UNIVERZA V LJUBLJANI
FAKULTETA ZA ELEKTROTEHNIKO

VISOKOŠOLSKI STROKOVNI ŠTUDIJ

DIPLOMSKO DELO

MIKROKRMILNIŠKI SISTEM ZA UPRAVLJANJE KORAČNIH MOTORJEV

Damjan Štritof

Mentor: doc. dr. Andrej Žemva

LJUBLJANA, 2001

Zahvala

Zahvaljujem se svojemu mentorju doc. dr. Andreju Žemvi za pomoč in nasvete pri izdelavi diplomskega dela ter vsem, ki so mi kakorkoli pomagali pri delu.

Posebna zahvala je namenjena očetu Andreju in materi Ani, ki sta mi omogočila študij in me nenehno podpirala, bratu Domnu, Mateji in prijateljem za potrpljenje in moralno podporo.

Povzetek

V diplomskem delu sem se ukvarjal z izdelavo mikrokrmilniškega sistema za upravljanje koračnih motorjev. Celoten sistem sestoji iz krmilnega dela, močnostnega dela in napajalnika. Za praktičen prikaz delovanja so uporabljeni trije koračni motorji ter model robotske roke. Uporabljen protokol za dvosmeren prenos podatkov od krmilnega dela do močnostnega dela je RS232.

Krmilni del predstavljajo matrična tipkovnica, dvovrstični LCD prikazovalnik, razširitveni priključek in RS232 vmesnik. Vse procese upravlja mikrokrmilnik Atmel AVR 90S8515. Izvorna koda je napisana v jeziku C in prevedena z GNU prevajalnikom AVRGCC v okolju Linux.

Krmilni del lahko nadomestimo s terminalskim programom, ki teče na osebnem računalniku. Prikazana je uporaba programskega jezika Microsoft Visual Basic 6.0

Močnostni del predstavljajo veriga transistorjev tipa Darlington, vezje za vklop releja, priključek za končna mikrostikala in RS232 vmesnik. Vse procese upravlja mikrokrmilnik Atmel AVR 90S2313. Izvorna koda je prav tako prevedena z GNU prevajalnikom AVRGCC.

Model robotske roke za praktični prikaz je sestavljen iz kompleta LEGO Technic z nekaj dodatnimi elementi. Za premikanje po prostoru so uporabljeni trije koračni motorji ter magnetno stikalo za prenašanje kovinskega bremena.

Ključne besede: koračni motor, mikrokrmilnik AT90S8515 in AT90S2313, AVRGCC prevajalnik, robotska roka.

Abstract

The main purpose of this B.Sc. thesis was to develop a microcontroller based stepper motor driver. Whole system is based on a controlling part, power part and power supply. Three stepper motors and model of robotic arm are used for practical demonstration of a system. For bidirectional data transmission from controlling part to power part protocol RS232 is used.

Controlling part consists of a matrix keypad, two lines LCD display, expansion connector and RS232 interface. All processes are managed by microcontroller Atmel AVR 90S8515. The source code is written in C language and compiled with GNU compiler AVRGCC on Linux platform.

Controlling part can be substituted with terminal software, running on personal computer. Small application is written in Microsoft Visual Basic 6.0.

Power part consists of Darlington transistors chain, circuit for controlling relay, connector for micro-switches and RS232 interface. All processes are managed by microcontroller Atmel AVR 90S2313. Source is compiled with AVRGCC as well.

For practical demonstration, model of robotized arm is made of LEGO Technic suite with some additional elements. Three stepper motors are used for area movement and magnetic switch for carry the load.

Keywords: stepper motor, microcontroller AT90S8515 in AT90S2313, AVRGCC compiler, robotized arm.

Vsebina

Kazalo slik

Kazalo tabel

1 Uvod

Osnovna naloga diplomskega dela je izdelava mikrokrmilniškega krmilnika koračnih motorjev za upravljanja preko RS232 serijskega vmesnika. V moji izvedbi je mogoče neodvisno upravljati štiri koračne motorje in eno magnetno stikalo, vendar lahko z opustitvijo nekaterih končnih mikrostikal upravljamo tudi z dvaintridesetimi motorji s spremembo močnostnega dela in izvorne kode.

Za praktičen prikaz delovanja sem uporabil model robotske roke s tremi koračnimi motorji, možna uporaba krmilnika pa je še:

- pri daljinskem nadzoru video kamer,
- pri premikanju zrcal in leč v svetlobnih in optičnih napravah,
- pri uporabi v antenskih rotatorjih za premikanje raznovrstnih anten.

Krmilni del je zasnovan tako, da z enostavnim uporabniškim vmesnikom (tipkovnica in LCD prikazovalnik) upravljamo s štirimi koračnimi motorji in magnetnim stikalom, ki so priključeni na močnostni del. Ker je uporabljen običajni RS232 vmesnik, lahko močnostni del preko standardnega serijskega kabla s prekrižanimi žicami (t.i. *null-modem* kabel) priključimo na katerikoli osebni računalnik.

Močnostni del pretvarja podatke, sprejete preko RS232 vmesnika, in aktivira izbrane koračne motorje ter magnetno stikalo. Obenem pošilja krmilnemu delu stanje šestih končnih koračnih mikrostikal ter morebitno stanje o napaki ali ponovnem zagonu vezja.

2 Izvedba močnostnega dela

Osnovno opravilo močnostnega dela krmilnika koračnih motorjev (v nadaljevanju SCU[1]) je upravljanje izbranih motorjev ter končnih mikrostikal. SCU je proti zunanji okolici povezan le preko običajnega RS232 vmesnika, zatorej ga lahko priključimo na katerikoli terminal z omenjenim vmesnikom.

2.1 Opis vezja

Srce vezja SCU je zmogljiv mikrokrmilnik AT90S2313 podjetja Atmel, ki vsebuje 2 *kB* programabilnega Flash in 128 *B* SRAM pomnilnika, 15 vhodno–izhodnih priključkov ter strojno podprt UART[2]. Kot pripadnika razvejane družine AVR mikrokrmilnikov ga odlikujeta širok nabor ukazov in velika hitrost izvajanja kode v primerjavi s sorodnimi izdelki drugih proizvajalcev (Microchip in Scenix).

Urino frekvenco f_{OSC} = 10 *MHz* za mikrokrmilnik U6 generira kvarčni kristal Q1; to je tudi najvišja vrednost, na katero ga lahko priključimo. Pri implementaciji strojnega UART-a moramo biti pozorni na t.i. bitno hitrost f_{BAUD}, ki je za dani mikrokrmilnik definirana kot:

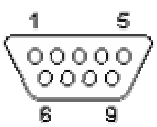
$$f_{BAUD} = \frac{f_{OSC}}{16(UBRR + 1)}, \qquad (1)$$

[1] SCU – *Stepper Controller Unit*, oznaka na tiskanem vezju
[2] UART – *Universal Asynchronous Receiver/Transmitter*, vmesnik med strojno opremo in RS232 napravami

pri čemer je vrednost UBRR[3] konstanta, zapisana v pomnilniku. V primeru SCU je za bitno hitrost 9600 *baud* pri urini frekvenci 10 *MHz* vrednost UBBR 64. Pri tej vrednosti je napaka manjša od 1%, kar je zgornja meja za doseganje kvalitetnih prenosov preko RS232 vmesnika. Povečevanje napake zmanjšuje šumno odpornost in s tem nezanesljivo sprejemanje in oddajanje paketov. Motnje se praviloma pojavljajo pri bitnih hitrostih, višjih od 9600 *baud*.

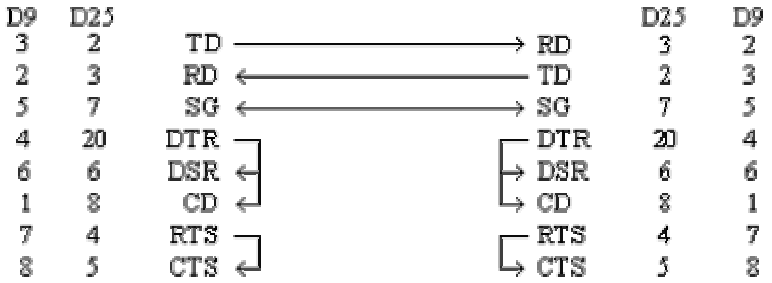
Mikrokrmilnikovi priključni sponki, namenjeni UART-u, nista primerni za neposredno priključitev na RS232 vmesnik. Logična nivoja pri mikrokrmilniku sta TTL združljiva (t.j. 0 in 5 *V*). Zaradi narave prenosa podatkov z RS232 protokolom (logično '0' predstavlja napetost, nižja od –12 *V* ter logično '1' predstavlja napetost, višja od +12 *V*) je potreben dodaten RS232/TTL vmesnik. V ta namen je v SCU vgrajeno vezje MAX232 (U5), ki samo generira potrebni napajalni napetosti in prilagodi napetostne nivoje. Tako je načrtovalcu prihranjen trud pri konstruiranju posebnega napajalnika.

RS232 vmesnik zahteva uporabo priključkov DB9 ali DB25. Slednji se danes skoraj ne uporabljajo. Za povezavo dveh terminalskih naprav se uporablja kabel z dvema ženskima priključkoma, na strani naprav pa dva moška priključka DB9. To je najbolj običajna uporaba za naprave, ki poleg podatkovnih signalov potrebujejo tudi ostale kontrolne signale (npr. zunanji modemi, starejši tiskalniki,…). Za povezovanje enostavnih terminalskih naprav na osebne računalnike pa se uporablja t.i. *null-modem* kabel, ki ima samo tri vodnike (GND, RX in TX). Oba podatkovna vodnika sta med sabo zamenjana, masa pa je priključena nespremenjeno. V SCU je uporabljen *null-modem* kabel, ki ga lahko priključimo na osebni računalnik ali na drug terminalski vmesnik. Naslednji dve sliki prikazujeta priključek DB9 in *null-modem* kabel:



Slika 2-1: Priključek DB9

[3] UBBR – *UART Baud Rate Register*, vrednost tega registra določa faktor deljenja za doseganje želene bitne hitrosti.



Slika 2-2: Null-modem kabel

Koristna lastnost mikrokrmilnikov nove dobe, v katere spadata tudi AT90S2313 in AT90S8515, je programiranje v vezju, ISP (ang. *In System Programming*). Tako pri vpisovanju nove strojne kode mikrokrmilnika ni potrebno fizično odstraniti iz vezja in ga vstaviti v programator, zato odpadejo tudi dragi, strogo namenski programatorji. V ta namen je na tiskanem vezju 6-pinski konektor za priključitev enostavnega univerzalnega programabilnega kabla na osebni računalnik. Mikrokrmilnik za ISP programiranje potrebuje naslednje signale: MOSI, MISO, SCK, RESET in GND, ki jih generira programator z ustrezno programsko opremo. Če na tiskanem vezju ni vira enosmerne napetosti, potrebujemo še VCC (+5 *V*).

Poglavitni del SCU predstavlja vzporedno 8-bitno vodilo, na katerega so priključeni multiplekser 74HC139 (U7), dva dvojna vmesnika (ang. *buffer*) 74HC244 (U2 in U4) in rele (RE1). Prvi štirje biti so priključeni na vhodne sponke obeh vmesnikov, izhodne sponke pa so vezane na polje tranzistorjev tipa Darlington, ki ga predstavljata integrirani vezji ULN2804 (U1 in U3). V vsakem od obeh vezij se nahaja osem tranzistorjev, tako da lahko krmilimo šestnajst izhodov z največjim tokom 500 *mA*.

Naslovna priključka multiplekserja 74HC139 (U7) sta povezana z mikrokrmilnikovimi sponkami 12 in 13, izhodi pa so povezani na omogočitvene priključke vmesnikov U2 in U4. Ustrezna naslovna kombinacija omogoči izbran vmesnik ter onemogoči ostale tri. Tako je v določenem trenutku mogoče krmiljenje le enega motorja, kar pa je neopazno z ustrezno spisanim gonilnikom. Naslednja tabela prikazuje vse možnosti pri naslavljanju motorjev:

Bit0	*Bit1*	*Izbran motor*
0	0	M1
0	1	M2
1	0	M3
1	1	M4

Tabela 2-1: Naslavljanje motorjev

Za proženje releja RE1 je predvidena napetost 5 *V* (logična '1'), ki jo generira mikrokrmilnik na priključku 14. Delilnik napetosti (upora R8 in R9) je namenjen za pravilen bazni tok NPN tranzistorja T1 pri visoki vhodni napetosti ter za popolno zaprtje ob nizki vhodni napetosti (0 *V*). Relejske sponke so sklenjene pri odprtem tranzistorju in razklenjene pri zaprtem.

2.2 Opis izvorne kode

Celotna programska koda je napisana v jeziku C in prevedena s prostim prevajalnikom AVRGCC, ki je nastal kot del projekta GNU[4], v okolju RedHat Linux (verzija 7.0). Za vpisovanje prevedenih strojnih datotek (*.ROM) v mikrokrmilnikov pomnilnik sem uporabil Atmel AVR Programmer v okolju Windows 2000, za strojno odstranjevanje napak (ang. *debugging*) in preverjanje vrednosti spremenljivk pa GDB (*GNU Project Debugger*) v okolju Linux.

Programska oprema v močnostnem delu krmilnika koračnih motorjev mora zagotoviti pretvorbo podatkovnega toka, sprejetega preko RS232 vmesnika, v aktiviranje izbranih motorjev. Omogočena mora biti tudi povratna informacija o položajih končnih mikrostikal in morebitnih napakah krmilnemu delu.

Izvorna koda je napisana kot neskončna zanka, ki se nikoli ne zaključi. Začetni del programa je namenjen nastavitvam vhodno-izhodnih vmesnikov, serijskega vmesnika ter testiranju releja in koračnih motorjev. Po uspešni inicializaciji program vstopi v neskončno ponavljajočo se zanko. Vsakokrat se prebere vrednost registra UDR (*UART Data Register*) in stanje na priključnih sponkah mikrostikal. Pri branju iz registra UDR preberemo en znak, poslan preko RS232, pri pisanju v ta register pa sprožimo prenos

enega znaka. Vsakokrat je potrebno preverjati vsebino statusnega registra USR (*UART Status Register*). Z vpisovanjem novega podatka lahko začnemo, ko je postavljen bit TXC (*Transmit Complete*). Sprejet podatek postane veljaven, ko je postavljen bit RXC (*Receive Complete*). Mikrostikala so vezana preko t.i. *pull-up* uporov na priključke PB3 do PB7 in PD6. Sklenjeno stikalo povzroči padec napetosti na pripadajočem priključku iz 5 *V* na 0 *V*.

