REPUBLIKA SLOVENIJA
MINISTRSTVO ZA ŠOLSTVO IN ŠPORT

Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski socialni sklad

PROGRAMIRANJE V AVTOMATIKI

ANDRO GLAMNIK

Višješolski strokovni program: Mehatronika
Učbenik: Programiranje v avtomatiki
Gradivo za 2. letnik

Avtor:

Andro Glamnik, univ. dipl. inž.
ZAVOD IRC, Ljubljana
Višja strokovna šola

Strokovni recenzent:
Mag. Slavko Murko, univ. dipl. inž.

Lektorica:
Bojana Samarin, univ. dipl. slov.

```
CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

681.527.7:004.4(075.8)(0.034.2)

GLAMNIK, Andro
   Programiranje v avtomatiki [Elektronski vir] : gradivo za 2.
letnik / Andro Glamnik. - El. knjiga. - Ljubljana : Zavod IRC,
2011. - (Višješolski strokovni program Mehatronika / Zavod IRC)

Način dostopa (URL): http://www.impletum.zavod-
irc.si/docs/Skriti_d
okumenti/Programiranje_v_avtomatiki-Glamnik.pdf. - Projekt
Impletum

ISBN 978-961-6857-14-7

258157824
```

Izdajatelj: Konzorcij višjih strokovnih šol za izvedbo projekta Impletum
Založnik: Zavod Irc, Ljubljana.
Ljubljana, 2011

Strokovni svet rs za poklicno in strokovno izobraževanje je na svoji 132. seji dne 23.9.2011 na podlagi 26. člena Zakona o organizaciji in financiranju vzgoje in izobraževanja (Ur.l. RS, št. 16/07-ZOFVI-UPB5, 36/08 in 58/09) sprejel sklep št.01301-5/2011/11-2 o potrditvi tega učbenika za uporabo v višješolskem izobraževanju.

Gradivo je sofinancirano iz sredstev projekta Impletum ‘Uvajanje novih izobraževalnih programov na področju višjega strokovnega izobraževanja v obdobju 2008–11’.
Projekt oz. operacijo delno financira Evropska Unija iz Evropskega socialnega sklada ter Ministrstvo RS za šolstvo in šport. Operacija se izvaja v okviru Operativnega programa razvoja človeških virov za obdobje 2007–2013, razvojne prioritete ‘Razvoj človeških virov in vseživljenjskega učenja’ in prednostne usmeritve ‘Izboljšanje kakovosti in učinkovitosti sistemov izobraževanja in usposabljanja’.

KAZALO VSEBINE

PREDGOVOR

V zadnjem času, v zadnjih desetletjih, se je izredno razvila tehnika avtomatizacije industrijskih procesov. Računalniški sistemi so postali zmogljivejši, uporabniku prijaznejši in predvsem cenejši. V 18. stoletju so iznašli parni stroj, nato pa je svet preživel štiri dolge cikluse, ki se imenujejo kondrativovi cikli (po ruskem ekonomistu Nikolaju Kondrativu):

- parni stroj in konoplja oziroma industrijske rastline,
- železnica in jeklo,
- elektrika in kemija,
- nafta in avtomobil.

V začetku osemdesetih let se pojavi novost, imenovana informacijska tehnika: z odkritjem tranzistorjev in razvojem na področju miniaturizacije, digitalizacije, strojne in programske opreme, laserske tehnologije, optoelektronike, Petri mrež (1961) ... vse to je povzročilo neverjetno dinamiko razvoja.

Izreden tehnično-ekonomski zagon in močno povezovanje izjemnih sprememb na področju socialnega, kulturnega in ekološkega vpliva na življenje predstavlja peti kondrativov ciklus. Ta ciklus pomeni spremembo razmišljanja in trgovanja posameznikov in zato se spreminjajo tudi strategije podjetij na nacionalnem in internacionalnem nivoju, tako silovit tehnološki skok pa povzroča strukturalne pretvorbe. Zelo pomemben faktor so poleg klasičnih produkcijskih faktorjev (kot so delovna sila, surovine, energija) postale informacije.

Izboljšanja kvalitete ni mogoče izraziti z eno številko, a moramo vedeti, da brez uvedbe kibernetizacije, informatizacije in avtomatizacije pri znatnem številu proizvodov zahtevanih standardov kvalitete ni mogoče doseči. Poleg teh neposrednih vplivov je seveda treba upoštevati še posredne vplive uvajanja te tehnologije, ti pa se kažejo predvsem v potrebi po boljši organizaciji in povečani urejenosti dela.

Živimo v času, ko je procesna avtomatizacija nujna. Čas izdelave produkta mora biti zelo kratek, obenem pa zahtevamo visoko natančnost, veliko hitrost izdelave in čim manj napak pri izdelavi.

V danem učbeniku so opisani orodja in procesi, ki jih potrebujemo za avtomatizacijo. V prvem poglavju je opisana uporabnost računalnika, delitev računalnikov, uporaba krmilnih mikroračunalnikov ter njihova delitev. Sledi poglavje, kjer so opisane vrste programskih jezikov, od nižjih do višjih programskih jezikov ter programski jeziki za programiranje obdelovalnih strojev in robotov. Programski jezik C++, ki bralca popelje po osnovnih gradnikih programskega jezika C++ in mu razloži delovanje, se nahaja v tretjem poglavju. Nato je podrobno opisan mikroprocesor, enote, iz katerih je sestavljen, kar omogoča bralcu, da spozna delovanje in procese mikroračunalnika, sledi programirljivi logični krmilnik ter njegovo delovanje. V zadnjih treh poglavjih pa je podrobno predstavljen programski jezik SIMATIC step 7, ki ga uporabljamo za programiranje SIMATIC-ovih krmilnikov. Podrobno je predstavljen tudi grafični program GRAPH za grafično programiranje koračnih krmilj. V zadnjem poglavju pa je predstavljen program za izdelavo SCADA sistemov WinCC Flexible.

1 SPLOŠNO O RAČUNALNIKU

Računalniki imajo v sodobnem času velik vpliv na naše življenje. Izvirajo že iz časov tik pred prvo in po drugi svetovni vojni, prvi uporabni računalniki (npr. Z3, ENIAC) pa so nastali ob koncu druge polovice 20. stoletja in so sprva bili dostopni le vojski in večjim raziskovalnim ustanovam. Ker je polprevodniška tehnologija v petdesetih in šestdesetih letih prejšnjega stoletja zelo napredovala, so računalniki postali manj zahtevni, zmogljivejši in cenejši in tako dostopnejši širši populaciji.

Tako so se razvile ideje, da bi računalnik upravljal tudi zahtevnejše procese, ne le obdelavo številčnih podatkov. Največji problem je bil v tem, da so bili še vedno razmeroma dragi in veliki, s tem pa primerni le za omejen spekter aplikacij.

S pojavom mikroprocesorjev so se razmere tako bistveno spremenile, da je računalnik postal »najbolj revolucionaren izdelek v zgodovini človeštva« (Gordon Moore, ustanovitelj INTELA). V današnjem času si življenja brez mikroprocesorjev ni mogoče več predstavljati, saj se z njimi srečujemo na vsakem koraku. V gospodinjstvu se brez pralnega ali pomivalnega stroja, brez mikrovalovne pečice in še nekaterih aparatov ne znajdemo več, prav tako si težko predstavljamo življenje brez telefona, radia, televizije, fotoaparata … Tudi avtomobili so opremljeni z mikroprocesorskimi sistemi.

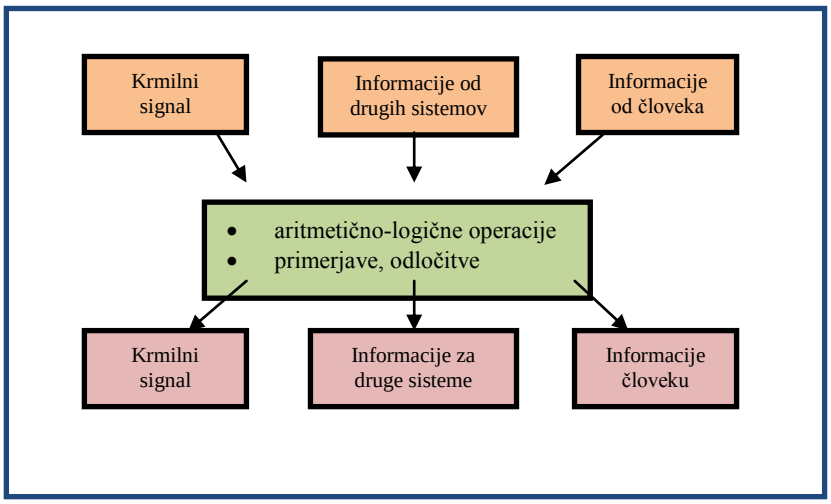
V komercialnih in industrijskih sistemih so na osnovi mikroprocesorjev zgrajeni sistemi za vodenje proizvodnje, merilne naprave, roboti …, ki se povezujejo v lokalne mreže, v katerih manjše računalnike do krmilnih sistemov v končnih proizvodnih celicah razporejeno nadziramo z močnejšimi. Ti potem proizvodnjo upravljajo z uporabo podatkovnih baz, upoštevajo poslovne odločitve itd.

Mikroračunalniško krmilje je v primerjavi z diskretno logiko sestavljeno iz manj komponent in je njihov razvoj hitrejši in preprostejši, testiranje lažje, cena nižja, možnost napake pa manjša. Isto elektroniko lahko uporablja več različnih naprav, saj so standardna mikroračunalniška vezja univerzalno uporabna, zaradi teh vezij pa so se sposobni hitro prilagajati spremembam, ker sta potrebna le nova konfiguracija in program. Vzdrževanje je preprostejše (manj elementov, univerzalni moduli, možnost samotestiranja, standardizirani postopki).

1.1 VLOGA RAČUNALNIKA

Računalnik si lahko predstavljate kot črno skrinjo, ki je inteligentni del med vhodnimi in izhodnimi podatki. Da računalnik lahko obdeluje podatke, mora znati:

- brati napisani program in ga izvajati,
- opravljati aritmetične in logične operacije nad podatki,
- izvajati primerjave med podatki in se odločati,
- pomniti program in podatke.



Slika 1: Delovanje računalnika
Vir: Lasten

1.2 DELITEV RAČUNALNIKOV

a) **Informacijski računalniki** –v pisarni pomagajo tajnici pri vodenju poslovanja ter v razvojnem oddelku inženirjem pri razvoju novih izdelkov.

b) **Krmilni računalniki,** ki v proizvodnem obratu krmilijo stroje in proizvodne linije, so večinoma **mikroračunalniki**, saj imajo zraven mikroprocesorja samo osnovne sestavne dele (RAM,ROM,V/I vmesnike). Tudi PC-ji se lahko uporabijo za krmilne računalnike z ustreznimi programskimi orodji in krmilnimi vmesniki.

c) **CAD/CAM** (*Computer Aided Design/Computer Aided Manufactury*): Lokalne računalniške mreže omogočajo povezavo informacijskih in krmilnih računalnikov v CAD/CAM sisteme ter še naprej v sisteme za menedžersko spremljanje proizvodnje.

1.3 MIKRORAČUNALNIK KOT KRMILNIK PROCESOV

Programirljivi logični krmilnik (PLK)
Njegova funkcija je nadomeščanje človeka pri krmiljenju in vodenju strojev in proizvodnih procesov v industriji.

Klasično krmilje s trajno ožičenim programom:

Elektromehanska krmilja: Osnovni gradniki so releji, stikala, ventili. Način delovanja je določen s funkcijskim načrtom in vezalno shemo.

Elektronska krmilja: Sestavljena so iz logičnih elementov v obliki čipov in imajo te elemente med seboj povezane po načrtu. Načrt se konstruira na podlagi zahtev pravilnostne tabele ali logične funkcije z metodami minimizacije digitalnih vezij. Funkcija delovanja krmilja je določena z medsebojno povezavo elementov. Tudi tu pravimo, da je program delovanja določen z ožičenjem.

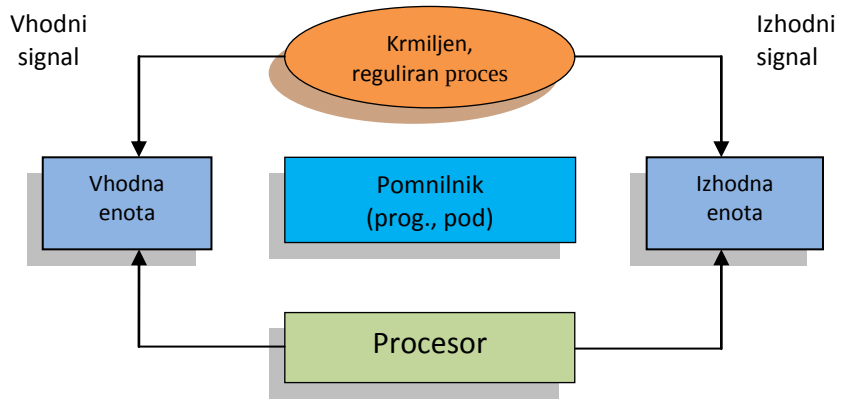
Programirljivi logični krmilnik (PLK):

Pri obsežnih krmiljih za vodenje in nadzor industrijskih procesov pa moramo ponavadi spremeniti program delovanja že v času preizkušanja. Včasih se tudi tehnološki postopek pogosto spreminja in zahteva nov program. Zato prihaja do zahtev po večji prilagodljivosti krmilja različnim potrebam. Takšnim krmiljem bi lahko spreminjali program, ne da bi morali spreminjati ožičenja.

Krmilni sistemi na osnovi PLK so sestavljeni iz :

- aparaturne ali strojne opreme oz. ***hardware*** (krmilnik, merilni pretvorniki, kontaktorji, stikala, ožičenje itd.)
- programske opreme – ***software*** (program delovanja krmilja, ki ga vpišemo v programski del pomnilnika). Za spremembo funkcije delovanja krmilja je potrebno le vpisati novi program v programski pomnilnik, medtem ko spremembe v strojni opremi praviloma niso potrebne.

V krmiljenem oziroma reguliranem procesu dajalniki signalov (senzorji) dajejo podatke o stanju in vrednosti posameznih veličin (temperatura, tlak, hitrost, pretok, pozicija itd.). Ti podatki se v vhodni enoti predelajo v binarne vrednosti, ki jih procesor lahko obdela. Procesor bere v skladu s programom posamezne vhodne podatke in ugotavlja stanja v procesu. Na osnovi stanj v procesu in programa pa daje preko izhodne enote ukaze v obliki napetostnih impulzov na izvršilne člene krmilja (releji, svetlobni in zvočni javljalniki, prikazovalniki, servo motor, koračni motor itd.).



Slika 2: Blokovni prikaz in delovanje PLK
Vir: Lasten

S programirljivim logičnim krmilnikom (PLK) lahko opravljate naslednje procesne funkcije:

- zajemanje in urejanje procesnih podatkov,
- izvajanje krmiljenja in regulacije,
- varovanje procesov v konfliktnih situacijah,
- analiza in ovrednotenje rezultatov procesa.

Povzetek:

Računalnik ima v današnjem času velik vpliv na naše življenje. Konec 20. stoletja so računalniki bili dostopni le vojski in nekaterim raziskovalnim ustanovam, nato pa je tehnologija v 50. in 60. letih zelo napredovala in računalniki so postali manj zahtevni, zmogljivejši in cenejši. Ko pa so se pojavili mikroprocesorji, je računalnik postal najbolj revolucionaren izdelek v zgodovini človeštva in v današnjem času se z njimi srečujemo prav na vsakem koraku. Računalnike delimo na informacijske, krmilne in CAD/CAM računalnike.

Programirljivi logični krmilnik (PLK) nadomešča človeka pri krmiljenju in vodenju strojev in proizvodnih procesov v industriji. Sestavljen je iz aparaturne ali strojne opreme (*hardware*) in programske opreme (*software*). Z njim lahko zajemamo in urejamo procesne podatke, izvajamo krmljenja in regulacije, varujemo procese v konfliktnih situacijah in analiziramo ter vrednotimo rezultate procesa.

Vprašanja:

1. Kako lahko definirate pojem računalnik?
2. Kaj mora računalnik znati, da lahko obdeluje podatke?
3. Od kod lahko računalnik dobiva podatke in kam jih lahko pošilja?
4. Kakšna je razlika v uporabnosti med informacijskimi in krmilnimi računalniki?
5. Kaj pomeni CAD/CAM?
6. Kakšna so trajno ožičena krmilja?
7. Katere so prednosti programirljivih logičnih krmilnikov pred trajnim ožičenjem krmilij?
8. Katere procesne funkcije lahko opravlja programirljivi logični krmilnik (PLK)?

2 NAČINI PROGRAMIRANJA RAČUNALNIKA

V tem poglavju boste spoznali pomen računalniškega programa ter značilnosti programskih jezikov, prav tako pa programska orodja in faze pri izdelavi programa. Razloženi so ***pojmi računalnik, računalniški program, programer*** **in** ***programiranje***. Predstavljeni so programski jeziki s primeri in razdeljeni glede na način programiranja, na čas prevajanja in na način vnosa. Sledi predstavitev programov za zajemanje in obdelavo podatkov Lego Mindstorm, Crocodile Technology in Labview (http://www.ni.com/labview/applications/) in osnovni pristop k programiranju.

2.1 RAČUNALNIK, RAČUNALNIŠKI PROGRAM IN PROGRAMIRANJE

Računalnik *(computer)* je stroj za obdelavo informacij, ki izvaja operacije ali instrukcije po računalniškem programu.

Računalniški program *(computer program)* so navodila stroju, kaj in kako naj obdeluje informacije.

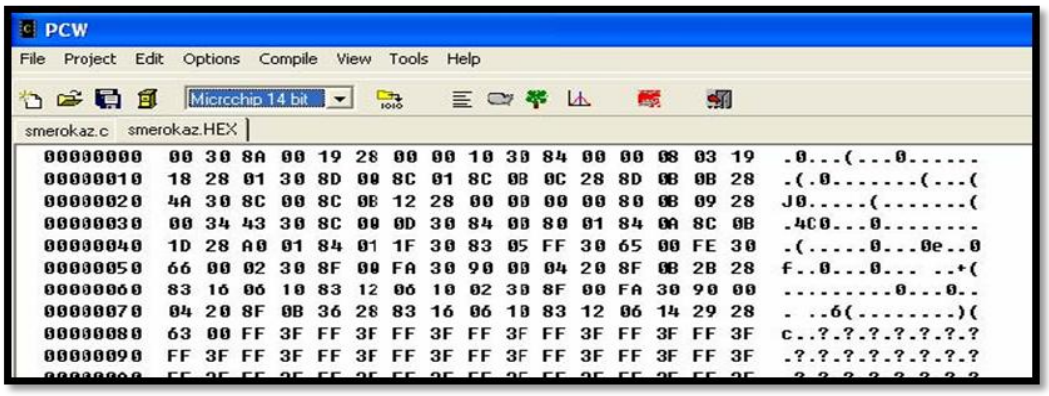
Programiranje *(programming)* je zapisovanje naloge v procesorju razumljivi obliki.

Programer *(computer programmer)* ali razvijalec programske opreme piše računalniške programe. Pri tem mora poznati probleme in rešitve nalog, ki jih bo pozneje reševal računalnik s pomočjo napisanega programa.

2.2 RAZDELITEV PROGRAMSKIH JEZIKOV S PRIMERI

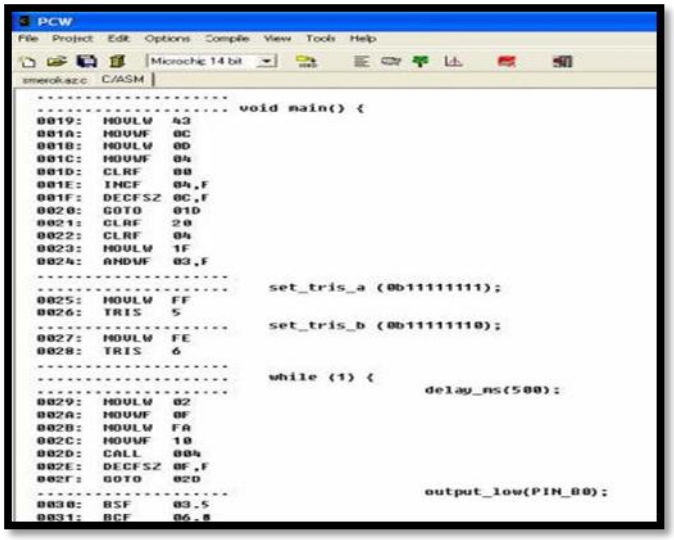
a) **Strojna koda računalnika Strojna koda**, **strojni jezik** ali **strojno besedilo programa** (angl. *Machine code* ali *Machine language*) je besedilo oziroma koda v izvršljivih datotekah, ki so jih iz izvornega besedila ustvarili prevajalniki ali zbirniki. Strojno besedilo programa je sestavljeno iz zaporedij izvršljivih strojnih ukazov, ki jih na osebnih računalnikih izvaja centralna procesna enota. Vsak procesorski sistem ima svojo arhitekturno zasnovo (platformo). To pomeni, da je strojni jezik zelo drugačen med različnimi procesorji (vendar enak za tiste na enaki zasnovi), tako da je treba za vsako platformo uporabiti drugačen prevajalnik (angl. *Compiler*), da prevede izvorno kodo v strojno. Čeprav najbolj odvisna od procesorske zasnove, je platforma skupek lastnosti vseh delov računalnika. V to štejejo tudi operacijski sistem in drugi pomembni programi (angl. *Software*) ali strojni deli (angl. *Hardware*). Centralna procesna enota neposredno razume objektno kodo, vendar za uspešno izvajanje na danem operacijskem sistemu potrebujemo prav tako določen izvršni zapis (angl. *Executable format*), ki je določen z ABI vmesnikom (angl. *Application Binary Interface*) operacijskega sistema. V izvršnih datotekah se torej nahaja strojna koda določenega izvršnega zapisa, kar je osrednji razlog za neprenosljivost računalniških programov med posameznimi operacijskimi sistemi. GNU/Linux in družina BSD uporabljajo npr. izvršni zapis ELF (angl. *Executable and Linkable Format*), Mac uporablja Mach-o, medtem ko Windows PE (angl. *Portable and Executable Format*).Izvršna datoteka s strojno kodo je lahko tudi brez izvršnega zapisa. Datoteko, ki jo je moč izvajati brez operacijskega sistema na centralni procesni enoti,

imenujemo ploščata izvršna datoteka (angl. *Flat binary file*). Strojno kodo pogosto označujemo kot prvo generacijo programskih jezikov.
Primer strojne kode prikazuje slika 3.



Slika 3: Primer programa v strojni kodi
Vir:Lasten

b) **Zbirnik** (angl. *Assembly language*, *assemble* –sestavljati) je nizko nivojski programski jezik druge generacije (jezik prve generacije je strojna koda), ki je napisan z mnemoniki. Posplošeno velja, da mnemoniki predstavljajo berljive inačice dvojiških zaporedij (ničle in enice), ki jih je potrebno sestaviti, da dobimo centralnemu procesorju razumljivo kodo. Natančneje, mnemoniki predstavljajo ukazne kode (angl. *Operation codes*, skrajšano *Opcodes*), ki so v centralni procesni enoti definirani po ISA arhitekturi (angl. *Instruction Set Architecture*). To kodo je nato običajno potrebno še povezati z določenimi strukturami, da dobimo delujoč izvedljiv program. Določene programske opreme zbirnikov, kot je na primer FASM, samo zamenjajo mnemonike in operande oziroma parametre z ustreznimi instrukcijami v strojnem programskem jeziku. Na ta način dobimo ploščate binarne izvršilne datoteke, ki vsebujejo (sicer odvisno od izkušenj posameznega računalniškega programerja) strojno kodo z izjemno algoritemsko učinkovitostjo. Programska koda v zbirniku je prikazana na sliki 4.



Slika 4: Primer programa v zbirniku
Vir: Lasten

c) **Visokonivojski jezik** (angl. *High-level language*) je programski jezik, zasnovan tako, da ustreza programerjevim zahtevam. Ni odvisen od interne strojne kode konkretnega računalnika. Visokonivojske jezike uporabljamo za reševanje problemov in jim velikokrat pravimo tudi problemsko orientirani jeziki – BASIC je bil na primer zasnovan tako, da je začetnikom omogočil hitro učenje, COBOL se uporablja za pisanje poslovnih programov, FORTRAN pa za programe, ki rešujejo znanstvene in matematične probleme. Nizkonivojski jeziki, v nasprotju z visokonivojskimi, močno odražajo značilnosti strojne kode določenega računalnika in jim zato pravimo tudi strojno orientirani jeziki. V nasprotju z nizkonivojskimi jeziki se je visokonivojskih jezikov relativno enostavno naučiti. Njihovi ukazi so namreč podobni človeškemu jeziku, zato programerju ni treba podrobno poznati notranjega ustroja računalnika. Vsak ukaz visokonivojskega jezika je ekvivalenten več strojnim ukazom. Visokonivojski programi so torej bolj kompaktni kot ekvivalentni nizkonivojski programi. Vendar pa moramo vsak visokonivojski program, preden ga lahko poženemo, prevesti v strojni jezik – bodisi s prevajalnikom bodisi z interpreterjem. Visokonivojski jeziki so zasnovani tako, da so prenosljivi. To pomeni, da program, napisan v visokonivojskem jeziku, lahko poženemo na vsakem računalniku, ki ima prevajalnik ali interpreter za ta jezik. Visokonivojski jezik je C++ prikazan na sliki 5.

```
*********************************************************************

#case              // Da loči male in velike źrke
#ZERO_RAM      // Briçe RAM po startu programa
/***********************************/

#include <16F84.h>
#USE DELAY (CLOCK=4000000)
#fuses XT,WDT,PUT,NOPROTECT

void main() {

   set_tris_a (0b11111111);
   set_tris_b (0b11111110);

   while (1) {
         delay_ns(500);
         output_low(PIN_B0);
         delay_ns(500);
         output_high(PIN_B0);
            }   // konec while 1

   }     // konec main
```

Slika 5: Primer programskega jezika v C++
Vir: Lasten

Za približanje strojnega ukaznega jezika človeškemu uporabimo programski jezik kot vmesno stopnjo. Programski jezik ni enak naravnemu jeziku. Na sliki 5 je prikazana programska koda v okolju C++, kjer pišete program z naborom ukazov, ki jih ponuja programsko orodje npr. while. Ta nam predstavlja zanko, z ukazom include pa lahko vključite potrebno knjižnico.

Naravni jezik ima zapletena pravila in obsežna slovnična pravila.
Programski jezik je nedvoumen in formalno opisljiv – je bliže strojnemu jeziku.

Programski jezik mora omogočati:

- **opis problema** (opis podatkov, rezultatov in relacij med njimi). Primer: izračun produkta – opis števil, rezultat – njihov produkt;

- **opis postopka** (opis zaporedja korakov, ki nas pripelje do rezultata). Algoritem za množenje, zapisan v osnovnih korakih.

Programski jezik je zbirka dogovorjenih ukazov, običajno v angleščini, ki opravijo določeno akcijo. Višji programski jezik ni vezan na tip mikroprocesorja. Izbrati morate ustrezen prevajalnik, ki napisan program iz izvorne kode (npr. C) prevede v ustrezno strojno kodo razumljivo procesorju (npr. EXE). Vsak ukaz se prevede v več strojnih kod. Programer potrebuje več programskih orodij. Izvorna koda se natipka v urejevalniku.

Editor – prevede se v modul strojne kode s pomočjo ustreznega prevajalnika.

Compiler – prevedeni modul se skupaj z že predhodno pripravljenimi moduli iz knjižnice (Library) združi v izvršljiv program s pomočjo povezovalnika.

Linker – delovanje programa se testira postopoma s pomočjo iskalca napak.

Debugger – običajno so vsa ta orodja združena v programskem okolju z meniji, npr. DEVCPP, CCSC, Visual Studio NET itd.

2.3 DELITEV PROGRAMSKIH JEZIKOV

Glede na čas prevajanja:

a) **Tolmač** (INTERPRETER – RUN TIME).
b) **Prevajalnik** (COMPILER) tvori kodo za procesor, na katerem deluje tudi sam prevajalnik. (npr. TurboC, Delphi za PC).
c) **Križni prevajalnik** (CROSS COMPILER) pa deluje na razvojnem računalniku (npr. PC-ju) in tvori programsko kodo za drug tip mikroprocesorja, za razne mikrokrmilnike (npr. PICC, BASCOM in industrijske krmilnike (npr. Mitshubishi-Melsec MEDOC ali Siemens STEP 7 – http://www.crocodile-clips.com/en/Crocodile_Technology/.)

Glede na način programiranja:

a) **Postopkovni** – BASIC, C-jezik, PASCAL (primerni za mikrokrmilnike in večje računalnike).

b) **Objektni** – C++, Delphi, JAVA (uporabljajo razrede objektov CLASS).

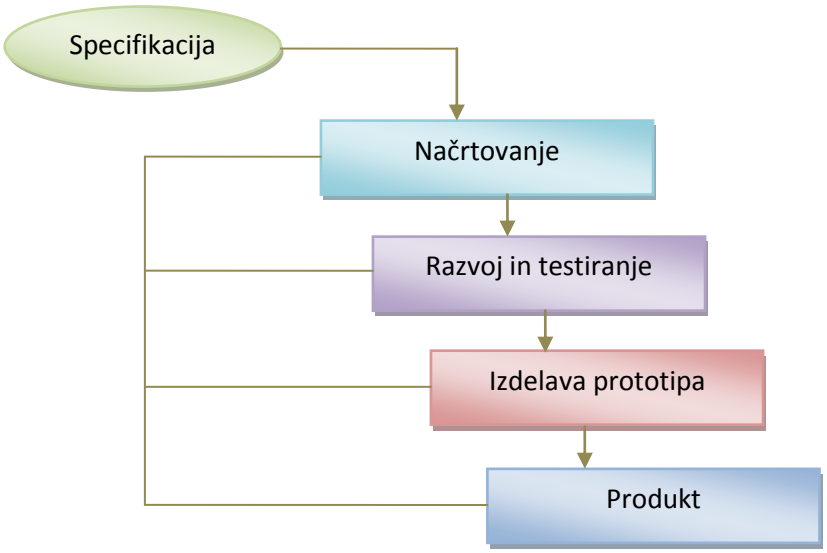
Glede na način vnosa programa:

a) **Besedilni (tekstualni)** – to so do sedaj omenjeni programski jeziki.
b) **Grafični** – Visual Basic, Visual C, LabVIEW.

To so jeziki nove generacije, programiranje poteka v grafičnem vmesniku. Programer zlaga kocke, module, v ozadju pa se tvori ustrezna programska koda.

2.4 PRISTOP K PROGRAMIRANJU

Razvoja programa se lotimo podobno kot razvoja drugih izdelkov ali produktov.
Proces načrtovanja in razvoja:



Slika 6: Proces načrtovanja in razvoja
Vir: Lasten

2.4.1 Faze programiranja

a) Načrtovanje (algoritem, diagram poteka).
b) Kodiranje (pisanje v izbranem programskem jeziku).
c) Prevajanje in testiranje (prevajalnik najde pravopisne napake, debugger pa omogoča, da najdemo tudi logične napake).
d) Dokumentacija programa (komentarji in opisi za poznejše razumevanje napisanega in navodila uporabnikom programa).

2.4.2 Algoritem

Rešitev naloge se izvede v korakih, v **algoritmu.**
Definicija: **Algoritem je koračen opis postopka, s katerim lahko rešimo problem.**

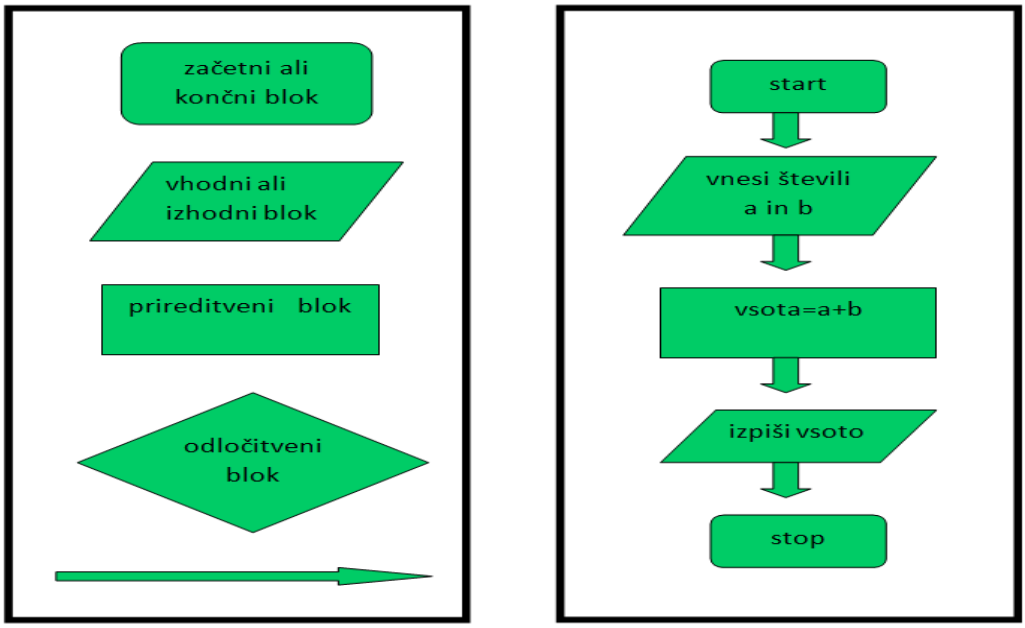
Primer: **Izdelava programa, ki sešteje dve števili**

- Deklariranje spremenljivk a, b, c;
- Vnesite prvo število in ga vpišite v spremenljivko a;
- vnesite drugo število in ga vpišite v spremenljivko b;
- seštejte spremenljivki a in b;
- izpišite spremenljivko c.

2.4.3 Diagram poteka (Flow Chart)

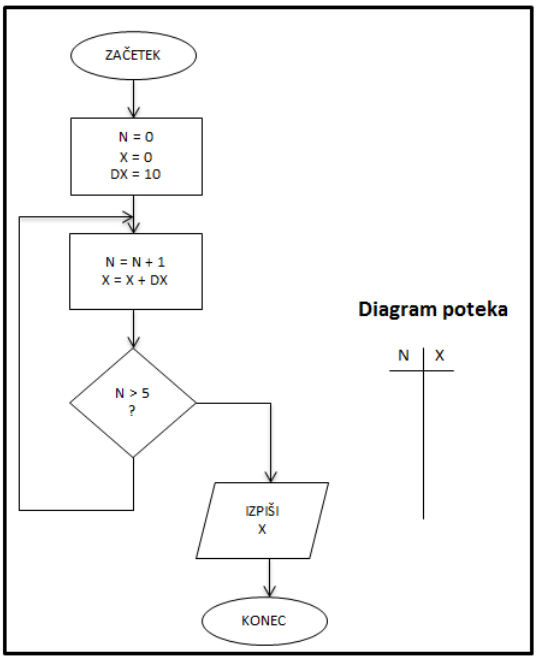
Je grafično prikazan algoritem, torej na človeku razumljiv način opisan potek dogodkov. Uporablja se pri načrtovanju programov in tudi pri opisovanju ostalih dejavnosti (npr. Domači zdravnik v knjižni obliki, Občinska navodila za pridobitev dovoljenj itd).

Pametno se je lotiti programskega problema na način diagrama poteka, da si razjasnite izvajanje po korakih in predvidite, kaj bo moral narediti računalnik.



Slika 7: Bloki diagrama poteka
Vir:Lasten

Nekateri učni programi omogočajo sestavo programa direktno v diagramu poteka. Tako sestavljen program lahko testirate ali prevedete v besedno obliko, kar močno poenostavi učenje programiranja.



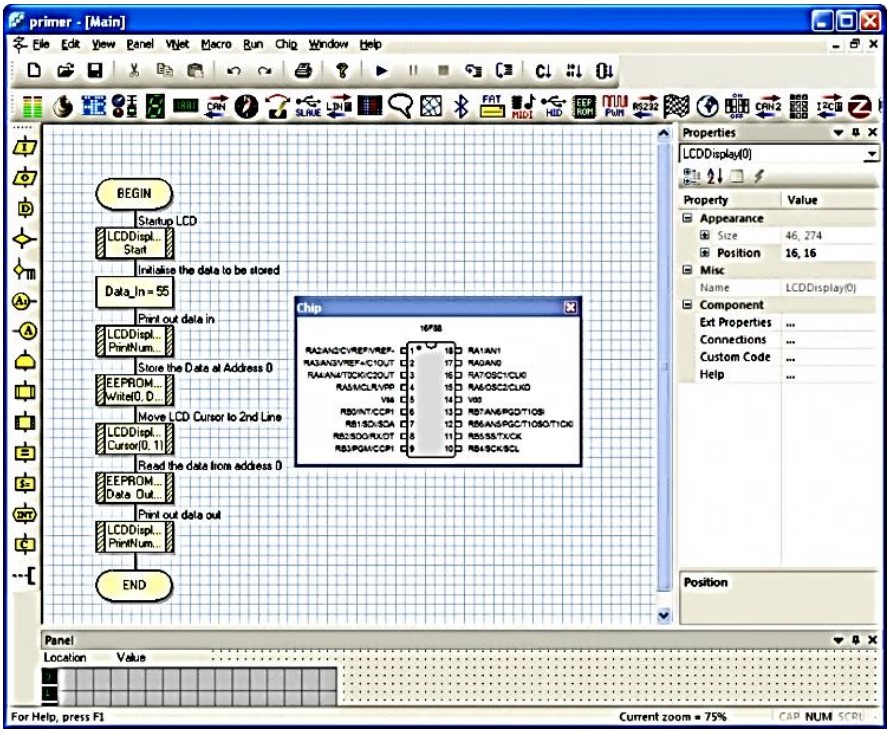
Slika 8: Primer diagrama poteka
Vir: Lasten

2.5 PROGRAMIRANJE MEHATRONSKIH NAPRAV LEGO MINDSTORM IN FLOWCODE

LEGO MINDSTORM in SIMBOT sta zelo primerna za učenje programiranja. Iz LEGO kock sestavite avtomobilček, ki ima vgrajen mikroračunalnik. Temu računalniku napišete program, da bo krmilil motorje in upošteval signale iz senzorjev.

Slika 9: Lego Mindstorm
Vir: http://mindstorms.lego.com/en-us/Default.aspx

Ena izmed možnosti programiranja je tudi Flowcode.

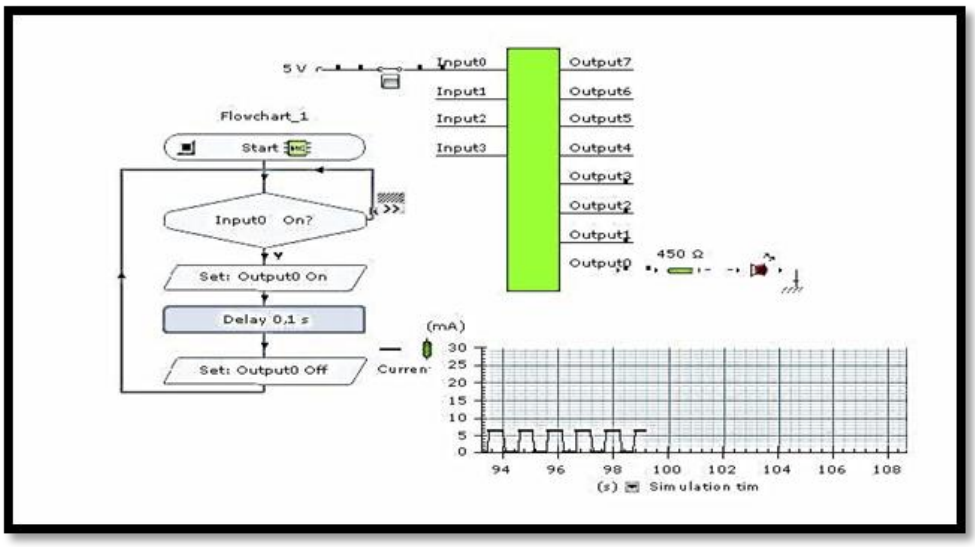


Slika 10: Primer programa v Flowcode
Vir: http://www.imagesco.com/microcontroller/flowcode-compiler.html

2.6 PROGRAM ZA SIMULACIJO CROCODILE TECHNOLOGY

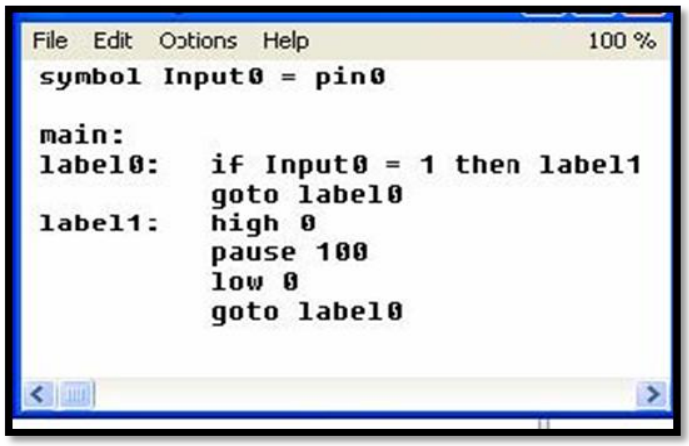
Simulacijski program Crocodile Technology je primeren za učenje programiranja mikrokrmilnika PIC. V programu izdelate električno shemo vezja z elementi iz knjižnice, v naslednjem koraku pa lahko sestavite tudi diagram poteka programa. V diagramu lahko uporabite vse programske ukaze, pogojne skoke, zanke. Prednost, ki nam omogoča takšen način programiranja, je vsekakor večja preglednost in lažja predstavitev dogajanja v programu. Ob zagonu simulacijskega programa je vidna tudi aktivnost posameznega vhoda oziroma izhoda.

Dani program pa vam omogoča tudi meritve vrednosti signalov z virtualnimi merilniki iz izris. Tako lahko npr. opazujete, kaj se dogaja na izhodu mikrokrmilnika.



Slika 11: Primer programskega okolja za programiranje mobilnih robotov
Vir: http://www.crocodile-clips.com/en/Crocodile_Technology/

Program je možno tudi izvoziti v BASIC ali direktno preko programatorja vpisati v realni čip.



