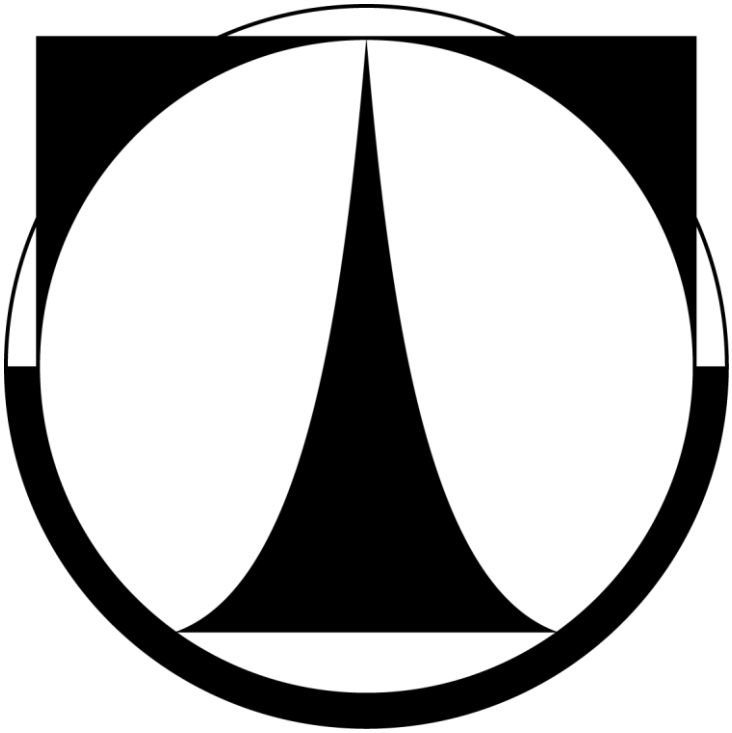
TECHNICKÁ UNIVERZITA V LIBERCI



FAKULTA STROJNÍ

KATEDRA APLIKOVANÉ KYBERNETIKY

SYSTÉM PRO ŘÍZENÍ POHYBU ANGULÁRNÍHO ROBOTA

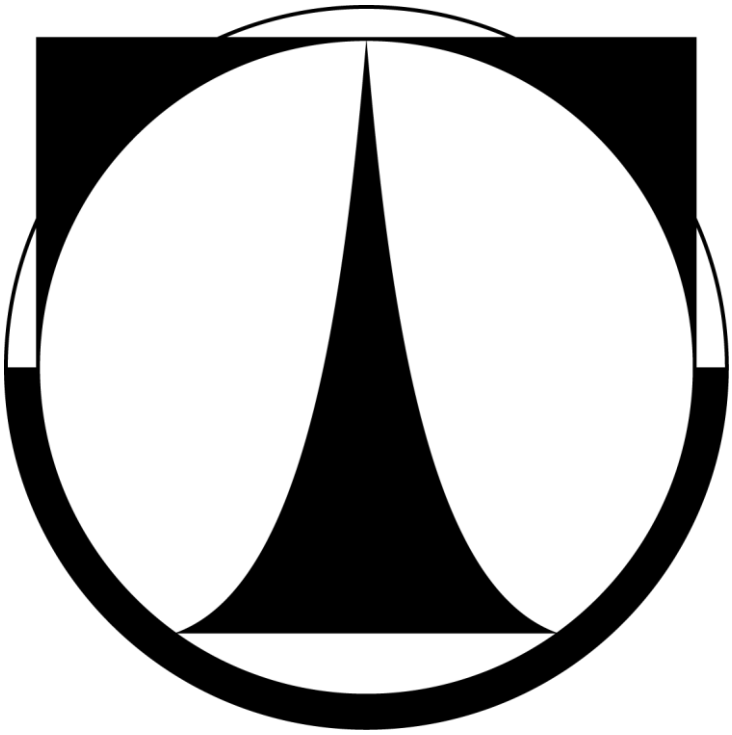
DIPLOMOVÁ PRÁCE

AUTOR PRÁCE: Iaroslav Kovalenko

VEDOUCI PRÁCE: Ing. Michal Moučka, Ph.D.

Liberec 2013

TECHNICAL UNIVERSITY OF LIBEREC



FACULTY OF MECHANICAL ENGINEERING

DEPARTMENT OF APPLIED CYBERNETICS

SYSTEM FOR MOTION CONTROL OF ANGULAR ROBOT

MASTER`S THESIS

AUTHOR: Iaroslav Kovalenko

SUPERVISOR: Ing. Michal Moučka, Ph.D.

Liberec 2013

ANOTACE

Obsahem této práce je návrh systému pro řízení pohybu laboratorního modelu angulárního robota, realizace řídicí elektroniky a výroba desky plošných spojů pro uskutečnění navrženého řešení.

V práci je podrobně popsán postup výroby řídicího systému s popisem základních modulů, včetně principů jejich práce v daném systému. Dále je velice podrobně popsané sériové rozhraní pro ovládání robota z jiného přístroje. V závěru práce jsem uvedl hlavní činnosti a možnosti daného systému a jeho klady pro řízení daného angulárního robota.

KLIČOVÁ SLOVA

Angulární robot, krokový motor, řídicí jednotka, mikroprocesor, řízeni krokových motorů

ANNOTATION

The topic of this thesis is to design system for motion control of laboratory model of angular robot, realization of control electronics and producing of circuit board for implementation proposed solution.

Produce process of control system was described in details with description of basics modules in this work. Then serial interface for robot controlling from another device was very detailed described. At the end of this work I specified the main actions and options of this system and its advantages for control of this angular robot.

KEY WORDS

Angular robot, step motor, control unit, microprocessor, step motors’ control

PROHLÁŠENÍ

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména §60 - školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat náhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce.

V Liberci dne _____

_____

Iaroslav Kovalenko

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK A POUŽITÝCH SYMBOLŮ

PIC – jednočipový mikroprocesor od firmy Microchip Technology

LPT – paralelní port pro přenášení několika bitu současně

COM – název sériového portu v počítači kompatibilním s IBM PC

USART – synchronní/asynchronní sériové rozhraní

SPI – sériové rozhraní

RISC – architektura mikroprocesoru

MSSP – modul mikroprocesoru pro podporu sériového rozhraní

I^2C – sériová sběrnice

LSB – nejméně významný bit

MSB – nejvýznamnější bit

Fosc – kmitočet mikroprocesoru

EUSART – modul mikroprocesoru pro podporu synchronního/asynchronního sériového rozhraní

GERBER – formát pro popis projektu výroby desky plošných spojů

IDE – vývojové prostředí

ÚVOD

V dnešní době už je robotická technika integrována do většiny výrob. Čím dál tím víc a víc závodů se snaží ušetřit peníze zavaděním úplně nebo částečně automatizovaných výrobních linek. Pro velké množství úkolů na těchto linkách, a také pro jednotlivé úlohy výroby, se používají roboty.

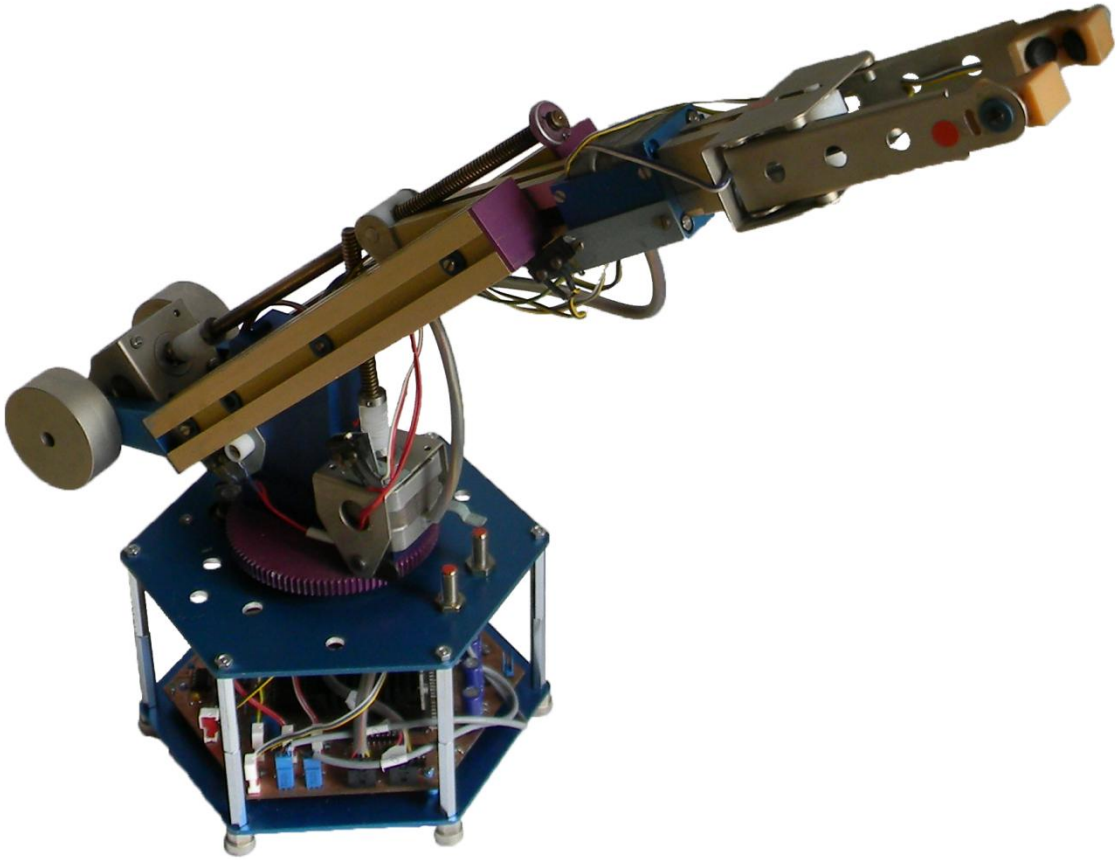
V současné době se nejvíce používají roboty angulárního typu, kolem 50% ze všech typů a tento počet stále roste. Proto je důležité seznámit studenty se základními znalostmi, které se tykají angulárních robotů. Například vyzkoušet možnosti pohybu robotu v pracovním prostoru, zjistit způsoby řízení pohonů robotů, a vyzkoušet některé z nich.

Proto jsem dostal, jako zadaní pro svou diplomovou práci, vytvoření systému pro řízení pohybu angulárního robota. Je to už docela starý manipulátor z roku 1991. Na jeho základě jsem musel vypracovat řídící desku a robot s touto deskou muže byt použit pro vyučování studentů.

Původní řídicí systém byl vytvořen na základě desky s tranzistory. Tento způsob ovládání měl spoustu nedostatků. Například mohl ovládat v jednom okamžiku jenom jeden motor. Polohy, ve kterých se mohl robot zastavovat, byly pevně určeny snímači. Také tento systém často přestával fungovat a musel být opraven.

Proto byl původní řídicí systém později nahrazen mikroprocesorovým systémem na bázi mikroprocesoru od firmy Intel 8051 (MCS-51). Tento robot byl součástí třídicího systému pro třídění materiálu. Po zavádění řídicího systému uměl řídit všechny motory současně a mohl se zastavovat v jakékoliv poloze svého pracovního prostoru. Nedostatkem tohoto systému ale bylo to, že byl pevně naprogramován na určitou úlohu (třídění materiálu). Také řešení na bázi mikroprocesoru 8051 už je hodně zastaralé. Proto jsem dostal za úkol vytvořit nový řídicí systém na základě novějšího mikroprocesoru PIC, který by mohl být ovládán z vnějšího přístroje, a mohl by se používat pro výuku studentů. Také jsem chtěl, aby se nový systém dal umístit uvnitř robota, aby byl více užitečný při laboratorních pracích.

1 POPIS ROBOTA



Obr. 1.1: Laboratorní angulární robot

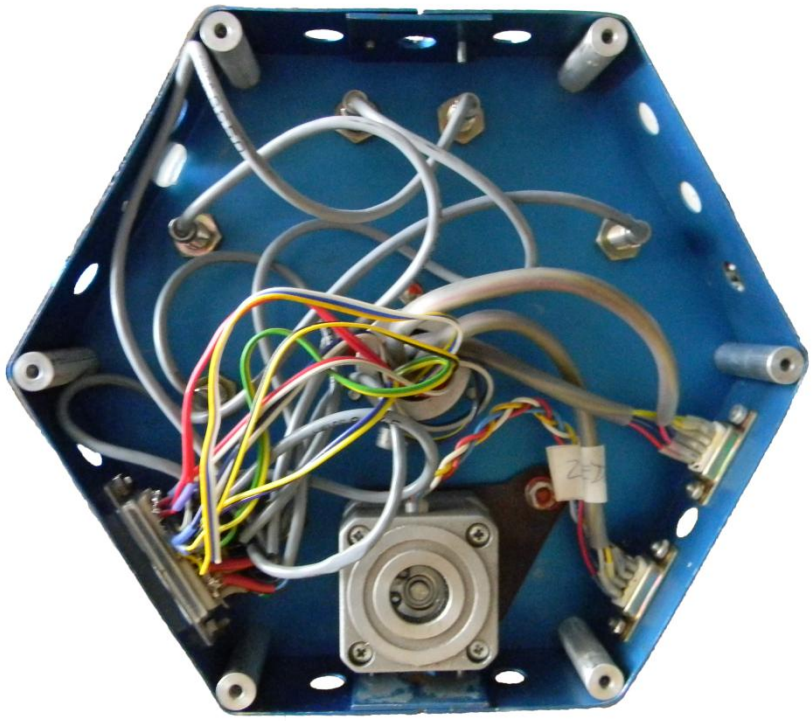
Robot, který je zobrazen na obrázku 1.1 je angulárního (antropomorfního, multiúhloveho) typu. To znamená, že se jeho polohovací ústrojí skládá ze třech rotačních jednotek. První rotační jednotka rotuje kolem svislé osy, a ostatní dvě jednotky rotují kolem rovnoběžné a vodorovné osy. Tyto roboty se mohou pohybovat po všech třech osách, a jejich pracovní prostor je velice pružný a závisí na rozsahu, hybnosti a rozměrech pohybových jednotek. Výhodou těchto robotu je jejich dobrá anatomie a tvar pracovního prostoru. Také jejich pohyblivost je dobrá a mají vysoký koeficient obslužnosti v pracovním prostoru. V současné době dosahuje celkový podíl robotů angulárního typu na celkovém počtu vyrobených robotů 50%. Většinou se používají pro technologické aplikace. [5]

Tento robot, který je zobrazen na obrázku 1.1, byl výroben v roce 1991 v Československu. Původně měl řídicí systém na bázi tranzistorové logiky. Řídicí jednotka se musela často opravovat a nevyužívala všechny možnosti robota. Například se mohl robot zastavovat jenom v určitých polohách, které byly pevně zadané snímači. Velkou nevýhodou také bylo to, že v jednom okamžiku se mohl pohybovat jenom jeden motor. Nebylo možné ovládat několik motorů současně.