Program vsakokrat pošlje stanje mikrostikal (*Switch_Bit*), napake (*Board_Error_Bit*) in ponovnega zagona (*Board_Reset_Bit*). V času inicializacije je postavljen bit ponovnega zagona; zbriše se ob vstopu v glavno zanko. Pri postavitvi kateregakoli mikrostikala ima krmilni del nalogo, da ustavi ali obrne smer vrtenja pripadajočih koračnih motorjev. Vsak motor ima v ta namen dve mikrostikali za določanje začetnega in končnega položaja.

Izbira motorja je izvedena s postavljanjem bitov (*Motor_Sel_Bit*) in s tem neposredno naslavljanje multiplekserja U7. Sekvenco, potrebno za vrtenje koračnih motorjev, generiramo s krmilnim delom. Vrednost bitov (*Data_Bit*) se preslika na podatkovno vodilo na portu D (PD2 do PD5). Na tiskanem vezju SCU je en rele, ki ga upravljamo z vrednostjo bita *Relay1_Bit*, en bit pa ostaja neuporabljen.

Spodnji dve tabeli prikazujeta razpored bitov v oddajnem (*Transmit*) in sprejemnem (*Receive*) registru UDR:

Bit	*Ime*
0	Board_Reset_Bit
1	Board_Error_Bit
2	Switch6_Bit
3	Switch5_Bit
4	Switch4_Bit
5	Switch3_Bit
6	Switch2_Bit
7	Switch1_Bit

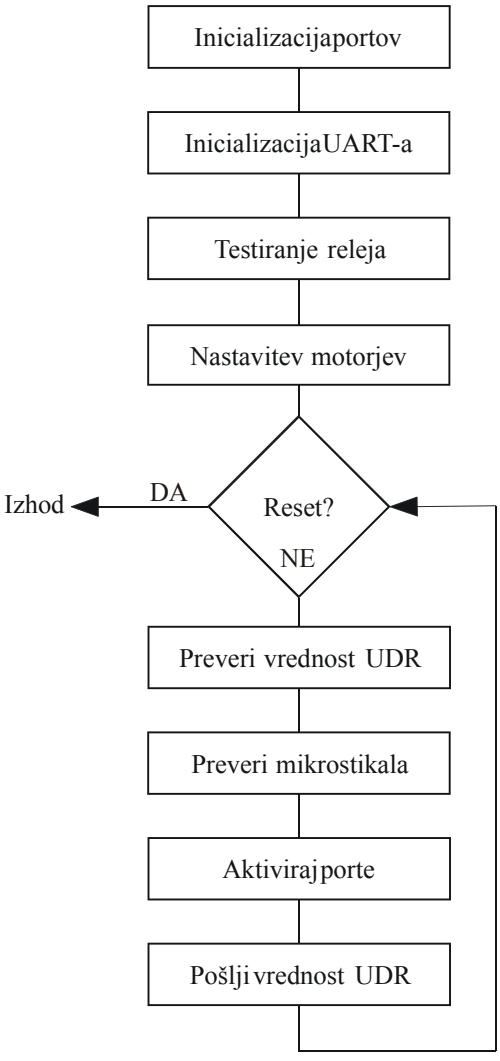
Tabela 2-2: UDR oddajni register

[4] GNU – okrajšava za *GNU's Not Unix*, projektna skupina za razvoj Unix-u podobnega prostega operacijskega sistema

Bit	*Ime*
0	Relay2_Bit
1	Relay1_Bit
2	Data4_Bit
3	Data3_Bit
4	Data2_Bit
5	Data1_Bit
6	Motor2_Sel_Bit
7	Motor1_Sel_Bit

Tabela 2-3: UDR sprejemni register

Diagram poteka prikazuje naslednja slika:



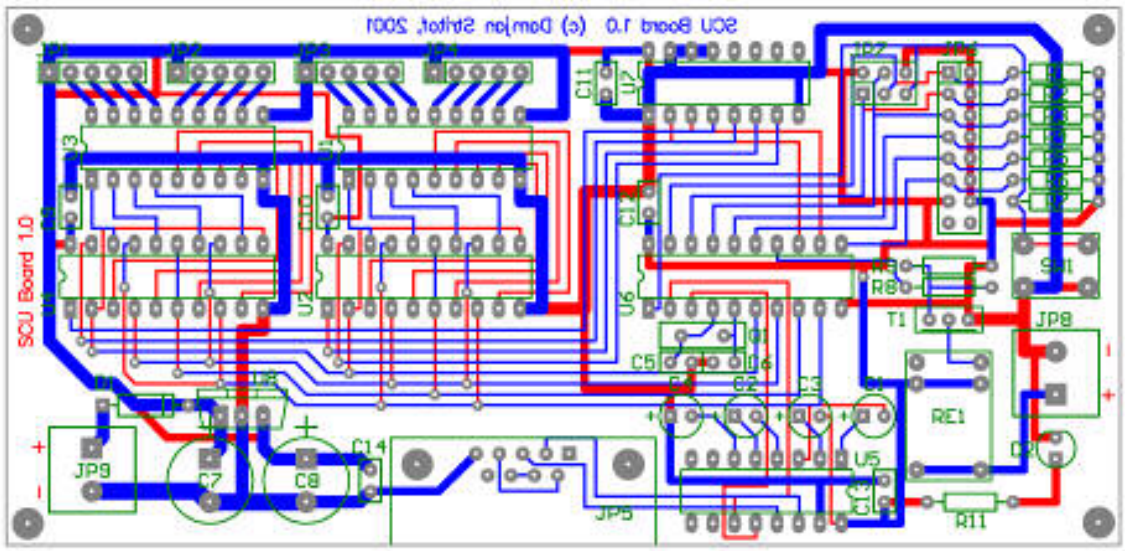
Slika 2-3: Diagram poteka močnostnega dela

2.3 Izdelava tiskanega vezja

Dvoplastno tiskano vezje (tudi tiskanina, PCB[5]) je bilo načrtovano s programskim paketom EDA/Client 98, podjetja Protel International Pty Ltd, po električni shemi, narisani z istim programom. Dimenzije ploščice so 132 x 64 *mm*, najmanjša debelina povezav je 15 *mil* in največja 80 *mil* (1 *mil* je enak 0.254 *mm*), debelina lukenj pa 0.8 in 1 *mm*. Vezje je izdelano v klasični (t.i. *through-hole*) tehnologiji.

Tiskano vezje je bilo v celoti izdelano v domači delavnici. Zrcaljena pozitiva za obe plasti (zgornjo in spodnjo) sta bila tiskana na prozorno folijo z laserskim tiskalnikom. Za prenos slike na tiskanino sem uporabil foto-postopek, pri katerem se na vsako stran dvoplastnega vitroplasta nanese svetlobno občutljiv lak, ki se suši v peči pri 75 ^{o}C. Nato se na vsako stran pritrdi folijo, ki jo osvetljujemo pod ultravijolično svetlobo. Po razvijanju ploščice v raztopini natrijevega luga sledi jedkanje v raztopini solne kisline, vode in vodikovega peroksida. Pred oksidacijo sem izdelano ploščico zaščitil z zaščitnim lakom.

Spodnja slika prikazuje večplastni pogled (*multilayer*) na tiskano vezje. Zgornja plast (*top layer*) je označena z rdečo, spodnja (*bottom l.*) z modro, obrisi elementov in tekst (*silkscreen*) pa z zeleno barvo:



Slika 2-4: Tiskano vezje močnostnega dela

[5] PCB - ang. *Printed Circuit Board*, tiskano vezje

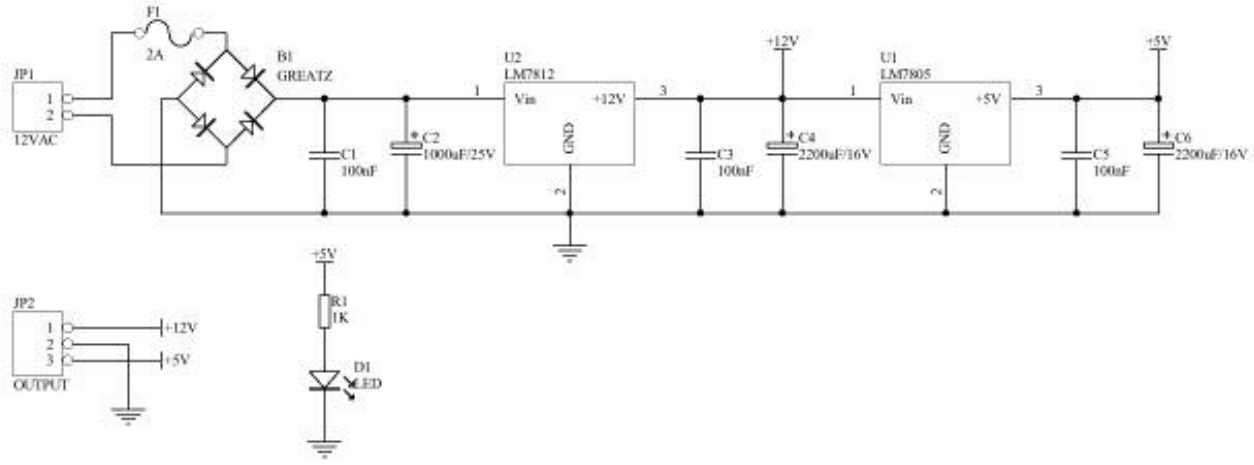
2.4 Napajalnik

Krmilnik koračnih motorjev za pravilno delovanje potrebuje vir enosmerne napetosti. Krmilnemu delu (SYS) je namenjena stabilizirana napetost vrednosti 5 *V*, močnostnemu delu (SCU) pa 12 *V*.

Pričujoče vezje je šolski primer napajalnika. Izmenična napetost vrednosti 12 *V* iz transformatorja je pripeljana preko varovalke na Greatzov mostič. Dobljena polnovalno usmerjena napetost se zgladi na kondenzatorjih C1 in C2. Napetostni stabilizator U2 na izhodu vzdržuje stalno vrednost 12 *V*. Koračni motorji za delovanje sicer ne potrebujejo stabilizirane napetosti; iz poskusov sem ugotovil, da delujejo na napetostih od 9 *V* do 15 *V*. Dobljena napetost se še enkrat zgladi na kondenzatorjih C3 in C4 ter stabilizira na izhodu vezja LM7805 (U1), kjer dobimo zglajeno napetost 5 *V*.

Logični del obeh vezij za delovanje potrebuje napetost 5 *V* ± 10%, kajti TTL vezja (vmesnika in multiplekser), EEPROM in mikrokrmilnik so zelo občutljivi na previsoko napetost. Med priključnima sponkama in maso sta nameščena še blokirna kondenzatorja vrednosti 100 *nF* za odpravljanje nezaželjenih motenj ob preklopu logičnih vezij. LED dioda D1 sveti ob prisotnosti enosmerne napetosti.

Uporaba opisanega napajalnika pri delovanju celotnega sistema ni nujna, saj oba dela (močnostni in krmilni) vsebujeta svoja napetostna stabilizatorja. Tako lahko sistem napajamo samo z laboratorijskim napajalnikom.



Slika 2-5: Električna shema napajalnika

3 Koračni motorji

Koračni motorji (ang. *stepper motor*) so enosmerni električni motorji brez krtačk (*komutatorskih*[6] sponk). Te so najbolj pogost vir okvar pri električnih motorjih, zato imajo motorji brez krtačk bistveno daljše zagotovljeno delovanje kot tisti z njimi. Zaradi tega je uporaba koračnih motorjev primerna tam, kjer se zahteva nepretrgano delovanje in/ali delovanje brez okvar (npr. uporaba v industriji).

3.1 Lastnosti

V primerjavi z običajnim enosmernim električnim motorjem zahteva koračni motor težavnejši strojni vmesnik. Pri aplikacijah, kjer se ne zahteva velike hitrosti vrtenja ali nadzora položaja in/ali hitrosti, imajo prednost običajni električni motorji, sicer pa je priporočljiva uporaba koračnih motorjev.

Ob predpostavki, da se koračni motor zaradi prevelikih obremenitev pri delovanju ne zaustavi oz. da ne spodrsava, lahko pravilno nadzorujemo tako hitrost in smer vrtenja kot tudi položaj rotorja brez uporabe povratnega mehanizma (senzorjev). To je pomembna lastnost, ker takšni mehanizmi stanejo vsaj toliko (ali še več) kot sam koračni motor. Najbolj značilni predstavniki povratnih senzorjev so optični in rotacijski enkoderji, tahometri in vztrajniki.

[6] Komutacija – spreminjanje smeri električnega toka

3.2 Navitja

Navitja (ang. *coils*) koračnih motorjev so običajno deklarirana za določeno enosmerno napetost in se odzivajo kot tuljave, priključene na napetost. Med običajnim delovanjem in tudi pri visokih koračnih frekvencah naj ne bi nikoli dosegle največjega dovoljenega toka.

Elektromagnetno polje, ki ga povzročajo navitja, je neposredno povezano s količino dovedenega toka: večje kot je polje, večji je navor. Največji navor zagotovimo, če ima vsako navitje pri vseh korakih na voljo največji dovoljeni tok. To najlažje dosežemo s povečanjem vzbujalne napetosti, vendar pa nikoli ne smemo preseči največjega dovoljenega toka. Tu proizvajalci najpogosteje uporabljajo *pull-up* upore med vsakim navitjem in napajalno napetostjo. Kompleksnejšo rešitev za omejitev toka pa predstavlja tudi t.i. tokovni rezalnik (ang. c*urrent-chopper*).

Običajno so vsa navitja del statorja, rotor pa je bodisi trajni magnet bodisi nazobčan blok iz mehkomagnetnega materiala.

3.3 Navor

Koračni motorji so deklarirani s t.i. držalnim navorom (ang. *holding torque*), ki je vrednost, potrebna za zadržanje rotorja na mestu ob nazivni vrednosti napetosti na navitju. Pomemben je tudi t.i. obračalni navor (ang. *turning torque*), ki je pomemben za dejansko obračanje rotorja ob nazivni napetosti. Na žalost ga proizvajalci le redko navajajo v svojih podatkih. Prednost koračnih motorjev pred običajnimi sta tudi veliko večji navor pri nizkih hitrostih in občutno nižja minimalna hitrost vrtenja. Nasprotno pa je maksimalna hitrost bistveno nižja, zato se koračni motorji pri velikih hitrostih ne uporabljajo (razen v redkih primerih ob menjavi prestavnega razmerja).