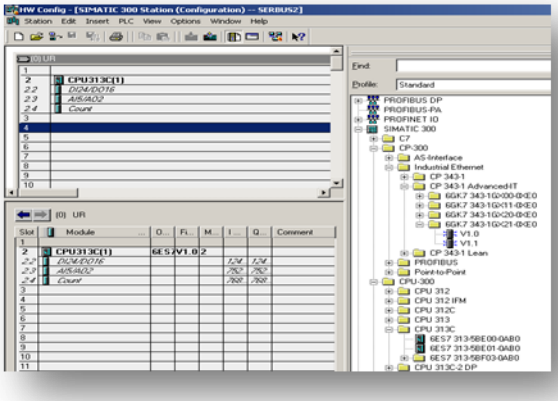
Slika 12: Primer programske kode v Basicu
Vir: http://www.justbasic.com/learnmore.html

2.7 PROGRAMIRANJE MIKROKRMILNIKOV – PLK

Tudi programiranje industrijskih krmilnikov je lahko grafično in hitro razumljivo.

Programska orodja za PLK podpirajo tri načine programiranja:
- preko relejske električne sheme krmilja (LADDER),
- z elementi digitalne elektronike – blok diagram (FBD),
- z besednimi ukazi – instrukcijami (INSTR),
- pa še kakšen dodatni grafični način.

Takšna programska orodja omogočajo nadzor in testiranje programa po nalaganju v krmilnik, nekatera pa tudi simulacijo delovanja kar na PC-ju (off-line).



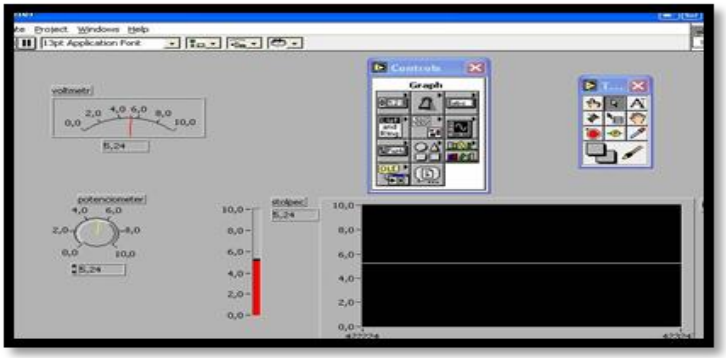
Slika 13: Primer PLK programskih jezikov
Vir: http://support.automation.siemens.com

2.8 PROGRAMIRANJE VIRTUALNE INSTRUMENTACIJE – LABVIEW

PC računalnik lahko uporabite kot industrijski krmilnik pri avtomatizaciji merilnih procesov. V ta namen se dobijo skupaj z aparaturno opremo (merilni vmesniki za PC ipd.) tudi grafična programska okolja za programiranje. V Sloveniji sta zelo razširjena oprema podjetja National Instruments in njihovo programsko orodje LabVIEW (http://www.ni.com/labview/applications/).

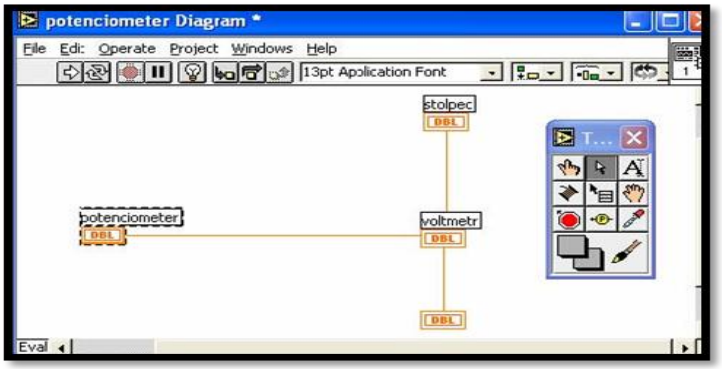
To programsko orodje je namenjeno avtomatizaciji meritev. Potrebujete merilne module in vmesnike, ki jih vgradite v računalnik (DAQ kartice), ti vmesniki pa posredujejo analogne in digitalne signale med merjencem in računalnikom. Uporabni so tudi klasični merilni instrumenti, ki imajo računalniški priključek, saj so lahko zunanje enote.

Izdelava programov je preprosta in pregledna. Sočasno sta odprti dve programski okni: čelna plošča (*Panel*), na katero nanesete in narišete prikazovalnike in gumbe za nadzor, ter blokovna shema (*Diagram*), v katerem v ozadju povežete elemente iz čelne plošče z žicami in omogočite pretok podatkov. Iz bogate knjižnice funkcij izberete za obdelavo merilnih podatkov ustrezne bloke, ki jih povežete v shemo.



Slika 14: Prikaz programiranja v programu LabVIEW
Vir: http://www.ni.com/labview/

Prikazovalnike in kontrolne gumbe za navidezni instrument nanesete na sprednjo ploščo (*Panel*) (VI-Virtual Instrument).



Slika 15: Diagram
Vir: http://www.ni.com/labview/

V ozadju v programskem oknu (*Diagram*) povežete izvore in porabnike podatkovnega toka ter zgradite programske zanke.

Program lahko prevedete v izvršljivo (exe) datoteko ali pa ga zaženete kar na razvojnem okolju. Računalnik bere merjene podatke iz zunanjih merilnih instrumentov ali merilnih vmesnikov ter se na njihovi osnovi odloča za ustrezno akcijo preko krmilnih izhodov iz PC-ja, zapis v datoteko na disk, javljanje na ekranu ali preko interneta.

Delovna orodja izbirate iz palete TOOLS, ki je dostopna v obeh oknih. Pri izgradnjo čelne plošče imate na desni tipki miške dostopno paleto CONTROLS, v blokovni shemi pa na istem mestu paleto FUNCTIONS, v kateri je knjižnica funkcij. Dobra pomoč v angleščini (*Help*) nam olajša delo in odkrivanje napak. Izdelani program iz obeh oken se shrani pod imenom *.VI (Virtualni Instrument). Pri učenju si lahko pomagate z množico že izdelanih primerov.

Program poženemo z  v delovni vrstici.

Če hočemo, da se avtomatsko ponavlja pa z  .

Za boljše razumevanje delovanja vključimo prikaz pretoka podatkov z  .

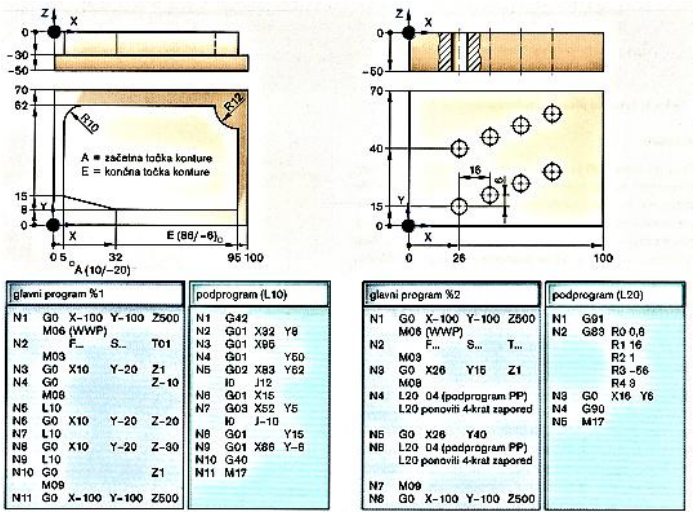
2.9 PROGRAMIRANJE NUMERIČNO KRMILJENIH STROJEV – CNC G-KODA

CNC (Computer Numerical Control) stroj je stružnica ali podoben kovinsko obdelovalni stroj, voden z računalniškim krmilnikom. G kode so standard za numerično krmiljene obdelovalne stroje.

CNC programerji pišejo programe iz tehnoloških risb in jih vnašajo v krmilnik stroja. Dobri strojniški CAD programi lahko iz narisanega 3D objekta avtomatsko tvorijo G-kodo za izdelavo objekta na CNC stroju.

Slika 16: CNC stroj
Vir: Lasten

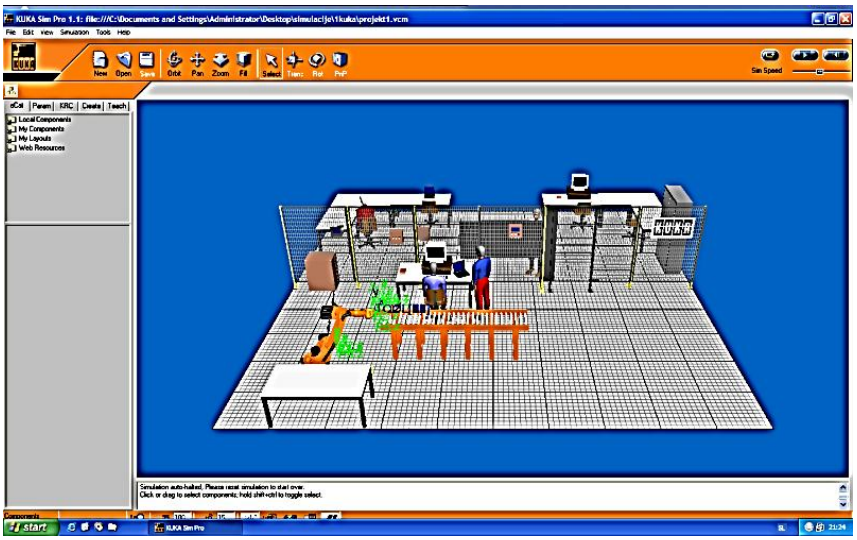
Spodnja slika prikazuje primer kode za programiranje CNC rezkalnega stroja, krmiljenega s PC računalnikom. Pod vsakim od objektov je navedena tudi koda za izdelavo objekta.



Slika 17: Primer CNC kode z vključenimi podprogrami
Vir: J. Bartenschlager, MEHATRONIKA 2009

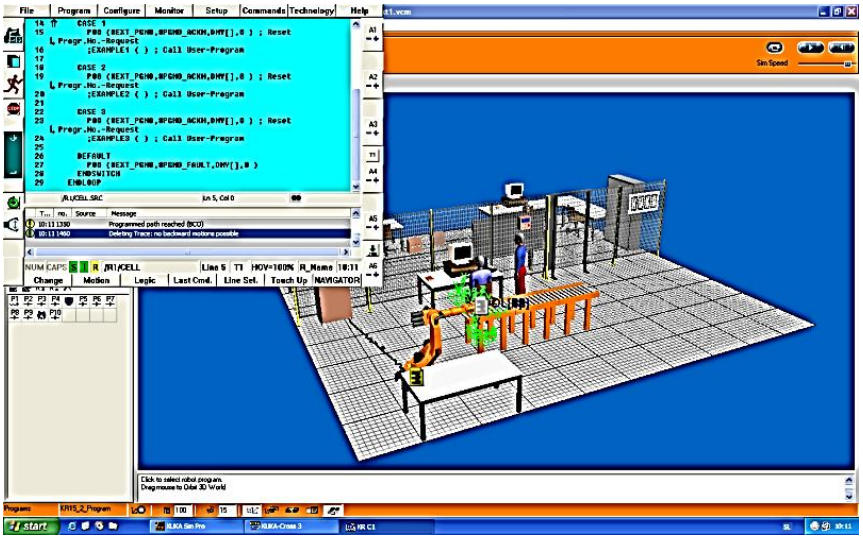
2.10 ROBOTSKI PROGRAMSKI JEZIK: INDUSTRIJSKI ROBOTI KUKA

Simulacijski program omogoča učenje gibov in akcij v simulatorju. Lahko si zgradite robotsko celico in napišete robotski program. Nato ga lahko preizkusite in prenesete na robotski krmilnik. Pred izgradnjo realne robotske celice je dobro napraviti simulacijo in v njej predstaviti vse robne pogoje delovanja.



Slika 18: Simulacijski program KUKA Sim Pro
Vir: http://www.kuka-robotics.com/en/products/software/

Poleg razvojnega programskega okolja lahko povežete tudi simulator s programom Office Lite, ki omogoča simulacijo programirne naprave.



Slika 19: Simulacijski program
Vir: http://www.kuka-robotics.com/en/products/software/

Na zgornji sliki je prikazana povezava simulacijskega programa KUKA Sim Pro in s simulacijskim podprogramom operacijskega panela Office Lite.

V primeru povezave programskega paketa Kuka Sim pro in Office Lite 4.1 lahko na osebnem računalniku izvajate iste akcije kot na realnem robotu. To prikazuje zgornja slika. Lahko izvajate gibe, zanke, nastavljate hitrosti in končani program direktno prenesete preko mreže na robotski KRC krmilnik.

Slika 20: Programirna naprava na podlagi snemanja gibov (Teach Box KR C2)
Vir: Lasten

Povzetek:

Programske jezike lahko v osnovi razdelimo na tri dele, in sicer strojna koda, zbirnik in višji programski jeziki. Strojna koda programa je sestavljena iz zaporedij izvršljivih strojnih ukazov, ki jih na osebnih računalnikih izvaja centralna procesna enota.

Zbirnik (angl. *assembly language, assemble* – sestavljati) je nizkonivojski programski jezik druge generacije (prve generacije je strojna koda), ki je napisan z mnemoniki. Posplošeno velja, da mnemoniki predstavljajo berljive inačice dvojiških zaporedij (ničle in enice), ki jih je potrebno sestaviti, da dobimo centralnemu procesorju razumljivo kodo.

Visokonivojski jezik (*high-level language*) je programski jezik, zasnovan tako, da ustreza programerjevim zahtevam. Ni odvisen od interne strojne kode konkretnega računalnika. Visokonivojske jezike uporabljamo za reševanje problemov in jim velikokrat pravimo tudi problemsko orientirani jeziki.

Pred začetkom programiranja je vsekakor potrebno sam problem, ki ga bomo reševali s programom, dobro razdelati. To napravimo z algoritmom, kjer problem razdelamo na posamezne korake in je tako lažje rešljiv.

Vprašanja:

1. Kaj pomeni pojem »računalniški program« in kaj »programer«?
2. Kakšne so lastnosti programskega jezika in kaj omogoča?
3. Katera programska orodja potrebuje programer in čemu služijo?
4. Razdelite programske jezike in podajte nekaj primerov.
5. Skozi katere faze mora programer pri izdelavi programa?
6. Kaj pomeni beseda algoritem?
7. Napišite algoritem za avtomat za kuhanje kave in ga pojasnite.
8. Kateri simboli so uporabljeni v diagramu poteka in kaj pomenijo?
9. Kateri načini pisanja programa so uporabljeni pri programirljivih logičnih krmilnikih?
10. Kako izgleda program za CNC stroj? Napišite vsaj 3 ukaze.
11. Kako se programirajo roboti? Opišite primer.

3 OSNOVE PROGRAMSKIH JEZIKOV

V poglavju PROGRAMSKI JEZIKI se bomo seznanili s programskimi jeziki in njihovimi osnovnimi ukazi. Ukazi so ponazorjeni s primeri.

Današnji računalniki razumejo le bitni jezik, ki mu pravimo tudi strojna koda (*machine code*). Če torej hočete, da naš računalnik izvede neko nalogo ali opravilo, mu morate to podati v obliki, ki jo razume, torej v enicah in ničlah. Ker pa nam ljudem razmišljanje v bitnem jeziku ni najbolj naravno, se je pojavila potreba po nekem višjem, človeku bolj razumljivem jeziku, ki ga nato lahko prevedete v strojno kodo. Programski jezik nam omogoča, da računalniku podajamo ukaze ter naloge z vsakdanjimi izrazi, kot sta npr. »beri in piši«. Eden izmed razvojnih programskih orodij je jezik C++, ki je vsestransko uporaben. Srečamo ga pri programiranju mikroprocesorjev, izdelavi programov za izpisovanje potnih nalogov, vse pogosteje pa je prisoten pri izdelavi spletnih aplikacij.

3.1 UKAZI V C++

Program je zaporedje ukazov, programiranje pa pisanje ukazov za procesor. Ukazi so v vsakem programskem jeziku drugačni. V tem poglavju bodo opisani ukazi v programskem jeziku C++.

Poleg C++ in C# poznamo še programske jezike, kot so Visual basic, Delphi, Java, SQL, Python, itd.

***Kratka zgodovina C++*:**
Konec sedemdesetih je Bjarne Stroustrup začel eksperimentirati z razširitvijo zelo razširjenega programskega jezika C. Programski jezik C++ je nadgradnja programskega jezika C, nekatere dodatke pa je Bjarne Stroustrup črpal iz že obstoječih jezikov, kot sta Simula67 ter Algol68. Ime C++ se prvič pojavi v letu 1983, v drugi polovici osemdesetih let pa C++ doživi korenite spremembe.

Ukazi v C++:

Glavni program (blok ukazov)

Vsi programi morajo imeti glavni program (*main*), drugače se ne morajo izvajati. To je jedro, v katerega boste začeli programirati. V vsak program vključite v metodo *main(),* ki ji sledita zavita oklepaja {}. V program morate navesti začetek in konec programa, za blok ukazov pa je primerno, da je unikatno označen.

```
int main()   (začetek glavnega programa)
{                          - začetek bloka
cout <<  "To je program v c++" ;
return 0;
}                                  - konec bloka
```

Na levi strani je prikazan program, ki nam izpiše stavek na zaslonu – to je program C++.

Knjižnice

Poznamo veliko knjižnic. Knjižice so datoteke, kjer je napisana strojna koda funkcij. Pri pisanju programov morate uporabljene knjižnice vključiti v program npr. *(#include<iostream>)* ter vključiti vhodno izhodni tok podatkov. Spodaj so navedene najpogosteje uporabljene knjižnice:

#include <iostream>	Vhodno-izhodni ukazi (knjižnica uporabljena samo v C++)
#include <stdlib.h>	Standardni ukazi
#include <string>	Za lažje delo z nizi, knjižnica je veljavna le v C++

Komentar

Komentar je namenjen programerju za lažje razumevanje programa. Pri kompleksnejših programih, ki niso dosledno komentirani, pride do velikih težav ob primeru, da želite karkoli spremeniti oz. popraviti. Značilnost komentarja je, da ga prevajalnik preskoči. Komentarje pišete kratko in jedrnato; lahko so enovrstični, ki jih označimo z (//) ali večvrstični, označeni z (/* …..*/).

Spremenljivke

Spremenljivke so košček pomnilnika, ločijo se po tipu ter velikosti. Tip spremenljivke nam pove, kaj vsi ti biti, ki jih neka spremenljivka zasede, sploh pomenijo. Ali gre za črko, število ali kaj drugega.

Pravila za poimenovanje spremenljivk:

- Prvi znak mora biti črka, ne številka.
- Preostali znaki so lahko črke, številke ali podčrtaji "_".
- V imenih ni presledkov.
- Pomembna je velikost črk. Vsakokrat jih morate zapisati povsem enako. Tako npr. *avtomobil* ni isto kot *Avtomobil*.
- Posebnost, pomembna za Slovence – ne uporabljajte šumnikov (č,ž,š).
- Navada je, da imena spremenljivk začnete z malo črko, pri sestavljenih besedah pa vsako naslednjo začnete z veliko. Nekaj primerov: *starostOtroka*, *hitrostAvtomobila*, *velikostNoge*.
- Imena spremenljivk naj bodo smiselna. Uporabljajte besede, ki povedo pomen spremenljivke npr. *premerKroga*, *dolzinaPalice*.

Psevdokoda za deklaracijo spremenljivke:
TIP IME_SPREMENLJIVKE;

Psevdokoda za definicijo spremenljivke:
IME_SPREMENLJIVKE = NEKA_VREDNOST;

Spremenljivka mora biti najprej deklarirana, šele potem jo lahko tudi definirate. Ta dva koraka lahko združimo:
TIP IME_SPREMENLJIVKE = NEKA_VREDNOST;

Najmanjša velikost spremenljivke, ki jo lahko naslovite v programskem jeziku C++, je en (1) byte, torej osem bitov, kar znese 256 (2^8) različnih stanj. Takšen tip se v C++ imenuje **char** (character), vanj pa lahko vpisujemo **cela števila**. Poleg njega poznamo tudi celoštevični **int**, ki ponavadi zasede 4 byte. Poznamo tudi manjše ter večje celoštevilčne tipe, ki zasedejo 2 ali 8 bytov, vendar pa je njihova deklaracija različna od prevajalnika do prevajalnika. Pri definiciji spremenljivk lahko uporabljate tudi predznake.

Primer:

```
 /*
Primeri deklaracij ter definicij celoštevilčnih spremenljivk.
 */
 // deklaracija
 char a;
 int  b;
 // deklaracija ter definicija, uporaba predznaka
 char c =  10;
 int  d =+10;
 char e =-10;
 int f =+10;
```

Program na levi strani prikazuje deklaracijo spremenljivk. Začne se z prikazom celovrstičnega komentarja, sledi deklaracija spremenljivk, npr. spremenljivka b je tipa integer.

C++ pozna tudi **modifikatorje** tipov (samo za celoštevilčne tipe): *short, long, unsigned, signed*. **Short** in **long** se uporabljata v navezi s tipom **int**, določata pa velikost le-tega (odvisno od posameznih prevajalnikov).

Psevdokoda:

MODIFIKATOR TIP IME_SPREMENLJIVKE;
MODIFIKATOR TIP IME_SPREMENLJIVKE = NEKA_VREDNOST;

Primer:

```
// uporaba short ter long

shortint majhno = 100;      // velikost spremenljivke naj bi bila 2 byta

longint veliko = 10000;    // velikost spremenljivke naj bi bila 8 bytov
```

Pri uporabi modifikatorjev lahko tip **int** izpustite, saj prevajalnik ugotovi, da gre za celoštevilčni tip **int.**

Modifikatorja **signed** in **unsigned** pa določata, ali imajo naši celoštevilčni tipi tudi predznak, oz. ali lahko z njimi zapisujete tudi negativna števila. Vsi osnovni celoštevilčni tipi znajo zapisovati tudi negativna števila (privzeti način). Na primer tip **char**. Ker je velik 1 byte, lahko z njim predstavite 256 različnih možnosti (stanj). Tako lahko z njim zapišete števila od −128 do +127 (tudi 0). Če na tipu **char** uporabite modifikator **unsigned**, negativna števila odpadejo, tako da lahko vanj zapišete števila od 0 do 255. To velja tudi za vse druge celoštevilčne tipe.

Primer:

```
// uporaba modifikatorjev signed ter unsigned

char a;          // -127 do +128, signed je privzeta vrednost
signedchar b;    // -127 do +128
unsignedchar c;  //    0 do 255
int d;           // - 21474836648 do +21474836647
unsignedint e;   //    0 do 42944967296
```

Lahko zapisujete tudi črke ter pripadajoče simbole. Imate več možnosti. Nekako najbolj uporabljena sta **ASCII** in **UNICODE** zapis. Zapis simbola po ASCII tabeli je velik en byte, medtem ko UNICODE uporablja za zapis 2 byta. UNICODE zapis je primeren predvsem za razvijanje prenosljivih programov za tuje trge (oz. jezike). ASCII pa nam nudi preprost ter predvsem manj pomnilniško požrešen zapis. ASCII zapis uporablja tako imenovano kodno tabelo, ki je odvisna od nastavitev sistema (jezik ter država, od tod izvirajo težave s šumniki). UNICODE pa uporablja drugačen zapis, namreč vsak simbol ima svojo unikatno vrednost. Črke in simboli so v računalniku zapisani kot številke, šele ko jih začnemo "gledati" ali izpisovati, dobijo svojo pravo obliko, ki jo mi razumemo. Tako, na primer, ima velika črka A vrednost 65, črka B vrednost 66 itd.

Primer:

```
char    a ='A';        // ascii zapis
wchar_t b ='B';    // unicode zapis
char    c = 65;     // ascii zapis, tokrat uporabim kar ascii vrednost ('A')
```

Poleg teh osnovnih tipov pozna C++ tudi logični **bool** tip, ki je po velikosti in zgradbi enak tipu **char**, ter tako imenovani ničelni tip **void** (nič).

Imena spremenljivk ne smejo imeti enakih imen kot ključne besede.
Seznam vseh ključnih besed, ki jih ne smemo uporabiti za imena spremenljivk:
asm, auto, break, case, catch, char, class, const, continue, default, delete, do, double, else, enum, extern, float, for, friend, goto, if, inline, int, long, new, operator, private, protected, public, register, return, short, signed, sizeof, static, struct, switch, template, this, throw, try, typedef, union, unsigned, virtual, void, volatile, wchar_t, while itd.

Imena spremenljivk se tudi ne smejo začeti s številko (npr. *100abcd*), lahko pa vsebujejo številke na katerikoli drugi poziciji (npr. *abcd123).*

Odločitev (IF stavek)

Tako kot v vsakdanjem življenju, kjer so potrebne nenehne odločitve, uporabljate tudi v programiranju odločitveni stavek (*if); primer:* če je spremenljivka a večja kot spremenljivka b, izpišite vrednost c. Pri pogojnem stavku imamo dve vrednosti rešitve, in sicer vrednost pravilna *(true)* in vrednost ni pravilna *(false).* Pogoj je trditev (zapis), ki lahko vsebuje oklepaje, vrednosti različnih tipov, spremenljivke, konstante različnih tipov in operatorje. Operatorji so posebni simboli, s katerimi lahko kombinirate spremenljivke oz. vrednosti.

Prva skupina operatorjev predstavlja matematične operacije, kot npr. seštevanje, odštevanje, množenje in deljenje.

Tabela 1: Skupina operatorjev

Operator	Ime	Primer	Povedano z besedami
+	plus	1 + 2	ena plus dva
-	minus	7 – 3	sedem minus tri
*	krat	5 * 2	pet krat dva
/	deljeno	9 / 3	devet deljeno s tri

Vir: Lasten

Relacijski operatorji primerjajo med seboj dve vrednosti, da bi ugotovili, ali sta enaki ali pa je prva večja ali manjša od druge.

Tabela 2: Skupina relacijskih operatorjev

Operator	Pomen	Primer	Povedano z besedami
==	je enako	a == b	a je enako b
!=	ni enako	c != d	c ni enako d
<	manjše kot	x = 10	x je manjši od 10
>	večje kot	y = 0	y je večji od 10
<=	manjše ali enako kot	starost <= 25	starost je manjša ali enaka 25 (vključuje 25)
>=	večje ali enako kot	višina >= 180	višina je večja ali enaka 180 (vključuje 180)

Vir: Lasten

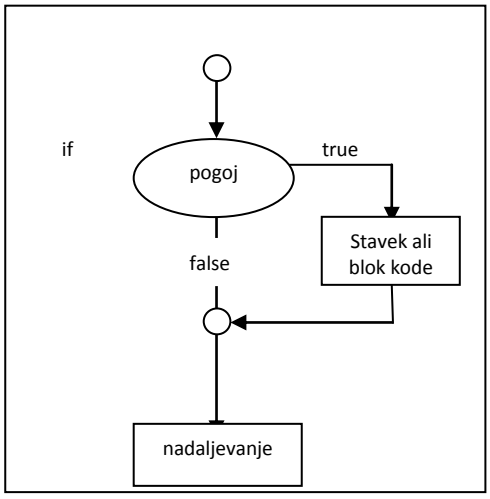
Pogoji se delijo na:
- enostavne,
- sestavljene (morajo vsebovati vsaj 1 logični operator).

Ukazi, napisani v okviru pogojnega stavka, se izvedejo, če je izpolnjen podan logični pogoj. Pogojni stavek IF uporabite, ko želite, da se neka koda izvede le, če je pogoj izpolnjen.

Splošna oblika IF stavka je naslednja:

```
 if (pogoj)          // DA akcije
{
          true
 }  else             // NE akcije
{
          false
 }
```

V primeru, da je pogoj odobren in je vrednost true, se bo akcija izvedla.
V primeru, da pogoj ni zadoščen, se bo program nadaljeval brez programske akcije.



Slika 21: Primer IF zanke
Vir: Lasten

Primer programa z if: (program prešteje in izpiše števila od 1 do 10):

```
#include<iostream>
using namespace std;
int main()
{
int stevec;
stevec=1;

nazaj:

cout<<stevec<<endl;
stevec=stevec+1;
if(stevec>10)
{
system("Pause");
return 0;
}
else
{
goto nazaj;
}
}
```

Program na levi prešteje in izpiše števila od ena do deset. Najprej v program vključite vhodno izhodni tok podatkov (iostream), nato se začne izvajati glavni program. Deklarirate spremenljivko števec in jo postavite na vrednost ena. Vrednost spremenljivke števec se povečuje za ena in to tako dolgo, dokler ni vrednost spremenljivke števec večja od deset.

Izpis

V C++ poznamo ukaz **cout,** s katerim izpišete neko besedilo ali številke na ekran. Podobna ukaza coutu sta *izpisit (*izpis teksta) *ter izpisist (*izpis števil). Ta dva ukaza uporabljate v »podprogramih«, ki bodo opisani nekoliko kasneje.

Zgradba:

```
cout<<" Neko besedilo ki se izpise na ekran "<<endl

Primer izpisa na ekran:

#include<iostream>
using namespace std;
int main()
{
cout<<"To je izpis na ekran (cout)"<<endl;
system("pause");
return 0;
}
```

Branje

Za branje v C++ uporabljate ukaza **cin** in **getline.**

Cin bere do prvega praznega znaka in zato ni primeren za večznakovni tip. Ta problem reši ukaz **getline, ki je v knjižnici string.**

Imate različne možnosti, in sicer:
getline(cin,ime_prostora); → berete, dokler ne stisnete enter – do konca vrstice;
getline(cin, ime_prostora,znak); → berete, dokler ne naletite na pravilen znak;
getline(ime_prostora,100); → berete tolikokrat, kolikor je napisano(v tem primeru 100);
getline(ime_prostora;500,b); → berete, dokler ne pridemo do b, lahko je največ 500 znakov, več ne, v tem primeru, da je znakov več kot 500, lahko berete do 500 znakov, ali pa do takrat, ko pridemo do znaka b).

Primer programa:

```
#include <iostream>
using namespace std;

int main () {
char name[256], title[256];

cout << "Vase ime je: ";
cin.getline (name,256);

cout << "Vas najljubsi film: ";
cin.getline (title,256);

cout << name << " vas najljubsi film je: " << title;

system("pause");
return 0;
}
```

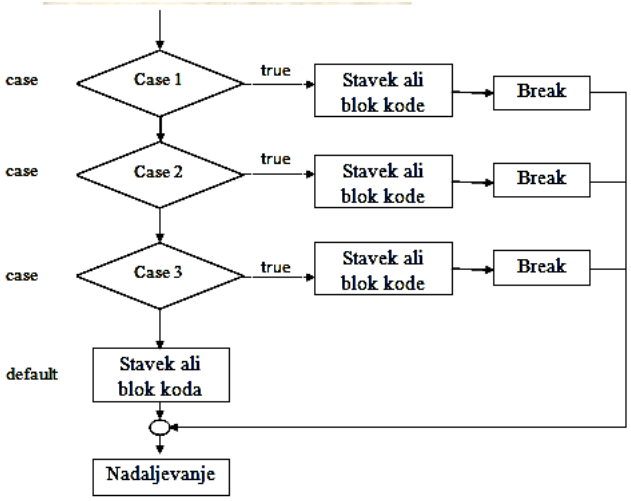
Izbira

Izbira – izberete med več variantami.

V C++ obstaja ukaz za izbiro, ki se imenuje **switch/case**.

Zgradba switch case:

```
switch (izraz)
{
case vrednost A; blok ukazov A
break;
case vrednost XY; blok ukazov XY
break;
default;              // ni obvezen
akcija default;
```



Slika 22: Prikaz ukaza izbira (switch/case)
Vir: Lasten

Ukaz:
break – preskoči celo strukturo;
continue – en korak, trenutno akcijo prekine;
goto– ta stavek uporabite, kadar ne želite, da se stavki v programski kodi izvajajo po vrsti (ukaza goto se izogibamo, ker lahko vodi k nestrukturirani kodi!!!).

Primer programa s switch:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
   int a;
nazaj:

   cout<<"1. Ime"<<endl;
   cout<<"2. Priimek"<<endl;
   cout<<"3. Razred"<<endl;

 cout<<"Izhod iz programa"<<endl;
   cout<<"Vnesi stevilko za prikaz podatkov: "<<endl;
   cin>>a;

   switch(a)
   {
         case 1:
           cout<<"Alen\n";
                                 break;

                                 case 2:
                                 cout<<"Korosec\n";
                                 break;

                                 case 3:
                                 cout<<"2.a\n";
                                 break;

                                 case 9:
                                 cout<<"Izhod iz programa\n";
         system("pause");
         return 0;

         default:
         cout<<"Stevilo ni veljavno! Vnse drugo"<<endl;

   }
   system("pause");
   system("cls");
   goto nazaj;
```

Polje (Array)

Polja so skupine enakih objektov ali podatkov. Oštevilčene so od 0 naprej. Predstavljena bo skupina celih števil (int). Pozor na format zapisa: {12, 7, 32, 15, 113, 0, 7}.
Poglejmo, kako bi bile te vrednosti zapisane v računalniku, zraven pa podajmo še tekoče številke posameznih podatkov v skupini:

Tabela 3: Prikaz vrednosti zapisanih v polju

indeks	0	1	2	3	4	5
vrednosti	12	7	32	15	113	7

Vir: Lasten

Tekočo številko podatka v takem polju imenujemo indeks. Če želite dobiti posamezni element polja, morate navesti ime polja, v oglatih oklepajih [ ], za imenom pa zapisati številko indeksa. Recimo, da daste takemu polju ime *tockeTekmovalcev*.

Polje je podatkovna struktura, ki se uporabi takrat, ko imate več spremenljivk istega tipa in za isti namen.

TIP_IME[n] (deklaracija polja)
n – indeks – število elementov
n-ti indeks je v C++ prepovedan!

Primer programa s poljem: (program izpiše števila od 0–9)

```
#include<iostream>
using namespace std;

int main()
{
int polje[10];
int i;

for(i=0;i<10;i=i+1)
{
polje[i]=i;
}

for(i=0;i<10;i=i+1)
{

cout<<polje[i]<<endl;
}

system("pause");
return 0;
}
```

Program na levi strani prikazuje uporabo polja.
Deklarirano je polje tipa integer in spremenljivka i, prav tako tipa integer.

For zanka pravi:
dokler je i manjši od deset, povečujte i za ena oz. napravite inkrement i-ja ter vrednosti zapisujte v polje. Z ukazom *cout* izpišete vrednosti iz polja.

Zanke

Zanka je osnovni ukaz, ki ga uporabite takrat, ko se del kode (zaporedno) ponavlja za isti namen.

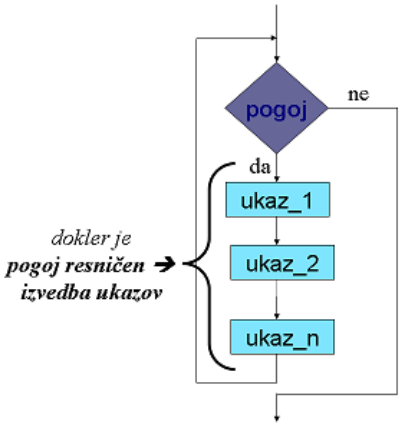
Teoretičen opis zank:
Vsaka zanka ima glavo in telo. V glavi se odločite ali boste ukaz ponavljali ali ne, v telesu pa so ukazi za ponavljanje. V glavi je pogoj, v pogoju pa vsaj ena spremenljivka, na katero lahko vplivate.

For Zanka

For zanka se ponavlja, dokler ni pogoj true. Primerna je takrat, ko je vnaprej znano število ponavljanj (ne izvede se, če pogoj ni izpolnjen).

Oblika For zanke:

```
for ( ;pogoj; )

{
Blok ukazov
Vpliv na spremenljivke, ki se nahajajo v pogoju
 }
```



Slika 23: Primer For zanke
Vir: Lasten

Primer programa s for zanko (program 10 x izpiše ime na ekran):

```
#include <stdio.h>
#include <iostream>
using namespace std;

int main()
{
int index;

for(index = 0 ; index < 10 ; index = index + 1)
cout<<"Alen"<<endl;
system ("pause");
return 0;
}
```

Program na levi strani prikazuje delovanje for zanke.
For zanke so v bistvu zgoščen zapis za:

- iniciacijo števca (ali drugega pogoja),
- nastavljanje in preverjanje pogoja in
- povečevanje števca.

While zanka

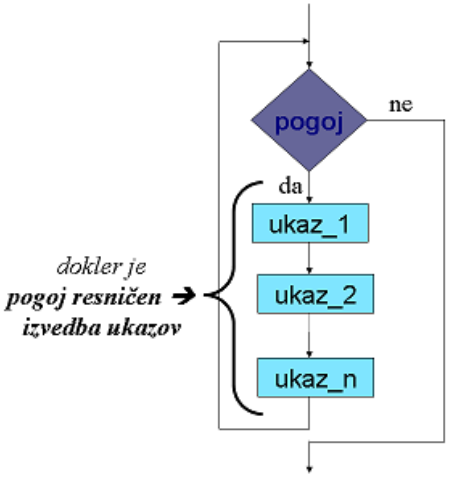
Oblika While zanke:

```
while (pogoj)
{
Blok ukazov
}
```

While zanka nadaljuje z izvajanjem, dokler je nek pogoj resničen. Ko postane pogoj neizpolnjen, se zanka prekine. Torej zanka počne ravno tisto, kar je razvidno že iz njenega imena.

Primer programa z While zanko (program 10 x izpiše ime na ekran):

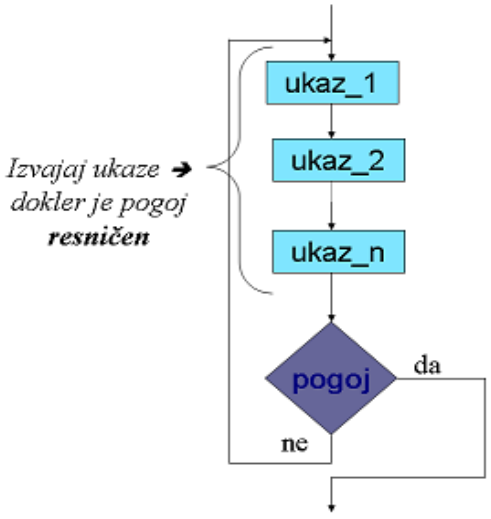
```
#include <stdio.h>
#include <iostream>
using namespace std;

int main() /
{
int count;

 count = 0;
 while (count < 10)
{
 cout<<"Alen"<<endl;
 count = count + 1;
 }
system ("pause");
return 0;
}
```



Slika 24: Primer While zanke
Vir: Lasten

Do While zanka

To zanko je priporočljivo uporabiti takrat, ko podatke že imamo in se na osnovi njih odločamo, ali bomo ponavljali ali ne. Zanka ponavlja, dokler je pogoj true. Do While zanka se za razliko od While zanke izvede vsaj enkrat, pa četudi pogoj ne bo izpolnjen.



Slika 25: Primer Do While zanke
Vir: Lasten

Oblika Do While zanke:

```
do
{
}while();
```

Primer programa z Do While zanko (program 10 x izpiše ime na ekran):

```
#include <stdio.h>
#include <iostream>
using namespace std;

int main() /
{
int i;

  i = 0;

  do
  {
    cout<<"Alen"<<endl;
    i = i + 1;
  } while (i < 10);
system("pause");
return 0;
}
```

V danem programu je uporabljena Do While zanka. Zanka se izvaja tako dolgo, dokler je spremenljivka i manjša od vrednosti 10. Vsakič, ko se zanka izvede, se spremenljivka i poveča za 1.

Podprogrami

Ob izvajanju glavnih programov prihaja do klicanja podprogramov. Večinoma se podprogrami uporabljajo, če se del kode ponavlja na različnih mestih in če se koda ponavlja zaporedno, a vsakič z različnimi podatki. Značilnost podprograma je, da mora delovati sam zase in ne sme biti odvisen od ostalih programov v kodi. Podprograme delimo v procedure in funkcije (procedura – ime podprograma ne uporabljamo za prenos rezultatov, funkcija – ime podprograma uporabimo za prenos rezultata).

Zgradba podprograma:

* GLAVA (V glavi je tip podprograma ter ime podprograma).

Parametri – so spremenljivke, uporabljene zgolj v podprogramu (v glavi).

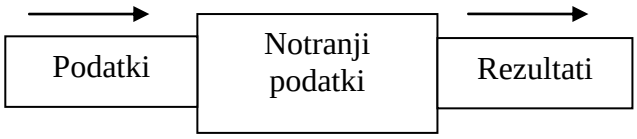
* TELO

- se začne z znakom za začetek bloka→ { in konča z znakom za konec bloka → };
- v telesu je koda (v kodi je lahko vse, kar poznamo, razen začeti ne smemo novega podprograma).

Delitev podprogramov:

- Procedure (ime podprograma **ne** uporabljamo za prenos rezultatov);
- Funkcija (ime podprograma uporabimo za prenos rezultatov).

Prenos podatkov v podprogram:



a) Globalne spremenljivke.

b) Prenos preko parametrov:
- po vrednosti (spremembe podatkov se ne ohranijo),
- po referenci (spremembe podatkov se ohranijo).

Referenca vpliva na mesto klicanja.

c) Preko datoteke;

d) Nič ne prenašamo in samo čitamo.

Prenos podatkov iz podprograma:

a) Globalna spremenljivka:
- vsaka sprememba se ohrani tudi izven podprograma.

b) Raba parametrov:
- vsaka sprememba parametra vpliva na spremenljivko, ki je dala podatke temu parametru.

c) Uporaba datoteke.

Vsak programski jezik ima poseben ukaz, da lahko s pomočjo imena funkcije prenesemo rezultat izven podprograma. V C++ je to ukaz **return.**

Povzetek:

Program je zaporedje ukazov, programiranje pa pisanje ukazov za procesor. Ukazi so v vsakem programskem jeziku drugačni, v tem poglavju so opisani ukazi v programskem jeziku C++.Poznamo veliko knjižnic za različne aplikacije. Knjižice so datoteke, kjer je napisana strojna koda funkcij in jih moramo vključiti v program, če želimo te datoteke v programu uporabiti. Spremenljivke so košček pomnilnika, ločijo se po tipu ter velikosti. Tip spremenljivke nam pove, kaj vsi ti biti, ki jih neka spremenljivka zasede, sploh pomenijo. Vsaka zanka ima glavo in telo. V glavi se odločamo ali bomo ponavljali ali ne. V telesu pa so ukazi za ponavljanje. V glavi je pogoj, v pogoju pa vsaj ena spremenljivka, na katero lahko vplivamo.

Vprašanja:

1. Naštejte lastnosti glavnega programa.
2. Kaj so knjižnice?
3. Kaj nam pove tip spremenljivke?
4. Kaj se zgodi, če na tipu char uporabimo modifikator unsigned?
5. Razložite razlike ASCII in UNICODE zapisov modifikatorjev.
6. Zapišite primer izpisa na ekran.
7. Napišite program, ki bo zahteval vnos dveh števil in bo izračunal ter izpisal vsoto, produkt, razliko in količnik.
8. Kaj je zanka? Naštejte nekaj zank in opišite Do While zanko.
9. Narišite diagram poteka in napiši program, ki bo zahteval vnos 10 števil in jih bo izpisal po vrstnem redu od največjega do najmanjšega.
10. Katere tipe spremenljivk poznamo v programskem jeziku C++?
11. Komentirajte spodnji program:

```
#include<stdio.h>
int main ()
{
    int a,b,c;

    a=5;
    b=6;
    c=7;
printf(″Spremenljivke:  %d,  %d,  %d,  a,  b,  c =;
```

4 MIKROPROCESOR

V poglavju MIKROPROCESOR boste spoznali osnove mikroprocesorja in njegove sestavne dele (aritmetično logična enota, krmilna enota, register). Predstavljeno je tudi delovanje mikroprocesorja in osnovne funkcije, na kratko pa je prikazano tudi programiranje mikroprocesorja.

Mikroprocesor je osnovni element, okoli katerega je zgrajen mikroračunalnik. Narejen je kot integrirano vezje z visoko stopnjo integracije. Je osrednji del računalnika, ki obdeluje podatke, nadzoruje in upravlja ostale enote. Izveden je v enem samem čipu, to je integriranem elektronskem vezju, ki je izdelano na silicijevi rezini. Oznaka modela mikroprocesorja veliko pove o njegovi procesni zmogljivosti.
V splošnem ga sestavljajo:

- aritmetično logična enota ALE,
- krmilna enota KE,
- registri,
- notranje vodilo.

4.1 ARITMETIČNO LOGIČNA ENOTA

Je enota z naslednjimi funkcijami:

- seštevanje, odštevanje, množenje, deljenje;
- pomik vsebin registrov;
- komplementiranje vsebin registrov;
- povečevanje in zmanjševanje vsebin določenih registrov za 1;
- logične operacije (negacija, konjunkcija, disjunkcija, ekvivalenca, antivalenca).

ALE je pridružen register stanj, v katerem so posamezni, med seboj neodvisni flip-flopi (zastavice), ki s svojimi stanji 0 ali1 opisujejo stanja v ALE.

Ta so lahko:

- rezultat aritmetične ali logične operacije je enak nič (bit Z – Zero);
- rezultat aritmetične operacije je negativen (bit N – Negative);
- rezultat aritmetične operacije oziroma njegov bitni zapis presega kapaciteto ciljnega registra (bit OV – Overflow), bit za prekoračitev;
- pri aritmetični operaciji je prišlo do prenosa (bit C – Carry);
- procesor pri izvajanju programa dovoljuje ali pa ne prekinitve (bit I – Interrupt).

4.2 KRMILNA ENOTA

KE vodi delovanje uP, tako da posreduje v skladu s sprejeto instrukcijo (ukaz programa) krmilne signale ostalim enotam. Izvajanje ene instrukcije ali ukaza lahko razdelimo na dve fazi:

- faza dostavljanja instrukcije (prevzem ukaza – fetch),
- faza izvajanja instrukcije (izvršitev ukaza – execute).

V prvi fazi prinaša KE inštrukcijo v instrukcijski register in dekodira operacijsko kodo. V drugi fazi pa v odvisnosti od operacijske kode spreminja stanja v uP in pošilja ustrezne krmilne signale.

4.3 REGISTRI

Število registrov in njihove oznake so pri raznih izvedbah uP različne. V grobem ločimo registre, ki so programsko dostopni in programsko nedostopni. Programsko dostopne registre lahko v programu naslavljate in tudi spreminjate njihove vsebine.

a. Akumulator ACC:
V 8-bitnem uP je 8-bitni register, preko katerega poteka največ operacij v procesorju:
- shranjevanje operandov, nad katerimi izvaja uP aritmetične in logične operacije;
- shranjevanje rezultatov teh operacij;
- posredništvo pri prenosu iz ene v drugo pomnilniško lokacijo ali iz vmesnikov v pomnilnik oz. obratno.

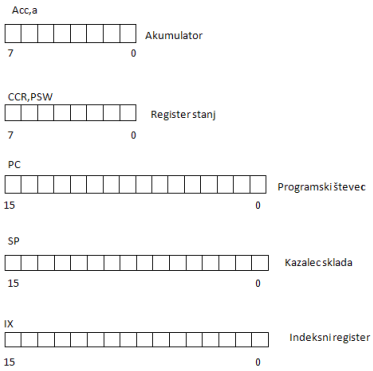
b. Programski števec – PC:
Ta 16-bitni register (*Program Counter*) vodi procesor skozi program. V njem je vpisan naslov naslednje instrukcije, ki jo bo začel izvajati procesor, ko bo zaključil tekočo instrukcijo.

c. Kazalec sklada – SP:
Poseben prostor v podatkovnem pomnilniku, ki se začenja z vnaprej določeno lokacijo, imenujemo sklad. Kazalec sklada (*Stack Pointer*) je 16-bitni register, v katerem je vpisan naslov prve prazne lokacije v skladu pri vpisovanju in zadnje polne pri čitanju iz sklada.

d. Indeksni register – IX:
Je 16-bitni register, ki ga uporabljamo za:
- shranjevanje naslovov pri indeksnem naslavljanju pomnilniških lokacij,
- števec programskih obhodov v zanki določenega programa.

Primer:
Sešteti morate N podatkov. V tem primeru boste v IX vpisali začetni naslov in ga po vsaki izvedeni operaciji povečali za 1, dokler ne boste sešteli vseh N podatkov.

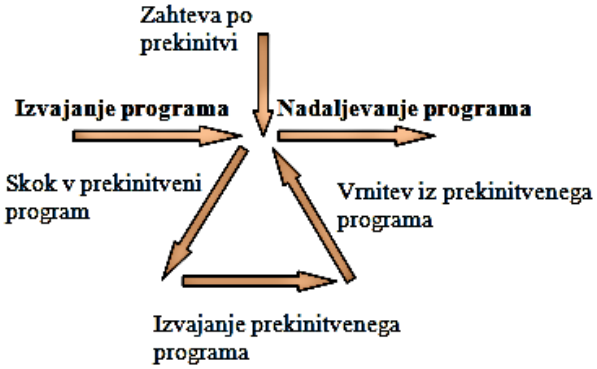
e. Register stanj – R, PSW:
Je 8-bitni register, ki opisuje stanje ALE (*Condition Code Register, Program Status Word*). Glej pri ALE.



Slika 26: Prikaz registrov ALE
Vir: Lasten

4.4 PREKINITVE (INTERRUPT)

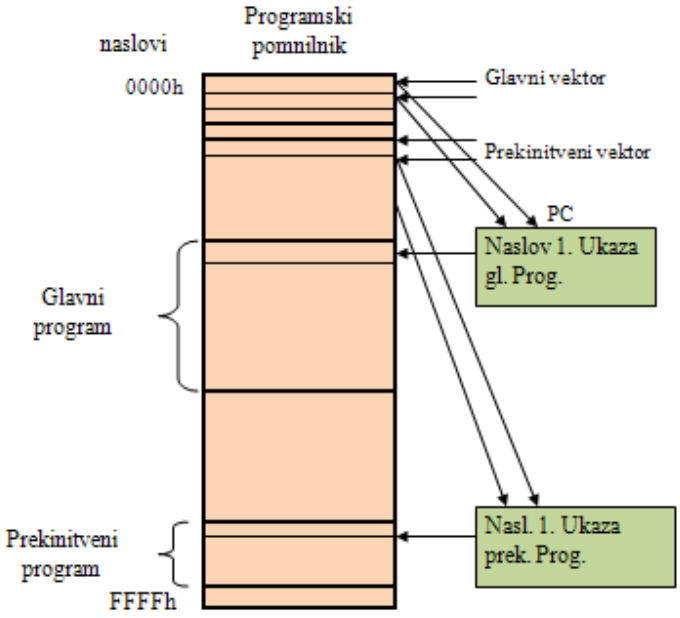
Ena od zelo pomembnih lastnosti vsakega uP je njegova sposobnost, da na zunanjo ali notranjo zahtevo prekine izvajanje trenutno izvajajočega programa in prične izvajati **prekinitveni servisni program**. Ob zaključku slednjega se vrne na prekinjeno mesto in nadaljuje z izvajanjem prekinjenega programa.



Slika 27: Prikaz delovanja prekinitev
Vir: Lasten

Osnovna zahteva pri prekinitvah je, da se po vrnitvi iz prekinitvenega servisnega programa v CPE vzpostavi stanje, ki je popolnoma enako tistemu ob nastopu prekinitve. Pravimo, da morajo biti prekinitve **transparentne – nevidne**. S tem dosežete, da se prekinjeni program prekinitev ne zaveda in da so njegovi rezultati enaki, ne glede na to, ali je bil prekinjen ali ne in tudi ne glede na to, v kateri točki je bil prekinjen.

Vektorske prekinitve
Večina uP uporablja to zvrst prekinitev, kjer je vektor pomnilniški naslov v programskem pomnilniku, na katerem je shranjen naslov prekinitvenega servisnega programa. Ker gre za naslov, se imenuje tudi **prekinitveni naslovni vektor**. Če je možnih več prekinitvenih virov, ima vsak vir svoj prekinitveni vektor.



Slika 28: Prikaz zasedbe programskega pomnilnika
Vir: Lasten

4.5 VIRI PREKINITEV

Zahtevo po prekinitvi lahko uP sporočajo različni viri.

Tako poznamo:
- zunanje vire (signali),
- notranje vire (časovniki, števci),
- serijsko komunikacijo.

4.6 MASKIRANJE PREKINITEV

Zahteva za prekinitev se servisira le, če je to dovoljeno (prekinitev omogočena), v nasprotnem primeru se zahteva ignorira in govorimo o maskiranju prekinitve. Ali neko prekinitev dovolimo ali ne je odvisno od tega, kaj v programu vpišemo v register za dovolitev prekinitev IE (*Interrupt Enable*), ki je sestavljen iz posameznih omogočitvenih bitov za posamezni prekinitveni vir.

Prioriteta prekinitev

V primeru istočasnega nastopa dveh ali več zahtev po prekinitvi imajo nekateri viri prednost pred drugimi. To podaja tabela prioritete proizvajalca uP. Ta določa tudi, ali lahko zahteva iz določenega vira prekine delovanje uP, če je v teku že določeni prekinitveni servisni program. Tudi prioriteto lahko določate programsko v registru IP (*InterruptPriority*).

Servisiranje prekinitve

Če je prekinitev zahtevana in omogočena, se začne servisiranje prekinitve:
- na sklad se shrani vsebina vseh pomembnih registrov, razen SP:
  - vsebina PC, da shranimo naslov ukaza, pri katerem je prišlo do prekinitve, oz. naslov naslednjega ukaza prekinjenega programa;
  - vsebina ACC, da shranimo trenutni rezultat oz. operand pred prekinitvijo;
  - vsebina registra stanj, da shranimo trenutno stanje ALE pred prekinitvijo;
  - vsebina IX, da shranimo v njem zapisan naslov pred prekinitvijo.

- v PC se prenese vsebina viru pripadajočega prekinitvenega naslovnega vektorja, ki je zahteval prekinitev;

- izvrševanje ukazov viru in vektorju pripadajočega prekinitvenega servisnega programa. Mikroprocesor ob vsaki prekinitvi sam shrani na sklad vsebino pomembnih registrov in jih tudi iz sklada vrne v registre. S tem je transparentnost zagotovljena, ne da bi moral uporabnik o njej posebej razmišljati;

- z ukazom, ki je zadnji v prekinitvenem programu in pomeni vrnitev iz prekinitve (*Return from Interrupt*) mikroprocesor, vrne podatke iz sklada nazaj v registre in vzpostavi stanje pred prekinitvijo.

4.7 SKLAD

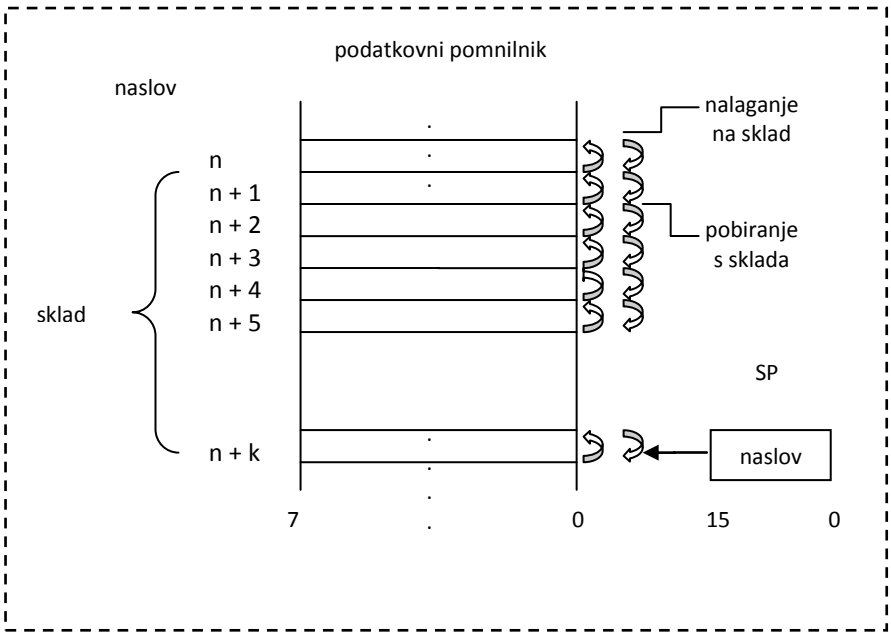
Pri večini uP se nekatere operacije vedno nanašajo na sklad. To so predvsem **prekinitve in klici podprogramov**. Sklad je pomemben tudi kot orodje za realizacijo nekaterih programskih rešitev (rekurzija).

Sklad definirate na naslednji način:

- rezervirate del podatkovnega pomnilnika za sklad. Kako velik del pomnilnika boste rezervirali, je odvisno od obsega uporabe sklada, ta pa od vrste programov, ki jih bo uP izvajal. V vsakem primeru pa morate zagotoviti, da se prostor, rezerviran za sklad, ne uporabi za nič drugega;
- v kazalec sklada vpišete vrednost, ki je enaka najvišjemu naslovu pomnilnika, rezerviranega za sklad.

Značilno za sklad je:

- da se širi od višjih naslovov proti nižjim (ali obratno);
- da SP kaže vedno na naslednjo prazno lokacijo v skladu, v katero lahko uP opravi naslednji vpis oziroma na zadnjo polno pri branju iz sklada;
- delovanje sklada je podobno delovanju registra LIFO (last in first out) – podatek, ki ste ga nazadnje shranili na sklad, je prvi, ki ga lahko preberemo.



Slika 29: Prikaz podatkovnega pomnilnika
Vir: Lasten

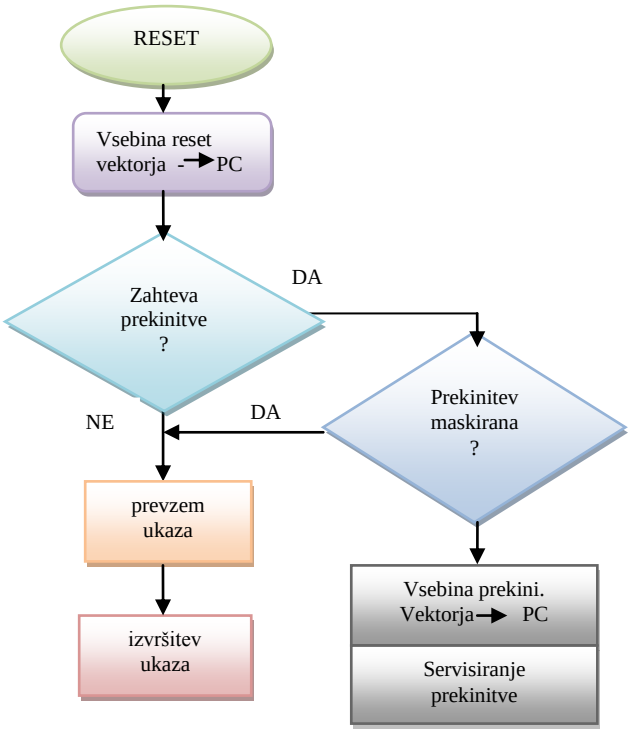
4.8 DELOVANJE MIKROPROCESORJA

Reset vektor (glavni vektor) je rezervirana lokacija v programskem pomnilniku, kamor programator vpiše naslov prvega ukaza v glavnem programu, v katerem program steče po resetu (slika 15). Zaradi tega kaže PC po resetu na glavni vektor. Reset uP pomeni, da se

njegovi registri postavijo v točno določena začetna stanja. Prekinitveni vektor določenega vira prekinitve pa je rezervirana lokacija v programskem pomnilniku, kamor programator vpiše naslov prvega ukaza prekinitvenega programa. Ker so naslovi običajno 16-bitni, lokacije programskega pomnilnika pa 8-bitne, zavzemajo vektorji dve ali več lokacij.

Mikroprocesor se pri svojem delovanju ravna po signalih ure (clock) ali takta. Njegova opravila so organizirana v tako imenovanih **strojnih ciklih**. En strojni cikel je lahko dolg eno ali več urinih period, kar je odvisno od izvedbe uP. Strojni cikli se imenujejo glede na dogajanja znotraj njih.

Tako poznamo:
- bralni cikel,
- pisalni cikel,
- prekinitveno prevzemni cikel,
- čisti strojni ali izvržbeni cikel.



Slika 30: Prikaz delovanja mikroprocesorja
Vir: Lasten

Bralni cikel
Pri tem ciklu uP pošlje na naslovne signale A0-A15 naslov lokacije, iz katere želi prebrati ukaz ali podatek. Na krmilno vodilo pošlje signal za branje (RD – read). Na podatkovnih signalih D0-D7 pa pričakuje ukaz ali podatek.

Pisalni cikel
Pri tem ciklu uP pošlje na naslovne signale A0-A15 naslov lokacije, v katero želi vpisati podatek. Na krmilno vodilo pošlje signal za pisanje (WR – write). Ta podatek pošlje na podatkovne signale D0-D7.
Program je sestavljen iz ukazov ali instrukcij, ki povedo uP, kaj mora narediti. Izvajanje vsake instrukcije poteka preko več strojnih ciklov, ki sestavljajo **instrukcijski cikel**. **Prevzem ukaza** ali fetch je eden ali več bralnih ciklov, v katerih uP bere iz lokacij programskega pomnilnika. **Izvršitev ukaza** ali execute pa je korak realizacije ukaza.

Primer: instrukcija c=a+b
- fetch
1. bralni cikel operacijske kode
- execute
1. bralni cikel operanda a
2. bralni cikel operanda b
3. strojni cikel seštevanja
4. pisalni cikel shranjevanja rezultata

4.9 POMNILNIK

Idealen pomnilnik bi moral imeti vse naslednje lastnosti:
- iz njega bi lahko izvajali neomejeno število branj (to lastnost imajo pomnilniki v obliki IC),
- vanj bi lahko opravljali neomejeno število vpisovanj (to lastnost ima samo RAM),
- ob izpadu napajalne napetosti bi moral vsebino zadržati (to lastnost imajo ROM in Flash RAM),
- vpisovanje bi moralo potekati enako hitro kot branje (samo RAM),
- kapaciteta pomnilnika in hitrost dostopa bi morala biti dovolj velika pri sorazmerno nizki ceni.

V grobem delimo pomnilnike v obliki IC na bralne in bralno-zapisovalne. Iz vseh je možno neomejeno branje, število vpisov pa se razlikuje.

V skupino, ki imajo končno število vpisov in zadržujejo vsebino tudi ob izpadu napajalne napetosti, sodijo:
- ROM (*read only memory*)
  Vpis vanj izvede proizvajalec. Uporaben je za program, konstante in tabele.
- PROM (*programable ROM*)
  Omogoča enkraten vpis vsebine, kar lahko izvede programer s pomočjo programatorja. Uporabnost je takšna kot pri ROM-u.
- EPROM (*erasable PROM*)
  S pomočjo UV svetlobe je možno njegovo vsebino izbrisati, tako da omogoča nekaj deset vpisov s pomočjo programatorja. Uporabnost je podobna kot pri PROM-u.

- EEPROM (*electric EPROM*)
  Vsebino je možno izbrisati kar v ciljnem sistemu, vendar le nekaj 100 000-krat. Zaradi tega nima enake uporabne zmožnosti kot RAM. Uporabljate ga lahko za podatke, ki se redkeje spreminjajo oz. jih spreminja uporabnik s pomočjo tipkovnice.
- FLASH RAM
  Je nižje cenovna verzija EEPROM-a in omogoča do 1000 vpisov.
  Vpisovanje v zgoraj naštete pomnilnike poteka po določeni proceduri, vsekakor pa počasneje kot pri pomnilniku, v katerega lahko neomejeno vpisujemo.
- RAM (*Random Access Memory*)
  Svojo vsebino ob izpadu napajalne napetosti izgubi. Uporaben je za shranjevanje podatkov. Če želite te podatke trajno ohraniti, morate uporabljati sistem neprekinjenega napajanja ali pa ob izpadu napajanja podatke prepišete v EEPROM. Če želite RAM uporabljati za programski pomnilnik, morate vsakič ob vklopu naložiti program.

4.10 VHODNO-IZHODNI VMESNIK

Poznamo več vrst vhodno-izhodnih vmesnikov. Nekateri od njih imajo programsko dostopne registre, s katerimi lahko uP komunicira. Razen podatkov, ki se prenašajo preko vmesnikov, pošilja uP ukaze, s katerimi določa, kako naj takšen vmesnik deluje (smer pretoka podatkov, usklajenost delovanja, možnost prekinitev ...).

Takšne vmesnike imenujemo **PIA (*Paralel Interface Adapter*).** Pomembno je, da ima vsak register vmesnika svoj naslov ali pa, da register določite s pomočjo naslovnega dekoderja. Tako lahko v programu vmesnik obravnavate kot običajno pomnilniško lokacijo, preko katere komunicirate z zunanjim svetom.

Za vmesnike pa največkrat uporabljamo registre, ki jim pravimo zadrževalniki (latch). Tem morate pošiljati kontrolne signale, ki te vmesnike odpirajo in zapirajo. Dobite jih s pomočjo naslova preko naslovnega dekoderja.

Vmesniki imajo naslednje vhodne kontrolne signale:

- CE (*chip enable*)
  Vmesnik je omogočen oz. pripravljen sprejemati podatke (vpisovanje), kadar je na tem vhodu 1log. Različica tega signala je CS (*chip select*).
- OE (*output enable*)
  Omogočeno je branje iz vmesnika oz. na izhodih vmesnika se pojavi v registru zapisana vsebina, kadar je na tem vhodu 1log. Takšne vmesnike ne morete uporabljati kot dvosmerne, kajti smer je določena z ožičenjem. Dvosmerni vmesniki imajo še dodaten kontrolni signal, s katerim lahko določate smer. Naslovni dekoder kreirate s pomočjo logične enačbe.

4.11 PROGRAMIRANJE MIKROKRMILNIKA

Vsak mikroprocesor pozna določene **ukaze**. Množico ukazov za neki mikroprocesor imenujemo **nabor ukazov.** Uporabnik uporablja nabor ukazov, tako da z njimi sestavi neko smiselno zaporedje. Temu zaporedju pravimo **program**, njegovemu sestavljanju pa **programiranje.**

Programiranje lahko opravljate na **strojnem ali zbirniškem nivoju**, kjer neposredno uporabljate nabor ukazov (zbirnik ali *assembler*). Druga možnost je programiranje v **višjem programskem jeziku**, ki je človeku bližji (Basic, Fortran, Pascal, C …).

Programer mora do podrobnosti razumeti problem, za katerega piše program. Njegovo delo lahko razčlenimo v naslednja opravila:

- **Analiza problema** je razvoj postopka, ki korak za korakom reši dani problem. Temu postopku pravimo **algoritem**.

- **Organizacija programa**. Algoritem organizirate v zaporedje operacij, pri čemer uporabljate **diagrame poteka**. Delo lahko opravljate s pomočjo osebnega računalnika, če imate orodje za risanje diagramov poteka.

- **Kodiranje**. Diagram poteka zapišete v zaporedje ukazov – program s pomočjo **urejevalnika teksta** za osebni računalnik. Shranite ga v t. i. izvorni datoteki.

- **Povezovanje in prevajanje**. Program povežete z drugimi deli programa, ki ste jih napisali že v preteklosti, ali pa s programi, ki jih dobite v knjižnici (napisal jih je proizvajalec prevajalnika in povezovalnika). Povezavo opravi **povezovalnik – linker**. Celoten program prevedete s **prevajalnikom – compilerjem** v obliko, ki jo razume uP. Povezovalnik in prevajalnik sta programski opremi za osebni računalnik.

- **Vstavljanje programa v mikroračunalnik** opravite s pomočjo **programatorja**, ki ga priključite na osebni računalnik s pripadajočo programsko opremo.

- **Preizkušanje**. Program preizkusite tako, da z izvajanjem na mikroprocesorju ugotovite ali da ustrezne rezultate. V ta namen izdelate testno kartico. Napake, ki jih pri tem odkrijete, odpravite tako, da se vrnete na kodiranje in ponovite postopek.

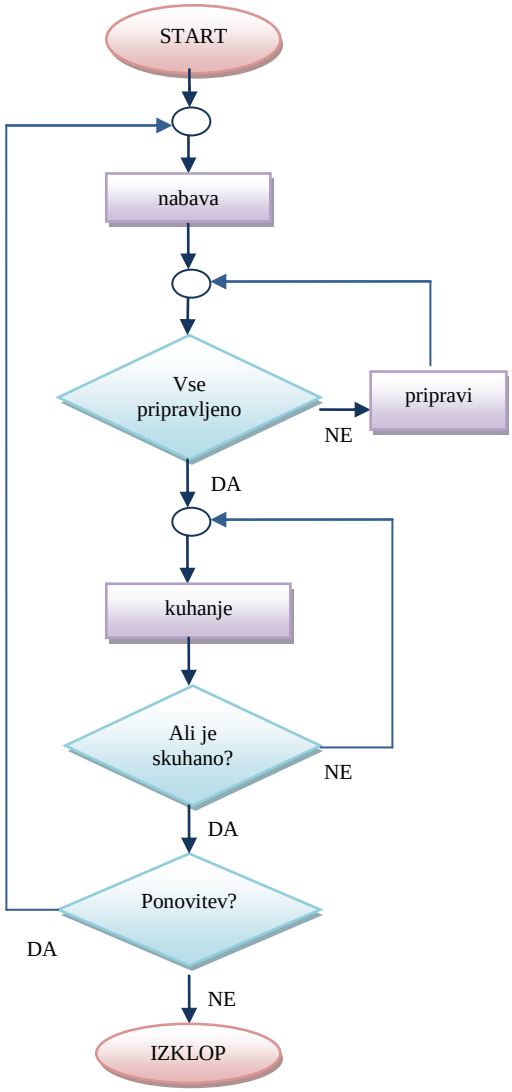
- **Dokumentiranje**. Za kasnejšo dodelavo ali spremembo programa je potrebno program dokumentirati. Dokumentacijo s potrebnimi komentarji uredite pri vseh opravilnih točkah (tehnični opis, poročilo, elaborat).

Primer:

Realizirajte prvi dve točki
za problem kuhanja kave.

1. Analiza problema:
a) nabava potrebščin (kava, sladkor in voda), opreme (štedilnik, žlica, kuhalna posoda) in energije (elektrika ali plin) in njihova priprava,
b) kuhanje,
c) testiranje (ali je že kuhano),
d) testiranje ponovitve,
e) izklop.

2. Organizacija programa:
diagram poteka



Slika 31: Primer programa
Vir: Lasten

4.12 PROGRAMIRANJE V ZBIRNEM JEZIKU

Zbirni jezik (zbirnik) ali **assembler** je jezik določenega procesorja oziroma družine mikrokontrolerjev, ki bazirajo na istem mikroprocesorju. Uporablja nabor oziroma zbirek ukazov, ki jih uP pozna.

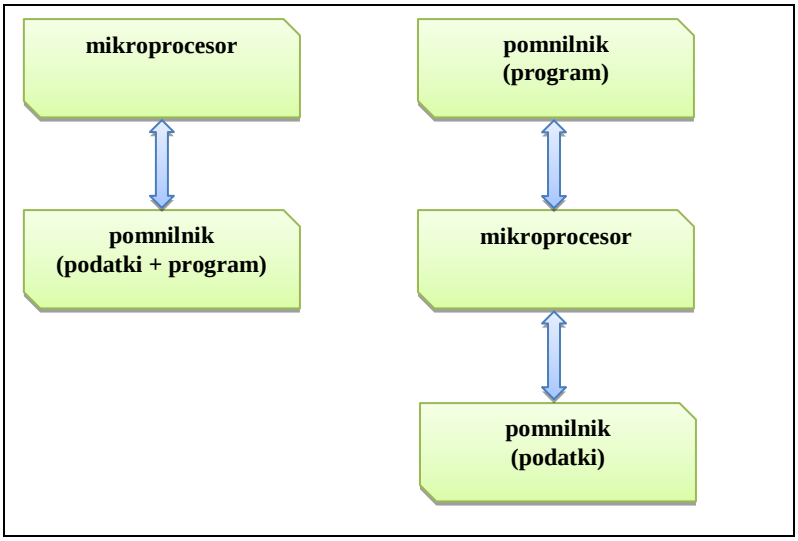
V vsakem ukazu mora biti prisotna informacija o dveh stvareh:
- operaciji,
- operandih.

Informacija o operaciji je vsebovana v t. i. **operacijski kodi**. Informacija o operandih pa je lahko podana na več načinov. V mikroračunalniku so operandi lahko shranjeni v registrih procesorja, v pomnilniku ali v registrih vhodnih naprav. V vsakem ukazu je zato potrebno na nek način povedati, kje se operandi nahajajo. Ker to običajno poveste z naslovom, na katerem se operand nahaja, pravimo temu naslavljanje. Kje se operandi nahajajo, lahko poveste na več načinov in tem pravimo **načini naslavljanja**.

Ukazi lahko v programskem pomnilniku zasedajo eno ali več lokacij. Tako poznamo eno-,dve- ali večzložne ukaze (zlog=byte=8 bitov).

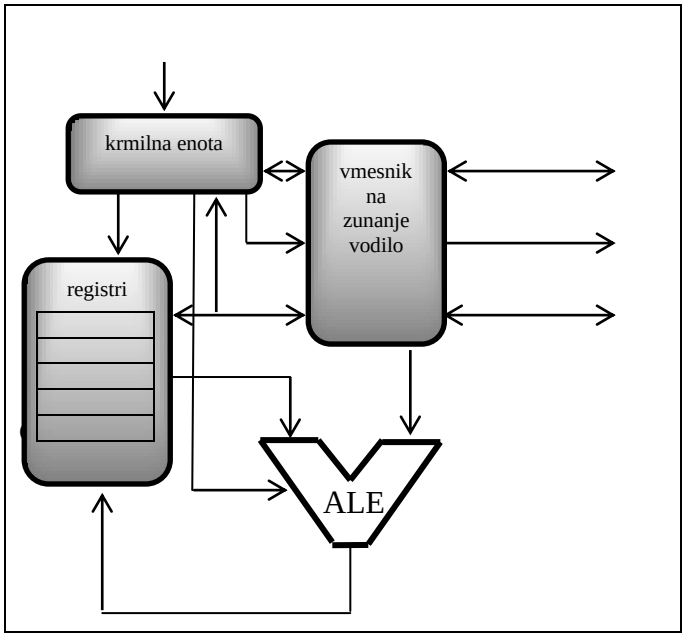
4.13 MODEL MIKROPROCESORJA

Arhitekture mikroprocesorjev se zelo razlikujejo med seboj, večina pa jih temelji na von Neumannovem modelu, kjer se podatki in programska koda nahajajo skupaj v istem pomnilniškem naslovnem področju. To je bila nesporno ustrezna in zelo koristna rešitev v časih, ko je obdelava ukazov zahtevala bistveno več časa kot njihov prenos. V zadnjem času se je ozko grlo preselilo na slednje. Von Neumannova arhitektura je postala šibka točka, ki jo predvsem v RISC mikroprocesorjih nadomešča harwardska: ta ima ločeni pomnilniški področji in podatkovne poti za podatke in ukaze. Prenos enih in drugih zato lahko poteka popolnoma vzporedno.



Slika 32: Model mikroprocesorjev
Vir: Lasten

Arhitekturo mikroprocesorja lahko v grobem razdelimo na tri glavne sestavne dele: **krmilno enoto, aritmetično-logično enoto in nabor registrov.**



Slika 33: Zgradba mikroprocesorja
Vir: Lasten

