Programski blok za upravljanje sekvencom

Teni, Matko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PROGRAMSKI BLOK ZA UPRAVLJANJE SEKVENCOM

Diplomski rad

Matko Teni

Osijek, 2019.

Sadržaj

1 UVOD

Automatizacija je proces kojim se nešto pravi automatskim, a također je stanje koje je rezultat tog istog procesa. Automatizacija je nastavak procesa mehanizacije, zato jer se automatskim može učiniti samo onaj proces koji je u dovoljnoj mjeri mehaniziran. Izraz automatizacija uveden je u automobilsku industriju oko 1946. godine kako bi opisao pojačanu uporabu automatskih uređaja i upravljanja u mehaniziranim proizvodnim linijama. Podrijetlo riječi pripisuje se D.S. Harderu, tada tehničkom menadžeru u Ford Motor Company. Automatizacija u širem smislu obuhvaća sve mjere i procese kojima se smanjuje udio ljudskog rada, opažanja i odlučivanja. Uobičajeno, automatizacija se može definirati kao izvođenje procesa pomoću programiranih naredbi u kombinaciji s automatskom kontrolom povratnih informacija kako bi se osiguralo pravilno izvršavanje naredbi. Rezultirajući sustav može raditi bez ljudske intervencije. Razvoj ove tehnologije postaje sve ovisniji o korištenju računala i računalnih tehnologija. Posljedično, automatizirani sustavi postaju sve sofisticiraniji i složeniji. Napredni sustavi predstavljaju razinu sposobnosti i performansi koja na mnogo načina nadmašuje sposobnost ljudi da sami obavljaju aktivnosti.

Industrijska automatika je sjecište znanja iz područja elektronike, strojarstva i računarstva. Cilj je stvaranje učinkovitog tehnološkog procesa. Kao primarni cilj industrijska automatika stvara mogućnost povećanja proizvodnje uz smanjenje troškova i poboljšanje kvalitete proizvoda. U konačnici, automatizacija proizvodnje rezultira većom produktivnosti i smanjenjem ljudske radne snage a time i mogućnost ljudske pogreške u proizvodnji ali i nestanak radnih mjesta. U automatskom radu postrojenja sekvence čine znatni dio regulacije rada. Sekvencijalnim upravljanjem izvršnim članovima u procesu zadajemo određeni niz naredbi. Procesi se moraju pokrenuti i zaustavljati sa prethodno precizno definiranim sekvencama akcija.

Tema ovog rada opisana je u pet poglavlja. U drugom poglavlju dan je kratki uvid u programabilne logičke kontrolere (PLC- e), način njihovog rada, koja je njihova primjena te princip njihovog funkcioniranja. U trećem poglavlju dan je uvid u sekvencijalno upravljanje i opisati će se pojednostavljena verzija jedne sekvence iz industrije. Četvrto poglavlje će predstaviti programsko rješenje. Prikazati će se kako je izvedena simulacija sustava te kojim programskim alatom. Prikazati će se i kako je ostvarena vizualizacija sustava te koji je programski alat bio potreban za

izradu. Zadnje poglavlje obuhvaća zaključak rada.

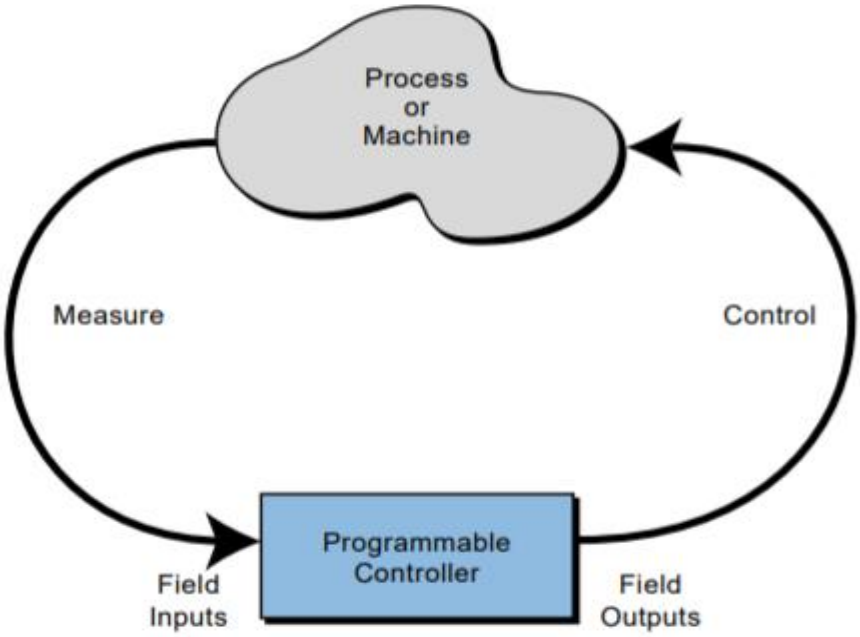
1.1 Zadatak diplomskog rada

Zadatak ovog diplomskog rada je izraditi sistemski blok za upravljanje sekvencom. Broj koraka sekvence treba biti definiran kao jedan od ulaza glavnog bloka. Drugi blok će upravljati pojedinim korakom sekvence i koristit će se onoliko puta koliko ukupno koraka ima sekvenca. Sve informacije o stanju sekvence potrebne za alarmiranje i vizualizaciju nalazit će se u glavnom bloku za upravljanje sekvencom (*Sequence Control Block*). Diplomski rad uključuje razvoj softvera i simulaciju u alatu Step7 i CoDeSys te vizualizaciju u alatu CoDeSys.

2 PROGRAMABILNI LOGIČKI KONTROLER – PLC

U industriji u prošlosti, klasična izvedba sustava upravljanja bila bi relejna logika. Relejna logika predstavlja releje i ostale komponente spojene žicama pomoću kojih se realizira neka logika upravljanja . No, postoji jedan veliki problem kod ovakvih sustava upravljanja, posebno ukoliko je sustav složen. Ukoliko se dogodi pogreška u logici, promjena upravljačke funkcije ili dodavanje novih komponenti mora se izvršiti prespajanjem žica [3]. Ukoliko je sustav velik i složen, što znači da će pogreške u logici, promjena upravljačkih funkcija ili dodavanje novih komponenata biti vrlo česti, tada ovaj način upravljanja postaje vrlo loš a sustav neiskoristiv. Upravo zbog toga javila se potreba da se uvede središnje računalo u sustave upravljanja. Središnje računalo za upravljanje sustavima regulacije naziva se programibilni logički kontroler (engl. *Programmable Logic Controller)* ili skraćeno PLC. Uvođenjem PLC-a u sustave upravljanja riješili su se prijašnji problemi. Ukoliko je došlo do pogreške u logici upravljanja, promjene upravljačke funkcije ili dodavanje novih komponenti jedino što se trebalo napraviti je promjena u programskom kodu. [3]. PLC –ovi imaju mogućnost spremanja instrukcija kao što su sekvence, brojači, aritmetike, manipulacija podatcima i komunikacija, za upravljanje industrijskim strojevima i procesima.

Slika 3.1. prikazuje konceptualni dijagram primjene PLC -a. [2]

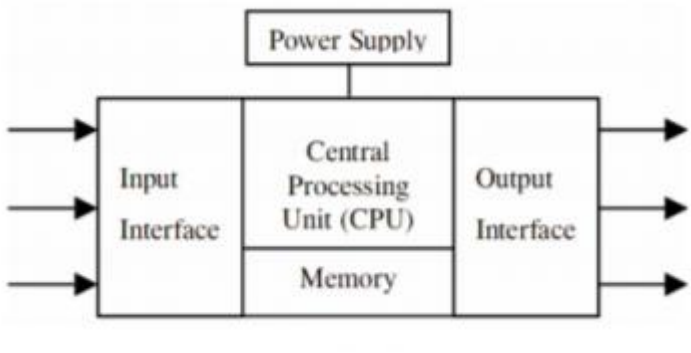


Sl. 2.1. Konceptualni dijagram primjene PLC –a [2]

Programabilni kontroleri se mogu jednostavno opisati kao industrijska računala sa specifično dizajniranom arhitekturom centralne jedinice (sami PLC) i sučeljem prema uređajima u polju. PLC-ovi su industrijski kontroleri čiji je dizajn zasnovan na principima jednostavnosti i praktične primjene [2].

Prvi PLC-ovi su bili samo zamjena relejne logike, te je njihova primarna funkcija bila odrađivanje sekvencijalnih operacija koje su prije bile implementirane relejima. U ove operacije ubrajamo ON/OFF upravljanje strojevima i procesima koji su zahtijevali operacije koje se ponavljaju, npr. transportna traka, uređaji za struganje, bušenje i slično. Kao što je već rečeno ova vrsta kontrolera je bila znatno poboljšanje u odnosu na releje. PLC-i su se lakše implementirali, koristili su znatno manje energije i prostora, imali su dijagnostičke pokazatelje koji su olakšavali pronalazak problema u sustavu i za razliku od releja mogli su se ponovno koristiti.[2]

Prema shemi koju možemo vidjeti na slici 2.2. PLC se sastoji od nekoliko funkcijskih cjelina.

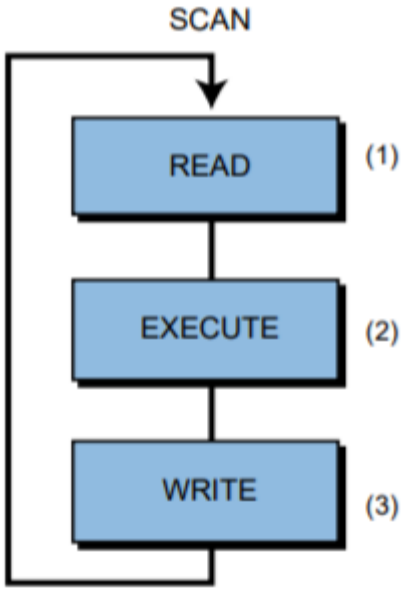


Sl. 2.2. Funkcijske cjeline PLC-a [3]

Središnji i glavni dio PLC-a je CPU (engl. *Central Processing Unit*). CPU je procesor, koji vođen zadanim programskim naredbama, izvodi osnovne radnje nad podatcima. Sljedeća funkcijska cjelina PLC-a je memorija. Najraniji PLC-i koristili su memoriju s magnetnom jezgrom i programiranje je bilo vrlo ograničeno zbog nedostatka memorije. U današnje vrijeme najčešće se koriste dvije vrste memorije EPROM (engl. *Erasable programmable read-only memory*) i RAM (engl. *Random Access Memory*)[3]. EPROM je tip memorijskog čipa koji se može po potrebi micati. Ovakva vrsta memorijskog čipa može zadržati podatke i nakon što se izgubi napajanje [3]. RAM je oblik primarne računalne memorije.

Izlazi i ulazi PLC-a mogu biti analogni ili digitalni. Moduli analognih ulaza najčešće su strujni ili naponski. Strujni analogni ulazi imaju raspon od 0-20mA ili od 4-20mA. Strujni ulazi koriste se ukoliko se strujna petlja priključuje na ulaz PLC-a. Isto tako, strujni analogni ulazi koriste se ukoliko se signal mora slati na velike udaljenosti. Zbog velike udaljenosti dolazi do pada napona zbog otpora kabla. Kada se koristi strujna petlja pad napona je zanemariv jer kroz petlju teče ista struja. Raspon od 4-20mA koristi se kako bi se moglo detektirati prekid žice, odnosno 0 mA, ali i zbog toga što se pomoću korištenja otpora od 250Ω vrlo jednostavno napravi konverzija u naponski ulaz raspona 1-5V. Naponski analogni ulazi raspona su 0-5V, 0-10V ili ±10V. Ovakvi ulazi koriste se u naponskim petljama koje se priključuju na ulaz PLC-a. Raspon koji se koristi ovisi o karakteristikama ulaznog signala. Kako bi PLC mogao obraditi signal koji dođe s analognog ulaza signal se mora diskretizirati. Kako bi se to ostvarilo koriste se A/D pretvornici koji uzorkuju ulazni signal i diskretiziraju ga. Kada PLC šalje izlazni signal na analogni izlaz potrebno ga je ponovno pretvoriti iz diskretnog u analogni. Tu se koriste D/A pretvornici. Digitalni ulazi pogodni su za spajanje sklopke, osjetila ili tipkala. Digitalni izlazi koriste se za spajanje pokretača motora ili malih motora. Digitalne signale potrebno je prilagoditi tako da budu razumljivi procesorskoj jedinici. Ta prilagodba uključuje optoizolaciju i filtriranje signala. Modul digitalnog ulaza najčešće je 24V DC. Signal digitalnih ulaza i izlaza sprema se kao 8-bitni podatak. Logičke jedinice i nule interpretiraju se na sljedeći način: -30-5V je logička 0, 13-30V je logička 1. Na slici 2.2. još možemo vidjeti kako je PLC-u potrebno napajanje kako bi radio.Osim ovih osnovnih funkcijskih cjelina, PLC-u se mogu dodavati razni moduli. Ovaj modularan dizajn velika je prednost kod složenijih sustava jer se vrlo lako mogu dodavati razni moduli za neke nove funkcije. [3]