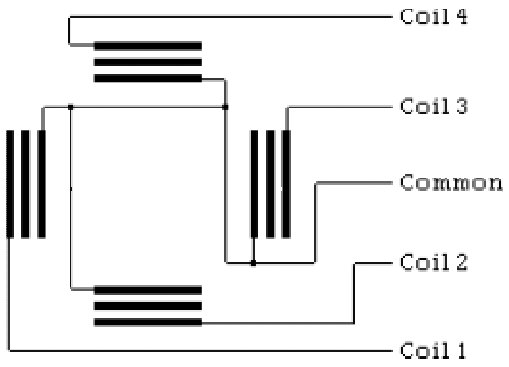
Kadar je nujen nadzor položaja rotorja ter hitrosti vrtenja in sta zahtevan navor ter hitrost v mejah zmogljivosti koračnega motorja, ta predstavlja ustreznejšo rešitev kot električni motor za manjše navore. Tako so koračni motorji uporabljeni v najbolj natančnih mehanskih delih (npr. pogoni računalniških diskovnih enot, tiskalniki, optični čitalniki,...).

3.4 Vrste motorjev

Koračne motorje v osnovi delimo v dve kategoriji: motorje s trajnim magnetom (ang. *permanent stepper motors*) in motorje s spremenljivim odporom (ang. *variable reluctance stepper motors*).

Predstavniki motorjev s trajnim magnetom so unipolarni in bipolarni koračni motorji. Značilnost unipolarnih motorjev je povezava vseh navitij v eni skupni točki in enostavno krmiljenje. Bipolarni motorji niso povezani v skupni točki, imajo pa občutno boljše razmerje med velikostjo in navorom kot unipolarni. Motorje s spremenljivim odporom imenujemo tudi hibridni koračni motorji, odlikujejo pa jih enostavno krmiljenje in najmanjša poraba toka.

Močnostni del krmilnika koračnih motorjev je namenjen bodisi unipolarnim bodisi hibridnim motorjem s štirimi navitji, kajti krmiljenje bipolarnih motorjev zahteva kompleksnejši vmesnik. Zaradi vsesplošne uporabe unipolarnih motorjev v diskovnih pogonih sem se odločil za motorje tipa STH-39D137-04, vzete iz zastarelih 5.25" disketnih enot. Vsak je sestavljen iz štirih 75 ohmskih navitij, ki se stikajo v eni točki (common). Naslednja slika prikazuje poenostavljeno zgradbo unipolarnega koračnega motorja:



Slika 3-1: Navitja unipolarnega motorja

3.5 Krmiljenje

Koračni motorji za delovanje potrebujejo zunanja vezja, ki generirajo vzbujalne sekvence. Te predstavljajo ponavljajoča zaporedja tokovnih impulzov skozi posamezna navitja. Glede na vrsto uporabe lahko motorje krmilimo s polnimi koraki[7] (ang. *full stepping*) ali s polovičnimi koraki (ang. *half stepping*).

Krmiljenje s polnimi koraki se uporablja pri aplikacijah, kjer se zahteva večji navor in manjša natančnost, vsakokrat pa vzbujamo po dve sosednji navitji. S takšnim krmiljenjem pridobimo približno 1.5-krat večji navor kot pri polovičnem, vendar se poveča tokovna poraba za 2-krat. Pri krmiljenju s polovičnimi koraki pa vzbujamo samo eno ali dve navitji in s tem podvojimo število korakov, povečamo natančnost in dosežemo najbolj gladko vrtenje.

Naslednji dve tabeli prikazujeta vzbujanje s polnimi in polovičnimi koraki:

Korak	*N1*	*N2*	*N3*	*N4*
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1
5	1	0	0	1
6	1	1	0	0
7	0	1	1	0
8	0	0	1	1

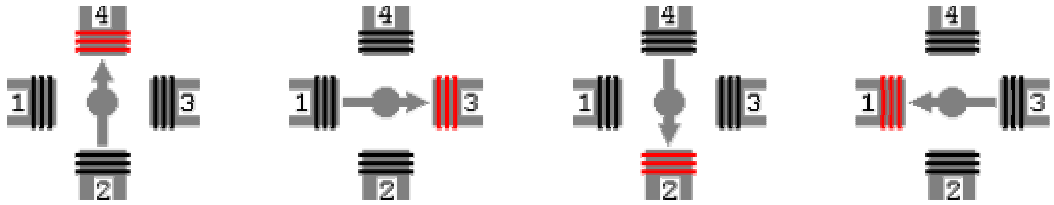
Tabela 3-1: Vzbujanje s polnimi koraki

Korak	*N1*	*N2*	*N3*	*N4*
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Tabela 3-2: Vzbujanje s polovičnimi koraki

[7] Krmiljenje s polnimi koraki – v tuji literaturi imenovano tudi *two winding combination.*

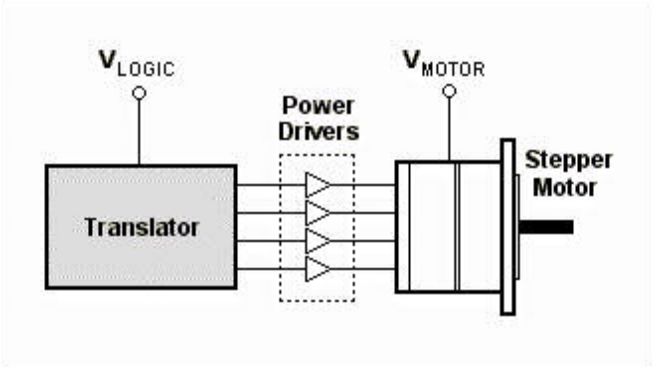
Poleg že opisanih obeh načinov pa lahko uporabljamo še vzbujanje enega navitja. Ta je najenostavnejši in vsebuje lastnosti obeh načinov (manjša tokovna poraba, večji navor in gladko vrtenje). Naslednja slika prikazuje zaporedje korakov pri vzbujanju enega navitja, s katerim dosežemo vrtenje v smeri urinega kazalca. Vrtenje v obratni smeri dosežemo z zamenjavo zaporedja.



Slika 3-2: Vrtenje v smeri urinega kazalca

Vsak koračni motor za pravilno delovanje potrebuje enosmerno napetost, ki jo generira krmilnik. Deklarirana napetost za uporabljene motorje je 12 *V* in ni združljiva s TTL nivoji, kajti največji tok skozi eno navitje (upornosti 75 *Ω*) znaša 160 *mA*. Motorjev zato ne moremo neposredno priključiti na mikrokrmilnikove sponke, na katerih nudi največjo napetost 5 *V* pri toku 20 *mA*.

Podjetje Allegro Microsystems je izdelalo visoko zmogljiva tranzistorska polja, družino vezij ULN28xx, ki se razlikujejo le po številu Darlingtonovih tranzistorjev. Namenjena so povezovanju nizkonapetostnih logičnih vezij z močnostnimi bremeni (tuljave, prikazovalniki in grelci). Izhodi vezij so tipa *open-collector* (aktivni pri nizki vhodni napetosti) in zaščiteni s pripenjalnimi diodami. Spodnja slika prikazuje običajno priključitev bremena na logično vezje:



Slika 3-3: Priključitev bremena na logično vezje

4 Izvedba krmilnega dela

Krmilnik koračnih motorjev za pravilno delovanje potrebuje vezje, ki generira sekvence korakov za posamezne motorje in skrbi za točnost. To nalogo opravlja krmilni del (v nadaljevanju SYS[8]); poleg tega nudi še nastavitev začetnih in končnih položajev, pomnenje različnih sekvenc in upravljanje drugih naprav, priključenih na RS232 vmesnik.

4.1 Opis vezja

Podobno kot v močnostnem delu, predstavlja osrčje krmilnega dela mikrokrmilnik. Uporabljen je zmogljivejši model Atmelove družine AVR, AT90S8515. Ima isti nabor ukazov kot AT90S2313, razlikuje pa se v velikosti SRAM in Flash pomnilnika (512 *B* in 8 *kB*) ter številu vhodno-izhodnih priključkov (32). Zmogljivejši mikrokrmilnik je uporabljen predvsem zaradi večjega števila priključenih komponent in obsežnosti izvorne kode, ki je v celoti napisana v programskem jeziku C.

Mikrokrmilnik v vezju ima 32 vhodno-izhodnih priključkov, dostopnih na štirih portih. Razen priključkov za serijsko programiranje (*ISP*) so vsi povezani preko *pull-up* uporovnih letvic z zunanjimi konektorji. Razporeditev priključkov prikazuje naslednja tabela:

Port	*Priključki*
A	JP2
B	JP1, ISP
C	LCD (podatkovne linije), JP5
D	LCD (krmilni signali), EEPROM in UART

Tabela 4-1: Razporeditev priključkov AT90S8515

[8] SYS – *System controller*, oznaka na tiskanem vezju

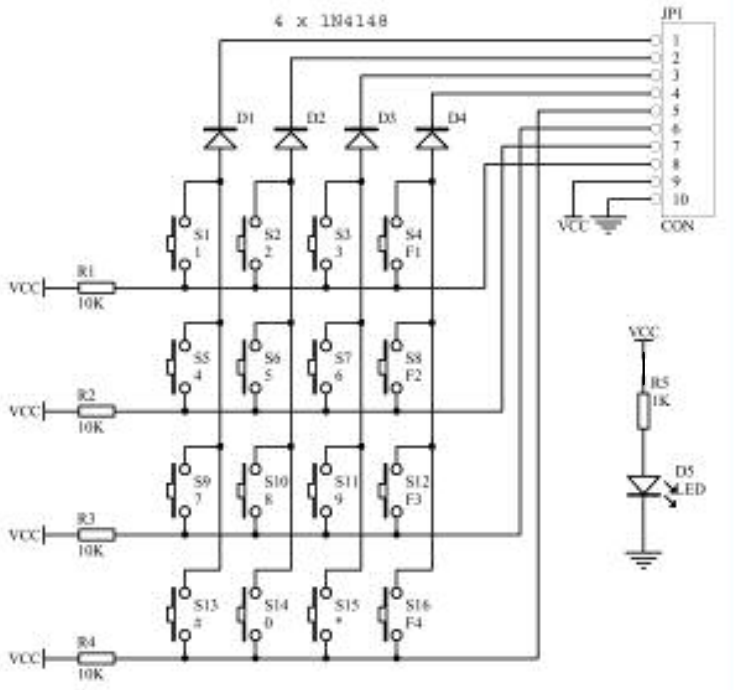
Iz tabele in shematskega načrta je razvidno, da si nekatere naprave delijo iste priključke. Tako so vsi trije signali za ISP konektor (MOSI, MISO in SCK) priključeni na port B vzporedno k JP1. Ko mikrokrmilnik normalno deluje, to ne predstavlja posebnih težav. Te se pojavijo med procesom programiranja, če vzporedno priključimo poljubno breme (npr. logično vezje ali uporovno verigo). Zaradi padca napetosti in nedefiniranih nivojev se ceneni ISP programatorji zmedejo, zato programska oprema javlja napake. Tako je priporočljivo povečati vrednost *pull-up* uporov iz 10 $k\Omega$ na npr. 100 $k\Omega$ oz. osamiti ISP konektor.

Tudi podatkovne linije LCD prikazovalnika so vzporedno priključene na konektor JP5, vendar ima njegovo logično vezje na vhodu vmesnike, ki jih lahko onemogočimo z nizkim kontrolnim signalom (*Enable*). Tako lahko z ustrezno napisano programsko opremo po potrebi dostopamo tudi do konektorja JP5 (pri tem je LCD prikazovalnik ugasnjen).

Za vnos in pregledovanje podatkov krmilni del potrebuje tipkovnico in LCD prikazovalnik. Zaradi zgornjih omejitev je tipkovnica priključena na edini prosti konektor (JP2 na portu A), ostala dva pa sta namenjena upravljanju dodatnih naprav oz. morebitni razširitvi. Uporabljena je enostavna matrična tipkovnica s 16 tipkami. Iz shematskega načrta je razvidno, da mora gonilnik oz. programska oprema najprej nasloviti vsako vrstico posebej (postaviti na logično '0') in odčitati stanje vseh stolpcev. Če je pritisnjena tipka v izbrani vrstici, je na pripadajočem stolpcu logična '1', sicer pa stanje visoke impedance (ker so diode zaporno polarizirane). Z ustrezno kombinacijo izbiranja vrstic in odčitovanja stolpcev lahko enostavno določimo, katera tipka je bila pritisnjena. Gonilnik mora odpraviti tudi odskakovanje tipk (t.i. *debounce effect)*. Pri mehanskih tipkah se preklop ne zgodi hipoma, temveč nekajkrat zaniha. Zaradi tega lahko mikrokrmilnik večkrat zazna pritisk tipke, kar je nesprejemljivo. Enostavna rešitev je kratka zakasnitev po vsakem branju stanja tipk. Na tiskanem vezju je vgrajena LED dioda D5, ki sveti ob prisotnosti napajalne napetosti. Shematski načrt tipkovnice prikazuje slika *4-1*.

V krmilniku je uporabljen dvovrstični LCD[9] prikazovalnik s 16 znaki (Hitachi LM016L). Pripada široki družini LCD prikazovalnikov z vgrajenim mikrokrmilnikom HD44780U, ki se priključujejo z osmimi ali štirimi podatkovnimi linijami (*DB0* do *DB7* oz. *DB0 do DB3*) in tremi kontrolnimi signali (*Enable*, *Register Select* in *Read/Write*). Tako lahko na SYS priključimo tudi druge prikazovalnike iz te družine (npr. štirivrstični LCD s 40 znaki etc.).

LM016L je tekstovni LCD in kot tak prikazuje znake velikosti 7 x 5 točk (višina x širina). Nabor znakov je zelo širok, saj vsebuje celoten ASCII[10] nabor, določimo pa lahko tudi 16 lastnih znakov, ki so zapisani v CG RAM-u (*Character Generator RAM*). Za določitev kontrasta potrebujemo enosmerno napetost V_{EE}, katere vrednost mora biti med 0 *V* in napajalno napetostjo V_{CC} (5 *V*). Najboljši kontrast dosežemo z napetostjo cca. 4.5 *V*, najlažje pa jo odjemamo z nastavljivega uporovnega delilnika, priključenega med maso in V_{CC}.



Slika 4-1: Shematski načrt tipkovnice

[9] LCD- *Liquid crystal display*, prikazovalnik na tekoče kristale
[10] ASCII – *American Standard Code for Information Interchange*, standard za prikaz znakov z binarnimi števili

Strojna oprema krmilnega dela dovoljuje uporabo tudi v drugih aplikacijah, kjer se zahteva krmiljenje preko RS232. S tem namenom je vgrajen tudi Atmel-ov serijski EEPROM[11] 24C512 kapacitete 64 *kB* (65536 8-bitnih besed). Povezava z mikrokrmilnikom je izvedena s t.i. *2-wire* serijskim protokolom, ki uporablja dva signala: SDA (*Serial Data*) in SCL (*Serial Clock Input*). Za potrebe SYS zadostuje kapaciteta pomnilnika v mikrokrmilniku, zato je omenjeni EEPROM neuporabljen.

