TIA Portal in Simatic S7

Vsebina

1 Uvod

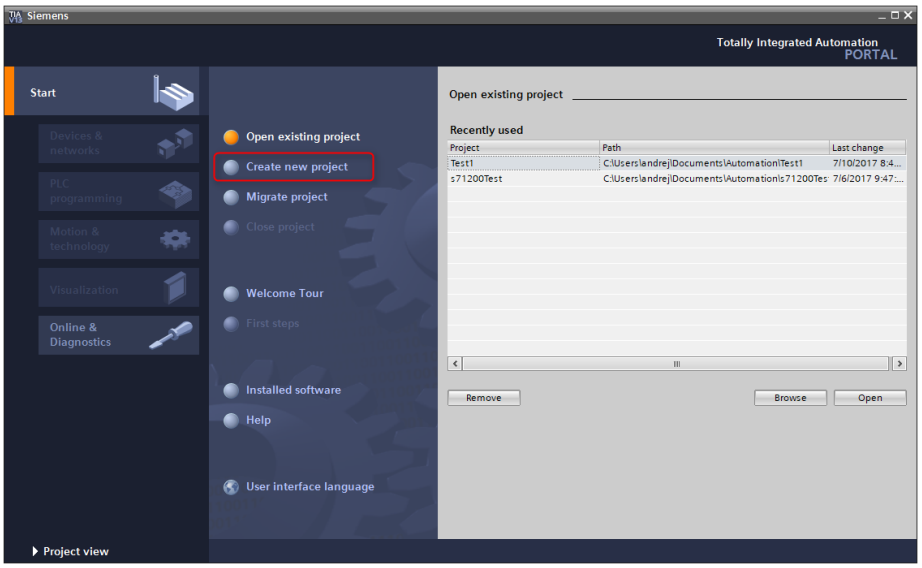
V tej skripti bomo spoznali osnove krmilnika SIMATIC, programiranja LAD, in GRAPH, obenem pa spoznali tudi zaslone na dotik znamke Siemens. Delo bo potekalo s programskim orodjem TIA Portal V13. Ker si bomo pomagali s simulatorjem, ne bomo potrebovali prave opreme, ampak samo PLCSIM in aplikacijo Simens_Simulation_S7.

Torej PLCSIM je virtualni krmilnik, ki nadomesti pravi krmilnik in se izvaja na lokalnem računalniku,

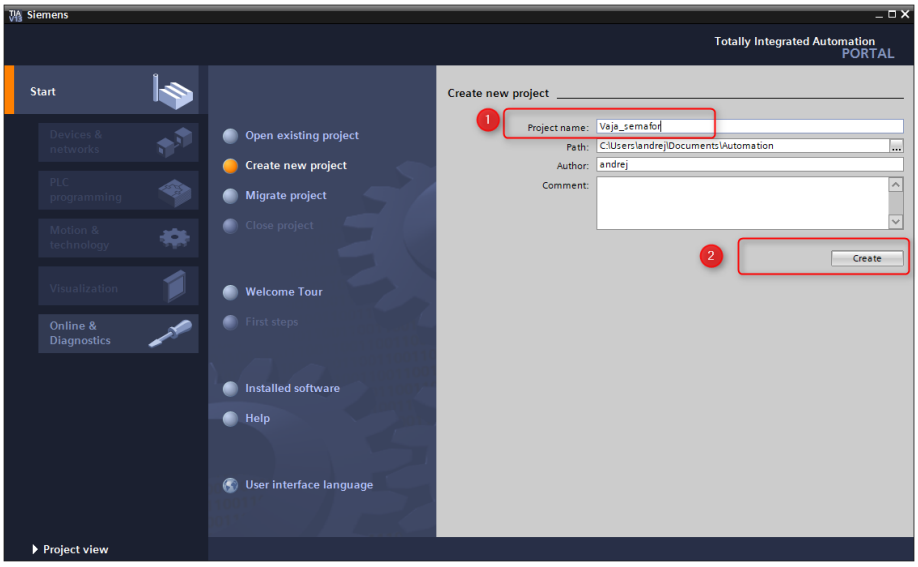
aplikacija Simens_Simulation_S7.exe pa nadomesti napravo oz. aktuatorje in senzorje.

2 Konfiguracija Hardwear-a

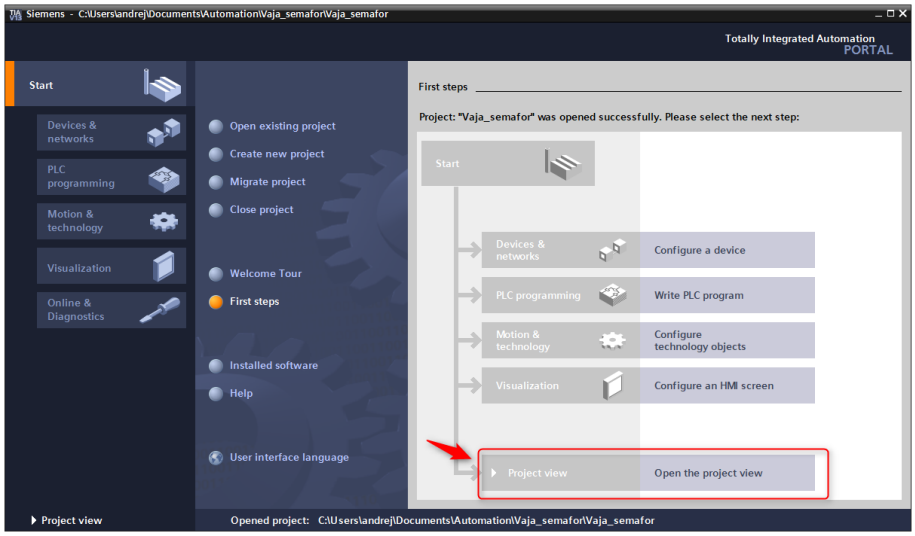
Zaženi TIA Portal.



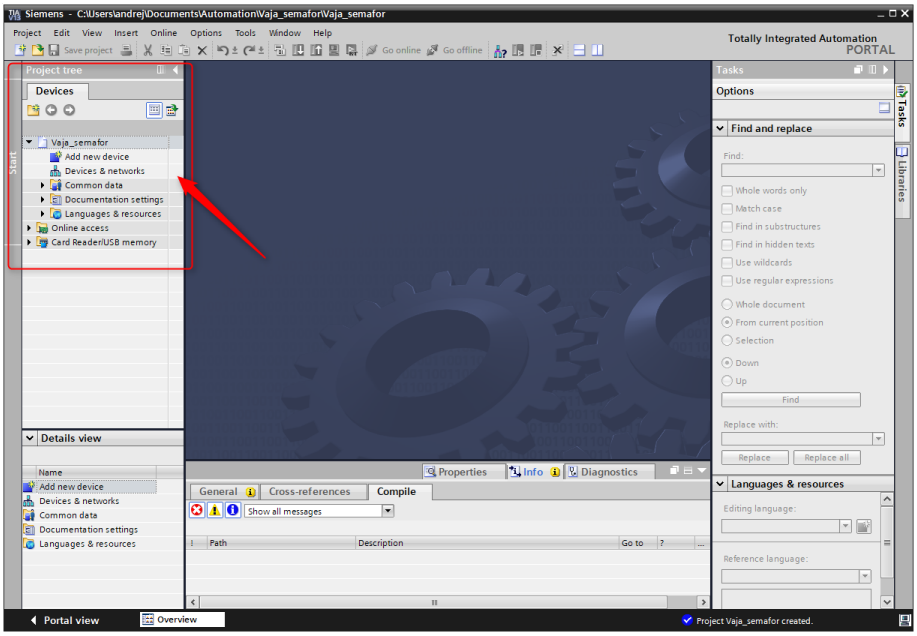
Na desni imamo že obstoječe programe, Mi pa bomo kreirali now projekt => Create new projet.



Nato poimenujemo projekt Vaja_semafor, ter kliknimo Create.

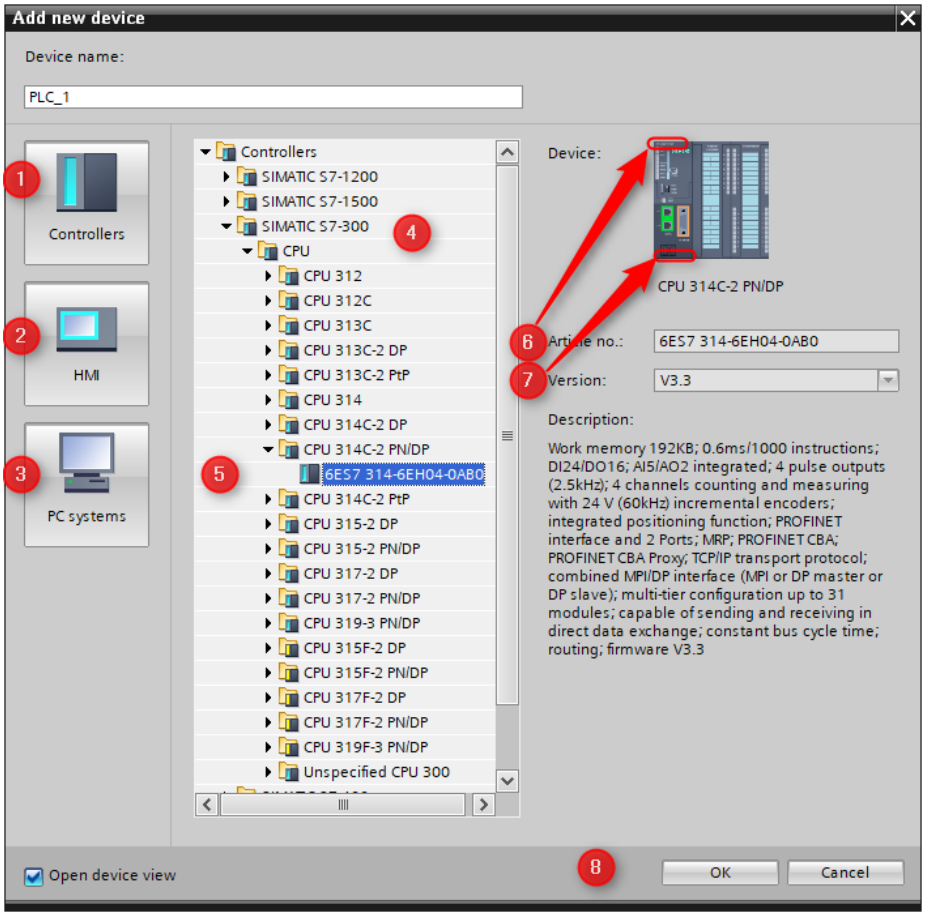


Nato klikni Open the project view



Odpre se projekt, kateri je zaenkrat še prazen, Znotraj projekta so zaenkrat mape, ki se nas ne tičejo.

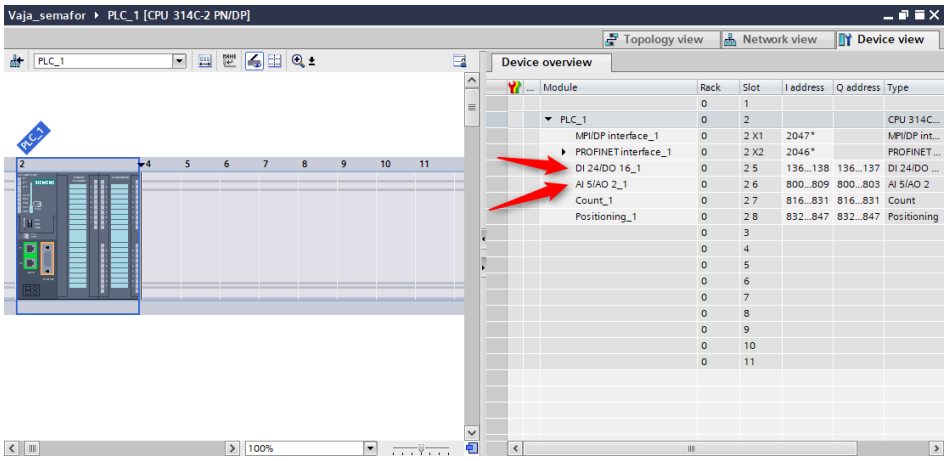
V projekt moramo dodati krmilnik. To storimo tako, da kliknemo na Add new device.



Imamo 3 možnosti

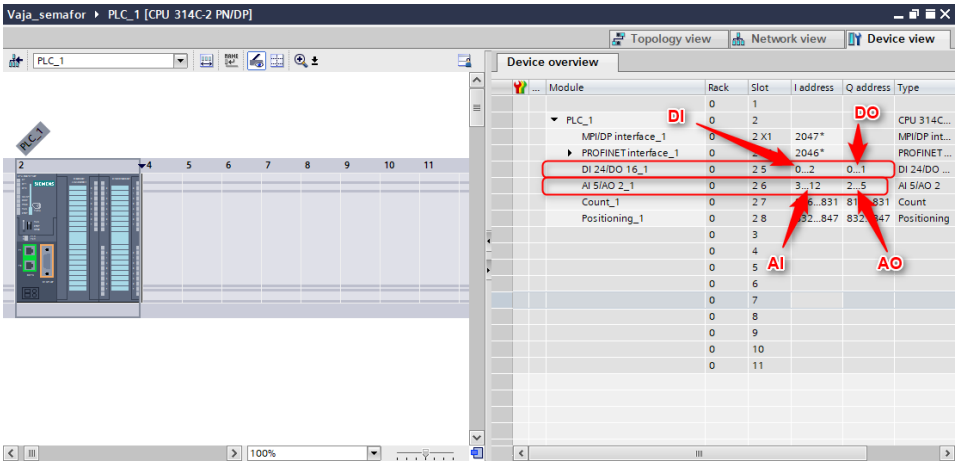
1. V projekt bomo dodali nov krmilnik (v mapi S7-300 izberi CPU 314C-2 PN/DP). Nato preverimo številko naprave (6) in verzijo firmware-a (7).
2. HMI – ko želimo v projekt dodati zaslon na dotik, katerega povežemo s krmilnikom (v kasnejšem poglavju)
3. PC system – ko želimo vključiti industrijski računalnik.

Po pritisku OK, se odpre okno z nastavitvami krminika.



Krmilnik ima 3 bajte vhodov in sicer na naslovih 136, 137, 138, kar pomeni, da imamo na voljo 24 dig. vh.
2 bajta ali 16 dig izhodov, 5 Analognih vhodov in 2 analogna izhoda.

Ker bi radi, da se naslovi začnejo z 0 in ne s 136,=>

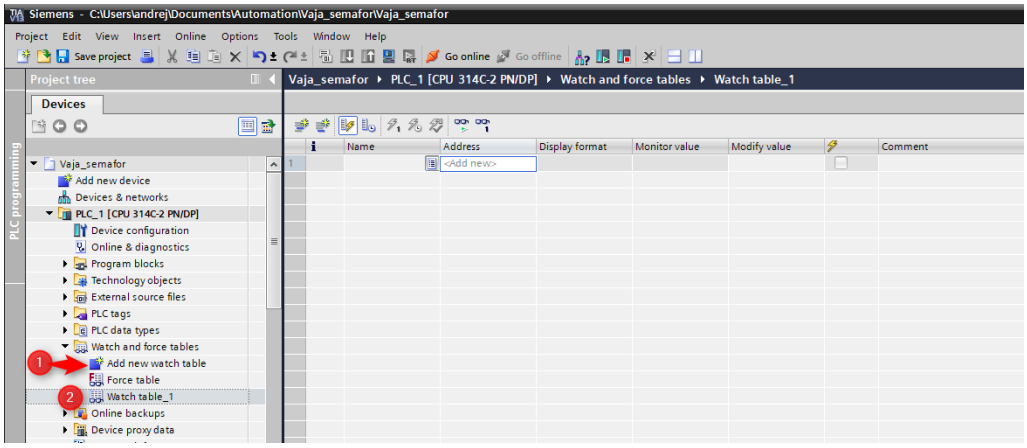


V tem delu lahko spremenimo naslove digitalnih in analognih vhodov in izhodov. Digitalne vh in izh. Postvimo tako kot je prikazano na zgornji sliki. Ker je bilo privzeto za dig. Vh. Začetni bajt 136, bi do prvehga dig. Vhoda prišli z naslovom I136.0 ker smo pa začetni bajt postavili na 0, je ta isti vh naslovljen kot I0.0

Shranimo projekt.

3 Testiranje

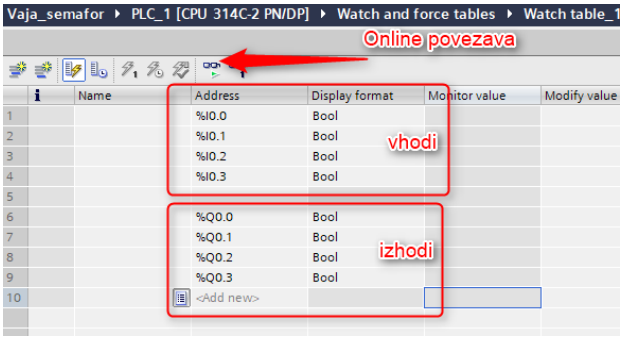
Kako testiramo, če je krmilnik oz. PLCSIM povezan z TIA.



V oknu project tree, poiščimo mapo Watch and force tables, ter klinimo Add new watch table. S tem bomo ustvarili tabelo, v kateri lahko opazujemo vse vh-izh vrednosti, counterje, timerje, spomin….

Ko odpremo novo ustvarjeno tabelo Watch table_1, imamo na desni strani okno, kamor vpisujemo naslove. Način označevanja digitalnih naslovov: (I1.3, Q0.1, M23.6) (I=> input, Q=> output, M=> marker-memory)

Napišimo prvih nekaj naslovov na digitalnem vhodu in izhodu.



Vpišemo nekaj vhodnih in izhodnih naslovov.

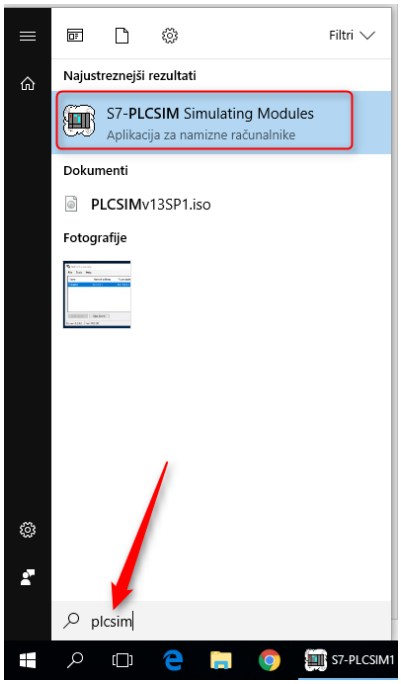
3.1 PLC + naprava

Ko se računalnik poveže na PLC

3.2 PLCSIM + simulator

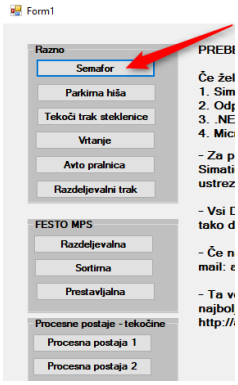
V tem delu je prikazano kako povežemo simulator z virtualnim krmilnikom PLCSIM.

Najprej v start meniju windowsa vpišite ime PLCSIM, ter ga odprite

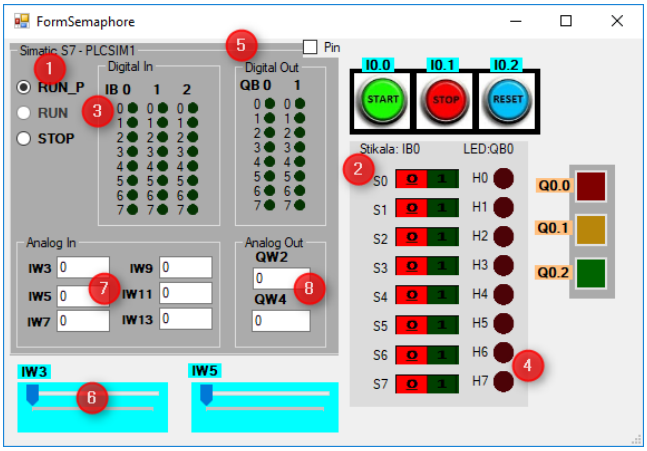


Odpre se virtualni krmilnik, ki bo opravljal vse funkcije tako kot pravi krmilnik, le da bo povezan na simulacijo mašine.

Odpri aplikacijo Simens simulation S7.exe.

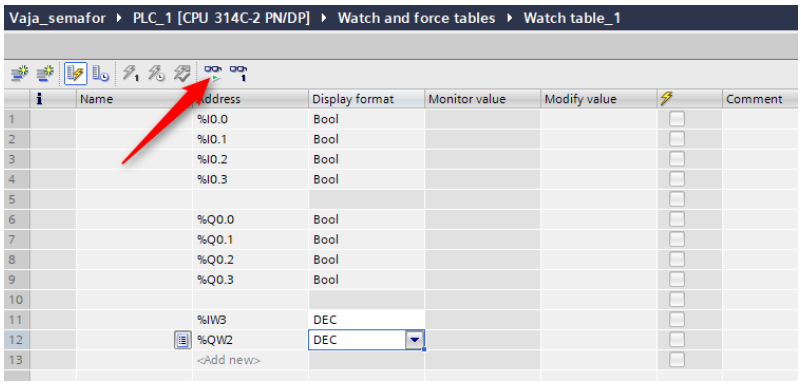


In izberi Semafor.