Još jedna bitna stvar za navesti je način rada PLC-a. Tijekom operacija, CPU odrađuje tri procesa: (1) čita ili prihvaća ulazne podatke iz polja preko ulaznog sučelja, (2) izvršava upravljački program spremljen u memoriji sustava i (3) upisuje ili ažurira izlazne vrijednosti preko izlaznog sustava. Proces sekvencijalnog čitanja ulaza, izvršavanja programa u memoriji i ažuriranja izlaza naziva se skeniranje (engl. *scanning*). Slika 2.3 prikazuje grafički prikaz skeniranja. [2]



Sl. 2.3. Grafički prikaz jednog skeniranja [2]

3 SEKVENCIJALNO UPRAVLJANJE

Kada se u praksi susrećemo sa uključi/isključi (engl. *on/off*) funkcijama ili uključi/isključi stanjima neke opreme u procesima kojima upravljamo, takav tip upravljanja se naziva logičko ili prekidačko upravljanje. Stanju uključi najčešće odgovara logička jedinica, a stanju isključi logička nula. Jednostavnost ovakvog tipa upravljanja ga čini pogodnim za upotrebu u automatskom upravljanju strojeva i procesa u kojima se zahtijeva da proces ili stroj slijede sekvencu operacija. Primjena logičkog upravljanja u sekvencama rada dovela je do termina sekvencijalno upravljanje.[4]

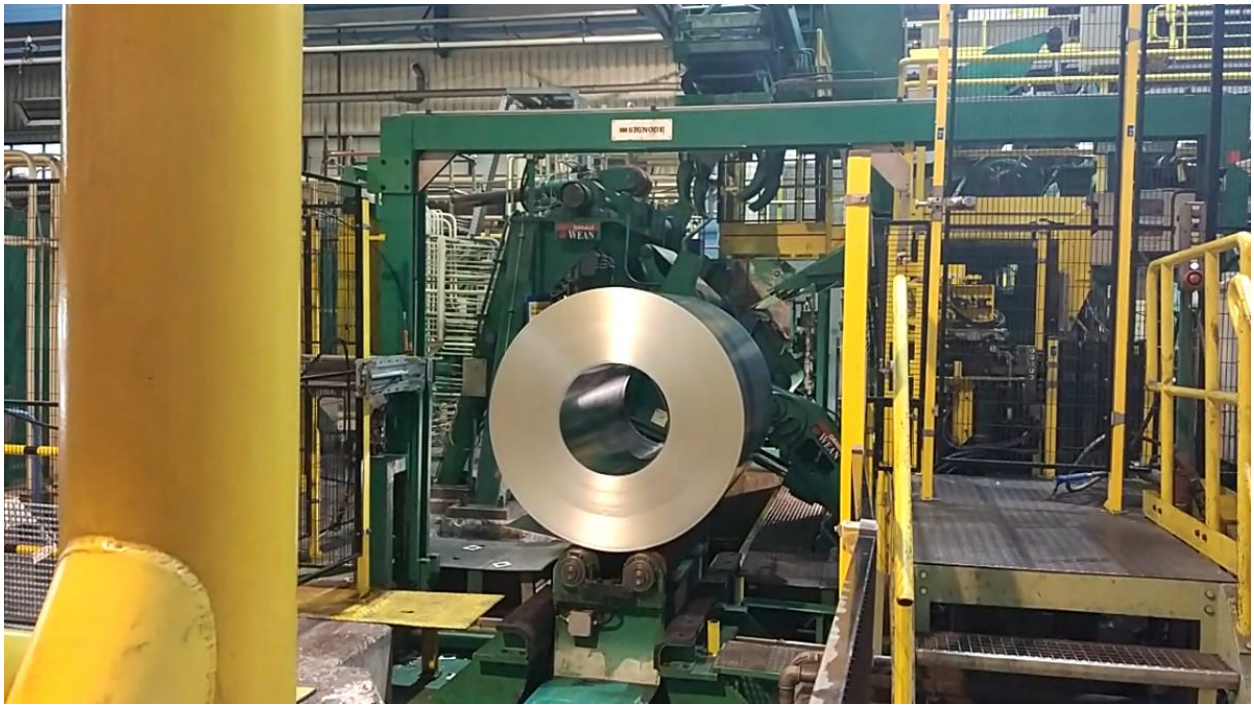
Sekvencijalno upravljanje može se realizirati pomoću elektromehaničkih releja, raznih pneumatskih i fluidnih komponenti, opreme na bazi poluvodiča (tranzistori, mikroprcesori…) i osobnih računala. Računala koja se koriste za realizaciju sekvencijalnog upravljanja su PLC-i. [4]

Sekvencijalno upravljanje ima sve veći značaj u upravljanju "kontinuiranim" procesima, jer oni nisu stvarno kontinuirani. Sustavi se moraju pokrenuti i zaustavljati sa prethodno precizno definiranim sekvencama akcija. Primjer za ovo je pokretanje termoelektrane koja radi na ugljen, za čije je puno opterećenje potrebno vrijeme od nekoliko sati. Taj prelazni režim, ili režim puštanja termoelektrane od stanja mirovanja do punog rada vodi se sekvencijalnim sustavom upravljanja. Isti je slučaj kada se elektrana zaustavlja; jednostavno mora biti ispunjen veliki broj sekvenci upravljanja prema točno definiranom redoslijedu i programu. [4]

Primjer jedne sekvence u industriji je transport obrađenog materijala na sedlo (engl. s*addle*) za skladištenje pomoću transportnog vozila (engl. *coil car*). Sekvenca koja će biti objašnjena je složenija u praksi nego što će u ovom radu biti izloženo. Na kraju samog proizvodnog procesa obrade ravnih proizvoda (engl. *flat products*) nalazi se jedan ili više namatača (engl. *recoiler*) tj. uređaja pomoću kojeg se traka (engl. *strip*) namata u oblik kotura (engl. *coil*). Na slici 2.1 možemo vidjeti primjer jednog namatača.

Sl. 3.1. Namatač

Konfiguracija tzv. izlaznog dijela ili sekcije (engl. *exit section)* ovisi o proizvodnom procesu. Na slici 2.2 možemo vidjeti izgled jednog kotura koji se transportira s transportnim vozilom. Može se primjetiti da transportno vozilo ima sedlo na kojem se prevozi kotur. Bitno je da je prilikom transporta kotur pozicioniran na središte sedla transportnog vozila zato što može doći do prevrtanja kotura s transportnog sedla.



Sl. 3.2. Transport kotura

U primjeru sekvence razmotriti ćemo slučaj kada se na izlazu proizvodne linije nalazi samo jedan namatač i vertikalna pozicija namatača je ista kao i vertikalna pozicija sedla za skladištenje. Materijal se namata na dio namatača zvan trn (engl. *mandrel*) koji ima oblik valjka koji se može proširiti (engl. *expand*) i skupiti (engl. *collapse*). Namotavanjem određenog promjera kotura na trn moramo s transportnim vozilom doći ispod samog trna. Transportno vozilo ima mogućnost kretanja u horizontalnom i vertikalnom smjeru. Raspon horizontalnog kretanja ovisi o konfiguraciji izlaza procesa, u primjeru pretpostavimo da ima horizontalni raspon od samog namatača te do sedla za skladištenje. U horizontalnom smjeru postavljamo vozilo u poziciju namatača te je bitno da prije ulaska u zonu namatača smanjimo vertikalnu poziciju tako da vozilo ima mogućnost prolaska ispod namatača. Ukoliko ne spustimo transportno vozilo može doći će do kolizije vozila i namatača. Vozilom dolazimo ispod namatača i podižemo vozilo sve dok ne dođe ispod kotura. Namještamo vozilo ispod kotura i vršimo skupljanje trna da se može vršiti daljnji transport. Mijenjamo horizontalnu poziciju prema izlazu tj. prema sedlu za skladištenje. Izlaskom iz zone namatača i ulaskom u zonu sedla za skladištenje povećavamo vertikalnu poziciju zato što prilikom dolaska na sedlo za skladištenje transportno vozilo mora biti iznad samog sedla. Dovođenjem vozila na horizontalnu poziciju sedla za skladištenje smanjujemo vertikalnu poziciju sve dok vozilo ne dođe na vertikalnu poziciju ispod sedla. Ovim postupkom smo prilikom smanjenja vertikalne pozicije kotur spustili na sedlo za skladištenje. Nakon toga sekvenca kreće

ispočetka ukoliko su ispunjeni uvjeti za pokretanje sekvence. Sekvenca započinje s radom na ručni ili automatski zahtjev. Prikazana sekvenca nešto pojednostavljena te je u praksi puno više čimbenika koji ograničavaju kada i koje akcije smijemo raditi.

4 PROGRAMSKO RJEŠENJE

Kao što je već napisano zadatak ovog diplomskog rada je izraditi sistemski blok za upravljanje sekvencom. Diplomski rad uključuje razvoj softvera i simulaciju u alatu Step7 i CoDeSys te vizualizaciju u alatu CoDeSys. Programsko rješenje podijeljeno je u dva dijela. U prvom dijelu opisano je programsko rješenje u SIMATIC Step 7 programskom okruženju te korisničko sučelje u SIMATIC WinCC programu. Korisničko sučelje (engl. *Human Machine Interface* –HMI) u WinCC-u nije dio zadatka ovog diplomskog rada ali je realizirano zbog bolje i jednostavnije primjene programskog bloka. HMI nam omogućava bolju konfiguraciju bloka u pogledu jednostavnosti mijenjanja pojedinih parametara i praćenja rada samog bloka u odnosu na praćenje samo u Step 7 programskom alatu. Drugi dio ovog poglavlja sadrži programsko rješenje u alatu CoDeSys te vizualizaciju u programskom alatu CoDeSys.

4.1 SIMATIC Step 7

U radu je prikazan samo dio mreža (engl. *network)* koje su korištene unutar bloka. Programsko rješenje je pisano u LAD (skraćeno od engl. *Ladder Logic*) logici i STL (skraćeno od engl. *Statment List*) logici. Ideja programskog bloka za sekvencu je da je univerzalan i da se može primijeniti u različitim sekvencama u industriji. Osnovna funkcija bloka je kontrolirani prelazak iz jednog koraka u drugi uz određene uvjete:

1. Pomoću varijable IN_StepNo određujemo koliko će koraka sekvenca podržavati, programski blok je ograničen na 8 koraka.
2. Za pokretanje određenog koraka moraju biti ispunjeni uvjeti (engl. *permissive*) za pokretanje tog koraka, u tu svrhu se koriste varijable IN_PermS1,IN_PermS2,…,IN_PermS8; npr. prije pokretanja koraka moraju raditi pumpe, grijači, rashladni sustav, alarmi su isključeni i slično.
3. Uvjet se može zaobići (engl. jump) pomoću *jump* varijabli, npr. uključen je alarm za povišenu temperature tekućine, temperature nije na kritičnoj razini pa se taj uvjet može zaobići
4. Broj koraka sekvence određen je varijablom IN_SeqNo.
5. Trajanje svakog koraka je vremenski određeno, prijelaz iz koraka u korak se ne može obaviti ukoliko nije prošlo minimalno vrijeme u prijašnjem koraku.
6. Ukoliko trajanje koraka prijeđe maksimalno zadano vrijeme a nije ispunjen uvjet za prijelaz u sljedeći korak, sekvenca se zaustavlja i prestaje s radom (engl. *timeout*).

7. Pomoću varijable IN_Suspend sekvencu postavljamo u obustavljeno stanje (odbrojano vrijeme u određenom koraku se zaustavlja) i ne može se prijeći u sljedeći korak.
8. Pomoću varijable IN_PermLostMask možemo izbjeći prekid sekvence u određenom koraku zbog gubitka uvjeta za taj korak. Uvjeti moraju biti ispunjeni prije pokretanja određenog koraka. Nakon pokretanja ti uvjeti više ne moraju biti zadovoljeni.
9. Varijabla IN_PermS9 služi za pokretanje "koraka" za prekid (engl. *abort*) u kojem sekvenca prelazi u stanje prekida ako je bilo koji uvjet za prekid ispunjen.

Prema tablici 4.1 možemo vidjeti popis ulaznih varijabli programskog bloka.

Tab. 4.1 Popis svih ulaznih varijabli programskog bloka

ULAZNE VARIJABLE		
Naziv	Tip podatka	Opis
IN_StepNo	INT	Broj koraka (eng. step) sekvence
IN_PermS1 . . IN_PermS8	INT	Uvjeti za pokretanje prvog-osmog koraka sekvence
IN_PermS9	INT	Uvjeti za prekid sekvence
IN_PermLostMask	BYTE	Ovisno o vrijednosti ove varijable prekidamo sekvencu ukoliko se izgube uvjeti za određeni korak sekvence ili se nastavlja rad sekvence iako su uvjeti izgubljeni
IN_SeqStart	BOOL	Varijabla pomoću koje pokrećemo rad sekvence
IN_Suspend	BOOL	Varijabla pomoću koje pauziramo rad sekvence
IN_CmdHMI	ANY	Pokazivač tipa ANY na adresu početne varijable podatkovnog bloka za

		naredbe koje dolaze s HMI-a
IN_AlarmHMI	ANY	Pokazivač tipa ANY na adresu prve varijable podatkovnog bloka u koji se spremaju varijable za alarmiranje na HMI-u
IN_StatHMI	ANY	Pokazivač tipa ANY na adresu prve varijable podatkovnog bloka u koji se spremaju varijable o svim statusima koji se prikazuju na HMI-u
IN_ProcHMI	ANY	Pokazivač tipa ANY na adresu prve varijable podatkovnog bloka u koji se spremaju sve procesne vrijednosti za prikazivanje na HMI-u

Sve ulazne varijable imaju prefiks IN_, slijedno tome sve izlazne varijable imaju prefiks OUT_. U tablici 5.1 mogu se uočiti varijable IN_CmdHMI, IN_Alarm, IN_StatHMI, IN_ProcHMI, koje su određene standardom za izradu HMI-a. Prilikom izrade HMI (skraćeno od engl. *Human Machine Interface*) iz Step-a 7 uzimaju se u obzir samo varijable potrebne za izradu HMI-a. Varijable se dijele na pojedine grupe ovisno o vrsti varijabli i infomacijama koje te varijable nose. U ovom radu varijable su podijeljene na četiri grupe:

1. CmdHMI (skraćeno od engl. *Commands from HMI*):
    - Naredbe koje se primaju s HMI-a. U ovom radu su to *jump* uvjeti, maksimalno i minimalno vrijeme za svaki korak sekvence.
2. AlarmHMI (skraćeno od engl. *Alarms to HMI*):
    - Sve varijable potrebne za signalizaciju i prikaz alarma i upozorenja na HMI-u, alarmi za *timeout* svakog koraka I alarm ako je sekvenca u prekidu.