Kot dodatni element sta uporabljena *piezzo*piskač in LED dioda, ki sta priključena na mikrokrmilnik (pin 4 na portu D). Služita predvsem za indikacijo npr. pritiska na tipke, napak, začetka ali konca programa,... Piskač je vezan preko tranzistorskega stikala, ki ga sestavljajo tranzistor T1 in upori R7 do R9, na napajalno napetost. Sprožimo ga s postavitvijo pripadajočega pina na logično '1', obenem pa sveti še LED dioda.

Podobno kot v močnostnem delu, potrebujemo za komunikacijo preko RS232 serijskega protokola posebno prilagodilno vezje. Uporabljen je Maxim-ovo vezje MAX232 s pripadajočimi kondenzatorji, izhodni (moški) konektor pa je tipa DB9. Format podatkov mora biti isti kot v močnostnem delu: hitrost 9600 baud, dolžina podatkov 8 bitov in en paritetni bit.

SYS za delovanje potrebuje stabilizirano enosmerno napetost vrednosti 5 *V*, ki jo lahko priključimo neposredno na vezje. Ker so logična vezja zelo občutljiva na previsoko napetost, je v krmilnem delu vgrajen interni stabilizator LM7805 (U1) z usmernikom in pripadajočimi kondenzatorji. Tako lahko na napetostni konektor priključimo enosmerno ali izmenično napetost, višjo od 7.5 *V*; oblika konektorja (bananski vtič) je namenjena uporabi cenenih napetostnih adapterjev.

[11] EEPROM - *Electrically erasable and programmable ROM*, električno zbrisljiv in programiljiv bralni pomnilnik

4.2 Opis izvorne kode

Celotna izvorna koda krmilnega dela je napisana v programskem jeziku C in prevedena s prostim GNU prevajalnikom AVRGCC. Za vpisovanje prevedenih strojnih datotek v mikrokrmilnikov pomnilnik sem uporabil Atmelov AVR ISP Programmer v okolju Windows 2000. Programska oprema v krmilnem delu mora zagotoviti dvosmerno komunikacijo preko RS232 vmesnika z močnostnim delom. Realizirati je potrebno tudi gonilnika matrične tipkovnice in LCD prikazovalnika.

Izvorna koda je napisana kot neskončna zanka, iz katere izstopimo s ponovnim zagonom oz. ob pritisku na tipko *Reset*. Začetni del programa je namenjen inicializaciji vseh vhodno-izhodnih vmesnikov, serijskega vmesnika in LCD prikazovalnika ter testiranju *piezo* piskača. Ob inicializaciji se na LCD prikazovalniku izpiše različica programske opreme (*AVR-SYS v1.0*) in avtorjevo ime. Po končani inicializaciji program vstopi v neskončno ponavljajočo se zanko. Vsakokrat se izvrši rutina *Scan_keyboard()*, ki vrne *ASCII* kodo tipke, ki je bila pritisnjena. Odločitvena zanka, realizirana s stavkom case, se glede na pritisnjeno tipko odloči med naslednjimi možnostmi (ostale tipke so v tej različici programske opreme neuporabljene):

Tipka	*Možnost*
1	Zavrti motor M1 za en korak v desno
2	Zavrti motor M1 za en korak v levo
3	Zavrti motor M2 za en korak v desno
4	Zavrti motor M2 za en korak v levo
5	Zavrti motor M3 za en korak v desno
6	Zavrti motor M3 za en korak v levo
F1	Aktiviraj magnetno stikalo
F2	Deaktiviraj magnetno stikalo

Tabela 4-2: Odločitvena zanka

Rutina *Scan_keyboard()* je namenjena odčitovanju pritisnjenih tipk. Vsakokrat se posamezno naslovi (postavi na logično '0') enega od stolpcev in prebere stanje vrstic. Zaradi že omenjenega odskakovanja tipk je potrebno stanje vrstic prebrati dvakrat (z zakasnitvijo 100 μs). Tipka je veljavna, če je stanje obakrat enako. Rutina nato iz matrike *Keys[4][4]* vrne dekodirano vrednost.

Rutina za aktiviranje oz. deaktiviranje magnetnega stikala v oddajnem registru *TX_Reg* postavi oz. zbriše vrednost bita *Relay1_bit* in sproži oddajanje.

Rutine za obračanje koračnih motorjev *M_rotate_right* in *M_rotate_left* v oddajnem registru *TX_Reg* postavljajo oz. brišejo enega izmed štirih podatkovnih bitov (*Data_bit*). Aktivni motor izberemo s postavljanjem oz. brisanjem bitov *Motor1_Sel_bit in Motor2_Sel_bit* v registru *TX_Reg*:

Motor1_Sel_bit	*Motor2_Sel_bit*	*Motor*
0	0	M1
0	1	M2
1	0	M3
1	1	M4

Tabela 4-3: Izbira aktivnega motorja

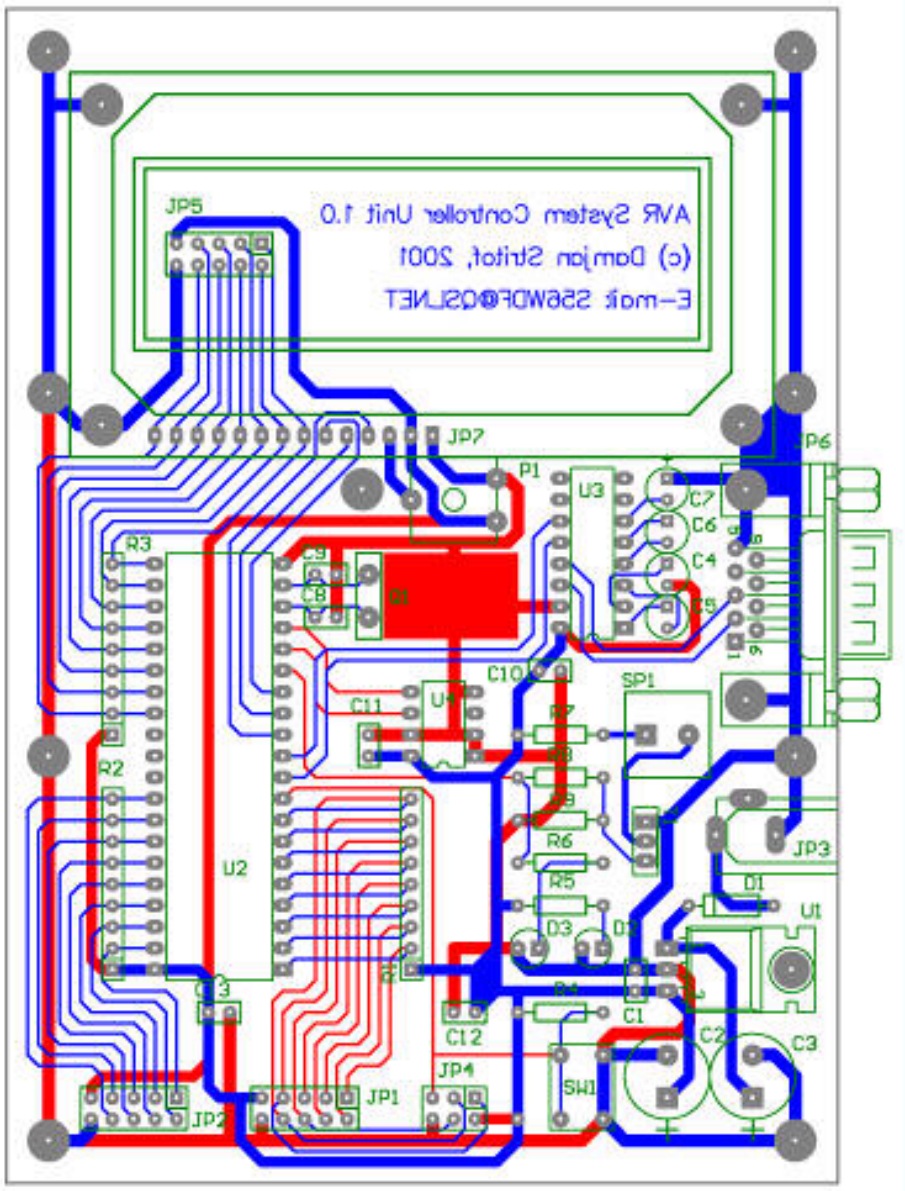
Vsakokratni pritisk ustrezne tipke povzroči premik pripadajočega motorja za en korak. Program preveri prejšnje stanje, zbriše poprej postavljene podatkovne bite in postavi nove. Zato je trenutno stanje vsakega motorja shranjeno v spremenljivkah *M_step_positive* (vrtenje v desno) in *M_step_negative* (vrtenje v levo). Program vsakokrat preveri stanje obeh spremenljivk, da generira ustrezno sekvenco korakov.

Zaradi časovne usklajenosti potrebujemo po vsakem generiranem koraku kratko zakasnitev (50 *ms*), ki je realizirana s funkcijo *Delay()*.

4.3 Izdelava tiskanega vezja

Krmilni del je bil prav tako kot močnostni v celoti načrtovan s programskim paketom EDA/Client 98. Dimenzije tiskanega vezja so 140 x 99 *mm*, najmanjša debelina povezav je 15 *mil* in največja 60 *ml* , debelina lukenj pa 0.8 in 1 *mm*. Vezje je izdelano v *through-hole* tehnologiji, zato so vsi uporabljeni elementi klasični.

Tiskano vezje je bilo v celoti izdelano v domači delavnici s foto-postopkom. Spodnja slika prikazuje večplastni (*multilayer*) pogled na tiskano vezje:



Slika 4-2: Tiskano vezje krmilnega dela

5 Izvedba krmilnega dela v okolju Visual Basic

Močnostni del krmilnika koračnih motorjev se proti priključenim napravam obnaša kot RS232 serijski terminal. To poglavje opisuje uporabo osebnega računalnika za krmiljenje močnostnega dela. Za razliko od strojne izvedbe, opisane v prejšnjem poglavju, nudi programska oprema več možnosti, preglednejše nastavitve in povezljivost z drugimi programi.

Programska izvedba krmilnega dela je bila zaradi enostavnosti načrtovana v okolju Visual Basic 6.0, podjetja Microsoft Corporation. Na tržišču je veliko grafičnih okolij za razvoj programov v različnih operacijskih sistemih. Tako se v Windows 9x/NT najpogosteje uporabljajo Borland Delphi in Visual C++, Microsoft Visual Basic ter Sun Java; v sistemih Unix pa je večino aplikacij napisano v jeziku C/C++ in Javi.

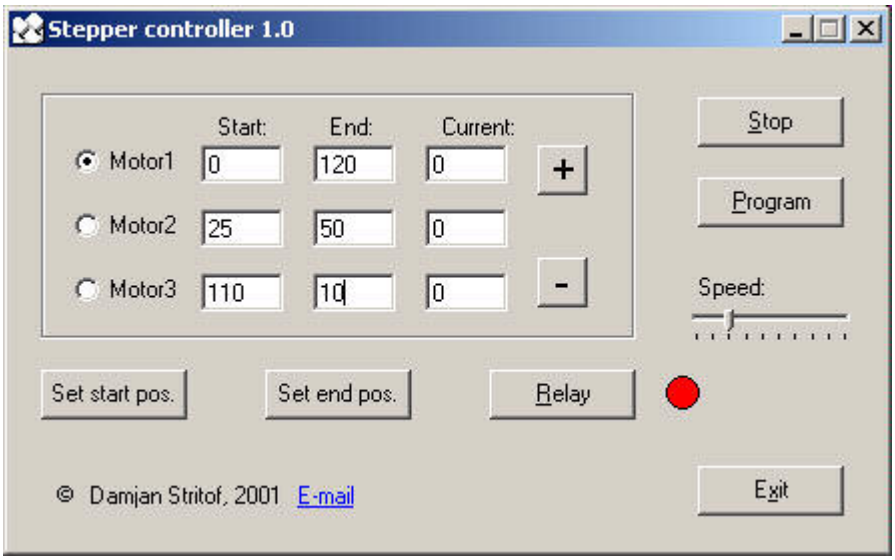
5.1 Opis delovanja

Glavno programsko okno predstavlja t.i. forma, kjer uporabnik s pritiskom na tipki ′+′ ali ′-′ določi začetne in končne položaje koračnih motorjev. Predhodno je potrebno omogočiti izbran motor s pritiskom na pripadajoč gumb, nato ga lahko poljubno premikamo, dokler ne dosežemo željenih položajev. Tipka *'Relay'* vklaplja oz. izklaplja magnetno stikalo, vendar je namenjena izključno testiranju relejskega sklopa.

Najpomembnejši del programa je avtomatsko pomikanje koračnih motorjev do prednastavljenih položajev, ki ga sprožimo s pritiskom na tipko *'Program'*, ustavimo pa s tipko *'Stop'*. Takrat se vsi motorji zavrtijo v osnovni položaj (stanje 0), nato pa se premaknejo do začetnega položaja, ki ga je določil uporabnik. Po kratki zakasnitvi, potrebni za umiritev mehanskih delov, se vklopi magnetno stikalo, ki pritegne kovinsko breme. Nato se po vrsti zavrtijo vsi motorji do končnega položaja. Hitrost vrtenja lahko

uporabnik spreminja s položajem drsnika. Glede na trenutni, začetni in končni položaj izbranega motorja se program odloči, kakšne podatke bo pošiljal močnostnemu delu, da se bodo motorji vrteli v pravo smer.

Ko vsi motorji dosežejo končno lego prvega cikla, se deaktivira magnetno stikalo in popusti breme. Po kratki zakasnitvi se vsi motorji po isti poti vrnejo v izhodiščno stanje, kjer se po istem postopku začne nov cikel. Spodnja slika prikazuje glavno programsko okno z vsemi nastavitvenimi gumbi:



Slika 5-1: Glavno programsko okno

Program bi lahko izboljšal z uvedbo dodatnih funkcij. Uporabniku prijaznejši program bi omogočal shranjevanje nastavitev, t.j. začetnih in končnih položajev vseh motorjev v tekstovno datoteko in njeno kasnejšo uporabo.

5.2 Opis programske kode

Kot je razvidno iz priložene izvorne kode, glavni del programa predstavlja časovnik, ki se osvežuje vsakih 10 *ms*. Ker se časovne funkcije v Visual Basic-u računajo glede na hitrost procesorjeve ure, je lahko omenjen osveževalni čas na drugem osebnem računalniku drugačen, kar pa ne predstavlja težav. Ob vsakem prelivu časovnika se shrani vsebina sprejemnega registra v globalno spremenljivko *RX_register* in zapiše vsebina oddajnega registra (*TX_register*) na komunikacijski vmesnik, ki je realiziran s sistemskim gonilnikom *comctl32.ocx*.

