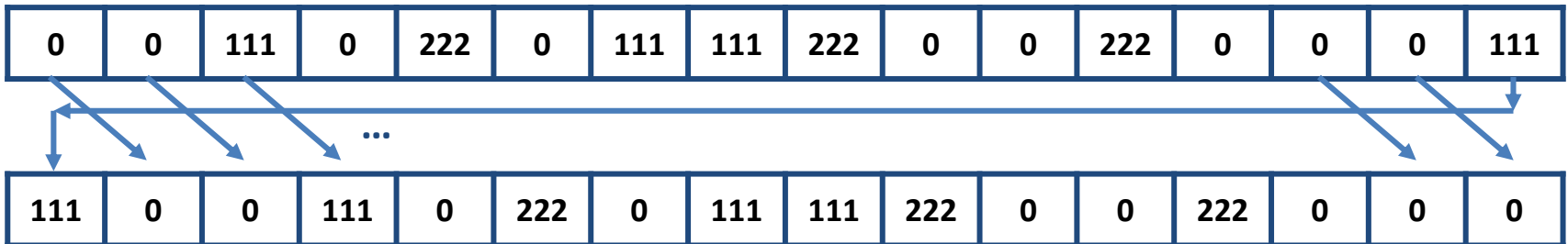
# Főprogram



- A karusszel folyamatosan mozog, tálcáiban a csomagokkal
- A csomagok a Load állomás felől érkeznek
- Csomag csak akkor érkezhets, ha a Load állomásnál lévő (0 pozíció) tálca üres
- A ProxyCar érzékelő felfutó éle jelzi egy tálca elhaladását
- A csomagok a diverterek segítségével boríthatók a make-up állomások csúszdájára
- Teli make-up állomásra tilos csomagot továbbítani
- A csomagokat a címke által jelzett állomásra kell eljuttatni

# Csomagok követése

- Az egyes pozíciókban lévő csomagok címkéit tároljuk egy 16 elemű tömbben
  - 111: „A” make-up állomásra továbbítandó
  - 222: „B” make-up állomásra továbbítandó
  - 0: üres tálca
- Proxy felfutó élére léptessük a tömböt (cirkulárisan): forgatás (ROR)



# Tömb forgatását végző függvény

```
FUNCTION Rotate : ARRAY[0..15] OF UINT
VAR_INPUT
    ArrayIn: ARRAY [0..15] OF UINT;
END_VAR
VAR
    i: INT;
END_VAR

Rotate[0]:=ArrayIn[15];
FOR i:=15 TO 1 BY -1 DO
    Rotate[i]:=ArrayIn[i-1];
END_FOR
```

# Főprogram

- ProxyCar felfutó élére
  - Tömb léptetése
  - Ha a 0 pozícióban a tálca üres, akkor a bemeneti állomás FB hívása `Strobe` jelzéssel Ha a 7. pozícióban 111 címkéjű csomag van és az A make-up állomás nincs tele, akkor a tálca billentése és 0 beírása a 7. pozícióba
  - Ha a 9. pozícióban 222 címkéjű csomag van és a B make-up állomás nincs tele, akkor a tálca billentése és 0 beírása a 9. pozícióba
- Minden ciklusban: állomás-funkcióblokkok hívása