Najpomembnejši del mikroprocesorja predstavlja krmilna enota, ki usklajuje delovanje posameznih delov mikroprocesorja po navodilih programske kode. Krmilna enota bere ukaz za ukazom iz programa, zapisanega v strojni obliki, jih dekodira in preko množice krmilnih signalov krmili njihovo izvajanje. Za izvajanje operacij nad podatki, ki jih obdeluje program, skrbi aritmetično-logična enota (ALE). Ta je sposobna opraviti različne matematične in logične operacije nad podatki različnih tipov.

Registri služijo za začasno hranjenje in obdelavo podatkov in njihovih naslovov. Izvedeni so kot zelo hitre pomnilne celice (običajno 5- do 10-krat hitrejše od zunanjega – delovnega). Obstaja več vrst registrov, od katerih so nekateri dostopni programerju, drugi pa so namenjeni internemu delovanju mikroprocesorja. Nekateri registri nastopajo tudi kot del krmilne oziroma aritmetične in logične enote. Posamezne enote mikroprocesorja so med seboj povezane preko več internih vodil. Širina teh vodil običajno po širini ustreza velikosti registrov. Prav tako je mikroprocesor z okolico povezan preko zunanjih vodil, ki jih pri večini mikroprocesorjev sestavljajo:

- ***naslovno vodilo*** – ta določa naslov pomnilniške lokacije, do katere želi mikroprocesor dostopati;
- ***podatkovno vodilo*** – preko njega poteka izmenjava vsebine (programskih ukazov in podatkov) med registri in celicami pomnilnika, torej v mikroprocesor ali iz njega;
- ***krmilne linije*** –krmilijo komunikacijo z okolico (povejo npr. ali želi mikroprocesor brati ali pisati v pomnilniško področje).

Zaradi tehnoloških omejitev so bile širine zunanjih vodil pri starejših mikroprocesorjih pogosto manjše od širine internih vodil. To je pomenilo, da je bilo potrebno za prenos ene besede izvesti dva ali več bralnih oz. pisalnih ciklov. V sodobnih mikroprocesorskih arhitekturah težimo k čim večji prepustnosti podatkov in nimajo takšnih omejitev.
Zunanje in notranje vodilo sta povezana preko posebnega vmesnika, katerega delovanje je pod nadzorom krmilne enote. Nekateri preprostejši mikroprocesorji in mikroračunalniki imajo

programski in podatkovni pomnilnik vgrajen na isti silicijevi rezini. Takšni mikroprocesorji seveda zunanjega naslovnega in podatkovnega vodila ne potrebujejo.

Povzetek:

Mikroprocesor je osnovni element, okoli katerega je zgrajen mikroračunalnik. Narejen je kot integrirano vezje z visoko stopnjo integracije. Je osrednji del računalnika, ki obdeluje podatke, nadzoruje in upravlja ostale enote. Je enota, v kateri se odvijajo funkcije seštevanja, odštevanja, množenja in deljenja. Krmilna enota vodi delovanje mikroprocesorja, tako da posreduje v skladu s sprejeto instrukcijo (ukaz programa) krmilne signale ostalim enotam.

Število registrov in njihove oznake so pri raznih izvedbah mikroprocesorja različne. V grobem ločimo registre, ki so programsko dostopni in programsko nedostopni. Programsko dostopne registre lahko v programu naslavljamo in tudi spreminjamo njihove vsebine. Ena od zelo pomembnih lastnosti vsakega mikroprocesorja je njegova sposobnost, da na zunanjo ali notranjo zahtevo prekine izvajanje trenutno izvajajočega programa in prične izvajati prekinitveni servisni program. Ob zaključku slednjega se vrne na prekinjeno mesto in nadaljuje z izvajanjem prekinjenega programa. Zahteva za prekinitev se servisira le, če je to dovoljeno (prekinitev omogočena). Če ni, se zahteva ignorira in govorimo o maskiranju prekinitve. Ali neko prekinitev dovolimo ali ne, je odvisno, kaj v programu vpišemo v register za dovolitev prekinitev IE (*Interrupt Enable*), ki je sestavljen iz posameznih omogočitvenih bitov za posamezni prekinitveni vir.

Pri večini mikroprocesorjev se nekatere operacije vedno nanašajo na sklad. To so predvsem prekinitve in klici podprogramov. Sklad je pomemben tudi kot orodje za realizacijo nekaterih programskih rešitev (rekurzija). Reset vektor (glavni vektor) je rezervirana lokacija v programskem pomnilniku, kamor programator vpiše naslov prvega ukaza v glavnem programu, v katerem program steče po resetu. Zaradi tega kaže programski števec po resetu na glavni vektor. Reset mikroprocesorja pomeni, da se njegovi registri postavijo v točno določena začetna stanja. Prekinitveni vektor določenega vira prekinitve pa je rezervirana lokacija v programskem pomnilniku, kamor programator vpiše naslov prvega ukaza prekinitvenega programa. Ker so naslovi običajno 16-bitni, lokacije programskega pomnilnika pa 8-bitne, zavzemajo vektorji dve ali več lokacij.

Mikroprocesor se pri svojem delovanju ravna po signalih ure (*clock*) ali takta. Njegova opravila so organizirana v tako imenovanih strojnih ciklih. En strojni cikel je lahko dolg eno ali več urinih period, kar je odvisno od izvedbe mikroprocesorja. Strojni cikli se imenujejo glede na dogajanja znotraj njih. Poznamo več vrst vhodno-izhodnih vmesnikov. Nekateri od njih imajo programsko dostopne registre, s katerimi lahko uP komunicira. Razen podatkov, ki se prenašajo preko vmesnikov, pošilja mikroprocesor ukaze, s katerimi določa, kako naj takšen vmesnik deluje (smer pretoka podatkov, usklajenost delovanja, možnost prekinitev.
Vsak mikroprocesor pozna določene ukaze. Množico ukazov za neki mikroprocesor imenujemo nabor ukazov. Uporabnik uporablja nabor ukazov tako, da z njimi sestavi neko smiselno zaporedje. Temu zaporedju pravimo program, njegovemu sestavljanju pa programiranje. Programiranje lahko opravljamo na strojnem ali zbirniškem nivoju, kjer neposredno uporabljamo nabor ukazov (zbirnik ali assembler). Druga možnost je programiranje v višjem programskem jeziku, ki je človeku bližji.

Vprašanja:

1. Opišite in skicirajte aritmetično-logično enoto.
2. Na kateri fazi lahko razdelimo izvajanje ene instrukcije ali ukaza krmilne enote?
3. Naštejte registre in ter pojasnite delovanje kazalca sklada – SP.
4. Razložite na skici osnovno zahtevo pri prekinitvah.
5. Pojasnite servisiranje prekinitve.
6. Kako definiramo sklad in kaj je značilno zanj?
7. Na shemi pojasnite delovanje programskega števca.
8. Pomen Reset Up-a.
9. Kaj je strojni cikel pri delovanju mikroprocesorja, katere poznamo (opišite enega izmed njih)?
10. Kaj je Paralel interface adapter?
11. Pojasnite delovanje programskega števca.

5 PROGRAMIRLJIVI LOGIČNI KRMILNIK (PLK)

V zadnjem desetletju se je področje krmilne tehnike precej spremenilo. Prej smo uporabljali izključno trdo ožičena krmilja, v sodobnem času pa se vse več uporabljajo tako imenovana programska krmilja (angleška kratica za programirljivi logični krmilnik je PLC – *Programmable Logic Controller*).

Največja prednost, ki jo imajo krmilniki v primerjavi z vsemi starejšimi sistemi krmiljenja, je, da imajo možnost programiranega delovanja. S programiranjem se lahko določa delovanje krmilnika, s tem pa je omogočena velika fleksibilnost sistema, saj so možnosti programiranja skoraj neomejene. Klasična krmilja so vsebovala fiksno zgrajeno logiko – bilo jo je zelo zapleteno, predvsem pa drago spreminjati oziroma popravljati.

V poglavju Programirljivi logični krmilnik (PLK) je prikazano, kako je zgrajen PLK in kako deluje. Predstavljeno je tudi krmiljenje naprave (v odprti in zaprti zanki). Predstavljena so nam sekvenčna krmilja. Podrobneje sta predstavljena Siemensov krmilnik in strojna oprema. Nekaj besed pa je namenjenih tudi programski opremi in programiranju.

PLK je digitalno delujoča elektronska naprava, ki na podlagi ukazov, shranjenih v programirljivem pomnilniku, izvaja logične, sekvenčne, časovne in aritmetične operacije ter s tem vodi različne naprave in procese preko digitalnih in analognih vhodov in izhodov.

5.1 ZGRADBA

Vhodna enota je enota za zajemanje procesnih spremenljivk in spremenljivk operaterja, ki njihove signale preoblikuje in prilagodi ter posreduje enoti za obdelavo. Pomembna pri tem je galvanska ločitev signalov.

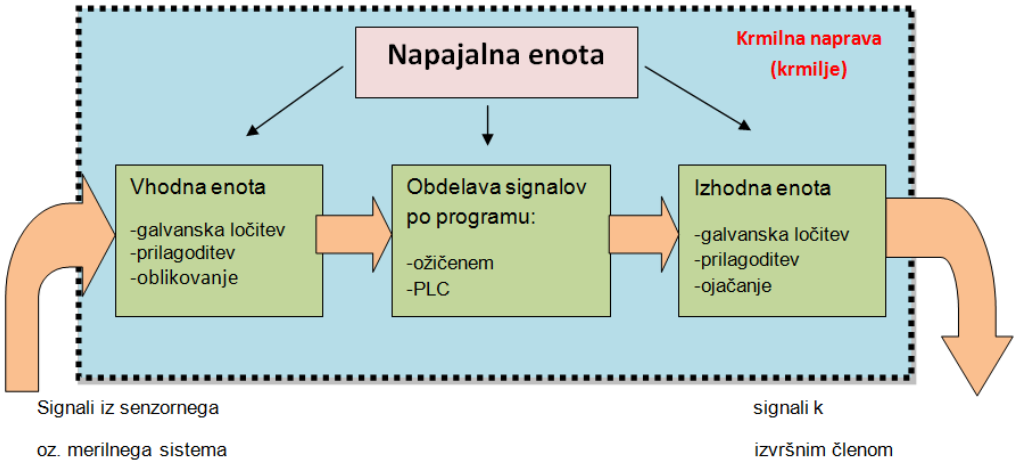
Enota za obdelavo signalov

a) v trajno ožičenih sistemih je to pogosto vezje z elektronskimi logičnimi, časovnimi, številnimi, računskimi … elementi, ki odpravljajo funkcijo krmilja;

b) v krmiljih s prostim programiranjem je to centralno procesna enota (mikroprocesor), skupaj s pomnilniki in perifernimi enotami; v programskem pomnilniku je shranjen krmilni program, ki odpravlja funkcijo krmilja.

Izhodna enota je enota za preoblikovanje, prilagoditev in posredovanje krmilnih signalov izvršnim členom procesa. Ponovno je pri tem pomembna galvanska ločitev signalov.

Napajalna enota je enota, ki oskrbuje krmilno napravo z napajalno napetostjo. Pri PPK so standardne izvedbe z enosmerno napetostjo 24 V.



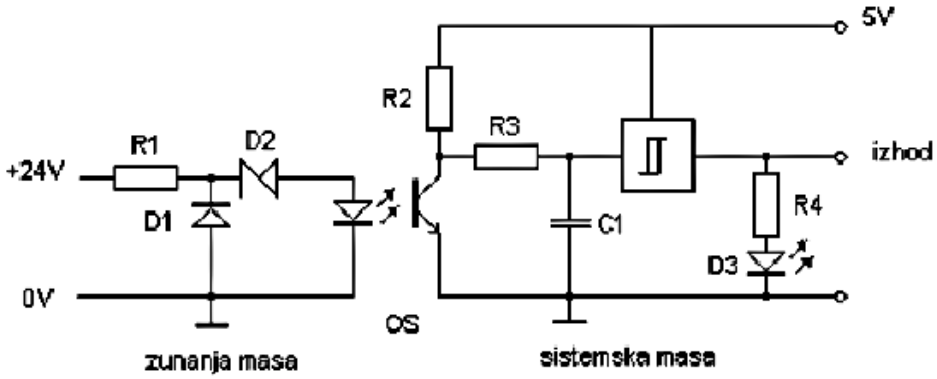
Slika 34: Zgradba krmilnega sistema
Vir: Lasten

Vhodni moduli

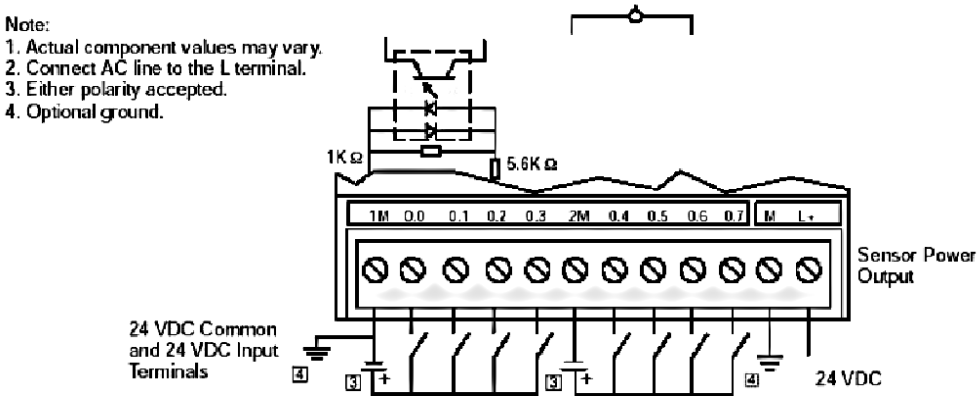
Da se motnje ne prenašajo, se uporablja obtoločilnik oz. oktokopter, ki odstrani motnje.

Poleg že omenjenih nalog (prilagajanje nivojev, galvanska ločitev, zaščita pred kratkim stikom) so še dodatne možnosti:

- možnost nastavljanja časa vhodnega filtra (izločanje hitrih preklopov na vhodu zaradi motenj),
- različno število vhodov (možnost razširitve),
- indikacija logične enice.



Slika 35: Električna shema vhodnega digitalnega modula za enosmerne napetosti
Vir: http://support.automation.siemens.com

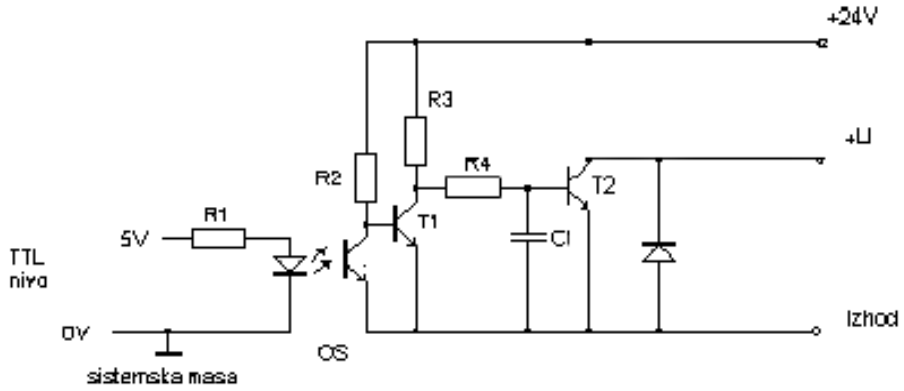


Slika 36: Priključitev vhodov na vhodni digitalni modul pri S7-300
Vir: http://support.automation.siemens.com

Izhodni moduli

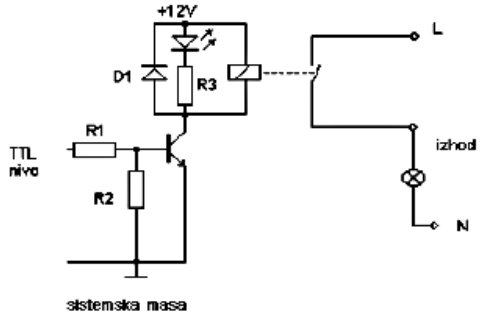
- Namenjeni so za prenos in oblikovanje signala, ki ga posreduje centralna enota krmilnika. Signali iz centralno procesne enote so navadno TTL-nivoja (5 V napajanje).
- Energijsko višji nivoji signalov dosežete s tranzistorskim ojačevalnikom, pomožnimi releji ali s traki.
- Tokovne obremenitve so v območju od nekaj mA do nekaj A.
- Galvanska ločitev sistemskega dela od zunanjega, procesnega.
- Indikacija ločične enice.

Porabnik je lahko priključen le na enosmerno napetost (manjše tokovne obremenitve).

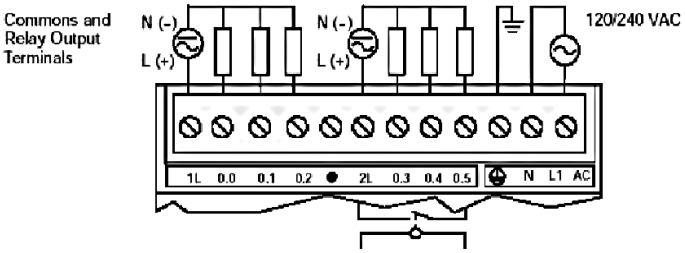


Slika 37: Digitalni izhodni modul s tranzistorskim izhodom
Vir: http://support.automation.siemens.com

Porabnik je lahko priključen na enosmerno ali izmenično napetost.



Slika 38: Digitalni izhodni modul z relejskim izhodom
Vir: http://support.automation.siemens.com



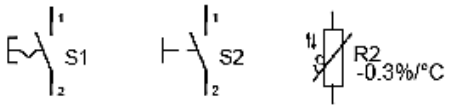
Slika 39: Priključitev izhodnih enot na digitalni relejski izhodni modul
Vir: http://support.automation.siemens.com

Dajalniki signalov

Naloga dajalnikov signala (senzorji, detektorji, merilni členi) je merjenje oz. detekcija veličine v krmilnem sistemu, pretvoriti signal v ustrezno obliko (ojačanje, temp. kompenzacija, offset …) in informacijo posredovati krmilni napravi.

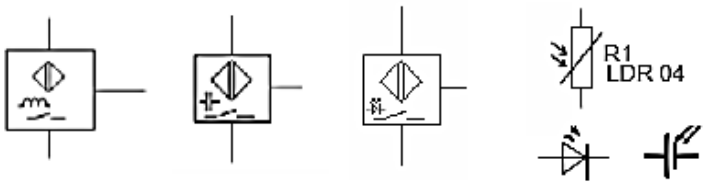
Poznamo različne vrste dajalnikov:

- **mehanski** pretvorijo silo, ki deluje na dajalnik v električni signal (mejna stikala, tipkala, lahko pa silo pretvorijo tudi v tok ali napetost),
- **uporovni** uporabljajo lastnost, da se spremeni upornost elementa. Uporabljajo se mostična vezja (uporovni lističi, termistorji, linearni in rotacijski potenciometri ...),



Slika 40: Mehanski in uporovni dajalniki položajev
Vir: Lasten

- **induktivni –** merjena fizikalna veličina vpliva na spremembo induktivnosti, izhod je lahko binaren z delovnim ali mirovnim kontaktom,
- **kapacitivni** delujejo na spremembi kapacitivnosti, merite lahko pomike, nivoje, uporaba kot približevalna končna stikala,
- **optoelektronski** delujejo na spremembi svetlobe, ki jo pretvorijo v el. napetost (npr. fotocelica), ali pa delujejo na spremembi ohmske upornosti (npr. fotoupor, fotodioda, pri sondah se uporablja UV ali IR svetloba),
- **CMOS-senzorji** (senzor in elektronika v enem čipu – senzorni efekt v Si, nižja cena, višja kvaliteta, manjše motnje).

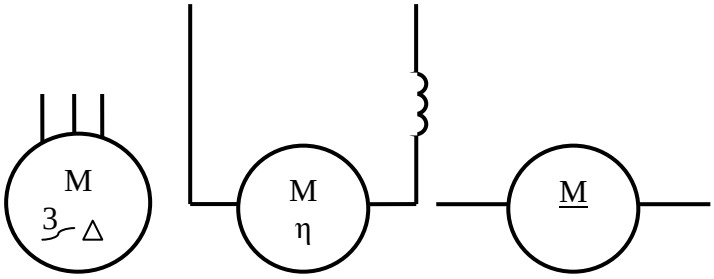


Slika 41: Induktivni, kapacitivni, optoelektronski in CMOS dajalniki signala
Vir: Lasten

Izvršni členi – aktuatorji

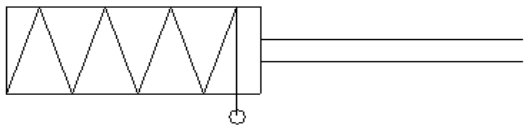
Naloga izvršnih členov je izvrševanje ukazov, javljanje in signalizacija o stanju procesa.

- Elektromotorni pogoni



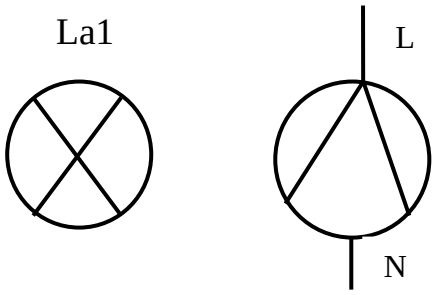
Slika 42: Simboli za nekatere elektromotorje
Vir: Lasten

- Pnevmatski cilindri



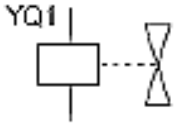
Slika 43: Pnevmatski cilinder
Vir: Lasten

- Svetila, grelci, ventilatorji



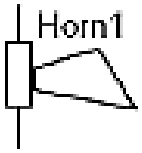
Slika 44: Svetila, grelci, ventilatorji
Vir: Lasten

- Ventili



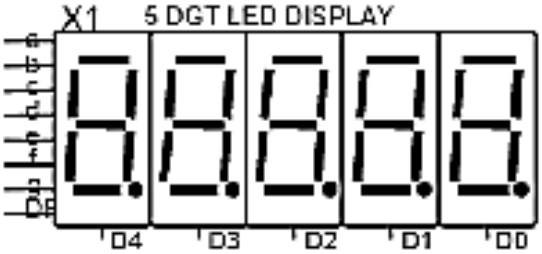
Slika 45: Ventili
Vir: Lasten

- Javljalniki



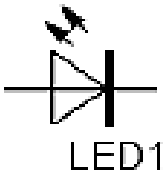
Slika 46: Javljalnik
Vir: Lasten

- Prikazovalniki



Slika 47: Prikazovalnik
Vir: Lasten

- Svetlobna signalizacija

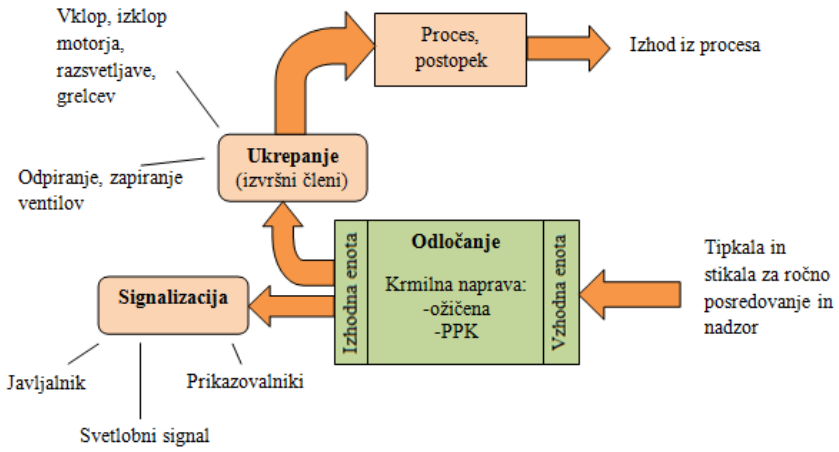


Slika 48: Svetlobna signalizacija
Vir: Lasten

Krmiljenje naprave

a) **<u>Krmiljenje v odprti zanki</u>** (kombinacija vhodnih spremenljivk s pomočjo krmilne naprave aktivira krmilne signale in jih preko izhodne enote posreduje izvršnim členov – iz procesa ni povratne informacije).

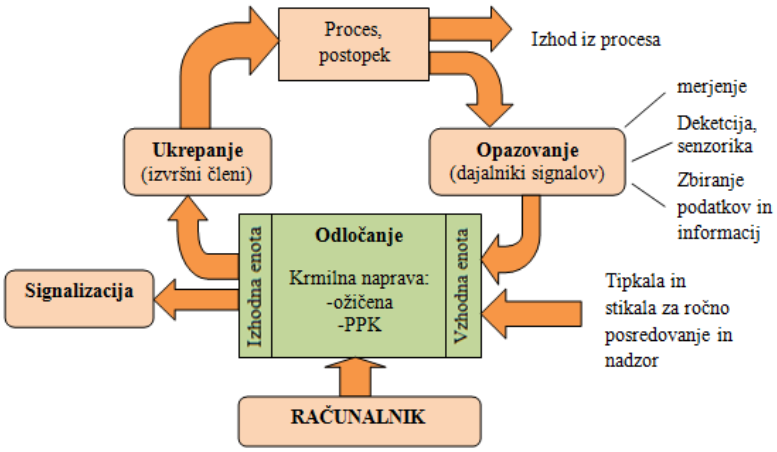
Delovanje oz. vodenje procesa poteka v dveh fazah:
- **Odločanje** (obdelava podatkov, signalov).
- **Ukrepanje** (posredovanje in izvrševanje ukazov, katere običajno spremlja ustrezna signalizacija).



Slika 49: Krmiljenje v odprti zanki
Vir: Lasten

b) **<u>Krmiljenje v zaprti zanki</u>** (najpogostejši način, predvsem pri krmiljenju industrijskih procesov; na vhod krmilne naprave prihajajo signali iz procesa).

Delovanje oz. vodenje procesa vsebuje dodatno fazo:
- **Opazovanje** (zbiranje informacij in podatkov)

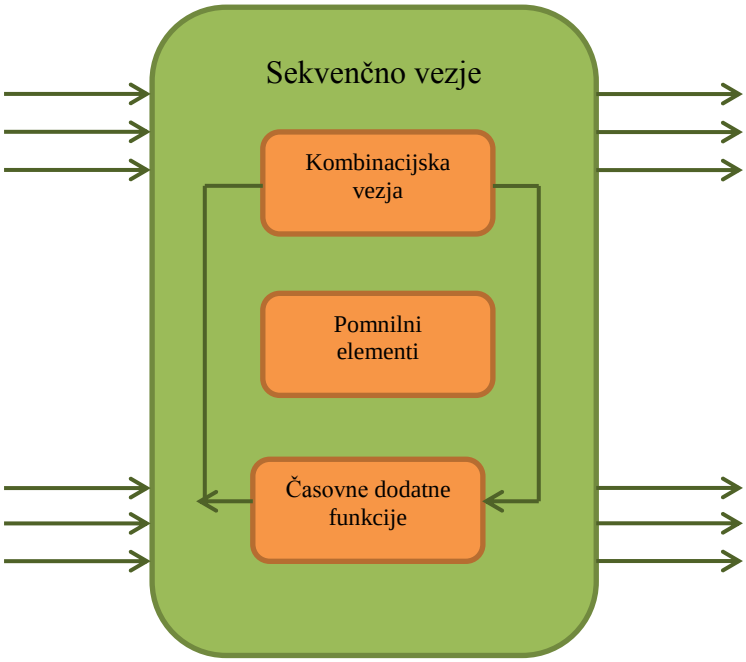


Slika 50: Krmiljenje v zaprti zanki
Vir: Lasten

5.2 SEKVENČNA KRMILJA

5.2.1 Lastnosti sekvenčnih krmilij

- Izhod ni odvisen le od trenutnega stanja vhodov, ampak tudi od predhodnih stanj sistema.
- Poleg osnovnih logičnih funkcij (kombinacijsko vezje) sekvenčna krmilja sestavljajo še **pomnilni elementi, časovne, števne in še druge dodatne funkcije.**
- Ista kombinacija vhodnih stanj se lahko preslika v različne izhodne kombinacije.
- Lahko jih primerjate z asinhronimi sekvenčnimi vezji in opisujete ter načrtujete s pomočjo diagrama stanja oz. tabele stanj ali s koračnimi verigami.



Slika 51: Sekvenčno vezje
Vir: Lasten

5.2.2 Prednosti sekvenčnih krmilij

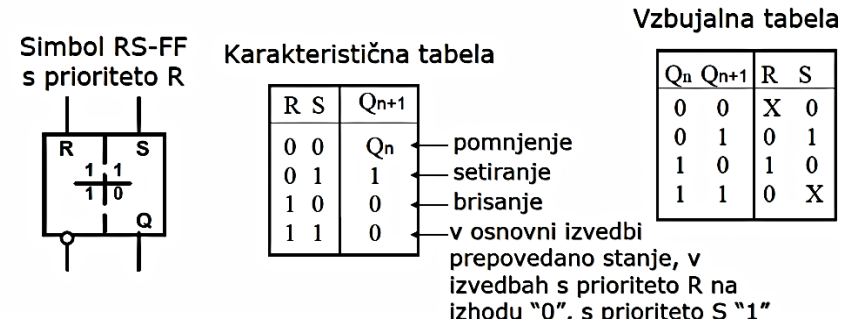
a) Je enostavnejši → manjši in enostavnejši program krmilnika.
b) Manjša občutljivost na šumne signale in motnje.
c) Sistem stabilnih stanj; prehod v novo stanje se izvede le ob določenih pogojih; na ostale spremembe sistem ne odgovarja.
d) Enostavnejše odkrivanje napak.

Sekvenčna krmilja delimo na:

- **Prosto delujoča krmilja** – na vhodu se lahko pojavi poljubna kombinacija v poljubnem zaporedju.
- **Koračno delujoča krmilja** – kombinacije vhodov se vedno pojavljajo v določenem zaporedju.

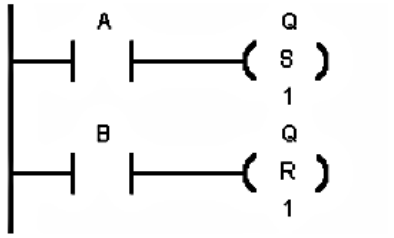
5.2.3 Gradniki sekvenčnih krmilij

Pomnilne (spominske) funkcije:

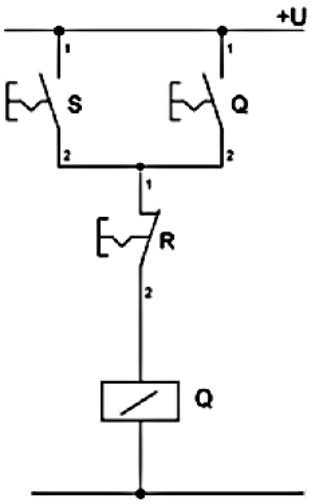


Slika 52: RS-funkcije
Vir: Simatic Step7 Help

Pomnilne (spominske) funkcije v lestvičnem diagramu:



Slika 53: RS-funkcija (lestvični diagram)
Vir: Simatic Step7 Help



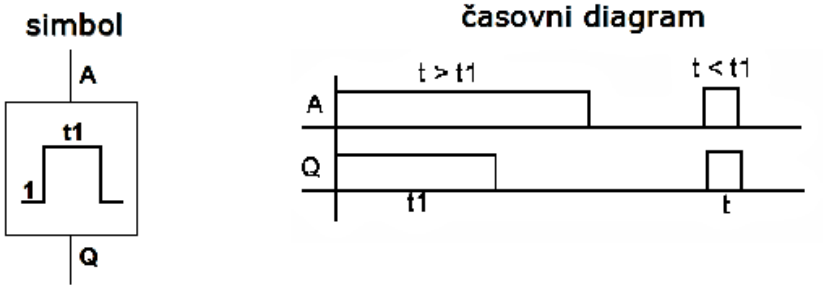
Slika 54: RS- funkcija
Vir: Simatic Step7 Help

V digitalni tehniki obstajajo še tri vrste FF: JK-, D- in T-FF (števci in registri). Kot samostojni FF v krmiljih ne nastopajo pogosto. Če jih potrebujemo, jih enostavno realiziramo (programiramo) na osnovi osnovnega RS-FF.

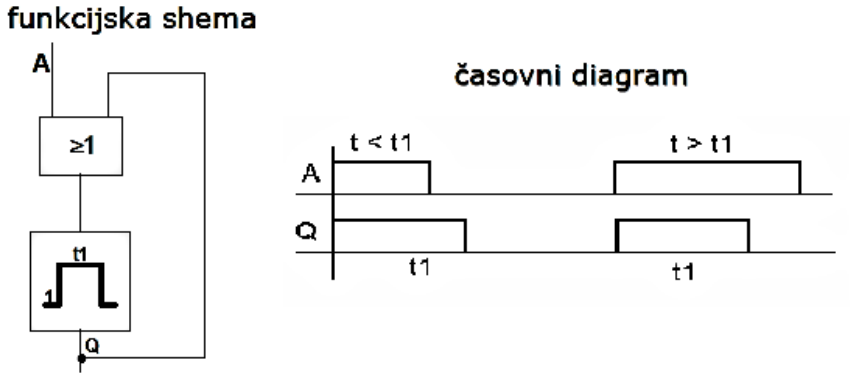
Časovne funkcije

Časovne funkcije se uporabljajo za določanje trajanja logičnih signalov – lahko skrajšujete, podaljšujete ali jih časovno premikate.

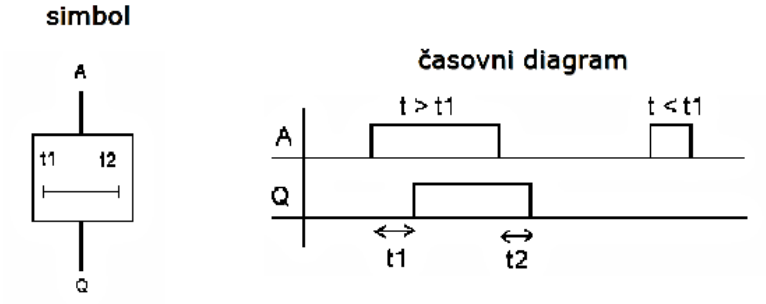
Poznamo tri osnovne vrste časovnih funkcij:
- skrajševanje dolgih impulzov,
- podaljševanje kratkih impulzov,
- časovnih premaknitev impulzov.



Slika 55: Skrajševanje impulzov
Vir: Simatic Step7 Help



Slika 56: Podaljševanje impulzov
Vir: Simatic Step7 Help

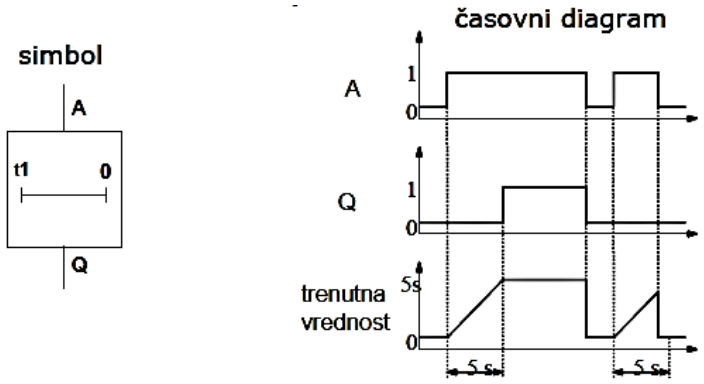


Slika 57: Premaknitev impulzov
Vir: Simatic Step7 Help

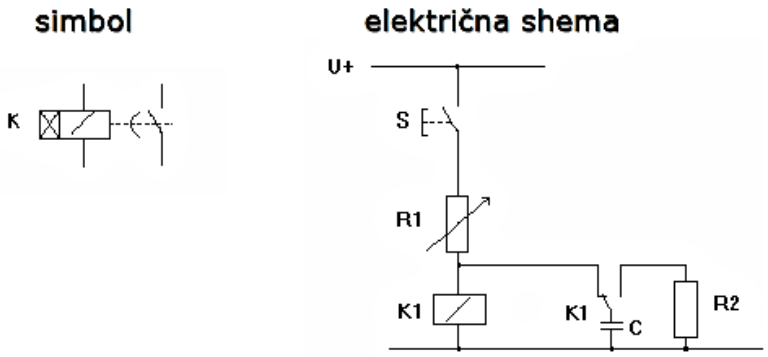
Posebne časovne funkcije:

V krmilni tehniki pogosto uporabljamo še dva načina posebnih časovnih funkcij. To sta zakasnitvi, ki ju realiziramo s pomočjo časovnih funkcij in logičnih vrat:

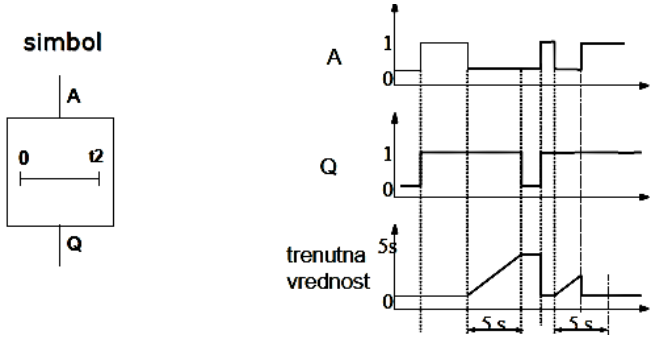
a) Zakasnitev vklopa.
b) Zakasnitev izklopa.



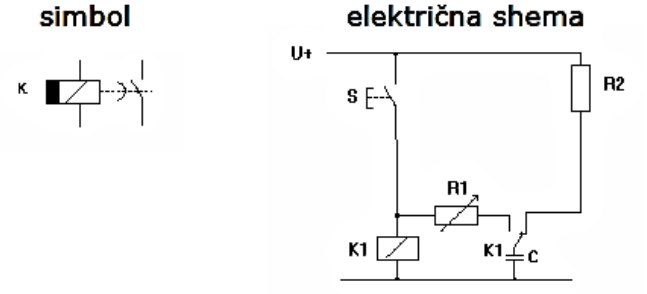
Slika 58: Pojasnitev zakasnitve vklopa
Vir: Simatic Step7 Help



Slika 59: Zakasnitev vklopa (rele)
Vir: Simatic Step7 Help



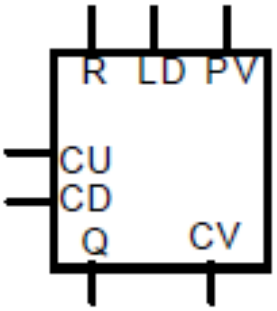
Slika 60: Pojasnitev zakasnitve izklopa
Vir: Simatic Step7 Help



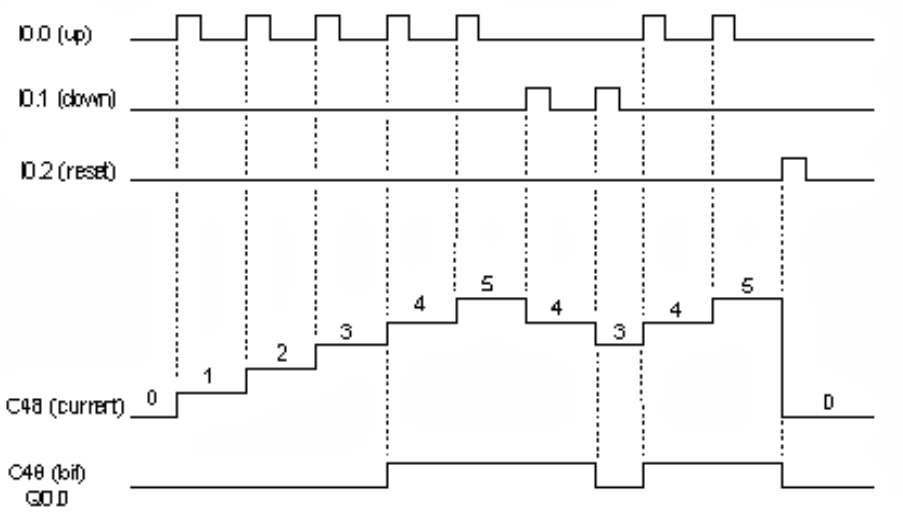
Slika 61: Izvedba zakasnitve izklopa (rele)
Vir: Simatic Step7 Help

Števci (štetje dogodkov)

CU: štetje gor (dinamični)
CD: štetje dol (dinamični)
R: reset
PV: nastavljena vrednost
LD: naloži vrednost PV (dinamični)
Q: izhod (bool)
CV: vrednost števca (celoštevilska vrednost)



Slika 62: Števec (Counter)
Vir: Simatic Step7 Help



Slika 63: Časovni diagram delovanja števca GOR/DOL
Vir: Simatic Step7 Help

5.3 ZNAČILNOSTI KRMILNIKA SIEMENS

Prosto programirljivi krmilniki so že od svojega začetka bistveni del avtomatizacije in vodenja sistemov. Njihova uporaba se je tako razširila, da je že skoraj popolnoma izrinila uporabo tako imenovane trdo ožičene logike in analognih krmilnikov. Vzrok za njihovo uveljavitev velja iskati predvsem v zmožnosti hitrega programiranja brez spremembe ožičenja.

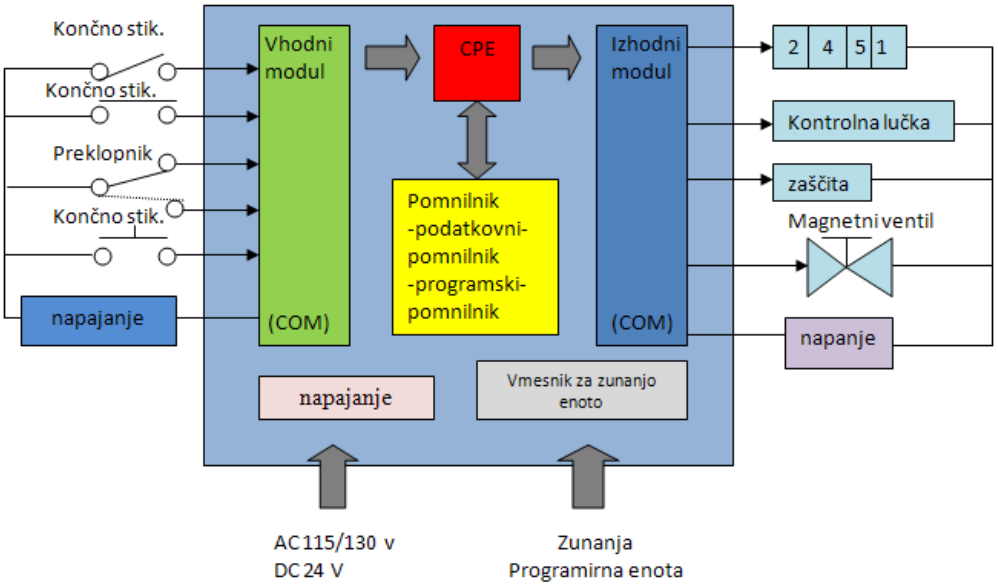
Zaradi množične proizvodnje in uporabe prosto programirljivih krmilnikov v industriji so ti postali:

- relativno poceni,
- zelo zanesljivi v delovanju,
- enostavni za uporabo (programiranje, montaža in servisiranje).

Uporaba prosto programirljivih krmilnikov je danes zelo razširjena. Glavna področja uporabe so:

- zajemanje in obdelava analognih ter digitalnih podatkov,
- reguliranje različnih sistemov,
- opravljanje računalniških operacij,
- kot vmesnik za izvajanje daljinskih komand,
- za komunikacijo.

Prosto programirljivi krmilniki predstavljajo računalnik brez zaslona in tipkovnice, vendar pa ima vse ostale komponente, kot so procesor, pomnilnik, vhodni/izhodni vmesniki ter komunikacijski vmesniki za povezavo z drugimi sistemi, ki delujejo na osnovi uporabniškega programa. Prilagojen je za delovanje v industrijskem okolju. Z razvojem krmilnikov so se razvila različna programska orodja, s pomočjo katerih pišemo uporabniške programe za različne vrste krmilnikov. S pomočjo uporabniških programov krmilniku dodelimo naloge, ki ji mora opraviti. Programe pišemo največkrat z računalniki. Pri tem moramo paziti, da uporabimo pravilen komunikacijski vmesnik za delovanje krmilnika.



Slika 64: Blokovna zgradba krmilnika
Vir: Lasten

5.4 STROJNA OPREMA SIMATIC SIEMENS

Krmilne sisteme SIMATIC delimo v tri podskupine, in sicer:

- programirljivi krmilniki SIMATIC S7,
- SIMATIC M7 sistema in
- kompletni krmilni sistemi SIMATIC C7.

Programirljivi krmilniki SIMATIC S7 se prav tako delijo glede na njihovo velikost oz. zmogljivost:

- spodnji razred krmilnikov, serija S7-200;
- srednji razred krmilnikov, serija S7-300;
- zgornji razred krmilnikov, serija S7-400.