Upravljanje močnostnega dela je možno samo s spreminjanjem globalne spremenljivke *TX_register*, povratne informacije pa so na voljo v spremenljivki *RX_register*. Izbirni gumbi za določanje aktivnega motorja sprožijo neposredno postavljanje bitov Motor_Sel_bit, kot je opisano v definicijah močnostnega dela. Vrtenje izbranega motorja dosežemo z zaporednim pošiljanjem korakov oz. s postavljanjem in brisanjem bitov *Data_bit* (izvajamo logični operaciji *XOR* in *NAND* nad vsebino registra). Magnetno stikalo aktiviramo s postavljanjem oz. brisanjem bita *Relay1_bit*.

Pritisk gumbov za nastavitev začetnih in končnih položajev motorja shrani vsebino tekstovnih oken, kjer je prikazano trenutno stanje, v pripadajoče globalne spremenljivke *M_start_position* in *M_stop_position*. Ko sprožimo sekvenčno vrtenje, program vstopi v zanko, kjer po vrsti odvrti motorje za željeno število korakov *M_step_number*. Če je začetni položaj motorja manjši od končnega (vrtenje v desno), je število korakov, potrebnih za vsak motor, določeno z:

$$M\_step\_number = M\_stop\_position - M\_start\_position \quad (2)$$

sicer pa z (vrtenje v desno):

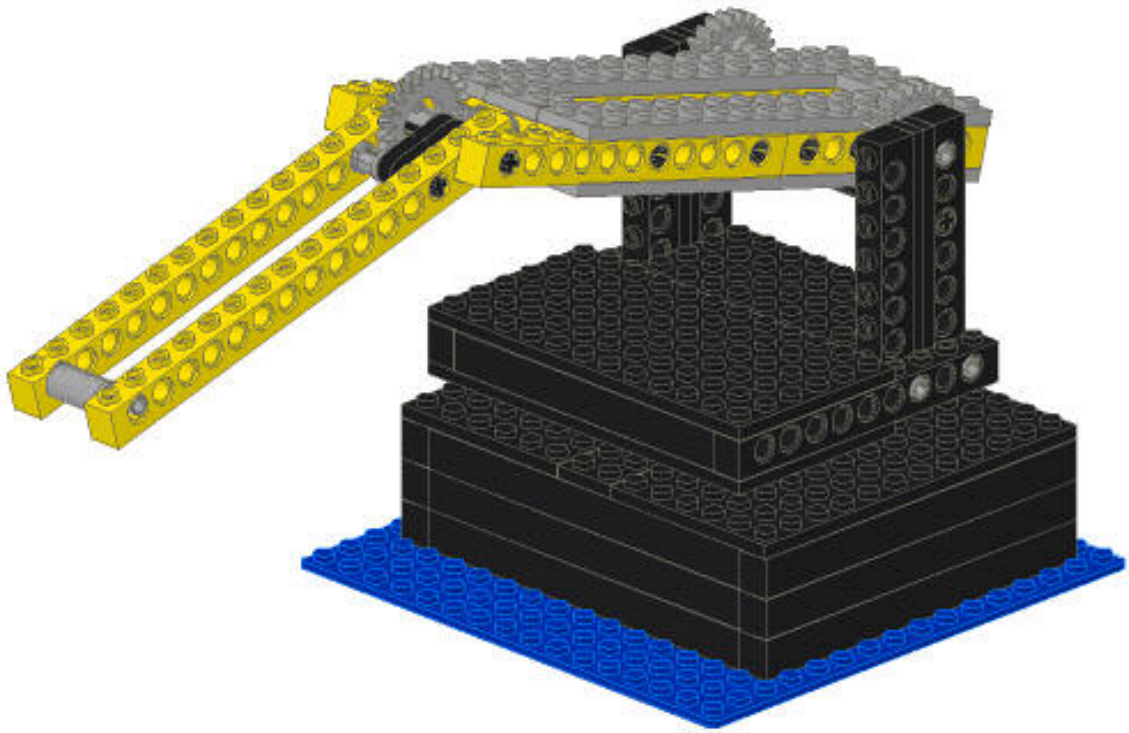
$$M\_step\_number = M\_start\_position - M\_stop\_position \quad (3)$$

Hitrost izvajanja je določena z vrednostjo drsnika, ki se pomnoži z osnovno zakasnitvijo, realizirano s funkcijo *Delay()*. Izvajanje lahko zaustavimo s pritiskom na tipko 'Pause', ki onemogoči časovnik; s ponovnim pritiskom pa se izvajanje nadaljuje. In navsezadnje, pritisk na tipko 'Exit' pošlje komunikacijskemu vmesniku vrednost '0', ga deaktivira in s tem omogoči uporabo drugim programom, zaustavi časovnik ter zapre glavno okno.

6 Izdelava modela robotske roke

Ena izmed možnosti uporabe krmilnika koračnih motorjev je upravljanje mehanske robotske roke. Zaradi zahtevnosti realizacije robotske roke iz kovinskih materialov sem za prikaz delovanja sistema izdelal model, ki je sestavljen iz gradnikov kompleta *Lego Technic*, treh koračnih motorjev tipa *STH-39D137-04* in povezovalnih kablov.

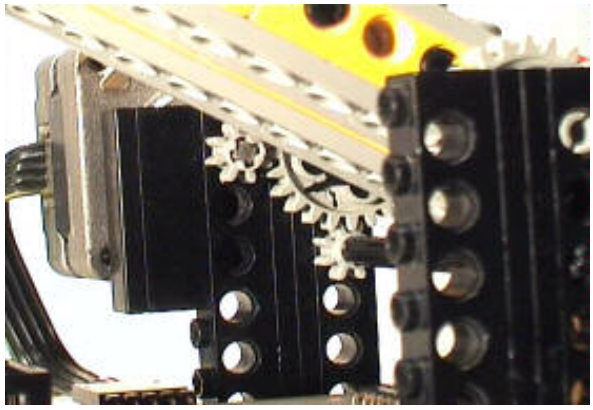
Kot je razvidno iz slike *6-1*, je model sestavljen iz treh delov. Nosilni del je namenjen vrtenju srednjega dela, na katerega je vpeta roka. Koračni motorji so pritrjeni na ohišje, rotorji pa so preko zobniških prenosov vpeti na posamezne vrtljive dele.



Slika 6-1: 3D model robotske roke

Slika *6-1* je bila izdelana s CAD[12] programskim paketom *MLCAD 2.0*, ki je namenjen izključno načrtovanju modelov v sistemu *Lego*, zato so izpuščeni vsi koračni motorji in povezovalni kabli. Na žalost programski paket ne omogoča naprednih funkcij (npr. mehanske in statične analize), zato je bil model robotske roke nekajkrat predelan, da so bile dosežene zadovoljive mehanske in statične lastnosti (težišče, obremenljivost in zmogljivost).

Prednost kompleta *Lego Technic* je predvsem enostavnost, dostopnost in cenenost izdelave, kar je bilo odločilnega pomena pri realizaciji modela. Vsekakor je največja pomankljivost nenatančnost, ki se kaže predvsem pri prenosnih mehanizmih, zato lahko model služi samo v demonstrativne namene. Moč koračnih motorjev se na vrtljive dele prenaša preko zobnikov. V modelu so uporabljeni trije prenosni sistemi, sestavljeni iz devetih podobnih zobnikov; pet jih ima 8, štirje pa 24 zob. Spodnja slika prikazuje zobniški prenos med koračnim motorjem in roko:



Slika 6-2: Zobniški prenosni sistem

Glavni vzrok za nenatančnost modela je prevelika zračnost med posameznimi zobniki, ki se ne prilegajo popolnoma drug drugemu. Zračnost je odvisna od toleranc pri izdelavi in nenatančnosti montaže zobnikov. Zaradi tega se mora koračni motor določen čas vrteti v prazno, da se začne premikati veliki zobnik in s tem celoten sistem. To nezaželjeno zakasnitev, ki je odvisna predvsem od obremenitve in položaja, lahko odpravimo z ustrezno napisano programsko opremo oz. z uporabo natančnejših zobnikov, kar pa v okviru kompleta *Lego Technic* ni mogoče.

[12] CAD – *Computer aided design*, računalniško podprto načrtovanje

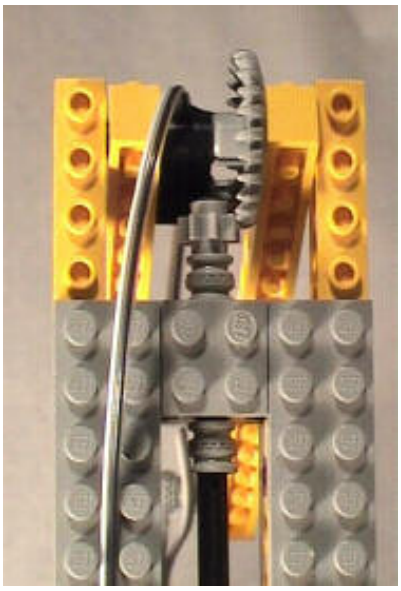
Prestavno razmerje dveh zobnikov *i* izračunamo po enačbi, v kateri upoštevamo število zob malega (Z_1) in velikega zobnika (Z_2):

$$i = \frac{Z_1}{Z_2} = \frac{8}{24} = 0{,}33 \tag{4}$$

Prestavno razmerje nam pove, za koliko stopinj se zavrti veliki zobnik, če se mali zobnik zavrti za eno stopinjo. Ker potrebuje koračni motor za en vrtljaj 360 korakov, je natančnost na velikem zobniku 0,33 stopinje, če se motor zavrti za en korak. Če upoštevamo še grobo ocenjeno zračnost, ki znaša približno ± 5% (odvisno od smeri vrtenja), je natančnost na velikem zobniku:

$$i = 0{,}33 \pm 5\% \tag{5}$$

Spodnja slika prikazuje zobniški prenos preko kardanske gredi, vpete na koračni motor, med obema premičnima deloma:



Slika 6-2: Zobniški prenos preko kardanske gredi

Vsi trije koračni motorji so natančno pritrjeni na plastično konstrukcijo s sekundnim lepilom, saj bi izdelava posebnih kovinskih nosilcev otežila realizacijo celotnega sistema. Za magnetno stikalo, ki služi za dviganje bremen, sem uporabil relejsko navitje. Preko dvožilnega kabla je priključeno na konektor JP8 močnostnega dela

krmilnika koračnih motorjev. Med procesom testiranja sem ugotovil, da preklop releja RE1 (in s tem magnetnega stikala), povzroči visok tokovni sunek, ki občasno resetira mikrokrmilnik U6. Zato sem vzporedno k priključnim sponkam dodal zaporno vezano diodo, ki ob preklopu omeji tokovni sunek.

Breme mora biti iz mehkomagnetnega materiala, saj se navadna jeklena pločevina med prenašanjem namagneti. Najprimernejši material za pritrditev bremena je relejska kotva ali transformatorska pločevina; slednja je bila uporabljena pri testiranju. Magnetna sila, ki pritegne kovinsko breme, je dovolj velika za prenašanje mnogo težjih bremen. Zaradi že omenjenih težav koračnih motorjev pa mora biti teža bremena kar najmanjša.

Ena izmed omejitev uporabe robotske roke je tudi njena velikost (oz. majhnost), ki jo pogojujejo dimenzije *Lego* gradnikov. Idealno bi bilo, če bi bil v podstavku robotske roke vgrajen močnostni del krmilnika z napajalnikom. Upoštevajoč, da morajo biti na močnostni del priključeni vsi koračni motorji, končna mikrostikala in magnetno stikal, je skupno število priključnih kablov 22 (stikala imajo skupno maso, motorji pa so vezani neposredno). V končni izvedbi sem za povezovanje koračnih motorjev uporabil večžilni ploščati kabel, tako da je omogočeno neovirano vrtenje v vse smeri.

7 Zaključek

V okviru diplomskega dela sem izdelal krmilnik koračnih motorjev, ki je sestavljen iz močnostnega in krmilnega dela. Celotna realizacija je obsegala risanje električnih shem, načrtovanje tiskanih vezji in njihovo izdelavo ter pisanje izvorne kode.

Kot nadomestilo krmilnemu delu sem s programskim paketom Visual Basic 6.0 izdelal računalniško aplikacijo in s tem nakazal več možnosti uporabe krmilnika. Celotno izvorno kodo za oba dela sem napisal v programskem jeziku C, ki sem ga med programiranjem dodobra spoznal. Prevajanje v strojno kodo sem izvršil s prostim GNU prevajalnikom AVRGCC.

Za prikaz delovanja sem iz gradnikov kompleta Lego Technic izdelal model robotske roke in opisal njegove prednosti ter slabosti in možnosti uporabe.

Pri načrtovanju krmilnika sem naletel na nekaj manjših težav, ki so bile večina programske narave. Električni shemi in tiskani vezji obeh delov krmilnika sta bili že ob prvem načrtovanju brez napak. Med procesom programiranja sem spremenil nekaj elementov in tako izboljšal celoten sistem.