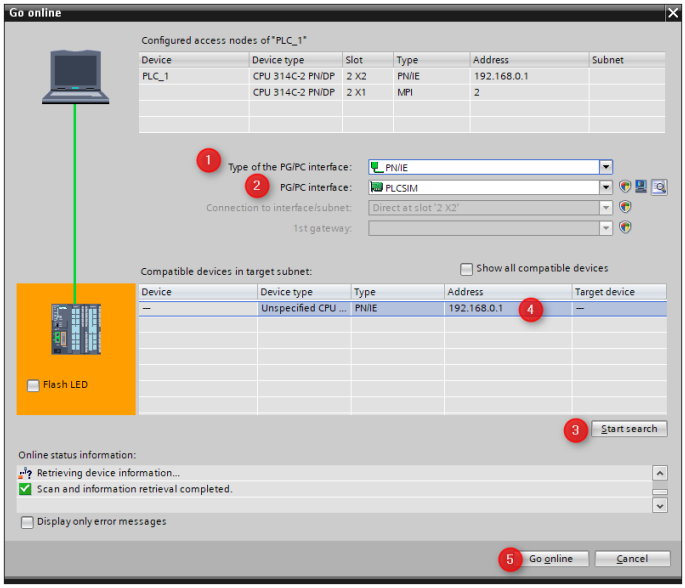


Ko se odpre simulacija, se le ta poveže na PLCSIM, kateri si potem izmenjujeta podatke o vh. In izhodih.

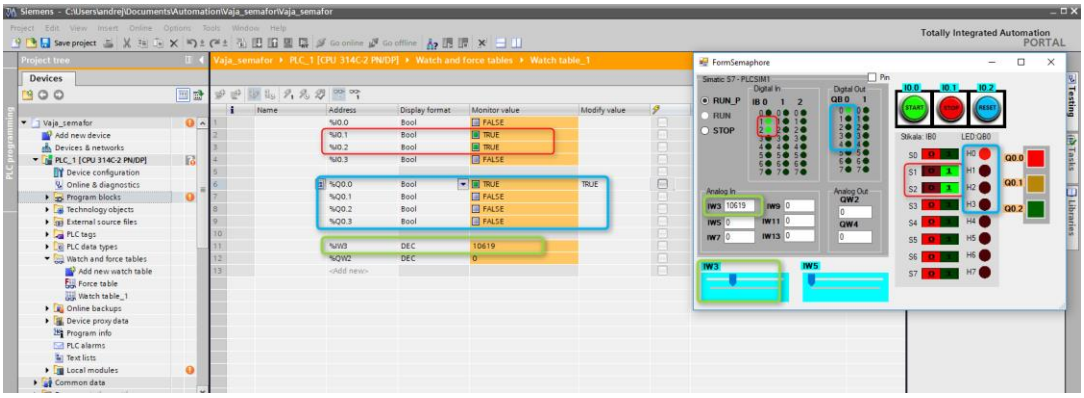
1. Če želimo da se vse skupaj začne, moramo izbrati možnost RUN_P. S tem se začne izmenjava podatkov med simulatorjem in PLCSIM.
2. Simulacija tipk, katera so povezana na prvih 8 digitalnih vhodov (I0.0 – I0.7)
3. Indikacija stanja digitalnih vhodov (tako kot pri pravem krmilniku) na voljo so 3 bajti naslovov od I0.0 – I2.7
4. Prvi bajt digitalnih izhodov (Q0.0 – Q0.7) povezanih na lučke.
5. Indikacija stanja na izhodu krmilnika
6. Slider, ki simulira napetost na prvih dveh AI. Ai0 (IW3), Ai1,(IW5).
7. Prikaz vrednosti na prvih 6 analognih vhodih.
8. Prikaz vrednosti na 2 analognih izhodih
9. Namenske tipke START, STOP, RESET, Naslovi so napisan nad tipkami
10. Semafor, Naslovi izhodov, ki jih vključijo so napisani na levi strani semaforja.



V TIA portalu zopet odpri watch tabelo in klikni na gumb Monitor all.

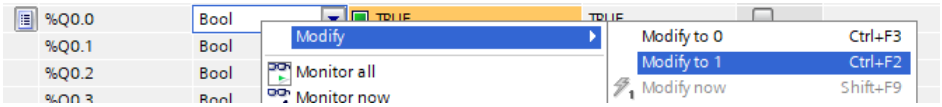


1. Najprej izberemo način komunikacije. Ker naš krmilnik komunicira po PROFINET (TCP/IP) protokolu izberimo možnost PN/IE
2. Izberemo pravi krmilnik ali virtualni krmilnik. Mi imamo PLCSIM
3. Klikni gumb Start search.
4. Ko najde vse možne fizične ali virtualne naprave, izberemo željeno.
5. In se povežemo online.



V simulatorju vključimo stikala (rdeča barva), in vrednosti s1 in s2 se pojavita v tabeli.

Prvi izhod postavimo v logično enico



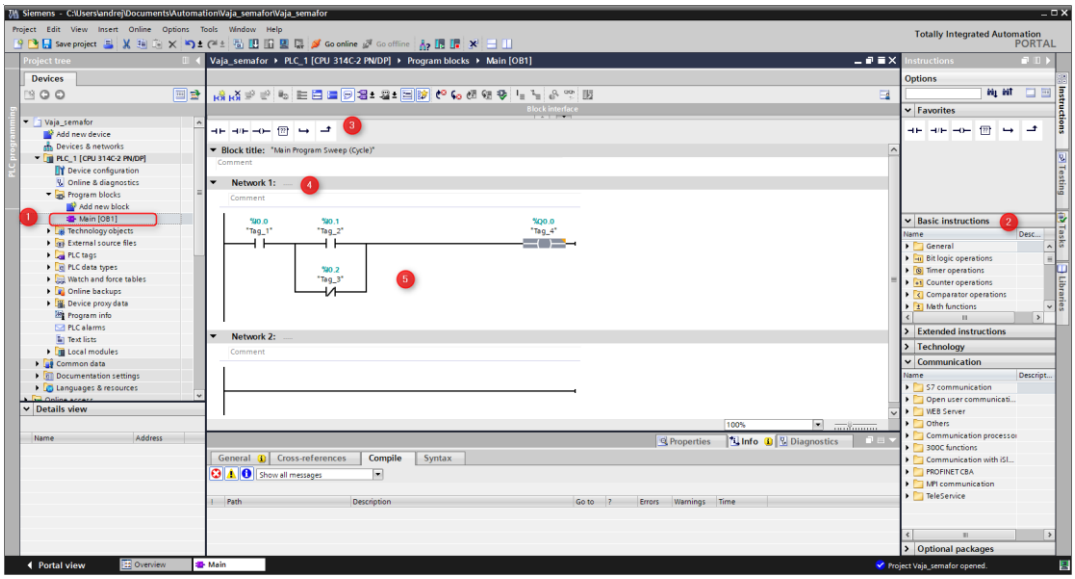
In na simulatorju zagori prvi dig. Izhod oz. prva luč.

Podobna zadeva je z analognimi vrednostmi. Analogni vhod zasede 2 bajta, zato je dolžine word in ga označimo IW# (IW3, IW10, IW400,…)

Analogni izhod označimo QW# (QW2, QW4,…)

Če smo vse do sem uredili, se lahko lotimo pisanja programov.

4 Jeziki



1. Če odpremo mapo program blocks, v njen najdemo vse naše programe, ki smo jih ustvarili. V njen je privzeto tudi Main[OB1]. OB1 je organizacijski blok, in se samo ta izvaja ciklično. (vsa napisana logika se izvede vsakih 10ms). Lahko napišemo tudi druge podprogrami, katere, če želimo da se izvajajo, kličemo iz OB1.
2. Ukazi in programski gradniki za pisanje programa oz. logike.
3. Hitra vrstica, kjer postavimo gradnike, ki jih največkrat potrebujemo
4. Network: Lahko dodamo več Networkov, v katerem napišemo logiko
5. Programska logika v posameznem networku

Vrstni red izvajanja logike:

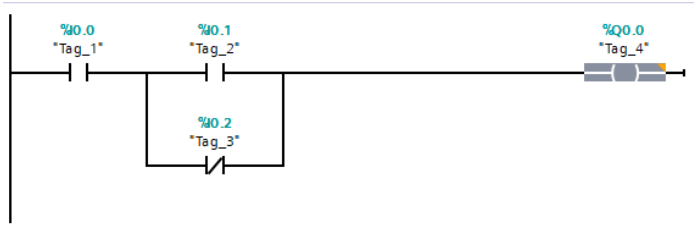
1. Preberejo se vrednosti na digitalnih in analognih vhodih

2. Začne se izvajati Network1
3. Bloki v networku se izvajajo od leve proti desne,
4. Ko se izvade zadnji blok v networku1, se začne izvajati naslednji network (od leve proti desni
5. Ko program obdela zadnji network, se na digitalne in analogne izhode zapišejo vrednosti
6. Ko mine cikel (10ms), se vse skupaj ponovi.

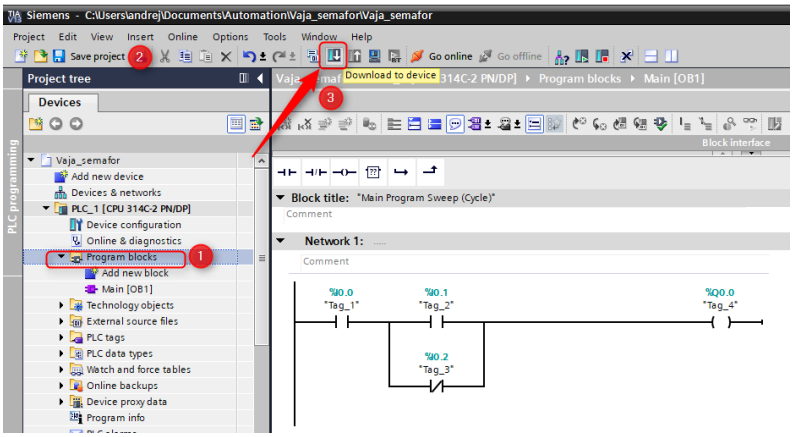
4.1 LAD

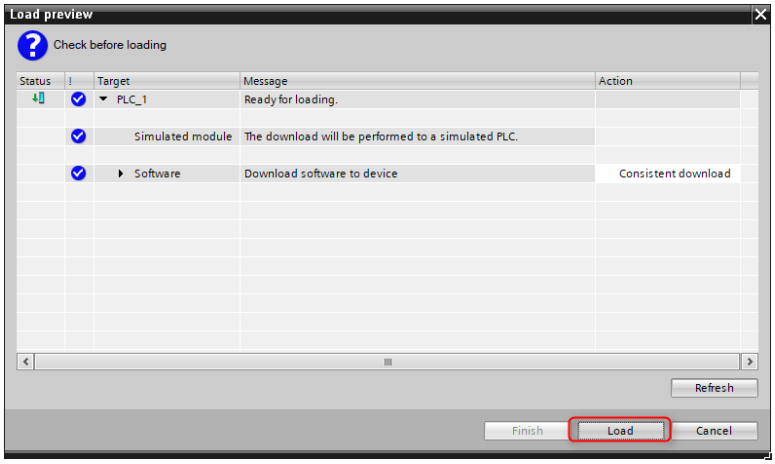
Ali lestvični program. Značilnost tega programa je črta, na katero postavljamo kontakte, bloke in tuljave.

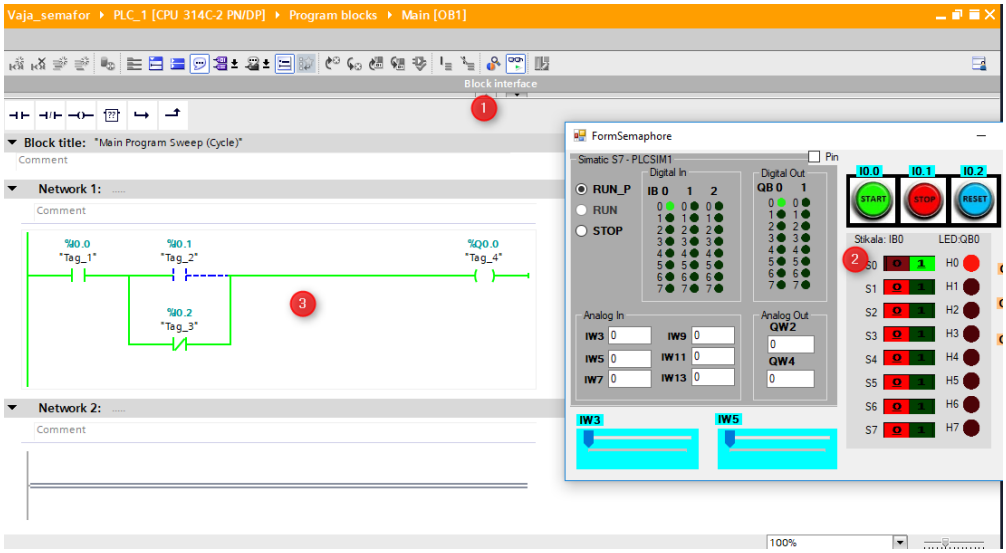
Primer funkcije q0.0 = I0.0 * (I0.1 + !I0.2)



Program preizkusimo tako shranimo program, da kliknemo na..,





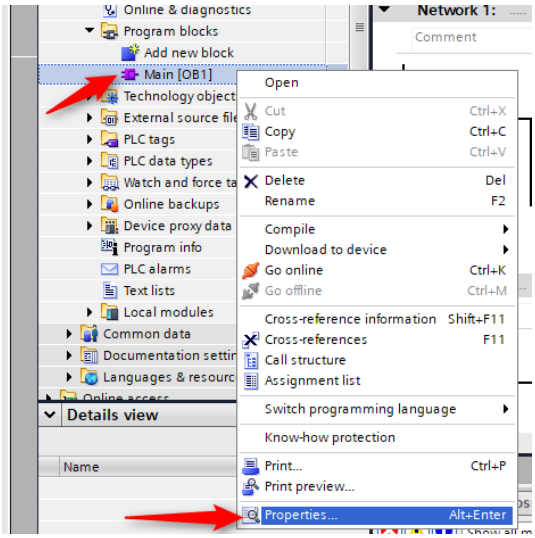


1. Če bomo kliknili na Monitor All, bomo lahko opazovali programsko logiko pri čemer pomeni zelena barva logično enico, ničla pa logično ničlo.
2. Če klikamo na prva 3 stikala na simulatorju, lahko opazujemo programsko logiko.
3. Ob spremembah na digitalnih vhodih, se spreminjajo tudi vrednosti v programu.

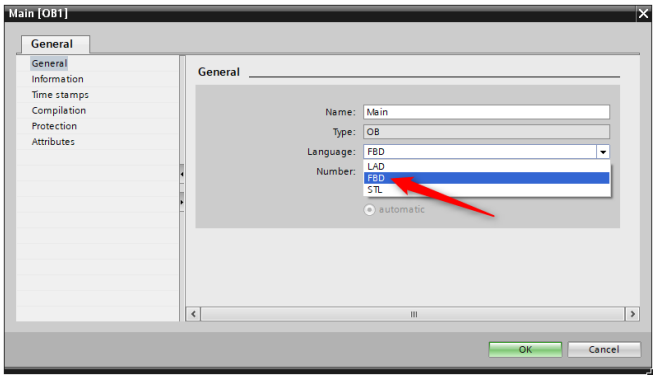
4.2 FBD

Lahko preklopimo pogled tudi v FBD. Značilnost tega pogleda so logična vrata IN, ALI, NE, XOR,…

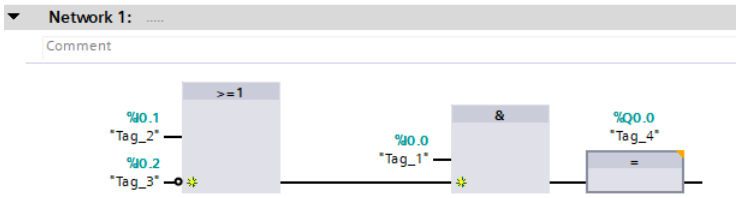
Če spremenimo pogled, isto funkcijo dobimo, desni klik na OB1 in properties:



V pogovornem oknu izberi FBD,

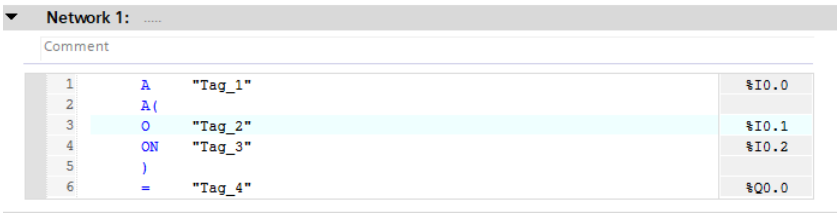


Funkcija, ki jo dobimo v FBD je



4.3 STL

Isto kot za FBD, lahko zamenjamo v STL in dobimo:



A => AND
O => OR
ON => OR NOT
= => zapis vrednosti na izhod

4.4 SCL

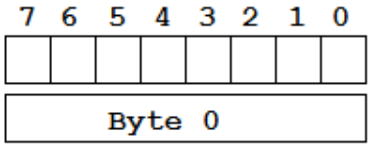
Zelo podoben kakšnemu C jeziku: var, IF, FOR, WHILE,…

5 Naslavljanje

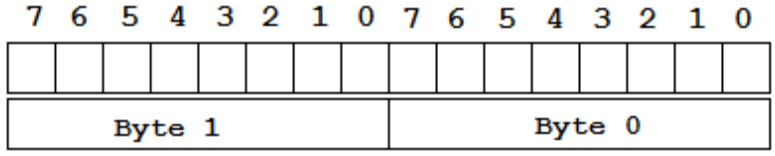
Type and Description	Size in Bits	Format Options	Range and Number Notation (lowest to highest value)_	Example
BOOL(Bit)	1	Boolean text	TRUE/FALSE	TRUE
BYTE (Byte)	8	Hexadecimal number	B#16#0 to B#16#FF	L B#16#10 L byte#16#10
WORD (Word)	16	Binary number Hexadecimal number BCD Decimal number unsigned	2#0 to 2#1111_1111_1111_1111 W#16#0 to W#16#FFFF C#0 to C#999 B#(0.0) to B#(255.255)	L 2#0001_0000_0000_0000 L W#16#1000 L word#16#1000 L C#998 L B#(10,20) L byte#(10,20)
DWORD (Double word)	32	Binary number Hexadecimal number Decimal number unsigned	2#0 to 2#1111_1111_1111_1111 1111_1111_1111_1111 DW#16#0000_0000 to DW#16#FFFF_FFFF B#(0,0,0,0) to B#(255,255,255,255)	2#1000_0001_0001_1000_ 1011_1011_0111_1111 L DW#16#00A2_1234 L dword#16#00A2_1234 L B#(1, 14, 100, 120) L byte#(1,14,100,120)
INT (Integer)	16	Decimal number signed	-32768 to 32767	L 1
DINT (Integer, 32 bits)	32	Decimal number signed	L#-2147483648 to L#2147483647	L L#1
REAL (Floating-point number)	32	IEEE Floating-point number	Upper limit: ±3.402823e+38 Lower limit: ±1.175 495e-38	L 1.234567e+13

5.1 Dolžine

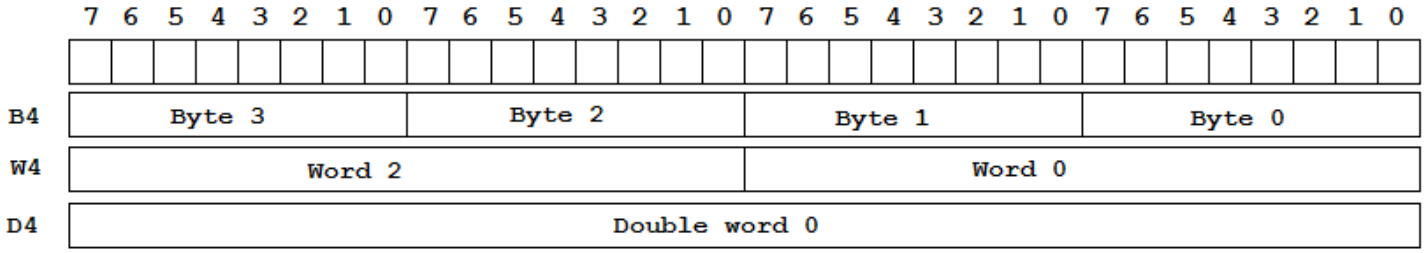
5.1.1 BYTE (8 bit)



5.1.2 WORD (16 bit – 2 Byte)



5.1.3 DOUBLE WORD (32bit – 2W – 4B)

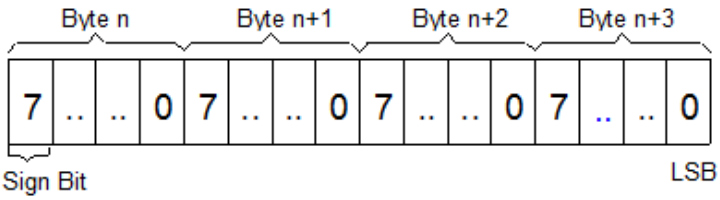


5.1.4 Obračanje bajtov in wordov

Vendar pozor. Kadar shranjujemo število tipa DINT dolžine MD, se teža bytov ravno obrne. M0.7 je MSB, M3.0 pa LSB. Ravno tako je pri številih tipa INT, dolžine MW.