Sistemi SIMATIC M7 omogočajo vključitev matematičnih operacij in obdelavo podatkovnih baz v krmilni program STEP7.

Sistemi SIMATIC C7 vključujejo programirljivi krmilnik družine S7-300 in nadzorno ploščo (OP) v eni napravi.

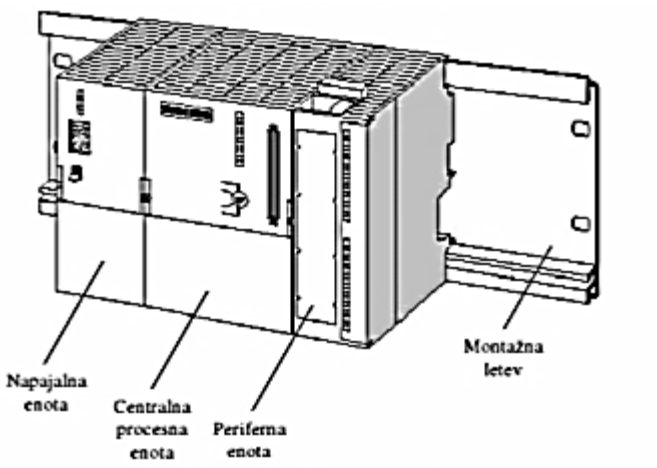
Preko naprav za posluževanje in opazovanje poteka komunikacija med operaterjem in napravo. SIMATIC nudi celo vrsto teh naprav, ki v glavnem podpirajo sisteme SIMATIC in tudi nekatere druge sisteme drugih proizvajalcev.

Vse naštete sisteme lahko programirate s posebnim programiranimi napravami PG, ki pa so zelo drage, zato se raje poslužujte osebnih računalnikov oz. prenosnikov, na katerih je naložen programski paket STEP 7.

Da sistemi delujejo pravilno, morajo tudi komunicirati med seboj. Poznamo različne sisteme za komunikacijo, ki se razlikujejo med seboj po hitrosti, zmogljivosti prenosa in seveda ceni.

Sistemi, ki jih najpogosteje zasledimo v industriji, so:

- PROFIBUS,
- PROFINET,
- SAFETYBUS in
- Industrijski thernet



Slika 65: Prikaz posameznih modulov krmilnega sistema Siemens
Vir: http://support.automation.siemens.com

5.5 SESTAVNI DELI PROSTO PROGRAMIRLJIVEGA SISTEMA

Napajalna enota (PS 307 5A)

Napajalna enota nam omogoča spremenitev napetosti iz 230V v 24V. To nam omogoča transformator, ki je v njej. Do enote pripeljemo fazni vodnik, nevtralni vodnik ter ozemljitev, po njih se pretaka napetost 230V. Iz enote pa dobimo napetost 24V, na katero lahko priključimo vse ostale komponente.

Slika 66: Napajalna enota
Vir: Lasten

Centralno procesna enota (CPU 313c)

V našem primeru smo uporabili CPU 313c, ki skrbi za procesiranje, obdelovanje, dograjevanje podatkov. To komponento bi lahko označili za najbolj pomembno in zahtevno v celotnem modularnem krmilniku. Seveda se na centralni procesni enoti nahajajo tudi vhodi in izhodi (DI 16/DO 16* DC 12V), na katerih se nahajajo signali senzorjev, motorja, releja in robota.

Slika 67: Centralno-procesna enota
Vir: Lasten

Komponente za postavitev mreže (CP 343-1 Advanced, Scalance X208)

Za vzpostavitev omrežja najprej potrebujete t. i. omrežni modul, na katerem je protokol Simatic Net, ki poteka na modulu CP 343-1 Advanced. Ta modul poskrbi za vzpostavitev omrežja, a pojavi se težava, in sicer, da ima ta modul le en vmesnik RJ 45, zato potrebujete razdelilnik, t. i. Scalance X208.

Slika 68: Komunikacijski procesor CP 343-1 Advanced
Vir: Lasten

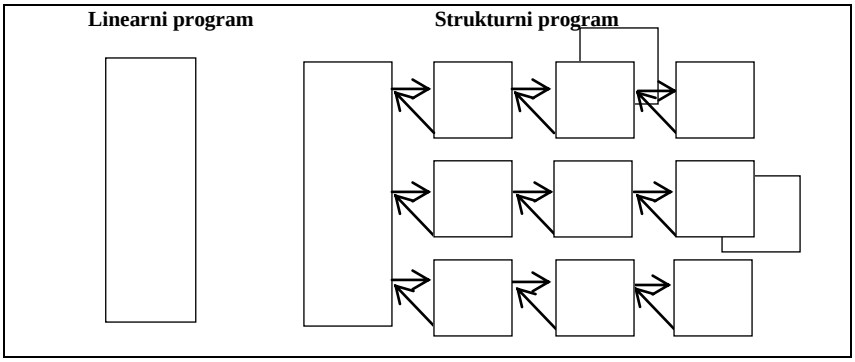
Slika 69: Switch Scalance X208
Vir: Lasten

5.6 PROGRAMSKA OPREMA SIEMENS

Za programsko podporo uporabnikom izdelkov skupine Simatic so v Siemensu razvili programski paket STEP 7. Programski paket je kompatibilen na operacijskih sistemih Microsoft Windows in ustreza standardu EN 61131-3 [7].

Razdelimo ga v štiri skupine izdelkov programske opreme:
- osnovna programska orodja (*Standard Tools*),
- inženirska programska orodja (*Engineering Tools*),
- programska orodja za delo v realnem času (*Runtime Software*) in
- programska orodja za povezavo človek – stroj (*Human Machine Interface*).

Struktura programa STEP 7:



Slika 70: Struktura programa STEP 7
Vir: Lasten

Ves uporabniški program je sklenjen v enem bloku (OB1). Procesor izvaja program linearno po stavkih, kot so vpisani. Takšna organizacija uporabniškega programa se najde pri sistemu za avtomatizacijo S7-200. Dodatno se lahko tudi pri njem uporablja tehnika podprogramov.

Strukturni program
Celoten uporabniški program je razdeljen na bloke. Blok je samodejen del programa, ki se razlikuje od preostalih po svoji funkciji, nalogi in prioriteti.

Ločimo sedem tipov različnih blokov:

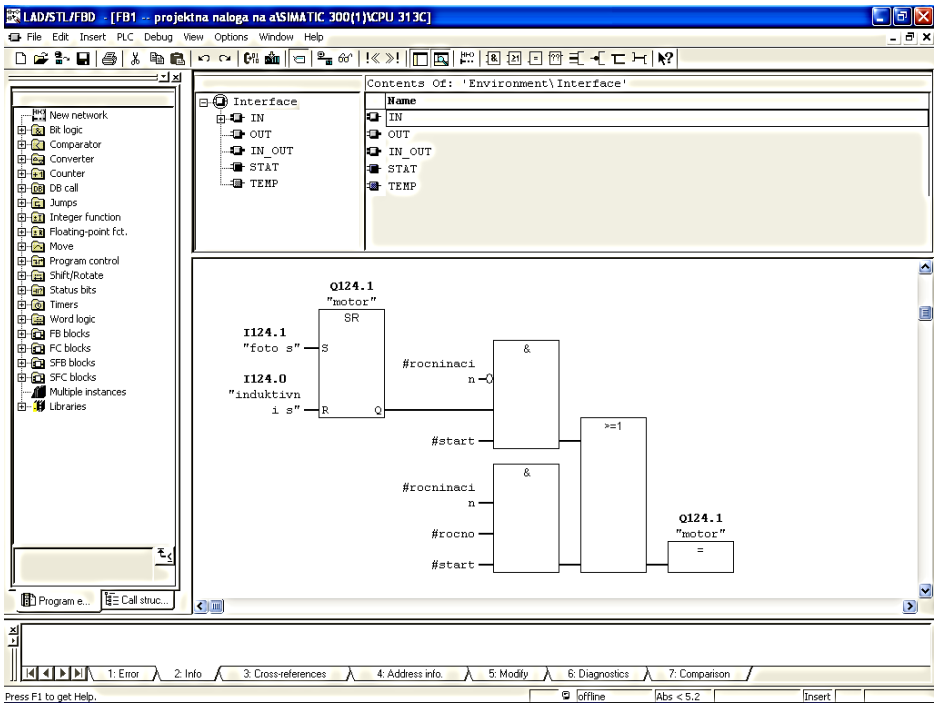
Organizacijski bloki (OB) regulirajo ciklično, časovno in alarmno obdelavo programov.
Lastnosti:
- uporabnikov dostop do operacijskega sistema,
- prioriteta po stopnjah in
- dodatne zagonske informacije o lokalnem skladu.

Funkcijski bloki (FB) vsebujejo tehnološke funkcije in imajo rezerviran del pomnilnika za shranjevanje lokalnih spremenljivk do naslednjega klicanja bloka.
Lastnosti:
- uporablja statične podatke,
- dodeljen podatkovni blok in
- primeren za programiranje kompleksnih, pogosto uporabljenih funkcij.

Funkcije (FC) prav tako vsebujejo tehnološke funkcije, vendar so brez funkcije pomnjenja za svoje lokalne spremenljivke.
Lastnosti:
- vrne v program svoj rezultat (brez statičnih podatkov) in
- v glavnem brez pomnjenja (nimajo dodeljenega podatkovnega bloka).

Podatkovni bloki (DB) služijo za shranjevanje različnih uporabniških podatkov, ki so na voljo vsem delom programa.
Lastnosti:
- strukturna shramba lokalnih podatkov in
- strukturna shramba globalnih podatkov (podatki so na voljo celotnemu programu).

Sistemski bloki (SFB, SFC, SDB) so bloki, ki vsebujejo sistemske funkcije. Ti bloki so sestavni del operacijskega sistema in ne uporabljajo uporabniškega pomnilnika. Bloke lahko uporabnik uporablja v svojem aplikativnem programu. SDB-ji vsebujejo podatke za konfiguracijo posameznih modulov in komunikacij.
Lastnosti:
- sistemski funkcijski bloki (SFB) so integrirani v operacijskem sistemu CPE,
- sistemske funkcije (SFC) so integrirane v operacijskem sistemu CPE in jih lahko uporabnik samo uporablja, ne more pa jih spreminjati; podobno kot blok FC je tudi blok SFC brez statičnih spremenljivk in
- sistemski podatkovni bloki (SDB) vsebujejo podatke za konfiguracijo sistema.



Slika 71: Program v FB1
Vir: Lasten

5.7 STEP 7 – PROGRAMSKA IN KONFIGURACIJSKA OPREMA ZA SIMATIC

STEP 7 je programska in konfiguracijska oprema za SIMATIC S7. Sestavlja ga več posameznih aplikacij, kjer vsaka od njih opravlja določeno funkcijo. Tako imamo na voljo funkcije, ki nas spremljajo od začetka ustvarjanja projekta do njegovega zaključka.

Funkcije lahko strnemo v naslednje skupine:
- funkcije za konfiguracijo strojne opreme,
- funkcije za konfiguracijo omrežij,
- funkcije za programiranje,
- funkcije za testiranje in servisiranje,
- funkcije za dokumentiranje in arhiviranje.

Glavni grafični vmesnik pri STEP 7 je SIMATIC Manager. Ta nam iz različnih aplikacij zbere vse potrebne podatke za oblikovanje projekta kot celote. Znotraj samega projekta so podatki razdeljeni glede na funkcijo in so predstavljeni kot objekti. Kadar želimo delati s posameznim projektom, se nam aktivira tudi ustrezno orodje za delo s tem objektom.

Pri programiranju imamo na razpolago dva načina:
- direktni vnos programa v centralno enoto (on-line programiranje) in
- programiranje pomnilnega modula v programirani napravi, brez povezave z avtomatizirano napravo.

Pomnilni modul se naknadno vstavi v centralno enoto (off-line programiranje). Programiranje uporabniškega programa poteka zaradi strukture krmilnika na ločeni programirani napravi. Prenos programa iz programirane naprave v krmilnik se lahko izvede s prenosom programa v

EPROM pomnilnik in s kasnejšim vstavljanjem v krmilnik, ali pa se programira direktno v pomnilnik krmilnika. Za to je potrebna večtočkovna povezava (MPI) med programirano napravo in krmilnikom.

Programski jezik STEP 7 nam dovoljuje programiranje s tremi programskimi metodami, ki so značilne za pomnilniško programirljiva krmilja:

- funkcijski načrt – FBD (*Function Block Diagram*),
- kontaktni način – LAD (*Ladder Logic*),
- nabor ukazov – STL (*Statement List*).

Funkcijski načrt – FBD:
FBD je programska metoda, pri kateri ukaze vnašate z grafično metodo. Funkcijski načrt uporablja logične bloke, znane iz Boolove algebre. Uporabljate ga lahko za kompleksne funkcije, ki so lahko predstavljene direktno z logičnimi bloki. Njegova velika prednost je direkten vnos programa, saj je programiranje z logičnimi bloki enostavno in hitro razumljivo. Ta programska metoda pa ni primerna za programiranje računskih operacij in zahtevnejše obdelave digitalnih vrednosti.

Kontaktni načrt – LAD:
LAD je prav tako grafična programska metoda, kjer so ukazi predstavljeni s simboli, ki izhajajo iz kontaktne tehnike. Glavna elementa kontaktnega načrta sta delovno in mirovno stikalo, zato je enostaven za priučitev. Slabost te programske metode je nepreglednost, zato ni primerna za komplekse naloge.

Nabor ukazov – STL:
STL je programska metoda, kjer ukaze lahko vnašate v pisni obliki. Ta programski jezik je zelo podoben zbirnemu jeziku in je sestavljen z vrsto mnemoničnih ukazov – to so ukazi za izvajanje. Vsak ukaz ustreza koraku procesorja skozi program. Prednost nabora ukazov je v hitrem izvajanju programa, saj sta optimirana tako čas obdelave kot tudi lokacija pomnilnika. V samem urejevalniku pa je delo še dodatno olajšano, saj lahko uporabljate simbolično opisovanje, možne pa so tudi funkcije za iskanje ter sprotno preverjanje pravilnega vnosa. Urejevalnik omogoča shranjevanje oz. skladiščenje večkrat uporabljenih programskih delov v standardno programsko knjižico, s čimer se omogoči kasnejša ponovna uporaba. Vsak ukaz je pri STL sestavljen iz operacije in operanda. Operand se dalje deli na označbo operanda in parameter. Pri samem ukazu je dodatno možno vpisati označbo, ki služi pri skokih, v urejevalniku pa nam je dovoljeno zaradi lažjega razumevanja programa vpisovati tudi komentarje.

Regulatorji za SIMATIC S7:

- Standardni PID regulator omogoča uporabo neprekinjenih regulatorjev, regulatorjev na pulzne in stopniščne odzive v uporabniškem programu,
- Modularni PID regulator se uporablja, če standardni PID ne zadostuje za reševanje avtomatizacijske naloge in
- mehki (*fuzzy*) regulatorji se uporabljajo za kreiranje logičnih sistemov. Ti sistemi se uporabljajo, ko je proces težko ali nemogoče matematično opisati, ko se procesi obnašajo nepredvidljivo, ko imamo opravka z nelinearnimi dogodki, a so izkušnje s procesi na voljo.

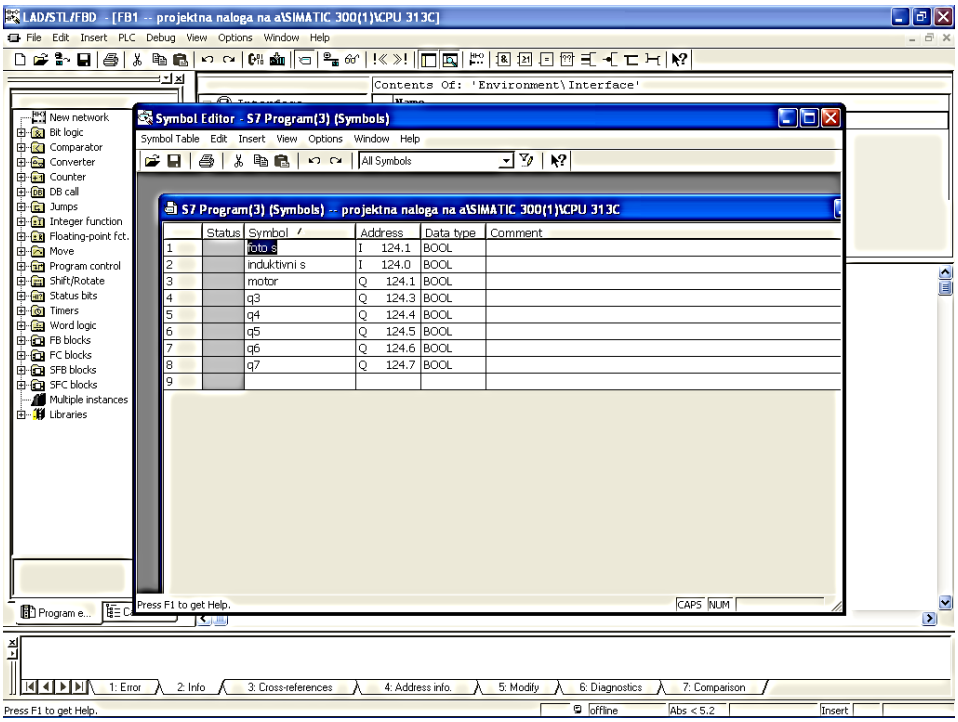
Programska orodja za povezavo človek – stroj:
HMI je poseben *software* za nadzor operatorjev in *monitoring*:

- SIMATIC WinCC SCADA predstavlja odprto procesno vizualizacijo,
- SIMATIC ProoTool in SIMATIC ProoTool/Lite so moderna orodja za konfiguracijo SIMATIC-ovih operaterskih panelov in kompaktnih enot SIMATIC C7 (izvedba Lite je samo za tekstovne panele) in
- Pro Agent omogoča hitro, načrtno procesno diagnosticiranje v načrtih in strojih, pri iskanju informacij o lokaciji in vzrokih napak.

5.8 PROGRAMIRANJE KRMILNIKA

Najprej morate napisati simbolno tabelo za vhodno/izhodne signale, ki jo kasneje še dopolnjujete z označevalci, števci in časovniki. Tudi simbolna tabela je napisana po standardu, vse lokacije v krmilniku so že vnaprej točno določene. Simbolna tabela se uporablja za lažje programiranje, saj bi v primeru pisanja signalov samo po lokacijah v krmilniku bil tak program zelo nepregleden in bi ga verjetno le s težavo naredili do konca.

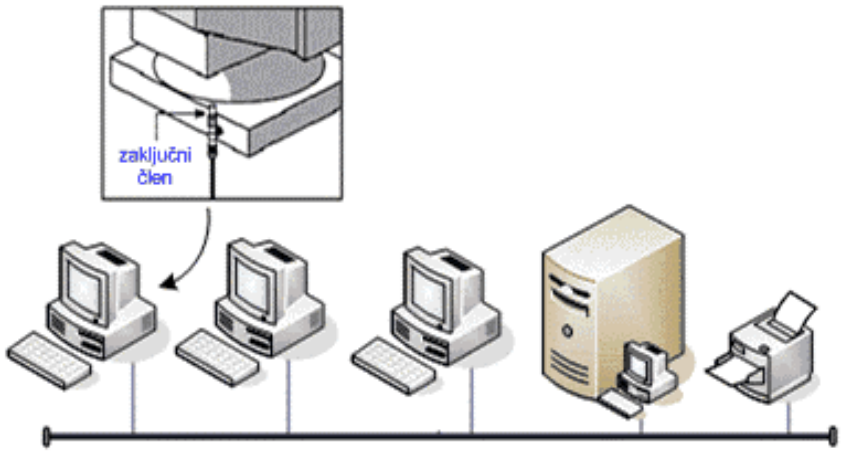
Na spodnji sliki je prikazan del simbolne tabele za vhodna signala. Ena vrstica tabele nam predstavlja en signal, in sicer ime (*symbol*), naslov ali lokacijo v krmilniku (*address*), podatkovni tip (*data type*) ter komentar (*comment*) signala. Zastavice, ki jih vidimo na začetku vrstice povedo, da so ti signali uporabljeni v varnostnih blokih.



Slika 72: Simbolna tabela
Vir: Lasten

5.9 POVEZAVA KRMILNIKA NA ETHERNET OMREŽJE

Ethernet je najpopularnejši protokol za lokalna omrežja. Ta industrijski standard so prevzeli mnogi proizvajalci omrežne strojne opreme. Danes mnogi problemi, ki bi lahko nastali zaradi nezdružljivosti strojne opreme različnih proizvajalcev, pri ethernetu praktično ne obstajajo. Danes omrežja ethernet delujejo s hitrostmi 10, 100 in 1.000 Mb/s (1 Gb/s), kar omogoča uporabo od domačih in malih poslovnih omrežij do visoko zmogljivih omrežnih hrbtenic.



Slika 73: Povezava omrežja
Vir: http://www.s-sers.mb.edus.si/gradiva/w3/omrezja/01_omrezja/01_omrezja.html

Standardi fizičnega sloja standarda ethernet opisujejo tipe kablov, iz katerih lahko zgradite omrežje, določate topologijo, največjo dolžino kabelskega segmenta in število obnavljalnikov, ki jih lahko uporabite. Pomembno je, da upoštevate standarde, saj je mehanizem CSMA/CD občutljiv na presluh in atenuacijo.

Tabela 4: Standardi

Oznaka	**Kabel**	**Topologija**	**Hitrost**	**Segment**
10Base5	RG8 koaksialni	Vodilo	10 Mb/s	500 metrov
10Base2	RG58 koaksialni	Vodilo	10 Mb/s	185 metrov
10BaseT	kategorija 3 UTP	Zvezda	10 Mb/s	100 metrov
FOIRL	večrodovno optično vlakno 62,5/125	Zvezda	10 Mb/s	1.000 metrov
10BaseFL	večrodovno optično vlakno 62,5/125	Zvezda	10 Mb/s	2.000 metrov
10BaseFB	večrodovno optično vlakno 62,5/125	Zvezda	10 Mb/s	2.000 metrov
10BaseFP	večrodovno optično vlakno 62,5/125	Zvezda	10 Mb/s	500 metrov
100BaseTX	kategorija 5 UTP	Zvezda	100 Mb/s	100 metrov
100BaseT4	kategorija 3 UTP	Zvezda	100 Mb/s	100 metrov

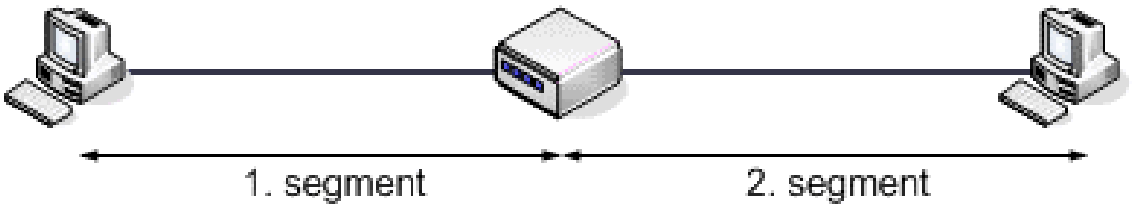
Vir: Lasten

5.10 OMREŽJE UTP ETHERNET

Vse ostale fizične topologije etherneta uporabljajo topologija zvezde pri kateri vsak segment povezuje računalnik z zvezdiščem. Najpopularnejši kabel v današnjem ethernetu je neoklopljena parica (UTP). Omogoča povezavo s hitrostmi 10 Mb/s, 100 Mb/s in 1000 Mb/s. 10BaseT uporablja samo dve parici, eno za sprejem in eno za oddajo.

Hitri ethernet (IEEE 802. 3u) opisuje dve specifikaciji, ki prav tako dovoljujeta največjo dolžino segmenta 100 metrov. 100BaseTX zahteva kable kategorije 5, ki imajo boljši prenos signalov. 100BaseT4 pa omogoča nadgradnjo omrežja s kabli kategorije 3. 100BaseT4 uporablja vse štiri parice.

Večina standardov za Gigabit ethertnet predpisuje optične kable, toda obstaja tudi možnost uporabe UTP po standardu IEEE 802.3ab. Standard 1000BaseT opisuje možnost nadgradnje obstoječega omrežja s kabli kategorije 5 (še bolje je, če so kabli po novih, strožjih, kriterijih pogosto označenih kot 5E). 1000BaseT doseže večje hitrosti z uporabo vseh štirih paric in uporabo pulzno-amplitudne modulacije PAM–5.



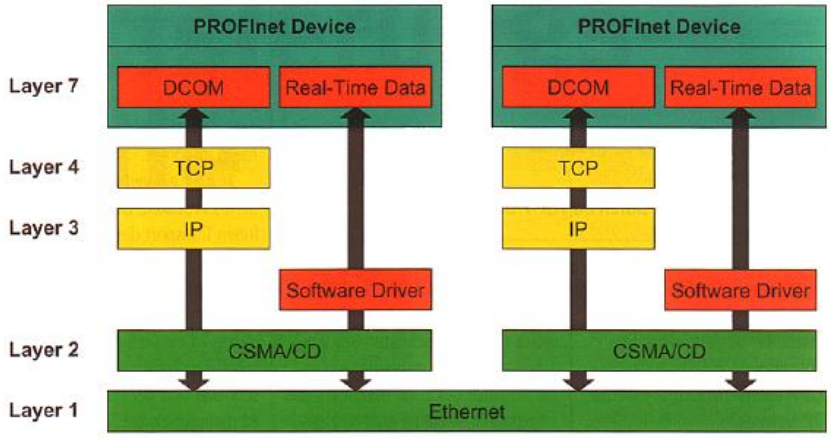
Slika 74: UTP Ethernet
Vir: http://www.s-sers.mb.edus.si/gradiva/w3/omrezja/01_omrezja/01_omrezja.html

5.11 OMREŽJE PROFINET

Vzporedno z industrijskim ethernetom že 15 let obstaja najbolj razširjen in največkrat uporabljen realno časovni protokol Profibus, za katerim stoji Siemens. Omrežje Profibus je namenjeno za hitre in zanesljive komunikacije v procesih avtomatizacije. Maksimalna hitrost prenosa podatkov je 12 Mbit/s, ki je bila v času uvajanja revolucionarna, danes pa ne pokriva več zahtev. Težave se lahko pojavijo tudi pri številu postaj, priključenih v omrežje, ki jih je lahko največ 127, vsak repetitor, ki poveča doseg omrežja, pa tudi predstavlja eno od teh naprav. Omrežje Profibus tako prihaja na skrajno mejo svoje zmogljivosti zaradi teh pomanjkljivosti

Siemens s pomočjo nekaterih partnerjev pripravlja nov protokol. Velike hitrosti prenosa pri ethernetu odpirajo nove možnosti za realnočasovne komunikacije. Ethernet v svoji klasični izvedbi ne zagotavlja determinističnega obnašanja, za nekatere sisteme pa je determinizem časa oddajanja ključnega pomena. V industriji se procesi delijo po reakcijskem času. Za večji del naprav je dovoljen reakcijski čas med 10 in 100 ms, kateremu je zadoščeno z uporabo klasičnega industrijskega etherneta z ustreznimi gonilniki. Profinet V2 ponuja optimiran

komunikacijski kanal na bazi programske opreme. Sinhronizacija takta pa predstavlja še višje zahteve za realno časovno obnašanje.



Slika 75: Profinet
Vir: Lasten

Povzetek:

Krmilnik je programirljiva naprava, ki vsebuje lasten mikroprocesor, RAM, hitre vhodno-izhodne enote. Omogoča regulacijo in krmiljenje celotne proizvodnje ali le ene naprave, lahko sprejema signale preko senzorjev ali direktno od objekta vodenja. Ti signali se potem v krmilniku obdelajo. Obdelani signali potujejo nazaj k objektu vodenja. Komunikacija med osebnim računalnikom in programirljivim logičnim krmilnikom ponavadi poteka preko serijskih vrat ali TCP/IP protokola. Programirljivi krmilnik je v osnovi CPU (*Central Processing Unit*), ki vsebuje program in je povezan z vhodno/izhodnimi napravami. Program nadzira PC (*Programmable Controller*) v tem smislu: ko se spremeni vhodni signal na vhodni napravi, se ustvari ustrezen odziv (glede na napisan program v pomnilniku krmilnika). Vhodne naprave so lahko fotoelektrični senzorji, stikala, omejevalniki in končna stikala. Izhodni signali so lahko 24V napetost, tokovni signal, signal, ki vklopi rele.

Vprašanja:

1. Zgradba krmilnega sistema. Kaj je napajalna enota?
2. Vrste dajalnikov signalov.
3. Lastnosti in prednosti sekvenčnih krmilij – na kaj jih delimo?
4. Za kaj uporabljamo časovne funkcije in vrste le-teh?
5. Naštejte glavna področja uporabe programirljivih krmilnikov.
6. Delitev krmilnih sistemov SIMATIC v podskupine in glede na velikost.
7. Opišite in skicirajte centralno procesno enoto.
8. Kaj veste o programski opremi Siemens?
9. Opišite kontaktni načrt (LAD).
10. Kakšna je razlika med kompaktnim in modularnim PLK-jem, navedite prednosti in slabosti posameznega tipa.

6 UVOD V PROGRAM SIMATIC MANAGER

V poglavju UVOD V SIMATIC MANAGER so vam predstavljeni prvi koraki v programskem okolju Simatic Manager. Poglavje nam prikazuje, kako se ustvari nov projekt, kako se nastavi strojna oprema in razloži osnovne funkcije.

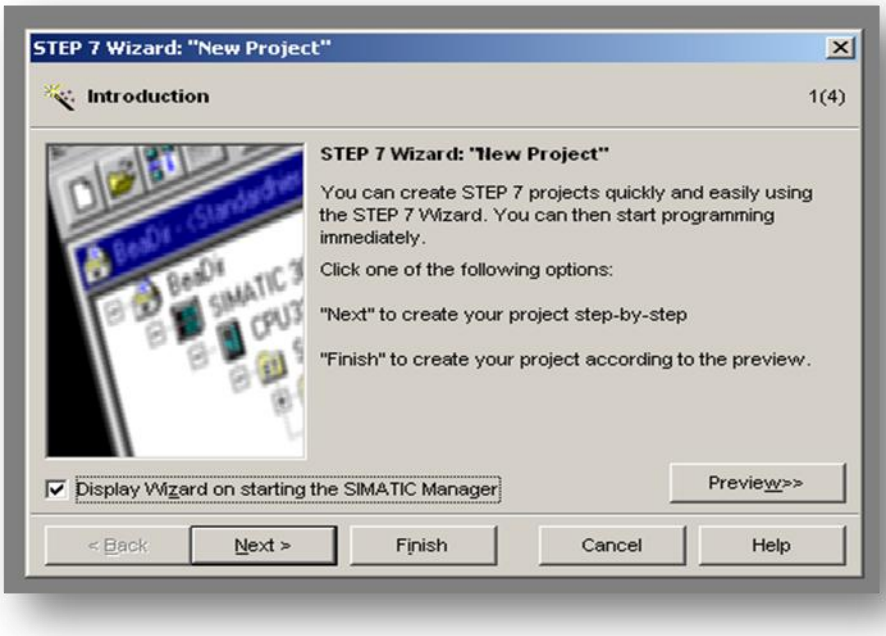
Kaj je SIMATIC STEP 7? STEP 7 je standardni programski paket, ki se uporablja za konfiguriranje in programiranje SIMATIC programirljivih logičnih krmilnikov.

Obstajajo naslednje različice STEP 7:
- STEP 7 Micro / DOS in STEP 7 Micro / Win za enostavnejše stand-alone aplikacije,
- STEP 7 za aplikacije na S7-300/S7-400 SIMATIC, SIMATIC M7-300/M7-400,
- SIMATIC C7.

6.1 NOV PROJEKT V SIMATIC MANAGERJU

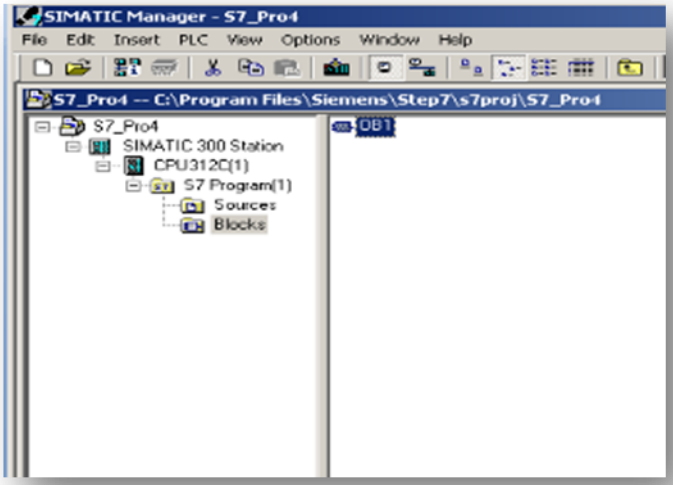
Nov projekt v Simatic Managerju

Ko odpremo Simatic, se nam odpre okno "*New Project*", kjer si nastavimo želene vrednosti, kot so izbira krmilnika, vrsta programiranja, naslov projekta in kam bomo program shranili. Simatic nam dopušča pisati program v STL, FBD in LAD diagramu. V katerem jeziku bomo pisali program, se lahko odločimo že na začetku novega projekta.



Slika 76: STEP 7 čarovnik
Vir: Lasten

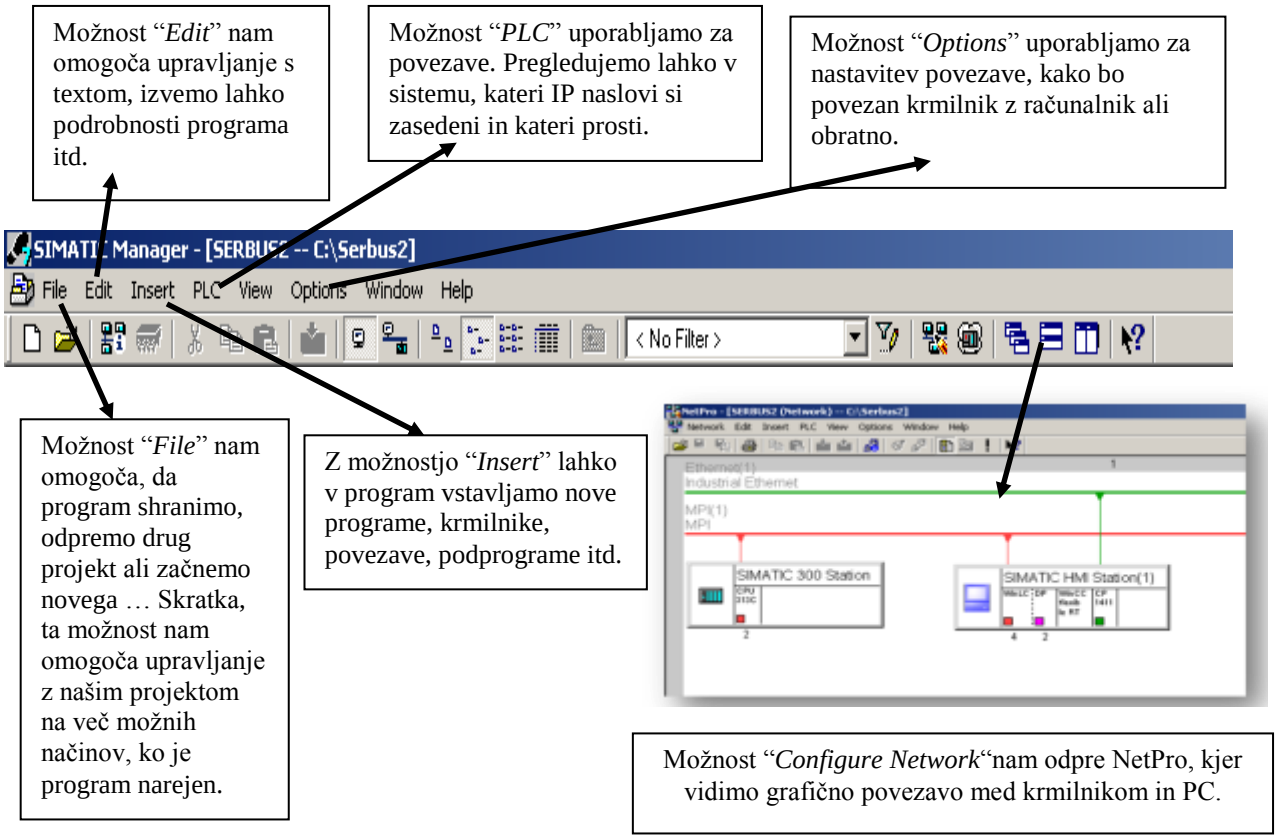
Ko si v začetnih okencih nastavimo želene vrednosti, se nam odpre nov Project v Simaticu. To zgleda nekako takole:



V levem stolpcu je vedno prikazano, kje se program nahaja, kakšen je naslov, kateri krmilnik uporablja, in naše programe, ki smo jih sprogramirali.

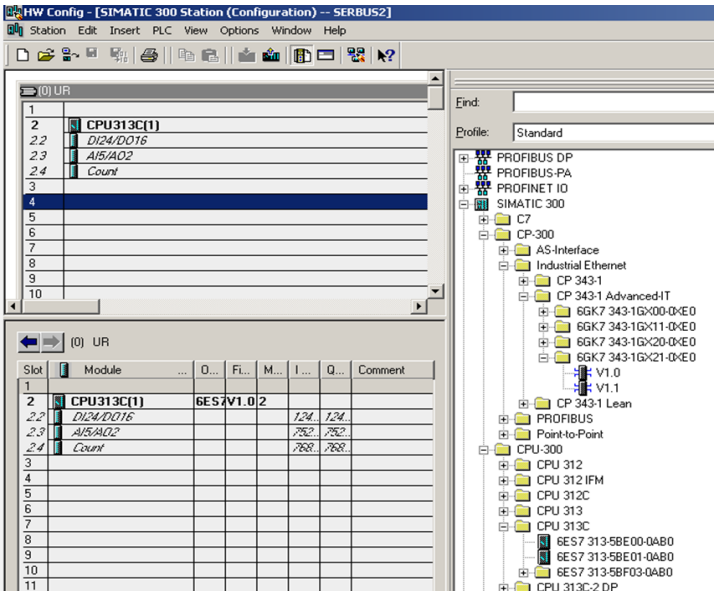
Slika 77: Projekt v Simatic Managerju
Vir: Lasten

<u>Orodna vrstica Simatic Manager-ja:</u>



Slika 78: NetPro
Vir: Lasten

Simatic Manager Hardware



Slika 79: Simatic Manager – Hardware
Vir: Lasten

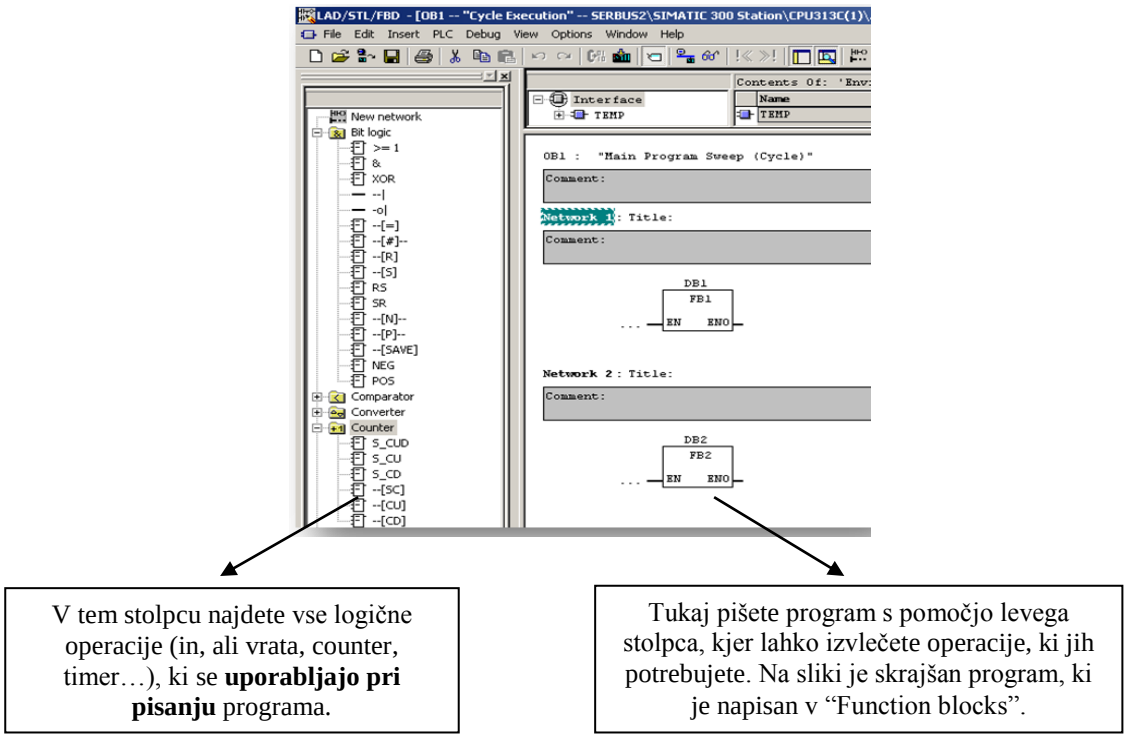
V Hardware-u nastavljate, kakšno povezavo boste imeli, izbirate lahko med MPI povezavo, industrijskim internetom in PROFIBUS-om.

Ko se odločite za povezavo, vas program vpraša o lastnostih krmilnika ipd., nato prepišete kar je potrebno v Hardware in povezava mora delovati.

V Hardware-u nastavite *MAC address* (MAC naslov kartice krmilnika), Subnet Mask (maska mora biti enaka na krmilniku in računalniku) ter mu dodelite IP naslov (naslov mora biti statičen naslov, da se ne spreminja in nam po določenem času ne prekine povezave).