Pro větší využití možností tohoto robota, byl v roce 2003 vytvořen nový řídicí systém na základě mikroprocesoru. Robot s novým řídicím systémem byl zapojen do třídicího systému, ve kterém přenášel materiál do pneumatického třídiče. Tento systém už uměl víc. Robot se mohl zastavovat v jakékoli poloze svého pracovního prostoru, dalo se ovládat několik motorů současně a mohl dostávat pevně určené příkazy z počítače a zapisovat je do vnitřní paměti. [8]

Největší nevýhoda tohoto robota spočívala v tom, že byl vytvořen pouze pro účely třídicího systému a nemohl se používat pro jiné aplikace. Dnes už je tento systém zastaralý a jen částečně funkční. Také nebyl dostatečně malý, aby mohl být umístěný dovnitř robota. Z tohoto důvodu se řídicí deska nacházela mimo robota.

Pro svůj řídicí systém jsem zvolil místo v dolní části robota, kde dříve byla spousta konektorů a kabelů. Je zobrazen na dolní části obr. 1.1. Původně tato část robota vypadala takto:



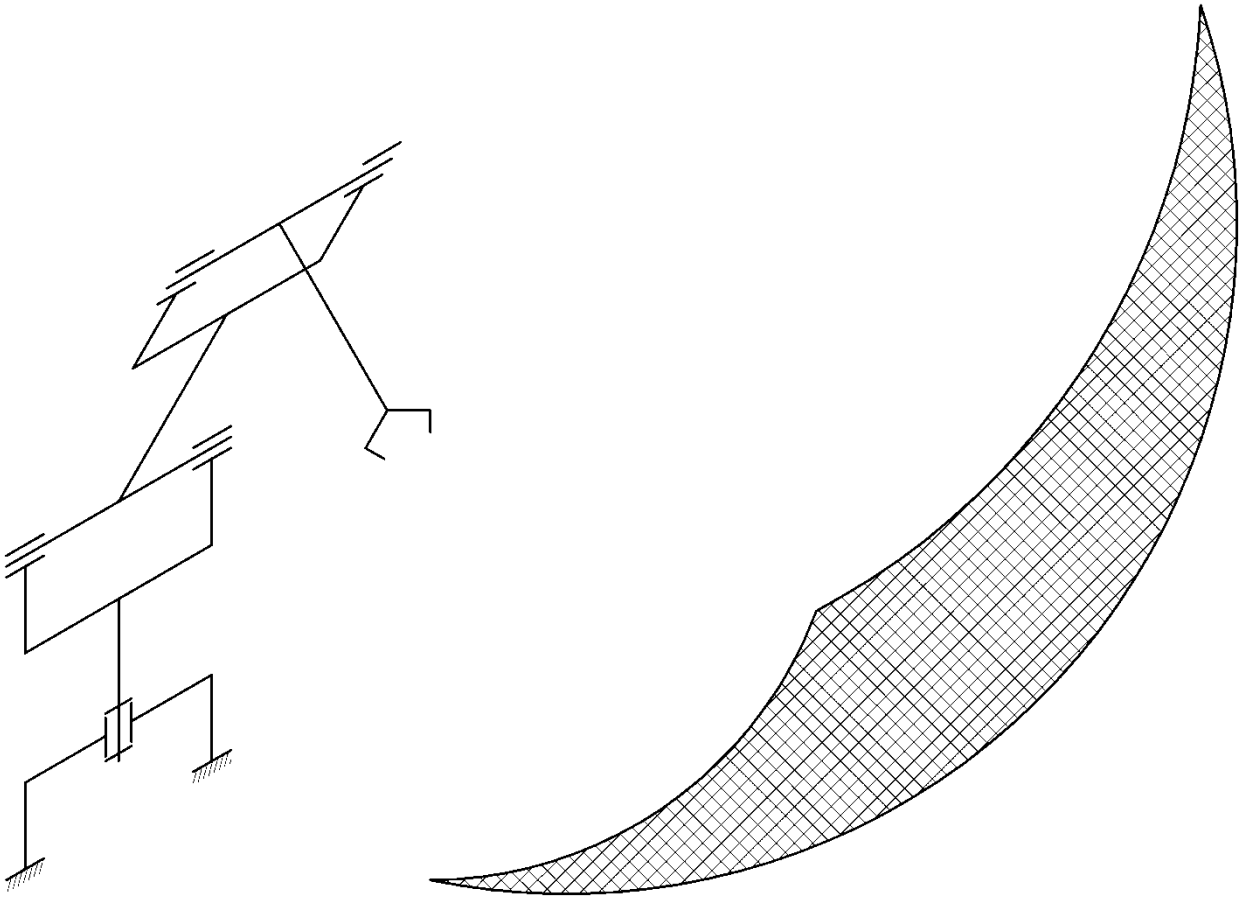
Obr. 1.2: Dolní krabice robota

Konektory čtyř motorů byli vyvedeny na dva 9-pinové konektory D-SUB (DE9M) – konektor jako v COM portu počítače. Z devíti pinů konektoru bylo použito osm. Konektory všech koncových spínačů a indukčních čidel byly vyvedené na jeden 25-pinový D-SUB (DB25M) – konektor jako v LPT portu počítače.

Řídicí deska se nacházela mimo robota a byla k němu připojena pomocí plošných kabelů.

Svoji desku jsem navrhoval tak, aby jí bylo možné umístit uvnitř krabice, uvedené na obrázku 1.2, uvnitř robota. Pro toto jsem se rozhodl proto, abych chránil desku před mechanickými a jinými typy poškození. Použitím tohoto způsobu se dá také vyhnout nutnosti používání dlouhých plošných kabelů pro připojení řídicího systému k robotu.

Na obrázku 1.1 je vidět, že jsem musel zvětšit dolní krabici, aby bylo možné umístit můj nový řídicí systém uvnitř robota. Hlavně ji ale bylo nutné zvětšit kvůli velkému chladiči.

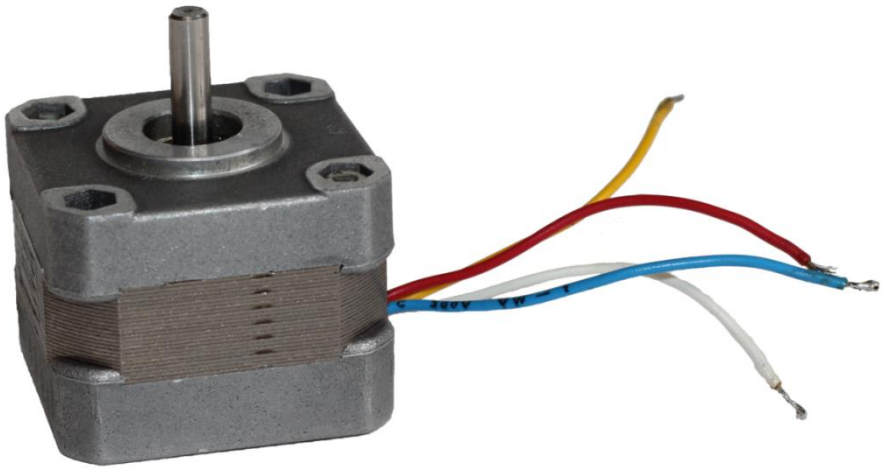


Obr. 1.3: Kinematické schéma a přibližný tvar pracovního prostoru robota ve 2-D

Na obrázku je vidět, že se robot skládá ze třech rotačních jednotek RRR a chapadla. Osa jedné z nich je svislá a zbývající dvě osy jsou vodorovné. První pohon se skládá s krokového elektromotoru a převodovky s ozubeným řemenem. Ostatní tři se skládají z krokových elektromotorů, pružných spojek, kuličkových šroubů a matic pro převádění rotačního pohybu motoru na posuv. Všechny čtyři krokové motory jsou bipolární a mohou byt ovládané pomoci přivedení pulzů stejnosměrného proudu na cívky motoru.

2 KROKOVÉ MOTORY

Krokový motor je elektromotor s několika cívkami. V tu chvíli, kdy v nějaké cívce vznikne proud, zafixuje se rotor v jedné poloze. Pro točení rotoru, je potřeba přivádět na cívky motoru proud v správném pořadí.



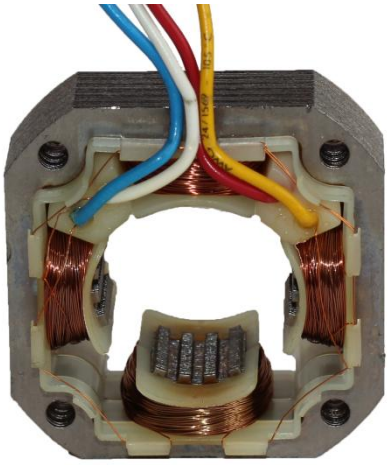
Obr. 2.1: Krokový motor MAE HY 100-1713-020 A4

Krokové motory jsou dobré tím, že nepotřebuji zpětnou vazbu pro zjištění polohy rotoru v určitý okamžik. Proto jen potřebují počítat kroky motoru. Ve skutečnosti to znamená, že stačí vědět, kolik impulzů přišlo na vstup motoru. Pokud víme, kolik kroků motor potřebuje, aby se otočil o jeden stupeň, tak můžeme zjistit uhel otočení hřídele motoru, a z toho je lehké vypočítat jeho polohu. S tím souvisí i největší nevýhoda krokových motorů. Muže se stát, že zatěžovací moment na hřídeli bude větší, než přídržný moment motoru. V takovém případě může motor nestíhat pohybovat se stejnou frekvencí, jako řídicí jednotka, a dochází ke ztrátě kroků. To znamená, že v cívce motoru teče proud, řídicí jednotka počítá jeden krok, a ve skutečnosti se rotor nepohybuje. Řídicí systém ztrácí informaci o přesné poloze motoru a muže dojít k nárazu (v případě robota) nebo rušení procesu výroby atd. Proto se většina krokových motorů používá v aplikacích s předem určeným maximálním momentem. V aplikacích je možnost zjištění polohy motoru najetím na referenční body. Většinou tuto operaci aplikace provede na začátku práce.

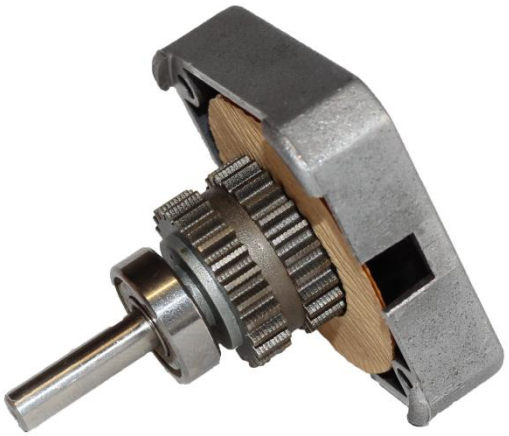
Velká nevýhoda spočívá také v tom, že když se rotor netočí, motor stejně odebírá proud. V důsledku toho se motor více ohřívá. Na druhou stranu je rotor zafixovaný v určité poloze a nepotřebuje vnější brzdění. Z toho vyplývá, že nejsou moc rentabilní. Problém s odebíráním proudu při zastaveném motoru jsem částečně vyřešil. Navrhl jsem možnost zastavení robota takovým způsobem, aby na cívkách nebyl žádný proud. Motor se však nachází v režimu čekání, a může se začít pohybovat v jakémkoliv okamžiku. Nevýhoda je jasná – motor nebrzdí a může dojit ke ztrácení kroků. Také není moc výhodný poměr krouticího momentu vůči hmotnosti samotného motoru. [1]

2.1 Konstrukce krokových motorů

Krokové motory se skládají ze statoru, na kterém se nacházejí budící cívky, a rotoru, který je vyplněn permanentním magnetem nebo feromagnetickým materiálem.



Obr. 2.2: Stator motoru MAE



Obr. 2.3: Rotor motoru MAE

Použití permanentního magnetu je lepší kvůli vzniku většího magnetického toku a v důsledku toho zvýšení momentu, kterého může motor dosáhnout. Také při vypnutém napájení je rotor zafixovaný. Na druhou stranu nejsou motory s permanentním magnetem v rotoru vhodné pro použití v aplikacích, kde je potřebná vysoká rychlost, kvůli vzniku elektromotorického napětí. Proto, když je potřeba vysokých rychlostí, většinou se nepoužívají krokové motory s permanentními magnety. Dnes už se více používají hybridní motory.

Jak vidíme na obrázku 2.3, je rotor krokového motoru usazen na kuličkových ložiscích. Také je vidět, že se rotor a stator skládají ze zubů, které jsou vroubkovány se stejnou roztečí. Je to uděláno kvůli zvýšení počtu kroku na otáčku což snižuje úhel jednoho kroku a zvyšuje přesnost polohování motoru. [1]

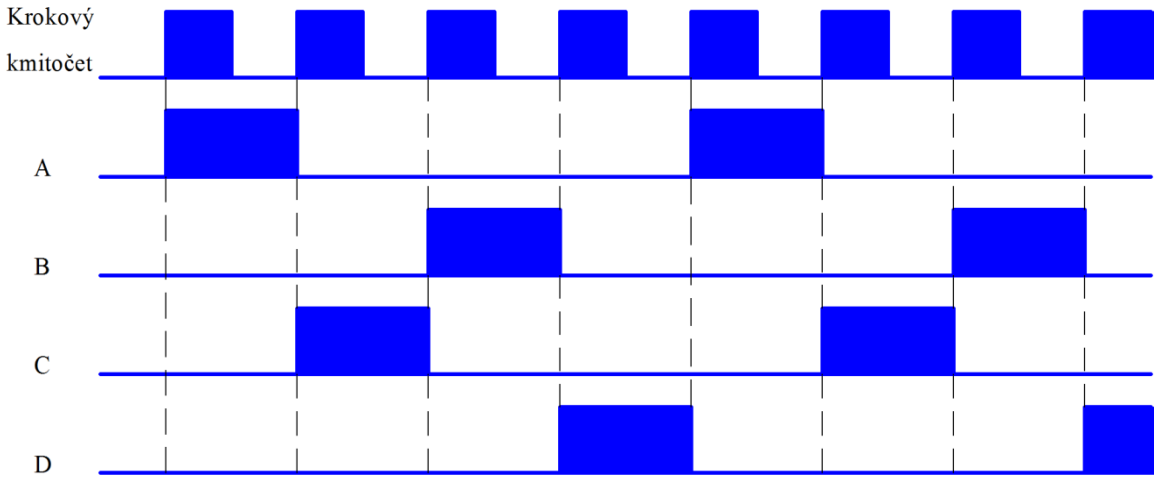
Motory, které používám v robotu, a které jsou na obrazcích č 2.1, 2.2 a 2.3 jsou hybridní. To znamená, že mají v rotoru permanentní magnet a mají větší počet pólů. V důsledku takový typ motorů dovoluje zmenšit velikost kroku až na 100 kroku na otáčku, což odpovídá kroku 3.6^{o} na otáčku. Navíc můj řídicí systém dovoluje snížit tento uhel na hodnotu 1.8^{o} na otáčku, s možností použití půlkrokového režimu řízení, což výhodně zvýší přesnost polohování motoru.