==========

Data Type	Length (bits)	Format	Format Examples	
			Min.	Max.
DINT	32	Signed integer	L# -2147483648	L#+2147483647



0 corresponds to the sign "+"
1 corresponds to the sign "-"

MSB: Most Significant Bit

LSB: Least Significant Bit

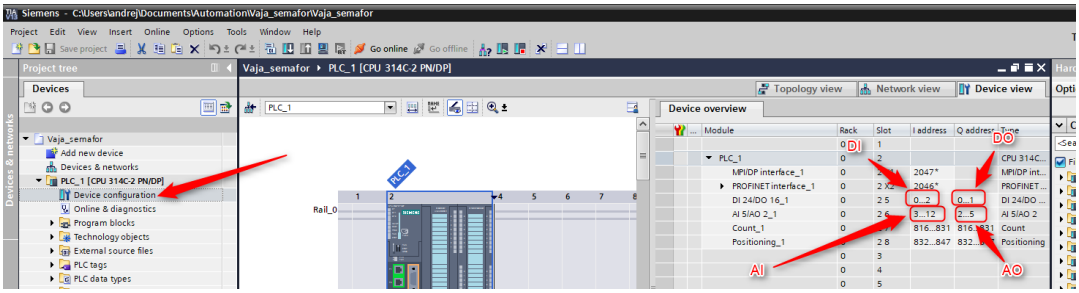
==========

Kako se pretvarja desetiško vrednost v dvojiško in obratno:
http://projekti.gimvic.org/2003/2b/sistemi/ostalo/dvojiski.htm

BIT	3	2	1	0	DEC	HEX
	0	0	0	0	**0**	**0**
	0	0	0	1	**1**	**1**
	0	0	1	0	**2**	**2**
	0	0	1	1	**3**	**3**

	0	1	0	0	**4**	**4**
	0	1	0	1	**5**	**5**
	0	1	1	0	**6**	**6**
	0	1	1	1	**7**	**7**
	1	0	0	0	**8**	**8**
	1	0	0	1	**9**	**9**
	1	0	1	0	**10**	**A**
	1	0	1	1	**11**	**B**
	1	1	0	0	**12**	**C**
	1	1	0	1	**13**	**D**
	1	1	1	0	**14**	**E**
	1	1	1	1	**15**	**F**

5.2 Vhodi



Do določenih naslovov lahko pridemo preko nzaka 'i', ki označuje input oz. vhod

5.2.1 Digitalni

Digitalne vhode naslovimo kot: I#,# pri čemer 1. # pomeni številko bajta, 2. # pa pomeni številko bita v bajtu. Torej ima bit lahko samo vrednost od 0-7.

V našem primeru lahko iz zgornje slike razberemo, da imamo 3 bajte dig. Vh. In sicer bajt 0, bajt 1 in bajt 2.

- Di0: I0.0
- Di1: I0.1
- …
- Di23: I2.7

5.2.2 Analogni

Analogni vhodi so dolžine dveh (zaporednih) bajtov, torej so dolžine WORD oz. imajo 16 bitno natančnost, zato jih označimo: kot IW#, pri čemer # pomeni številko prvega bajta. (razpon je lahko od -10V do +10V oz. vrednost, ki jo dobimo: -32768 do 32767).

Primer: IW3 (analogni vhod na bajtu 3 in 4), IW800, IW5, IW10,…

Kateri naslov bomo vzeli, moramo pogledati v nastavitve krmilnika

V našem primeru je zadeva sledeča:

1. AI1: IW3
2. AI2: IW5
3. AI3: IW7
4. AI4: IW9
5. AI5: IW11

5.3 Izhodi

5.3.1 Digitalni

Podobno kot vhodi so tudi izhodi določeni z bitom. Koliko izhodov imamo, lahko razberemo iz zgornje slike. V našem primeru imamo 2 bajta dig izhodov in sicer

- DO0: Q0.0
- DO1: Q0.1
- …
- DO15: Q2.7

5.3.2 Analogni

Kar se tiče natančnosti in dolžine ravno tako kot pri analognih vhodih, torej zasede 2 bajta. Iz zgornje slike lahko razberemo, da imamo 4 bajte analognih izhodov, torej 2 analogna izhoda in sicer na naslovih

1. AO1: QW2
2. AO2: QW4

5.4 Spomin

Kadar želimo kakšen podatek shraniti, lahko uporabimo vgrajeni spomin oz. Marker oz. Memory.

Kadar želimo priti do naslova za shranjevanje, uporabimo znak 'M'. Lahko shranimo posamezni bit, cel bajt naenkrat, število INT, DINT, REAL,…

5.4.1 Biti

Kadar shranjujemo posamezne bite, gre največkrat za situacijo shranjevanja z SET – RESET funkcijo. (primer: Vklop mašine s Start tipko, Izklop mašine s Stop tipko,….)

Ravno tako kot pri digitalnih vh in izh, se tudi bit spomina označi z: M#.# (pri čemer je 1. # št. Bajta, 2.# pa št. Bita.)

Primer: M0.0, M0.3, M10.5, M80.0, M80.1,…

5.4.2 Števila

Lahko si v spomin shranimo tudi števila. Običajno gre za kakšne števce, skaliranje analognih vhodov, razna matematična preračunavanja, parameteriziranje,….

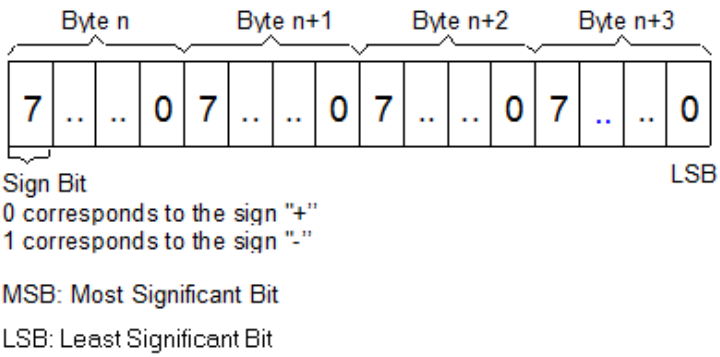
Kakšno število bomo shranili, moramo najprej vedeti kakšne je dolžine to število (poglavje 5. Naslavnjalnje)

Če bomo shranili število INT, to pomeni da je dolžine WORD oz 2 bajta.

Če želimo shraniti število DINT ali REAL, bo le to število zasedlo 4 bajte, zato moramo paziti da na bajtih kjer želimo shraniti število že nimamo kakšnega drugega pomembnega podatka.

Poleg tega ne pozabimo, da se števila shranjujejo tako, da se pomembnost bajtov obrača.

Data Type	Length (bits)	Format	Format Examples	
			Min.	Max.
DINT	32	Signed integer	L# -2147483648	L#+2147483647



MSB: Most Significant Bit

LSB: Least Significant Bit

Vaje – vnos bitov in števil v spominski prostor M

Če ne pazimo kam shranjujemo vrednosti, lahko pride do prekrivanja vrednosti v spominu, posledica tega pa so izgubljeni ali zavajajoči podatki. Če po vrstnem redu shranjujemo spodnje podatke, dobimo sledečo sliko: (števila, ki so podčrtana, so prekrila duge shranjene vrednosti.

M0.0 = true
M1.3 = true
M1.0 = true
M5.2 = true
MB1 = 78 B(0100 1110) (povozimo bit M1.0 in M1.3, katere smo prej zapisali v spomin, zato sta ta dva povožena)
MB7 = 10 B(0000 1010)
MW6 = 14557 B(0011 1000 1101 1100)
MD8 = 0 B(0000 0000 0000 0000 0000 0000 0000 0000)
M9.4= true (posledica tega je, da dobi MD8 vrednost 1048576)

Byte | bit	7	6	5	4	3	2	1	0
0								1
1	0	1	0	0	**1**	1	1	**0**
2								
3								
4								
5						1		
6	0	0	1	1	1	0	0	0
7	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**
8	0	0	0	0	0	0	0	0
9	0	0	0	**1**	0	0	0	0
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
12								
13								
14								
15								

Podšrtana števila so prepisala prejšnjo vrednost

5.5 Direktno

Direktno pomeni da uporabimo fizični naslov npr: I1.3, Q2.1, M4.2, MW10, MD13,…

5.6 Simbolno

Vsak naslov poimenujemo in ga vidimo v mapi PLC Tags.

6 Organizacija podatkovnih tipov

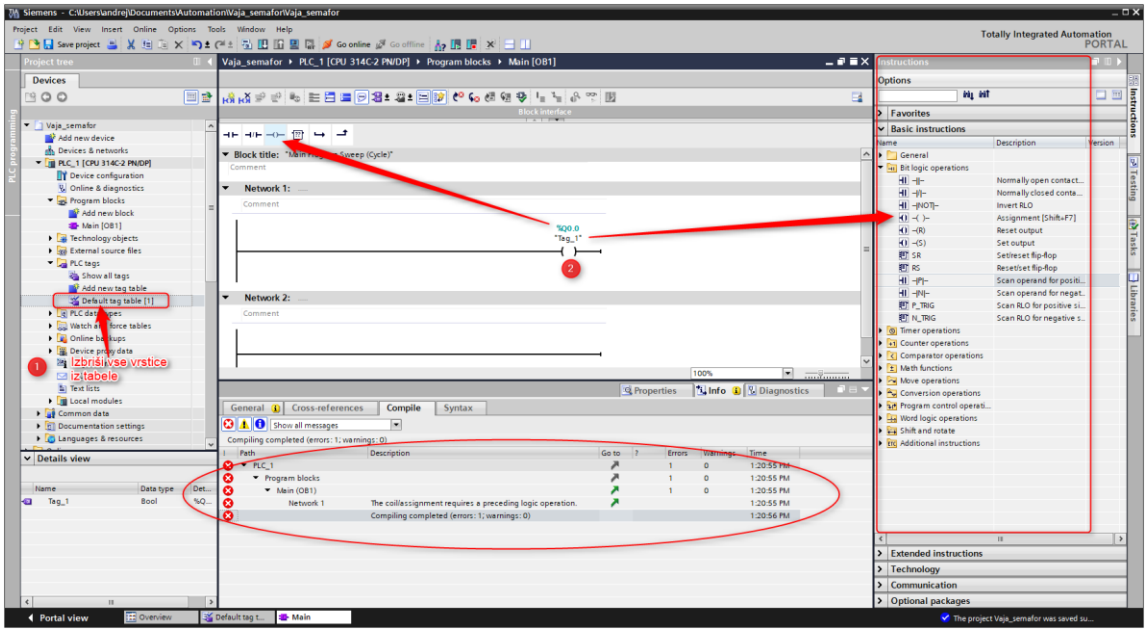
7 Osnovni bitni gradniki

Če želimo delati z bitno logiko, imamo v levem oknu orodjarno (Tool box), v kateri najdemo različne gradnike za logiko.

7.1 Assigment

Najprej izbrišimo vse vrstice (Tage) iz mape PLC tags / default dag table.

Dodajmo tuljavo Coil, katera označuje zapis vrednosti na naslov (Izhod ali spomin) Mi bomo zapisali na prvi digitalni naslov torej nad tuljavo napišimo naslov Q0.0.

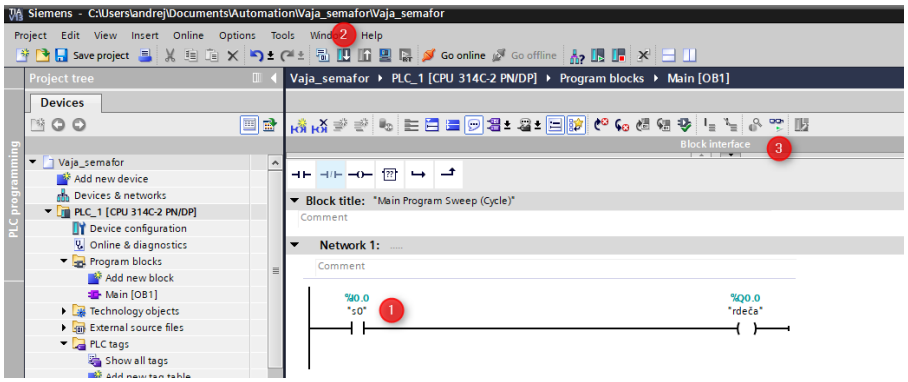


Ko napišemo naslov, nam za ta naslov dodeli tudi simbolno ime ali TAG. Ime je generično in se nadaljuje s povečevanjem indexa. Če želimo to ime spremeniti, z miško desno kliknemo na ime in izberemo Rename tag,

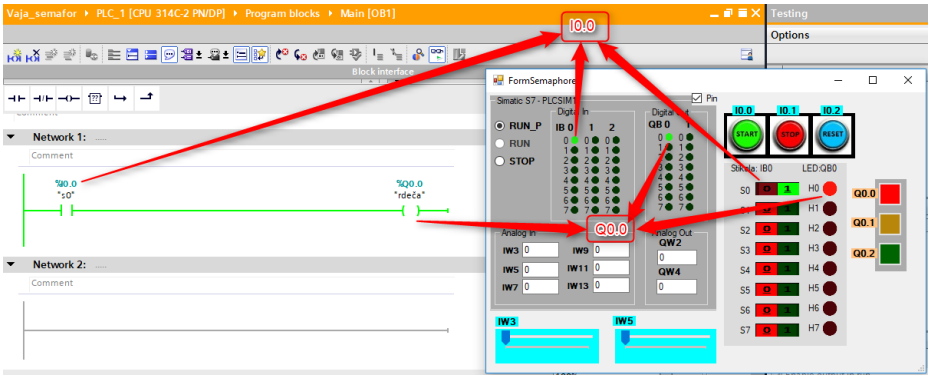
Napišemo ime in kliknemo Change.
V default tag table sedaj vidimo da je za naslov Q0.0 dodeljeno simbolno ime rdeča.
Ko kliknemo na gumb compile, se pojavi napaka, to pa zato, ker pred izhodom ni nobene logike ali gradnika.

7.2 NOC

Dodajmo odprt kontakt pred Coil in mu dodelimo naslov I0.0 in spremenimo ime taga s0.
Naložimo program na PLCSIM in preizkusimo simulacijo. Ob enem moramo tudi zagnati PLCSIM (RUN_P).

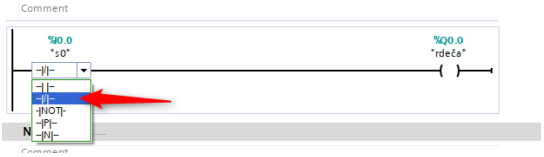


Če vključimo stikalo S0, se prižge tudi luč.



7.3 NCC

Dvokliknimo na kontakt in izberimo možnost NCC



Ali iz orodne vrstice izberemo ta element.

Ponovno naloži program in ga testiraj. Luč gori če je stikalo S0 ugasnjeno če pa prižgano pa luč ne sveti.

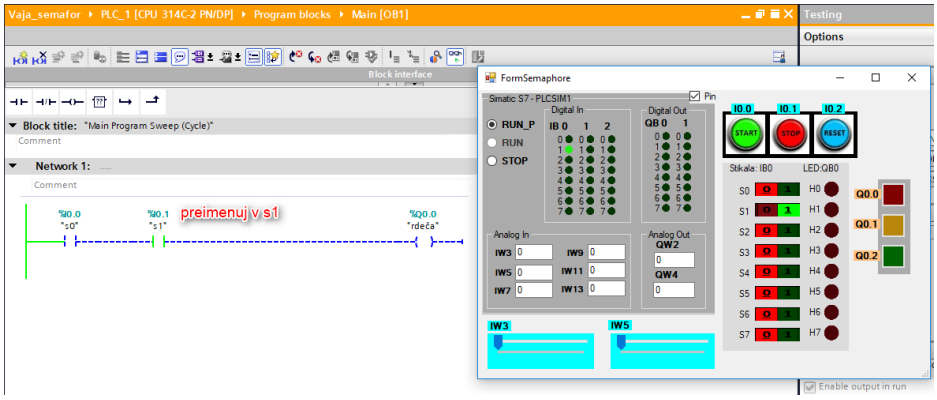
7.4 Vaje

7.4.1 Logični IN

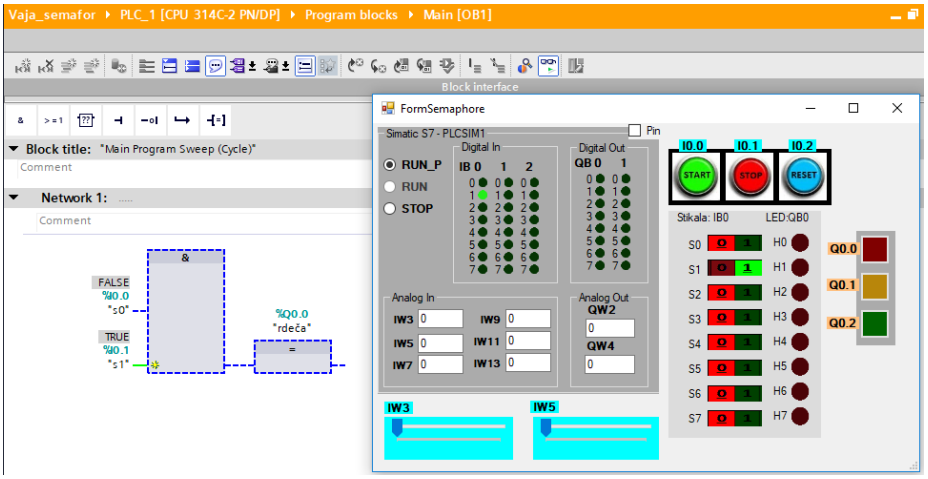
Logično in funkcijo v LAD sestavimo z zaporedno vezavo dveh kontaktov, tako kot je na spodnji sliki. V tem primeru bo na izhodi '1' samo, če bosta oba kontakta sklenjena, torej če sta oba naslova v '1'.

Logična enačba:

Rdeča = s0*s1

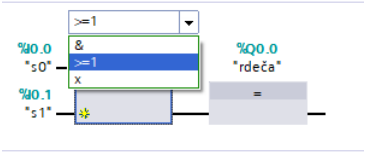


Če spremenimo pogled v FBD dobimo simbolni zapis.

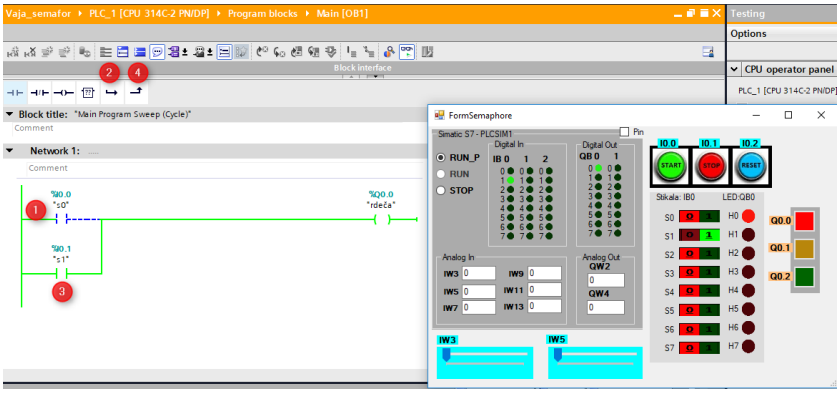


7.4.2 Logični ALI

V FBD lahko funkcijo spremenimo z dvoklikom na simbol &, tako da se nam ponudijo možnosti & AND), >=1 (OR) in x (XOR). Izberimo kot vidimo na spodnji sliki.



Če funkcijo pretvorimo v LAD dobimo vzporedno vezavo dveh kontaktov. To pomeni, da je na izhodu logična enica če je vsaj eden izmed naslovov v '1'.

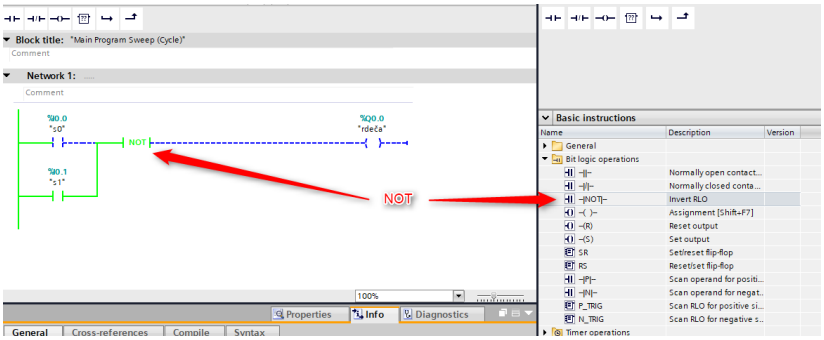


Postopek v LAD:

1. Označimo del kjer želimo vejitev
2. Kliknemo na ikono »open branch«
3. Dodamo kontakt ali funkcijo
4. Zapremo funkcijo

7.4.3 Logični NE

Če želimo celo ali del funkcije negirati, enostavno dodamo inštrukcijo NOT, kot vidimo na spodnji sliki. Tako smo iz ALI funkcije dobili NE ALI.