Ustvarjanje programa

Ko nastavite Hardware, lahko začnete ustvarjati program.

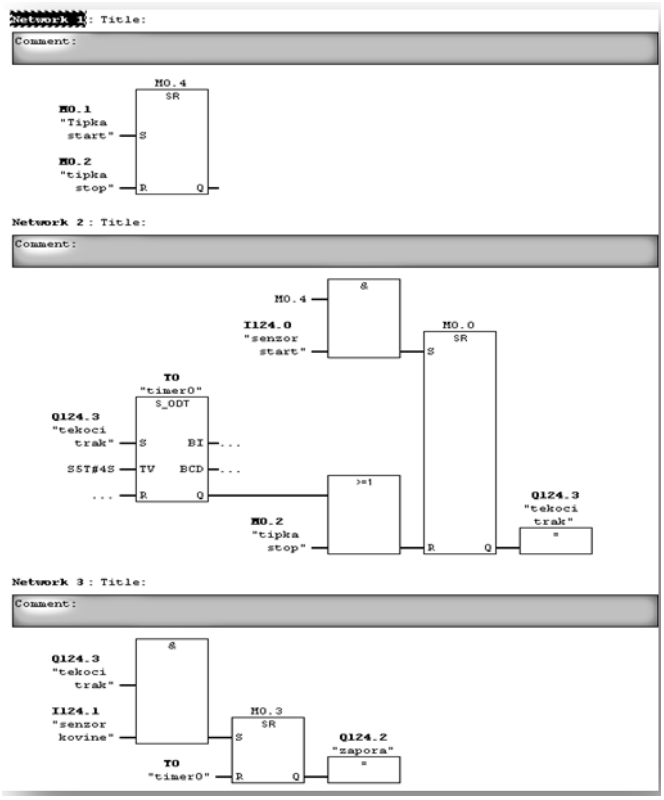


V tem stolpcu najdete vse logične operacije (in, ali vrata, counter, timer…), ki se **uporabljajo pri pisanju** programa.

Tukaj pišete program s pomočjo levega stolpca, kjer lahko izvlečete operacije, ki jih potrebujete. Na sliki je skrajšan program, ki je napisan v "Function blocks".

Slika 80: Ustvarjanje programa
Vir: Lasten

Program
Program v FBD-ju na prikazuje spodnja slika:



Slika 81: Program v Simatic STEP 7 v FBD
Vir: Lasten

Povzetek:

Programsko orodje SIMATIC STEP 7 omogoča programiranje in konfiguriranje krmilnikov serije 300 in 400. V programu morate najprej definirati strojno opremo, ki jo boste uporabljali v projektu. Izbrati morate ustrezno povezavo med osebnim računalnikom in krmilnikom za prenos programa in diagnosticiranje. Definirati morate ustrezen protokol za povezavo periferije s krmilnikom. Program lahko pišete v različnih načinih, npr. z bloki ali z lestvičnim diagramom. Novejši paketi pa nam omogočajo tudi programiranje v grafičnem načinu, katerega opisuje poglavje 7.

Vprašanja:

1. Kaj lahko poveste o Novem projektu v Simatic Managerju?
2. Kateri elementi oz. deli sestavljajo HW Config?
3. Na katere elemente morate biti pazljivi pri sestavi Hardware-a?
4. Napišite program za dvoročni vklop v LAD-u, FBD-u ter STL-u.
5. Razložite pomen oznak OB1, FB2, FC3, DB2 ter povezovanje med posameznimi bloki.

7 OBLIKOVANJE SEKVENČNEGA KONTROLNEGA SISTEMA, KI TEMELJI NA PRIMERU VRTALNEGA STROJA

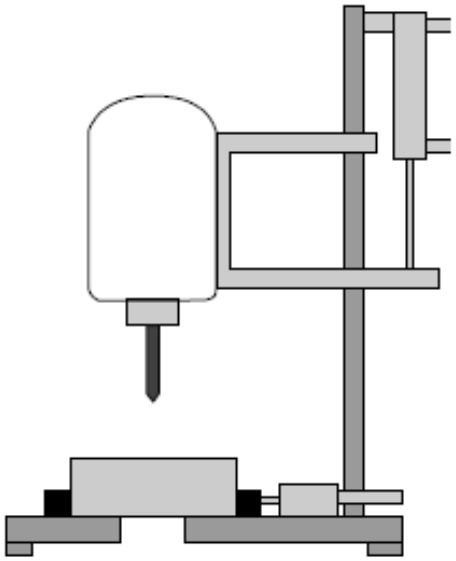
V poglavju OBLIKOVANJE SEKVENČNEGA KONTROLNEGA SISTEMA, KI TEMELJI NA PRIMERU VRTALNEGA STROJA je prikazan delovni proces na primeru vrtalnega stroja. Predstavljene so njegove zahteve in postopek za ustvarjanje sekvenčnega kontrolnega sistema. Prikazane so vam tudi tehnološke naloge in risbe, razloženo je tudi, kako se ustvari simbolna tabela, kako se definirajo vhodi in izhodi.

7.1 ZAHTEVE

V prvem koraku se boste naučili učinkovito konfigurirati sekvenčni kontrolni sistem, v naslednjem koraku pa boste vodeni korak za korakom skozi naloge, ki jih morate opraviti v SIMATIC Managerju in vS7-GRAPH, da boste lahko:

- ustvarili sekvenčni kontrolni sistem,
- naložili na CPU, in
- ga testirali.

Pravilen programski primer je založen s S7-GRAPH kot projekt, ki se imenuje "Zen02_01_S7GRAPH_Drill".



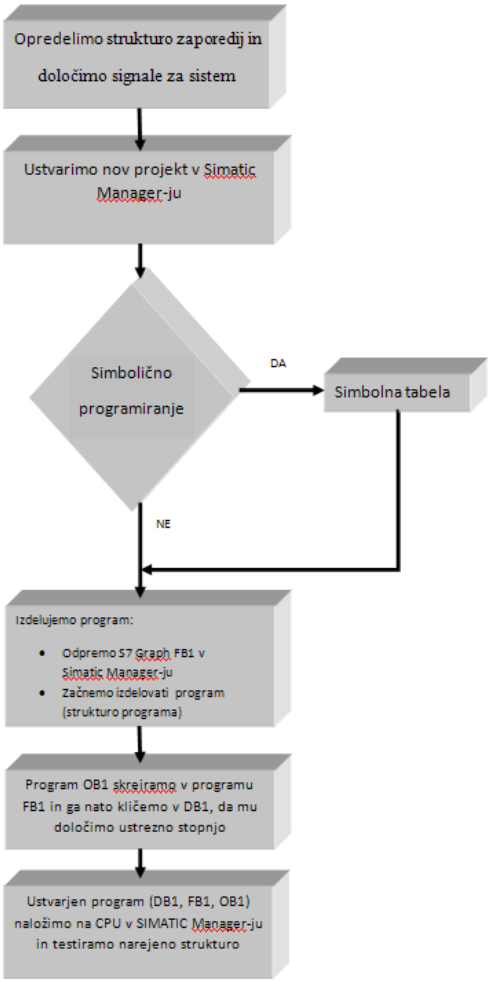
Slika 82: Prikaz delovnega procesa
Vir: S7 GRAPH za S7 – 300/400

Da lahko programirate in testirate primer "Vrtalnega stroja", potrebujete naslednjo strojno in programsko opremo:

- programska naprava ali PC s STEP 7 s standardnim paketom in S7-GRAPH z optimalnim paketom,
- MPI povezavo na programirljivi logični krmilnik,
- programirljivi logični krmilnik (v našem primeru S7-300), sestavljen iz naslednjih komponent: standardni tir, 24V napajanje, CPU 314 in digitalni izhodni/vhodni modul (8DI-8DO),
- Kot alternative PLK-ju : "PLK simulacija" S7 izbirni paket.

7.2 POSTOPEK ZA USTVARJANJE SEKVENČNEGA KONTROLNEGA SISTEMA

Tokovni diagram prikazuje proces za ustvarjanje sekvenčnega kontrolnega sistema za primer svedra:



Slika 83: Diagram poteka programa
Vir: Lasten

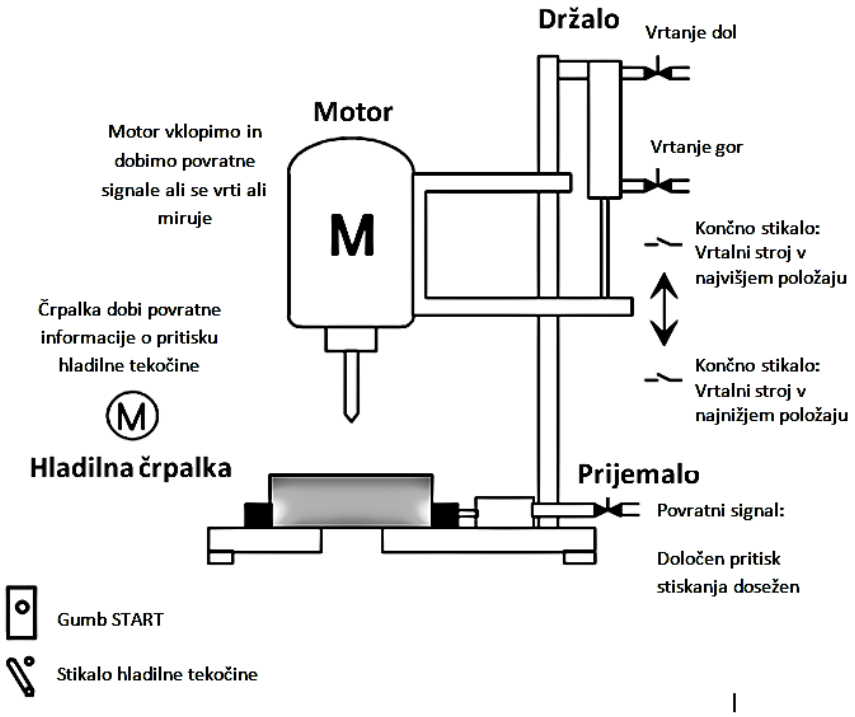
7.3 TEHNOLOŠKE NALOGE IN FUNKCIJSKI DIAGRAM

Najprej je potrebno sprogramirati sekvenčno kontrolni sistem za avtomatiziranje vrtalnega stroja. Naloga stroja je prikazana v tehnološki risbi in sekvenčni proces je prikazan v funkcijskem diagramu.

7.4 TEHNOLOŠKA RISBA – SESTAVA VRTALNEGA STROJA

Vrtalni stroj je sestavljen iz naslednjih elementov:

- vrtalni stroj s povratno informacijo s sveder deluje/ne deluje,
- start tipka in tipka za hladilno tekočino,
- hladilna črpalka s povratno informacijo za pritisk hladilne tekočine,
- vpenjalna naprava s povratno informacijo za pritisk vpenjanja,
- tir za spuščanje/višanje svedra z limitom za vrtanje gor in dol.



Slika 84: Tehnološka shema
Vir: S7 GRAPH za S7 – 300/400

ZAČETNO STANJE:

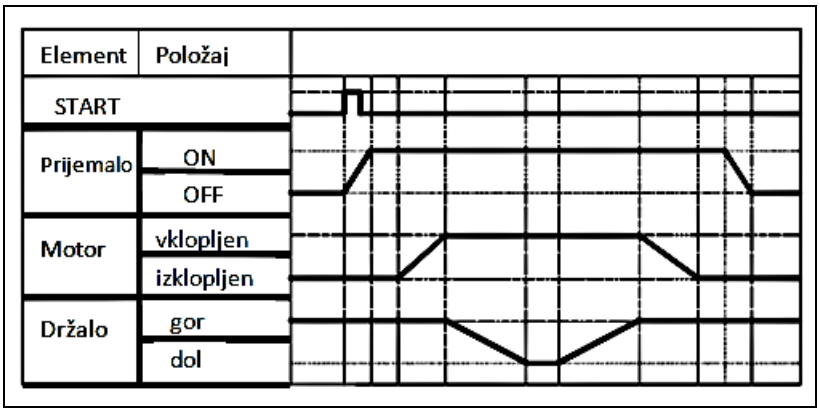
Začetno stanje vrtalnega stroja je definirano kot naslednje:

- Vrtalni stroj in hladilna črpalka sta ustavljeni.
- Vrtalni stroj je čisto v zgornji poziciji.
- Ni delovnega kosa v vpenjalni napravi.

7.5 FUNKCIJSKI DIAGRAM – VRTALNO ZAPOREDJE

Celotno vrtalno zaporedje je definirano v naslednje sekcije:

- vstavljanje delovnega materiala (ročno),
- če je potrebno, aktivirajte tipko za hladilno tekočino (odvisno od materiala),
- aktiviranje stroja s start tipko (motor prične delovati),
- vstavljanje delovnega materiala s pravilno silo,
- aktiviranje hladilne črpalke (če je že prej aktivirana tipka za hladilno tekočino),
- nižanje vrtalnega stroja v najnižjo pozicijo (za vrtanje),
- ustavite 0.5 sekund na najnižji poziciji (vrtanje),
- dvigovanje vrtalnega stroja z ročico na najvišjo pozicijo,
- odstranjevanje delovnega materiala, ugasni vrtalni motor in hladilno črpalko,
- odstranjevanje delovnega materiala (ročno).



Slika 85: Funkcijski diagram
Vir: Lasten

7.6 IZBOR STRUKTURE SEKVENČNIKA

Preden ustvarite program za sekvenčnik, morate vključiti koncept, pri katerem razdelite vrtalne operacije v posamezne korake. Osnovni koncept je tehnološka risba in diagram poteka.

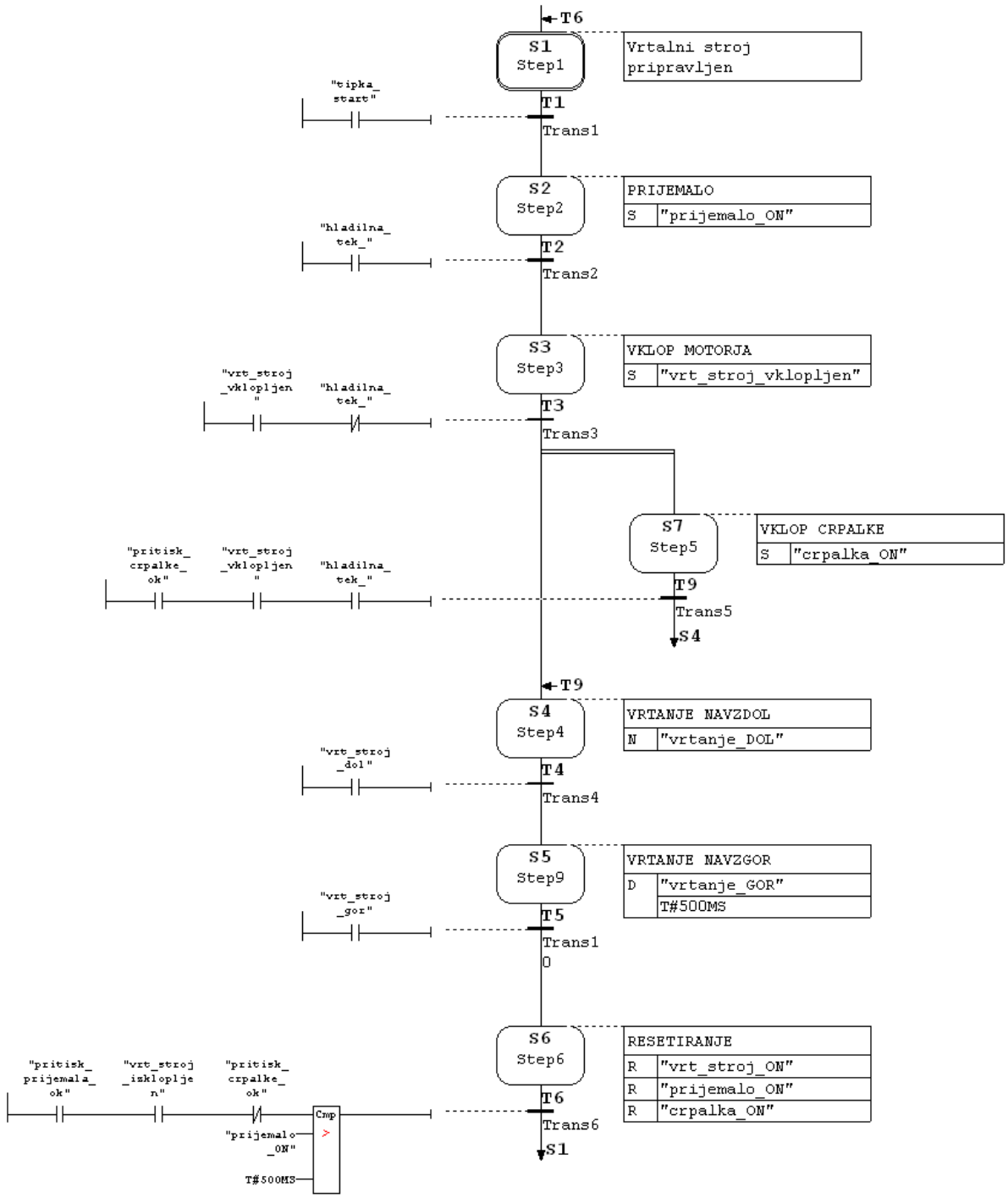
7.7 DELJENJE VRTALNIH PROCESOV V POSAMEZNE KORAKE – STRUKTURA SEKVENČNIKA

Vrtalni proces je opisan v S7-GRAPH v obrazcu sekvenčnika. Sekvenčnik prikazuje sekvenco posameznih korakov in pogoje, ki kontrolirajo procese, kako gredo na naslednji korak.

Za opredelitev strukture sekvenčnika se držite naslednjih korakov:

1. Razdelite vrtalni proces v korake in opredelite zaporedje korakov.
2. »Korak S2 sledi S1« ali »Korak 3 sledi ali korak S4 ali S7«.

3. Za vsak posamezni korak opredelite naloge, ki morajo biti opravljene v koraku (npr. v koraku S1 je naloga »vrtalni stroj pripravljen«, ali v S3 je naloga »ugasni vrtalni stroj«).
4. Nato se odločite za vsak posamezni korak, kakšen pogoj mora biti opravljen, da lahko gre proces na naslednji korak (npr. T1 ima pogoj »Vrtalni stroj deluje – start tipka aktivirana« ali za T5 »vrtalni stroj je v najvišji poziciji«).



Slika 86: Prikaz programa
Vir: Lasten

Opredelitev sistemskih signalov

Ko ste razdelili vrtalni proces v posamezne korake, morate definirati vhodne in izhodne parametre za vsak posamezni korak. Osnove oblikovalskim konceptom je tehnološka risba in diagram poteka.

7.8 DEFINIRANJE IZHODOV IN VHODOV

Naredite seznam izhodov in vhodov za vrtalni stroj v obliki tabele.

Če hočete program simbolično, vpišite želeno simbolično ime (npr. vhod I 0.4 »Cl_press_ok«) za absolutne vhode in izhode pa kakšne komentarje, da bo program laže razumljiv (npr. »sila za stisk delovnega materiala je dosežena«).

V primeru vaje je domneva, da so stikala in konektorji vrtalnega stroja nadzorovani z vhodi in izhodi iz digitalnega izhod/vhod modula od S7-300 programirljivega logičnega krmilnika. Izhod/vhod modul ima 8 vhodov in 8 izhodov. Standardne vrednosti naslovov vhodov in izhodov modula v reži 4 so naslednje: I 0 do I 0.7 in Q = 0.0 do Q 0-7.

Tabela 5: Simbolna tabela

Absolutni naslovi	**Simbolični naslovi**	**Razlaga**
Vhodi	**v programu (I)**	
I 0.0	Vrt_stroj_vklopljen	Povratni signal nam pove, če se motor vrti
I 0.1	Vrt_stroj_izklopljen	Povratni signal nam javi, da se motor ne vrti
I 0.2	Vrt_stroj_dol	Končno stikalo, da je motor v najnižjem položaju
I 0.3	Vrt_stroj_gor	Končno stikalo, da je motor v najvišjem položaju
I 0.4	Pritisk_prijemala_ok	Povratni signal nam javi, da je pritisk prijemala dosežen.
I 0.5	Hladilna_tek_	Če imamo dovolj hladilne tekočine (odvisno od obdelovanca)
I 0.6	Pritisk_črpalke_ok	Povratni signal, da je pritisk črpalke dosežen.
I 0.7	Tipka_start	Start tipka vrtanja.
Izhodi	**v programu (Q)**	
Q 0.0	Vrt_stroj_ON	Vklopi motor za vrtanje.
Q 0.1	Črpalka_ON	Vklopi hladilno črpalko.
Q 0.2	Vrtanje_DOL	Držalo se pomika proti najnižnji poziciji vrtanja.
Q 0.3	Vrtanje_GOR	Držalo se pomika proti najvišji poziciji vrtanja.
Q 0.4	Prijemalo_ON	Prijemalo stisne do določenega pritiska.

Vir Lasten

7.9 POSTOPEK USTVARJANJA NOVEGA PROJEKTA

Projekti za sekvenčno-kontrolne sisteme se ne razlikujejo od drugih projektov v STEP 7.

Za ustvarjanje novega projekta v SIMATIC Managerju sledite spodnjim navodilom:
1. V orodni vrstici izberite **File > New.**
2. Projekt imenujte »vrtalni stroj«.

Vstavljanje S7 programa

V tem primeru konfiguracija strojne opreme ni potrebna, dokler je uporabljen standardni naslov modula IZHOD/VHOD v reži 4. Tako lahko takoj vstavite S7 program v projekt mapo v SIMATIC Managerju. S7 program služi kot mapa za bloke iz uporabniškega programa, izvornih datotek in simbolov.

Sledite spodnjim navodilom:
1. Izberite projekt »vrtalni stroj«.
2. Izberite komando **Insert > Program > S7 Program.**
3. Imenujte S7 program kot »vrtalni stroj program«

Mapa izvornih datotek, blokov in simbolov se ustvari avtomatsko takoj, ko vstavite S7 program. Prazna 0B1 je tudi ustvarjena v mapi BLOCKS.

7.10 DEFINIRANJE SIMBOLNE TABELE

Ko programirate v STEP 7, delate z naslovi, kot so I/O signali, bitov pomnilnika, števci, časovniki, bloki podatkov in funkcijskimi bloki. Te naslove lahko vstavljate v programe v obliki absolutnega formata (npr. I1.1, M2.0, FB21).

Videli boste, da je program veliko bolj razumljiv in lažji za branje, če uporabljate simbole (na primer Motor_A_ON) namesto absolutnih naslovov. Za omogočanje uporabe simbolov lahko vpišete ime, absolutne naslove, podatkovni tip in komentar za vsak uporabljen naslov.

Ko ste definirali simbol, je le-ta lahko uporabljen čez celoten program v programskem modulu.

Ustvarjanje simbolne tabele

Če hočete napisati program z uporabo simbolnih naslovov, je priporočljivo ustvariti simbolno tabelo.

1. Odprite simbolno tabelo v mapo »program vrtalnega stroja«, s tem da dvojno kliknete na **Symbols.**
2. Uredite tabelo kot je prikazano spodaj.
3. Shranite simbolno tabelo z uporabo komande **Table > Save.**

Vpisi od 1 do 14 so potrebni za simbolično predstavitev izhodov in vhodov. Vpisi od 15 do 18 dovoljujejo simbolično predstavitev blokov.

S7 Program(1) (Symbols) -- glamink\SIMATIC 300 Station\CPU313C(1)

	Status	Symbol	Address	Data type	Comment
1		G7_STD_3	FC 72	FC 72	
2		Vrt_stroj_vklopljen	I 0.0	BOOL	Povratni signal nam pove, èe se motor vrti
3		Vrt_stroj_izklopljen	I 0.1	BOOL	Povratni signal nam javi, da se motor ne vrti
4		Vrt_stroj_dol	I 0.2	BOOL	Koněno stikalo, da je motor v najnižjem položaju
5		Vrt_stroj_gor	I 0.3	BOOL	Koněno stikalo, da je motor v najvišjem položaju
6		Pritisk_prijemala_ok	I 0.4	BOOL	Povratni signal nam javi, da je pritisk prijemala dosežen.
7		Hladilna_tek_	I 0.5	BOOL	Če imamo dovolj hladilne tekoèine (odvisno od obdelovanca)
8		Pritisk_crpalke_ok	I 0.6	BOOL	Povratni signal, da je pritisk èrpalke dosežen.
9		Tipka_start	I 0.7	BOOL	Start tipka vrtanja
10		INIT_SQ	M 0.0	BOOL	
11		Cycle Execution	OB 1	OB 1	
12		Vrt_stroj_ON	Q 0.0	BOOL	Vklopi motor za vrtanje.
13		Crpalka_ON	Q 0.1	BOOL	Vklopi hladilno èrpalko.
14		Vrtanje_DOL	Q 0.2	BOOL	Držalo se pomika proti najnižnji poziciji vrtanja.
15		Vrtanje_GOR	Q 0.3	BOOL	Držalo se pomika proti najvišji poziciji vrtanja
16		Prijemalo_ON	Q 0.4	BOOL	Prijemalo stisne do doloèenega pritiska
17		Temp_M_crpalka_off.T	Q 0.5	BOOL	
18		TIME_TCK	SFC 64	SFC 64	Read the System Time
19					

Slika 87: Urejevalnik simbolne tabele
Vir: Lasten

7.11 USTVARJANJE S7-GRAPH FUNKCIJSKEGA BLOKA IN PROGRAMIRANJE SEKVENČNIKA

Ustvarjanje S7-GRAPH funkcijskega bloka

S7-GRAPH funkcijski blok bo vseboval sekvenčnik. Za ustvaritev S7-GRAPH funkcijskega bloka sledite spodnjim navodilom:

1. Odprite mapo **»Blocks«** v »mapi vrtalni program« v SIMATIC Managerju.

2. Izberite komando **Insert > S7 Block > Function Block.**

3. Nastavite S7-GRAPH kot jezik v **»Properties«** oknu.

Rezultat: prazen funkcijski blok s standardno številko 1 je narejen v mapi **»Blocks«**.

Programiranje sekvenčnika

Ko boste zagnali S7-GRAPH urejevalnik z dvojnim klikom na FB1, sistem samodejno vstavi prvi korak in prvo tranzicijo. Priporočeno je ustvariti strukturo v »sekvenčniku« v obliki displeja. Za prikaz pogojev in nalog aktivirajte **view > Display with > conditions and actions** komando. Z uporabo miške in sekvenčno orodno vrstico v levem kotu ekrana lahko potem pozicionirate vse ostale korake in tranzicije, alternativno vejo in skok od konca sekvenčnika do začetka sekvenčnika. Za to obstajata dve metodi.

Metoda 1: »Direktna« metoda

1. Izberite prehod 1 in potem kliknite dokler  **insert step + transition,** dokler ne prispete do koraka/prehoda 6.
2. Izberite korak 3 in potem izberite ikono  **Odprite alternativno vejo** – odpre alternativno vejo za pridobitev hladilne tekočine. Veja se začne s prehodom 7.
3. Z miško izberite ikono  **insert step + transition**. vstavite korak 7 (S7) in prehod 8 (T8)
4. Izberite ikono  **close alternative branch** in potem izberite prehod 3.
5. Nato dokončajte strukturo sekvenčnika, s tem da najprej izberete prehod 6 in potem kliknete ikono  **insert jump** in potem izberete korak 1.

Metoda 2: » povleci in spusti«

1. Vrnite se na SIMATIC Manager in potem ustvarite funkcijski blok FB2 v mapi **»Blocks«** kot je opisano zgoraj. Nato še enkrat izberete »GRAPH« kot jezik.
2. Zaženite S7-GRAPH urejevalnik z dvojnim klikom na FB2 v mapi **»Blocks«.**
3. Izberite komando **Insert > Drag-and-Drop**.
4. Z miško izberite ikono  **insert step + transition** in potem kliknite na zadnjo tranzicijo od invidualnih elementov, dokler ne prispete na korak/prehod 6.
5. Izberite ikono  **open alternative branch** in potem odprite alternativno vejo za hladilno tekočino s tem, da kliknete na korak 3. Veja se začne z prehodom 7.
6. Z miško kliknite na ikono  **insert step + transition**, da vstavite korak 7 (S7) in prehod 8 (T8)
7. Izberite ikono  **close alternative branch** in potem izberite prehod 8 ter potem prehod 3.
8. Nato zaključite sekvenčno strukturo z  **insert jump** s tem, da kliknete prehod 6 in potem korak 1.

Opomba:

Zaprite FB2, preden zaženete programske koračne akcije. Ustvarili ste ta FB zato, da ste poskusili drugo metodo za ustvarjanje sekvenčne strukture. Ko zaprete ta FB, odgovorite na vsa okna z **NO** za vaje, naprej boste delali s FB1.

7.12 PROGRAMSKE AKCIJE

Obstajata tudi dve metodi, ki sta na voljo za ukrepe, programiranje koraka in prehodov: Direct in Drag-and-Drop.

Spodaj je opisan postopek prevzema, ki ste ga izbrali z menijskim ukazom Insert > Drag-and-Drop:

1. Izberite menijski ukaz Insert > Action.
   Rezultat: miškin kazalec se nato pojavi, kot je prikazano spodaj:

2. Vstavite prazno akcijo s klikom na polje delovanja.

   Vpišite akcije.
   Ukrep vključuje navodila in naslov. Za program vrtanja štirih različnih navodil so potrebni koraki:

   •S – set izhod
   •R – reset izhod
   •N– Nonholding: dokler je korak aktiven, je stanje signalov
   •D – Delay: naslov je nastavljen na logično 1, po določenem času poteče

   Po aktiviranju koraka reset se deaktivira korak.

7.13 PREHODI PROGRAMIRANJA

Bit logic navodila"normalno odprt kontakt", "normalno zaprt kontakt" in"primerjalni" se uporabljata za korak, ki omogoča pogoje v prehode. Če želite program prehodov:

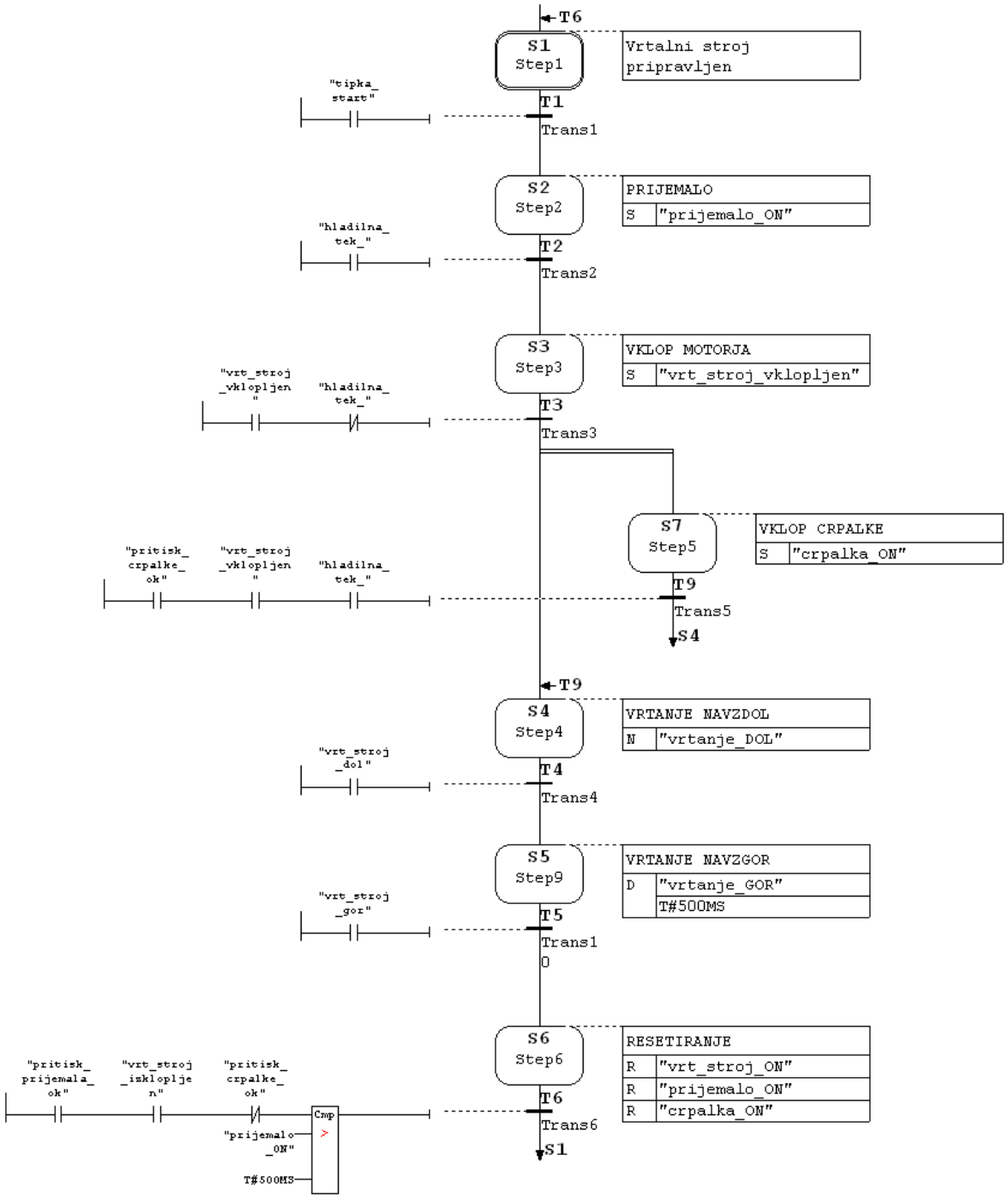
1. Set"LAD" pogled in izberite ustrezno ikono"LAD /FBD" orodna vrstica


vstavite navadno odprt stik


vstavite običajno zaprt kontakt


vstavite primerjavo

2. Položaj simbolov na ustreznih mestih s klikom na prehodu črte.
Lahko zapustite vstavite način kadar koli s tipko ESC.
3. Vnesite naslove. Kliknite ikono z zahtevanim poljem besedila. Nato vpišite absolutni ali simbolični naslov (na primer I0.7., "Start_switch").
4. Če želite, lahko vnesete komentar za sekvenčnik. V"sekvencer" pogled, komentar polje v levem zgornjem kotu in se lahko odprejo s klikom z miško.



Slika 88: Program
Vir: Lasten

Zgornja slika prikazuje končano zaporedje.

Pri programiranju primerjalnika lahko uporabite sistemske informacije in jih zapišete po korakih ter jih postavite za naslove.

Naslovi imajo naslednji pomen:

KORAK_T: trenutno aktivacijo ali čas zadnje aktivacije.
KORAK_U: trenutno aktivacijo ali zadnjo aktivacijo brez motenj.

7.14 PROGRAMMING MONITORING FUNCTIONS

Programiranje nadzorne funkcije:

1. Dvakrat kliknite na korak 2 (step 2) za spreminjanje iz sekvenčnega pogleda *("sequencer" view)* v pogled po korakih *("single step " view).*

2. Izberite ikono 
Vstavite primerjevalnik *("comparator")* v "LAD/FBD" vrstico z orodji.

3. Vstavite primerjevalnik *("comparator")* na ustrezno mesto na linijo "supervision" in v njega vnesite želeni čas.

7.15 IZVEDLJIVOST S7 GRAPH

S7-GRAPH ima na voljo dve možnosti za ustvarjanje FBS.

Celotna koda:

Celotna koda,ki je potrebna za izvajanje vseh S7-GRAPH FB, je vključena v FB. Če imate več S7-FBS GRAPH, to pomeni precejšnje povečanje pomnilnika.

Standardne FC zahteve:

Da bi zmanjšali zahteve za pomnilnik, ima S7-GRAPH naslednji dve možnosti:

- Uporabljate standardni FC, ki vsebuje glavno kodo oddelkov za vse FBS. Ta FC se kopira v svoj projekt samodejno, ko izberete to možnost.
- FBS, proizvedeni z uporabo te metode, so precej manjši.

Za ta primer uporabi možnost urejanja "Full code".

Najprimernejši FC je odvisen od zmogljivosti vašega CPU. Izberite enega izmed naslednjih standardnih FCS:

Številka FC	**Funkcionalnost**
FC72	Je privzet, delate s FC72. Ne pozabite, da mora biti vaš CPU, ki omogoča obdelavo blokov z več kot 8 kilobajtov.
FC70/FC71	Ta dva FCS sta velikosti manj kot 8 kilobajtov, zato se lahko naložita na manjše CPU. FC70 uporablja diagnostično funkcionalnost SFC17/18 in se lahko uporabi samo na CPU, da so te funkcije na voljo. Če vaš CPU nima te funkcije, morate uporabljati FC71 in brez diagnostične zmogljivosti. Če želite preveriti, ali vaš CPU vsebuje te SFCs, izberite menijski ukaz **PLK > Obtainable Nodes** v SIMATIC Manager ali pa kliknite ustrezni gumb v orodni vrstici. Odpri mapo "Blocks" v S7 programu.
FC73	Ta blok potrebuje manj kot 8 kilobajtovpomnilnika, tako da se lahko izvaja na vseh CPU. Z uporabo tega FC občutno zmanjša zahteve za pomnilnikFBSS7-GRAPH. Izbrati morate tudi možnost "Interface Description: Memory minimized" v nastavitvah bloka. Naslednje omejitve, do, however, apply: ustvarjeni bloki nimajo zmogljivosti diagnosticiranja. Ko opazuješ sekvenčni nadzorni sistem, boš videl prikaz stanja le za izbrane aktivne elemente.

Nastavitev izvedljivosti:

Izberite ukaz v meniju Možnosti> Blokiraj Nastavitve in določite, da se S7 GRAPH-FB lahko izvede s standardno FC številko, pri prevajanju/save kartici.
Vpišite število FC-ja, ki se ujema z učinkovitostjo vašega CPU-ja.

Blok samodejno kopira uporabnikov projekt, če izberete FC70/71, FC72 ali FC73 kot blok število, toda to še ni FC s to številko v ciljnem projektu.
Če želite uporabiti drugo številko za standard blok, ga morate prekopirati in se preštevilčiti.

Shranjevanje in zapiranje sekvenčnika:

Ko shranite sekvenčnik, se ti samodejno zbirajo.
1. Izberite ukaz v meniju File> Save.
   Rezultat: "Izberite stopnje DB" pogovorno okno se odpre s privzeto stopnjo DB (DB1).