2.2 Řízení krokových motorů

Existují tři základní způsoby řízení krokových motorů: vlnový, celokrokový a půlkrokový. Níže jsou krátce popsány tyto typy řízení.

2.2.1 Vlnový režim (*angl.* «wavedrive mode» nebo «one-phase-on»)

V tomto režimu řízení je v jednom okamžiku aktivní pouze jedna fáze. Body rovnováhy rotoru jsou stejné jako „přirozené“ body rovnováhy rotoru u vypnutého motoru.

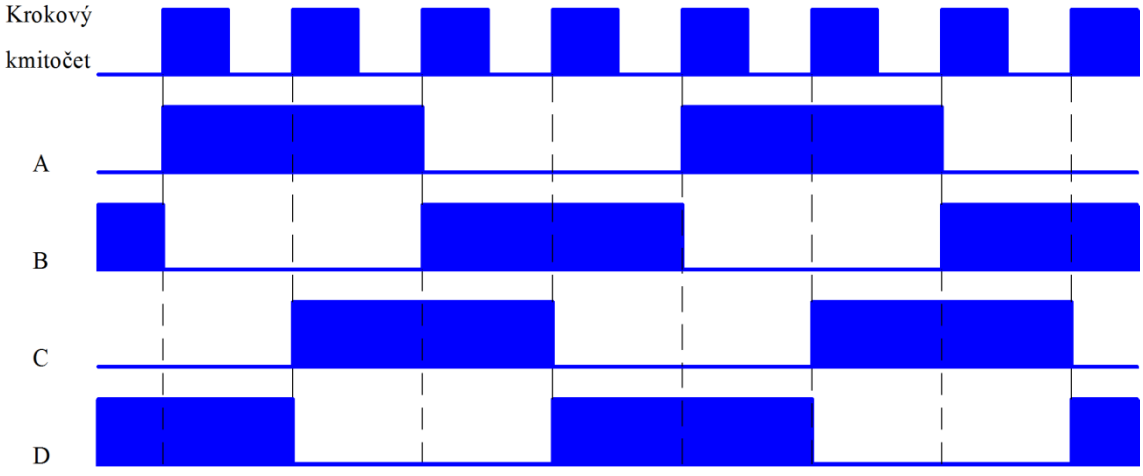


Obr. 2.4: Průběh proudu na vstupech krokového motoru ve vlnovém režimu ovládání

Nedostatek tohoto způsobu řízení spočívá v tom, že se pro bipolární motor v jednom okamžiku využívá pouze 50% vinutí. To znamená, že při takovém způsobu ovládání nemůže být na motoru dosaženo maximálního momentu. [1]

2.2.2 Celokrokový režim (*angl.* «full step mode» nebo «two-phase-on»)

V daném režimu jsou dvě fáze zapnuté v jednom okamžiku. Rotor se fixuje v přechodném stavu mezi póly statoru.

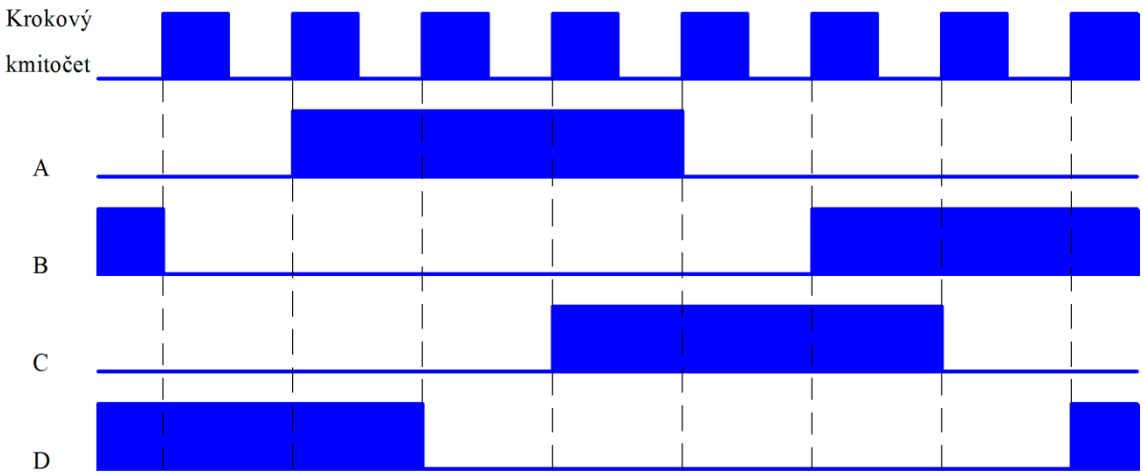


Obr. 2.5: Průběh proudu na vstupech krokového motoru v celokrokovém režimu ovládání

Při takovém způsobu řízení je krouticí moment o 40% větší, než ve vlnovém režimu. Počet kroků na jednu otáčku zůstává stejný, ale body rovnováhy rotoru jsou posunuté o půl kroku. Důležité je to, že taková poloha bude v rotoru, pokud je zapnuté napětí. V okamžiku, kdy vypneme nebo zapneme motor, se rotor otočí o půl kroku. Proto, aby se motor neposouval při zastavení, je potřeba podávat na motor „brzdový proud“. Ve většině případů on nemusí mít nominální velikost, může být menší. Ale občas, pokud se robot potřebuje zastavit a je zatížený, tak můžeme „brzdový proud“ nechat na stejné velikosti jako nominální. Tak dosáhneme maximálního brzdového momentu, což je pro krokový motor obvyklá situace. Takovým způsobem se dá vyhnout použití vnější brzdy. [1]

2.2.3 Půlkrokový režim (*angl.* «half step mode» nebo «one and two-phase-on»)

Je to kombinace dvou předchozích režimů. Velikost kroku se rovná půlce základního kroku.



Obr. 2.6: Průběh proudu na vstupech krokového motoru v půlkrokovém režimu ovládaní

Tento způsob řízení je dobře použitelný, protože motor s větším počtem kroků na otáčku je dražší, a je možné dostat 200 kroků na otáčku místo 100. Jak vidíme na obrázku, každý druhý krok je zapnutá jenom jedna fáze. Kvůli tomu je uhel otáčení rotoru (krok) dvakrát menší než při využití prvních dvou způsobů. Kromě zvýšení počtu kroků na otáčku, je možné použitím tohoto způsobu zmenšit rezonance. Na druhou stranu se v půlkrokovém režimu nedá dosáhnout maximálního krouticího momentu motoru.

2.3 Základní charakteristiky krokových motorů angulárního robota

V angulárním robotu z mé diplomové práce jsou pohony třech os robotu vytvořené pomocí krokových motorů MAE HY 100-1713-020 A4, které jsou zobrazené na obrazcích 2.1, 2.2, 2.3. Z důvodu, že jsou motory už docela staré, se nepodařilo najit charakteristiky motorů od výrobce, proto se níže uvedené charakteristiky mohou od skutečných lišit, protože jsem většinu z nich zjišťoval sám.

- Maximální napětí – 40V
- Nominální proud – 0.33A
- Krouticí moment – 0.137 Nm*
- Přídržný moment – 0.017 Nm
- Řídící kmitočet – 60 – 700 Hz
- Úhel kroků – 3.6^{o}
- Odpor – 24 Ω
- Indukce – 29 mH
- Počet výstupů - 4
- Hmotnost motoru – 0.2 kg

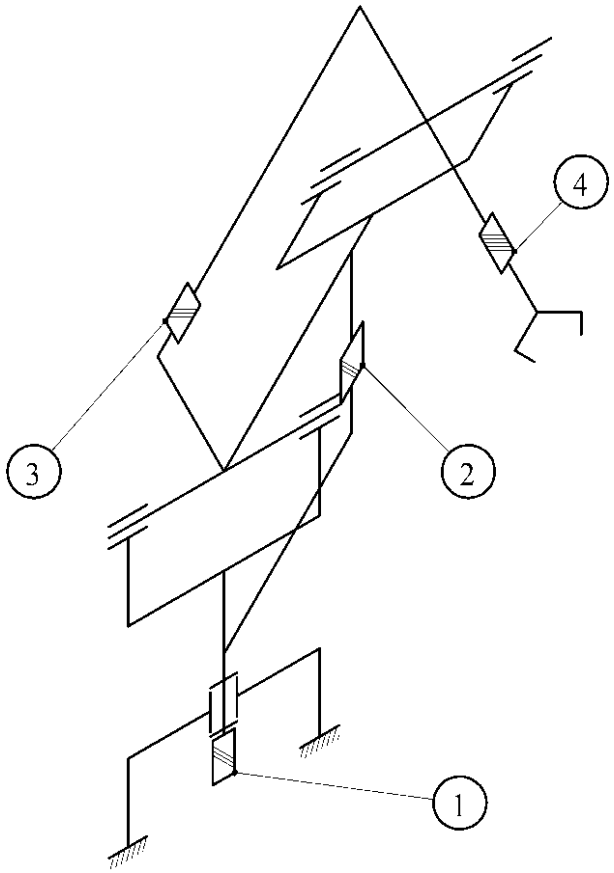
*Největšího krouticího momentu motor dosahuje při řídicí frekvenci cca. 300 Hz

Krouticí momenty, uvedené výše, jsou pouze orientační. Pro své aplikace jsem je neměřil, protože v robotu nebylo příliš důležité je přesně zjišťovat.

Také jsem měl další problémy, související s tím, že motory, které jsou v robotu, už jsou 20 let staré. Občas se stávalo, že některé motory nefungovaly tak, jak by měly. V některých situacích už jsem myslel, že při zkouškách motoru došlo ke spálení jedné cívky motoru – odpor jedné cívky byl nekonečně vysoký. Nebyl to ale problém motoru. Většinou byly problémy způsobené spojením mezí cívkami motoru a řídící deskou. Tak jsem musel vyměnit nebo opravit kabely.

2.4 Označení krokových motorů robota

V diplomové práci jsou krokové motory dále označené jako:



1 – pohon, který rotuje celou horní částí robota kolem svislé osy a je umístěný uvnitř dolní krabice, je označen jako COXA.

2 – pohon, který zajišťuje svisly pohyb robota je označen jako FEMUR.

3 – pohon, který zajišťuje pohyb robota dopředu a dozadu je označen jako TIBIA.

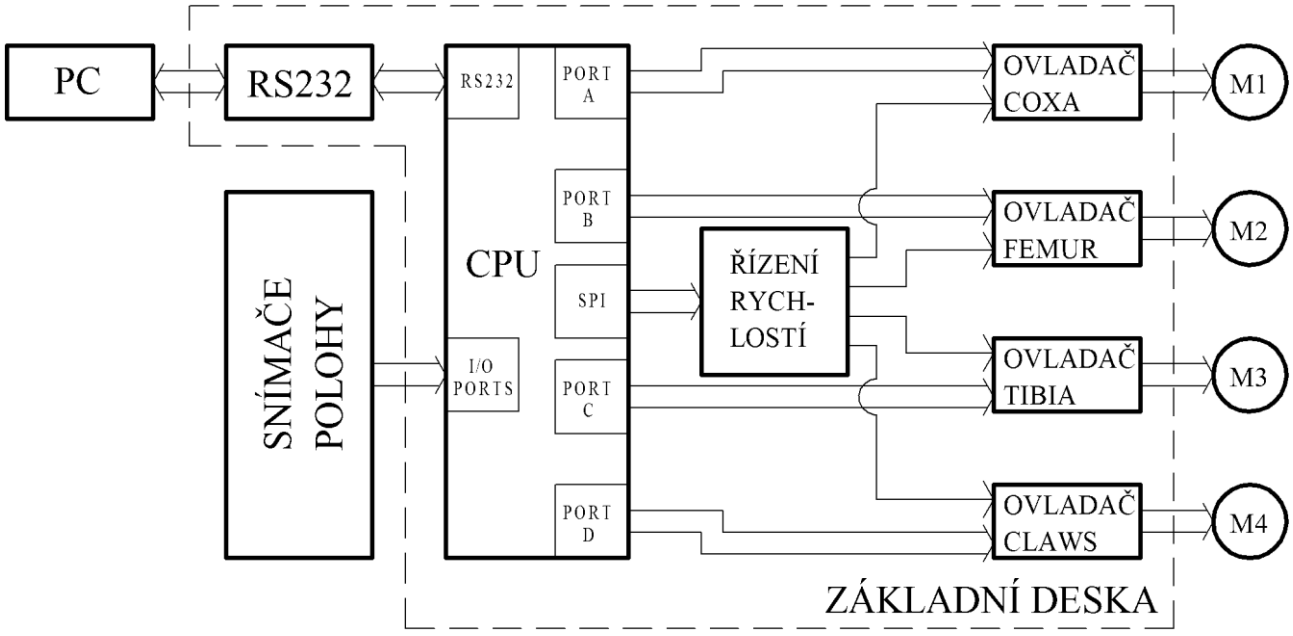
4 – pohon, který pohybuje chapadlem, je označen jako CLAWS.

Obr. 2.7: Kinematické schéma robota
s označením pohonů

3 POPIS ZÁKLADNÍ DESKY

Řídicí desku angulárního robota jsem projektoval postupně. Občas jsem na ni navazoval další moduly pro přidání dalších funkcí robotu. Proto se dá finální verze desky rozdělit na několik hlavních funkčních bloků:

- Řídicí jednotka – mikroprocesor PIC18F4420 (na obr. 3.1 označen jak CPU)
- Komunikace s vnějším zařízením – deska RS232
- Blok řízení rychlosti – základem je časovač, zapojeny jako generátor, NE555
- Ovladače motorů – L297+L298



Obr. 3.1: Blokové schéma řídicího systému

3.1 Mikroprocesor PIC18F4420

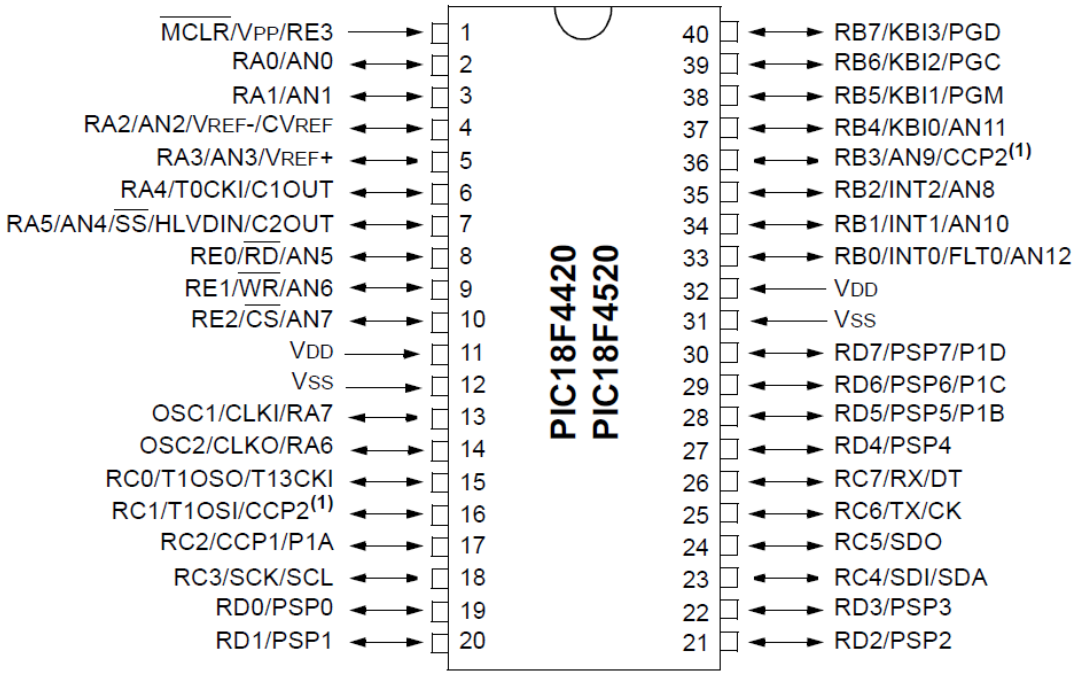
Od začátku práce jsem věděl, že budu potřebovat zajistit současné ovládání všech motorů. To znamená, že pro každý motor ve stejném okamžiku může vzniknout potřeba zadávat rychlost, směr otáčení, vypočítat kroky, které motor prošel nebo kolik kroků mu ještě zbývá. Také v krokových motorech je ještě spousta různých nastavení, které také musejí byt nastavené. Mikroprocesor by měl také ještě umět komunikovat s nějakým vnějším zdrojem, který by nějak rozumně řídil pohyb robotu. Kvůli tomu jsem musel zvolit rychlý procesor s velkým počtem nožiček, který by měl uvnitř modul pro komunikaci s jinými zařízením prostřednictvím nějaké sběrnice, atd.