3. StatHMI (skraćeno od engl. *Status to HMI*):
    - Svi važni statusi koje želimo prikazati na HMI-u, statusi uvjeta za svaki korak sekvence i status svakog koraka.
4. ProcHMI (skraćeno od engl. *Process Monitor to HMI*):
    - Sve važne procesne veličine koje prikazujemo na HMI-u, trenutno vrijeme u svakom koraku sekvence.

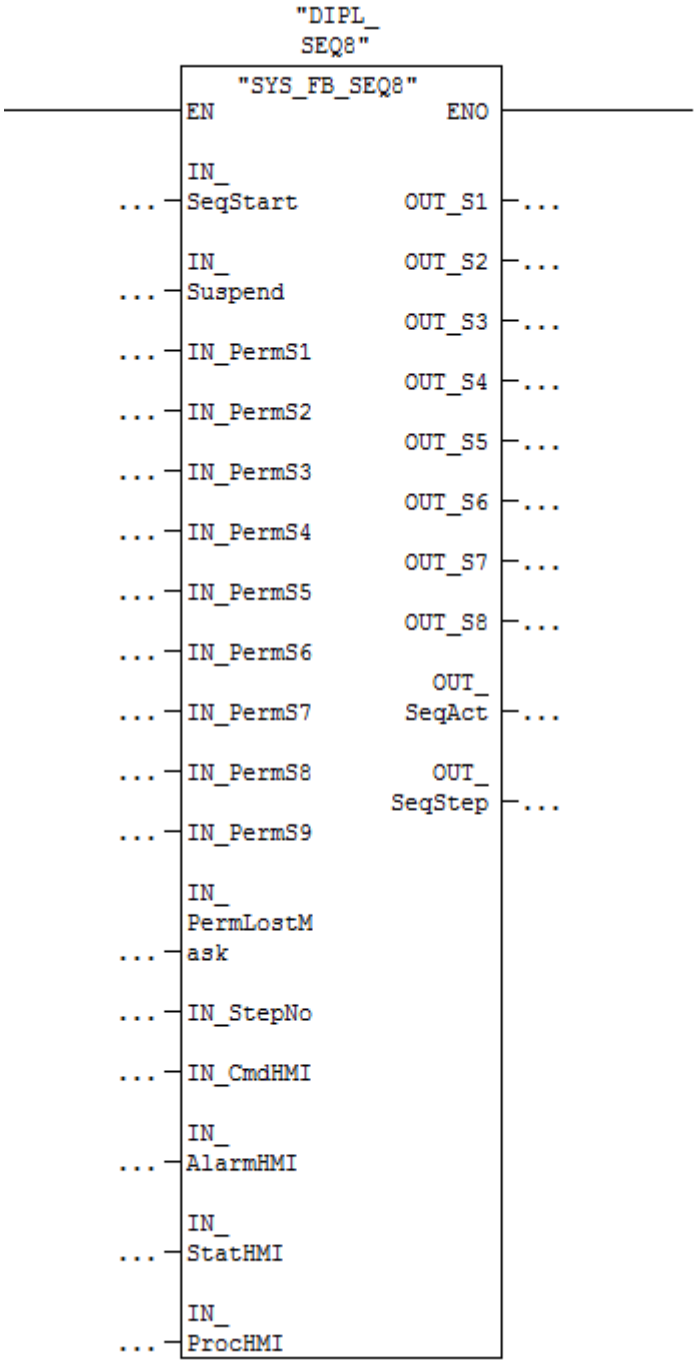
Za navedene grupe podataka kreiramo zasebne strukture u podatkovnim blokovima (engl. *Data Block*- skraćeno DB) koji služe za komunikaciju između PLC-a i HMI-a.

U tablici 5.2 možemo vidjeti popis izlaznih varijabli programskog bloka.

Tab. 4.2. Popis svih izlaznih varijabli programskog bloka

IZLAZNE VARIJABLE		
Naziv	Tip podatka	Opis
OUT_S1 . . OUT_S2	BOOL	Status prvog-osmog koraka
OUT_SeqAct	BOOL	Status sekvence (aktivna/ne aktivna)
OUT_SeqStep	INT	Prikaz trenutnog aktivnog koraka

Na slici 4.1 prikazan je izgled programskog bloka unutar Step 7 programskog okruženja.

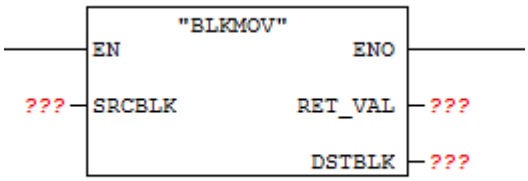


Sl. 4.1.Izgled programskog bloka u Step 7 programskom okruženju

Programsko rješenje u Step-u 7 možemo podijeliti na tri dijela:

1. Učitavanje svih ulaznih veličina izvan programskog bloka (dolazne veličine s HMI-a i iz drugih funkcija/funkcijskih blokova)
2. Kontrola rada sekvence
3. Ispisivanje svih izlaznih veličina i veličina potrebnih za prikaz na HMI-u

U prvom dijelu programskog rješenja učitavamo dolazne vrijednosti, u našem slučaju to su vrijednosti iz DB-a gdje su definirane sve komande koje dolaze sa HMI-a. Za mali broj varijabli ovaj proces je jednostavan i vrši se na način da se učitava jedna po jedna varijabla. Za puno varijabli to postaje nepregledno i vremenski zahtjevno. U radu su definirane varijable sa/prema HMI-u kao strukture podataka za razliku od jednostavnijeg pristupa definiranja pojedinačnih varijabli. Struktura podataka je skup varijabli koje mogu biti različitog tipa, grupirane pod zajedničkim imenom. Strukture se vrlo jednostavno mogu kopirati upotrebom BLKMOV funkcije (slika 4.2). U našem primjeru umjesto definiranja 80 zasebnih variabili i kasnijeg pristupu svakoj od njih kreirane su 4 strukture i sa 4 poziva BLKMOV funkcije sve su informacije primljene odnosno poslane na HMI.



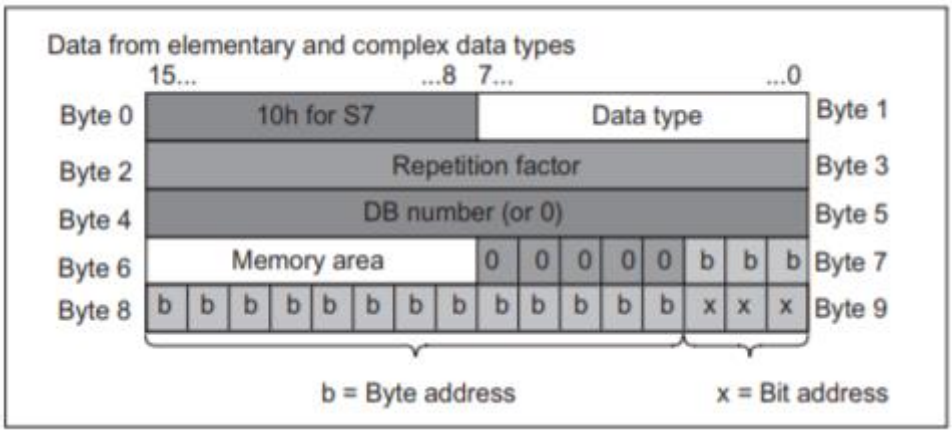
Sl. 4.2. Sistemska funkcija SFC20 BLKMOV

Funkcija ima jednu ulaznu i dvije izlazne vrijednosti. SRCBLK I DSTBLK varijable su tipa ANY, a RET_VAL varijabla je tipa INT. SRCBLK označava početnu adresu od koje se vrši prijenos podataka, uz početnu adresu moramo definirati i dužinu podataka za koju se kopiraju podatci. Sličan princip vrijedi i za DSTBLK, definiramo početnu adresu od koje funkcija treba vršiti upis podataka te se podatci upisuju za dužinu definiranu u SRCBLK. RET_VAL je varijabla tipa INT koja nam služi za kontrolu izvršenja funkcije. Ukoliko se dogodila greška prilikom unosa varijabli RET_VAL poprima određenu vrijednost (engl. *error code*). Ukoliko je vrijednost jednaka 0, nema greške.

U gotovo svakom industrijskom postrojenju koriste se desetci raznih sekvenci. Prednosti korištenja sistemskog bloka sa definiranim strukturama podataka koje se izmjenjuju sa HMI-em donosi sljedeće prednosti:

- Manja mogućnost nastanka pogreške prilikom definiranja varijabli jer se jednostavno cijela struktura kopira i samo se promjeni ime strukture dok varijable unutar strukture ostaju nepromijenjene.
- Brzi razvoj novih sekvenci.
- Budući da je na strani PLC-a sekvenca standardizirana samim time moguće je standardizirati i grafički prikaz sekvence na HMI-u.
- Postiže se standardizacija sekvence na razini cijelog postrojenja što olakšava upravljanje sekvence sa strane operatera. Naredbe za kontrolu toka sekvence su jednake za sve sekvence.

Iz tablice 5.1. može se uočiti da je varijabla IN_CmdHMI tipa ANY. ANY tip podatka se definira pomoću 10 *byte*-ova te prilikom sastavljanja varijabli tipa ANY moramo osigurati da je svih 10 *byte*-ova ispravno postavljeno zato što blok čita cijeli sadržaj varijable. Na slici 5.2 može se vidjeti što predstavlja svaki od 10 *byte*-ova u bilo kojoj varijabli tipa ANY.



Sl. 4.3. Struktura ANY tipa podatka

Opis byte-ova varijable tipa ANY:

- *Byte* 0 – po standard Step-a 7 vrijednost ovog byte-a mora biti postavljena na B#16#10.
- *Byte* 1 (*Data type*)- vrijednost određuje s kojom vrstom podataka radimo, npr. želimo prebaciti 10 varijabli tipa WORD, moramo ovu vrijednost postaviti na B#16#04 za taj tip podatka.

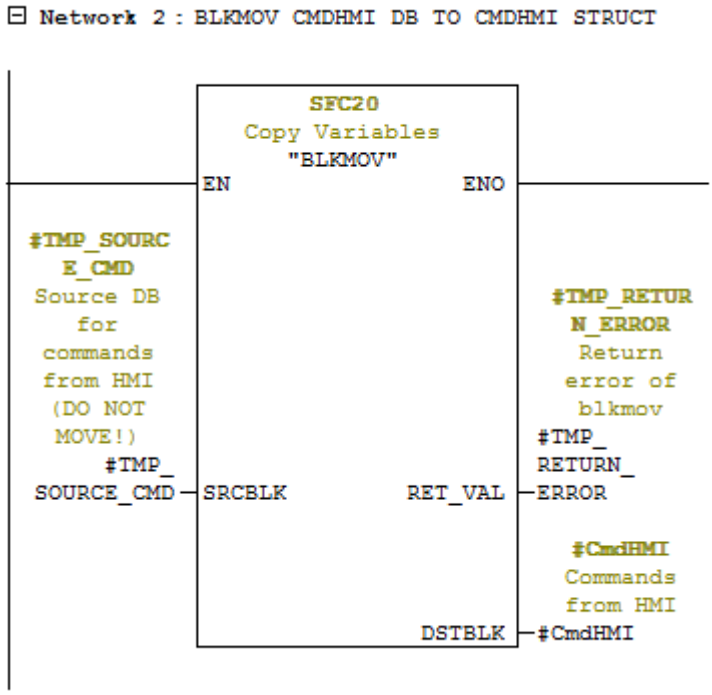
- *Byte* 2 i *Byte* 3 (*Repetition factor*)- određuje količinu podataka zadanog tipa koja će se kopirati. Ako uzmemo prethodni primjer ova vrijednost mora biti postavljena na 10
- *Byte* 4 i *Byte* 5 (*DB number*)- ukoliko se kopiranje vrši iz nekog DB-a u ovaj WORD upisujemo broj tog DB-a.
- *Byte* 6 (*Memory area*)- vrijednost određuje u ili iz koje vrste memorijskog prostora ili u koju vrstu memorijskog prostora se vrši kopiranje podataka. U slučaju ovog programskog rješenja to je vrijednost B#16#84 (16#84 je oznaka za DB memorijski prostor).
- Zapis adrese na koju se kopira je u formatu *Byte.Bit.*
- Bitovi 9.3 – 9.7, 8.0-8.7 te 7.0-7.2: služe za zapis *byte* adrese na koju ili iz koje se kopiraju podatci.
- Bitovi 9.0-9.2 služe za zapis bit adrese na koju ili iz koje se kopiraju podatci.