2. Sprejmite nastavitve s klikom na "OK".
   Rezultat: primer skupine podatkov se samodejno ustvari v "Blocks" mapo.

Opomba!
Opozorilo "S1 je brez vsebine", v prevajalniku dnevnika preprosto (compiler log-a) pomeni, da nihče ne ukrepa in da je programiran v koraku 1.

3. Če želite zapreti snemalnik, izberite v meniju ukaz File> Close.

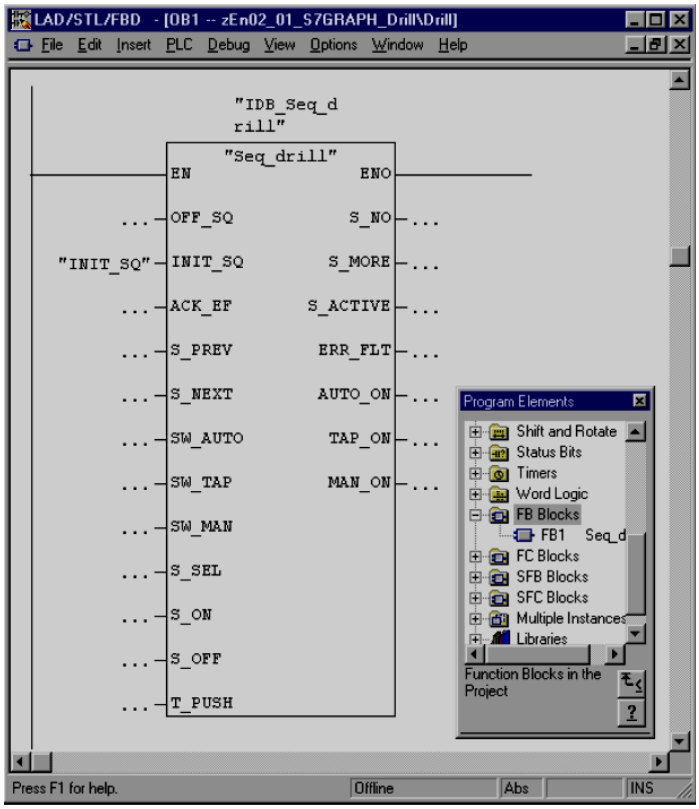
Programiranje OB1:

Zaporedni kontrolni program za urjenje je poklican ter vklopljen v organizacijskem bloku OB1. Lahko ustvarite OB1v FANT, FBD, STL ali SCL (ustvarjeno v LAD).

ProgramskiOB1, kot je prikazan v naslednjem diagramu. Spremljajte korake :
1. Odprite "Blocks "mapa v "Drill Program" S7 program v SIMATIC Managerju.
2. Začnite z LAD/STL/FBD in uredite z dvojnim klikom OB1.

3. Z menijem izberite pogled in nato izberite programski jezik LAD.
4. Izberite segment 1 in vstavite "sekvenčnik klican", uporabite programski elementni katalog z dvojnim klikom FB1 (Seq_drill).
5. Natipkajte ime ustreznega primera podatkovnega bloka (IDB_Seq_urjenje) čez LAD box.
6. Izbran vhodni parameter INIT_SQ vstavite v normalno odprt element uporabljajoč"LAD", izberite orodno vrstico in označite M0.0 ("INIT_SQ"). Uporabite ta parameter, lahko nastavite sekvenčnik k začetnemu koraku (v example step1) v spletni način.
7. Izberite menu >FILE>SAVE in zaprite OB z MENU>FILE>CLOSE.

BELEŽKA:
Vsi drugi bločni parametri so lahko neveljavni za primer.



Slika 89: Programiranje OB1
Vir: Lasten

Nalaganje programa na CPU in testiranje sekvenčnika

Nalaganje uporabniškega programa

Da bi omogočili nalaganje programa na CPU, morate najprej naložiti vse bloke (DB1, FB1, OB1, FC70/71, FC72 in/ali. FC73) na CPU programirljivega logičnega krmilnika v SIMATIC Managerju. Sedaj sledite spodaj opisanim korakom:

1. Odprite "Drill Program" S7 program v SIMATIC Managerju in izberite mapo "Blocks".
2. Izberite ukazni menu **PLK > Download.**

Pozor!
Najboljše, da S7-GRAPH blok naložite v načinu STOP, kajti v nasprotnem primeru se sekvenčnik nastavi na svoje začetno stanje.
S7-GRAPH blok bi moral nalagati samo v RUN-P načinu, kadar je sekvenčnik v prvotnem načinu ali načinu OFF. Če nalagate bloke na sekvenčnik v drugi državi, ali blok ponovno nalagate, se lahko pojavijo težave pri sinhronizaciji sekvenčnika s procesom.

Testiranje uporabniškega programa
Za preizkus uporabniškega programa, potrebujete internetno povezavo s CPU.
1. Odprite projektno okno v SIMATIC Managerju.
2. Odprite sekvenčnik z dvojnim klikom na FB1.
3. V meniju izberite ukaz **Debug > Monitor.**

Rezultat: Status programa je prikazan (prvi korak je aktiven). Aktivni koraki so nato prikazani z barvo.

Pozor!
Spremljanje časa je sprogramirano v drugem koraku. Če aktivni korak presega konfiguriran čas spremljevalnega časa (500ms) v nadzornem stanju, sistem prepozna napako in motnja koraka je označena z rdečo barvo. Če do te okvare pride, morate najprej izpolnjevati vse pogoje, ki se zahtevajo za izvršitev tega koraka, šele nato se bo izvedel naslednji korak. Z uporabo PG funkcije **Debug > Control Sequencer** lahko vnesemo potrdilo (glej tudi "Control Sequencer").
To ne velja za inčni način, saj je treba korak, ki omogoča, da je pogoj izpolnjen, potrditi v enem ciklu.

Testna funkcija: Control Sequencer

Control Sequencer je testna funkcija, s katero lahko testirate sekvenčnik in S7-GRAPH v vseh načinih. Vse nastavitve in vpisi za Dialog box imajo enak učinek kot ustrezni FB parametri.
Vpisi v ″Control Sequencer″ Dialog box so lahko drugačni kot nastavitve, ki jih uporabite za urejanje sekvenčnika. Nastavitve Dialog box-a imajo prioriteto.

″Control Sequencer″ Dialog box

″Control Sequencer″ Dialog box uporabljamo na dva načina, kot izhodni prikaz nastavitev na zaslonu (display) in kot vhodni način, kjer lahko spremenite trenutno stanje. Če zasledite napako, inicializirate sekvenčnik, če pa želite spremeniti korak na ročni način, greste v dialog box pod ukaz DEBUG > CONTROL SEQUENCER.

Acknowledge

Če je potrjena opcija ″Acknowledge errors″, morate napako potrditi s pritiskom na tipko ″Acknowledge″. Po tej poti lahko potrdite motnje, ki so nastale –npr. nastavljeni čas je bil presežen v Step 2.
Preden potrdite napako, se morate prepričati, da nadzor in/ali blokada (napaka) nista več izpolnjena.

Ko pride do napake, lahko omogočite naslednji korak v sekvenčniku, s katerim izpolnimo korak, ki izpolnjuje pogoj, ki ga omogoči zadnji korak v ciklu, v katerem ima prioriteto pred nadzorom. Napake morate vseeno potrditi.
Če je napaka povzročena z nemotenim časom aktivacije koraka, lahko preseženi nastavljeni čas na sekvenčniku spremenimo s klikom na ″acknowledge″ (potrditev).
To je mogoče zato, ker je Step name spremenljivka, nastavljena na ″0″, ko jo potrdite.

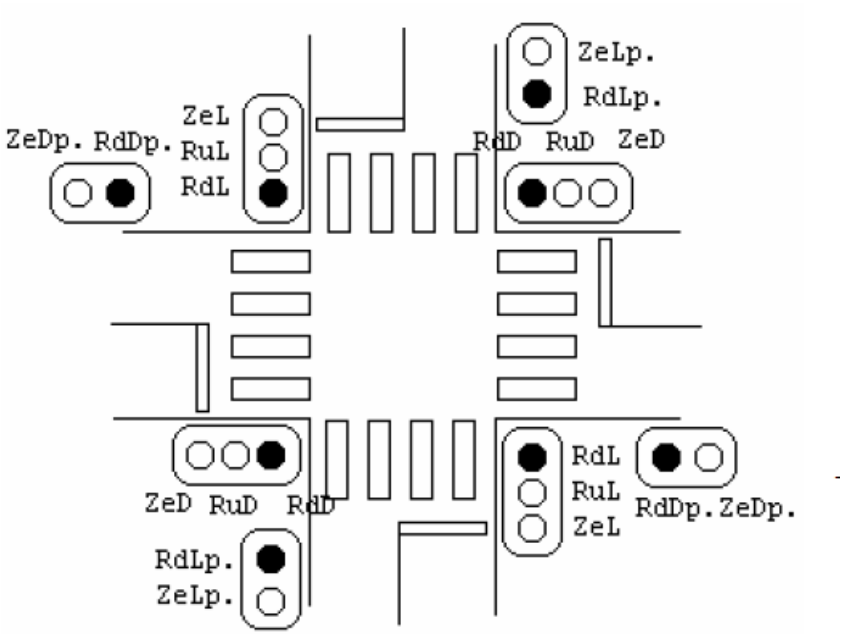
Inicializiraj

Z ″Initialize″gumbom lahko resetirate sekvenčnik z določenim začetnim korakom.

Povzetek:
Grafični način programiranja omogoča lažjo predstavitev problema, saj programiranje poteka v obliki korakov. Pri vsakem koraku imamo pogoje, ki morajo biti izvedeni, in akcije, ki se tvorijo, če so vsi pogoji izpolnjeni. Programiranje v grafičnem načinu omogoča tvorjenje podprogramov in pogojne skoke. Glavni program pišemo v funkcijskem bloku, ki ga nato kličemo v glavnem bloku OB1.

Vprašanja:

1. Na primeru razložite oznake S, R, D, N.
2. V katere sekcije je definirano celotno vrtalno zaporedje?
3. Navedite prednosti in slabosti programiranja s programskim orodjem GRAPH.
4. Opišite postopek programiranja sekvenčnika po direktni metodi.
5. Opišite postopek programiranja nadzorne funkcije.
6. Kako lahko zmanjšate zahteve za pomnilnik s S7-GRAPH-om?
7. Na spodnji sliki je prikazano semaforizirano križišče. S programskim orodjem S7 GRAPH avtomatizirajte delovanje križišča.



8 PROGRAMI ZA VIZUALIZACIJO PROCESOV – SCADA

Nadzor proizvodnih procesov je naloga SCADA sistemov, hkrati pa so po CIM shemi podrejeni MES sistemom. SCADA sistemi nadzorujejo delovanje krmilnih enot in preko tega nadzora vplivajo na proizvodni proces. Naloga SCADA sistemov so nadzor, spremljava in vodenje proizvodnje. SCADA sistemi omogočajo veliko odprtost sistema, saj se lahko podatki zajemajo iz različnih virov, ustrezno obdelajo in nato koristijo kot navodila za izvedbo proizvodnega procesa.

Sodobni SCADA sistemi imajo možnost povezave z raznovrstno programsko opremo. Te povezave omogočajo prenose najrazličnejših podatkov, ki so potrebni za delovanje. Ta odprtost sistemov je posledica dejstva, da vsa sodobna avtomatizacija teži k vse večji povezavi sistemov med seboj.

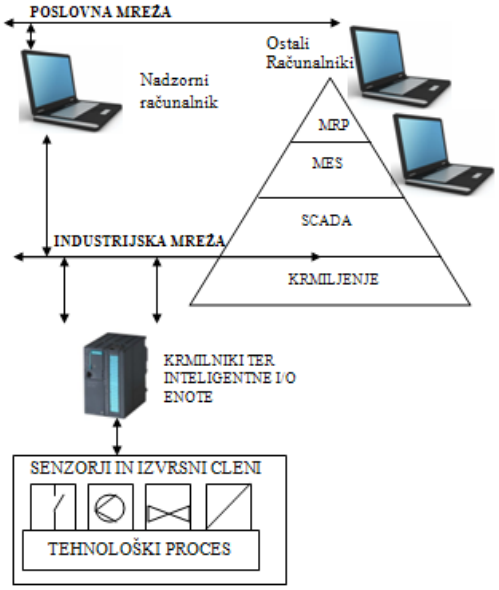
Simatic WinCC je namenski programski paket, s katerim se projektirajo Siemensovi nadzorni sistemi v okolju Windows. Namenjen je uporabi vizualizacije procesa in za razvijanje uporabniškega vmesnika za operaterje. Uporabniki lahko opazujejo grafično prikazane procese, ki se osvežijo takoj, ko pride do spremembe stanja sistema. Sestavljen je iz kar nekaj orodij, med katerimi so pomembnejši: računalnik upravljalec procesnih spremenljivk, grafični urejevalnik, arhiviranje procesnih spremenljivk in globalna skripta.

Glavni namen pri izdelavi nadzornega sistema je uporaba že obstoječe programske in strojne opreme. Na računalniku lahko istočasno obratuje samo en projekt. Pri povezavi s strojno opremo je edina potrebna nadgradnja povezava računalnika in krmilnika, vendar njen potek še ni točno določen. Ko ste povezani s krmilnikom, morate vpisati procesne spremenljivke v bazo, ko pa vnašate procesne spremenljivke, imate na voljo različne podatkovne tipe spremenljivk. V grafičnem oblikovalcu izdelate zaslonske slike, pri arhiviranju procesnih spremenljivk pa lahko določite arhivske procesne spremenljivke. Na slikah lahko uporabite tudi različne tipe dinamik ter večino podatkov prikažete z dinamičnimi povezavami.

8.1 SCADA SISTEM

SCADA so namenska programska orodja, ki so namenjena zajemanju in obdelavi podatkov. SCADA je kratica za angleške izraze *Supervision, Control, Alarm DataAcquisitions*, kar pomeni nadzor, kontrolne, alarmirne podatkovne enačbe. Že iz imena sledi, da gre za sisteme, ki združujejo v sebi več funkcij:

- zajemanje podatkov,
- kontrola podatkov,
- odločanje o obnašanju sistema.



Slika 90: SCADA sistem
Vir: Lasten

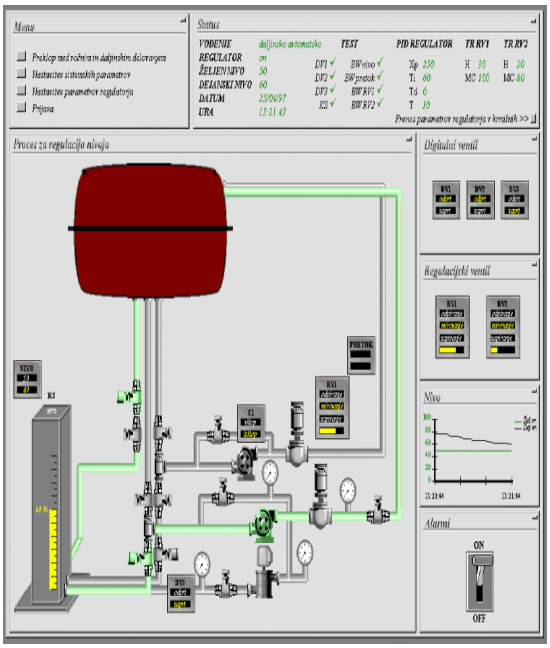
SCADA sistemi so logično nadaljevanje razvoja avtomatizacije, ki teži k vse večji uporabi sodobne računalniške tehnike. Že z uvedbo prosto programirljivih enot (PLK-jev) v avtomatizacijo je nastala prava revolucija, z uvajanjem SCADA sistemov pa se ta razvoj nadaljuje.

SCADA sistemi so povezali faze krmiljenja in odločanja v en sistem. Pred temi sistemi je nadzor procesa baziral izključno na krmilnih enotah. Krmilne enote so zajemale podatke, izvajale odločitve na osnovi teh podatkov te tako krmilile proces.

Prednosti splošno uporabnih SCADA sistemov so:

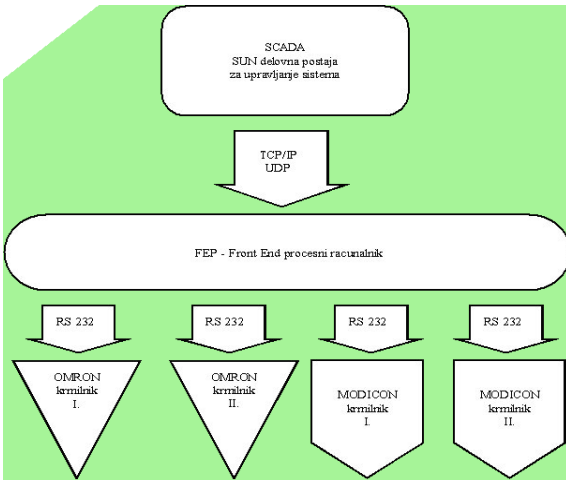
- neodvisnost od krmilne opreme,
- sistemi so primerni za najrazličnejše vrste nalog,
- prenosljivost sistemov,
- v nekaterih primerih neodvisnost od tipa računalniške opreme in operacijskega sistema,
- razširjenost sistemov po vsem svetu,
- izredno hiter razvoj novih verzij sistemov.

Tipični predstavniki splošno uporabnih SCADA sistemov so InTouch, Fix, Factory Link itd. Intouch in Fix sta SCADA sistema za PC računalnike in skupaj pokrivata približno 70 % svetovnega trga SCADA aplikacij na PC računalnikih (http://www.automation.siemens.com/).



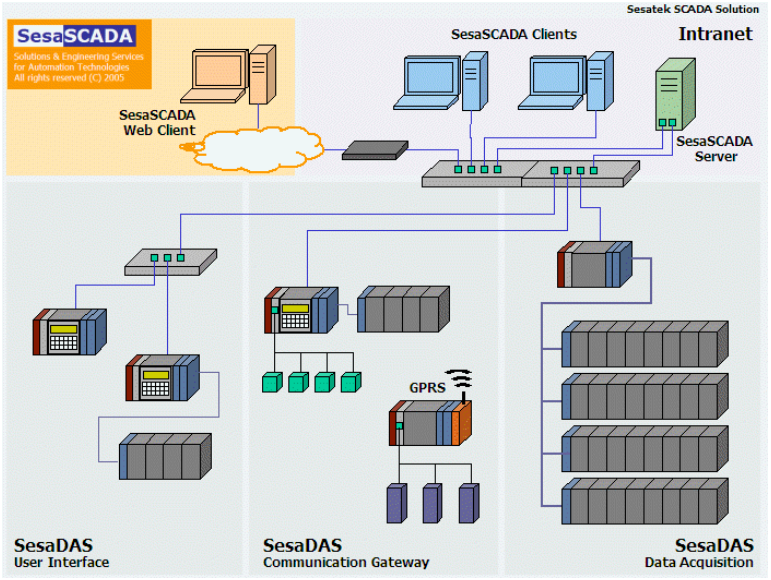
Slika 91: Grafični prikaz SCADE
Vir: Lasten

Fizična povezava krmilne opreme SCADA sistemov je pogojena z vrsto vmesnika, ki ga podpirajo krmilniki računalniške opreme. Programsko pa je komunikacija s krmilniki pri večini SCADA sistemov izvedena preko posebnih programskih vmesnikov, t.i. gonilnikov. Gonilniki služijo za izmenjavo podatkov med krmilniki in SCADA sistemi. Vsak krmilnik pozna svoj gonilnik. Gonilnike običajno izdeluje proizvajalec krmilnika ali proizvajalec SCADA sistema, možno pa je tudi dobiti gonilnik od neodvisnega izdelovalca.



Slika 92: Blok shema procesov
Vir: Lasten

Slika na levi prikazuje Blok shemo procesov za komunikacijo s SCADA sistemom in PLK krmilniki, ki tečejo na računalniku protokolnega pretvornika.



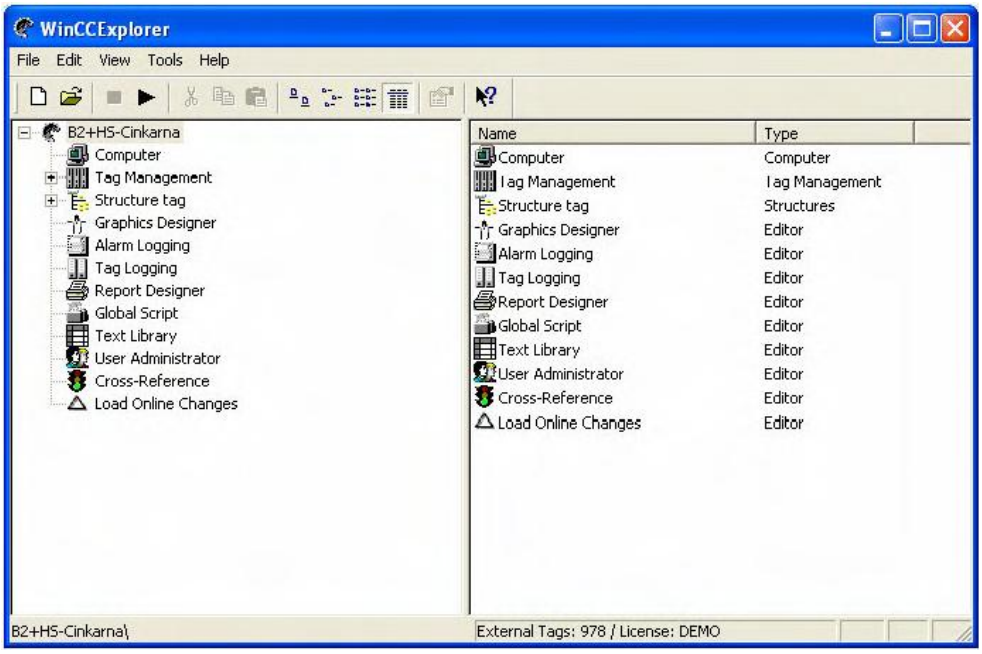
Slika 93: Prikaz delovanja SCADE
Vir: http://www.sielcosistemi.com/en/products/

8.2 OPIS PROGRAMSKE OPREME SIMATIC WINCC FLEXIBLE

Simatic WinCC je namenski programski paket, s katerim se projektirajo Siemensovi nadzorni sistemi v okolju Windows. Nahaja se v sklopu programskega paketa Simatic Manager, to pa pomeni, da se skupaj z ostalim projektom shranijo tudi podatki in slike. Enaka programska oprema za vse prikazovalne programe je njegova največja prednost. Komunikacija poteka med WinCC in uporabnikom na eni strani, na drugi pa med WinCC in sistemom za avtomatizacijo.

Namenjena je uporabi vizualizacije procesa in za razvijanje uporabniškega vmesnika za operaterje. Uporabniki lahko opazujejo grafično prikazane procese, ki se osvežijo takoj, ko pride do spremembe stanja sistema.
Če projektirate s pomočjo programskega paketa WinCC, lahko ustvarite slike in sporočila ter jih povežete s krmilnikom. Sestavljen je iz kar nekaj orodij, med katerimi so pomembnejši:

- Računalnik *(Computer)*: osnovne nastavitve (zagonska slika, nastavitev jezika, zaklep določenih tipk itd.);
- Upravljalec procesnih spremenljivk *(Tag Management)*: izdelava notranjih procesnih spremenljivk *(Tag)* in zunanjih procesnih spremenljivk (nastavitev komunikacije s krmilnikom …);
- Grafični urejevalnik *(Graphic Designer)*: izdelava slik za vodenje procesa;
- Arhiviranje procesnih spremenljivk *(Tag Logging)*: izdelava zgodovine procesnih spremenljivk;
- globalna skripta (*Global Script*): izdelava funkcij in akcij (*Visual Basic Script in C*).



Slika 94: Raziskovalec WinCC
Vir: Lasten

8.2.1 Računalnik (Computer)

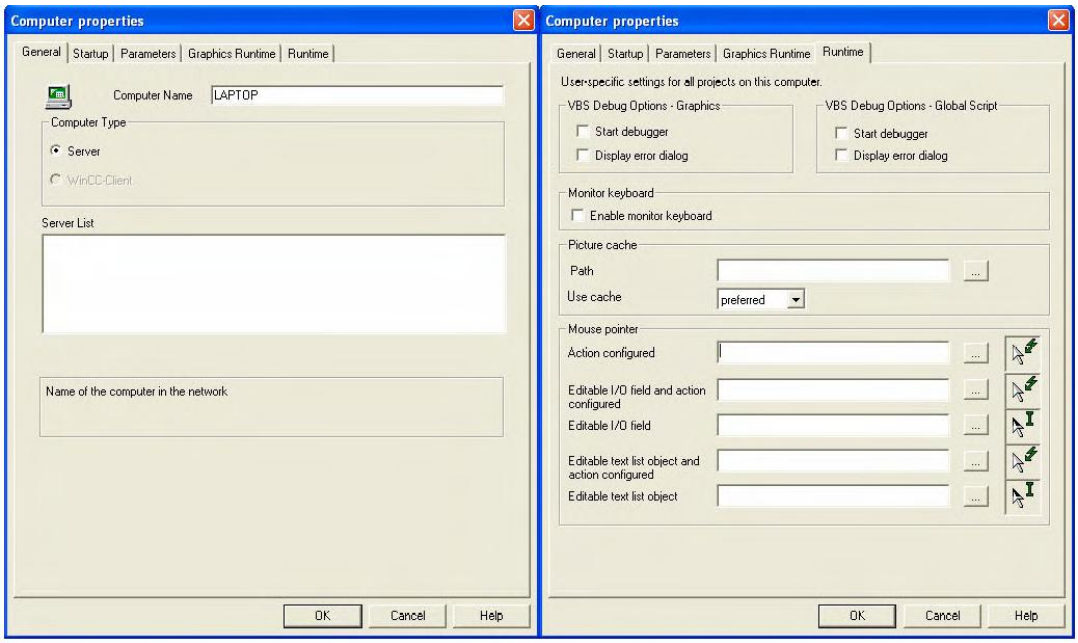
To orodje nam pomaga določiti imena in lastnosti vseh računalnikov, ki bodo vodili določen proces. Če uporabite le en računalnik, določite, da je ta strežnik.

Lastnosti, ki jih lahko določite računalniku so naslednje:
- splošno,
- zagon,
- parametri,
- vizualno izvajanje,
- izvajanje.

Med splošnimi lastnostmi določite ime računalnika – to mora biti enako tistemu, ki ga operacijski sistem Windows uporablja za prijavo v omrežje. Tu izberete tudi, ali je izbrani računalnik strežnik ali odjemalec ter določite tudi seznam vseh strežnikov. Katere komponente se bodo zagnale ob zagonu vizualizacije, določite z zagonskimi lastnostmi.

Parametri določajo jezik vizualizacije, nastavitve časa in blokado določenih kombinacijskih tipk (s katerimi preprečite izhod iz vizualizacije in preklop na drugo okno).

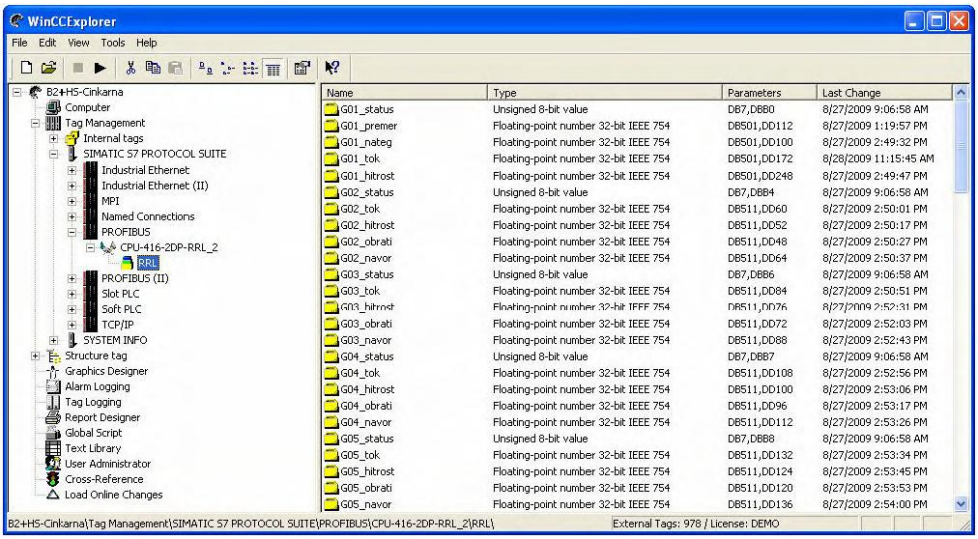
Katera slika naj se prikaže ob zagonu in kako naj se prikaže (celotni način ali prikaz v oknu itd.) določite med lastnostmi vizualnega izvajanja, med lastnostmi izvajanja pa lahko omogočite razhroščevanje ter določitev miškinega kazalca.



Slika 95: Splošne in izvajane lastnosti računalnika
Vir: Lasten

8.2.2 Upravljalec procesnih spremenljivk (Tag Management)

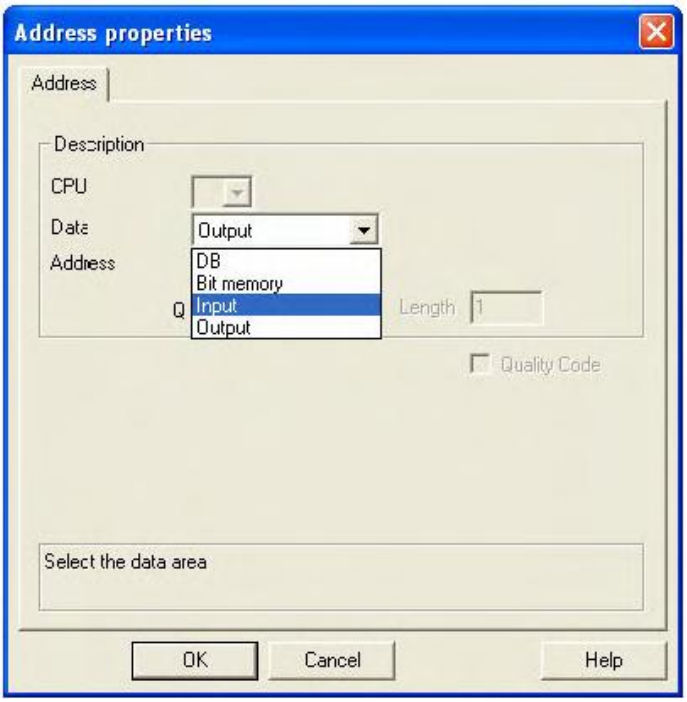
To orodje je namenjeno povezavi računalnika s krmilnikom, izdelavi notranjih in izhodnih procesnih spremenljivk.



Slika 96: Upravljalec procesnih spremenljivk
Vir: Lasten

Da povežete računalnik s krmilnikom, morate najprej dodati gonilnik za povezavo preko določenega protokola. Če sta računalnik in krmilnik povezana preko protokola PROFIBUS, dodate gonilnik za PROFIBUS. Nato temu gonilniku določite povezavo, povezavi določite vse parametre glede na nastavitve krmilnika (naslov postaje, številka držala za krmilnik in module, številka reže, kjer se nahaja modul z vmesnikom, preko katerega povezujete krmilnik in računalnik).

Skupine procesnih spremenljivk lahko ustvarite tako med notranjimi kot tudi pri vsaki povezavi zunanjih procesnih spremenljivk, vendar skupine v skupini ni možno narediti. Zunanje procesne spremenljivke lahko dodate neposredno v povezavo ali pa v skupino procesnih spremenljivk, ki je ustvarjena v tej povezavi. Sestavljena je iz imena, podatkovnega tipa in naslova v krmilniku. Naslov lahko kaže na podatkovni blok, vhod, izhod ali spominski bit *(Bit Memory).*



Slika 97: Možni naslovi procesnih spremenljivk
Vir: Lasten

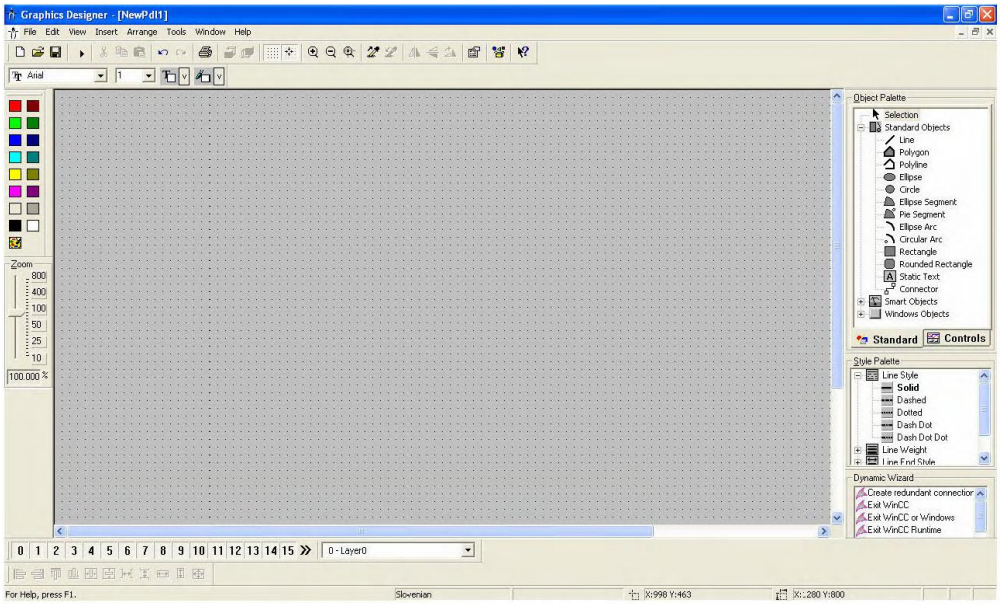
Številko podatkovnega bloka in zlog podatkovnega bloka morate določiti v primeru, če želite, da je zunanja procesna spremenljivka povezana s podatkovnim blokom. Če povezujete vhode, izhode ali spominski bit, je dovolj, če določite le številko zloga, če pa je podatkovni bit binaren, pa le številko bita.

Na računalnik, na katerem se izvaja vizualizacija, se shranjujejo notranje procesne spremenljivke, kar pomeni, da gre za neko vrsto notranjih spremenljivk. Lahko jih dodate neposredno med notranje procesne spremenljivke ali pa v skupino, ki ste jo ustvarili med notranjimi procesnimi spremenljivkami. Sestavljen je le iz imena in podatkovnega tipa, saj naslova ne potrebuje, ker WinCC notranjim procesnim spremenljivkam avtomatsko priredi naslove RAM-a v računalniku.

8.2.3 Grafični urejevalnik (Graphics Designer)

Grafični urejevalnik je vrsta orodja, s katerim lahko izdelamo vmesnik človek-stroj (HMI), torej je primeren za izdelavo slik za vizualno vodenje procesa. Slika je sestavljena iz komponent, grafični urejevalnik pa nudi razne grafične elemente oz. osnovne objekte Windowsov, ki se uporabljajo zelo pogosto (gumb, drsnik, opcijsko izbiranje, izbirno polje ipd.) ter razna interaktivna orodja (merilna naprava, digitalna ali analogna ura, orodje za izris

vrednosti procesnih spremenljivk ipd.). Slika je nosilec *(container)* vseh komponent in zato lahko imajo tako slika kot tudi vse komponete, ki so na sliki, svoje lastnosti in dogodke.



Slika 98: Okno programa grafični urejevalnik
Vir: Lasten

Lastnosti in vrednosti so lahko statične ali dinamične. Če želite, da ima neka lastnost statično vrednost, ji priredite konstanto, v nasprotnem primeru, ko želite, da se vrednost lastnosti spreminja, pa jo povežete s procesno spremenljivko.

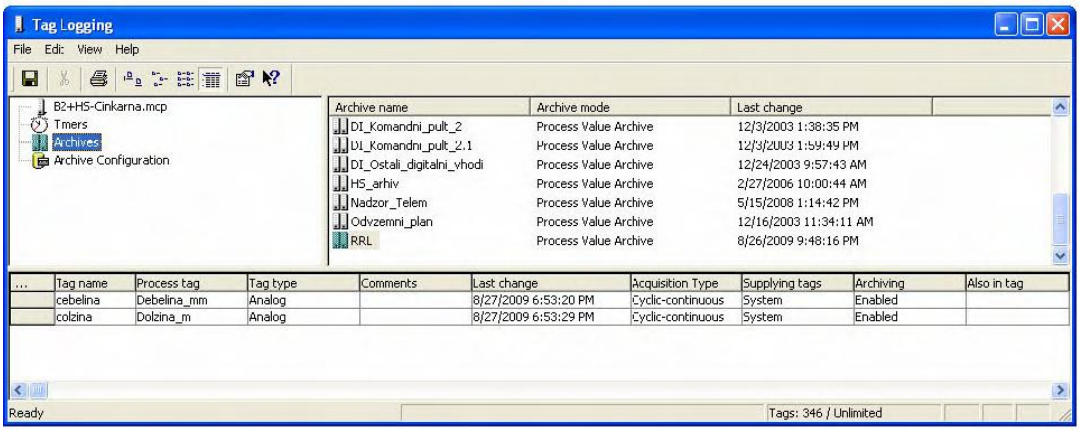
Lastnost lahko s procesno spremenljivko povežete na več načinov:

- z dinamičnim dialogom,
- z neposredno povezavo s procesno spremenljivko,
- z akcijo v C-jeziku ali
- z akcijo v skriptnem jeziku VBS.

Kadar sta podatkovna tipa procesne spremenljivke in lastnosti enaka, procesna spremenljivka pa vsebuje točno tako vrednost, kot jo želimo vpisati v lastnost, uporabite neposredno povezavo s procesno spremenljivko, če pa podatkovna tipa nista enaka, potem uporabite dinamični dialog ali enega od programskih jezikov (akcijo v C-jeziku ali akcijo v skriptnem jeziku VBS).

8.2.4 Arhiviranje procesnih spremenljivk (Tag Logging)

Pri arhiviranju procesnih spremenljivk lahko določite arhivske procesne spremenljivke (Archive Tag) in sicer tako, da izberete procesne spremenljivke, ki se bodo arhivirale, cikel njihovega zajemanja in cikel arhiviranja.



Slika 99: Arhiviranje procesnih spremenljivk
Vir: Lasten

Arhivske procesne spremenljivke se delijo na tri tipe:

- analogna arhivska procesna spremenljivka, ki hrani številske vrednosti procesnih spremenljivk (npr. nivo snovi v silosu);
- binarna arhivska procesna spremenljivka, ki hrani binarno vrednost procesnih spremenljivk (npr. vklop ali izklop sistema);
- procesno nadzorovana arhivska procesna spremenljivka, ki hrani vrednost procesnih spremenljivk, ki so bile poslane sistemu arhiviranja kot blok podatkov, (npr. procesne vrednosti zaporedja meritev).

Arhivske procesne spremenljivke lahko obdelate tudi z matematičnimi funkcijami (npr. povprečje) – obdelate jih preko nekega časovnega intervala, nato pa jih shranite kot stisnjen arhiv vrednosti procesnih spremenljivk.

8.2.5 Globalna skripta (Global Script)

Globalna skripta vsebuje orodji C-Editor in VBS-Editor. Osnovna struktura obeh urejevalnikov je zelo podobna, saj oba vsebujeta akcije, standardne funkcije in funkcije projekta. Urejevalnik za programski jezik C vsebuje še posebno skupino notranjih funkcij kamor spadajo standardne C-funkcije in nekaj dodatnih za delo z WinCC-jem. Visual Basic Script (VBS-Editor) prav tako uporablja notranje funkcije, le da te niso neposredno vidne v posebni skupini funkcij, pač pa gre za standardne funkcije jezika Visual Basic Script, te pa naj bi programer že poznal.

Akcije so funkcije s proženjem. Klicale naj bi ostale funkcije (standardne funkcije ali funkcije projekta), prav tako pa je z njimi mogoče napisati del programske kode v akcijo. Proženje akcije se izvaja s časovnikom ali s spremembo izbrane procesne spremenljivke. Proženje s časovnikom je lahko ciklično ali aciklično. Pri cikličnem proženju izberete periodo, pri acikličnem pa točen čas in datum enkratnega proženja. Proženje glede na spremembo procesne spremenljivke določite tako, da izberete notranjo ali zunanjo procesno spremenljivko ter določite periodo preverjanja spremembe procesne spremenljivke.