PIC18F4420 je osmibitový mikroprocesor od firmy Microchip Technology. Je založen na principech Harvardské architektury s RISC instrukční sadou. Vysokorychlostní mikroprocesor s velkým počtem nožiček je možná nejlepším řešením pro tuto aplikaci.

Základní charakteristiky: (potřebné pouze pro řízení robotu). [3]

- Taktovací kmitočet – vnitřní generátor do 4MHz
- Flash paměť programu – 16KB
- Paměť dat – 786B
- EEPROM paměť – 256B
- Dva vektory přerušení
- Čtyři I/O porty
- Modul EUSART (podpora RS-232)
- Modul MSSP (podpora SPI)
- Čtyři časovače
- Zdroje přerušeni – 20

Tento mikroprocesor jsem použil hlavně kvůli tomu, že podle rychlostních charakteristik zvládne řídit čtyři motory současně. Také podporuje práci s SPI datovou sběrnici, kterou jsem potřeboval pro změnu řídicí frekvence a rychlosti motoru. Podpora RS-232 je důležitá pro možnost ovládání z vnější aplikace. Pro přechod z pětivoltové logiky na logiku protokolu RS-232 jsem ale musel dodatečně zhotovit desku s čipem MAX232.



Obr. 3.2: Piny mikroprocesoru PIC18F4420

Skoro každý pin mikroprocesoru může mít několik funkcí. Nejobtížnější při programování takového mikroprocesoru je správné nakonfigurování každého užívaného pinu, aby všechny moduly fungovaly tak, jak mají.

Tab. 3.1: Popis vstupů/výstupů mikroprocesoru

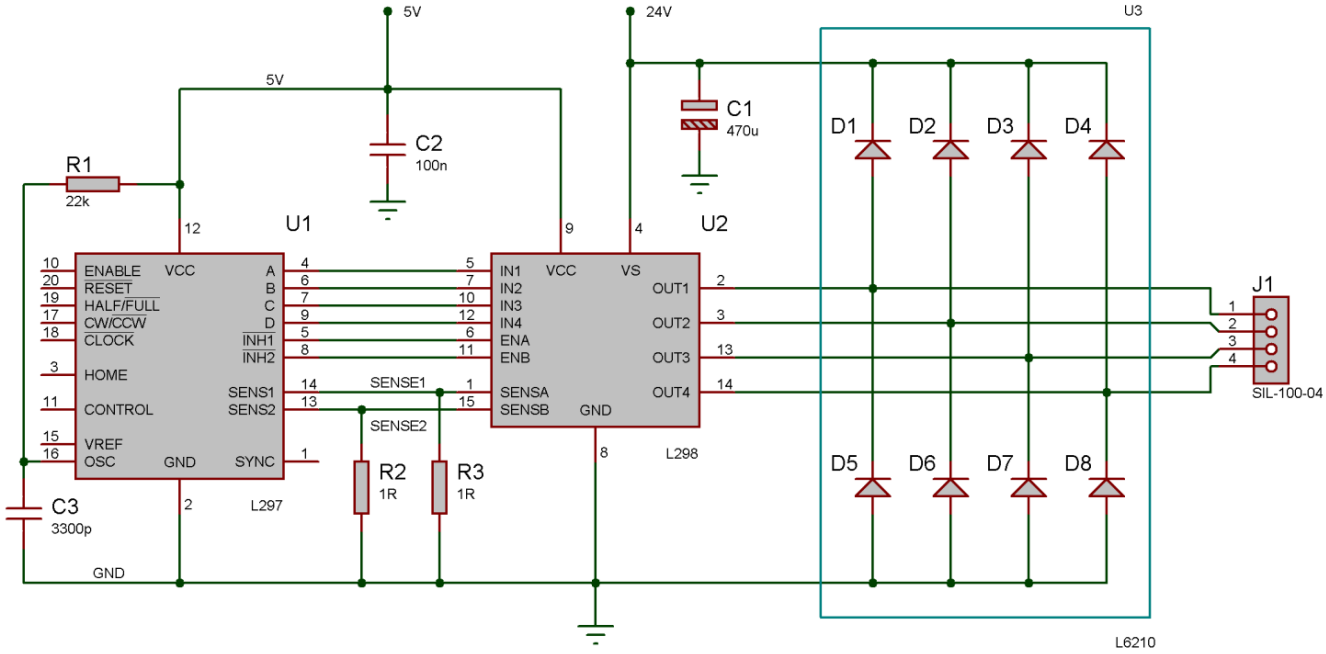
Pin č.	Název	Popis
1	V_{pp}	Vstup programovacího napětí
2	RA0	Výstup směru otáčení motoru COXA
3	RA1	Výstup pro zapnutí půlkrokového režimu motoru COXA
4	RA2	Výstup pro resetovaní motoru COXA
5	RA3	Povolovací výstup motoru COXA
6	RA4	Vstup indukčního čidla č. 1
7	RA5	Vstup indukčního čidla č. 2
8	RE0	Výstup pro povolení zápisu potenciometru č. 1
9	RE1	Výstup pro povolení zápisu potenciometru č. 2
10		Nepoužívám
11	V_{DD}	Napájecí napětí
12	V_{SS}	Uzemnění
13	RA7	Koncový spínač motoru FEMUR (dolní poloha)
14	RA6	Koncový spínač motoru FEMUR (horní poloha)

Pin č.	Název	Popis
15	RC0	Výstup směru otáčení motoru TIBIA
16	RC1	Výstup pro zapnutí půlkrokového režimu motoru TIBIA
17	RC2	Výstup pro resetovaní motoru TIBIA
18	SCK	Výstup řídícího kmitočtu SPI sběrnice
19	RD0	Výstup směru otáčení motoru CLAWS
20	RD1	Výstup pro zapnutí půlkrokového režimu motoru CLAWS
21	RD2	Výstup pro resetovaní motoru CLAWS
22	RD3	Povolovací výstup motoru CLAWS
23	SDI	Vstup SPI sběrnice
24	SDO	Výstup SPI sběrnice
25	TX	Výstup RS-232
26	RX	Vstup RS-232
27	RD4	Povolovací výstup motoru TIBIA
28	RD5	Koncový spínač motoru CLAWS (otevřeno)
29	RD6	Koncový spínač motoru CLAWS (zavřeno)
30	RD7	Koncový spínač motoru CLAWS (objekt uvnitř)
31	V_{SS}	Uzemnění
32	V_{DD}	Napájecí napětí
33	RB0	Výstup směru otáčení motoru FEMUR
34	RB1	Výstup pro zapnutí půlkrokového režimu motoru FEMUR
35	RB2	Výstup pro resetovaní motoru FEMUR
36	RB3	Povolovací výstup motoru FEMUR
37	RB4	Koncový spínač motoru TIBIA (horní poloha)
38	RB5	Koncový spínač motoru TIBIA (dolní poloha)
39	PGC	Vstup pro programovací kmitočet
40	PGD	Vstup pro programovací data

Jak je vidět z této tabulky, skoro všechny piny mikroprocesoru jsou využité.

3.2 Ovladač krokových motorů

Ovládání krokových motorů jsem navrhl pomocí spojení dvou součástek od firmy ST Microelectronics – L297+L298 a 8 ochranných diod v jednom pouzdře L6210.



Obr. 3.3: Ovladač krokových motorů

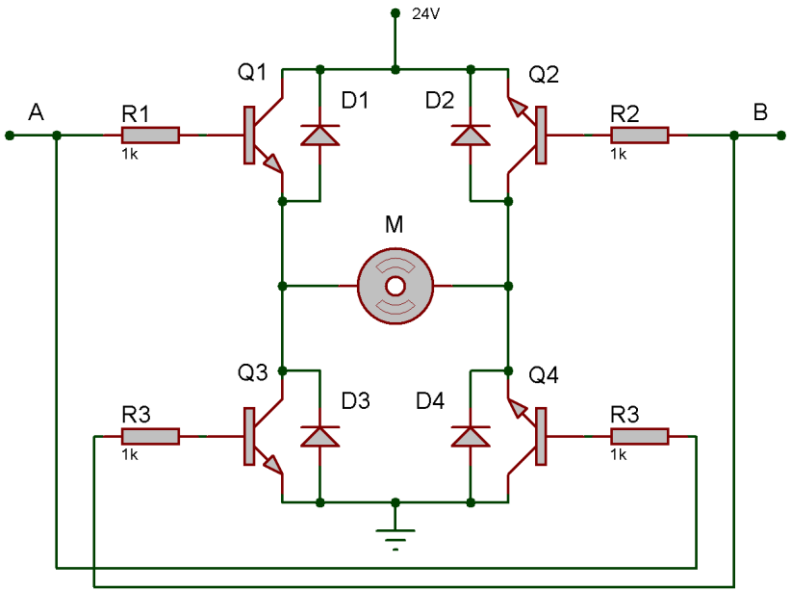
Na tomto obrázku je zobrazen jeden ze čtyř ovládačů krokových motorů. Skládá se ze třech základních částí:

- U1 – L297 – ovládač krokového motoru
- U2 – L298 – dvojice H-můstků
- U3 – L6210 – dvojice můstků ze Schottkyho diod

3.2.1 L297

Ovládač krokového motoru L297 je určený konkrétně pro řízení krokových motorů. Na výstupech A, B, C, D generuje čtyřfázové řídicí signály pro dvoufázové bipolární a čtyřfázové unipolární krokové motory. Podporuje všechny tři způsoby ovládání – vlnový režim (*angl.* «wavedrive mode» nebo «one-phase-on»), celokrokový režim (*angl.* «full step mode» nebo «two-phase-on») a půlkrokový režim (*angl.* «half step mode» nebo «one and two-phase-on»). Na vstup potřebuje přivádět pouze krokový kmitočet, směr otáčení a signály pro výběr způsobu ovládání. Kvůli tomu, že je řídicí sekvence generovaná uvnitř, je zatížení mikroprocesoru velmi nízké, což je v našem případě výhoda, protože ovládání 4 motorů současně bude mikroprocesoru odebírat 90% času. [2]

3.2.2 L298



Obr. 3.4: Zapojení H-můstku

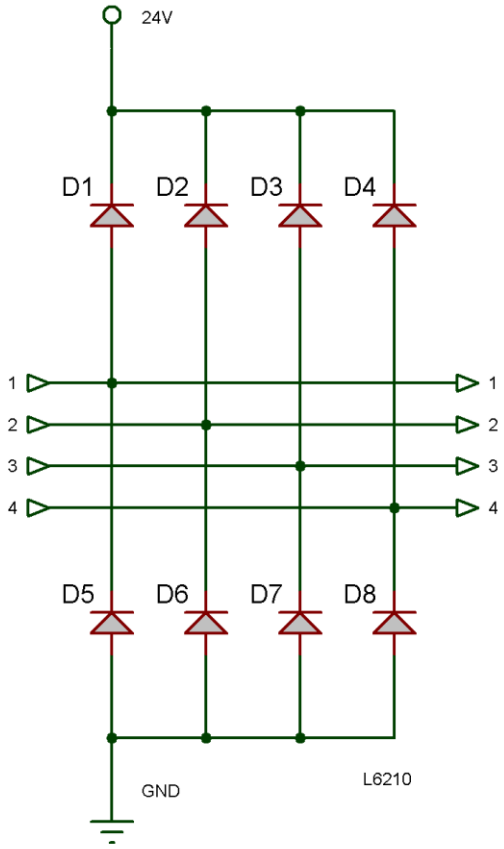
Při bipolárním řízení je výhodné zapojit každou cívku krokového motoru do H-můstků. Na obrázku 3.4 je vidět, že pokud bude na vstupu A řídicí signál logické jedničky a na vstupu B logická nula, proud v cívce motoru poteče jedním směrem. Pokud na vstupu B bude signál logické jedničky, a na vstupu A bude logická nula, tak proud v cívce motoru poteče v opačném směru.

Vzhledem k tomu, že v bipolárním krokovém motoru jsou dvě protilehlé cívky, dostaneme navzájem opačně orientované magnetické pole. Pro řízení dvou cívek potřebujeme dva H-můstky, proto je výhodné použít dvojice H-můstků – integrační obvod L298.

Po několika zkouškách jsem zjistil, že se L298 při maximálním zatížení hodně ohřívá. Kvůli tomu jsem zvolil L298 v pouzdře Multiwatt ve svislém provedeni, aby bylo možné přidat vnější chladič. Maximální napětí, které může obvod snášet je 46V. Napětí krokových motorů robotu je 24V. Logická část obvodu pracuje na napětí 5V. [4]

3.2.3 L6210

Bohužel L298 neobsahuje ochranné diody jako H-můstek na obrázků 3.4. Proto je potřeba přidat další součástku L6210 – dvojici můstků ze Schottkyho diod.