U radu je korišteno dinamičko definiranje ANY varijable, tako što je definirana početna adresa i duljina podataka koje želimo kopirati. Pojedine vrijednosti bitova/*byte*-ova ANY varijable spremljene su u lokalne varijable, tj. u TEMP (skraćeno od eng. *Temporary*) memorijski prostor. Na taj način moguće je programski manipulirati adresom i dužinom pokazivača tokom izvođenja programa ukoliko je to potrebno.

```
// Loading pointer of source db
    LAR1  P##IN_CmdHMI

//Creating Any pointer for source db
    L    W#16#1002              //DATA TYPE TO COPY=02 (BYTE)
    T    LW    0

    L    82                     //AMOUNT OF DATA TO COPY
    T    LW    2

    L    W [AR1,P#4.0]          //DESTINATION DB NUMBER
    T    LW    4
```

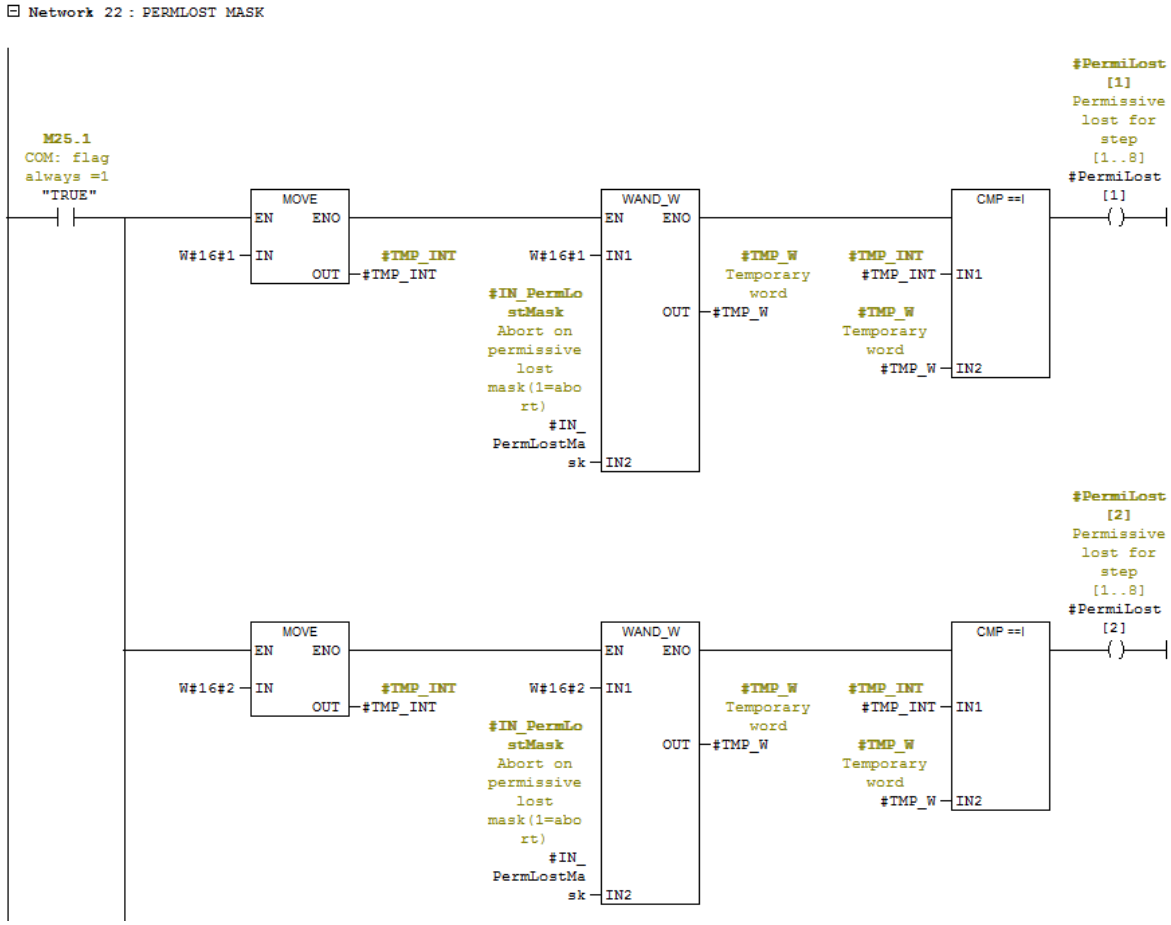
```
L    B#16#84              //TYPE OF MEMORY AREA(84 DATA BLOCK)
T    LB    6

L    0
T    LB    7

L    W [AR1,P#8.0]        //START ADRESS FROM WHERE TO COPY
T    LW    8
```

Ručno kreiranu ANY varijablu koristimo kao početnu ili odredišnu adresu. (Slika 4.4)



Sl. 4.4. Kopiranje vrijednosti iz CmdHMI DB-a u CmdHMI strukturu unutar funkcijskog bloka

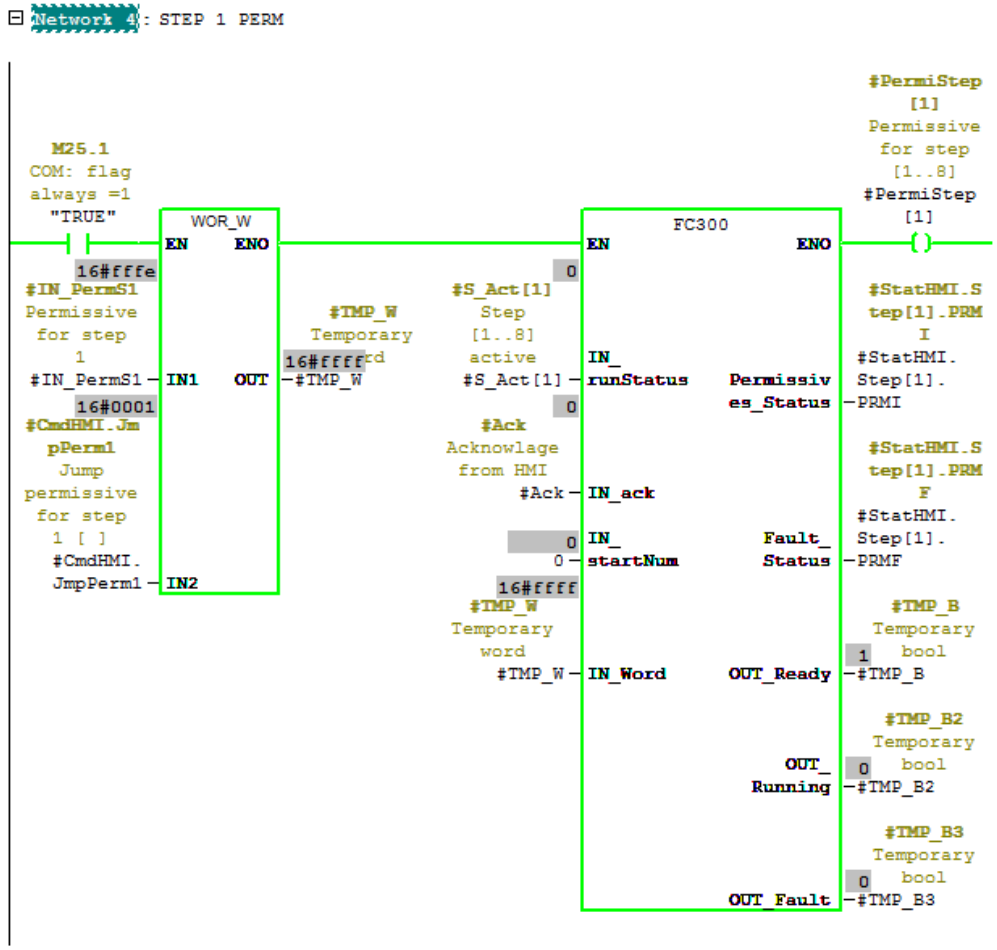
Kao što se može vidjeti iz tablice ulaznih podataka imamo ulazni signal *IN_PermLostMask* tipa podatka *byte*. Svaki bit u ovom *byte*-u predstavlja jedan korak sekvence nulti bit predstavlja prvi korak sekvence te sedmi bit predstavlja osmi korak sekvence. Ukoliko je vrijednost određenog bita jednaka 1 to znači da taj određeni korak ne smije izgubiti uvjete za rad tog koraka. Na slici 5.4. može se vidjeti kako se provjeravaju vrijednosti bitova u navedenoj masci. Provjeravamo jesu li vrijednosti određenih bitova jednake 1, ukoliko je to slučaj varijabla *PermiLost[1]* ima vrijednost logičke jedinice. Postupak se ponavlja za sve korake sekvence.



Sl. 4.5. Maska uvjeta sekvence

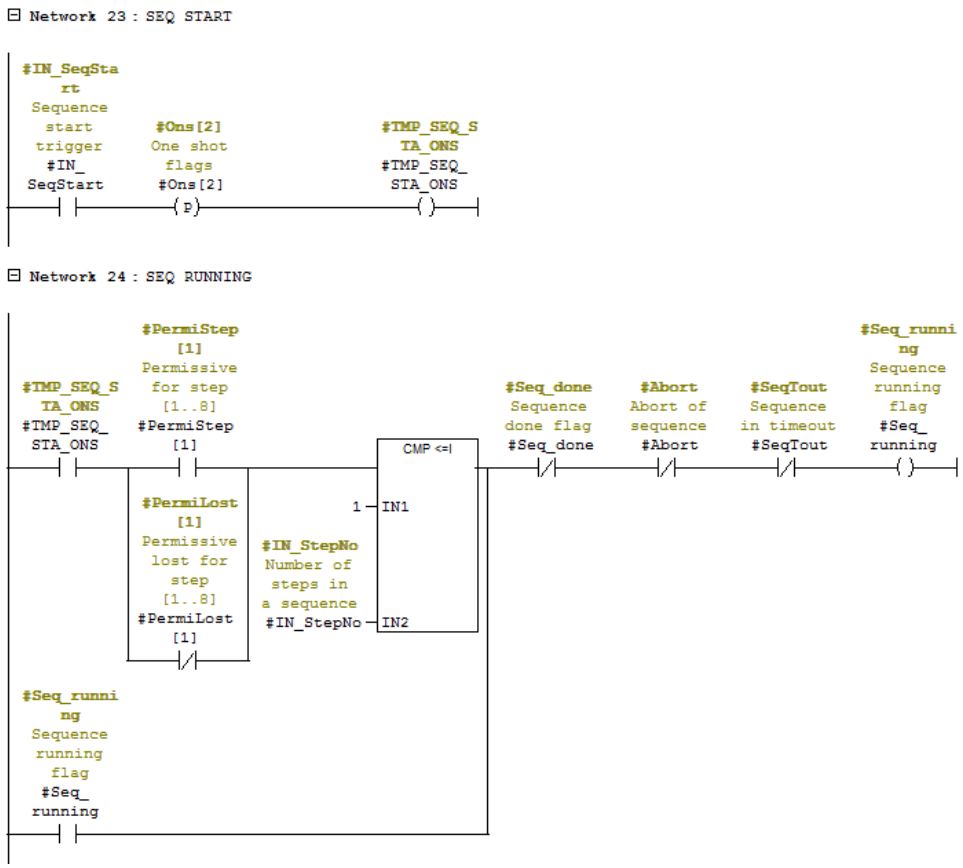
Uvjete za pokretanje možemo zaobići pomoću naredbi koje šaljemo s HMI-a. Unutar strukture *CmdHMI* nalaze se varijable *JmpPerm1*, *JmpPerm2*, …, *JmpPerm8* tipa podatka INT. Vršimo ILI logičku operaciju s varijablama *IN_PermS1* i *CmdHMI.JmpPermS1*. Uzmimo za primjer da je neki od bitova ulaznog uvjeta (*IN_PermS1*) npr. nulti bit, jednak 0. Forsiranjem tog istog bita s HMI-a postavljamo nulti bit varijable *CmdHMI.JmpPerm1* u logičku jedinicu. Logičkom operacijom ILI ove dvije varijable uz uvjet da su svi ostali bitovi varijable *IN_PermS1* jednaki 1 za rezultat

dobijemo -1 (ili u heksadekadskom obliku W#16#FFFF). Rezultat prosljeđujemo na ulaz sistemskog bloka koji se koristi za upravljanje uvjetima. Ako su svi uvjeti zadovoljeni varijabla *PermiStep[1]* postavlja se u vrijednost logičke jedinice. Postupak se ponavlja i za ostale korake sekvence. Na slici 5.6. prikazan je prijašnje objašnjeni primjer.