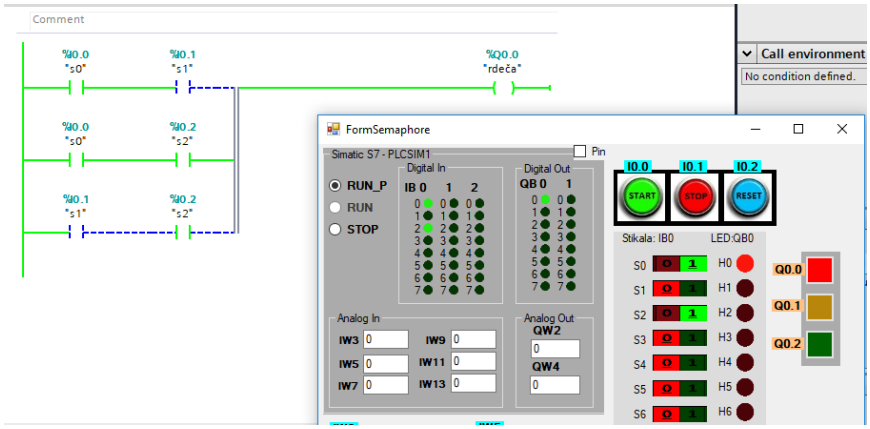
7.4.4 Sestavljene funkcije

Funkcije lahko tudi sestavimo iz osnovnih v kompleksnejše.

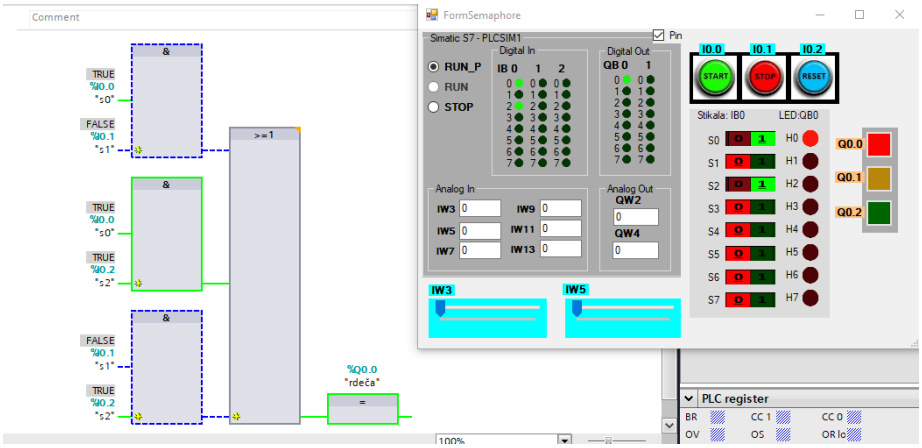
Sestavimo funkcijo, tako, da bo rdeča luč svetila v primeru, da sta vsaj dva izmed treh stikal vključena.

Zapis funkcije: rdeča = s0*s1 + s0*s2 + s1*s2

LAD:



FBD:

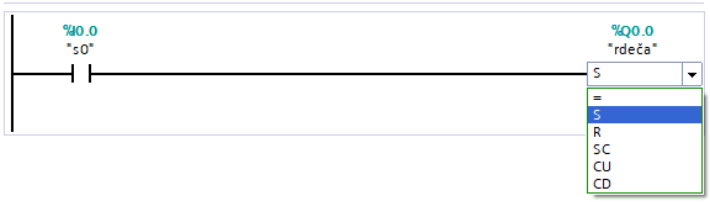


7.4.5 SET

Kadar želimo trajno postaviti nek naslov. Ko pride logična enica do tuljave SET, se naslov (bit spomina ali DO) postavi v '1'. V logični enici ostane tudi če gre nato signal v '0'.

Primer vklop mašine s tipko.

Dvoklik na izhod (rdeča – Q0.0) v našem primeru izberemo 'S' – SET

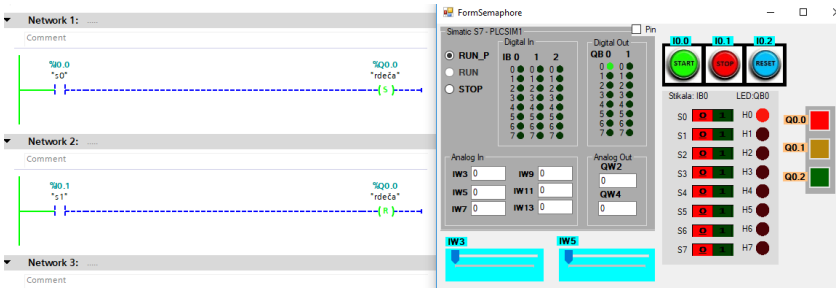


Izhod bo v '0' dokler do njega ne pride vsaj en pulz '1'. Od takrat naprej bo izhod v '1'

7.4.6 RESET

Če želimo funkcijo, s katero s pulzom naslov postavimo v '0' uporabimo R

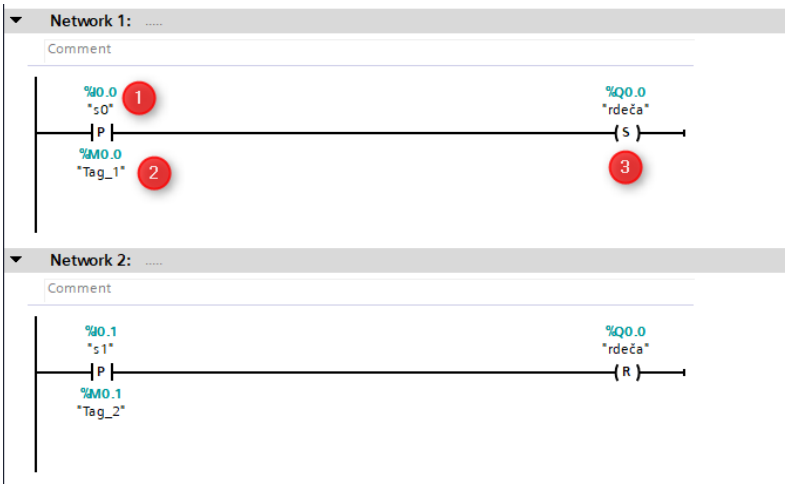
Vaja: s stikalom s0 vključimo rdečo luč, s stikalom s1 pa ugasnemo.



7.4.7 P

Funkcija na izhodu da samo kratek impulz '1' v ko na vhodu pride do spremembe iz '0' v '1'. Običajno jo potrebujemo takrat, ko želimo da se nek dogodek izvede samo enkrat in ne vsak cikel (vsakih 10ms)

1. Signal, ki ga želimo uloviti
2. Nek bit spomina, ki je nujno potreben za delovanje funkcije (ta bit ne sme biti uporabljen nikjer drugje!)
3. Pulz enega cikla '1', katerega največkrat kombiniramo s set in reset funkcijo.



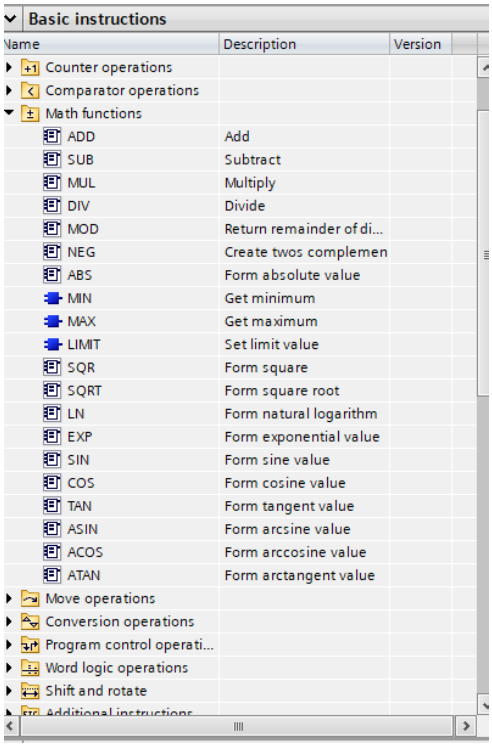
7.4.8 N

Ista funkcionalnost kot P, le da ta funkcija da kratek impulz na izhodu, ko pride na vhodu do spremembe iz '1' v '0'.

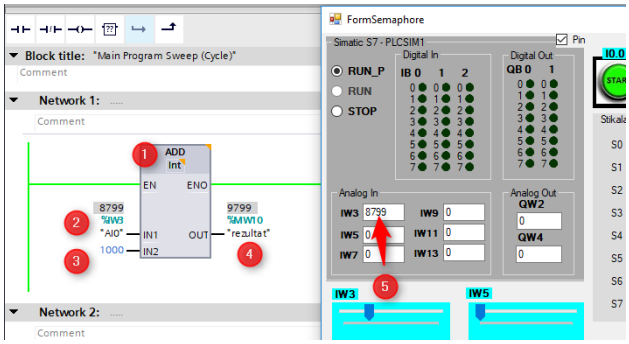
8 Vgrajene funkcije

8.1 Številske funkcije

Kadar želimo matematično opisati problem, kot so na primer enačbe. Imamo možnost izbire med različnimi podatkovnimi tipi: INT, DINT, REAL



Za primer si poglejmo seštevanje:



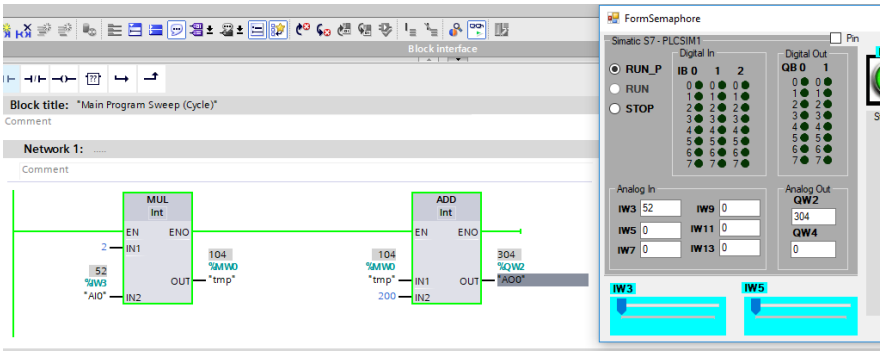
1. Najprej dodamo blok ADD, nato pa mu določimo s katerimi podatkovnimi tipi bomo računali. Ker bomo v našem primeru uporabili kar analogni vhod AI0 na naslovu IW3, kateri je tipa INT, mu moramo tudi le tega določiti z dvoklikom na ???
2. Na IN1 prvi seštevanec smo dodelili analogni naslov IW3 smo dodelili tag ime AI0,
3. IN2 drugi seštevanec, kateri je lahko konstanta ali drug naslov spomina,…. V našem primeru smo vpisali konstanto 1000.
4. Na izhodu bomo dobili seštevek IN1 in IN2
5. Testiramo tako, da pomaknemo slider IW3.

Vaja: Sestavi linearno enačbo Y = kx + n, pri čemer je

- K=2
- X= IW5
- N = 200

Rezultat naj se shrani na izhod QW2.

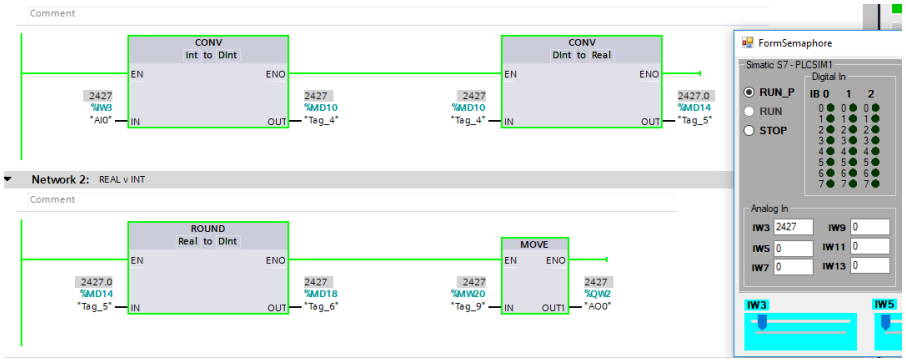
Rešitev:



V matematiki vemo, da ima množenje prednost pred seštevanjem, tukaj pa moramo za to poskrbeti mi. Kot sem že omenil v začetnem poglavju se bloki izvajajo od leve proti desni, zato moramo množenje dati pred seštevanje.

Uporabili smo tudi naslovni prostor MW10 zato da smo vanj shranili vmesni izračun, katerega smo potem uporabili v seštevanju.

8.2 Konverterji



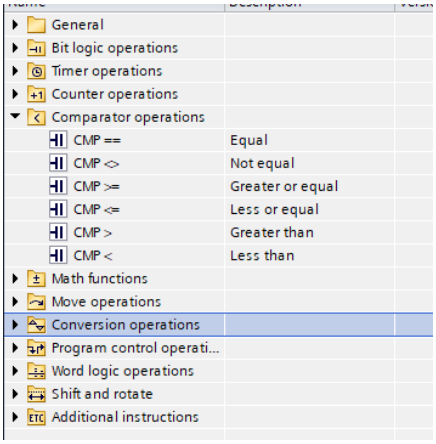
Pri konverziji med INT in REAL moramo vmesni rezultat shraniti na poljubno lokacijo dolžine DWORD (4bajte) končni rezultat pa smo shranili na lokacijo MD14.

Konverzija v obratni smeri pa je sledeča:

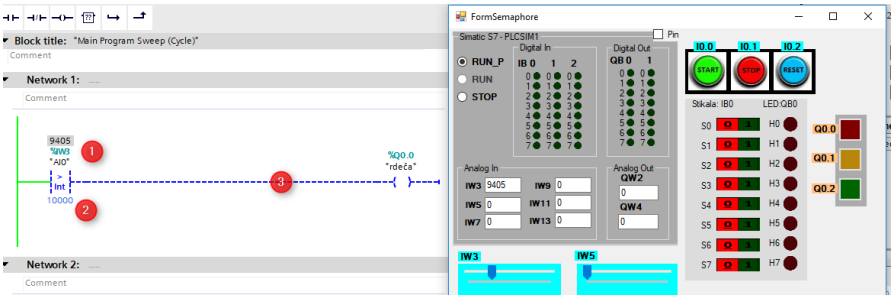
Iz lokacije MD14 vrednost zaokrožimo iz REAL v DINT, shranimo na MD18 (bajti 18, 19, 20, 21). Če pa želimo vrednost iz DINT v INT, moramo vzeti samo 2 bajta. Ker so pri številih vrednosti bajtov shranjene v obratnem vrstnem redu, moramo namesto bajt 18 in 19 vzeti 20 in 21, torej MW20. Le to lokacijo MW20 prenesemo na izhod QW2.

8.3 Komparatorji

Komparatorji so namenjeni primerjanju dveh vrednosti in sicer večji manjši ali enako.



Za primer bomo zopet vzeli primerjanje analognega vhoda s številko in sicer če IW3 preseže vrednost 10000, naj sveti rdeča luč.



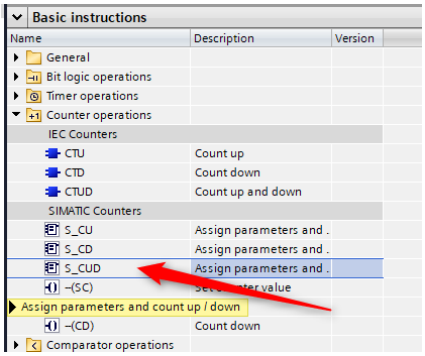
1. Vrednost, ki jo želimo primerjati
2. Limita
3. Izhod je v '1', če je vrednost 1 večja od vrednosti 2, drugače je izhod v '0'

Vaja: naredi program, da zelena luč sveti, če je IW3 v območju med 5000 in 10000. Če je vrednost IW3 izven tega območja, naj sveti rdeča.

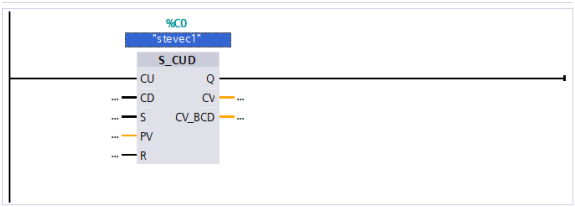
8.4 Counterji

Counterje oz. števce uporabljamo, ko želimo šteti neke dogodke. Uporabimo lahko različne izvedbe števcev in sicer štetje gor, dol, in gor-dol. Izberemo tistega, kateri nam bolj ustreza.

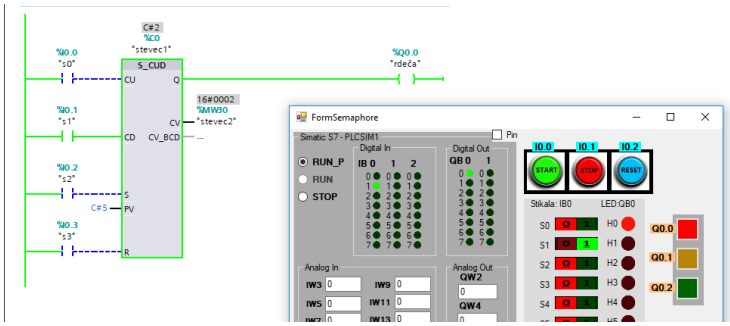
Mi si bomo pogledali štetje gor-dol.



Števec uporabimo v networku, ter mu dodelimo naslov C0, ter ga preimenujmo v stevec1.



- CU – ko pride do spremembe iz '0' v '1', se števcu C0 poveča vrednost za 1
- CD – ko pride do spremembe iz '0' v '1' se števcu C0 vrednost zmanjša za 1
- S – postavi vrednost števca na vrednost, ki je na vhodu PV
- PV – vrednost na katero števec postavimo s vhodom S
- R – resetiramo števec na 0
- Q – na izhodu da '1', če je vrednost števca večja od 0, drugače pa da na izhodu '0'
- CV – vrednost števca, shranimo na poljuben naslovni prostor dolžine WORD (v našem primeru MW30)



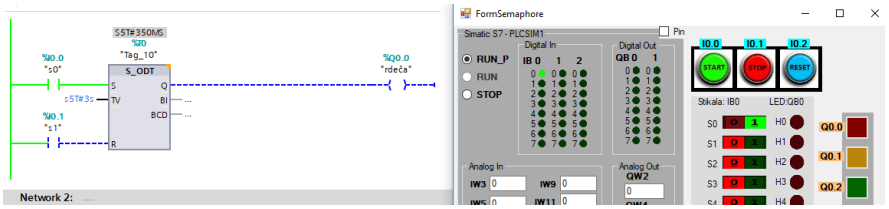
Običajno ga uporabimo v kombinaciji s komparatorji

8.5 Timerji

V S7 imamo na voljo več različnih časovnikov – timerjev, ki so zgrajeni glede na namen. Sicer lahko z vsakim timerjem rešimo vse situacije, vendar jih je dobro poznati, da s čimmanj dela dosežemo želeni rezultat.

Naslov, ki ga vpišemo je T0, T1, T2,…T99

Mi si bomo pogledali samo ODT – On Delay Timer, ki deluje na sledeče. (vklop z zakasnitvijo)

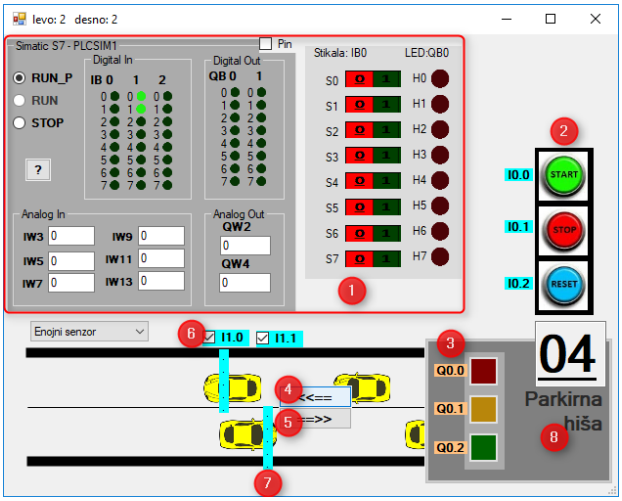


Ko se na S pojavi '0', če čas TV začne odštevati. (v našem primeru 5s) ko se čas izteče, se izhod Q postavi v '1'. Če se medtem na R pojavi '1', se odštevanje prekine.

Najlažje je, če vsak časovnik preizkusite kako se obnaša, in le nato izberete tistega, ki vam najbolj ustreza.

9 Vaje

9.1 Parkirna hiša



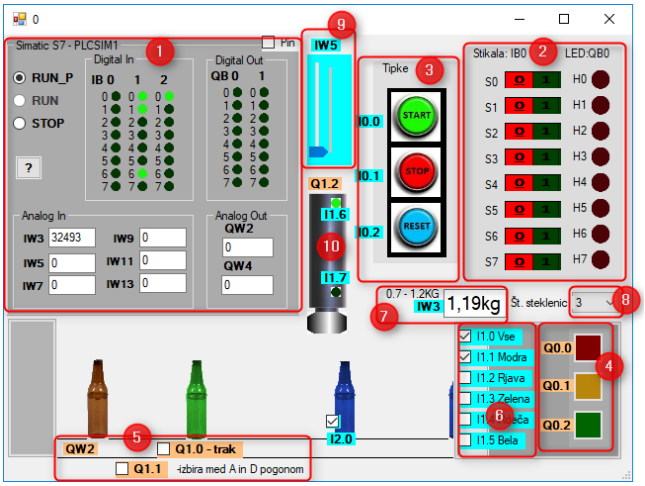
Opis simulacije:

1. Simulacija krmilnika, 8 stikal in 8 luči
2. Start, stop, reset tipke, ki so vezane vzporedno s prvimi tremi stikali.
3. Semafor (rdeča, rumena, zelena) vzporedno vezane s prvimi tremi lučmi
4. Gumb za pošiljanje avtomobila iz garaže
5. Gumb za pošiljanje avtomobila v garažo
6. Senzor, ki zazna avto, ki gre iz garaže
7. Senzor, ki zazna avto, ki gre v garažo
8. Garaža in števec trenutnih avtomobilov. (če kliknemo na številko, se garaža resetira)

Naloga:

- Izdelaj program, kateri bo beležil število avtomobilov v parkirni hiši s 6 parkirnimi mesti.
- Število avtomobilov prikazuj na izhodu QW2
- Če parkirna hiša ni polna, naj sveti zelena luč
- Če je parkirna hiša polna, naj sveti rdeča luč
- Če je prostih več kot polovica parkirnih mest in ni polno, naj poleg zelene sveti še rumena.