Obr. 3.5: L6210

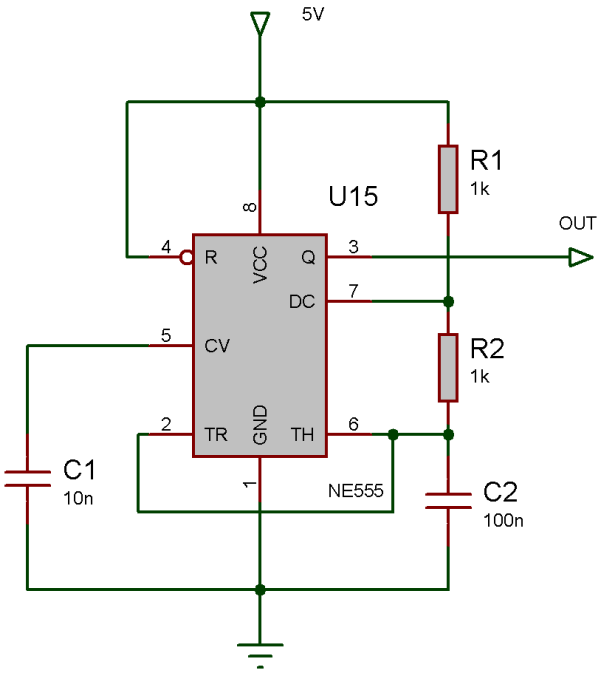
Použil jsem ho, protože bipolární motory vyžadují velice rychlé zpracovaní a malou zotavovací dobu. Schottkyho dioda může pracovat ve velmi rychlých spínacích obvodech.

Tab. 3.2: Základní charakteristiky L6210 [9]

Parametr	Označení	Hodnota	Jednotky
Max. propustný proud	I_f	2	A
Max. zpětné napětí (přes 1 diodu)	V_r	50	V
Provozní teplota okolí	T_{amb}	70	^oC
Rozsah teploty při skladování	T_{stg}	-55 do +150	^oC

3.3 Řízení rychlosti

Ovládač krokových motorů L297 na vstup č. 18 CLOCK potřebuje krokový kmitočet, na kterém závisí rychlost. Generátory pulzů musí byt čtyři, pro každý motor jeden. Proto jsem použil obvod z časovačem NE555 od firmy Texas Instruments, který se dá zapojit tak, abych z něj udělal generátor krokového kmitočtu.



Obr. 3.6: Schéma zapojeni časovače NE555 pro generování krokového kmitočtu

Toto schéma zapojení na výstupu OUT generuje pulzy konstantního kmitočtu. Když ho použijeme pro náš případ, tak se budou všechny čtyři motory pohybovat vždy stejnou rychlostí. Abychom měli možnost měnit v jakémkoliv okamžiku rychlost motoru programem, rozhodl jsem se přidat místo odporu R2 číslicový potenciometr, který se dá ovládat z mikroprocesoru přes nějaké rozhraní. Pro tento cíl jsem si zvolil digitální potenciometr DS1267-100 od firmy Dallas Semiconductor (MAXIM Integrated). Tato součástka se skládá ze dvou digitálních potenciometrů, což znamená, že nám stačí jenom dvě součástky pro čtyři motory.

DS1267-100 komunikuje s mikroprocesorem prostřednictvím sériové sběrnice SPI.

3.3.1 Popis SPI sběrnice

Mikroprocesor PIC18F4420 podporuje komunikaci pomocí SPI rozhraní. Pro tu komunikaci má vnitřní modul MSSP (Master Synchronous Serial Port Module) který je vhodný pro práci s jinými mikroprocesorovými periferními zařízeními nebo s jinými mikroprocesory. Také MSSP podporuje komunikaci v režimu I^2C (Inter-Integrated Circuit).

MSSP modul používá tři registry pro nastavení. Jeden status registr (SSPSTAT) a dva ovládací registry (SSPCON1 a SSPCON2). Pomocí nastavení bitů v těchto registrech je možné nastavit individuální konfiguraci MSSP modulu pro práci v SPI nebo I^2C režimu. Také je důležitý registr buffer (SSPBUF), kam se uloží data pro další zápis nebo pro čtení. Pro operaci příjmu dat se ještě používá Posuvný Registr (SSPSR), který spolu s registrem SSPBUF tvoří dvojitý buffer přijímače. Při přenosu dat SSPBUF netvoří dvojnásobný buffer s registrem SSPSR, data pro přenos se zapisují v SSPBUF a kopírují se do SSPSR.

V SPI režimu se osm datových bitů posílá a přijímá současně. Mikroprocesor podporuje všechny čtyři režimy SPI. Proto se obvykle používají čtyři vodiče:

- Vystup (SDO) – RC5/SDO
- Vstup (SDI) – RC4/SDI/SDA
- Kmitočet (SCK) – RC3/SCK/SCL
- Výběr podřízeného (SS) – RA5/SS [3]

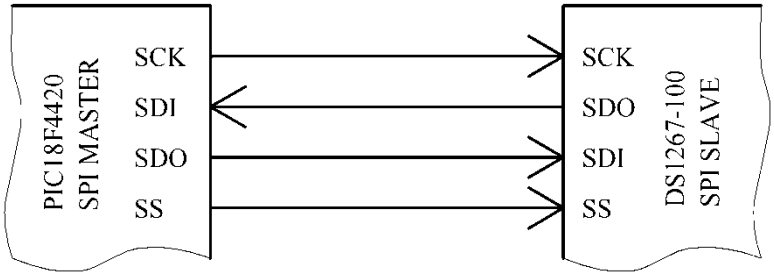
SDO výstup jednoho zařízeni musí být připojen na SDI vstup druhého.

Pro komunikaci s potenciometrem jsem zvolil nadřízený režim práce procesoru. To znamená, že procesor muže iniciovat komunikaci kdykoliv, protože kontroluje SCK. Procesor rozhoduje, kdy podřízený přístroj přijme data podle protokolu. V nadřízeném režimu se data začínají přenášet nebo přijímat, když je registr SSPBUF plný. Když SPI pracuje jenom v přijímacím režimu, výstup SDO může byt naprogramovaný jako vstup.

Polarita kmitočtu může být vybraná nastavením čtvrtého bitu registru SSPCON1 CKP. V nadřízeném režimu může být kmitočet odesílání dat nastavený na:

- FOSC/4
- FOSC/16
- FOSC/64
- Výstup timer2/2

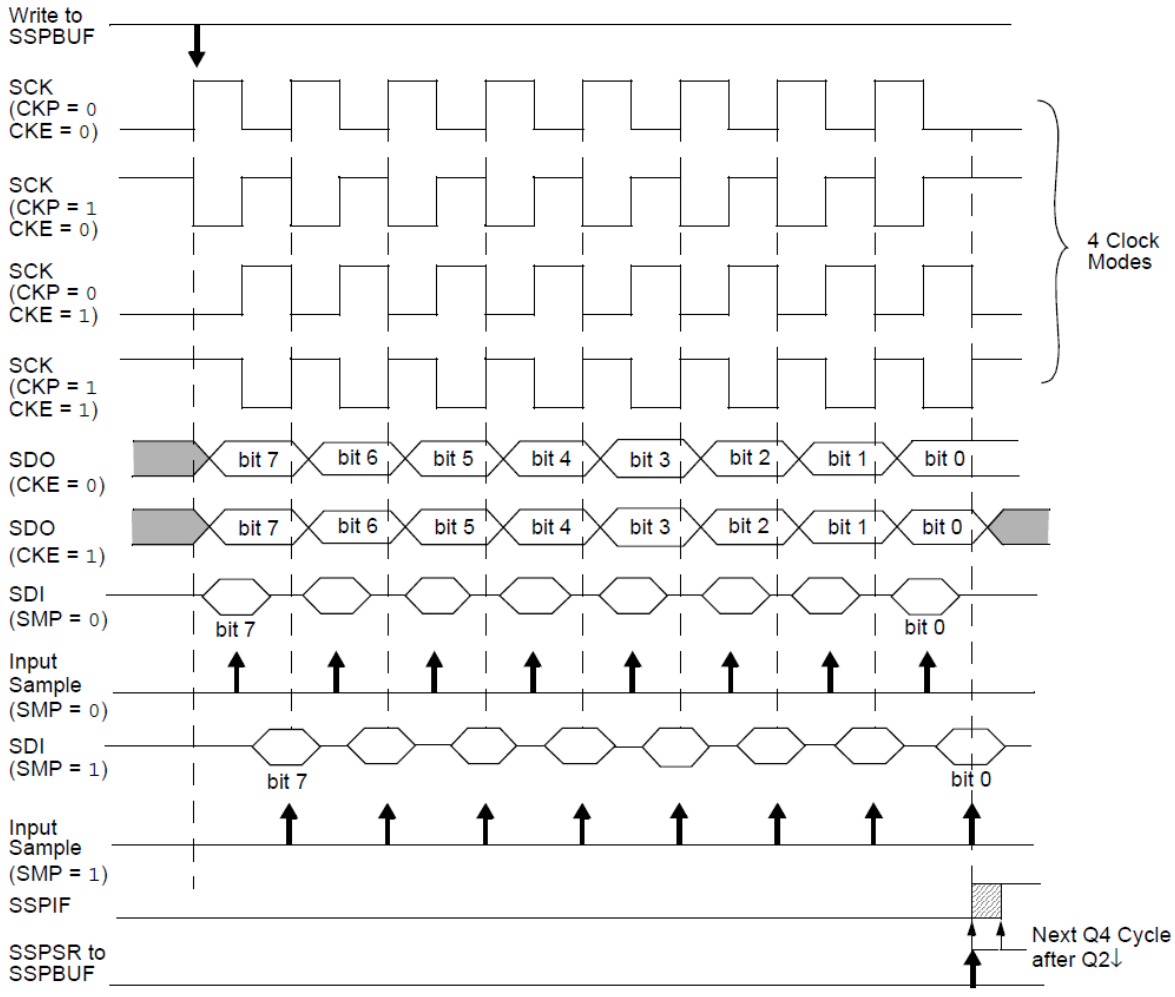
Maximální přenosová rychlost může dosáhnout až 10 Mbps (40 MHz).



Obr. 3.7: Komunikace přes sběrnici SPI

Pro komunikaci s jedním čipem DS1267 je zbytečně používat výstup procesoru SS, protože není potřeba řešit, s jakým přístrojem musí procesor komunikovat. Použil jsem však dva potenciometry, proto jsem musel řešit komunikaci se dvěma přístroji přes SPI. Proto jsem nepoužíval standardní pin MSSP modulu SS, ale zvolil jsem jiné porty procesoru RE0 – SS pro první potenciometr a RE1 – SS pro druhý potenciometr.

DS1267-100 neposílá data mikroprocesoru, proto druhý drát SDI-SDO, který je vidět na obrázku 3.7 nepoužívám. Stejně jsem ho však musel zapojit, protože když je zapnutý modul MSSP, tak se nedá využít tento vstup pro jiné účely.



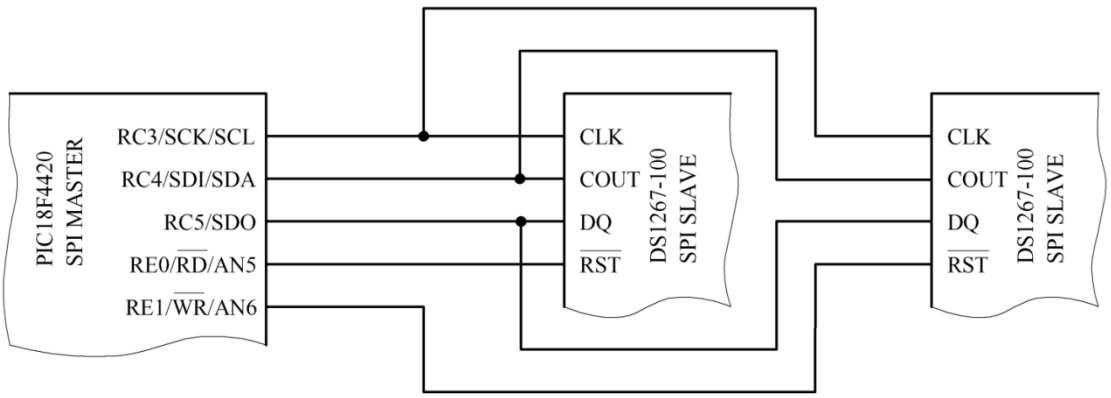
Obr. 3.8: Tvar vlny v SPI režimu (Nadřízený režim) [3]

Na obrázku 3.8 vidíme průběhy signálů v závislosti na nastavení jednotlivých bitů registru SSPCON1 a SSPSTAT.

3.3.2 Dvojitý číslicový potenciometr DS1267-100

Na obrázku 3.6 je zobrazené schéma generátoru pro generaci signálů s konstantním kmitočtem. Proto pro možnost měnit kmitočet signálů a s ním i rychlost pohybu motorů, musíme místo rezistoru R2 přidat číslicový potenciometr, který by se dal ovládat z mikroprocesoru. Pro tento cíl jsem zvolil dvojitý číslicový potenciometr DS1267 s maximálním odporem 100 KΩ. Tato součástka se skládá ze dvou potenciometrů v jednom pouzdře. Každý se skládá z 256 odporových sekcí. Mezi každou sekci a dvěma konci potenciometru jsou tečky, na kterých se muže nacházet elektronický přepínač, který se chová jako elektronický jezdec. Pozice, na kterou může být jezdec umístěn, na poli odporů, se nastavuje pomoci osmibitové hodnoty. Pro nastavení této hodnoty potenciometr komunikuje s mikroprocesorem pomoci třidrátového sériového rozhraní. To znamená, že toto rozhraní dovoluje číst data z potenciometru a zapisovat je do něj. [6]

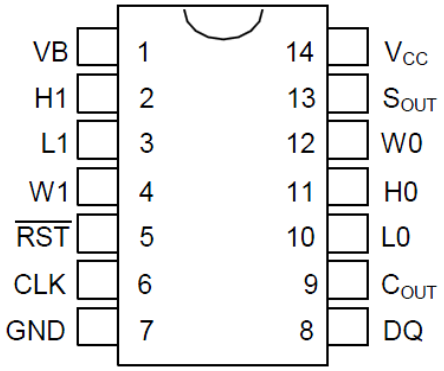
Existuje několik možných způsobů zapojení potenciometrů do řady. Buď může být každý potenciometr sériově zapojen v jednom pouzdře nebo může být každá součástka zapojená sériově pro zvýšení celkového odporu. Také SPI rozhraní dovoluje řídit několik součástek v paralelním zapojení. Pro takový způsob komunikace se používají standardní tři vodiče pro přenos dat a jeden vodič pro každý další přistroj. Tím pádem pro zapojení dvou potenciometrů DS1267 přes sériové rozhraní, potřebujeme pět vodičů. [6]



Obr. 3.9: Paralelní zapojení mikroprocesoru a dvou potenciometrů

Z obrázku 3.9 je vidět, že pro výběr potenciometru používáme dva signály z výstupu RE0 a RE1 mikroprocesoru. Když chceme komunikovat s potenciometrem č. 1, na výstup RE0 přivádíme logickou jedničku, a na vystup RE1 přivádíme logickou nulu. Pokud chceme komunikovat s potenciometrem č. 2, naopak na výstup RE0 musíme přivádět logickou nulu, a na výstup RE1 logickou jedničku.