Sl. 4.6. Upravljanje uvjetima koraka sekvence

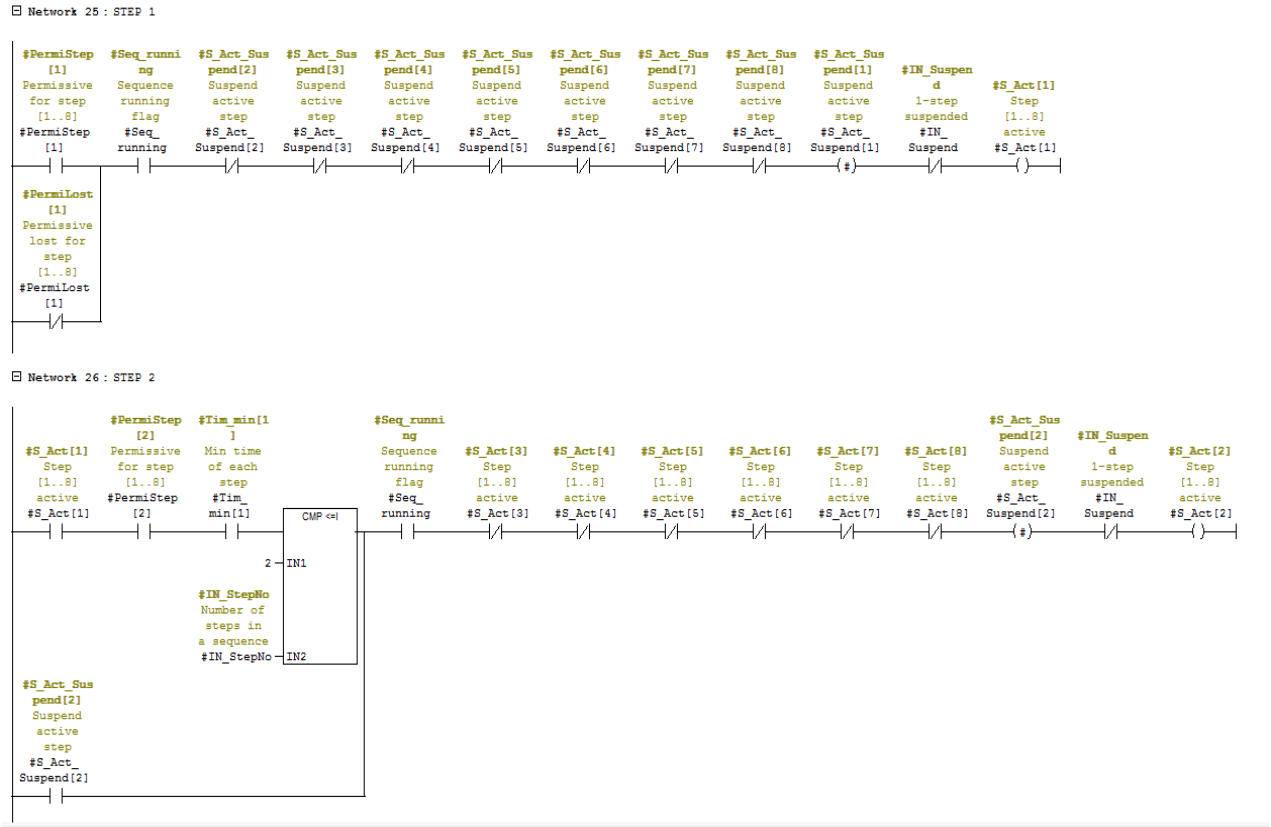
Sekvenca počinje s radom tek kada na ulazu u blok na ulaznom signalu *IN_SeqStart* imamo logičku jedinicu i ukoliko su zadovoljeni svi uvjeti za prvi korak sekvence. Na slici 5.7 možemo vidjeti za koje uvjete postavljamo određenu vrijednost varijable *Seq_running***.** Pokretanje se vrši detektiranjem rastućeg brida varijable *IN_SeqStart* zbog izbjegavanja kontinuiranog paljenja/gašenja sekvence.



Sl. 4.7. Postavljanje vrijednosti varijable Seq_running

Za postavljanje varijable u logičku vrijednost jedan moraju biti ispunjeni uvjeti za prvi korak sekvence te broj koraka sekvence mora biti jednak ili veći od 1. Jednom kada se postavi vrijednost u logičku jedinicu varijabla drži sama sebe (engl. *latch*) u stanju logičke jedinice sve dok se ne promijene vrijednosti jedne od tri varijable (*Seq_done*, *Abort*, *SeqTout*) u logičku jedinicu.

Uz uvjete da su svi uvjeti za prvi korak sekvence zadovoljeni i na ulazu dobijemo signal za pokretanje sekvence, programski blok ulazi u prvi korak sekvence te postavlja vrijednost varijable *S_Act[1]* u logičku vrijednost jedan. Prvi korak sekvence prestaje s radom kada se promjeni vrijednost varijable *Seq_running* u logičku vrijednost nula ili kada varijabla *S_Act_Suspend[2]* promjeni vrijednost iz logičke nule u logičku jedinicu. Varijable *S_Act_Suspend[3]*, *S_Act_Suspend[4], …, S_Act_Suspend[8]* se koriste za ograničavanje rada prvog koraka sekvence. Prvi korak sekvence ne smije biti aktivan ukoliko su aktivni viši koraci sekvence. Varijabla *S_Act_Suspend[1]* sprema trenutnu vrijednost do te varijable. To nam omogućava zaustavljanje sekvence s varijablom *IN_Suspend* bez prekida koraka sekvence.

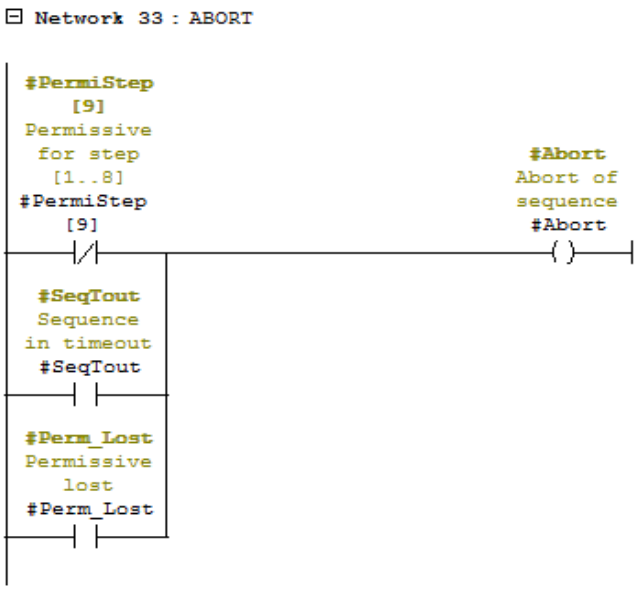


Sl. 4.8. Prijelaz iz prvog u drugi korak sekvence

Prelazak u drugi korak sekvence moguć je ako su zadovoljeni uvjeti za drugi korak, prvi korak sekvence je aktivan, proteklo je minimalno vrijeme za prvi korak i broj koraka sekvence je jednak ili veći od dva. Prelaskom u drugi korak sekvence automatski se prekida rad prvog koraka. Ista logika upravljanje se primjenjuje za ostale korake sekvence.

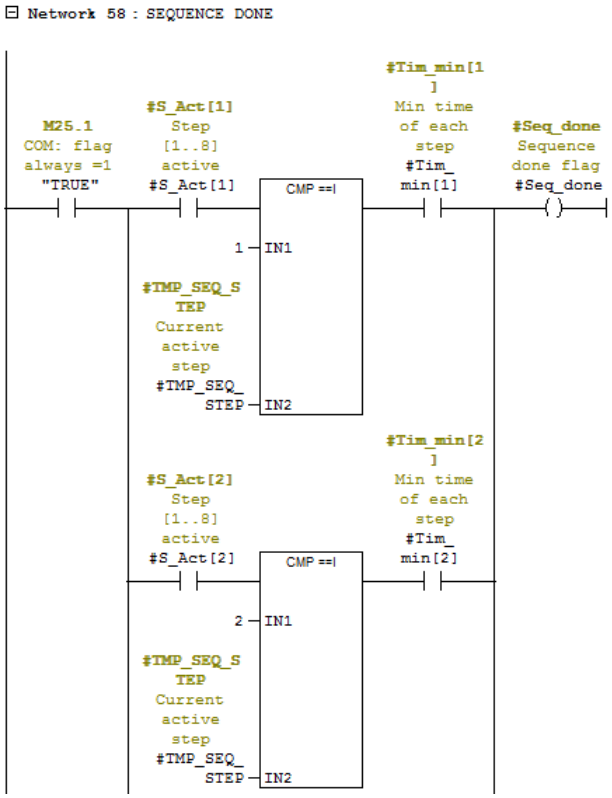
Prekid sekvence se odrađuje u tri slučaja (slika 5.9.):

- ako nisu zadovoljeni svi uvjeti za 9. korak sekvence, tj. korak za prekid
- vrijeme izvođenje koraka je prešlo maksimalno vrijeme izvođenja
- uvjeti za rad koraka koji se trenutno izvodi nisu više ispunjeni



Sl. 4.9. Prekid rada sekvence

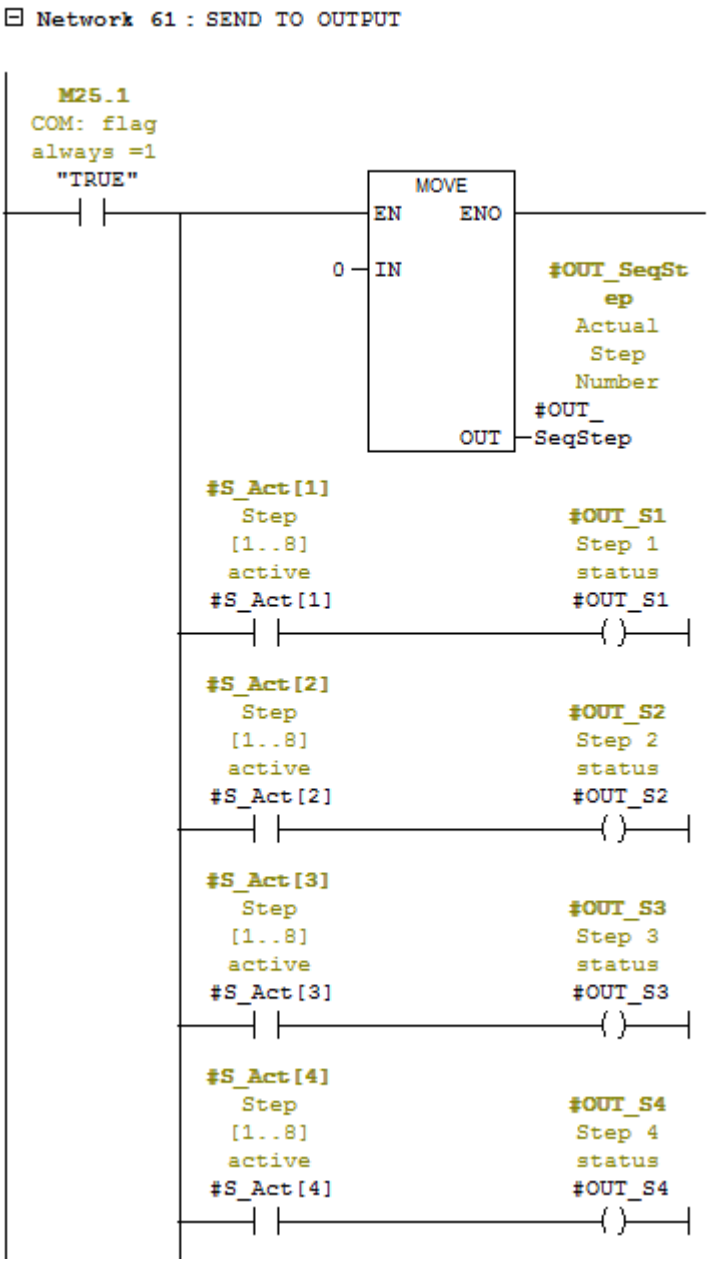
Varijabla *Seq_done* služi za određivanje kraja sekvence. Na slici 5.10. možemo vidjeti logiku za postavljanje vrijednosti *Seq_done* varijable.



Sl. 4.10. Postavljanje vrijednosti varijable Seq_done

Varijabla *Seq_done* ima vrijednost logičke jedinice ukoliko je trenutni korak sekvence ujedno i zadnji korak sekvence te je prošlo minimalno vrijeme u kojom je zadnji korak sekvence radio. Promjenom vrijednosti iz logičke nule u logičku jedinicu varijable *Seq_done* sekvenca prestaje s radom.

U posljednjem dijelu programa prosljeđujemo vrijednosti korištenih varijabli unutar bloka na izlazne varijable bloka i upisujemo vrijednosti u strukture unutar bloka. Vrijednosti varijabli unutar struktura upisujemo u odgovarajuće podatkovne blokove ovisno o korištenoj strukturi.