9.2 Tekoči trak steklenic in tehtanje le teh



1. Določimo hitrost tekočega traku
2. Določimo, koliko različnih barv bomo imeli na tekočem traku
3. Tekoči trak po katerem se pomikajo steklenice
4. Senzorji, kateri porepoznajo barvo steklenice in tehtnica
5. Tehtnica, ki tehta steklenice, dokler so v območju senzorja. Steklenice imajo naključno maso med 0.7 do 1.2 kg. Vrednost tehtnice gre na IW3. (ko ima IW3 vrednost 32767, tehtnica meri 1.2kg)

Naloga 1:

- Izdelaj program, kateri vključi tekoči trak, ko pritisnemo START tipko
- Izključi tekoči trak ob pritiski na STOP tipko
- Trak naj se zažene s START tipko. Ko v merilnico pride 6 steklenic, naj se trak ustavi.
- Ko operater zamenja zaboj, le to potrdi s START tipko, trak naj nadaljuje s pomikanje steklenic
- Ko trak pomika steklenice, naj gori zelena luč
- Ko je v merilnici 6 steklenic in je trak ustavljen naj utripa rumena luč (1Hz)
- Če v merilnico pride steklenica s težo manjšo od 0.75kg ali večjo od 1.15kg, naj se trak ustavi in prižge RDEČA luč. Operater le to napako potrdi s START tipko in trak naj nadaljuje.
- Dodelaj program tako, da naj se trak zaradi stabilizacije meritve ustavi za 0.5s, ko steklenica pride v merilnico.
- Dodelaj program tako, da bo merilnica razvrščala tudi barve. Torej, ko pride 6 steklenic iste barve v merilnico.

Naloga 2: Predelaj zgornji nalogo tako, da ne bomo šteli steklenic, ampak bomo merili težo steklenic. Namesto 6 steklenic, naj bo meja za poln zabojnik vsaj 10kg steklenic.

10 FC, FB, DB

Kadar želimo napisati del programa, ki bi ga radi uporabili večkrat ali na več delih, lahko zgradimo svojo funkcijo (FC) ali funkcijski blok (FB). DB pa se uporablja skupaj s FB, kamor se shranjujejo statiče spremenljivke, ali pa za shranjevanje podatkov, parametrov, meritev,…

10.1 Funkcije

Ko pridemo do situacije, ko že v naprej vemo, da bomo kakšen del programa potrebovali večkrat, lahko tudi v več porjektih, si zgradimo lastno funkcijo (Function Call – FC)

Tipičen primer je preračunavanje analogne vrednosti (napetosti na AI) v neko mersko enoto (npr. kilogrami, litri, mm,…)

Ker želimo, da so funkcije namenjene splošni uporabi in ne samo za točno določen projekt, običajno funkciji kot vhodne podatke podamo mejne vrednosti in spremenljivi del, kot izhodni del pa vrednosti, ki jo izračunamo.

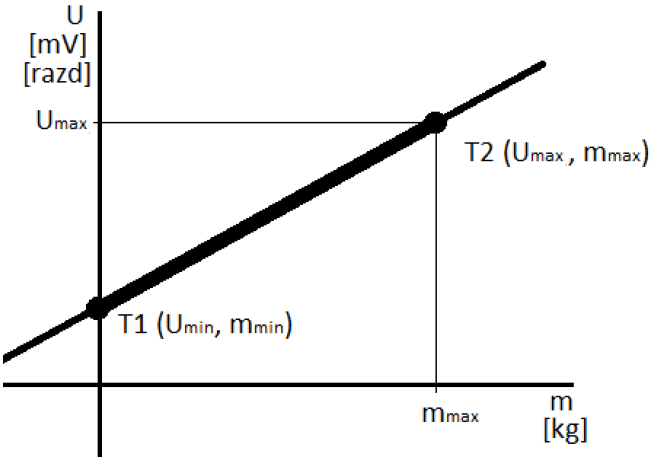
V industriji se bolj ali manj srečujemo z linearnimi pretvorbami, zato bomo tudi sami zgradili funkcijo za preračunavanje analogne vrednosti napetosti v kg, mm, L,…

Linearno enačbo v splošnem zapišemo kot Y = kx + n. Če pogledamo v splošnem, bo Y naša merska enota, torej kilogrami, k – konstanta (naklon premice), x – napetost na AI (0 – 32767), n odmik premice od izhodišča.

Kot pogled na proces želimo vnesti mejne vrednosti in sicer vrednost na AI pri minimalni teži, ter vrednost na AI pri maximalni teži.

http://www.educa.fmf.uni-lj.si/izodel/sola/2000/dira/marusa/Index3.htm

http://www2.arnes.si/~ljspssb4/linearna/enacba_dve.html



Torej moramo kilograme izračunati po enačbi:

$$m = \frac{\mathrm{U} - U_{min}}{\mathrm{k}}$$

$$k = \frac{U_{max} - U_{min}}{m_{max} - m_{min}}$$

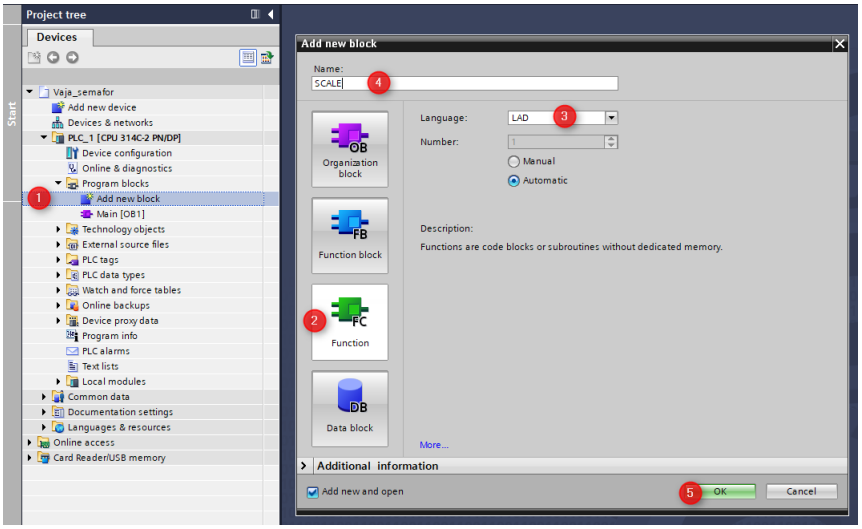
N pa dobimo iz enačbe pri eni izmed točk npr:

Y = kx + n (teoretično enačba premice)

U = k*m + n (v našem primeru pa je sledeče)

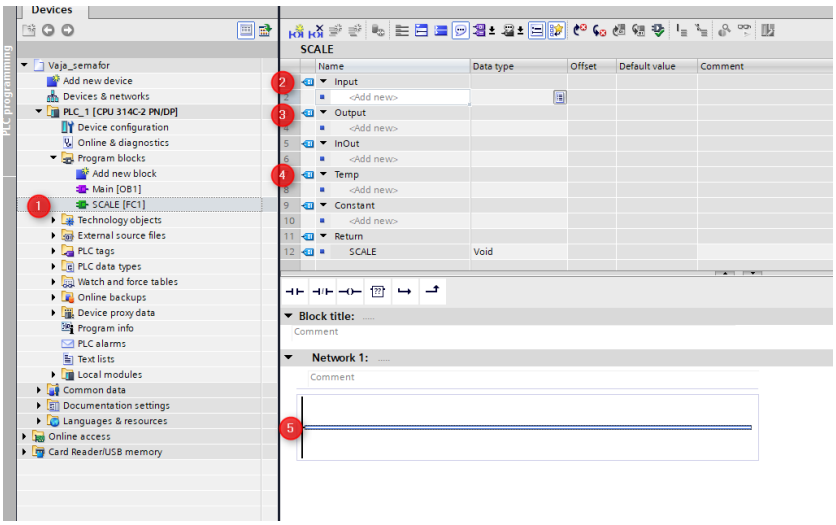
n = U – k*m (offset)

Naredimo svojo funkcijo in jo poimenujmo SCALE



1. Dodajmo nov blok
2. Izberimo Funkcijo
3. Jezik LAD
4. Ime SCALE
5. Kliknimo OK.

Sedaj odprimo blok.



1. Dvokliknemo na zeleno ikono SCALE
2. Če razširimo zgornji del okna, v njem dobimo sledeče nastavitve in sicer Vhodi
3. Izhodi
4. Začasne spremenljivke
5. Prostor, kjer se gradi program

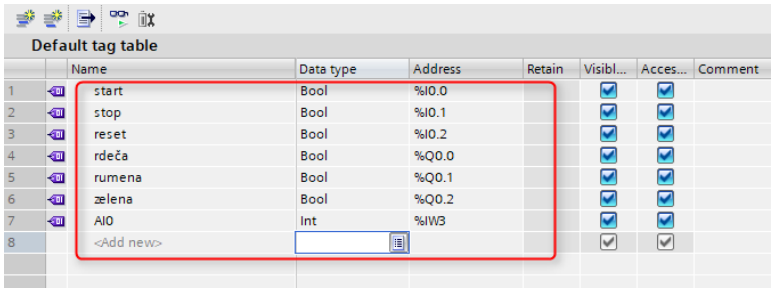
Kot smo si zadali cilj, bomo izdelali splošno funkcijo za skaliranje, v katero bomo kot vhodne podatke vnesli:

- Umax – največja vrednost na AI (pri m=1.2kg)
- Umin – najmanjša vrednsot na AI (pri m=0kg)
- Mmax – največja masa (1,2kg)
- Mmin – najmanjša masa (0kg)
- U – trenutna vrednst na AI

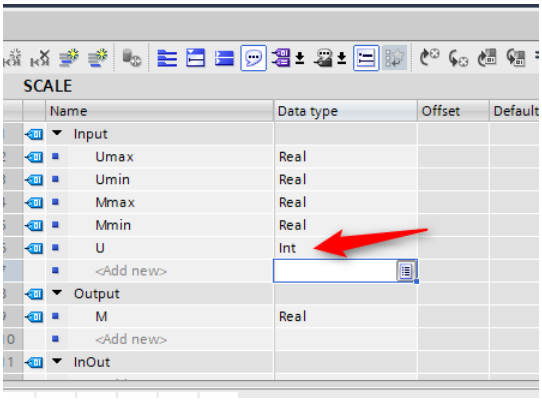
Izhod pa bo naša trenutna masa m v kg.

Zaradi preciznosti bodo vse vh-izh. Smremenljivke tipa Real.

Projekt shranimo kot »skaliranje«, ter iz mape PLC tags/Default tag table izbrišimo vsa imena dopiši le tista, kot vidiš na spodnji sliki.



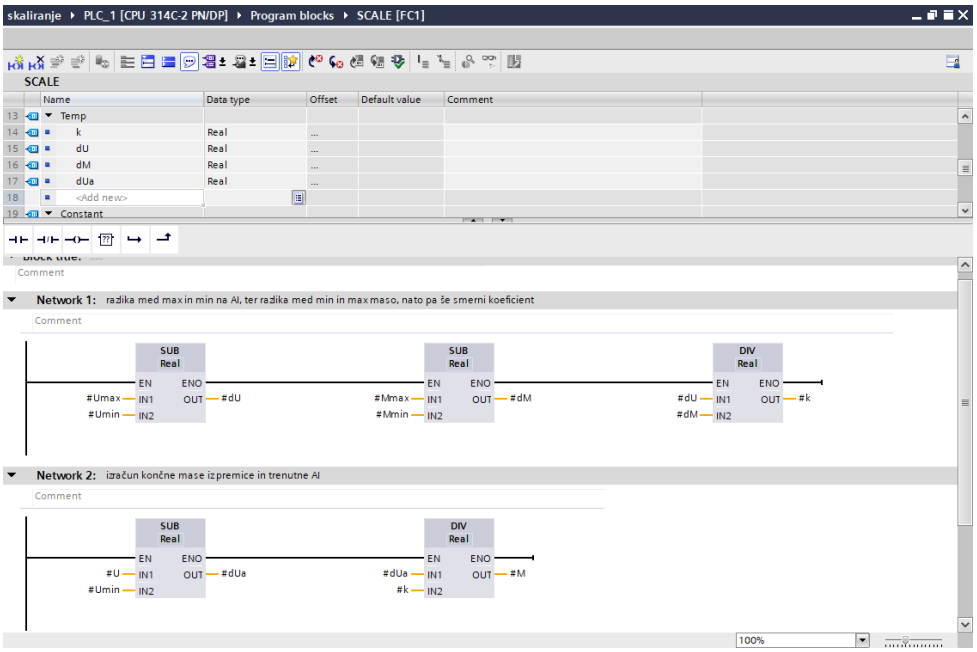
Nato pod Input dodaj 5 vhodnih spremenljivk, katere morajo biti tipa Real razen U naj bo INT, ter en izhod M ravno tako tipa Real



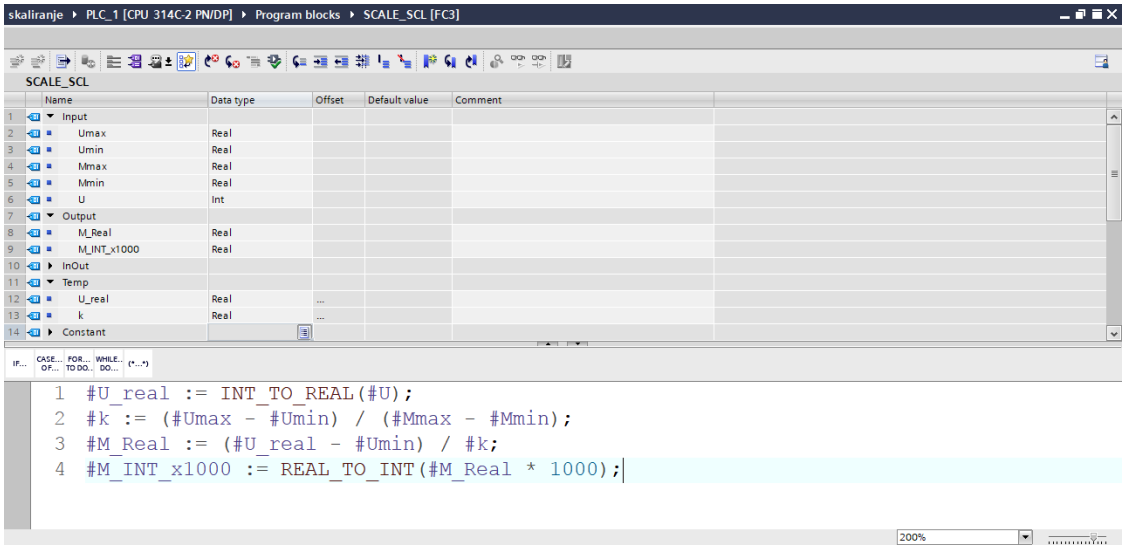
Najprej izračunajmo smerni koeficient po spodnji enačbi in jo dodajmo kot TEMP spremenljivko. Če potrebujemo še kakšno spremenljivko za vmesne izračune jo ravno tako napišemo pod mapo TEMP.

Izračunajmo $k = \frac{U_{max} - U_{min}}{m_{max} - m_{min}}$ in $m = \frac{\mathrm{U} - U_{min}}{\mathrm{k}}$

Rešitev v LAD

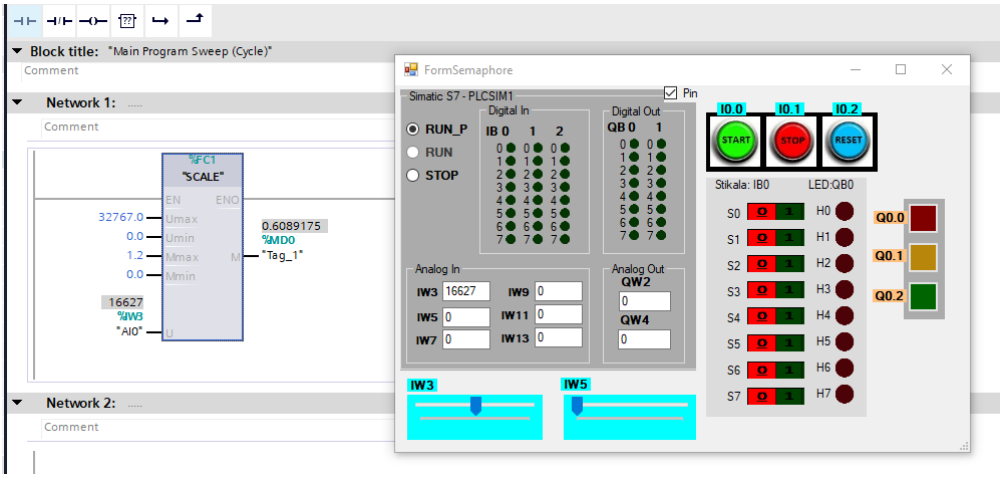


Rešitev v SCL:



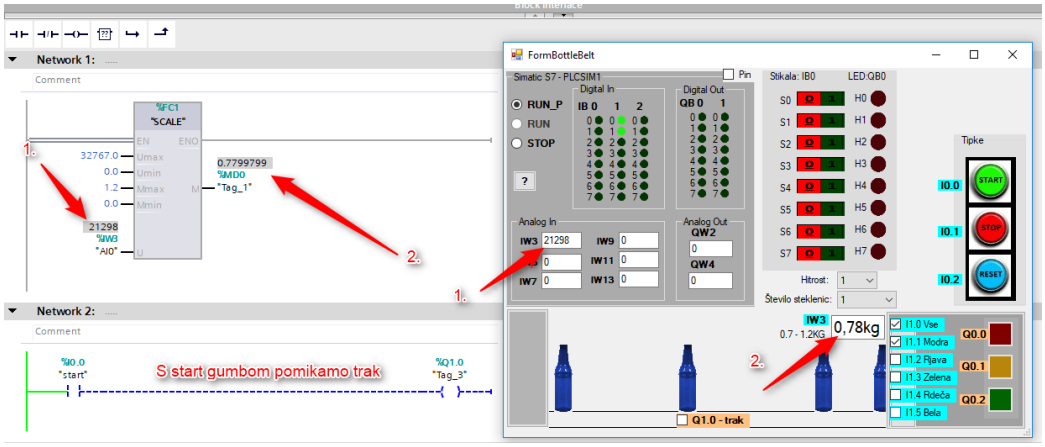
Sedaj pa funkcijo preizkusimo.

Odpremo Main in v dodamo našo funkcijo, ter vpišemo vse vrednosti



- Umax – 32767
- Umin – 0
- Mmax – 1.2
- Mmin – 0
- U – IW3 (AI0)
- M – MD0 (samo za to, da shranimo vrednost)

Lahko pa pogledamo tudi pri tehtnici:



10.2 Funkcijski bloki

Funkcijski bloki se običajno uporabljamo, kadar želimo napisati program, ki ni odvisen samo od vhodnih stanj, ampak tudi od trenutnih izhodnih stanj, zaradi česar se morajo ta stanja med delovanjem tudi shraniti, zato vsakemu funkcijskemu bloku pripada tudi svoj datablok, v katerega se shranjujejo vsi potrebni podatki. Običajno gre lahko tudi za program, ki je sestavljen po korakih ali uporablja števce, časovnike, pa tudi pri krmiljenju aktuatorjev npr. ventilov, motorjev,…, sploh, če se ta funkcija oz. program večkrat ponavlja. Tako kot pri Funkcijah (FC), tudi pri datablokih lahko uporabimo vhode in izhode funkcije, to pa zaradi fleksibilnosti.

10.3 Databloki

10.4 Data bloki za shranjevanje podatkov in parameteriziranje

11 Koračna veriga (SCL)

12 GRAPH

Vsaka koračna naprava deluje po točno določenem zaporedju. Korak1, korak2, korak3,…korakN. Kdaj se program premakne iz enega koraka v drugega, je odvisno od vhodnih stanj in/ali časov oz. zakasnitev. Tem rečemo prehodni pogoji.

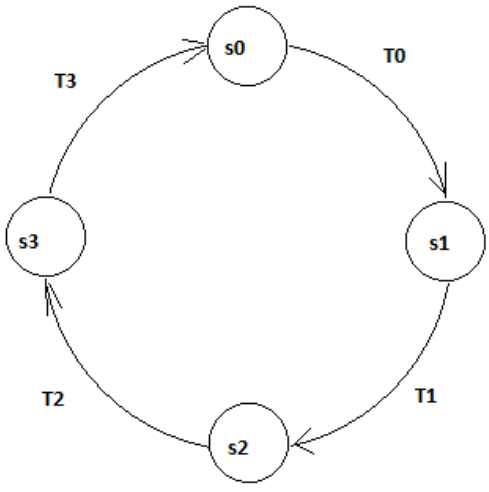
Posamezni korak označimo S0, S1, S2, S3, pogoje za prehod pa označimo z T0, T1, T2, T3.