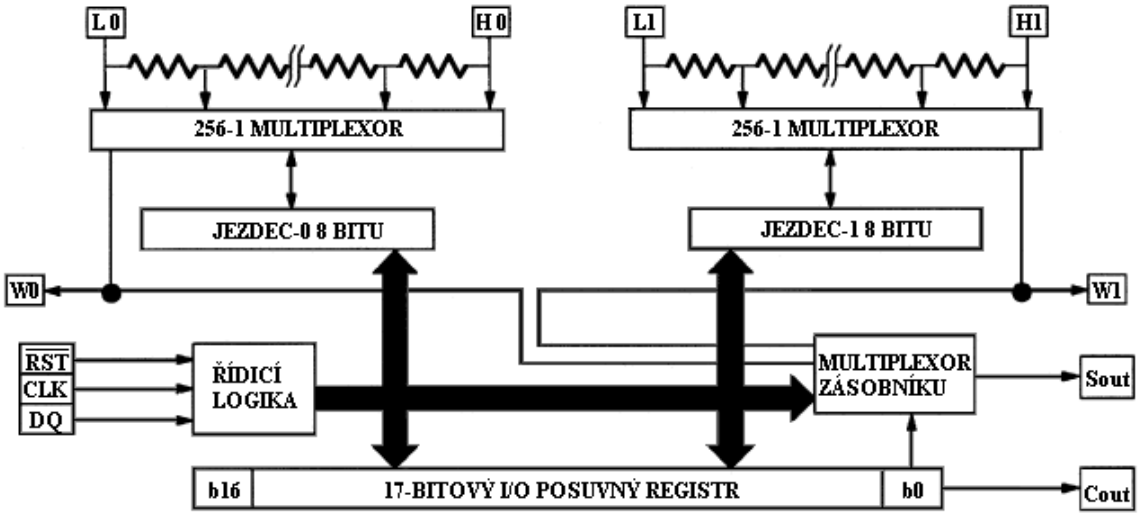
Pro svou aplikaci jsem použil potenciometr v provedení 14-Pin DIP.



Obr. 3.10: Dvojitý číslicový potenciometr DS1267-100 v provedeni 14-Pin DIP [6]

Výstupy H1 a W1 jsou výstupy prvního potenciometru, a výstupy H0 a W0 jsou výstupy druhého potenciometru. CLK, /RST, DQ jsou vstupy SPI sběrnice.

Každý potenciometr uvnitř DS1267 se nastavuje osmibitovou hodnotou. Tyto hodnoty se zapisují do 17-bitového posuvného registru, který DS1267 používá pro ukládání pozic dvou jezdců a bitu výběru zásobníku. Pro náš případ bude bit výběru zásobníku vždy nulový, protože nepoužíváme sériové zapojení potenciometrů.



Obr. 3.11: Blokové schéma vnitřní realizace DS1267 [6]

Jak je vidět na výše uvedeném obrázku, /RST signál se používá pro výběr potenciometru, s kterým chce mikroprocesor komunikovat. Tato součástka je vybraná, když je na vstupu /RST vysoká úroveň signálu. Tj. pro komunikaci s vybraným čipem musí být na vstupu /RST logická jednička. CLK vstup používáme pro časovou synchronizaci přenosu dat mezi přijímačem a vysílačem. Vstup DQ slouží pro přenos dat pro nastavení jezdce do 17-bitového posuvného registru.[6]

Obr. 3.12: 17-bitový I/O posuvný registr

Bit 0 na obrázku 3.12 je bit výběru zásobníku, který pro tuto aplikaci vždy nechávám nulovým. Bity od 1 (LSB) do 8 (MSB) nastavují hodnotu potenciometru č. 1. Bity od 9 (LSB) do 16 (MSB) nastavují hodnotu potenciometru č. 0.

Přenos dat vždy začíná zápisem do prvního bitu výběru zásobníku, pak následuje zápis dat pro nastavení potenciometru č. 1 a na konci pro potenciometr č. 0. V okamžiku, když jsou data zapsány do posuvného registru, musí být vždy poslané dál. V opačném případě muže dojit k chybnému zápisu do posuvného registru a ve výsledku ke špatnému nastavení DS1267. Tak to bylo uděláno pro ověření správnosti odeslaných dat. Jelikož jsem nepotřeboval pokaždé ověřovat poslaná dat, tak jsem udělal pauzu před ukončením komunikace (za účelem počkání na příchod dat). Po ukončení přenosu musí mikroprocesor uzavřít komunikaci udržením logické nuly na vstupu /RST, pro zabránění změnám hodnoty posuvného registru. Jezdci budou nastavení na potřebnou polohu v okamžiku, když mikroprocesor ukončí komunikaci, tzn. když bude na vstupu /RST nízká úroveň. Při zapnutí DS1267 se potenciometry nastaví na hodnotu 50 KΩ – polovinu maximálního odporu nebo v binární soustavě 1000 0000. [6]

Pro zápis dat do posuvného registru jsem posílal tři bajty. První bajt byl nulový (0b00000000), pro nastavení bitu výběru zásobníku na nulu. Další bajt měl potřebnou hodnotu pro nastavení prvního potenciometru, a poslední, třetí bajt měl hodnotu pro nastavení odporu pro potenciometr č. 0. Data jsem posílal sériově s rychlostí FOSC/64.

Tab. 3.3: Základní elektrické charakteristiky DS1267-100

Parametr	Označení	Min	Typ	Max	Jednotky
Napájecí napětí	V_{CC}	4.5		5.5	V
Logická jednička	V_{IH}	2.0		V_{CC}+0.5	V
Logická nula	V_{IL}	-0.5		+0.8	V
Odporové vstupy	L, H, W	-6.0		+6.0	V
Napájecí proud	I_{CC}		22	650	µA
Vstupní průsak	I_{LI}	-1		+1	µA
Odpor jezdců	R_W		400	1000	Ω
Výstupní průsak	I_{LO}	-1		+1	µA
Záložní proud	I_{STBY}		22		µA
Tolerance odporu		-20		+20	%

Parametr	Označení	Min	Typ	Max	Jednotky
Absolutní linearita			±0.75		LSB
Relativní linearita			±0.3		LSB
Teplotní koeficient			750		ppm/C
CLK Kmitočet	f_{CLK}	DC		10	MHz
Šířka pulzu CLK	t_{CH}	50			ns
Čas na nastavení dat	t_{DC}	30			ns

3.4 Komunikace s vnějším zařízením (RS-232)

Podle zadání jsem musel navrhnout komunikaci (případně ovládaní) robota z vnějšího přístroje. Pro tento účel jsem se rozhodl používat rozhraní RS-232. V oblasti osobních počítačů je tento standard už dávno nahrazen jiným standardem sériového přenosu dat – standardem USB. Ale v průmyslu, kde je spolehlivost důležitější, než rychlost, se většinou používá rozhraní RS-232 nebo novější modifikace RS-422 a RS-485.

Pro ovládání robota jsem se rozhodl používat počítač. Programoval jsem robota tak, aby ho bylo možné, bez úpravy vnitřního programu, kontrolovat z jiného zařízeni, které má COM port a dovoluje přenášet 16 bytů za sebou.

3.4.1 Popis rozhraní RS-232 z hlediska mikroprocesoru PIC18F4420

Mikroprocesor PIC18F4420 dovoluje komunikaci pomocí synchronního/asynchronního sériového rozhraní. Pro tento účel mikroprocesor obsahuje samostatný modul EUSART (Enhanced Universal Synchronous Receiver Transmitter). Modul může být nakonfigurován jako plně duplexní asynchronní systém pro práci s periferními zařízeními, jako v našem případě. Za periferní zařízení považujeme počítač nebo jiné obdobné zařízení, které má sériový port a může přenášet data v odpovídajícím formátu. Modul může být také nakonfigurovaný pro práci v napůl duplexním režimu, jako synchronní systém. Tento režim je vhodný pro práci s integrovanými obvody, například s A/D nebo s D/A převodníky, se sériovou EEPROM a dalšími. [3]

Tento modul také podporuje funkci automatického probuzení, automatické korekce přenosové rychlosti (baud rate). Také nabízí možnost automatické změny konfigurace pinů ze vstupních na výstupní, pokud je potřeba. Například při použití obyčejného nebo kříženého kabelu.

Modul EUSART používá pro nastavení tři registry. Registr stavu a řízení přenosu dat (TXSTA), registr stavu a řízení příjmu dat (RCSTA) a registr řízení přenosové rychlosti (BAUDCON). [3]

Pro přenos v asynchronním režimu používáme dva piny mikroprocesoru RC6/TX/CK a RC7/RX/DT.

Tab. 3.4: Konfigurace modulu EUSART v PIC18F4420

Parametr *	Popis
USART_TX_INT_OFF	Vypnutá generace přerušení při odesílaní dat
USART_RX_INT_ON	Zapnutá generace přerušení při příjmu dat
USART_ASYNCH_MODE	Zapínaní asynchronního režimu práce
USART_EIGHT_BIT	Osmibitová velikost paketu
USART_CONT_RX	Neustálý příjem dat
USART_BRGH_LOW	Nízká rychlost komunikace
6 **	Dělič rychlosti

* Parametr ze standardní knihovny usart.h kompilátoru MPLAB C18

** Dělič rychlosti – koeficient se používá pro nastavení rychlosti, blízké k předem určené

Jelikož jsem zvolil nízkorychlostní komunikaci, používám následující vztah pro výpočet děliče rychlosti:

$$Baud\ Rate = F_{OSC}/[64(n+1)]\,, \tag{3.1}$$

kde je:

Baud Rate – požadovaná rychlost,

F_{OSC} – pracovní kmitočet mikroprocesoru,

n – dělič rychlosti.

Pro náš případ je výpočet následující:

$$Baud\ Rate = 1000000/[64(n+1)]. \tag{3.2}$$

Z tohoto vztahu se dá přibližně vypočíst přenosová rychlost a dělič rychlosti. Z různých možností bude nejlepší ta varianta, kde bude nejmenší chyba:

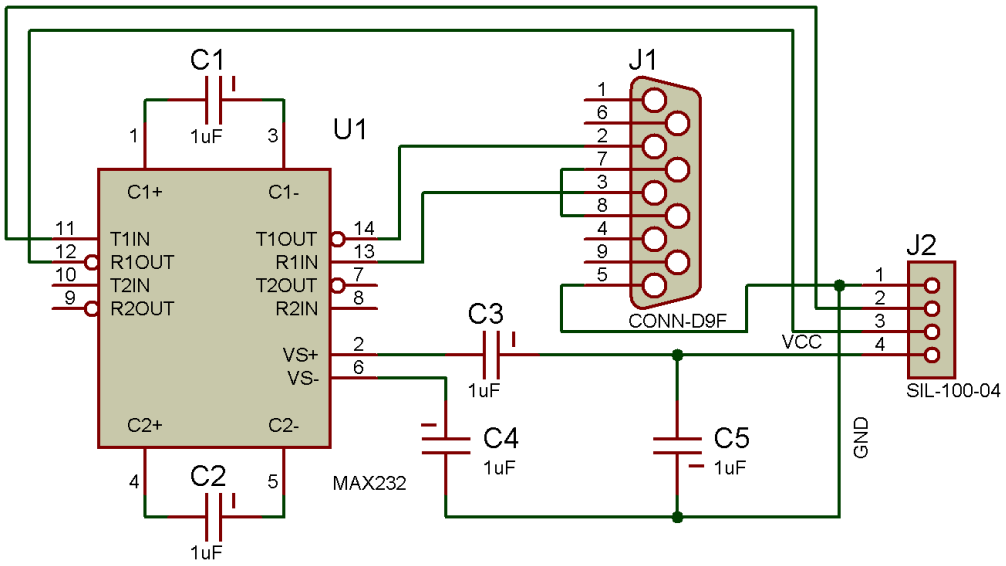
$$Chyba = \big(Vypočtený\ Baud\ Rate - Žádaný\ Baud\ Rate\big)/Žadaný\ Baud\ Rate. \tag{3.3}$$

Žádané přenosové rychlosti jsou předem určené a jsou následující 300bd, 1200bd, 2400bd, 4800bd, 9600bd, 19200bd, 57600bd, 115200bd.

Ověřil jsem v praxi všechny možné rychlosti pro zadaný režim a zadaný kmitočet. Správně komunikace fungovala jenom v jednom případě a navíc tato možnost měla největší chybu ze všech. Tak jsem zvolil rychlost 2400bd a dělič rychlosti jsem nastavil na hodnotu 6. Při takových hodnotách rychlosti je chyba 6.99%.

Bohužel není možné připojení výstupů mikroprocesoru přímo na RS232 port. Proto je potřeba použít převodník na převádění RS232 signálu na TTL/CMOS logiku. Pro tento účel jsem navrhnul další desku na základě součástky od firmy Maxim Integrated MAX232, která se běžně používá pro tyto účely.

3.4.2 Dodatečné schéma pro RS-232 komunikace



Obr. 3.13: Schéma pro komunikaci po sériové lince podle standardu RS-232

Na obrázku 3.13 je konektor J2 pro připojení desky na základní desku robota. Označení výstupů konektorů:

1 – pin č. 1 na J7 (viz Obr. 4.2 schéma elektrického obvodu) – GND

2 – pin č. 2 na J7 – Tx výstup mikroprocesoru

3 – pin č. 3 na J7 – Rx vstup mikroprocesoru

4 – pin č. 2 na J2 – +5V

Pro připojení přes COM port jsem použil devítipinový D-Sub konektor DE-9F označený na obrázku 3.13 jako J1.

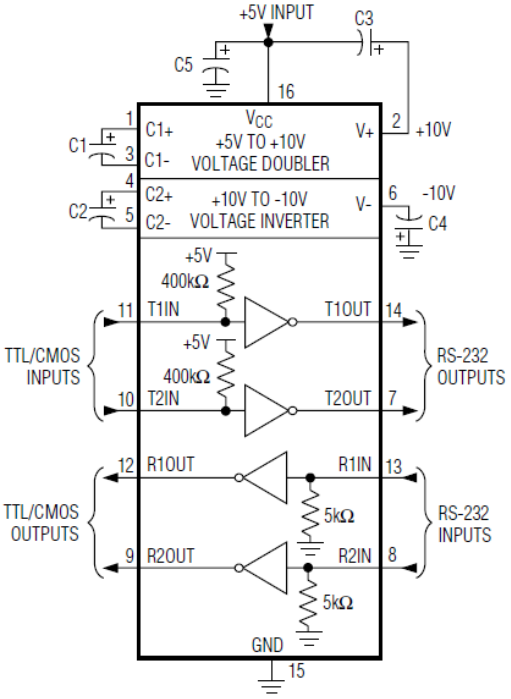
Důležitě je pro připojení používat nekřížený kabel, protože jsem provedl křížení přímo na desce.

Integrovaný obvod MAX232 jsem použil pro převádění signálu Rx a Tx z TTL/CMOS logiky na logiku RS-232. Tento integrovaný obvod zajišťuje úroveň výstupného napětí RS-232 na přibližně 10 voltů převáděním vstupního napětí 5V pomocí nábojové pumpy. Použití tohoto integrovaného obvodu zjednodušuje realizaci RS-232 rozhraní v zařízeních. Jednoduchost je způsobena tím, že není potřeba komplikovat napájecí zdroje pouze pro využití v RS-232. Občas může vstupní napětí z některých přístrojů RS-232 dosáhnout 25V. MAX232 ho sníží na standardní

úrovně napětí v TTL logice 5V. Obvod dovoluje převádět signály ze dvou USART modulů. Pro mou aplikaci jsem použil pouze polovinu obvodu.