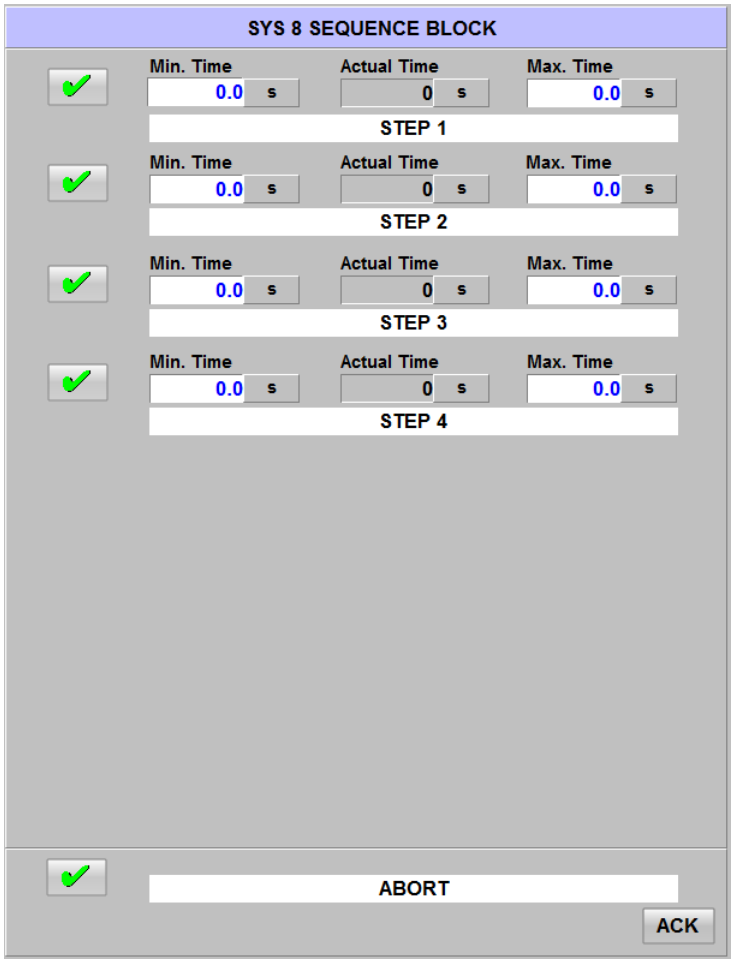


Sl. 4.11. Ispis vrijednosti varijabli na izlazne varijable bloka

4.2 SIMATIC WinCC

Vizualizacija programskog rješenja iz Step-*a* 7 napravljena je u u SIMATIC WinCC programskom alatu. U svrhu pojednostavljenja konfiguracije programskog bloka, dio konfiguracije se postavlja u vizualizaciji programskog rješenja. Kompletna konfiguracija programskog bloka u Step-u se ne može napraviti bez parametara koji se konfiguriraju u vizualizaciji. U ovome radu prikazan je izgled vizualizacije u simuliranom radu.

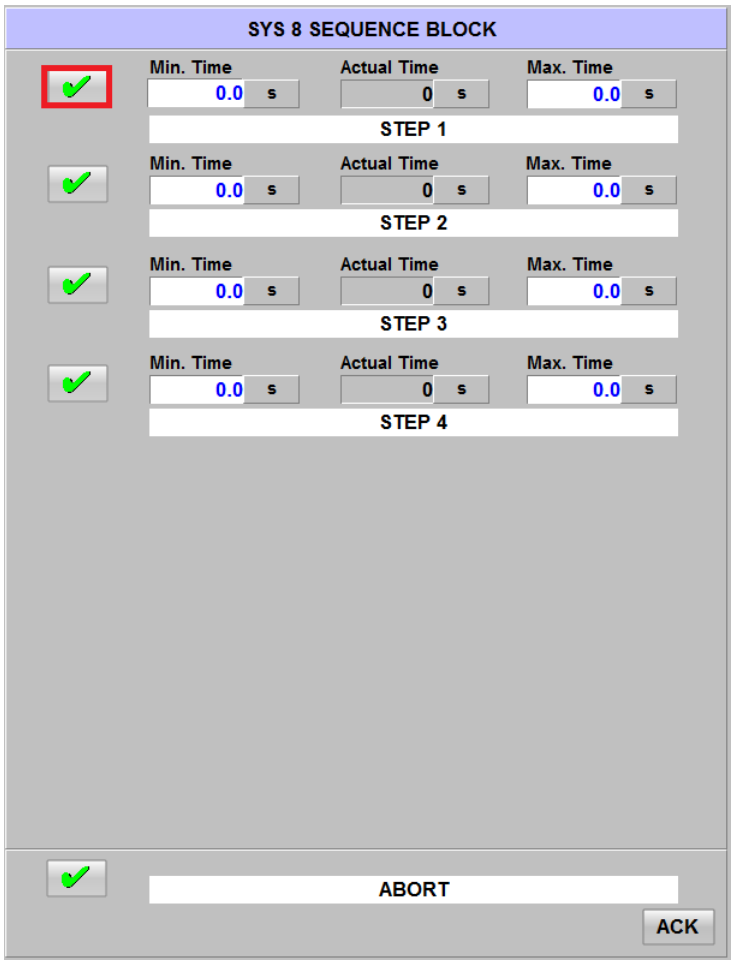
Izgled vizualizacije prikazan je na slici 4.12. Prikazana su 4 koraka sekvence. Broj koraka sekvence je jedna od ulaznih vrijednosti programskog bloka i ovisno o toj vrijednosti prikazujemo broj koraka sekvence na vizualizaciji. Za svaki korak sekvence moramo upisati minimalno, maksimalno vrijeme. Trenutno vrijeme u aktivnom koraku je informacija za korisnika. Korak za prekid sekvence je uvijek vidljiv.



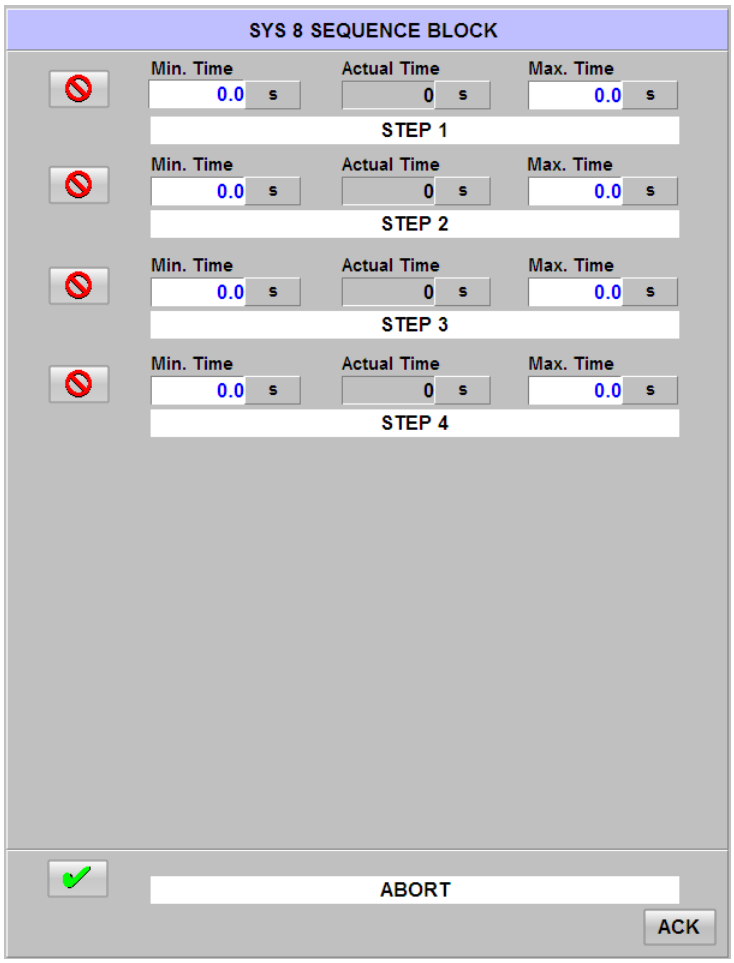
Sl. 4.12 Vizualizacija programskog rješenja

Minimalno vrijeme koraka sekvence možemo postaviti pritiskom na element ispod natpisa *Min. Time* prikazanog na slici 4.12. Na isti način postavljamo i maksimalno vrijeme. Ako prilikom unosa vremena za jedan od koraka sekvence unesemo samo minimalno vrijeme a maksimalno vrijeme ima vrijednost 0, prilikom rada sekvence navedeni korak će biti aktivan sve dok ne unesemo maksimalno vrijeme. Nakon unosa maksimalnog vremena sekvenca će nastaviti s radom ili će se prekinuti njezin rad ovisno o unesenoj vrijednosti. Ako je uneseno maksimalno vrijeme koje je manje od trenutnog vremena tog koraka prekinuti će se rad sekvence.

Označeni element na slici 4.13. predstavlja kumulativni status uvjeta za prvi korak sekvence. Ukoliko su svi uvjeti prvog koraka zadovoljeni prikazujemo zelenu kvačicu. Ako barem jedan od uvjeta nije zadovoljen prikaz možemo vidjeti na slici 4.14.

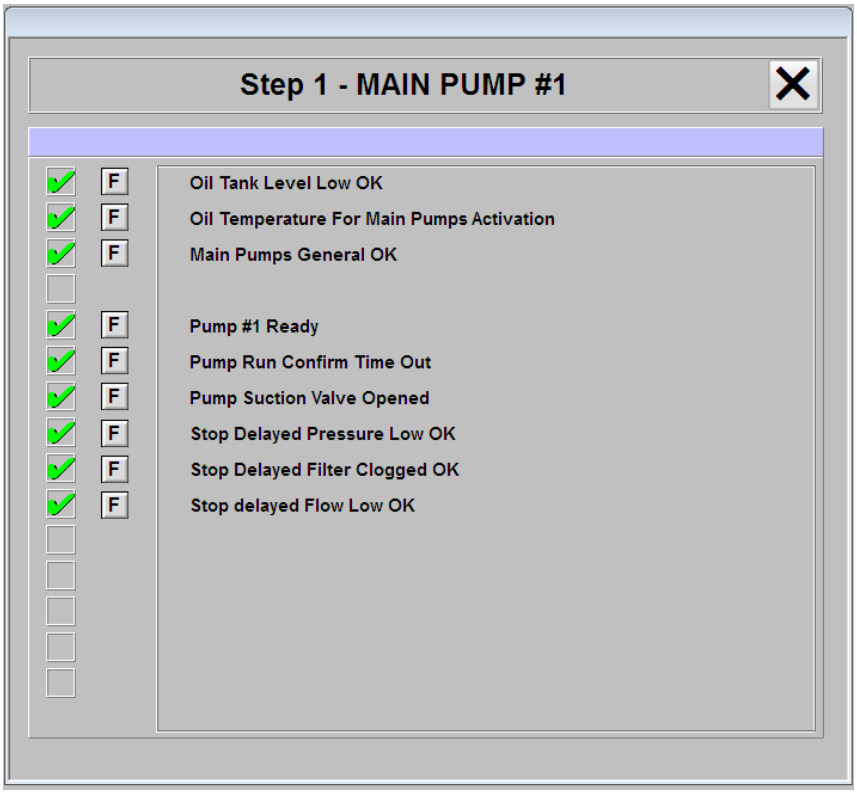


Sl. 4.13 Kumulativni status uvjeta koraka – svi uvjeti zadovoljeni

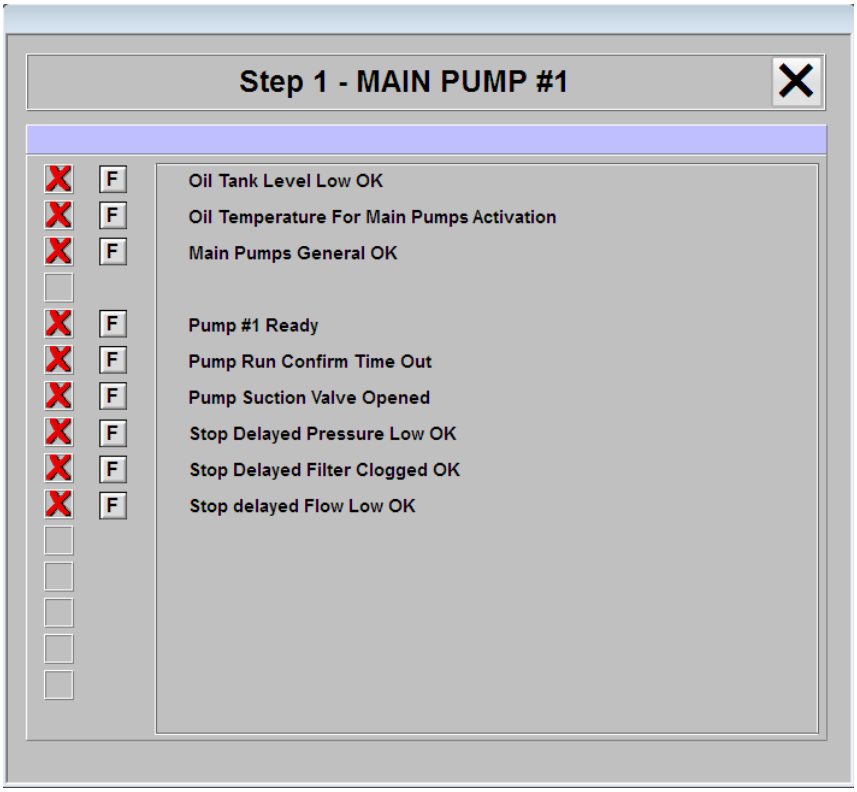


Sl. 4.14. Kumulativni status uvjeta koraka – jedan ili više uvjeta nisu zadovoljeni

Pritiskom miša na navedeni element otvara se prozor za prikaz uvjeta u kojem su prikazani statusi i opis svakog bita u uvjetu odgovarajućeg koraka. Na slici 4.15.prikazan je izgled prozora kada su svi uvjeti zadovoljeni te na slici 4.16. izgled prozora kada nisu zadovoljeni uvjeti. Korišteni su već gotovi prikazi koji se koriste u praksi.