Bistvo vsega je, da se program začne s prvim korakom S0. Ko so v S0 izpolnjeni vsi pogoji za prehod, se aktivira drugi korak S1, kateri avtomatsko izključi svoj predhodni korak, torej S0. Na ta način pridemo do zadnjega koraka. Če želimo, se lahko zopet aktivira prvi korak, ki pa izključi zadnji korak.

Ko je aktiven določen korak S, le ta postavi izhode v določeno stanje, in jih drži toliko časa, dokler ne preidemo v drug korak. Te izhode oz. stanja lahko držimo samo za čas koraka, ali pa s SET ali RESET postavimo trajno do naslednje spremembe.

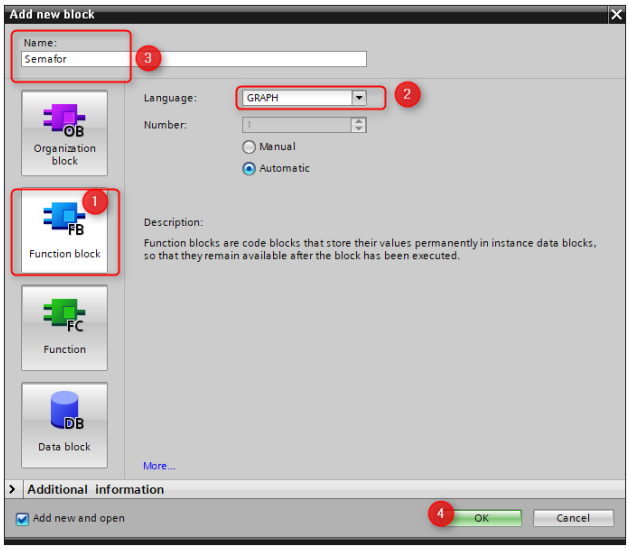
Prehod T je običajno vezava digitalnih vhodov, lahko pa vključuje tudi časovnike, komparatorje ipd,…

Shema delovanja:

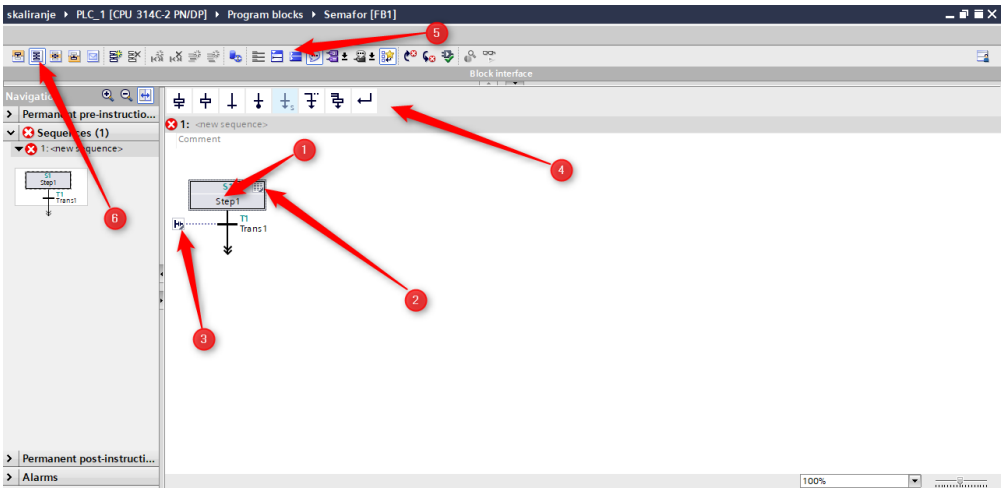


Dodajmo nov blok s klikom na Add new block.

Poskrbimo, da izberemo Function block, zamenjajmo jezik v GRAPH in ne LAD, ter dodelimo ime bloka »Semafor«.

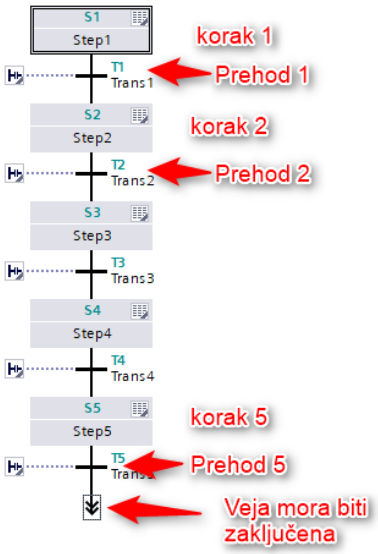


Ko se odpre blok, si na spodnji sliki poglejmo nekaj najbolj uporabljenih gumbov.



Opis gumbov:

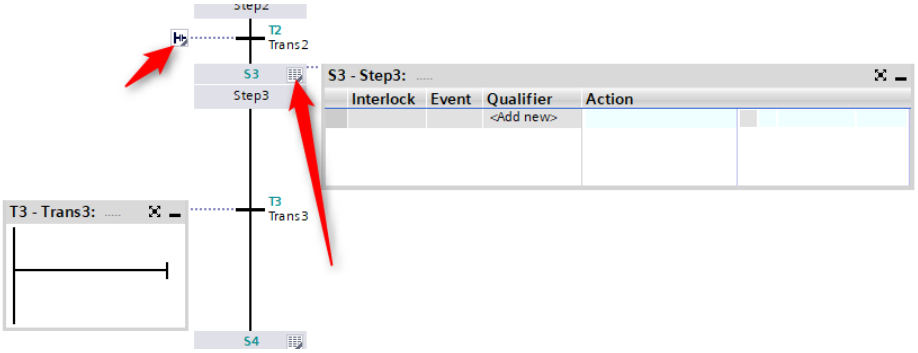
1. Korak
2. Odpremo akcjo oz. ukaz
3. Odpremo prehod oz. pogoje za naslednji korak
4. Bljižnica z ukazi
5. Pokaži / skrij prehode in akcije
6. Pokaži celo sekvenco



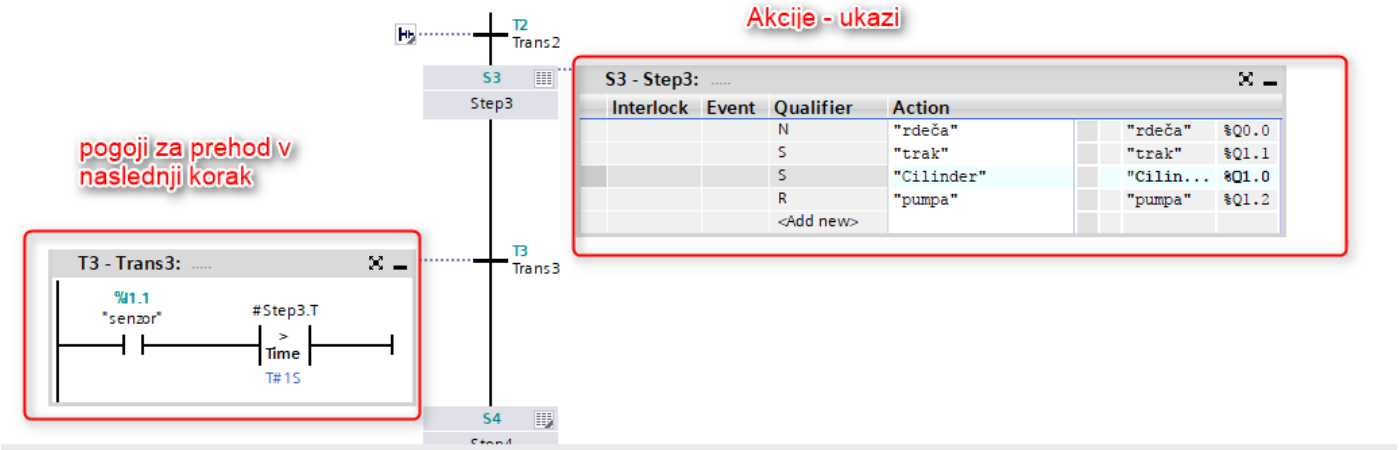
V resnični sekvenci si koraki sledijo v zaporedju. Program se začne v koraku 1, in je v njem, dokler ni pogoj oz. prehod izpolnjen. Ko se to zgodi, se program nadaljuje v koraku 2, korak 1 pa se ugasne. Naenkrat je aktiven lahko samo 1 korak.

12.1 Step + Transition

Če odpremo prehod in akcijo, sta oba prazna. Pri prehodu je pogoj takoj izpolnjen, zato program izvede akcije (če bi jih imeli) samo 1x, nato pa takoj skoči v naslednji korak.



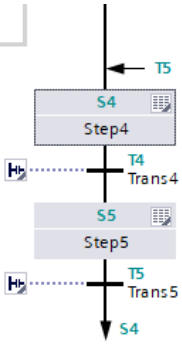
Če dodamo nekaj akcij in pogojev lahko izgleda takole



12.2 Jump, sequence end

Vsaka veja na koncu mora biti zaključena ali z sequence end ali jump ukazom. Če program doseže ukaz sequence end, se sekvenca neha izvajati, ker je prišla do konca. Program lahko ponovimo samo iz LAD, če na vhod INIT_SQ pride '1', ga resetiramo od zjunaj.

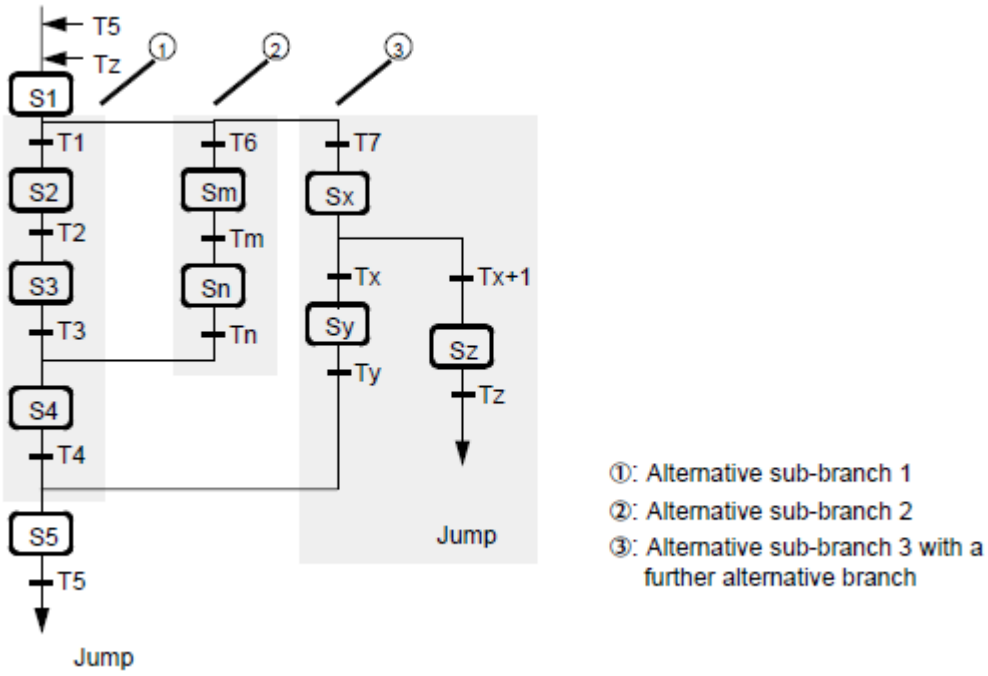
Če izberemo JUMP, pa program skoči na poljuben korak.



Zgornji primer prikazuje uporabo JUMP ukaza kateri skoči na korak 4 (step4). Ti skoki bi se ponavljali.

12.3 Alternative branch

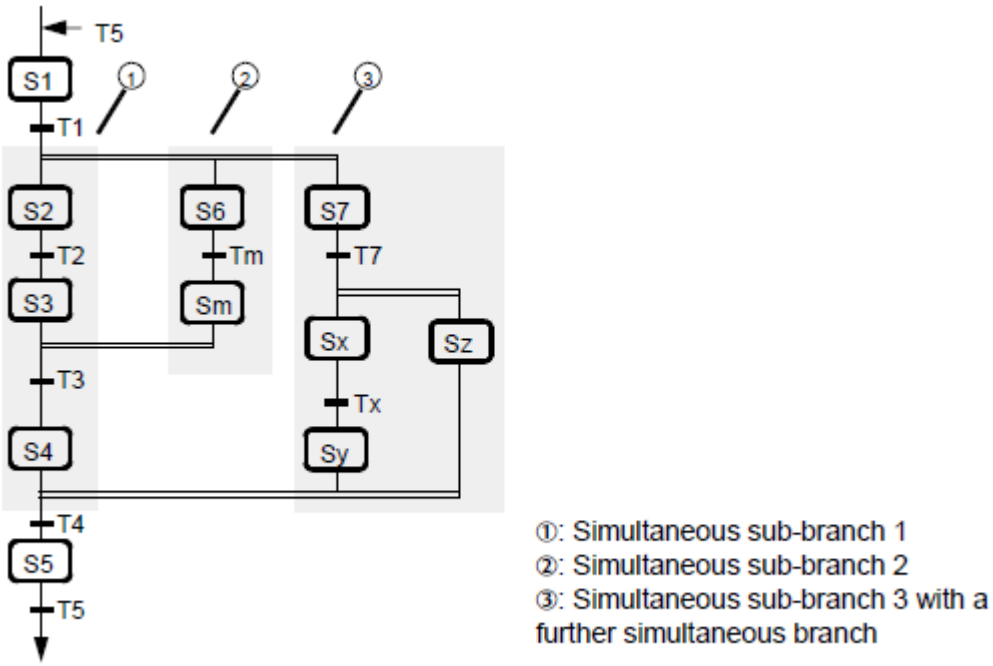
Uporabimo, kadar imamo na voljo več možnih potekov procesa.



Kadar imamo vzporednih 2 ali več alternativnih vej, se istočasno lahko izvaja samo 1. Torej, tista veja, kateri se prehod najprej odpre, tista veja se bo izvedla, ostali dve pa ne. Pomembno je tudi, da se vzporedni prehodi med seboj izključujejo, drugače lahko dobimo nepredvidljive situacije. V primeru, da je odprtih več prehodov hkrati, bo program vedno izbral najbolj levo vejo. Torej v našem primeru, (T1, T2, T3) bo izbral T1.

12.4 Simultanius branch

Kadar želimo istočasno in neodvisno upravljati z več procesi naenkrat, le ti pa so običajno med seboj neodvisni.

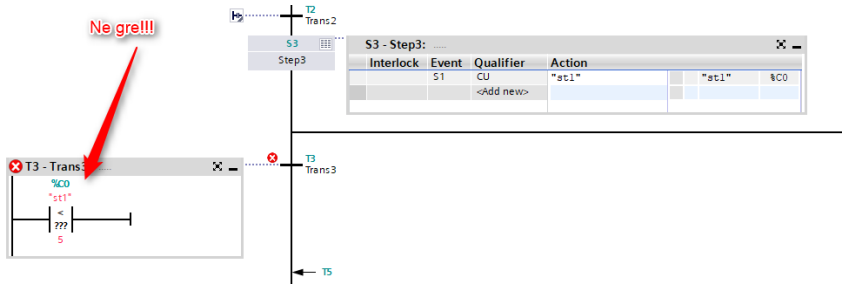


Ko se izvajata procesa v veji 1 in 2 hkrati, se program nadaljuje v S4 šele, ko se procesa v obeh vejah zaključita. Če je korak S3 že zaključen, T3 odprt, le ta čaka, da se proces odvije tudi v drugi veji. Torej dokler se Tm ne odpre, nato pa izvede Sm.

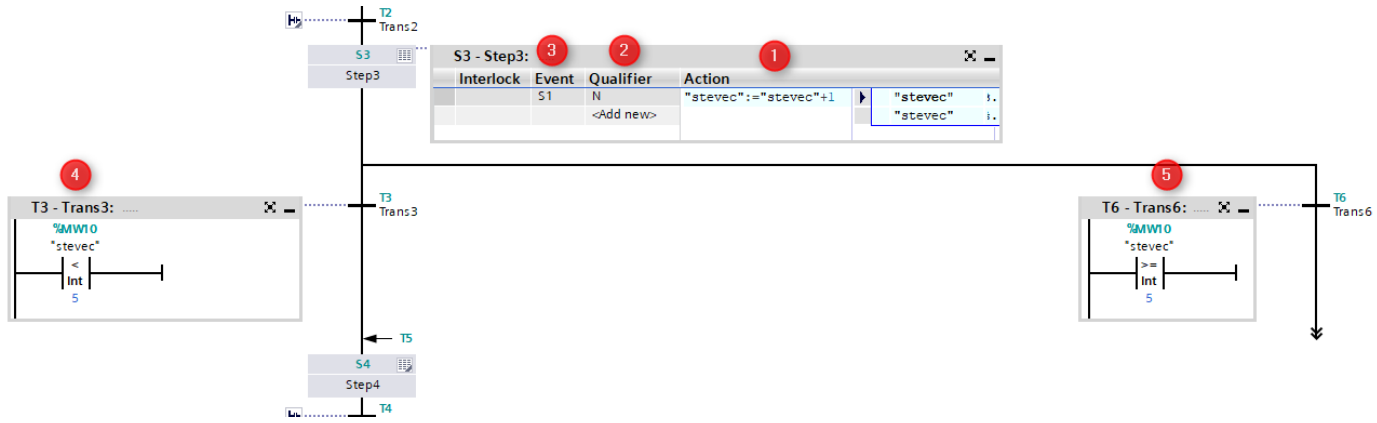
Korak S5 se izvede šele, ko so izvedeni S4, Sy, Sz.

12.5 Counterji

V graphu pa s števci žal ne moremo delati na tak način, zato je bolje, če si sami zgradimo svoj števec z matematičnimi operacijami (prištevanje, odštevanje)



Če želimo uporabljati komparator, moramo imeti v primerjavi število tipa INT, DINT ali REAL. Counter je 16bit število, katero pa vsekakor ni INT.



V zgornjo rešitev števca lahko opišemo na sledeč način.

1. V stolpcu action določimo da se bo stevec povečal za 1 z ukazom (stevec := stevec + 1) Števec sem izbral na naslovu MW10.
2. Qualifuer N določa, da se bo zadeva izvajala samo v tem koraku
3. Event določi, da se bo povečanje izvedlo samo 1x in sicer, ko program vstopi v korak 3. Če bi za event izbrali S0, bi to pomenilo, da bi se števec povečal, ko program zapusti korak3.
4. Komparator primerja števec z vrednostjo. Običajno se uporabi na tak način (po prvi veji do želene vrednosti, ko to vrednost presežemo pa po drugi veji. Paziti moramo le, da zajamemo vsa števila)
5. Pogoj, ko prepušča, ko števec preseže število

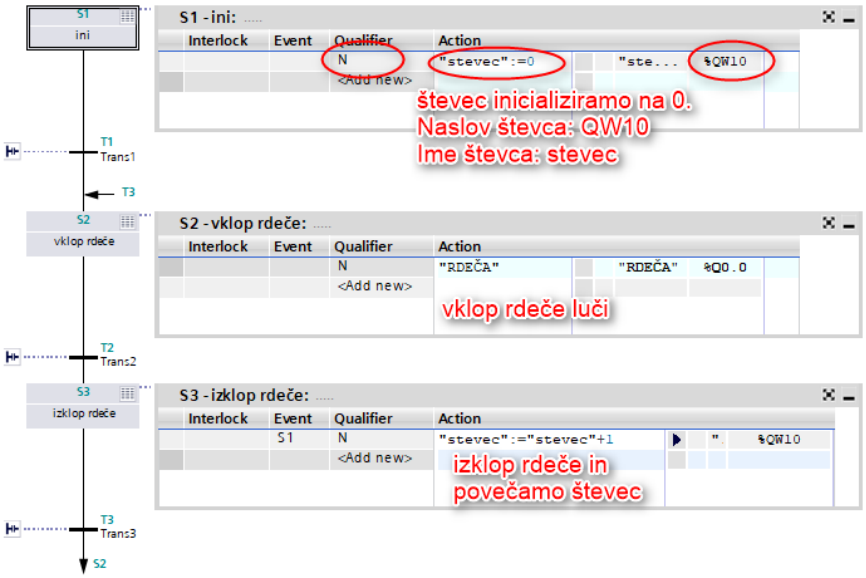
POZOR!!!, preden števec povečamo, ga moramo nekje postaviti tudi na začetno vrednost. Najbolje ga je v prvem koraku postaviti na 0 z ukazom stevec := 0

Primer:

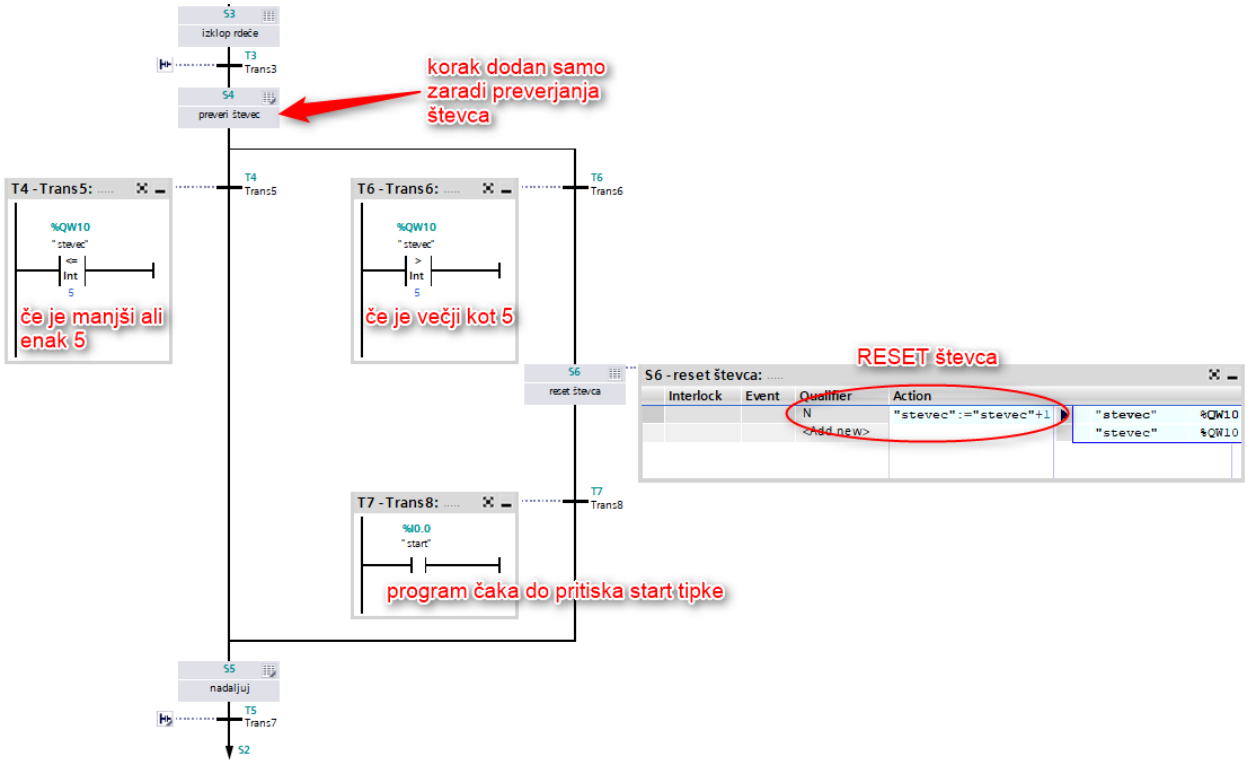
Želimo narediti program za utripanje luči. Ko luč 5x utripne, se utripanje ustavi. Utripanje ponovno poženemo s start tipko:

Spodnja slika prikazuje, da po pritisku na START, RDEČA luč začne utripati

Dodelajmo program, ki bo štel utripe (kolikokrat gre program čez vejo)

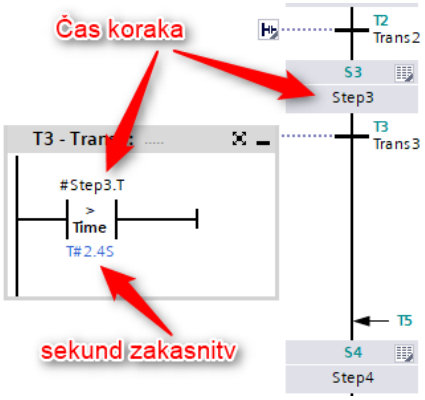


Po 5. utripih ustavimo števec



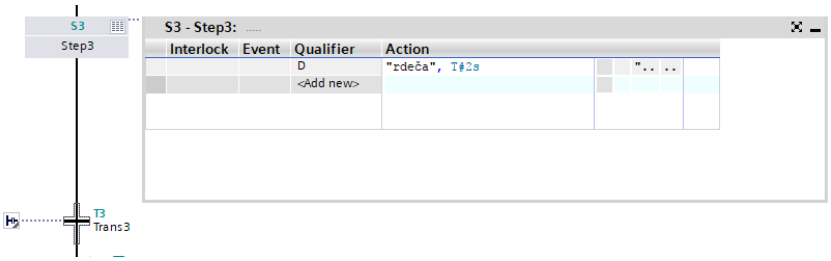
12.6 Timerji

Časovnike uporabljamo predvsem za to, da vsaj toliko časa ostanemo v določenem koraku.

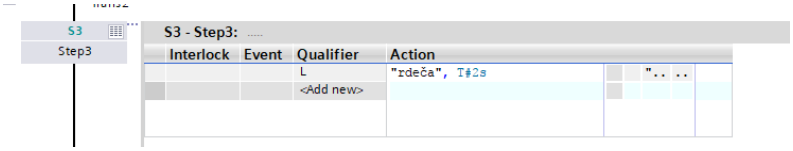


Primer prikazuje, da program čaka v step3 natanko 2,4 sekunde.

Naslednji timer pa je primeren za akcijo. Če uporabimo D, določenemu izhodu zakasnimo vklop za določen čas. V spodnjem primeru za 2 sekundi.



Če se uporabi L, pa se določen izhod vključi takoj ima pa definiran največji določen čas. V spodnjem primeru se rdeča luč ugasne po 2 sekundah koraka.

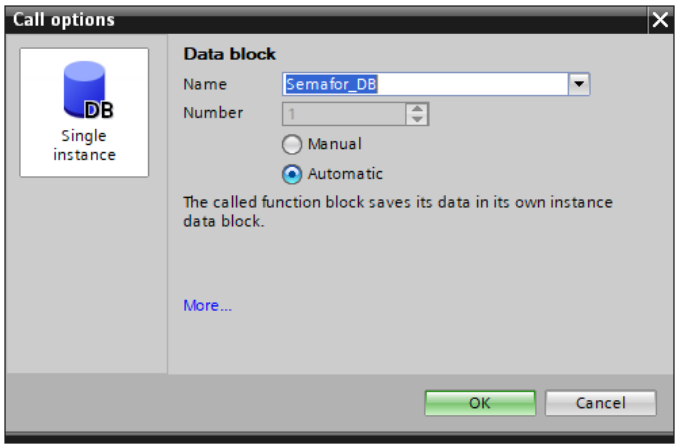


12.7 Arimetika

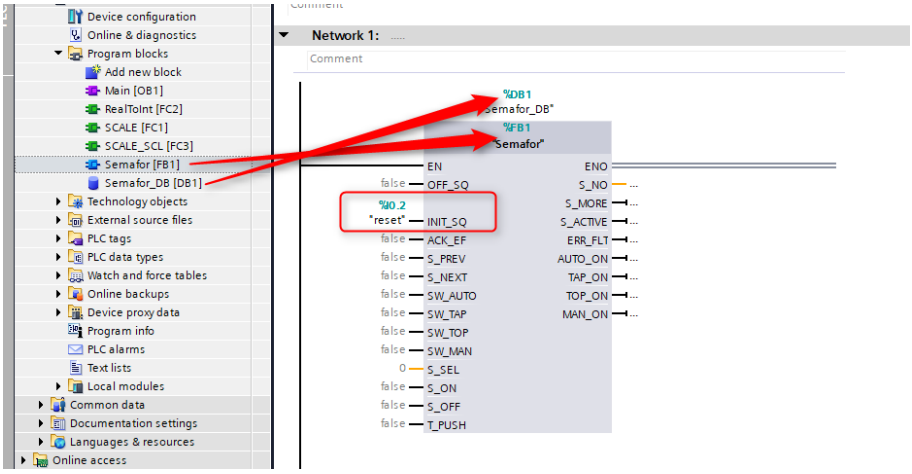
Za razna preračunavanje lahko uporabimo razne matematične funkcije. Včasih si pomagamo še s spominom za izračun vmesnih vrednosti. Na spodnji sliki je primer merjenja nivoja vode v rezervoarju. Ko vstopimo v step3, se vključi pumpa, ob enem pa računamo količino vode preko AI0. Na nam ni potrebno imeti meja 0-32767, le to število lahko pretvorimo v litre, s komparatorjem pa potem primerjamo litre.

12.8 Klic bloka

Blok katerega smo ustvarili, lahko kličemo iz MAIN(OB1) in sicer tako, da blok iz orodne vrstice samo povlečemo na črto v LAD. Ko storimo to, se pokaže naslednje okno



FB za svoje delovanju potrebuje tudi datablok (DB). V DB so shranjene začasne spremenljivke, vrednosti, stanje korakov, aktivni in neaktivni koraki… zaenkrat pustimo vse privzeto in kliknimo OK.



Kar dobimo v LAD, je naš FB Semafor, nad njim je naslov DB. Kar gre tukaj še omeniti je vhod INIT_SQ. Ta vhod v primeru, da se spremeni stanje iz '0' ➔ '1', resetira cel blok tako, da program skoči na prvi step (Initial step).

POZOR, ko želimo naložiti program na krmilnik, moramo prvič izbrati vse objekte v mapi Program Blocks, torej klikni na mapo Program blocks, ter vse skupaj prenesi na krmilnik.

12.9 Vaje

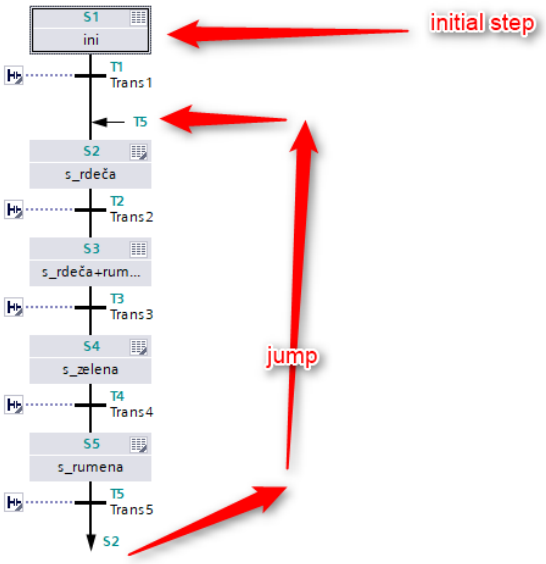
12.9.1 Semafor

Če še niste, ustvarite nov DB z imenom Semafor.

Izdelajmo semafor z naslednjimi koraki in sicer v vsakem koraku mora goreti kombinacija luči (Rdeča, Rumena, Zelena)

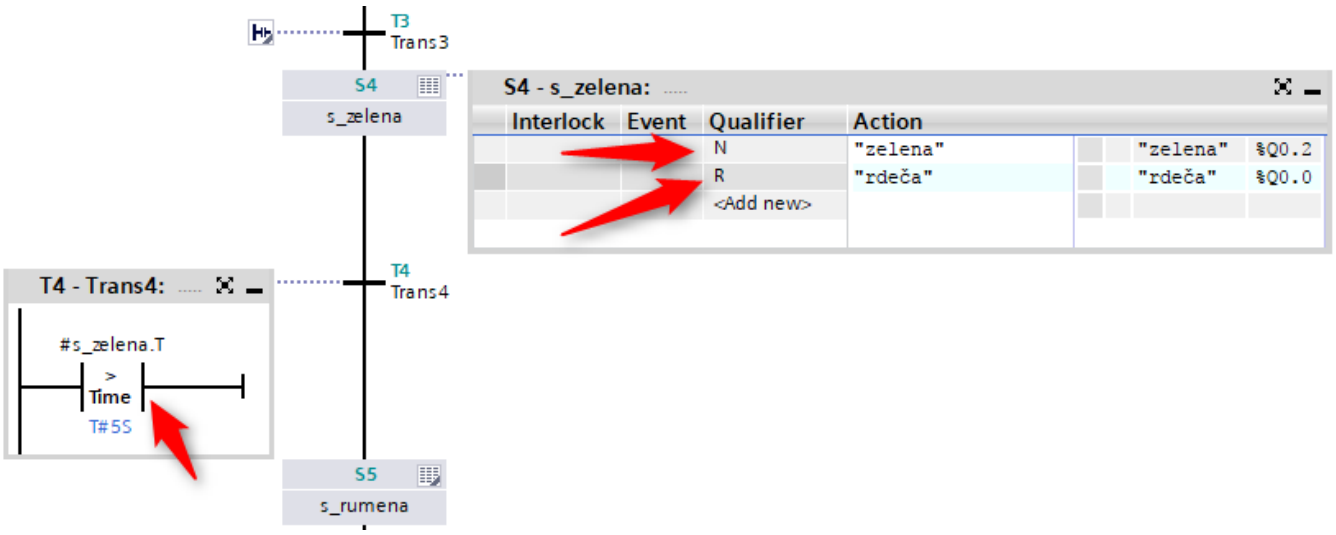
1. S start tipko naj se zažene semafor
2. RDEČA (5s)
3. RDEČA + RUMENA (1s)
4. ZELENA (5s)
5. RUMENA (1s)
6. JUMP na 2. korak

V program dodaj 5 korakov, tako kot je na spodnji sliki.



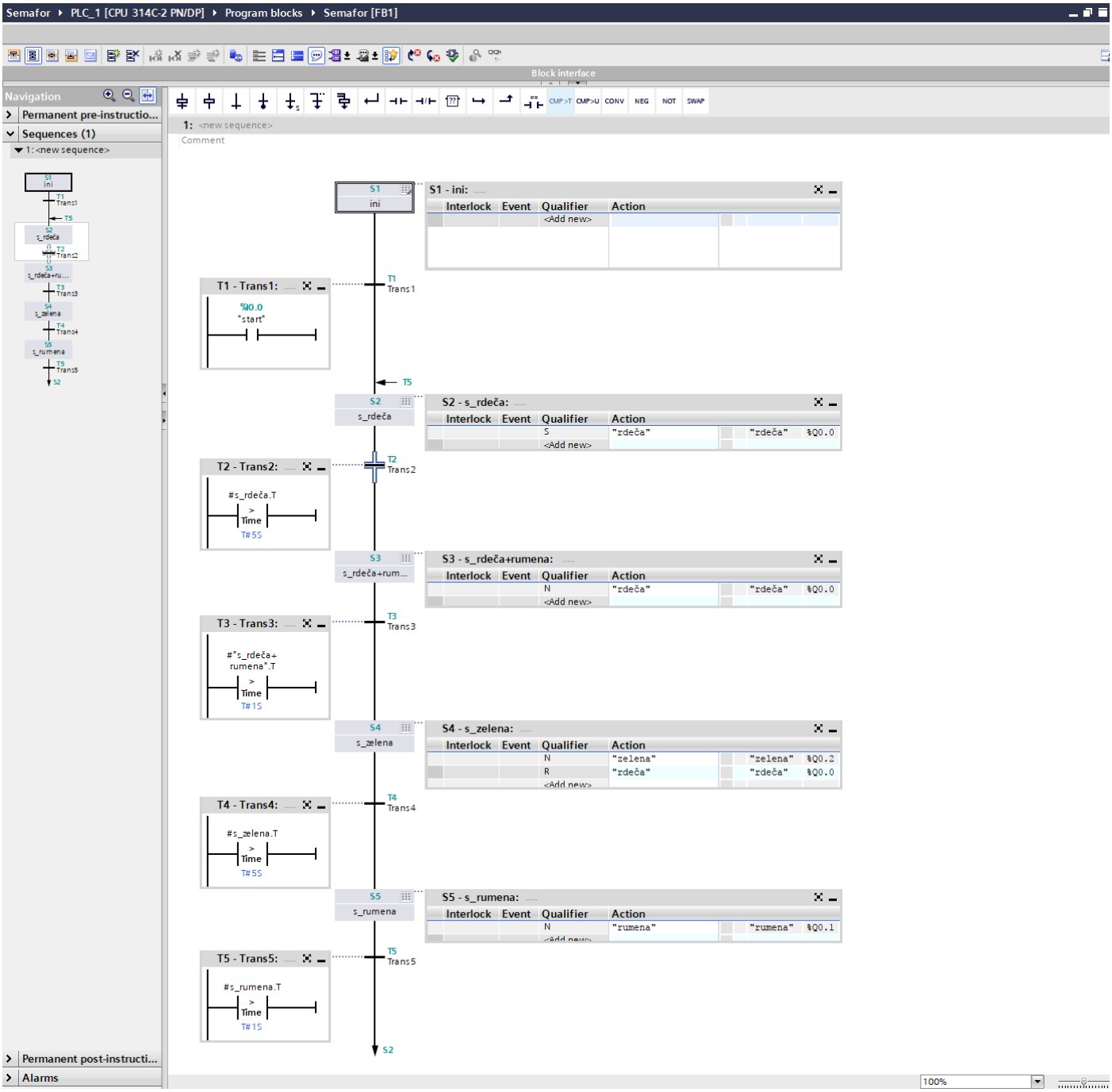
V našem primeru bomo uporabljali samo ukaze S (set), R (reset) in N (as long as step is active), zato moramo premisliti, kakšen ukaz bomo podali, da bomo imeli čimmanj dela. Med posameznimi koraki mora preteči določen čas. Le tega bomo določili s komparatorjem časa CMP>T, kar pomeni, da meri čas koraka in ko le ta preseže določen čas, naprej spusti lokično '1'.

Primer enega korak:



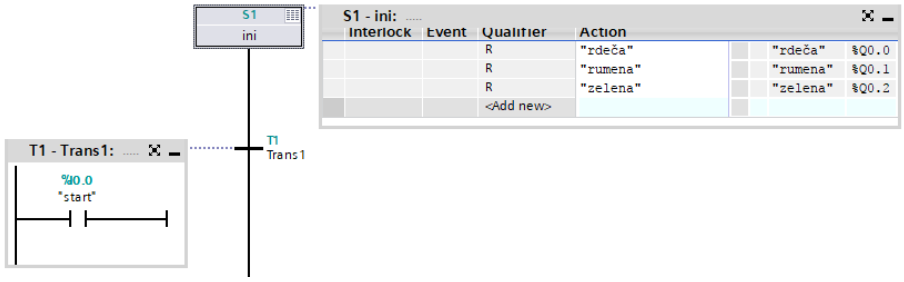
Določimo čas, ter za čas tega koraka vključimo zeleno luč, rdečo pa resetiramo, ker je iz koraka s2 ostala vključena.

Še rešitev celotnega programa



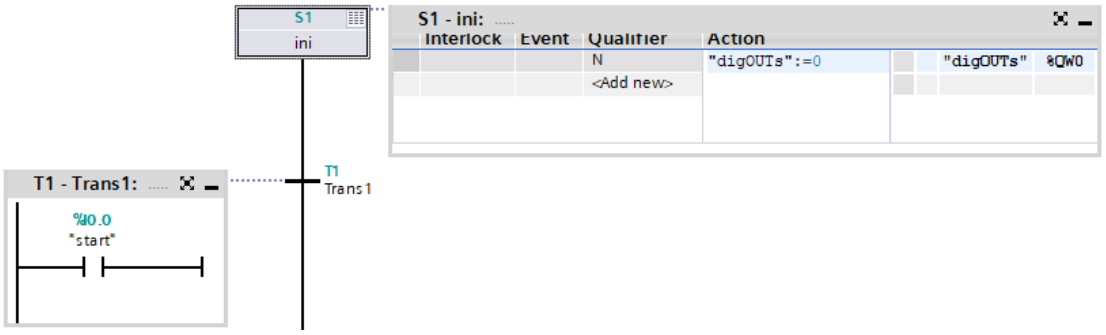
Ko testirate program, se ob pritisku na reset tipko vse luči ugasnejo, razen rdeča, če se takrat program nahaja v s2 ali s3, ker smo rdečo luč setirali. Če jo želimo v prvem koraku izključiti, jo moramo le tam resetirati. Tako lahko resetiramo vsak bit posebej, ali pa več bitov skupaj:

Primer vsakega posebej:

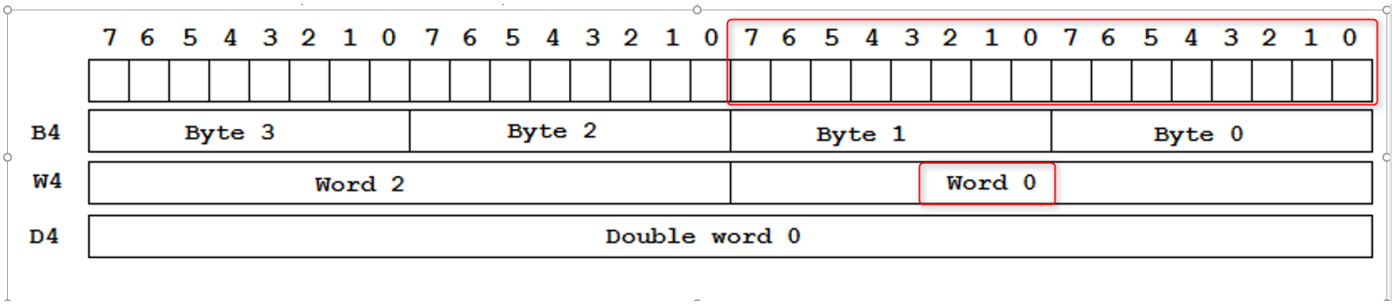


Kaj pa če imamo več signalov. V takem primeru pa lahko vse bite v izhodnem wordu postavimo na desetiško vrednost 0. Pri tem številu so vsi biti v obeh bajtih '0'.