Standardne funkcije so lastne funkcije – programer jih napiše z namenom, da se uporabljajo v različnih projektih. Shranjene so v mapi, kjer se nahaja WinCC.

Funkcije projekta so funkcije, ki so posebej napisane za uporabo le v danem projektu in so zato so shranjene v mapi projekta.

C-Editor zažene prevajalnik ob shranjevanju funkcij ter izpiše opozorila in napake po končani pretvorbi. VBS-Editor prevajalnika ne vsebuje, saj se koda sproti izvršuje in pretvarja (kar je običajno za skriptne jezike), za preverjanje sintaktičnih napak pa ima možnost »Syntax check«.

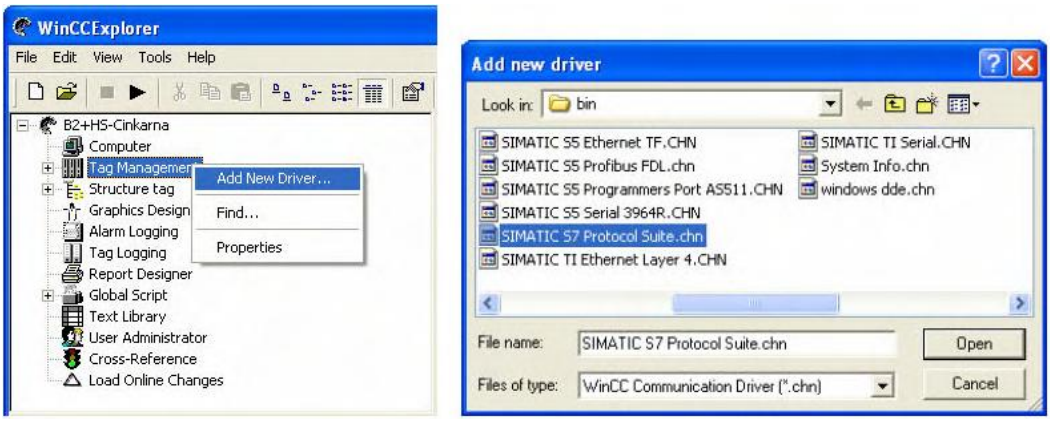
8.3 OPIS IZDELAVE NADZORNEGA SISTEMA

Glavni namen pri izdelavi nadzornega sistema je uporaba že obstoječe programske in strojne opreme. Na računalniku lahko istočasno obratuje samo en projekt. Nadgradnja ni potrebna, saj se vsi parametri, ki jih potrebujete, že nahajajo v krmilniku.

8.3.1 Povezava s strojno opremo

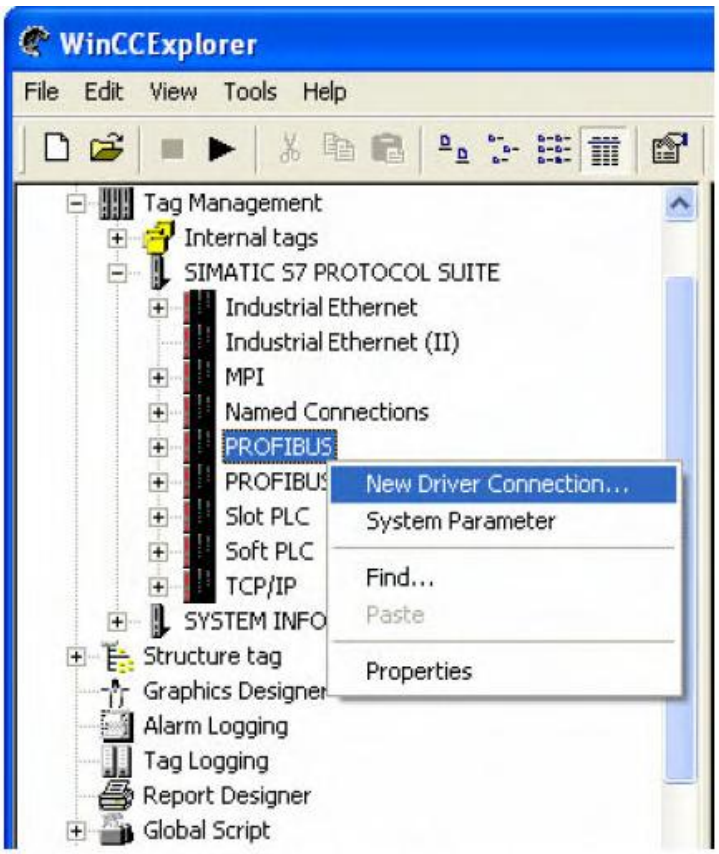
Edina potrebna nadgradnja je povezava računalnika in krmilnika, vendar njen potek še ni točno določen. Za obdobje programiranja in testiranja lahko povezava poteka preko vodila Profibus, za nadaljnjo uporabo pa povezava preko industrijskega etherneta.

Za vzpostavitev povezave s krmilnikom morate naložiti ustrezne gonilnike. V raziskovalcu programa WinCC izberete Tag Management, nato v parametrih izbrete sklop Siemensovih povezav, ki so zbrane pod skupnim imenom SIMATIC S7 Protocol Suite. V okvir te skupine so vključeni naslednji protokoli: Industrial Ethernet, Industrial Ethernet 2, MPI, Named Connections, Profibus, Profibus 2, Slot PLC, Soft PLC in TCP/IP. Slika 100 prikazuje vmesnik za izbiro gonilnikov.

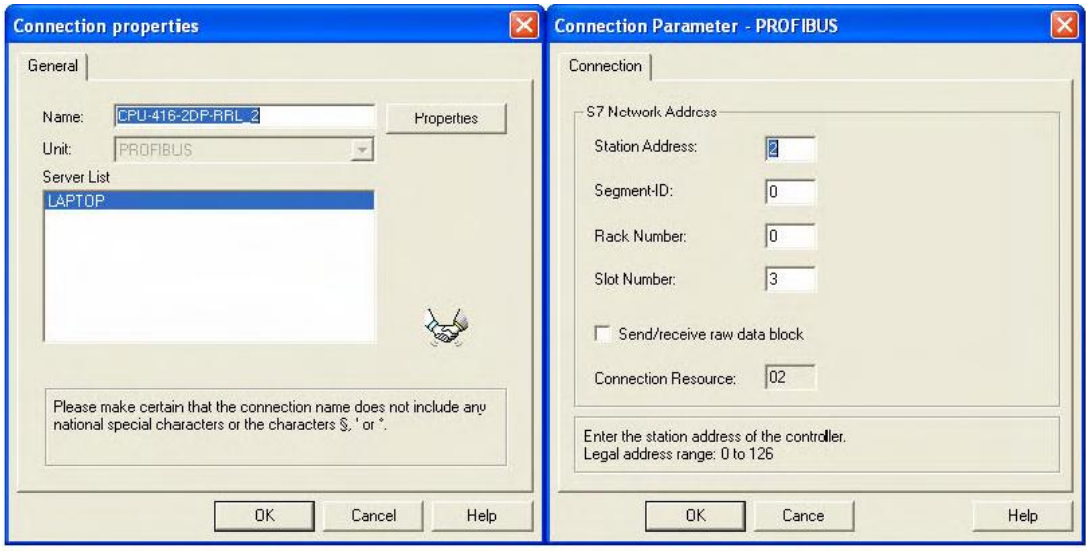


Slika 100: Prikaz izbire gonilnikov
Vir: Lasten

Na protokolu Profibus za gradnjo projekta najprej ustvarite novo povezavo. Nastavitev parametrov v programu WinCC za protokol Profibus je prikazana na slikah 101 in 102.



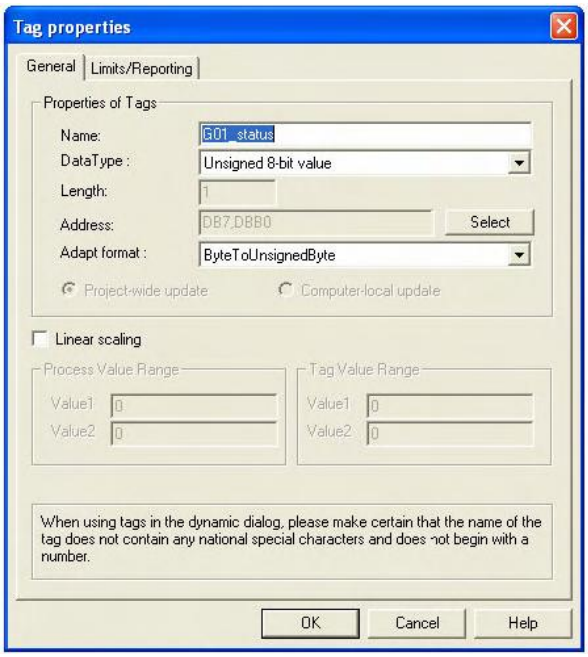
Slika 101: Tvorjenje nove povezave na protokolu Profibus
Vir: Lasten



Slika 102: Prikaz konfiguracije nove povezave na protokolu Profibus
Vir: Lasten

8.3.2 Vnos podatkov v WinCC

Ko ste povezani s krmilnikom, morate vpisati procesne spremenljivke v bazo, ko pa vnašate procesne spremenljivke, imate na voljo naslednje podatkovne tipe spremenljivk: bitna, predznačena in nepredznačena 8-/16-bitna in podatkovni tip RAW. Vse to nastavite v meniju lastnosti procesne spremenljivke (slika 103).



Slika 103: Lastnosti procesne spremenljivke
Vir: Lasten

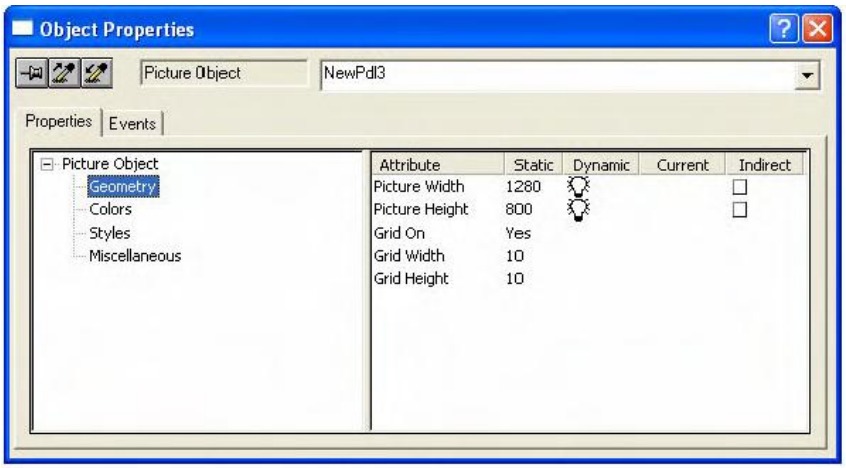
Ko imate določeno ime in tip procesne spremenljivke, morate določiti še, kje se spremenljivka nahaja, in sicer to določite pod poljem Naslov (slika 104).

Slika 104: Lastnosti naslova procesne spremenljivke
Vir: Lasten

Pod poljem Podatki *(Data)* izberete, ali se spremenljivka nahaja v podatkovnem bloku *(DB – data block)*, v spominskem bitu *(bit memory – flag)*, v izhodnih *(output)* ali vhodnih *(input)* vratih. Če je podatek v DB, morate določiti tudi številko DB in naslov, v nasprotnem primeru pa samo lokacijo.

8.3.3 Izdelava zaslonskih slik

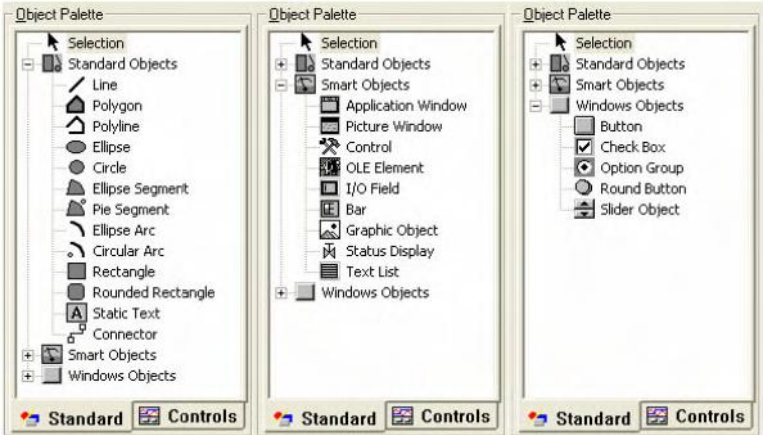
V WinCC-jevem programu Grafični oblikovalec *(Graphics Designer)* izdelate zaslonske slike poteka Začnete s prazno sliko, ki ji morate določiti velikost slike (slika 105). Sliko nastavite v lastnosti objekta pod geometrijo, tam pa lahko nastavite še pomožno mrežo in njene dimenzije.



Slika 105: Grafične nastavitve nove slike
Vir: Lasten

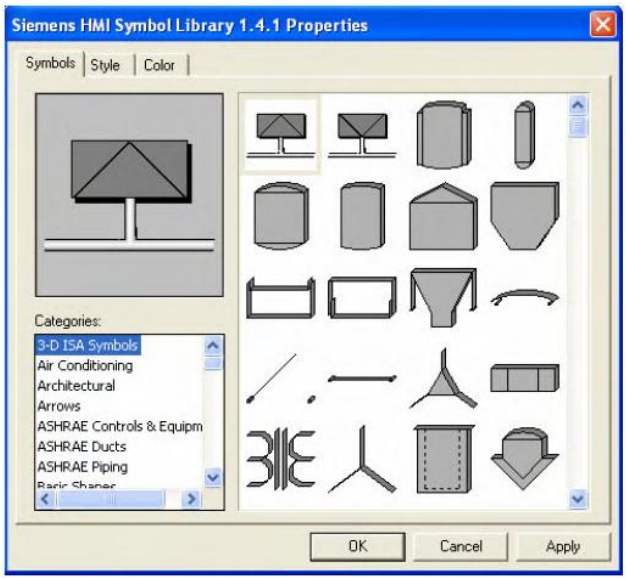
Delo v grafičnem oblikovalcu temelji na objektih, ki si jih ustvarite s paleto objektov (slika 106). Objekti so razdeljeni v tri sklope:

- standardni,
- pametni in
- Windows objekti.



Slika 106: Paleta objektov
Vir: Lasten

S standardnimi objekti lahko kreirate statični tekst in like (črte, krogi, elipse, kvadrati itd.), s pametnimi pa kreirate objekte, ki se lahko spreminjajo glede na določeno vrednost. Pod Windows objekti se nahajajo razni gumbi, ki vas bodo spominjali na Windowse. Obstaja tudi knjižnica z že izdelanimi objekti, ki jih lahko uporabite (slika 107).

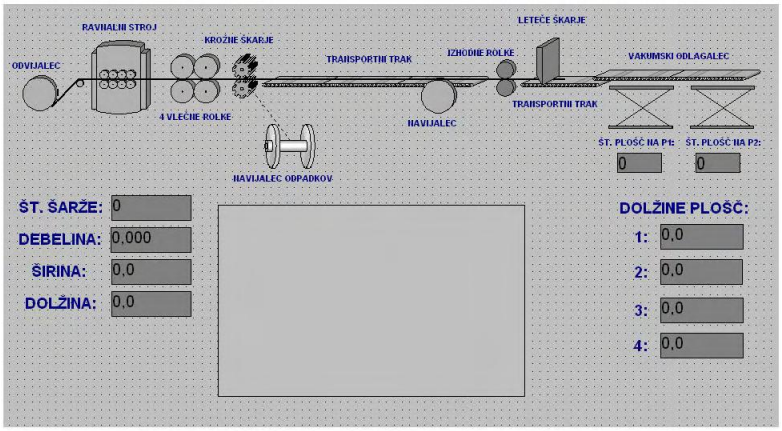


Slika 107: Knjižnica vnaprej izdelanih objektov
Vir: Lasten

Kreirani objekti ležijo na različnih plasteh slike (slika 108). Te določajo, v kakšnem vrstnem redu so objekti vidni od zgornje do spodnje plasti. Objekte na nižji plasti prekrivajo objekti na višji plasti, lahko pa skrijete eno ali več plasti, da slike lažje narišete.

Slika 108: Nastavitev plasti
Vir: Lasten

S pomočjo narisanih objektov in objektov iz knjižnice kreirate sliko projekta (slika 109).

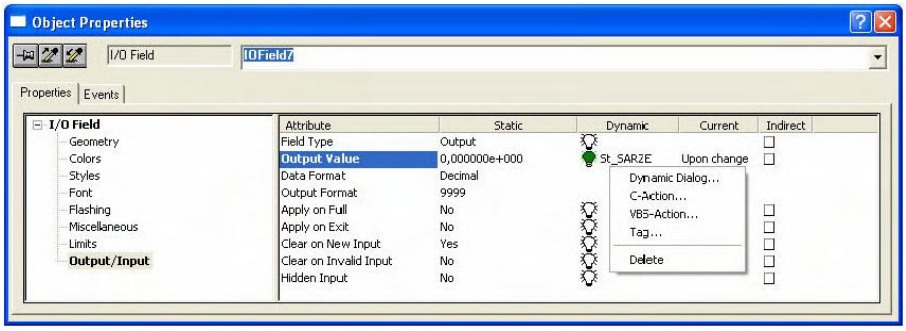


Slika 109: Slika projekta v grafičnem oblikovalcu
Vir: Lasten

8.3.4 Uporaba dinamike na slikah

Slika je ustvarjena, a je še statična, zato morate za vsak objekt nastaviti vse lastnosti (barve, stil pisave, geometrijo, povezave s procesnimi spremenljivkami …) in dogodke (akcije ob klikih z miško na objekt, pritiskih na tipke, dobljenim in izgubljenim fokusom …). Tako lastnosti kot dogodki so lahko odvisni od stanja procesnih spremenljivk ali od stanja drugih objektov.

Do okna Lastnosti objekta (slika 110) pridete z desnim klikom na objekt in izbiro Lastnosti (Properties). Stolpec Dinamika (Dynamic) vam pove, kakšno vrsto dinamične povezave ima določeno lastnost.

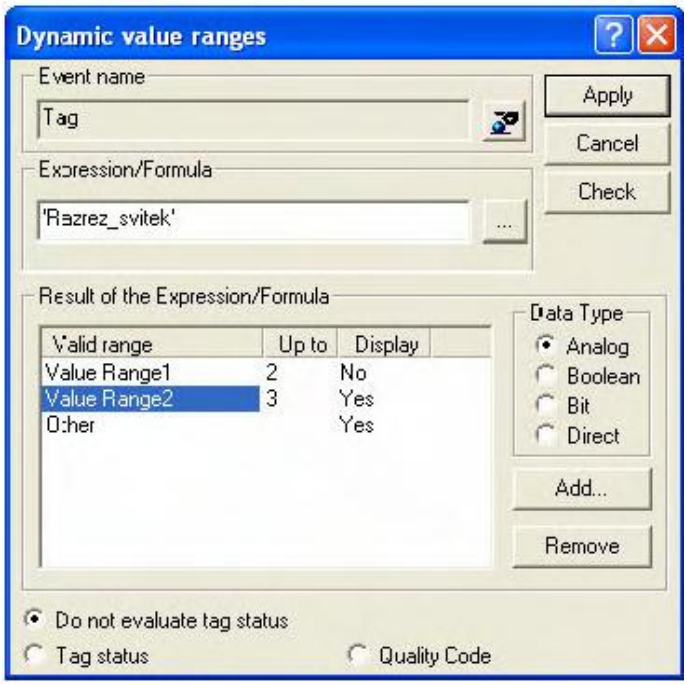


Slika 110: Možnosti dinamike lastnosti objekta
Vir: Lasten

Na voljo so štiri tipe dinamike:

- dinamični razpon,
- program v jeziku C,
- program v jeziku VBS,
- procesna spremenljivka.

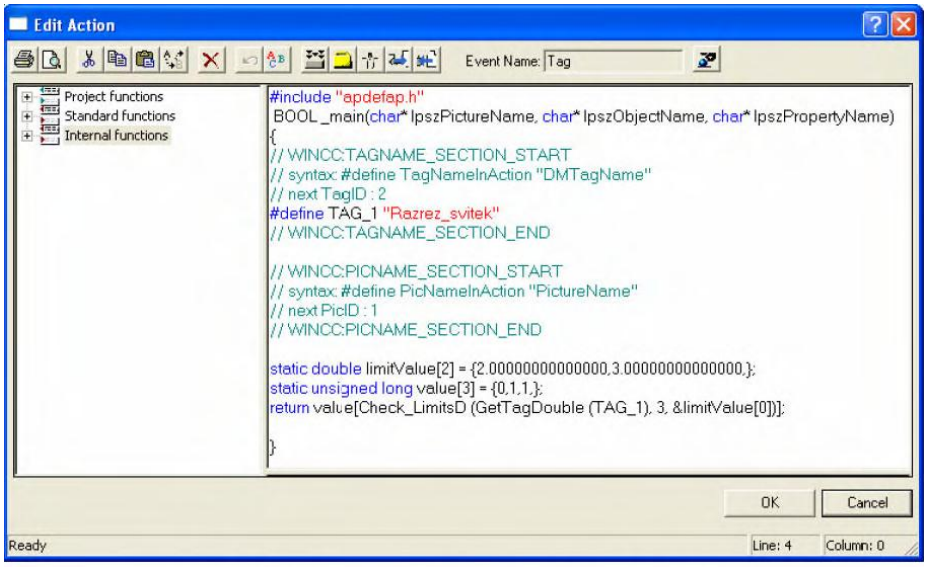
Pri dinamičnem razponu (slika 111) lahko spreminjate lastnosti objeta preko izraza oz. formule. Glede na prožilni signal (polje *Event*), ki je lahko procesna spremenljivka, standardni cikel, cikel slike ali cikel okna, se med delovanjem izračuna nova vrednost lastnosti iz izraza ali formule (v polju *Expression/Formula*). V odvisnosti od velikosti rezultata se nova vrednost postavi glede na postavitveno tabelo (v polju *Result of the Expression/Formula*). Oblika te tabele je odvisna od podatkovnega tipa (določenega v polju *Data Type*) rezultata.



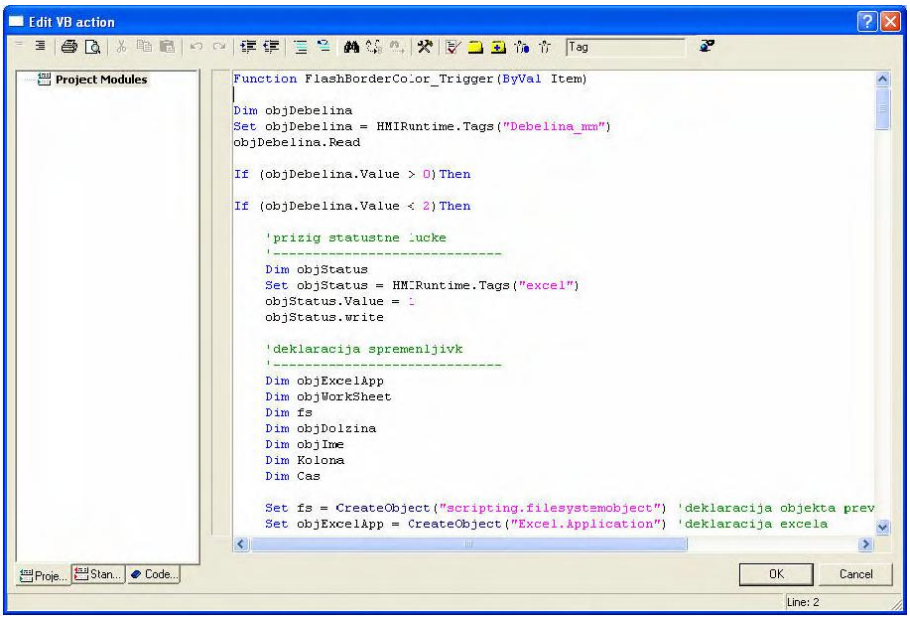
Slika 111: Okno dinamičnega razpona
Vir: Lasten

Naslednja dinamika (slika 112) uporablja programski jezik C. S to dinamiko lahko ustvarjate lastnosti objektov pa tudi za reakcije ob dogodkih nad objektom. Uporablja se najbolj standardna različica programskega jezika C: ANSI-C izbirate pa lahko med že vnaprej pripravljenimi standardnimi funkcijami, povsem proste roke pa imate tudi pri pisanju kode. Zavedati se morate, da je hitrost izvajanja med delovanjem nižja kot pri drugih tipih dinamike.

Tudi VBS-skripta se tako kot C-skripta (slika 113) lahko uporablja tako za ustvarjanje lastnosti objektov kot tudi za reakcije ob dogodkih nad objektom. Uporabljen skriptni jezik je Visual Basic in tudi tu so na voljo že vnaprej pripravljeni moduli.

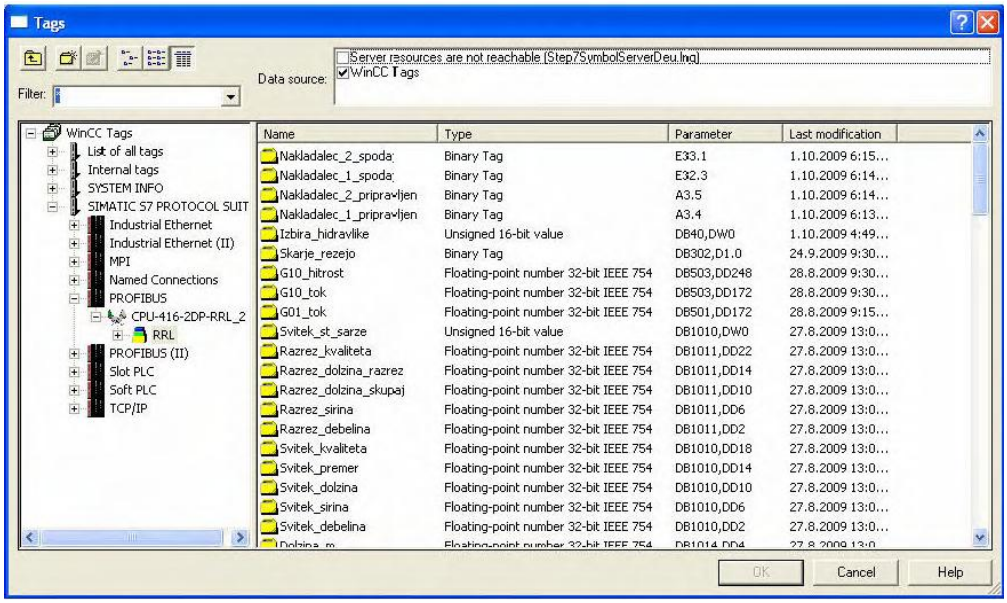


Slika 112: Okno za vpis skript v programskem jeziku C
Vir: Lasten



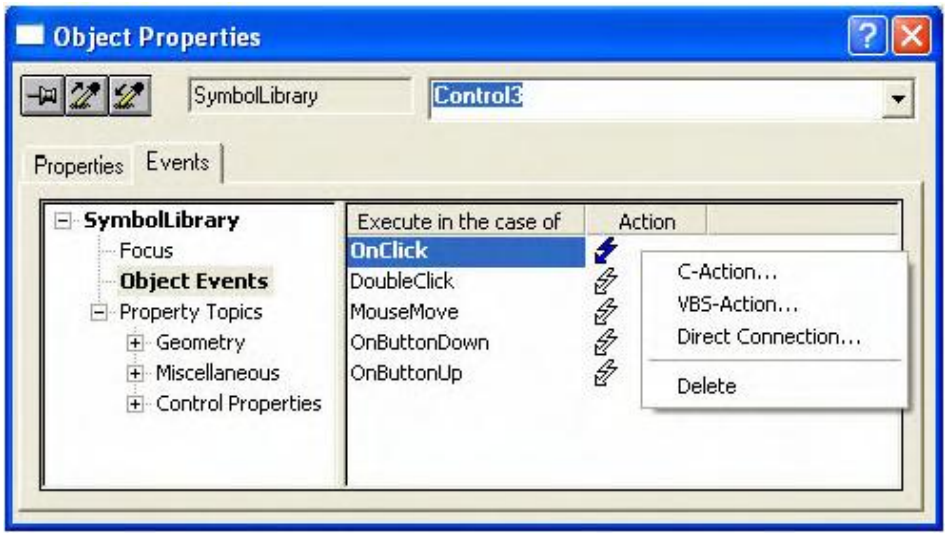
Slika 113: Okno za vpis skript v programskem jeziku VBS
Vir: Lasten

Lastnosti objektov neposredno glede na vrednost spremenljivke lahko spreminjate s povezavo s procesno spremenljivko. Ta povezava lahko omogoča grafičen prikaz spremembe izmerjene vrednosti. Izbirate lahko med vsemi procesnimi spremenljivkami, ki so na voljo v projektu – vse procesne spremenljivke, vidne v WinCC-jevem programu Upravljalec procesnih spremenljivk (*Tag Management*). Če je spremenljivk veliko, jih lahko filtrirate glede na poljuben niz znakov (polje *Filter*). V oknu s procesnimi spremenljivkami (na desni strani) vidite pot do procesne spremenljivke ter njegovo ime (*Name*), podatkovni tip (*Type*), naslov (*Parameter*) ter datum zadnje spremembe (*Last Modified*). Na sliki 114 so prikazane procesne spremenljivke z vsemi parametri.



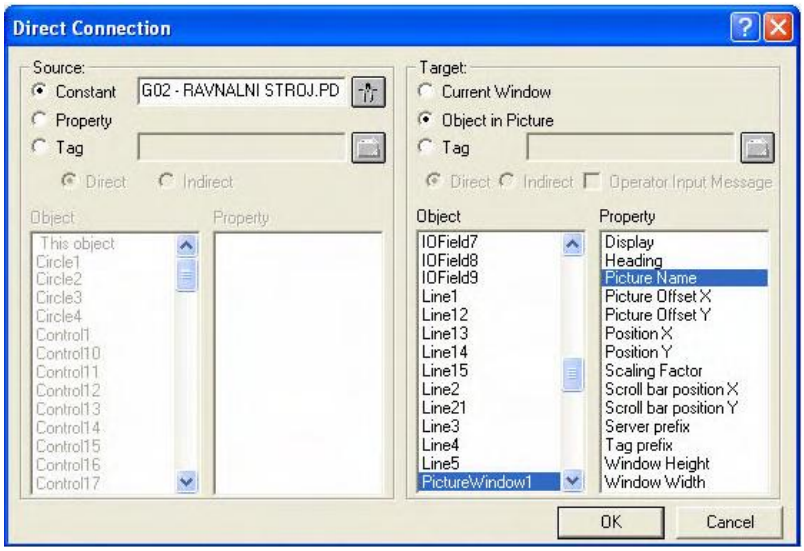
Slika 114: Okno za izbiro procesne spremenljivke
Vir: Lasten

Poleg Lastnosti objekta imate še Dogodke objekta (slika 115), kjer lahko nastavite interakcijo tega objekta. Stolpec Akcija (*Action*) vam pove, kakšna vrsta reakcije se bo izvedla ob določenem dogodku nad objektom. Na voljo so trije tipi reakcij. Prav tako kot pri dinamiki sprememb lastnosti objekta, lahko dobite reakcijo preko skript v programskem jeziku C ali v programskem jeziku VBS – reakcija se tu ne razlikuje, obstaja pa še reakcija preko neposredne povezave.



Slika 115: Možnost reakcij na dogodku nad objektom
Vir: Lasten

Slika 116 nam prikazuje, kako se pri neposredni povezavi v primeru dogodka nad objektom kot reakcija na ta dogodek vrednost izvornega elementa (*Source*) uporabi za ciljni element (*Target*). Kot izvorne elemente lahko uporabite konstante, lastnosti objektov na sliki ali procesne spremenljivke, kot ciljne elemente pa trenutno okno, lastnosti objektov in slik ali procesne spremenljivke.

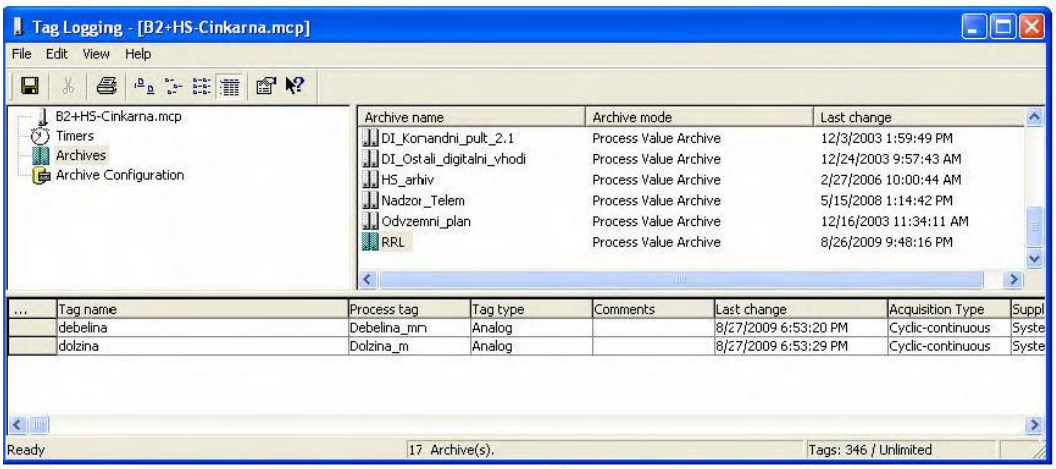


Slika 116: Okno za nastavitev neposredne povezave
Vir: Lasten

8.3.5 Prikaz podatkov

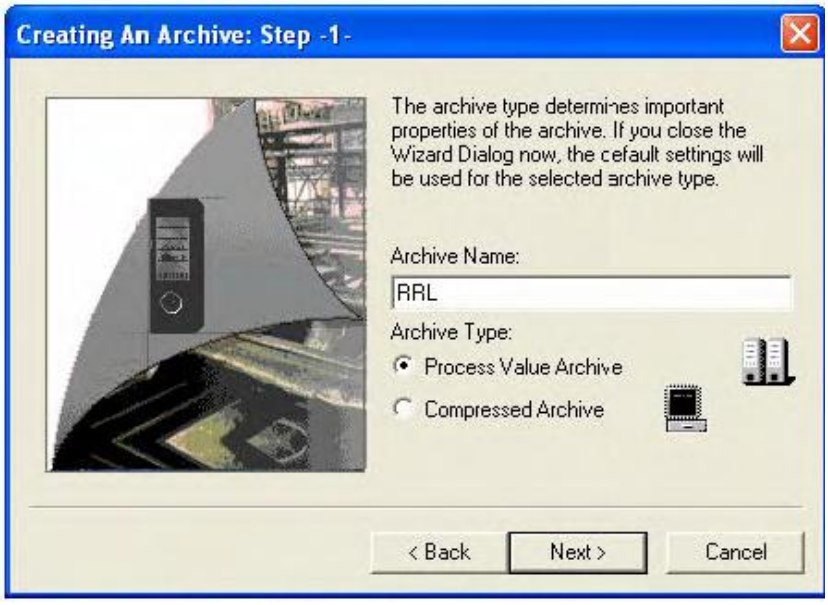
Večino podatkov lahko prikažete z dinamičnimi povezavami, kjer zadostuje samo trenutno stanje podatka, vendar takrat ni razvidno, kako se skozi čas ta vrednost spreminja. Za to nalogo je najbolj primeren graf, za njegov izris pa morate najprej arhivirati podatke.

Za konfiguriranje arhiviranja procesnih spremenljivk uporabite WinCC-jev program Arhiviranja procesnih spremenljivk (*Tag Logging).*

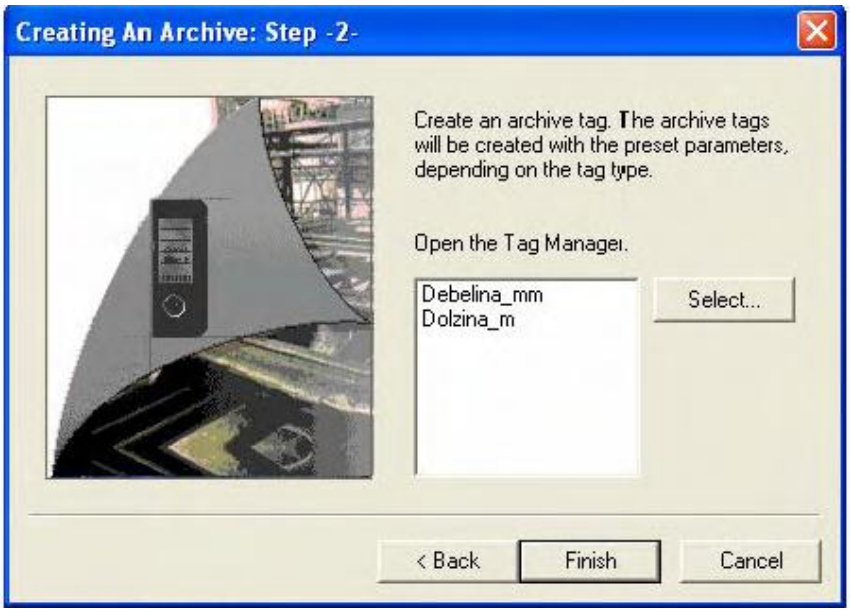


Slika 117: Urejevalnik Arhiviranja procesnih podatkov
Vir: Lasten

Konfiguracija arhiviranja procesnih spremenljivk poteka v treh korakih. Uporabite čarovnika za izdelavo arhiva (slika 118) in s pomočjo njega določite ime in vrsto arhiva, nato (slika 119) določite še, katere vrednosti boste arhivirali.

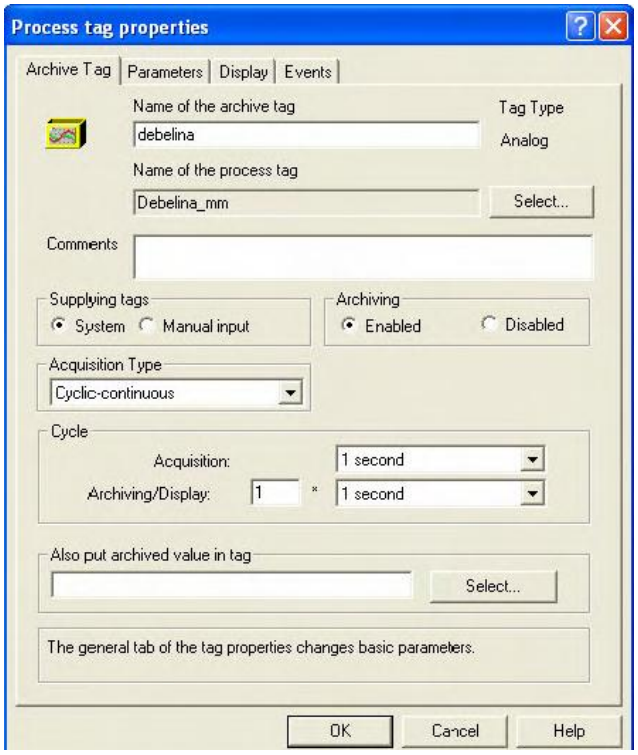


Slika 118: Določitev imena in izbira vrste arhiva
Vir: Lasten



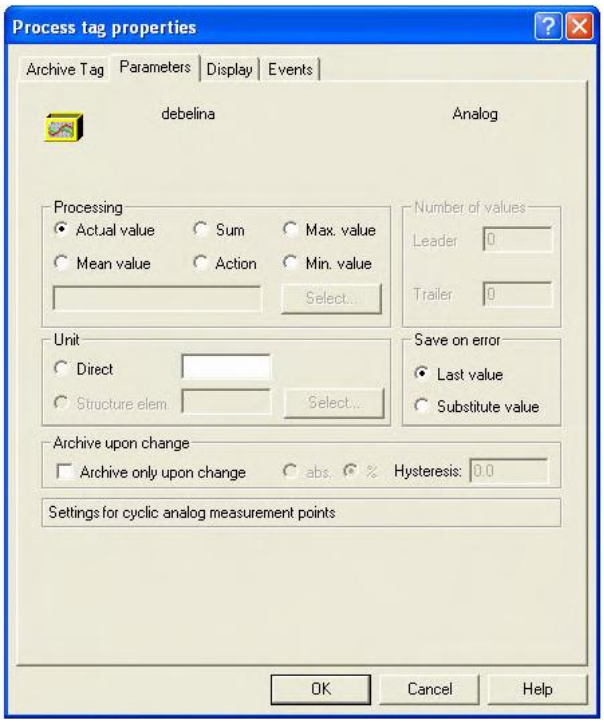
Slika 119: Izbira procesnih spremenljivk, ki jih bomo arhivirali
Vir: Lasten

Parametre posameznih procesnih spremenljivk, ki se nahajajo v tem arhivu, lahko določite, ko imate ustvarjen arhiv. Z desnim klikom nad arhivski procesni spremenljivki v spodnjem delu okna Urejevalnika arhiviranja procesnih podatkov (slika 120) odprete okno Lastnosti arhivskih procesnih spremenljivk in navezavo na procesno spremenljivko, omogočeno ali onemogočeno arhiviranje, nastavitve zajemanja podatkov (ciklično, selektivno ciklično, aciklično ob spremembi) ter cikla zajemanja in arhiviranja v obliki mnogokratnika in njegovega cikla. Npr. če želite, da se procesna spremenljivka arhivira vsakih 15 sekund, nastavite mnogokratnik 15 in za cikel izberete sekundo (15*1 sekunda).



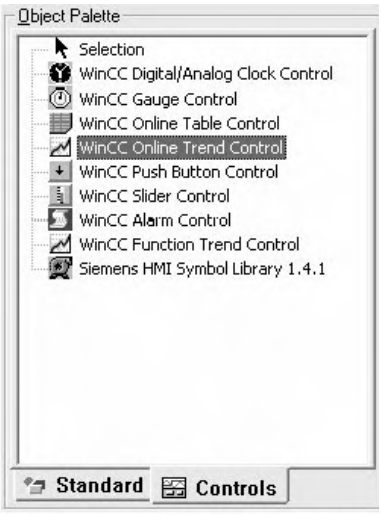
Slika 120: Osnovne lastnosti arhivskih procesnih spremenljivk
Vir: Lasten

Na zavihku parametrov arhivskih procesnih spremenljivk (slika 121) nastavite parametre arhiviranja. V primeru binarnih arhivskih procesnih spremenljivk lahko izberete arhiviranje v primeru spremembe stanja, ob prehodu stanja iz 0 v 1, ob prehodu stanja iz 1 v 0 ali vedno. V primeru analognih arhivskih procesnih spremenljivk pa lahko nastavite predhodno obdelavo vrednosti ter mersko enoto arhiviranih procesnih spremenljivk, lahko pa tudi nastavite arhiviranje samo ob spremembi vrednosti.



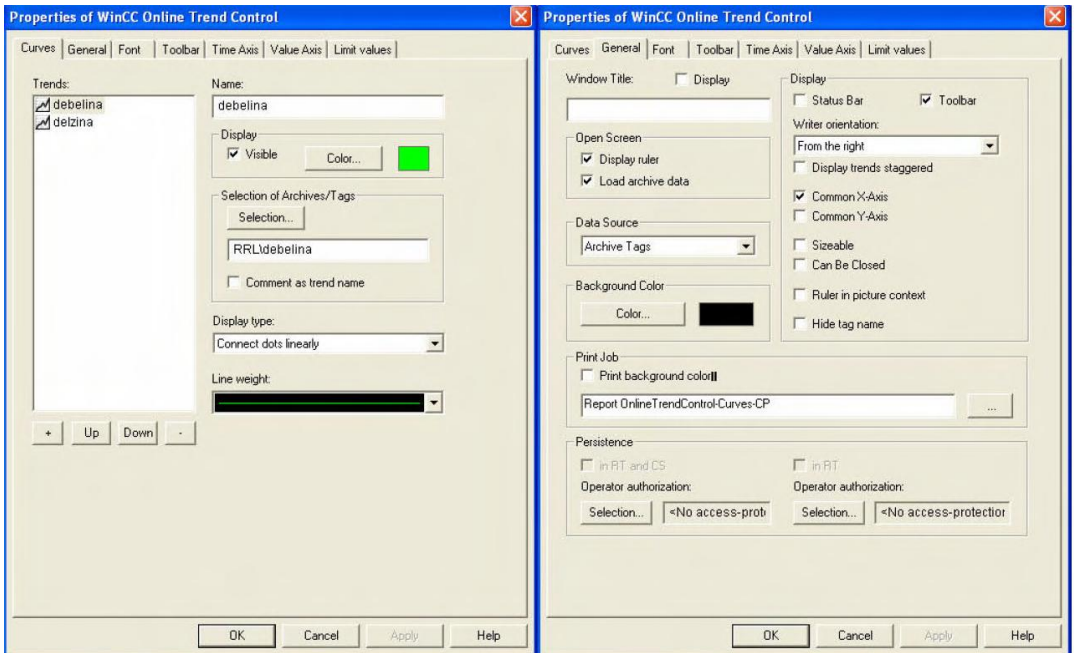
Slika 121: Parametri arhivskih procesnih spremenljivk
Vir: Lasten

Ko imate želene procesne spremenljivke arhivirane, jih lahko uporabite v grafu. Vrnete se v Grafičnega oblikovalca in v kontrolni paleti objektov izberete WinCC Online Trend Control (slika 122).



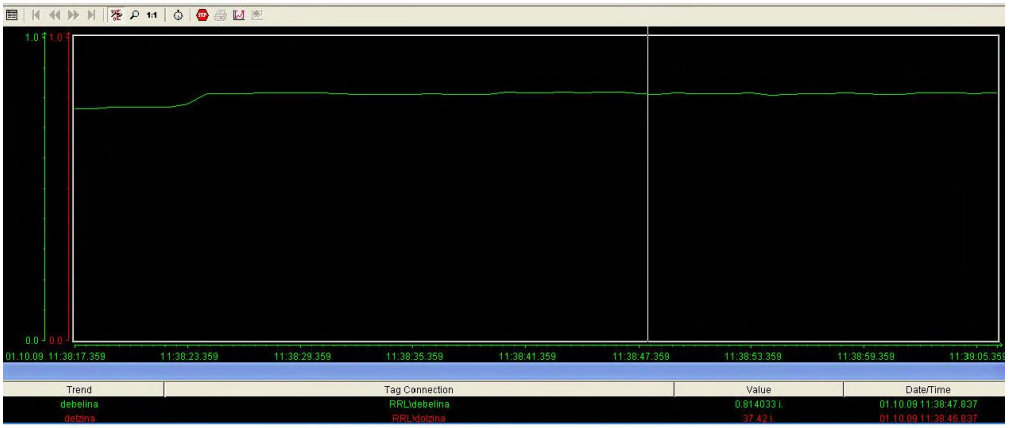
Slika 122: WinCC Online Trend Control
Vir: Lasten

Z izbranim orodjem določite pozicijo in velikost grafa. Ko označite, kje se bo nahajal, se pojavi nastavitveno okno (slika 123), kjer se v zavihku nastavijo osnovni parametri, kot so ime objekta, barva ozadja, prenos arhivskih vrednosti v prikazno sliko, prikaz orodne in statusne vrstice, premik po x in y osi, prikaz procesnih spremenljivk in ostalo. V zavihku Krivulje se nastavijo parametri za izris krivulj. Dodate lahko nove krivulje, ki se razvrstijo po vrsti za prikaz, vsaki krivulji pa se določi, iz katere procesne spremenljivke se prenašajo vrednosti. Lahko izberete tudi barvo in debelino izrisane krivulje. Ko so ti parametri obdelani, jih zaprete. Pojavi se orodje za prikaz toka dogodkov. Če s tipko miške dvakrat kliknemo na objekt, se temu orodju lahko nastavi še več parametrov, a je za osnoven prikaz zadovoljiva že nastavitev naštetih parametrov.



Slika 123: Nastavitev parametrov
Vir: Lasten

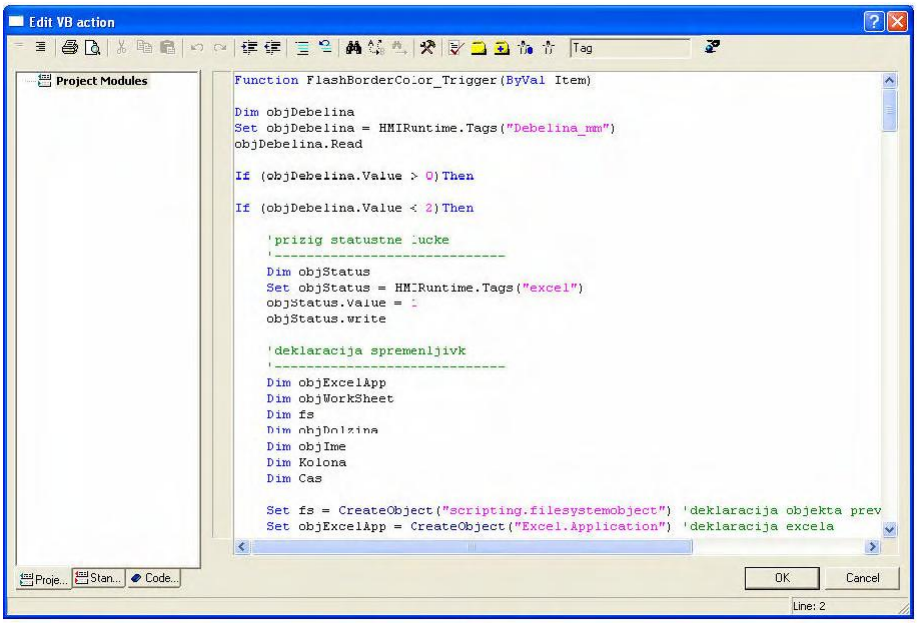
Z nastavljenimi parametri se izriše graf (slika 124). Da lahko odčitate vrednosti, uporabite ravnilo, s katerim se postavite na odsek grafa.



Slika 124: Graf
Vir: Lasten

8.3.6 Program za arhiviranje podatkov

Z WinCC-jevim programom Tag Logging arhivirate podatke. Za pregled je potreben še dodaten program, ki ni na voljo. Zahtevano je, da se podatki nahajajo v Excelu, kjer lahko podatke pregledujete na kateremkoli računalniku in tako niste omejeni na WinCC ali katerikoli drug strokovni program. Za rešitev te zahteve v VBS-Editorju (slika 125) ustvarite program, s katerim je možno trenutne rezultate shraniti neposredno v Excelovo tabelo.

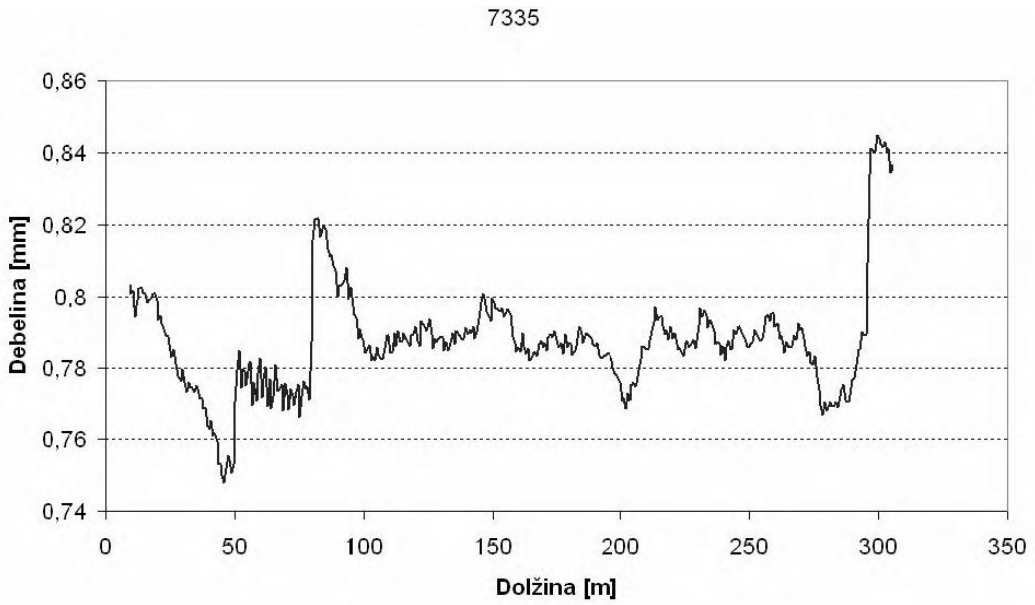


Slika 125: VBS Editor
Vir: Lasten

Program je sestavljen iz več korakov. Najprej preverja podatke in če je vrednost nad določeno vrednostjo, merilnik ni vstavljen, pod to vrednostjo pa merilnik meri in s tem lahko začne arhivirati. Nato v ozadju zažene Excel, preveri, če za trenutno šaržo že obstaja datoteka in jo odpre. Če pa datoteka še ne obstaja, program odpre datoteko, ki vsebuje obliko poročila, ter jo takoj shrani pod trenutnim datumom in številko šarže. Za vsako šaržo ustvari novo datoteko, in to zato, da lahko linija nadaljuje z isto šaržo tudi v primeru, če bi se zaradi kateregakoli razloga ustavila.

Ko ima program odprto datoteko, se postavi v prvo kolono prve vrstice drugega lista in preveri, če je celica prazna ali ne. Če je prazna, zapiše v to vrstico podatke, v nasprotnem primeru pa se premakne za eno vrstico nižje in ponovno preveri, če je celica prazna. Vsakič, ko se zapišejo podatki, se negira vrednost statusne luči, ki posledično utripa. S tem oznanjuje, da se vrednosti arhivirajo.

Arhiviranje se konča, ko podatek preseže določeno vrednost. Že prej je bilo omenjeno, da to pomeni, da merilnik ne meri več. S tem shrani in zapre datoteko. To datoteko lahko sedaj odpremo na kateremkoli računalniku, ki ima nameščen Excel. Na prvem listu se nahaja graf (slika 126) v odvisnosti od dolžine z naslovom številke šarže, na drugem pa se nahajajo podatki v tabelarični obliki (tabela 6).



Slika 126: Graf v Excelu
Vir: Lasten

Tabela 6: Pregled podatkov

	A	B	C	D
1.	Št. šarže	Dolžina (m)	Debelina (mm)	Datum in čas
2.	7335	9.117492830490	0.80305986754	14. 12. 2010 6:33:33
3.	7335	9.690349586002	0.80877865432	14. 12. 2010 6:33:34
4.	7335	10.000027391055	0.81823948301	14. 12. 2010 6:33:36
5.	7335	10.456988834963	0.81273403211	14. 12. 2010 6:33:37
6.	7335	11.947609395831	0.71727198376	14. 12. 2010 6:33:38
7.	7335	11.967944668271	0.74737459385	14. 12. 2010 6:33:41
8.	7335	12.009467484902	0.74736593725	14. 12. 2010 6:33:43
9.	7335	12.690385027495	0.74583029421	14. 12. 2010 6:33:44
10.	7335	13.460938489177	0.79583558374	14. 12. 2010 6:33:45
11.	7335	14.129735051491	0.73330482974	14. 12. 2010 6:33:47
12.	7335	14.999648881283	0.75634944642	14. 12. 2010 6:33:49
13.	7335	15.384839155534	0.75939090921	14. 12. 2010 6:33:51
14.	7335	16.319826319826	0.73829585930	14. 12. 2010 6:33:52
15.	7335	16.545467364825	0.74846677882	14. 12. 2010 6:33:53
16.	7335	17.889696789764	0.72827361181	14. 12. 2010 6:33:54
17.	7335	18.003748983728	0.72393849282	14. 12. 2010 6:33:55
18.	7335	19.000989898911	0.76319826311	14. 12. 2010 6:33:56
19.	7335	20.855346373628	0.79639738291	14. 12. 2010 6:33:57
20.	7335	21.938475932048	0.79384928746	14. 12. 2010 6:33:58
21.	7335	22.984738273492	0.75639284768	14. 12. 2010 6:33:59
22.	7335	23.95867389201	0.76593752222	14. 12. 2010 6:34:00
23.	7335	24.55783759264	0.75462948538	14. 12. 2010 6:34:02

Vir: Lasten

Poleg te datoteke se na enak način ustvari še datoteka, poimenovana z datumom. V to datoteko se arhivirajo vse vrednosti tega dne.

Povzetek:

SCADA so namenska programska orodja, ki so namenjena zajemanju in obdelavi podatkov. SCADA je kratica za angleške izraze *Supervision, Control, Alarm Data Acquisitions*, kar pomeni nadzor, kontrolne, alarmirne podatkovne enačbe. Že iz imena sledi, da gre za sisteme, ki združujejo v sebi več funkcij:

- zajemanje podatkov,
- kontrola podatkov,
- odločanje o obnašanju sistema.

V tem poglavju je opisana programska oprema SIMATIC WINCC FLEXIBLE, obširneje predvsem njena glavna orodja, prav tako pa je opisana izdelava nadzornega sistema.

Vprašanja:

1. Kakšna programska orodja so SCADA in katere funkcije združujejo?
2. Naštejte prednosti splošno uporabnih SCADA sistemov.
3. Kako poteka komunikacija v programskem orodju za ustvarjanje grafičnega vmesnika?
4. Kaj lahko vstavljate v sistem z orodjem TOOLS?
5. Kaj omogoča gumb Runtime System?
6. Opišite namenski programski paket Simatic WinCC.
7. Naštejte najpomembnejša orodja programskega paketa WinCC in jih na kratko opišite.
8. Kaj veste o grafičnem urejevalniku?
9. Katere tipe dinamike, ki jih lahko uporabite na slikah, poznate?
10. Opišite program za arhiviranje podatkov.

9 LITERATURA IN VIRI

Colnarič, M. *Osnove digitalne tehnike v računalništvu*, UM FERI, Maribor, 2002.

Colnarič M., Verber, D. *Mikroprocesorji*, UM FERI, Maribor 2000. Dostopno na naslovu: rts.uni-mb.si/misc/materiali/mikrorac/mp.pdf.

Bartenschlager, J. *Mehatronika*, Založba Pasadena, 2009.

Siemens, Simatic. Programming with STEP 7

Getting Started S7-GRAPH for S7-300/400 Programming Sequential Control System, User Manual 2009.

Siemens, Simatic HMI. Operating and Monitoring with WinCC, User Manual 2009.

Siemens, Simatic. S7-300 Automation System, CPU Specifications: CPU 31x, User Manual 2009.

Siemens, Simatic. Distributed I/O System, User Manual 2009.

Siemens, Simatic. S7-300 Programmable Controller Module Specifications, User Manual 2009.

http://www.cplusplus.com/doc/tutorial/

http://www.bloodshed.net/devcpp.html

http://www.w3schools.com/

http://www.sourceboost.com/Products/Flowcode/Overview.html

Projekt **Impletum**
Uvajanje novih izobraževalnih programov na področju višjega strokovnega izobraževanja v obdobju 2008–11

Konzorcijski partnerji:

Operacijo delno financira Evropska unija iz Evropskega socialnega sklada ter Ministrstvo RS za šolstvo in šport. Operacija se izvaja v okviru Operativnega programa razvoja človeških virov za obdobje 2007–2013, razvojne prioritete Razvoj človeških virov in vseživljenjskega učenja in prednostne usmeritve Izboljšanje kakovosti in učinkovitosti sistemov izobraževanja in usposabljanja.