# Főprogram – deklarációs rész

```
PROGRAM PLC_PRG
```

```
VAR_INPUT
```

```
ProxyCar      AT %IX0.0: BOOL; (* Karusszel érzékelő *)
ProxyLoad     AT %IX0.1: BOOL; (* Bemeneti állomás KÉ *)
ProxyA1       AT %IX0.3: BOOL; (* A make-up állomás bemeneti KÉ *)
ProxyA2       AT %IX0.4: BOOL; (* A make-up állomás kimeneti KÉ *)
ProxyB1       AT %IX0.5: BOOL; (* B make-up állomás bemeneti KÉ *)
ProxyB2       AT %IX0.6: BOOL; (* B make-up állomás kimeneti KÉ *)
```

```
END_VAR
```

```
VAR_OUTPUT
```

```
ConvLoad      AT %QX0.0: BOOL; (* Bemeneti állomás futószalag *)
DiverterA     AT %QX0.1: BOOL; (* A make-up állomás diverter *)
ConvA         AT %QX0.2: BOOL; (* A make-up állomás futószalag *)
DiverterB     AT %QX0.3: BOOL; (* B make-up állomás diverter *)
ConvB         AT %QX0.4: BOOL; (* B make-up állomás futószalag *)
```

```
END_VAR
```

```
VAR
```

```
Carousel:    ARRAY [0..15] OF UINT; (* Karusszel-tömb *)
MakeUpA:     FBMakeUp; (* A make-up állomás FB-példánya *)
MakeUpB:     FBMakeUp; (* B make-up állomás FB-példánya *)
Load:        FBLoad;   (* Bemeneti állomás FB-példánya *)
CStep:       R_TRIG;   (* Karusszel léptetésének figyelése *)
```

```
END_VAR
```

```
CStep(CLK:=ProxyCar);
```

ProxyCar felfutó élének figyelése

```
IF CStep.Q THEN
```

Tömb léptetése

```
    Carousel:=Rotate(Carousel);
```

```
    IF (Carousel[7]=111 AND NOT MakeUpA.Full) THEN
```

```
        Diverter1:=TRUE;
```

```
        Carousel[7]:=0;
```

Csomag továbbítása a make-up állomás felé

```
    ELSE
```

```
        Diverter1:=FALSE;
```

```
    END_IF;
```

```
    IF (Carousel[9]=222 AND NOT MakeUpB.Full) THEN
```

```
        Diverter2:=TRUE;
```

```
        Carousel[9]:=0;
```

```
    ELSE
```

```
        Diverter2:=FALSE;
```

```
    END_IF;
```

Új csomag betöltésének engedélyezése (Strobe=TRUE)

```
    IF Carousel[0]=0 THEN
```

```
        Load(Proxy:=ProxyLoad, Strobe:=TRUE,
```

```
        LoadedTag=>Carousel[0], Conveyor=>ConvLoad);
```

```
    END_IF
```

FB hívások minden ciklusban

```
END_IF;
```

```
MakeUpA(Proxy1:=ProxyA1, Proxy2:=ProxyA2, Conveyor=>ConvA);
```

```
MakeUpB(Proxy1:=ProxyB1, Proxy2:=ProxyB2, Conveyor=>ConvB);
```

```
Load(Proxy:=ProxyLoad, Strobe:=FALSE, Conveyor=>ConvLoad);
```

# Alternatív megoldás

```
CStep (CLK:=ProxyCar) ;  
IF CStep.Q THEN
```

```
    Carousel:=Rotate (Carousel) ;  
    Diverter1:=(Carousel[7]=111) AND (NOT MakeUpA.Full) ;  
    Diverter2:=(Carousel[9]=222) AND (NOT MakeUpB.Full) ;  
    carousel[7]:=BOOL_TO_INT (NOT Diverter1) *Carousel[7] ;  
    carousel[9]:=BOOL_TO_INT (NOT Diverter2) *Carousel[9] ;
```

Csomagtovábbítás másképp:  
logikai függvény használata

Tömb elemeinek nullázása: ha a diverter-kimenet 1,  
akkor 0-t (INT), egyébként az eredeti értéket írjuk be

```
    IF Carousel[0]=0 THEN  
        Load (Proxy:=ProxyLoad, Strobe:=TRUE,  
            LoadedTag=>carousel[0], Conveyor=>ConvLoad) ;  
    END_IF  
END_IF ;  
MakeUpA (Proxy1:=ProxyA1, Proxy2:=ProxyA2, Conveyor=>ConvA) ;  
MakeUpB (Proxy1:=ProxyB1, Proxy2:=ProxyB2, Conveyor=>ConvB) ;  
Load (Proxy:=ProxyLoad, Strobe:=FALSE, Conveyor=>ConvLoad) ;
```