MAX232 obsahuje dvě nábojové pumpy, které převádějí +5V na ±10V, což je úroveň signálů, vhodných pro rozhraní RS-232. Kondenzátor C1 se používá pro převádění +5V na +10V, a kondenzátor C2 pro inverzi +10V na -10V. [7]

Až po vyrobení desky, jejíž schéma je zobrazeno na obrázku 3.13, jsem zjistil, že by bylo lepší použít novější integrovaný obvod od stejného výrobce MAX232A, který je plně kompatibilní s MAX232, jen využívá kondenzátory menší kapacity (0.1µF) a má větší přenosovou rychlost. To by dovolilo použit keramické kondenzátory místo elektrolytických a zvětšit přenosovou rychlost až na 200kbps (při MAX232 – 64kbps).



Obr. 3.14: Schéma zapojení integrovaného obvodu MAX232 [7]

Všechny kondenzátory nábojové pumpy pro tento obvod musí mít hodnotu 1µF. [7]

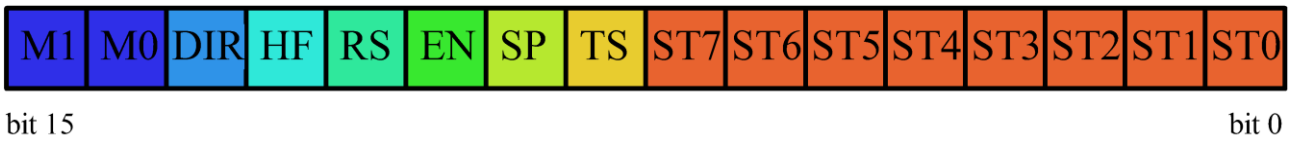
3.4.3 Popis rozhraní pro řízení pohybu robota

Podle zadání musí robot dostávat řídící příkazy z vnějšího přístroje. Pro tuto komunikaci jsem zvolil standard RS-232 a rozhodl jsem se, že pro ukázku funkčnosti vyrobeného řídicího systému budu používat počítač jako vnější přístroj. Pro tuto komunikaci jsem však ještě musel navrhnout a realizovat vlastní rozhraní. Podrobný popis rozhraní je uvedený v této části.

Pro přenos dat jsem použil protokol RS-232. Popisovat základy tohoto protokolu nebudu. Pouze napíšu, že ho používám pro přenos symbolů po sériové lince. Jeden symbol představuje jeden bajt. Dohromady se jeden příkaz robotu skládá ze 16 bajtů, což znamená 16 symbolů. Každý symbol může mít hodnotu „0“ nebo „1“. Pokud převedeme tyto symboly podle ASCII tabulky do šestnáctkové soustavy (v této soustavě se zobrazuje paměť mikroprocesoru), tak uvidíme, že nule odpovídá číslo 30h, a jedničce 31h. Pro řízení z jiného přístroje, musíme předávat tato čísla. To znamená, že pro řízení robota stačí poslat 16 bajtů za sebou, a v každém bajtu musí byt číslo 30h nebo 31h. Po přenosu příkazů robota, se symboly převádějí na dvojkovou soustavu v takovém smyslu, že jednomu symbolu (jednomu bajtu) odpovídá jeden bit. Symbolu „0“ odpovídá 0b, symbolu „1“ odpovídá 1b. Pak už se 2 bajty zapisují do EEPROM.

Použití takového způsobu přenosu dat chrání před špatným zpracováním řídicího příkazu robotem, v případě ztráty dat nebo chyby při přenosu dat. Mohl jsem umístit 16 řídících bitů do dvou bajtů a předat robotu jenom dva bajty místo 16. Kdyby však při přenosu dát došlo ke změně jednoho bitu, robot by to nezjistil a vykonával by špatný příkaz, který po něm uživatel nechtěl. Ve způsobu, který jsem použil, v případě, že dojde ke změně jednoho bitu, změní se význam celého bajtu, i když to bude jiný znak podle ASCII tabulky mimo „0“ nebo „1“, tak robot nebude mít možnost ho převést, a nic dělat nebude. Pravděpodobnost toho, že se místo symbolu „0“ při vzniku chyby objeví znak „1“ nebo naopak, je velice malá. Při zkouškách vůbec nedošlo ke vzniku chyby při přenosu dat.

Níže na obrázku 3.15 jsou zobrazeny funkce každého bitu řídicího registru.



Obr. 3.15: Řídicí registr motoru

Tento registr se skládá ze dvou 8 bitových registrů. Do jedné části se zapisují příkazy motorů, do druhé části se zapisuje velikost vzdálenosti posunutí motoru.

bit 15-14 **M1:M0**: Bity výběru motoru, pro který posíláme řídicí příkaz

00 = Motor COXA

01 = Motor FEMUR

10 = Motor TIBIA

11 = Motor CLAWS

bit 13 **DIR**: Bit výběru směru pohybu motoru

Kdy M1:M0 = 00:

1 = Ve směru hodinkových ručiček

0 = Proti směru hodinkových ručiček

Kdy M1:M0 = 01:

1 = Nahoru

0 = Dolu

Kdy M1:M0 = 10:

1 = Nahoru

0 = Dolu

Kdy M1:M0 = 11:

1 = Otevírat

0 = Zavírat

bit 12 **HF**: Bit pro výběr režimu krokovaní motoru

1 = Půlkrokový režim

0 = Celokrokový režim

bit 11 **RS**: Bit resetování motoru

1 = Neresetovat motor

0 = Resetovat motor

bit 10 **EN**: Bit povolení pohybu

1 = Pohyb povolený

0 = Pohyb není povolený

bit 9 **SP**: Bit výběru rychlosti

1 = Vysoká rychlost

0 = Nízká rychlost

bit 8 **TS**: Bit zastavovaní všech motoru

1 = Zastavit všechny motory

0 = Nezastavovat motory

bit 7-0 **ST7:ST0**: Bity pro zadání vzdálenosti pohybu motoru

11111111 = Maximální vzdálenost

.

.

.

0000001 = Minimální vzdálenost

Pro každý motor v paměti mikroprocesoru je vyhrazený řádek, kde jsou uložené hodnoty pro řízení jeho pohybu. Z této paměti bere řídící program hodnoty pro nastavování řídících bitů mikroprocesoru a dalších řídících součástek.

Bity M1:M0 slouží pro výběr řádku, do kterého budou zapsané další data. Například, když po sériové lince přijde následující příkaz: (pro dlouhá čísla dále budu používat šestnáctkovou soustavu pro lepší přehled).

30	30	31	31	31	31	30	30	30	30	30	30	30	31	31	31

Do buňky paměti EEPROM s adresou 80h se zapíše požadována rychlost motoru COXA, do paměti s adresou 81h se zapíše směr pohybu motoru COXA, v daném případě ve směru hodinkových ručiček. Do adresy 82h se zapíše 1, což bude znamenat půlkrokový režim pohybu. Do adresy 83h se uloží 1, což znamená neresetovat motor. Do adresy 84h také se uloží 1, což bude znamenat povolení pro začátek pohybu. Do adresy 86h se uloží číslo 00000111b, což bude znamenat pohyb o vzdálenost $^{7}/_{256}$ od maximální vzdálenosti.

Tab. 3.5: Část EEPROM paměti mikroprocesoru PIC18F4420

ADR (HEX)	0	1	2	3	4	5	6
	RYCHLOST	SMĚR	PŮLKROK	RESET	POVOLENI	NEPOUŽÍVÁ SE	DISTANCE
080	COXA	COXA	COXA	COXA	COXA		COXA
090	FEMUR	FEMUR	FEMUR	FEMUR	FEMUR		FEMUR
0A0	TIBIA	TIBIA	TIBIA	TIBIA	TIBIA		TIBIA
0B0	CLAWS	CLAWS	CLAWS	CLAWS	CLAWS		CLAWS

Bit RS se používá pro udržení motoru v režimu resetu. Ve skutečnosti to znamená, že se motor nachází v režimu brzdění. Například, když chceme zabrzdit motor, bit RS musí byt nastavený na 0 a bit EN na 1. V případě, že chceme, aby se motor točil, musí být bity RS a EN nastavené na 1. Když je bit EN nastavený na 0, zbytek bajtu nemá žádný význam, protože pohyb motoru není povolený.

V okamžiku, kdy po sériové lince přijde příkaz, hlavní program se zastaví a mikroprocesor začne zpracovávat přerušení. V tomto přerušení mikroprocesor zpracuje data, převede je do tvaru vhodného pro použití a uloží data do paměti. Poté už začíná pracovat dál s novými daty. To znamená, že uživatel může v jakýkoliv okamžik zasáhnout do práce robota a zastavit ho nebo poslat nový řídící příkaz, který robot začne ihned splňovat.

Bit TS má vliv na každý motor a nezávisí na tom, pro který motor je určen daný řídící příkaz. Když tento bit bude mít hodnotu 1, tak se všechny motory zastaví. Když bit TS bude mít hodnotu 0, tak nemá vliv na práci žádného z motorů.

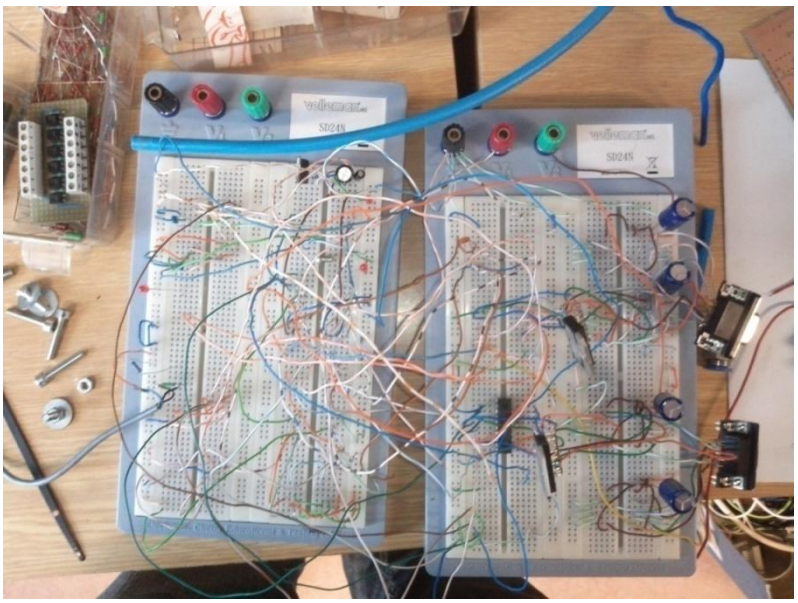
4 VÝROBA DESKY PLOŠNÝCH SPOJŮ

Na začátku návrhu desky jsem musel vyřešit největší problém – zvolit základní řídicí jednotku, jádro systému, které by umožnilo využít co nejvíce možností krokových motorů. Na tomto řešení záviselo, jak bude vypadat celý systém, jaké budou způsoby řízeni, jak bude zatížený mikroprocesor, rychlost celého systému a tak dále. Tím jádrem se stalo řešení od ST Microelectronics – tři součástky, které byli speciálně projektované pro řízení krokových motorů. Je to driver L297 + L298 + L6210. Podrobněji jsou popsány ve třetí kapitole.

4.1 Zkoušky na nepájivém kontaktním poli

Zkoušel jsem životaschopnost tohoto řešení, na začátku pro jeden motor, na nepájivém poli. Na této etapě jsem se setkal se spoustou různých druhů problémů. Například prvním vážným problémem byla těžká inicializace modulů mikroprocesoru, s kterým jsem se setkal poprvé. Následujícím problémem byla úplná nefunkčnost prvního motoru, s kterým jsem začal pracovat. Tento problém byl způsobený tím, že robot nebyl velmi dlouho používán a kvůli tomu se občas ztrácel kontakt mezi motorem a řídicím polem. A pro krokový motor se takové druhy problémů dají těžko zjistit. Setkával jsem se se spoustou dalších problémů, které musel jsem vyřešit.

Poté jsem zvětšoval nepájivé pole, přidával jsem další moduly pro zlepšení funkčnosti systému, rozšiřoval jsem systém pro ovládání čtyř motorů současně. A to trvalo do doby, kdy jsem se dostal do takového stavu, ve kterém už nebyla možná práce na nepájivém poli. Počet drátků už byl větší, než 150, a nebyla už žádná možnost s tím nějak rozumně pracovat. Na této etapě jsem se rozhodl začít vyrábět první verzi desky plošných spojů.



Obr. 4.1: Nepájivé kontaktní pole bez většiny součástek

4.2 Základní elektrické zapojení a seznam součástek

Pro výrobu desky jsem musel mít hotové celé základní elektrické schéma, které je uvedeno na obrázku 4.2.