8 Literatura

[1] *Atmel AT90S8515 Datasheet*, Atmel Corporation, 2000

[2] *Atmel AT90S2313 Datasheet*, Atmel Corporation, 1999

[3] *Atmel AVR Instruction Set*, Atmel Corporation, 1999

[4] B. W. Kernighan in D. M. Ritchie, *Programski jezik C*, Založba FERI, 1990

[5] Rich Neswold, *GNU Development Environment for the AVR Microcontroller*, 2001

[6] Paul Calverley, *Programming with AVRGCC using AVR Studio*, 2000

[7] *C Style Guide*, NASA - Goddard Space Flight Center, 1994

[8] *Using AVR UART in C - Application note 306*, Atmel Corporation, 1999

[9] *Allegro ULN2804 Datasheet*, Allegro Microsystems Inc., 2000

[10] Jason Johnson, *Working with Stepper Motors*, 1998 (http://eio.com/jasstep.htm)

[11] Ben Wirz, *A Stepper Motor Overview*, 1998 (http://www.wirz.com/stepper/)

[12] *Maxim MAX232 Datasheet*, Maxim Integrated Products, 2000

[13] Aleš Šuler, *Visual Basic 6.0 – Zvijače in nasveti*, Založba Flamingo, 2000

[14] Ian Harries, *Stepper Motors*, 1999 (http://www.doc.ic.ac.uk/~ih/doc/stepper/)

9 Priloge

9.1 Izpis izvorne kode močnostnega dela

```
/*
=====================================================================
 Title:        STEPPER CONTROLLER UNIT
 Author:       Damjan Stritof (damjan.stritof@campus.fe.uni-lj.si)
 Date:         7.2.2001
 Purpose:      SW for control four stepper motors via RS232 interface
 Updated:      14.3.2001
=====================================================================
*/

#include <io2313.h>
#include <io.h>

/* Enable pins for motors and relay: */
#define Motor1 0x00
#define Motor2 0x02
#define Motor3 0x01
#define Motor4 0x03
#define Relay  PINB2

/* Switches for start position: */
#define Switch1       PINB7
#define Switch2       PINB6
#define Switch3       PINB5
#define Switch4       PINB4
#define Switch5       PINB3
#define Switch6       PIND6

/* Motor direction (4 bit bus at port D): */
#define Step1R 0x20
#define Step2R 0x10
#define Step3R 0x08
#define Step4R 0x04
#define Step1L 0x04
#define Step2L 0x08
#define Step3L 0x10
#define Step4L 0x20

/* RS232 TX bits */
#define Board_Reset   0x80
#define Board_Error   0x40    // Not used!
#define Switch6_bit   0x20
#define Switch5_bit   0x10
#define Switch4_bit   0x08
#define Switch3_bit   0x04
#define Switch2_bit   0x02
#define Switch1_bit   0x01

/* RS232 RX bits */
#define Relay2_bit    7       // Not used!
#define Relay1_bit    6
#define Data4_bit     5
#define Data3_bit     4
#define Data2_bit             3
#define Data1_bit             2
#define Motor2_Sel_bit        1
```

```
#define Motor1_Sel_bit       0

/* Global variables: */
typedef unsigned char u08;

u08 RX_Reg;    // UART RX register
u08 TX_Reg;    // UART TX register
u08 portb;     // PORTB register
u08 portd;     // PORTD register

/* Function prototypes: */
void Delay_Step (void);
void Delay_Motor (void);
void Delay_Relay (void);
void Init_UART (void);
void Init_Ports (void);
void Transmit_Byte (char);
void Relay_Test (void);
void Relay_On (void);
void Relay_Off (void);
void Motor_Init (void);
char Receive_Byte (void);

/* Main program starts here: */
int main (void)
{
       //char data=0;

       Init_Ports ();
       Init_UART ();
       Relay_Test ();
       //Motor_Init ();
       while (1){

              /* Check bits in UART data register and set/clear needed port bits: */
              Receive_Byte ();

              //if (bit_is_set (UDR, Relay2_bit))  // Not used at this time!
              //     Relay2_On ();
              //else
              //     Relay2_Off;

              if (bit_is_set (UDR, Relay1_bit))
                     Relay_On ();
              else
                     Relay_Off ();

              if (bit_is_set (UDR, Data4_bit))
                     sbi (PORTD, 5);
              else
                     cbi (PORTD, 5);

              if (bit_is_set (UDR, Data3_bit))
                     sbi (PORTD, 4);
              else
                     cbi (PORTD, 4);

              if (bit_is_set (UDR, Data2_bit))
                     sbi (PORTD, 3);
              else
                     cbi (PORTD, 3);

              if (bit_is_set (UDR, Data1_bit))
                     sbi (PORTD, 2);
              else
                     cbi (PORTD, 2);

              if (bit_is_set (UDR, Motor2_Sel_bit))
                     sbi (PORTB, 0);
              else
                     cbi (PORTB, 0);

              if (bit_is_set (UDR, Motor1_Sel_bit))
                     sbi (PORTB, 1);
              else
                     cbi (PORTB, 1);

              /* Check switch position and start transmitting: */
```

```
                if (bit_is_set (PINB, Switch1))
                        TX_Reg |= Switch1_bit;
                else
                        TX_Reg &= ~Switch1_bit;

                if (bit_is_set (PINB, Switch2))
                        TX_Reg |= Switch2_bit;
                else
                        TX_Reg &= ~Switch2_bit;

                if (bit_is_set (PINB, Switch3))
                        TX_Reg |= Switch3_bit;
                else
                        TX_Reg &= ~Switch3_bit;

                if (bit_is_set (PINB, Switch4))
                        TX_Reg |= Switch4_bit;
                else
                        TX_Reg &= ~Switch4_bit;

                if (bit_is_set (PINB, Switch5))
                        TX_Reg |= Switch5_bit;
                else
                        TX_Reg &= ~Switch5_bit;

                if (bit_is_set (PIND, Switch6))
                        TX_Reg |= Switch6_bit;
                else
                        TX_Reg &= ~Switch6_bit;
                        TX_Reg |= Board_Reset;

                Transmit_Byte (TX_Reg);

                }

        while (1);              // Loop for ever (never reached!)
}
/* End of main program */

/* Initialize all I/O ports */
void Init_Ports (void)
{
        outp(0x07,DDRB);        // all pins on port B are inputs except PINB0-PINB2
        outp(0xbe,DDRD);        // all pins on port D are outputs except PIND0 and PIND6
        portb=0;
        portd=0;
}

/* Delay routine for overcome the switch bounce: */
void Delay_Step (void)
{
        int i,j,k;
        for (i=0;i<255;i++)
                for (j=0;j<80;j++)
                        k++;
        return;
}

/* Delay routine needed for motor steps: */
void Delay_Motor (void)
{
        int i,j,k;
        for (i=0;i<255;i++)
                for (j=0;j<50;j++)
                        k++;
        return;
}

/* Delay routine needed for Relay_Test: */
void Delay_Relay (void)
{
        int i,j,k=0;
        for (i=0;i<255;i++)
                for (j=0;j<255;j++)
                        k++;
                        k++;
                        k++;
        return;
```

```
}

/* Initialize UART (9600 baud @ 10 MHz): */
void Init_UART (void)
{
       TX_Reg = 0;
       outp (64,UBRR);
       outp ((1<<RXEN)|(1<<TXEN),UCR);      /* Enable TX and RX */
       return;
}

/* Transmit 1 byte: */
void Transmit_Byte (char data)
{
       while (bit_is_clear(USR,UDRE));
       outp (data, UDR);
       return;
}

/* Receive 1 byte: */
char Receive_Byte (void)
{
       char data;
       while (!bit_is_set(USR,RXC));
       data=inp (UDR);
       return data;
}

/* Turn relay on: */
void Relay_On (void)
{
       sbi (PORTB,Relay);
       return;
}

/* Turn relay off: */
void Relay_Off (void)
{
       cbi (PORTB,Relay);
       return;
}

/* Switch relay on and off: */
void Relay_Test (void)
{
       Relay_On ();
       Delay_Relay ();
       Delay_Relay ();
       Relay_Off ();
       return;
}

/* Setup all stepper motors for initial positions: */
void Motor_Init (void)
{
       TX_Reg |= Board_Reset;        // Set Board_Reset bit in TX_Reg
       Transmit_Byte (TX_Reg);       // Start transmitting

       portb |= Motor1;      // Start with Motor1
       portb &= ~Motor3;
       outp (portb, PORTB);
       while (!(bit_is_clear(PIND, Switch6))){     //TBD!!
              portd |= Step1L;
              portd &= ~Step4L;
              outp (portd,PORTD);
              Delay_Step ();
              portd |= Step2L;
              portd &= ~Step1L;
              outp (portd,PORTD);
              Delay_Step ();
              portd |= Step3L;
              portd &= ~Step2L;
              outp (portd,PORTD);
              Delay_Step ();
              portd |= Step4L;
              portd &= ~Step3L;
              outp (portd,PORTD);
              Delay_Step ();
```

```
        }
        TX_Reg |= Switch6_bit;        // Set Switch6_bit in TX_Reg
        Transmit_Byte (TX_Reg);

        portb |= Motor2;      // Start with Motor2
        portb &= ~Motor1;
        outp (portb,PORTB);
        while (!(bit_is_clear(PINB,Switch4))){      //TBD!!
               portd |= Step1L;
               portd &= ~Step4L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step2L;
               portd &= ~Step1L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step3L;
               portd &= ~Step2L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step4L;
               portd &= ~Step3L;
               outp (portd,PORTD);
               Delay_Step ();
        }
        TX_Reg |= Switch5_bit;        // Set Switch5_bit in TX_Reg
        Transmit_Byte (TX_Reg);

        portb |= Motor3;      // Start with Motor3
        portb &= ~Motor2;
        outp (portb,PORTB);
        while (1){            //TBD!!
               portd |= Step1L;
               portd &= ~Step4L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step2L;
               portd &= ~Step1L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step3L;
               portd &= ~Step2L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step4L;
               portd &= ~Step3L;
               outp (portd,PORTD);
               Delay_Step ();
        }
        TX_Reg |= Switch4_bit;        // Set Switch6_bit in TX_Reg
        Transmit_Byte (TX_Reg);

        /* Not used in motor setup!
        portb |= Motor4;      // Start with Motor4
        outp (portb,PORTB);
        while (1){
               portd |= Step1L;
               portd &= ~Step4L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step2L;
               portd &= ~Step1L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step3L;
               portd &= ~Step2L;
               outp (portd,PORTD);
               Delay_Step ();
               portd |= Step4L;
               portd &= ~Step3L;
               outp (portd,PORTD);
               Delay_Step ();
        }
        */
        return;
}
```