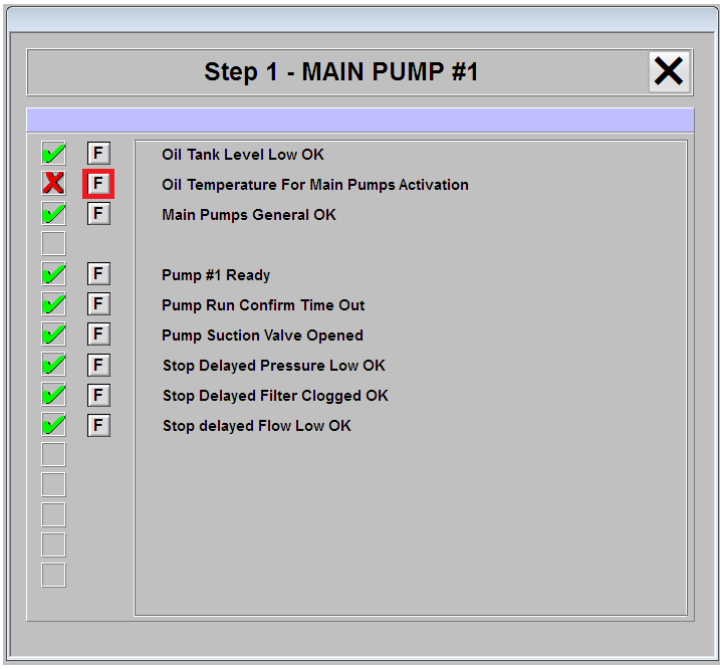


Sl. 4.15. Prozor za prikaz statusa bitova u uvjetu koraka- svi bitovi uvjeta su postavljeni



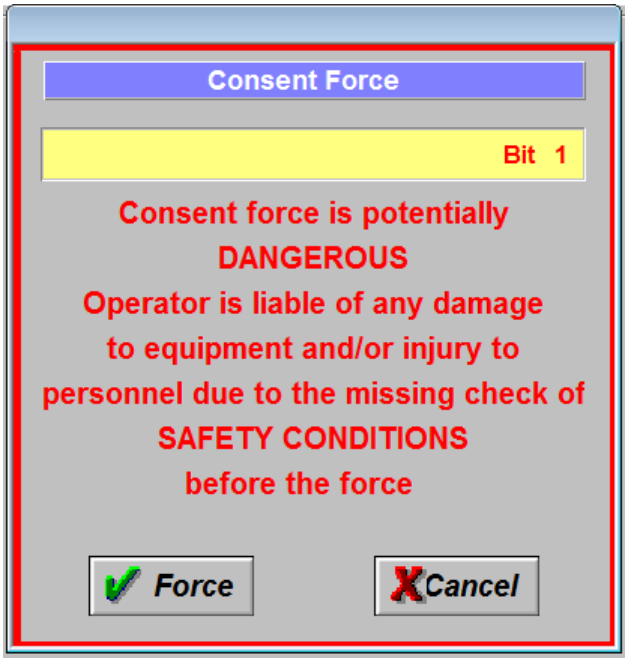
Sl. 4.16. Prozor za prikaz statusa bitova u uvjetu koraka- jedan ili više bitova uvjeta nisu postavljeni

U slučaju da jedan ili više uvjeta nije zadovoljeno, možemo ih zaobići tj. simulirati da je taj uvjet zadovoljen. Uzmimo za primjer da jedan uvjet nije zadovoljen. Pritiskom miša na označeni element na slici 4.17. možemo simulirati da je navedeni uvjet zadovoljen.



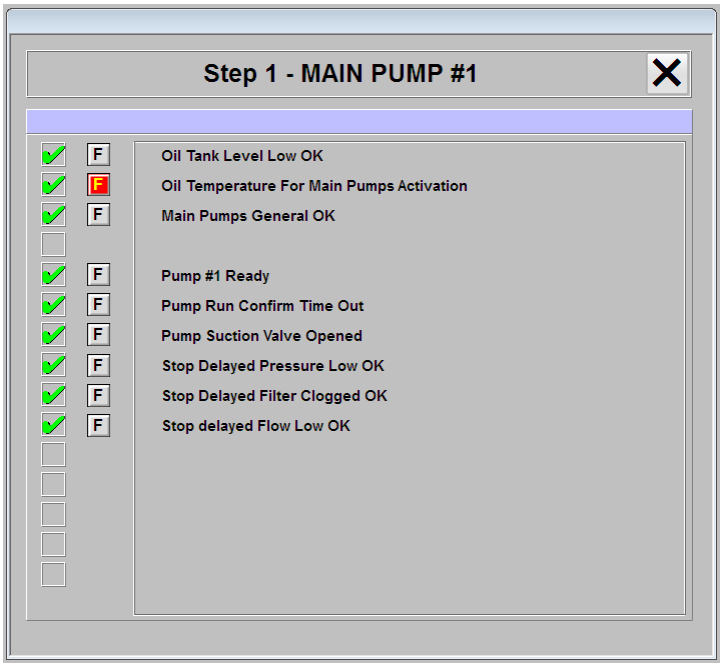
Sl. 4.17. Prozor za prikaz statusa bitova u uvjetu koraka- jedan bit uvjeta nije zadovoljen

Pritiskom na označeni element otvara se prozor za potvrdu navedene radnje. Prikaz prozora za potvrdu možemo vidjeti na slici 4.18.



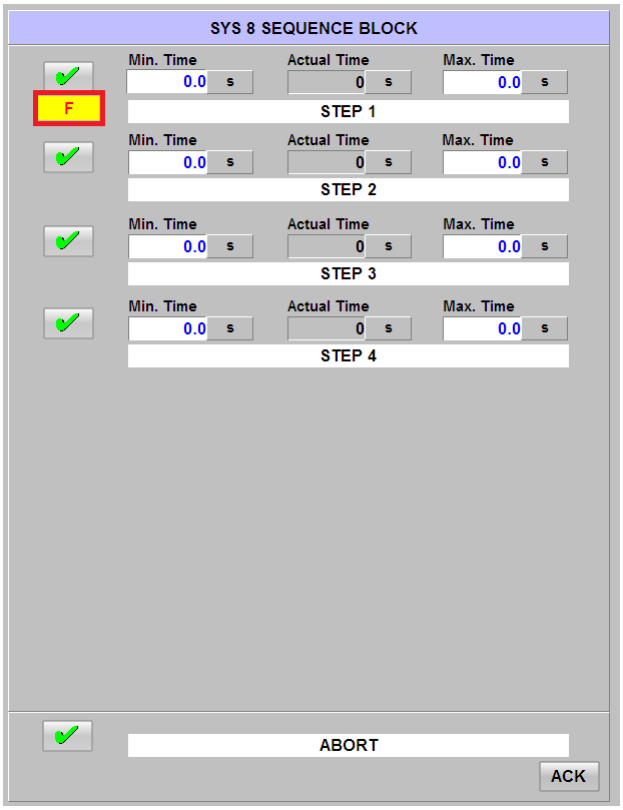
Sl. 4.18. Prozor za potvrdu postavljanja bita uvjeta

Pritiskom na tipku *Force* simuliramo logičku vrijednost jedan za odgovarajući bit, u ovom primjeru bit 1. Na glavnom prozoru za prikaz stanja uvjeta koraka prikazujemo prijašnju radnju kao što je prikazano na slici 4.19.



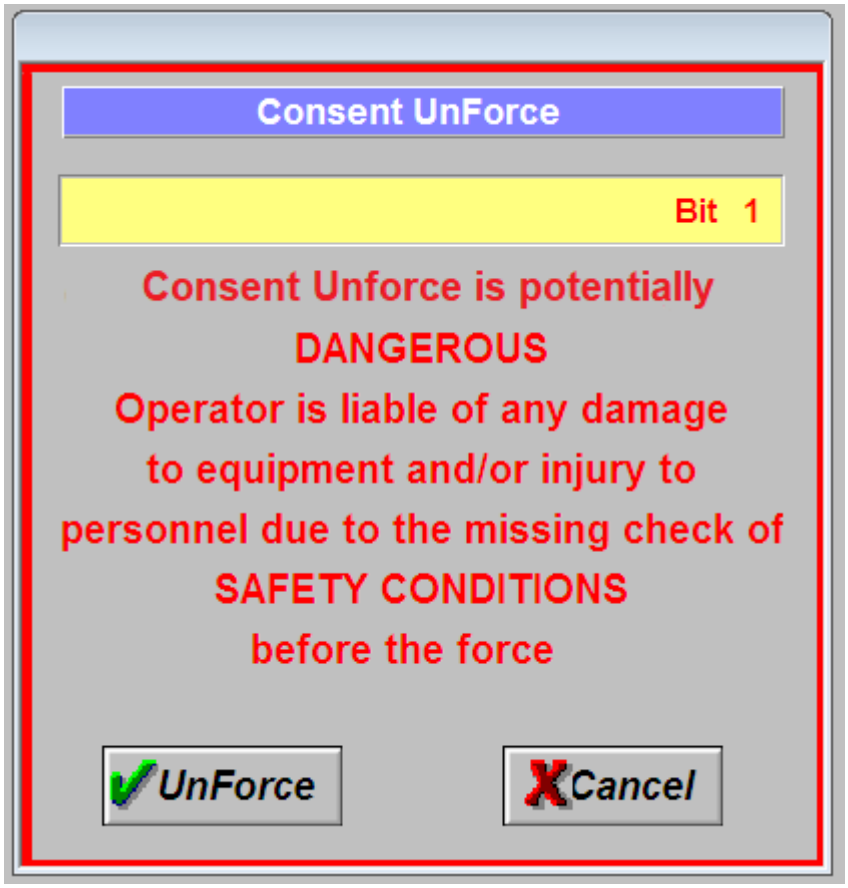
Sl. 4.19. Prikaz postavljenog bita putem HMI-a u prozoru za prikaz uvjeta

Na glavnom prozoru vizualizacije prikazujemo da je vrijednost jednog ili više bitova simulirana u logičku vrijednost jedinice. (Slika 4.20.)



Sl. 4.20. Prikaz postavljenog bita putem HMI-a na glavnom prozoru vizualizacije

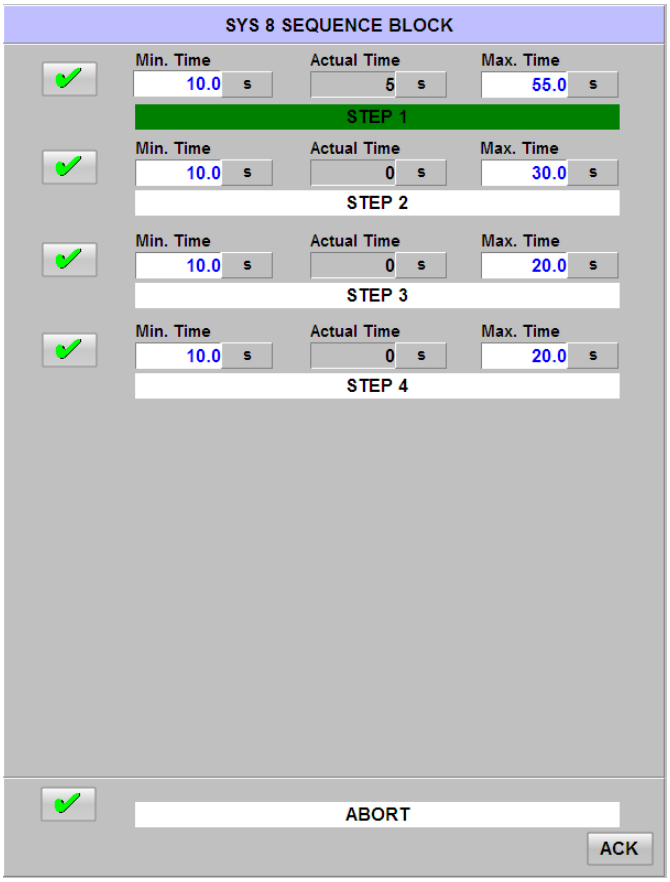
Prilikom rada postrojenja mogu se dogoditi kvarovi na strojevima i senzorima unutar postrojenja. U tom slučaju ako nam je uvjet da za neki od koraka sekvence mora biti ispravan senzor, npr. senzor za nisku temperaturu hidrauličkog ulja ali on nije ispravan. Senzor ne mora biti ključan za rad koraka sekvence te smijemo promijeniti vrijednost tog uvjeta kao da je senzor i dalje ispravan. Nakon otklona kvara možemo ponovno vratiti prikaz stvarne vrijednosti uvjeta pritiskom na isti element pomoću kojeg smo simulirali vrijednost. Ponovnim pritiskom otvara se prozor na slici 4.21. Ova mogućnost je korisna i prilikom testiranja sekvence prije puštanja u pogon.