Ukaz za to je **QW0 := 0** seveda se QW0 potaga, zato mu lahko spremenimo ime npr: digOUTs



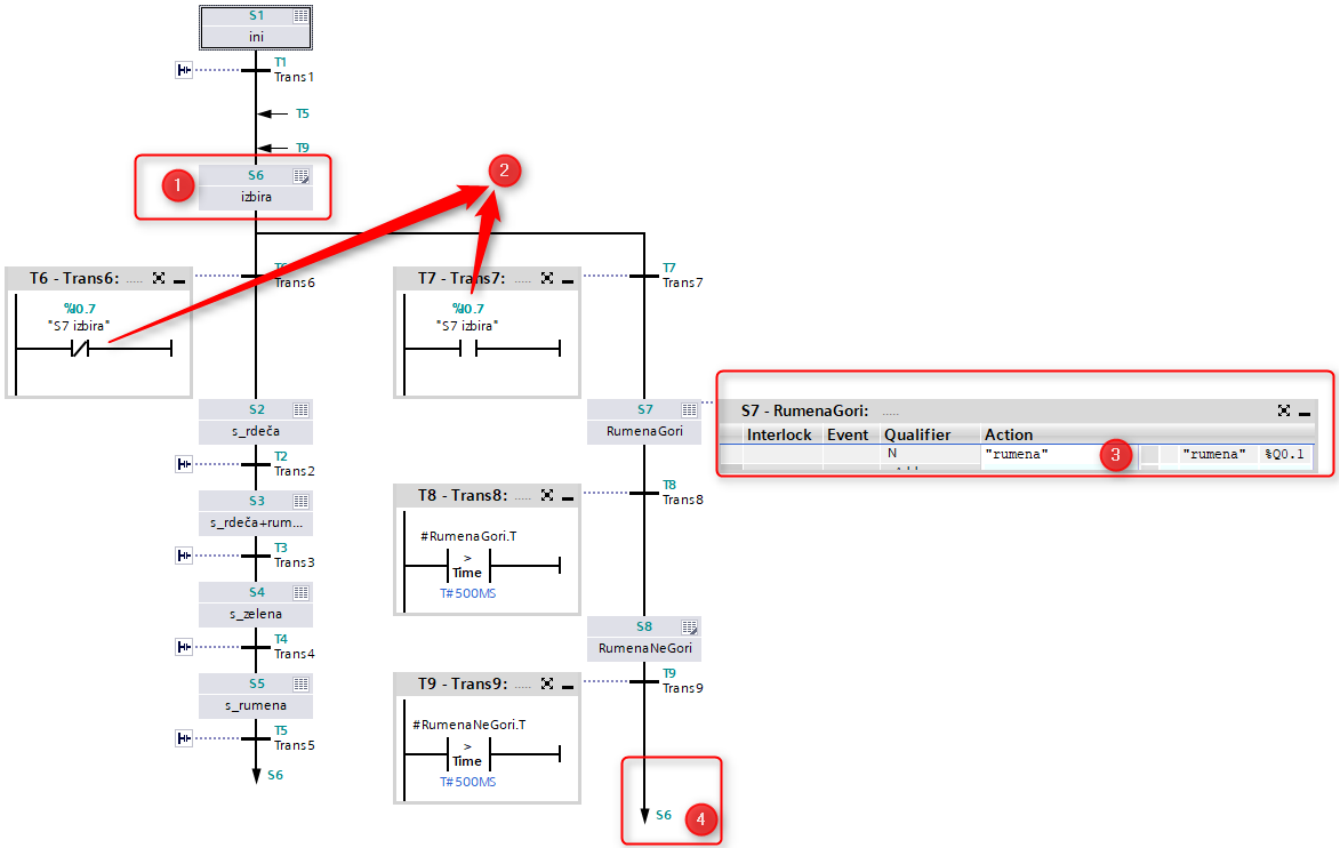
Še enkrat, na spodnji sliki je označen QW0. Torej je prvi naslov v wordu Q0.0, zadnji pa Q1.7.



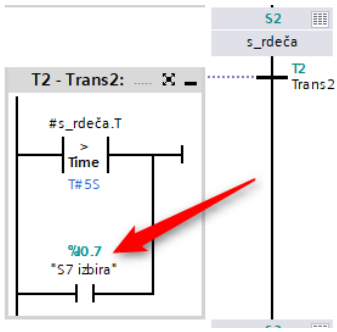
Nadgradnja naloge: Program želimo dodelati tako, da bo ob vklopu stikala »S7 Izbira« utripala rumena luč s frekvenco 1Hz. To pomeni, da 0.5s gori in 0.5s ne gori.

Iz spodnje slike poglejmo:

1. Vrinemo korak (nima akcije) kateri je namenjen samo izbiranju pogojev.
2. Pogoji po kateri veji se bo program nadaljeval, je odvisen od stikala »S7 izbira«, kateri izključuje veje med seboj. Če ni vključen gre program na S2, če pa je vključen pa na S7.
3. V koraku S7 gori rumena luč 0.5s v koraku S8 pa ni nobene akcije, ker smo v prejšnjem koraku rumeno vključili z ukazom N, kar pomeni, da se ob izstopu programa iz koraka luč sama ugasne.
4. Skok na izbiro: zopet se preveri, kakšno je stanje stikala.



Če bi radi, da se ob vklopu stikala S7 izbira semafor takoj preklopi v rumeno utripajočo, vzporedno vsem pogojem za čas vežemo še stikalo S7 izbira. Če bo stikalo vključeno, bo ob vstopu v korak pogoj že izpolnjen, torej bo luč blinknila za 10ms.



12.9.2 Vrtanje in sistem s tremi cilindri

Pregled simulacija

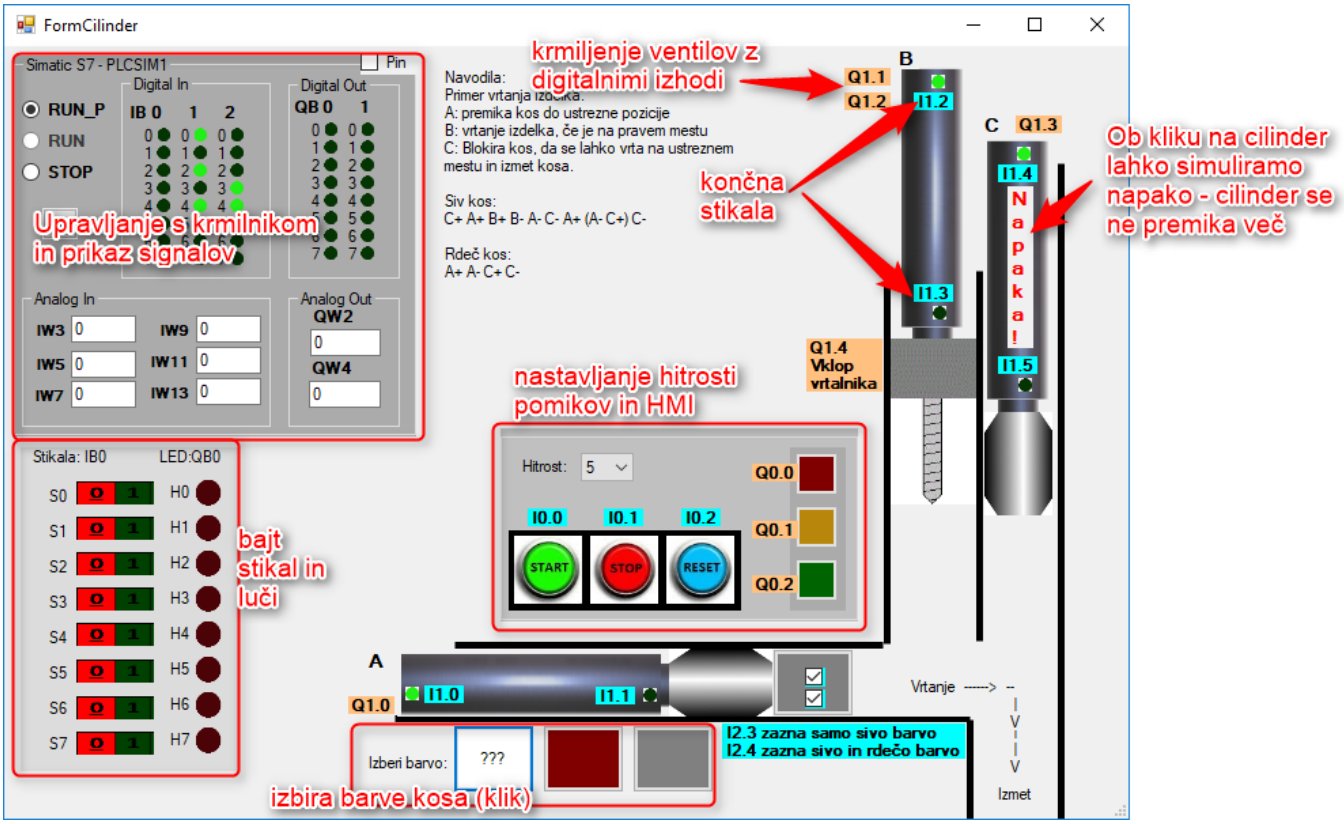


Tabela signalov:

Vhodni nslov	pomen	Izhodni naslov	pomen
I0.0	Start	Q0.0	RDEČA luč
I0.1	Stop	Q0.1	RUMENA luč
I0.2	Reset	Q0.2	ZELENA luč
I0.7	S7 - stikalo	Q0.3	Luč H3
I1.0	a0 - Cilinder A je skrčen	Q0.4	Luč H4
I1.1	a1 - Cilinder A je iztegnjen	…	…
I1.2	b0 – B je skrčen	Q0.7	Luč H7
I1.3	b1 – B je iztegnjen	Q1.0	A – iztegni cilinder A (monostabilen)
I1.4	c0 – C je skrčen	Q1.1	B – skrči cilinder B (bistabilen)
I1.5	c1 – C je iztegnjen	Q1.2	B – iztegni cilinder B (bistabilen)
I2.3	Obdelovalni kos je sive barve	Q1.3	C – iztegni cilinder C (monostabilen)
I2.4	Obdelovalni kos je rdeče ali sive barve	Q1.4	VRTALNIK - vklop

Najprej v mapi PLC Tags vpišimo vse signale:

Semafor ▸ PLC_1 [CPU 314C-2 PN/DP] ▸ PLC tags ▸ Default tag table [29]

Default tag table

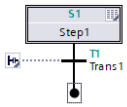
	Name	Data type	Address	Retain	Visibl...	Acces...	Comm
1	start	Bool	%I0.0		☑	☑	
2	stop	Bool	%I0.1		☑	☑	
3	reset	Bool	%I0.2		☑	☑	
4	s3	Bool	%I0.3		☑	☑	
5	s4	Bool	%I0.4		☑	☑	
6	s5	Bool	%I0.5		☑	☑	
7	s6	Bool	%I0.6		☑	☑	
8	s7	Bool	%I0.7		☑	☑	
9	a0	Bool	%I1.0		☑	☑	
10	a1	Bool	%I1.1		☑	☑	
11	b0	Bool	%I1.2		☑	☑	
12	b1	Bool	%I1.3		☑	☑	
13	c0	Bool	%I1.4		☑	☑	
14	c1	Bool	%I1.5		☑	☑	
15	siv kos	Bool	%I2.3		☑	☑	
16	rdeč ali siv kos	Bool	%I2.4		☑	☑	
17	RDEČA	Bool	%Q0.0		☑	☑	
18					☑	☑	
19	RUMENA	Bool	%Q0.1		☑	☑	
20	ZELENA	Bool	%Q0.2		☑	☑	
21	H3	Bool	%Q0.3		☑	☑	
22	H4	Bool	%Q0.4		☑	☑	
23	H5	Bool	%Q0.5		☑	☑	
24	H6	Bool	%Q0.6		☑	☑	
25	H7	Bool	%Q0.7		☑	☑	
26	A	Bool	%Q1.0		☑	☑	
27	B+	Bool	%Q1.1		☑	☑	
28	B-	Bool	%Q1.2		☑	☑	
29	C	Bool	%Q1.3		☑	☑	
30	VRTALMIK	Bool	%Q1.4		☑	☑	
31	<Add new>				☑	☑	

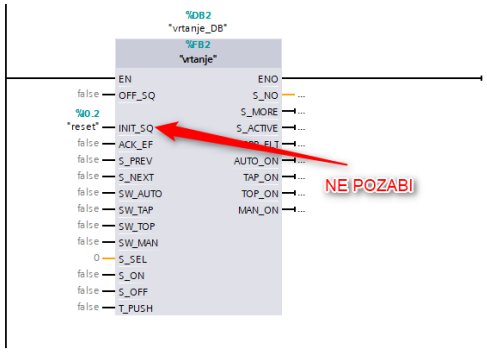
Naloga:

1. Program naj se začne izvajati ob pritisku na start tipko
2. Če je kos na začetnem mestu, se najprej iztegne C, da prepreči kosu, da bi se pomaknil nad izmet, ampak, da se zaustavi nad mestom vrtanja. Nato se iztegne A, kateri začne porivati kos na mesto vrtanja. Na mestu vrtanja nimamo končnih stikal, zato za pogoj uporabimo timer oz. časovnik. Preden spustimo vrtalnik, ga moramo najprej vključiti, nato iztegnemo B. Ko B pride do konca, se B umakne, nato se umakne A, nato C, nato se C iztegne, in na koncu skrči. Namig: Če je cilinder aktiven več kot 2 koraka, uporabi SET, drugače N.
3. Dodelaj program tako, da se bo ob pritisku na reset tipko naprava postavila v izhodiščni položaj (vsi cilindri skrčeni, vrtalnik ugasnjen (v točki signalizacija - sveti zelena luč)
4. Izdelaj program, da bo ločeval med barvama. V sivega vrta luknje, v rdečega pa ne, torej rdečega naj cilindri porinejo v izmet brez vrtanja.
5. Izdelaj signalizacijo: Če je naprava pripravljena na delovanje naj gori zelena luč. Če naprava deluje, naj gori rumena luč. Če pride do napake na cilindru, naj gori rdeča luč.
6. Dodelaj program tako, da bo vrtal v vse barve, če bo vključeno stikalo S7, drugače pa ločuje barve.
7. Dodelaj program tako, da bo štel koliko kosov je zvrtal. Če zvrta 5 kosov, pomeni, da je zabojnik poln. Le to signaliziraj z rumeno utripajočo lučjo. Ko operater zamenja zaboj, le to potrdi s start tipko in program naj nadaljuje z delom.
8. Dodelaj program tako, da se ob pritisku na stop tipko program ustavi. Program naj nadaljuje ob ponovnem pritisku na start tipko.
9. Dodelaj program, da se cilinder A, ko se kos pojavi na svojem mestu začne iztegovati z zakasnitvijo 1s.

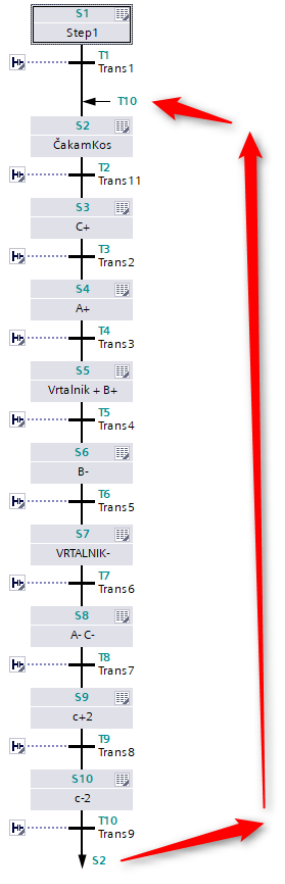
Začetek:

Dodajte nov FB in ga poimenujte »vrtanje. Sekvenco takoj zaključi z sequence end, ter shrani. Izvedi klic bloka v MAIN(OB1) LAD.



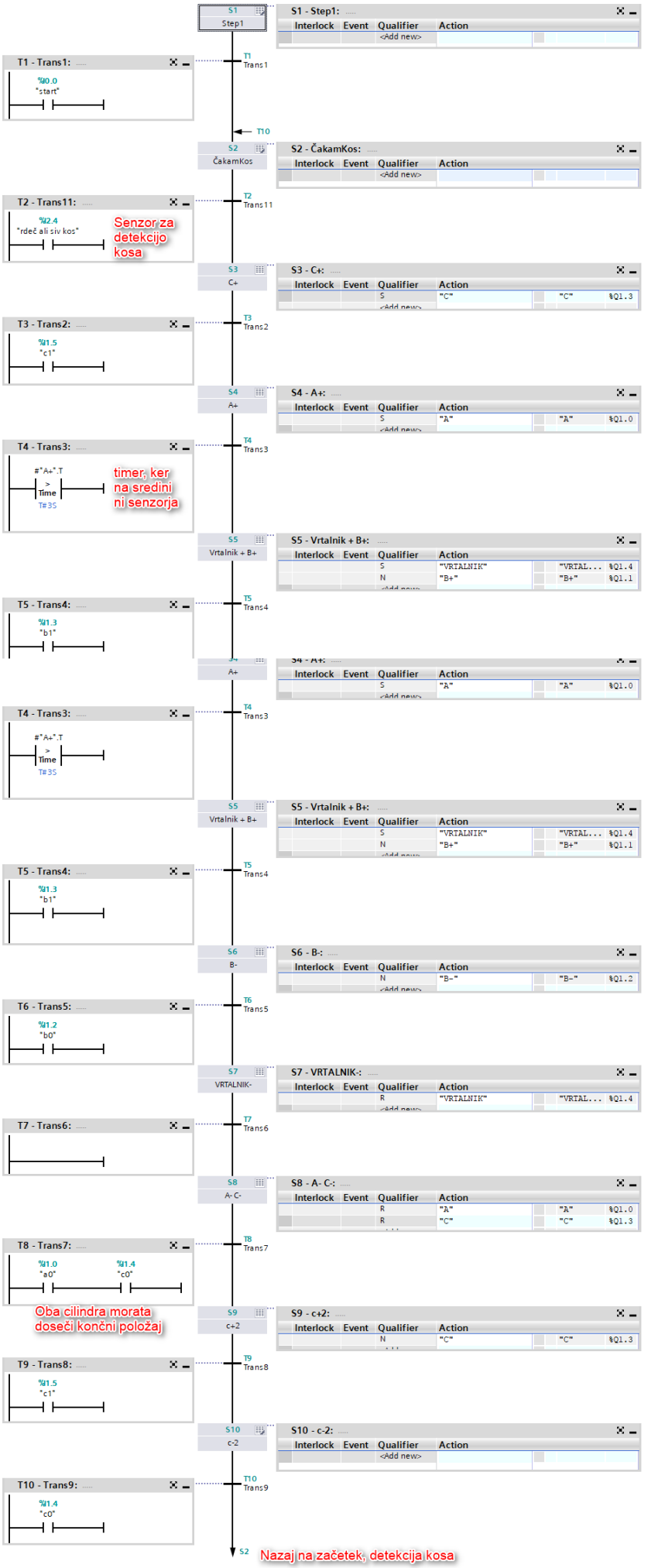


1. Najprej moramo definirati zaporedje cilindrov, pri čemer + pomeni iztegni, - pa pomeni skrči.
   Zaporedje: Čakam kos, C+, A+, (VRTALNIK+ B+), B-, VRTALNIK-, (A- C-), C+, C-
   Če si ogledamo korake, opazimo, da bomo cikel izvedli v 8 korakih. Ne pozabimo, da je prvi korak namenjen inicializaciji. Dodaj korake, jump na koncu in jih poimenuj



Sedaj pa v vsakem od korakov ključimo ali izključimo določene izhode, ter določimo pogoj za prehod v naslednji korak. Če so cilindri opremljeni s končnim stikali, jih uporabimo za pogoj, drugače uporabimo timer komparator.

Rešitev sekvence:


S1
Step1
S1 - Step1:
Interlock Event Qualifier Action
<Add new>
T1 - Trans1:
%I0.0
"start"
T1
Trans1
T10
S2
ČakamKos
S2 - ČakamKos:
Interlock Event Qualifier Action
<Add new>
T2 - Trans11:
%I2.4
"rdeč ali siv kos"
Senzor za detekcijo kosa
T2
Trans11
S3
C+
S3 - C+:
Interlock Event Qualifier Action
S "C" "C" %Q1.3
T3 - Trans2:
%I1.5
"c1"
T3
Trans2
S4
A+
S4 - A+:
Interlock Event Qualifier Action
S "A" "A" %Q1.0
T4 - Trans3:
#"A+".T
>
Time
T#3S
timer, ker na sredini ni senzorja
T4
Trans3
S5
Vrtalnik + B+
S5 - Vrtalnik + B+:
Interlock Event Qualifier Action
S "VRTALNIK" "VRTAL... %Q1.4
N "B+" "B+" %Q1.1
T5 - Trans4:
%I1.3
"b1"
T5
Trans4
A+
T4 - Trans3:
#"A+".T
>
Time
T#3S
T4
Trans3
S5
Vrtalnik + B+
S5 - Vrtalnik + B+:
Interlock Event Qualifier Action
S "VRTALNIK" "VRTAL... %Q1.4
N "B+" "B+" %Q1.1
T5 - Trans4:
%I1.3
"b1"
T5
Trans4
S6
B-
S6 - B-:
Interlock Event Qualifier Action
N "B-" "B-" %Q1.2
T6 - Trans5:
%I1.2
"b0"
T6
Trans5
S7
VRTALNIK-
S7 - VRTALNIK-:
Interlock Event Qualifier Action
R "VRTALNIK" "VRTAL... %Q1.4
T7 - Trans6:
T7
Trans6
S8
A- C-
S8 - A- C-:
Interlock Event Qualifier Action
R "A" "A" %Q1.0
R "C" "C" %Q1.3
T8 - Trans7:
%I1.0
"a0"
%I1.4
"c0"
T8
Trans7
Oba cilindra morata doseči končni položaj
S9
c+2
S9 - c+2:
Interlock Event Qualifier Action
N "C" "C" %Q1.3
T9 - Trans8:
%I1.5
"c1"
T9
Trans8
S10
c-2
S10 - c-2:
Interlock Event Qualifier Action
<Add new>
T10 - Trans9:
%I1.4
"c0"
T10
Trans9
S2
Nazaj na začetek, detekcija kosa

2.

13 HMI

13.1 Konfiguracija HMI

13.2 Povezovanje s PLC tagi

13.3 Alarmi

14 Webserver