9.2 Izpis izvorne kode krmilnega dela

```
/*
=====================================================================
 Title:        SYSTEM CONTROLLER UNIT
 Author:       Damjan Stritof (damjan.stritof@campus.fe.uni-lj.si)
 Date:         1.4.2001
 Purpose:      SW for keyboard and LCD interface with RS232 port and
               one 8 bit bidirectional port
 Updated:      8.6.2001
=====================================================================
*/

#include <io.h>
#include <io8515.h>
#include "global.h"
#include "timer.h"
#include "lcd.h"

/* Definitions */
#define Piezo  PIND5

/* RS232 TX bits */
#define Relay2_bit    0x80    // Not used!
#define Relay1_bit    0x40
#define Data4_bit     0x20
#define Data3_bit     0x10
#define Data2_bit     0x08
#define Data1_bit     0x04
#define Motor2_Sel_bit0x02
#define Motor1_Sel_bit0x01

/* RS232 RX bits */
#define Board_Reset   0x80
#define Board_Error   0x40    // Not used!
#define Switch6_bit   0x20
#define Switch5_bit   0x10
#define Switch4_bit   0x08
#define Switch3_bit   0x04
#define Switch2_bit   0x02
#define Switch1_bit   0x01

/* Keyboard connections */
#define Col1   PINA0  // Columns
#define Col2   PINA1
#define Col3   PINA2
#define Col4   PINA3
#define Row1   PINA7  // Rows
#define Row2   PINA6
#define Row3   PINA5
#define Row4   PINA4

#define NO_KEY ' '
#define SEL_COL_0   { cbi(PORTA, 3); cbi(PORTA, 2); cbi(PORTA, 1); sbi(PORTA, 0); }
#define SEL_COL_1   { cbi(PORTA, 3); cbi(PORTA, 2); sbi(PORTA, 1); cbi(PORTA, 0); }
#define SEL_COL_2   { cbi(PORTA, 3); sbi(PORTA, 2); cbi(PORTA, 1); cbi(PORTA, 0); }
#define SEL_COL_3   { sbi(PORTA, 3); cbi(PORTA, 2); cbi(PORTA, 1); cbi(PORTA, 0); }

const unsigned char Keys[4][4] =
{ {'1', '2', '3', 'A' },
  {'4', '5', '6', 'B' },
  {'7', '8', '9', 'C' },
  {'*', '0', '#', 'D' } };

/* Global variables: */
u08    TX_Reg;        // UART RX register
u08    RX_Reg;        // UART TX register
u08    tipka;
u08    M1_step_positive;     // motor steps
u08    M2_step_positive;
u08    M3_step_positive;
u08    M1_step_negative;
u08    M2_step_negative;
u08    M3_step_negative;

/* Function prototypes */
void Init_Ports(void);
```

```
void Init_LCD(void);
void Init_UART(void);
void Piezo_beep(void);
void Transmit_Byte(char data);
char Receive_Byte(void);
unsigned char Scan_keyboard (void);
void Relay_On (void);
void Relay_Off (void);
void M1_rotate_right (void);
void M2_rotate_right (void);
void M3_rotate_right (void);
void M1_rotate_left (void);
void M2_rotate_left (void);
void M3_rotate_left (void);

/* Start of main program */
int main(void)
{
	TX_Reg=0;
	M1_step_positive=4;
	M2_step_positive=4;
	M3_step_positive=4;
	M1_step_negative=32;
	M2_step_negative=32;
	M3_step_negative=32;

	Init_Ports();
	Piezo_beep();
	Init_LCD();
	Init_UART();

	while (1){
		tipka=Scan_keyboard();
		switch (tipka){
			case '1': M1_rotate_right();
			case '4': M1_rotate_left();
			case '2': M2_rotate_left();
			case '5': M2_rotate_left();
			case '3': M3_rotate_left();
			case '6': M3_rotate_left();
			case '7': break;
			case '8': break;
			case '9': break;
			case '*': break;
			case '#': break;
			case 'A': Relay_On(); break;
			case 'B': Relay_Off(); break;
			case 'C': break;
			case 'D': break;
		}
	}
/* End of main program */
}


/* Initialize all ports */
void Init_Ports(void)
{
	outp(0x0F,DDRA);	// outputs: PINA0..PINA3, inputs: PINA4..PINA7
	outp(0xFF,DDRC);	// all pins are outputs
}

/* Initialize LCD: */
void Init_LCD (void)
{
	u08 counter;
	lcd_init((0<<LCD_ON_CURSOR),LCD_FUNCTION_DEFAULT);
	lcd_clrscr();
	lcd_gotoxy(0,0);
	lcd_puts("  AVR-SYS v1.0  ");
	lcd_gotoxy(0,1);
	lcd_puts("D. Stritof, 2001");
	for (counter=0; counter<255; counter++){
		delay(2000);
	}
	lcd_clrscr();
	return;
}
```

```
/* Initialize UART (9600 baud @ 8 MHz): */
void Init_UART (void)
{
        TX_Reg=0;
        outp(51,UBRR);
        outp((1<<RXEN)|(1<<TXEN),UCR);       // Enable RX and TX
        return;
}

/* Transmit 1 byte: */
void Transmit_Byte (char data)
{
        while (bit_is_clear(USR,UDRE));
        outp (data, UDR);
        return;
}

/* Receive 1 byte: */
char Receive_Byte (void)
{
        char data;
        while (!bit_is_set(USR,RXC));
        data=inp (UDR);
        return data;
}

/* Short beep sound: */
void Piezo_beep(void)
{
        u08 beep;
        for (beep=0; beep<5; beep++){
                sbi(PORTD,Piezo);
                delay(1500);
                cbi(PORTD,Piezo);
                delay(1500);
        }
        return;
}

/* Routine for scanning keyboard: */
unsigned char Scan_keyboard (void)
{
        unsigned char row, col;

        /* Select column 0 and scan all rows */
        SEL_COL_0;

        if (bit_is_set(PINA, Row1))
                delay(200);
        if (bit_is_set(PINA, Row1)){
                row=0;
                col=0;
                return Keys[row][col];
        }

        if (bit_is_set(PINA, Row2))
                delay(200);
        if (bit_is_set(PINA, Row2)){
                row=1;
                col=0;
                return Keys[row][col];
        }

        if (bit_is_set(PINA, Row3))
                delay(200);
        if (bit_is_set(PINA, Row3)){
                row=2;
                col=0;
                return Keys[row][col];
        }

        if (bit_is_set(PINA, Row4))
                delay(200);
        if (bit_is_set(PINA, Row4)){
                row=3;
                col=0;
                return Keys[row][col];
```

```
}

/* Select column 1 and scan all rows */
SEL_COL_1;

if (bit_is_set(PINA, Row1))
       delay(200);
if (bit_is_set(PINA, Row1)){
       row=0;
       col=1;
       return Keys[row][col];
}

if (bit_is_set(PINA, Row2))
       delay(200);
if (bit_is_set(PINA, Row2)){
       row=1;
       col=1;
       return Keys[row][col];
}

if (bit_is_set(PINA, Row3))
       delay(200);
if (bit_is_set(PINA, Row3)){
       row=2;
       col=1;
       return Keys[row][col];
}

if (bit_is_set(PINA, Row4))
       delay(200);
if (bit_is_set(PINA, Row4)){
       row=3;
       col=1;
       return Keys[row][col];
}

/* Select column 2 and scan all rows */
SEL_COL_2;

if (bit_is_set(PINA, Row1))
       delay(200);
if (bit_is_set(PINA, Row1)){
       row=0;
       col=2;
       return Keys[row][col];
}

if (bit_is_set(PINA, Row2))
       delay(200);
if (bit_is_set(PINA, Row2)){
       row=1;
       col=2;
       return Keys[row][col];
}

if (bit_is_set(PINA, Row3))
       delay(200);
if (bit_is_set(PINA, Row3)){
       row=2;
       col=2;
       return Keys[row][col];
}

if (bit_is_set(PINA, Row4))
       delay(200);
if (bit_is_set(PINA, Row4)){
       row=3;
       col=2;
       return Keys[row][col];
}

/* Select column 3 and scan all rows */
SEL_COL_3;

if (bit_is_set(PINA, Row1))
       delay(200);
if (bit_is_set(PINA, Row1)){
```

```
                row=0;
                col=3;
                return Keys[row][col];
        }

        if (bit_is_set(PINA, Row2))
                delay(200);
        if (bit_is_set(PINA, Row2)){
                row=1;
                col=3;
                return Keys[row][col];
        }

        if (bit_is_set(PINA, Row3))
                delay(200);
        if (bit_is_set(PINA, Row3)){
                row=2;
                col=3;
                return Keys[row][col];
        }

        if (bit_is_set(PINA, Row4))
                delay(200);
        if (bit_is_set(PINA, Row4)){
                row=3;
                col=3;
                return Keys[row][col];
        }
        return NO_KEY;
}

void Relay_On (void)
{
        TX_Reg |= Relay1_bit;
        Transmit_Byte(TX_Reg);
        delay(100000);
        return;
}

void Relay_Off (void)
{
        TX_Reg &= ~Relay1_bit;
        Transmit_Byte (TX_Reg);
        delay(100000);
        return;
}

void M1_rotate_right (void)
{
        TX_Reg &= ~Motor1_Sel_bit;    // select M1
        TX_Reg &= ~Motor2_Sel_bit;
         Transmit_Byte (TX_Reg);

        TX_Reg |= M1_step_positive;

        if (M1_step_positive == 4){
                TX_Reg &= ~(8|16|32);
                M1_step_positive = 8;
         }
        if (M1_step_positive == 8){
                TX_Reg &= ~(4|16|32);
                M1_step_positive = 16;
         }
        if (M1_step_positive == 16){
                TX_Reg &= ~(4|8|32);
                M1_step_positive = 32;
         }
        if (M1_step_positive == 32){
                 TX_Reg &= ~(4|8|16);
                M1_step_positive = 4;
         }

         Transmit_Byte (TX_Reg);
         delay(50000);
}

void M2_rotate_right (void)
{
```

```
        TX_Reg &= ~Motor1_Sel_bit;    // select M2
        TX_Reg |= Motor2_Sel_bit;
         Transmit_Byte (TX_Reg);

        TX_Reg |= M2_step_positive;

        if (M2_step_positive == 4){
               TX_Reg &= ~(8|16|32);
               M2_step_positive = 8;
         }
        if (M2_step_positive == 8){
               TX_Reg &= ~(4|16|32);
               M2_step_positive = 16;
         }
        if (M2_step_positive == 16){
               TX_Reg &= ~(4|8|32);
               M2_step_positive = 32;
         }
        if (M2_step_positive == 32){
                TX_Reg &= ~(4|8|16);
               M2_step_positive = 4;
         }

         Transmit_Byte (TX_Reg);
         delay(50000);
}

void M3_rotate_right (void)
{
        TX_Reg |= Motor1_Sel_bit;     // select M3
        TX_Reg &= ~Motor2_Sel_bit;
         Transmit_Byte (TX_Reg);

        TX_Reg |= M3_step_positive;

        if (M3_step_positive == 4){
               TX_Reg &= ~(8|16|32);
               M3_step_positive = 8;
         }
        if (M3_step_positive == 8){
               TX_Reg &= ~(4|16|32);
               M3_step_positive = 16;
         }
        if (M3_step_positive == 16){
               TX_Reg &= ~(4|8|32);
               M3_step_positive = 32;
         }
        if (M3_step_positive == 32){
                TX_Reg &= ~(4|8|16);
               M3_step_positive = 4;
         }

         Transmit_Byte (TX_Reg);
         delay(50000);
}

void M1_rotate_left (void)
{
        TX_Reg &= ~Motor1_Sel_bit;    // select M1
        TX_Reg &= ~Motor2_Sel_bit;
         Transmit_Byte (TX_Reg);

        TX_Reg |= M1_step_negative;

        if (M1_step_negative == 4){
               TX_Reg &= ~(8|16|32);
               M1_step_negative = 32;
         }
        if (M1_step_negative == 8){
               TX_Reg &= ~(4|16|32);
               M1_step_negative = 4;
         }
        if (M1_step_negative == 16){
               TX_Reg &= ~(4|8|32);
               M1_step_negative = 8;
         }
        if (M1_step_negative == 32){
                TX_Reg &= ~(4|8|16);
```

```
                M1_step_negative = 16;
        }

        Transmit_Byte (TX_Reg);
        delay(50000);
}

void M2_rotate_left (void)
{
        TX_Reg &= ~Motor1_Sel_bit;    // select M2
        TX_Reg |= Motor2_Sel_bit;
        Transmit_Byte (TX_Reg);

        TX_Reg |= M1_step_negative;

        if (M2_step_negative == 4){
                TX_Reg &= ~(8|16|32);
                M2_step_negative = 32;
        }
        if (M2_step_negative == 8){
                TX_Reg &= ~(4|16|32);
                M2_step_negative = 4;
        }
        if (M2_step_negative == 16){
                TX_Reg &= ~(4|8|32);
                M2_step_negative = 8;
        }
        if (M2_step_negative == 32){
                TX_Reg &= ~(4|8|16);
                M2_step_negative = 16;
        }

        Transmit_Byte (TX_Reg);
        delay(50000);
}

void M3_rotate_left (void)
{
        TX_Reg |= Motor1_Sel_bit;     // select M3
        TX_Reg &= ~Motor2_Sel_bit;
        Transmit_Byte (TX_Reg);

        TX_Reg |= M3_step_negative;

        if (M3_step_negative == 4){
                TX_Reg &= ~(8|16|32);
                M3_step_negative = 32;
        }
        if (M3_step_negative == 8){
                TX_Reg &= ~(4|16|32);
                M3_step_negative = 4;
        }
        if (M3_step_negative == 16){
                TX_Reg &= ~(4|8|32);
                M3_step_negative = 8;
        }
        if (M3_step_negative == 32){
                TX_Reg &= ~(4|8|16);
                M3_step_negative = 16;
        }

        Transmit_Byte (TX_Reg);
        delay(50000);
}
```

9.3 Izpis programske kode v Visual Basicu

```
'RS232 registers'
Dim TX_register As Integer
Dim RX_register As Variant

'Motor start, end and current positions'
Dim M1_start_position As Integer
Dim M2_start_position As Integer
Dim M3_start_position As Integer
Dim M1_stop_position As Integer
Dim M2_stop_position As Integer
Dim M3_stop_position As Integer
Dim M1_current_position As Integer
Dim M2_current_position As Integer
Dim M3_current_position As Integer

'RS232 TX bits'
Const Relay2_bit = 128
Const Relay1_bit = 64
Const Data4_bit = 32
Const Data3_bit = 16
Const Data2_bit = 8
Const Data1_bit = 4
Const Motor2_sel_bit = 2
Const Motor1_sel_bit = 1

'Direction bits'
Dim Step_positive As Integer
Dim Step_negative As Integer

'Position of motors'
Dim Steps_left As Integer
Dim Steps_right As Integer
Dim Direction As String
Dim Relay_state As Integer
Dim Current_motor As Integer

'Send e-mail'
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal
hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As
String, ByVal lpDirectory As String, ByVal lpShowCmd As Long) As Long

Sub Delay()

    PauseTime = Slider1.Value * 0.01
    Start = Timer
    Do While Timer < Start + PauseTime
        DoEvents
    Loop

End Sub

Sub Rotate_left()

    TX_register = TX_register Xor Step_negative
    If Step_negative = 4 Then
        TX_register = TX_register And Not (8 Or 16 Or 32)
        Step_negative = 32
    ElseIf Step_negative = 8 Then
        TX_register = TX_register And Not (4 Or 16 Or 32)
        Step_negative = 4
    ElseIf Step_negative = 16 Then
        TX_register = TX_register And Not (4 Or 8 Or 32)
        Step_negative = 8
    ElseIf Step_negative = 32 Then
        TX_register = TX_register And Not (4 Or 8 Or 16)
        Step_negative = 16
    Else
    End If

    If Current_motor = 1 Then
        M1_current_position = M1_current_position - 1
    ElseIf Current_motor = 2 Then
        M2_current_position = M2_current_position - 1
    ElseIf Current_motor = 3 Then
        M3_current_position = M3_current_position - 1
```

```
    End If

    Call Delay

End Sub

Sub Rotate_right()

    TX_register = TX_register Xor Step_positive
    If Step_positive = 4 Then
        TX_register = TX_register And Not (8 Or 16 Or 32)
        Step_positive = 8
    ElseIf Step_positive = 8 Then
        TX_register = TX_register And Not (4 Or 16 Or 32)
        Step_positive = 16
    ElseIf Step_positive = 16 Then
        TX_register = TX_register And Not (4 Or 8 Or 32)
        Step_positive = 32
    ElseIf Step_positive = 32 Then
        TX_register = TX_register And Not (4 Or 8 Or 16)
        Step_positive = 4
    Else
    End If

    If Current_motor = 1 Then
        M1_current_position = M1_current_position + 1
    ElseIf Current_motor = 2 Then
        M2_current_position = M2_current_position + 1
    ElseIf Current_motor = 3 Then
        M3_current_position = M3_current_position + 1
    End If

    Call Delay

End Sub

Private Sub Command1_Click()
    'Exit button: close RS232 and disable timers'
    TX_register = 0
    MSComm1.Output = Chr(TX_register)

    If MSComm1.PortOpen = True Then
        MSComm1.PortOpen = False
    End If

    'Stop timers'
    Timer1.Enabled = False
    'Timer2.Enabled = False

    End
End Sub

Private Sub Command2_Click()
    If Relay_state = 0 Then
        TX_register = TX_register Xor Relay1_bit
        Shape1.BackColor = &HFF00&
        Relay_state = 1
    Else
        TX_register = TX_register And Not Relay1_bit
        Shape1.BackColor = &HFF&
        Relay_state = 0
    End If

End Sub

Private Sub Command3_KeyPress(KeyAscii As Integer)

    'Rotate selected motor to positive direction'
    TX_register = TX_register Xor Step_positive
    If Step_positive = 4 Then
        TX_register = TX_register And Not (8 Or 16 Or 32)
        Step_positive = 8
    ElseIf Step_positive = 8 Then
        TX_register = TX_register And Not (4 Or 16 Or 32)
        Step_positive = 16
    ElseIf Step_positive = 16 Then
        TX_register = TX_register And Not (4 Or 8 Or 32)
        Step_positive = 32
```

```
    ElseIf Step_positive = 32 Then
        TX_register = TX_register And Not (4 Or 8 Or 16)
        Step_positive = 4
    Else
    End If

    If Current_motor = 1 Then
        M1_current_position = M1_current_position + 1
    ElseIf Current_motor = 2 Then
        M2_current_position = M2_current_position + 1
    ElseIf Current_motor = 3 Then
        M3_current_position = M3_current_position + 1
    End If

End Sub

Private Sub Command4_KeyPress(KeyAscii As Integer)

    'Rotate selected motor to negative direction'
    TX_register = TX_register Xor Step_negative
    If Step_negative = 4 Then
        TX_register = TX_register And Not (8 Or 16 Or 32)
        Step_negative = 32
    ElseIf Step_negative = 8 Then
        TX_register = TX_register And Not (4 Or 16 Or 32)
        Step_negative = 4
    ElseIf Step_negative = 16 Then
        TX_register = TX_register And Not (4 Or 8 Or 32)
        Step_negative = 8
    ElseIf Step_negative = 32 Then
        TX_register = TX_register And Not (4 Or 8 Or 16)
        Step_negative = 16
    Else
    End If

    If Current_motor = 1 Then
        M1_current_position = M1_current_position - 1
    ElseIf Current_motor = 2 Then
        M2_current_position = M2_current_position - 1
    ElseIf Current_motor = 3 Then
        M3_current_position = M3_current_position - 1
    End If

End Sub

Private Sub Command5_Click()
    Dim i, j, k As Integer

    M1_start_position = CInt(Text1.Text)
    M1_stop_position = CInt(Text2.Text)
    M2_start_position = CInt(Text4.Text)
    M2_stop_position = CInt(Text5.Text)
    M3_start_position = CInt(Text7.Text)
    M3_stop_position = CInt(Text8.Text)

    ' Rotate Motor 1 to start position'
    TX_register = TX_register And Not Motor1_sel_bit
    TX_register = TX_register And Not Motor2_sel_bit
    Current_motor = 1

    If M1_current_position > M1_start_position Then
        For i = 1 To (M1_current_position - M1_start_position)
            Call Rotate_left
        Next i
    Else
    End If

    If M1_current_position < M1_start_position Then
        For i = 1 To (M1_start_position - M1_current_position)
            Call Rotate_right
        Next i
    Else
    End If

    TX_register = TX_register And Not Motor1_sel_bit
    TX_register = TX_register And Not Motor2_sel_bit

    ' Rotate Motor 3 to start position'
```