Sl. 4.21. Prozor za poništavanje postavljanja bita s HMI-a

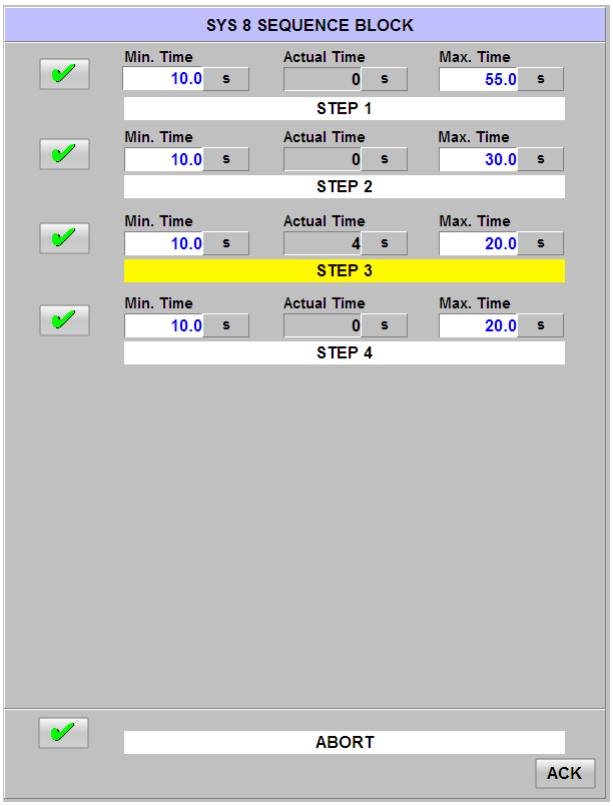
Na glavnom prozoru vizualizacije više ne prikazujemo da je simuliran jedan ili više bitova uvjeta.

Podešavanjem potrebnih vremena i uvjeta koraka sekvence i dolaskom signala za pokretanje započinje rad sekvence. Prikaz sekvence u radu možemo vidjeti na slici 4.22.

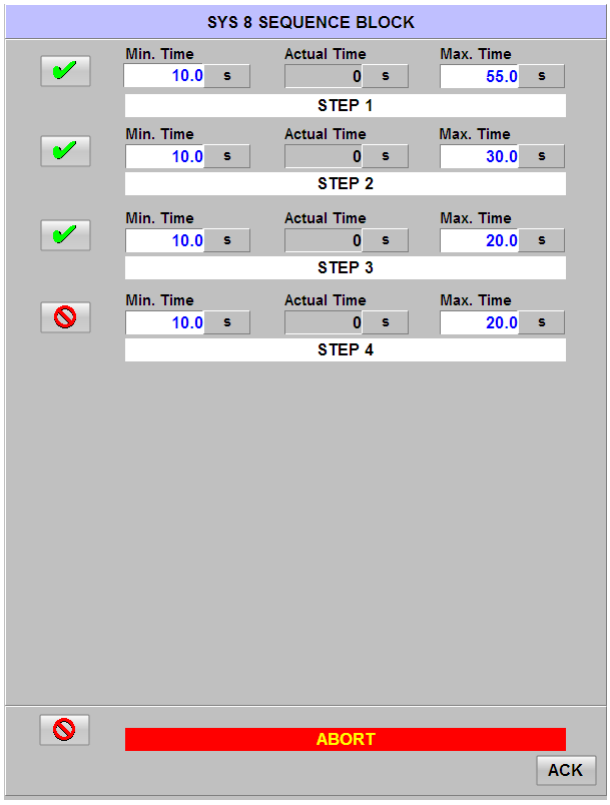


Sl. 4.22. Korak u aktivnom stanju

Osim u normalnom radu korak sekvence može biti obustavljen ili u prekidu. Prikaz obustavljenog koraka vidljiv je na slici 4.23. Prekid sekvence osim o programskim uvjetima prekida ovisi i o signalima iz polja, npr. nema napajanja za dio postrojenja u kojem se odvija rad sekvence, pritisnuta je gljiva za opasnost u istom dijelu postrojenja ili slično. Prekid sekvence prikazan je na slici 4.24. Pritiskom na element za prikaz statusa uvjeta za *Abort* korak možemo provjeriti koji je od uvjeta uzrok prekida sekvence.



Sl. 4.23. Korak u obustavljenom stanju

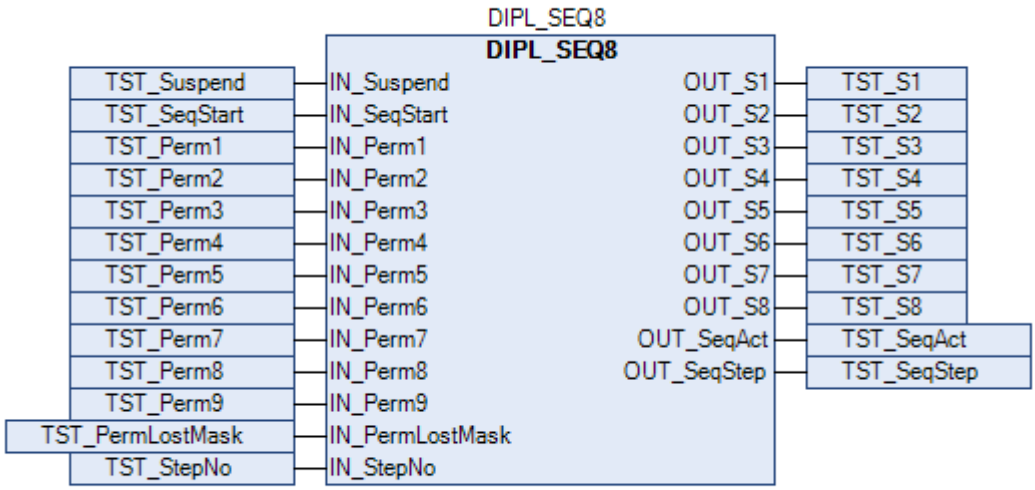


Sl. 4.24. Sekvenca u prekidu

Opisan rad programskog rješenja i vizualizacije ostvaren je pomoću Step 7 simulatora koji simulira rad stvarng PLC-a. Komunikacija između WinCC vizualizacije i Step 7 programa rađena je preko lokalne *ethernet* mreže.

4.3 Codesys

Programsko rješenje zadatka ovog diplomskog rada u CODESYS-u pisano je programskim jezikom CFC (engl. *The Continuous Function Chart)*. Programsko rješenje u CODESYS programskom alatu temelji se na logici programskog rješenja pisanog u SIMATIC Step 7 te zbog toga detaljna programska rješenja nisu prikazana. Na slici 4.25. prikazan je programski blok u CODESYS programskom alatu.



Sl. 4.25. Programski blok za upravljanje sekvencom u CODESYS programskom alatu

Na slici možemo primijetiti da programski blok ima manje ulaznih veličina od programskog rješenja u Step-u 7. Jedna od prednosti CODESYS programskog alata u usporedbi sa Step 7 programskim alatom je ta što CODESYS omogućava vizualizaciju programskog rješenja. Zbog toga se vrijednosti varijabli čitaju izravno iz programa.

Vizualizacija u programskom alatu CODESYS temelji se na vizualizaciji u WinCC programskom alatu. Funkcionalnosti vizualizacija su jednake. Na slici 4.26. prikazana je vizualizacija u CODESYS programskom alatu.

Sl. 4.26. Vizualizacija programskog bloka u CODESYS programskom alatu

5 ZAKLJUČAK

U praksi se susrećemo s uključi/isključi (engl. *on/off*) funkcijama ili uključi/isključi stanjima neke opreme u procesima kojima upravljamo. Takav tip upravljanja se naziva logičko upravljanje. Jednostavnost ovakvog tipa upravljanja čini ga pogodnim za upotrebu u automatskom upravljanju strojevima i procesima u kojima se zahtijeva da proces ili stroj slijede sekvencu operacija. Primjena logičkog upravljanja u sekvencama rada dovela je do termina sekvencijalno upravljanje. U automatskom radu postrojenja sekvence čine znatni dio regulacije rada.

Zbog važnosti ispravnog rada sekvenci, učestalosti njihovog korištenja i zbog pojednostavljenja programiranja rada sekvenci došlo je do potrebe za izradom sistemskog bloka za upravljanje sekvencom. Blok mora biti univerzalan i njegova konfiguracija mora biti jednostavna. Broj koraka sekvence definiran je kao jedan od ulaza u programski blok. Početak rada sekvence kontroliran je signalima iz polja. Isto tako uvjeti za određeni korak čitaju se iz polja. Programski blok ima mogućnost obustavljanja rada koraka sekvence i nastavka rada nakon što više ne želimo obustaviti njezin rad. Signal za obustavljanje rada nalazi se na ulazu u blok. Na izlazu bloka postavljamo stanje svakog koraka. Na izlazu bloka još prikazujemo status sekvence i ukoliko je ona aktivna prikazujemo u kojem koraku se sekvenca nalazi. U radu su prikazana dva programska rješenja: u SIMATIC Step 7 programskom alatu s odgovarajućom vizualizacijom u SIMATIC *WinCC* programskom alatu i CODESYS programskom alatu s vizualizacijom u istom alatu. Rezultat rada je različit od postavljenog zadataka zato što se predstavljeno rješenje pokazalo kao bolje od traženog.

Kao poboljšanje ovoga rada predlaže se ograničavanje unosa vremena nakon što je sekvenca aktivna. Također se predlaže uvođenje prikaza uvjeta koraka sekvence koji su bili zadovoljeni prilikom ulaska u određeni korak ali su tokom rada ti uvjeti izgubljeni na kratki period vremena. Ukoliko je taj period vremena vrlo mali nećemo moći vidjeti koji je od uvjeta koraka prekinuo rad sekvence.

LITERATURA

[1] Prof.dr.sc Nedeljko Perić, AUTOMATSKO UPRAVLJANJE predavanja, Zagreb, 2004

[2]L.A.Bryan, E.A.Bryan, Programmable Controllers Theory and Implementation, Second Edition, Atalanta, 1997 dostupno na: http://www.etf.ues.rs.ba/~slubura/Procesni%20racunari/PCTI/094410732X%20Programmable%20controllers%20theory%20and%20implementation.pdf

[3]Žagar, B. (2016). Upravljanje pozicijom u otvorenoj petlji pomoću digitalnih davača pozicije te digitalnih i analognih aktuatora (Diplomski rad). Preuzeto s https://urn.nsk.hr/urn:nbn:hr:200:751949

[4] https://www.ucg.ac.me/skladiste/blog_2583/objava_21480/fajlovi/esau7.pdf

[5] Statement List (STL) for S7-300 and S7-400 Programming Reference Manual

[6] CODESYS Control V3 Manual

POPIS SLIKA

SAŽETAK

U ovom radu je izrađen programski blok za upravljanje sekvencom. Sekvencijalnim upravljanjem izvršnim članovima u procesu zadajemo određeni niz naredbi. Procesi se moraju pokrenuti i zaustavljati sa prethodno precizno definiranim sekvencama akcija. Sekvencijalno upravljanje procesima se uglavnom vrši pomoću PLC-ova. Programski blok razvijen u ovom radu podržava sekvence od najviše 8 koraka.. Broj koraka definiran je kao jedan od ulaza u programski blok. Početak rada sekvence upravljan je signalima iz polja kao i samo izvođenje koraka sekvence. Korak sekvence može se nalaziti u tri stanja: aktivno stanje, kada je izlaz koraka aktivan; obustavljeno stanje, kada je izlaz koraka neaktivan i sekvenca nije u prekidu; u prekidu, kada je izlaz koraka neaktivan i sekvenca je u prekidu. Izlazi programskog bloka su signalizacija broja trenutno aktivnog koraka, postavljanje vrijednosti za svaki korak i status sekvence. Programska rješenja izrađena su u SIMATIC Step 7 i CODESYS programskim alatima, a vizualizacija je izrađena u SIMATIC WinCC i CODESYS programskim alatima.

Ključne riječi: STEP 7, CODESYS, *ladder logic*, sekvencijalno upravljanje, PLC

ABSTRACT

In this thesis, a program block for sequence control was developed. In sequential control we specify a series of commands for process actuators.A process must be started and stopped with a predefined sequence of actions. Sequential control is mostly implemented using the PLC. The program block developed in this work supports sequences consisting of at most 8 steps. The number of steps is defined as an input into the program block. The sequence start as well as execution of the sequence steps is controlled by field signals. There are 3 possible states of sequence steps: active state, where the step output is active; paused state, where the step output isn't active and the sequence isn't in the abort state;the abort state, where the step output isn't active and the sequence is in the abort state.The program block outputs are the number of the current active step, the status of each step and the status of the sequence. The software solutions were created in SIMATIC Step 7 and CODESYS software tools, and visualization was created in SIMATIC WinCC and CODESYS software tools.

Ključne riječi: STEP 7, CODESYS, ladder logic, sequence control, PLC

ŽIVOTOPIS

Matko Teni rođen je 19.8.1993. godine u Osijeku. Nakon završene osnovne škole u Josipovcu, upisuje Elektrotehničku i prometnu školu u Osijeku u kojoj završava razrede s odličnim i vrlo dobrim uspjehom. Po završetku srednje škole upisuje Sveučilišni preddiplomski studij elektrotehnike na Elektrotehničkom fakuletu u Osijeku 2012. godine. Nakon prve godine studija se prebacuje na Sveučilišni preddiplomski studij računarstva. Godine 2016. završava preddiplomski studij računarstva te stječe status prvostupnog inženjera računarstva. Iste godine upisuje sveučilišni diplomski studij smjer Procesno računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

Matko Teni