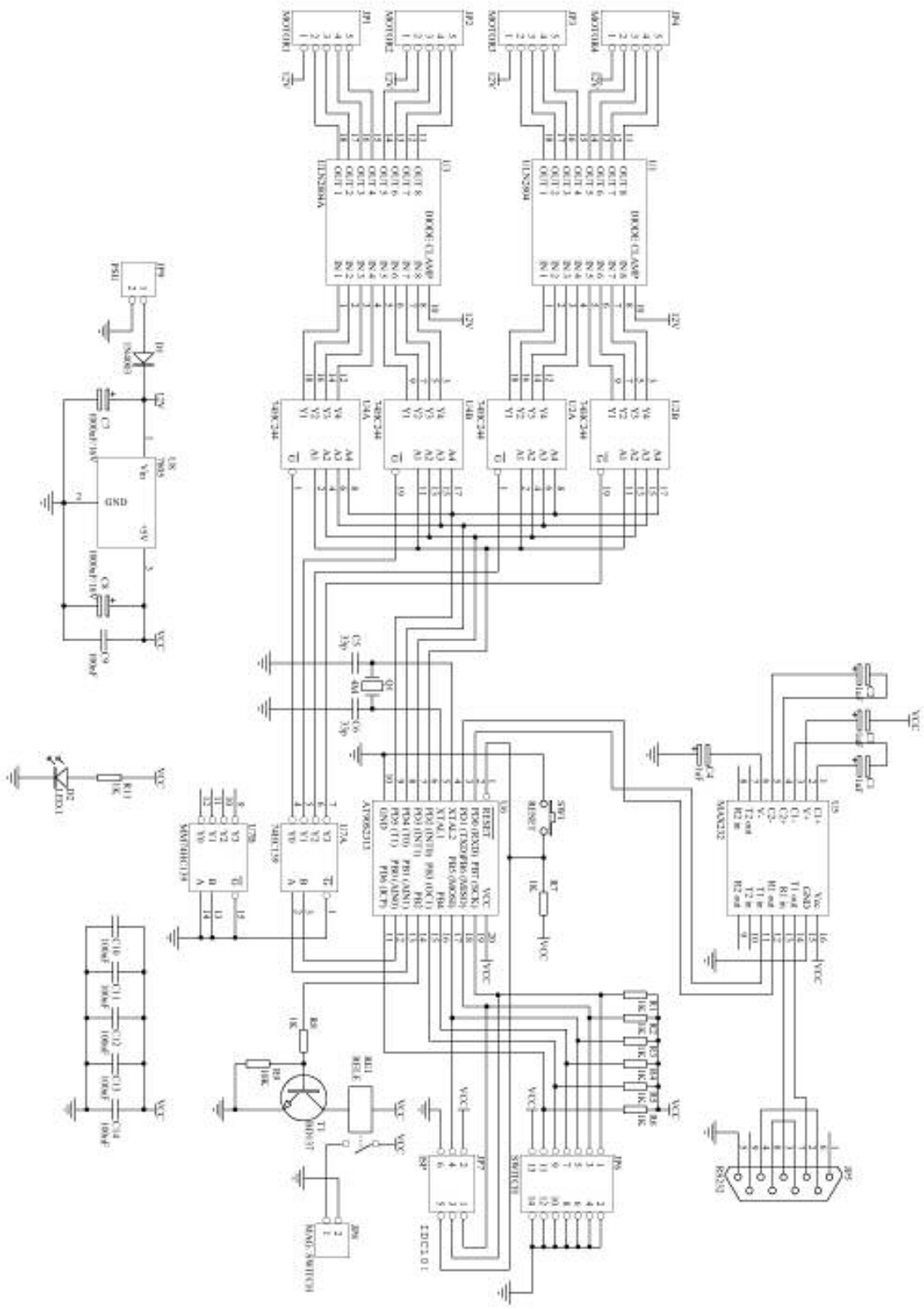
```
TX_register = TX_register Xor Motor1_sel_bit
TX_register = TX_register And Not Motor2_sel_bit
Current_motor = 3

If M3_current_position > M3_start_position Then
    For i = 1 To (M3_current_position - M3_start_position)
        Call Rotate_left
    Next i
Else
End If

If M3_current_position < M3_start_position Then
    For i = 1 To (M3_start_position - M3_current_position)
        Call Rotate_right
    Next i
Else
End If

TX_register = TX_register And Not Motor1_sel_bit
TX_register = TX_register And Not Motor2_sel_bit

' Rotate Motor 2 to start position'
TX_register = TX_register And Not Motor1_sel_bit
TX_register = TX_register Xor Motor2_sel_bit
Current_motor = 2

If M2_current_position > M2_start_position Then
    For i = 1 To (M2_current_position - M2_start_position)
        Call Rotate_left
    Next i
Else
End If

If M2_current_position < M2_start_position Then
    For i = 1 To (M2_start_position - M2_current_position)
        Call Rotate_right
    Next i
Else
End If

TX_register = TX_register And Not Motor1_sel_bit
TX_register = TX_register And Not Motor2_sel_bit

'Switch Relay ON'
If Relay_state = 0 Then
    TX_register = TX_register Xor Relay1_bit
    Shape1.BackColor = &HFF00&
    Relay_state = 1
Else
    MsgBox ("Relay is already switched ON!")
    Shape1.BackColor = &HFF&
    Relay_state = 0
End If

'Wait some time'
Call Delay
Call Delay
Call Delay
Call Delay

'Rotating Motor 3'
TX_register = TX_register Xor Motor1_sel_bit
TX_register = TX_register And Not Motor2_sel_bit
Current_motor = 3

If M3_start_position < M3_stop_position Then
    For i = 1 To (M3_stop_position - M3_start_position)
        Call Rotate_right
    Next i
Else
End If

If M3_start_position > M3_stop_position Then
    For i = 1 To (M3_start_position - M3_stop_position)
        Call Rotate_left
    Next i
Else
End If
```

```
    'Rotating Motor 2'
    TX_register = TX_register And Not Motor1_sel_bit
    TX_register = TX_register Xor Motor2_sel_bit
    Current_motor = 2

    If M2_start_position < M2_stop_position Then
        For i = 1 To (M2_stop_position - M2_start_position)
            Call Rotate_right
        Next i
    Else
    End If

    If M2_start_position > M2_stop_position Then
        For i = 1 To (M2_start_position - M2_stop_position)
            Call Rotate_left
        Next i
    Else
    End If

    'Rotating Motor 1'
    TX_register = TX_register And Not Motor1_sel_bit
    TX_register = TX_register And Not Motor2_sel_bit
    Current_motor = 1

    If M1_start_position < M1_stop_position Then
        For i = 1 To (M1_stop_position - M1_start_position)
            Call Rotate_right
        Next i
    Else
    End If

    If M1_start_position > M1_stop_position Then
        For i = 1 To (M1_start_position - M1_stop_position)
            Call Rotate_left
        Next i
    Else
    End If

    'Wait some time'
    Call Delay
    Call Delay
    Call Delay
    Call Delay

    'Switch Relay OFF'
    TX_register = TX_register And Not Relay1_bit
    Shape1.BackColor = &HFF&
    Relay_state = 0

End Sub

Private Sub Command7_Click()
    If Timer1.Enabled = True Then
        Timer1.Enabled = False
        Command7.Caption = "&Start"
    ElseIf Timer1.Enabled = False Then
        Timer1.Enabled = True
        Command7.Caption = "&Stop"
    End If

End Sub

Private Sub Command8_Click()
    If Current_motor = 1 Then
        M1_start_position = M1_current_position
    ElseIf Current_motor = 2 Then
        M2_start_position = M2_current_position
    ElseIf Current_motor = 3 Then
        M3_start_position = M3_current_position
    End If

    Text1.Text = M1_start_position
    Text4.Text = M2_start_position
    Text7.Text = M3_start_position
End Sub

Private Sub Command9_Click()
```

```
    If Current_motor = 1 Then
        M1_stop_position = M1_current_position
    ElseIf Current_motor = 2 Then
        M2_stop_position = M2_current_position
    ElseIf Current_motor = 3 Then
        M3_stop_position = M3_current_position
    End If

    Text2.Text = M1_stop_position
    Text5.Text = M2_stop_position
    Text8.Text = M3_stop_position
End Sub

Private Sub Form_Load()

    'Check if other devices use COM port'
    If MSComm1.PortOpen = True Then
        MsgBox ("Error: this port is already in use!")
    End If

    'RS232 initialization: use COM1'
    MSComm1.CommPort = 1
    MSComm1.Settings = "9600,N,8,1"
    MSComm1.InputLen = 0
    MSComm1.PortOpen = True
    TX_register = 0

    'Timers'
    Timer1.Enabled = True
    'Timer2.Enabled = True

    'Relay is switched off'
    Relay_state = 0

    'Set Motor2'
    'Current_motor = 2

    Step_positive = 4
    Step_negative = 32

End Sub

Private Sub Frame1_DragDrop(Source As Control, X As Single, Y As Single)
    TX_register = TX_register Xor Motor1_sel_bit Xor Motor2_sel_bit
End Sub

Private Sub Label6_Click()
    Call ShellExecute(0&, vbNullString, "mailto:damjan.stritof@campus.fe.uni-
lj.si?subject=SC Info Request", vbNullString, vbNullString, vbNormalFocus)
End Sub

Private Sub Option1_Click()
    TX_register = TX_register And Not Motor1_sel_bit
    TX_register = TX_register And Not Motor2_sel_bit
    Current_motor = 1
End Sub

Private Sub Option2_Click()
    TX_register = TX_register And Not Motor1_sel_bit
    TX_register = TX_register Xor Motor2_sel_bit
    Current_motor = 2
End Sub

Private Sub Option3_Click()
    TX_register = TX_register Xor Motor1_sel_bit
    TX_register = TX_register And Not Motor2_sel_bit
    Current_motor = 3
End Sub

Private Sub Text1_Change()
    M1_start_position = Text1.Text
End Sub

Private Sub Text2_Change()
    M1_stop_position = Text2.Text
End Sub

Private Sub Text4_Change()
```

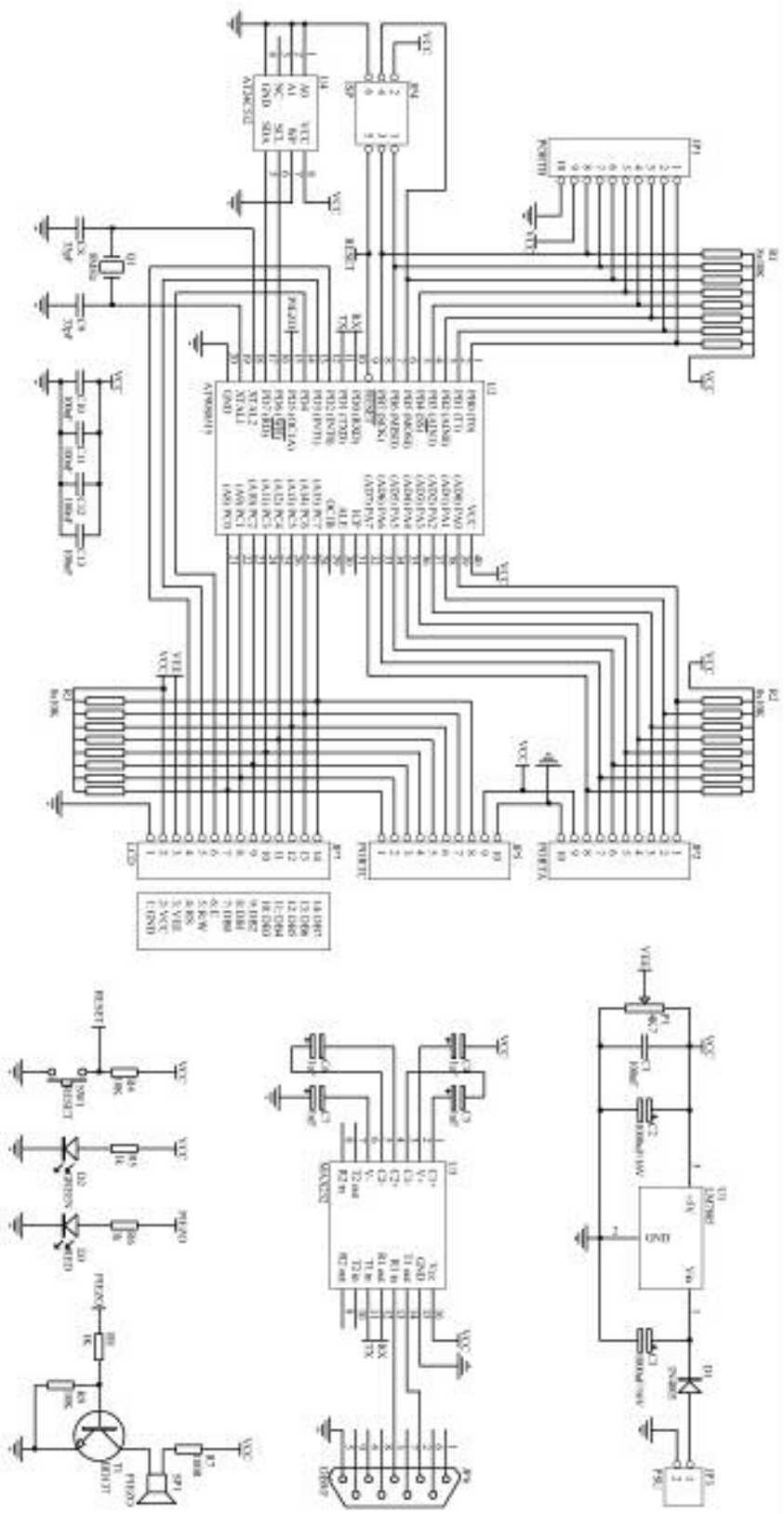
```
    M2_start_position = Text4.Text
End Sub

Private Sub Text5_Change()
    M2_stop_position = Text5.Text
End Sub

Private Sub Text7_Change()
    M3_start_position = Text7.Text
End Sub

Private Sub Text8_Change()
    M3_stop_position = Text8.Text
End Sub

Private Sub Timer1_Timer()
    'Main communication routine'
    MSComm1.Output = Chr(TX_register)
    RX_register = MSComm1.Input

    'Update position indicators'
    Text3.Text = M1_current_position
    Text6.Text = M2_current_position
    Text9.Text = M3_current_position
End Sub
```

9.4 Shematski načrt močnostnega dela



Slika 9-1: Shematski načrt močnostnega dela

9.5 Shematski načrt krmilnega dela



Slika 9-2: Shematski načrt krmilnega dela

Izjava

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja doc. dr. Andreja Žemve. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Damjan Štritof