Tab. 4.1: Seznam součástek

Označení	Název / Jmenovitá hodnota	Počet	Poznámka
	Integrované obvody		
U1, U7, U12, U17	L297 – řadič krokových motorů	4	ST Microelectronics
U2, U8, U13, U18	L298 – dvojice H-můstků	4	ST Microelectronics
U3, U9, U14, U19	L6210 – dvojice můstků se Schottkyho diodami	4	ST Microelectronics
U4	TL431 – třísvorkový napěťový regulátor	1	Texas Instrument
U5, U11, U15, U20	NE555 – precizní časovač	4	Texas Instrument
U6, U16	DS1267-100 – digitální potenciometr	2	Dallas Semiconductor
U10	PIC18F4420 – mikroprocesor	1	Microchip
	Rezistory		
R1	22KK	1	
R2	K12K	1	
R3, R4, R8, R9, R17, R18, R30, R31	1RJ	8	
R5, R10, R22, R29	1KK	4	
R6, R7	3K3K	2	
R11, R13, R15, R19, R21, R23, R25, R27	10KK	8	
R12, R14, R16, R20, R24, R26, R28,	K30K	7	
RV1	10K	1	
	Kondenzátory		
C1, C4, C8, C12	470u	4	
C2, C5, C7, C9, C11, C13, C15, C17	100n	8	
C3	3300p	1	
C6, C10, C14, C16	10n	4	

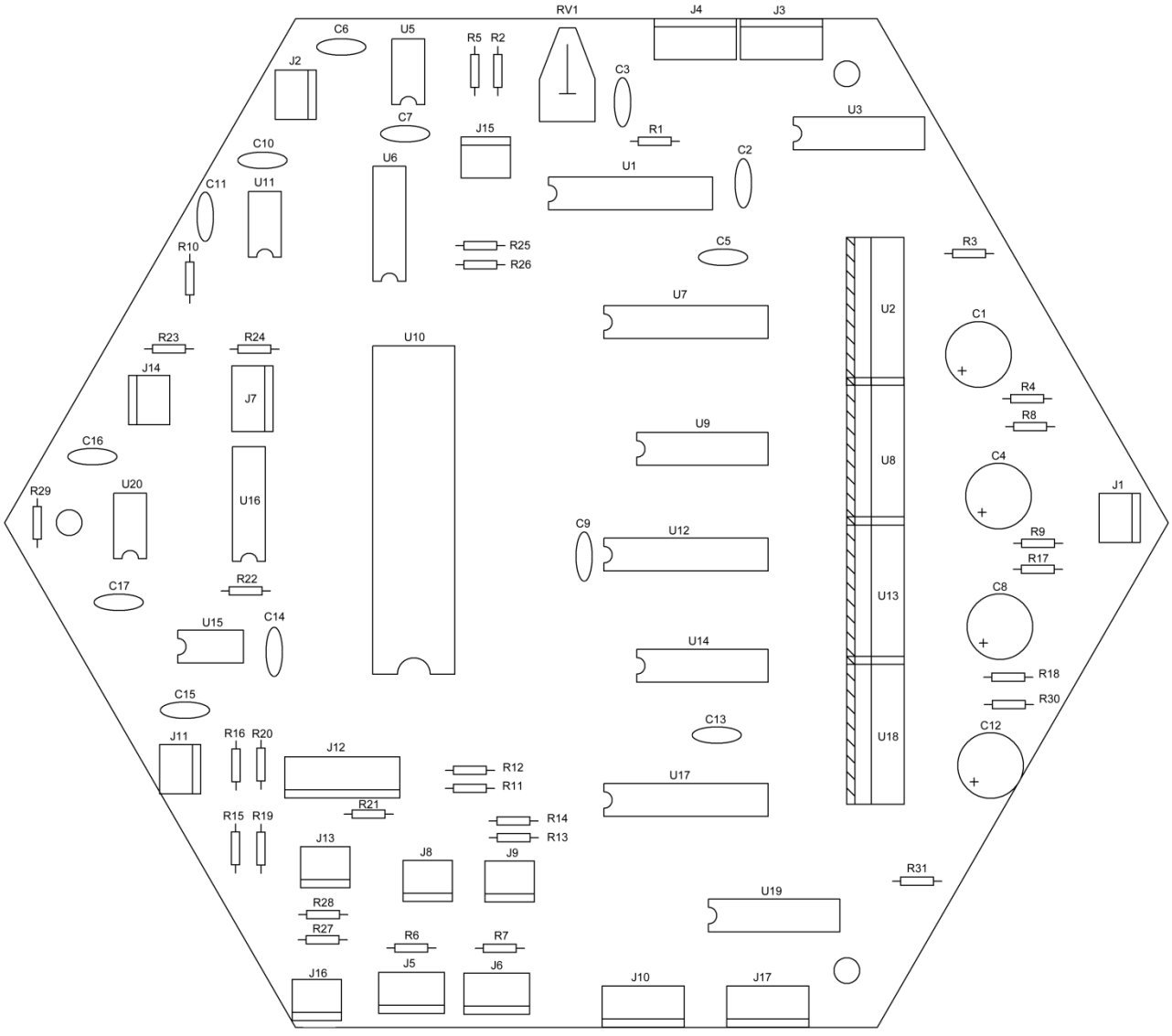
Pokračovaní tab. 4.1: Seznam součástek

Označení	Název / Jmenovitá hodnota	Počet	Poznámka
	Konektory		
J1, J2, J8, J9, J11, J13-J16	SIL-100-02	9	
J3, J4, J10, J17	SIL-100-04	4	
J5-J7	SIL-100-03	3	
J12	SIL-100-06	1	

4.3 Deska plošných spojů

Na základě schématu, které je na obrázku 4.2, jsem musel udělat rozvody vodičů pro výrobu desky plošných spojů ve formátu, který by mohl byt dál použitý speciálními přístroji. Nejčastěji používaný je formát GERBER, ve kterém jsou příkazy stroji zapsané jako jednoduché textové příkazy.

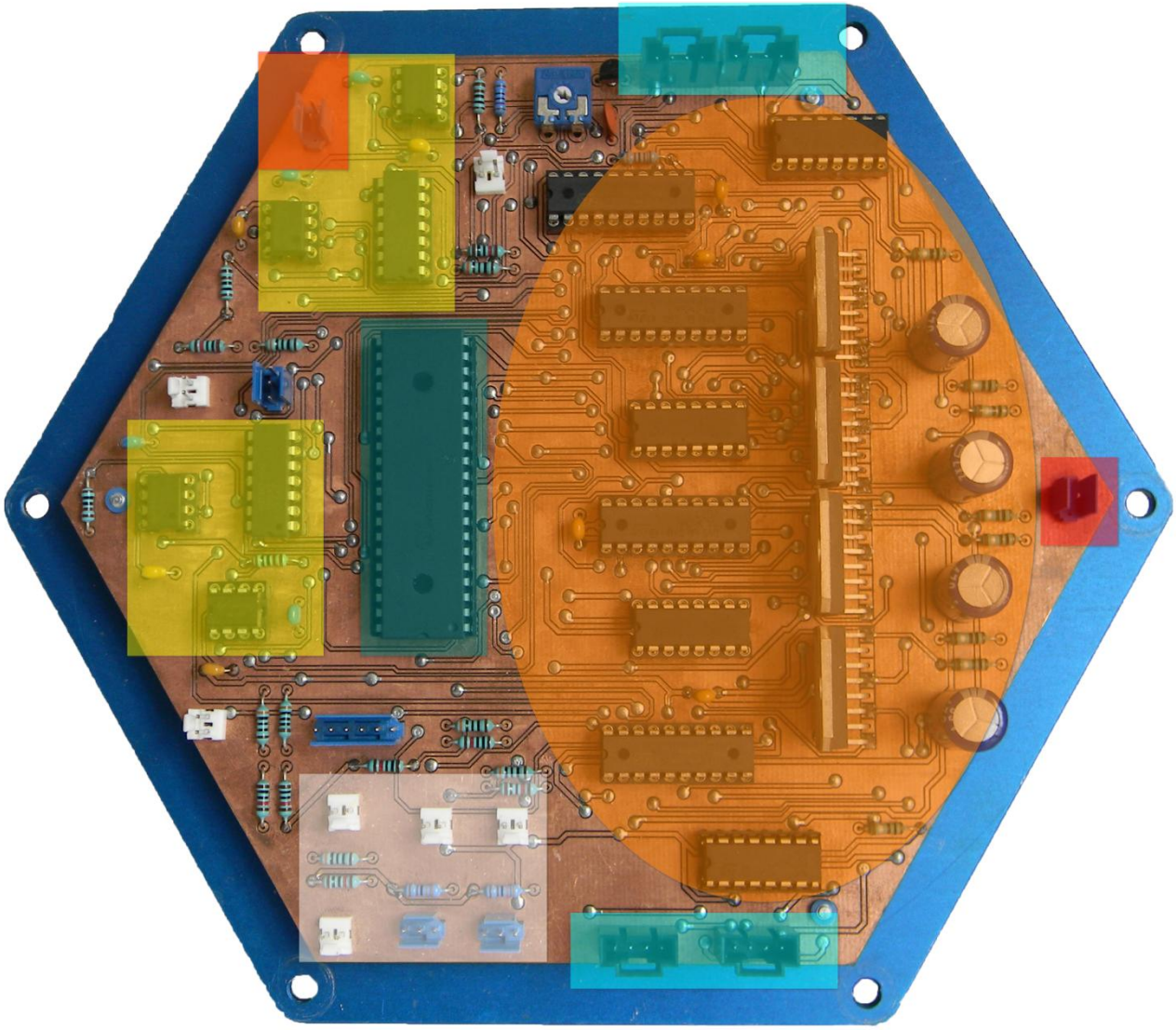
Základním pravidlem, které jsem musel dodržovat při výrobě této desky, byla nutnost jejího umístění v dolní krabici, pod robotem. Dolní krabice je šestiúhelník s délkou hrany 100 mm. Proto jsem musel projektovat dvouvrstvou desku, do které se mi s velkými problémy podařilo umístit všechny součástky.



Obr. 4.3: Umístění součástek na desce plošných spojů

Deska, která je zobrazena na obrázku 4.3, je finální verze, která je teď použita v robotu. Byla vyrobená na Katedře aplikované kybernetiky na speciálním stroji od firmy LPKF, který je také pohaněn krokovými motory.

To je už třetí verze desky. První byla vyrobená podle špatného návrhu a po jejím vyrobení na výše uvedeném stroji, jsem zjistil, že otvory pro nýtování byly menší než nýty, a nedalo se spojit horní a dolní vrstvu desky. Pak zde byla druhá verze desky. Po připájení všech součástek, jsem při zkouškách zjistil, že některé drážky vodičů jsou propojené mezi sebou. Najít některá místa propojení se mi nepodařilo. Dospěl jsem k závěru, že se při pájení příliš ohřívala některá pájecí místa a cín protekl pod drážky, kde ho nebylo vidět. Proto jsem se po několika týdnech hledání chyby a zničení dvou součástek L298 rozhodl vyrobit další desku, která už byla poslední a plně funkční.



Obr. 4.4: Deska plošných spojů s označením jednotlivých modulů

Na tomto obrázku je zobrazena hotová deska se všemi součástkami, která je na kryt dolní krabice umístěná pomocí spojovacích sloupků. Také jsou na obrázku barevně označené jednotlivé moduly řídící desky:

- Zelený obdélník – mikroprocesor (kapitola 3.1)
- Oranžový ovál – ovladač krokových motorů (kapitola 3.2)
- Dva žluté obdélníky – moduly řízení rychlosti (kapitola 3.3)
- Dva červené obdélníky – konektory napájeni
- Dva modré obdélníky – konektory motorů
- Bílý obdélník – konektory koncových spínačů

Tab. 4.2: Popis konektorů desky plošných spojů

Označení	Popis	Počet pinů
J1	Napájení motorů 24V	2
J2	Napájení součástek desky 5V	2
J3	Konektor motoru COXA	4
J4	Konektor motoru FEMUR	4
J5	Konektor pro připojení indukčního čidla č. 1 (IČ1)	3
J6	Konektor pro připojení indukčního čidla č. 2 (IČ2)	3
J7	Konektor pro připojení desky RS-232	3
J8	Konektor pro připojení koncového spínače č. 1 (KS1)	2
J9	Konektor pro připojení koncového spínače č. 2 (KS2)	2
J10	Konektor motoru TIBIA	4
J11	Konektor pro připojení koncového spínače č. 3 (KS3)	2
J12	Konektor programátoru PRESTO	6
J13	Konektor pro připojení koncového spínače č. 4 (KS4)	2
J14	Konektor pro připojení koncového spínače č. 5 (KS5)	2
J15	Konektor pro připojení koncového spínače č. 6 (KS6)	2
J16	Konektor pro připojení koncového spínače č. 7 (KS7)	2
J17	Konektor motoru CLAWS	4

5 POPIS PROGRAMU

Pro vytvoření řídícího programu pro mikroprocesor PIC18F4420 jsem použil programové prostředky od stejného výrobce, jako procesor. Jako IDE jsem použil program od Microchip MPLAB X IDE v1.70. Kompilátor je také od Microchip pro práci s mikroprocesorem řady PIC18 MPLAB C18.

Celý program je napsaný v jazyce C s použitím některých standardních knihoven od Microchipu. Verze obou programů, které jsem použil, jsou bezplatné, ale mají některá omezení. Například v kompilátoru MPLAB C18 po třech měsících práce přestává fungovat debugger. Pro mě však tyto možnosti nebyly tak důležité.

5.1 Knihovny, použité v programu

Pro svůj program jsem používal některé knihovny, které jsou součástí kompilátoru.

```
#include <p18f4420.h>          //Pripojeni zakladni knihovny pro PIC18F4420
#include <delays.h>            //Pripojeni knihovny delays.h
#include <spi.h>               //Pripojeni knihovny spi.h
#include <usart.h>             //Pripojeni knihovny usart.h
```

Knihovna P18F4420.h je základní knihovna pro specifikaci procesoru z řady PIC18. Obsahuje periferní rutiny a definuje všechny speciální funkční registry (SFR). Periferní rutiny, které jsou v knihovně určené pro použití periferních přístrojů a definici periferních rozhraní. [10]

Knihovnu delays.h používám pro zastavení procesoru na určitou dobu. Funkce z této knihovny spouští speciální kód, který splňuje určitý počet cyklů. A po tu dobu procesor pracuje nad výpočtem tohoto kódu a zastaví běh základního programu. [10]

Tab. 5.1: Funkce knihovny delays.h [10]

Funkce	Popis
Delay1TCY	Pauza o hodnotě jeden instrukční cyklus
Delay10TCYx	Pauza o hodnotě 10 instrukčních cyklů
Delay100TCYx	Pauza o hodnotě 100 instrukčních cyklů
Delay1KTCYx	Pauza o hodnotě 1.000 instrukčních cyklů
Delay10KTCYx	Pauza o hodnotě 10.000 instrukčních cyklů

Například pro generaci pauzy o hodnotě 100 tisíc instrukčních cyklů Delay10KTCYx(10).

Knihovnu spi.h jsem používal pro inicializaci SPI sběrnice, kterou jsem použil pro komunikaci s digitálními potenciometry (viz kapitolu 3.3). Použil jsem pouze jednu funkci:

```
OpenSPI(SPI_FOSC_64,MODE_00,SMPEND); //Initializace SPI sbernice
```

- SPI_FOSC_64 – kmitočet = $F_{OSC}/64$
- MODE_00 – SPI Master Mode
- SMPEND – vzorkovaní vstupních dat na konci přenosu

Knihovna Usart.h je určena pro práci s modulem pro sériový port EUSART (viz kapitolu 3.4). Funkce pro inicializaci modulu:

```
//--------------------Inicializace USART-------------------------
    OpenUSART(USART_TX_INT_OFF &      //Vypnut preruseni pri posilani dat
              USART_RX_INT_ON &       //Zapnut preruseni pri prijemu  dat
              USART_ASYNCH_MODE &     //Zapnut asynchronni rezim
              USART_EIGHT_BIT &       //8-bit velikost paketu
              USART_CONT_RX &         //Rezim prijimani dat
              USART_BRGH_LOW,6);      //Delic rychlosti
```

5.2 Přerušení

Mikroprocesory řady PIC18F mají dva vektory přerušení. Přerušení vyšší priority a přerušení nižší priority. Vektor přerušení vyšší úrovně je 0008h a nižší úrovně 0018h. Přerušení vyšší priority muže byt zpracováno, když se vykonává přerušení nižší úrovně. Ve skutečnosti to znamená, že když bit ve speciálním registru, který ukazuje, že je splněná podmínka pro generaci přerušení – je na hodnotě 1, tak mikroprocesor přeruší splnění základního programu. Pak začne splnění programu z adresy 0008h v případě, že přerušení bylo vysoké úrovně nebo z adresy 0018h, pokud přerušení bylo nízké úrovně.

V mikroprocesoru se mohou za určitých podmínek generovat přerušení ze spousty modulů, ale potřeboval jsem pro svou práci jen přerušení z Timer1 a modulu EUSART. Pro modul EUSART (komunikace s vnějším přístrojem) jsem zvolil přerušení s vysokou prioritou, pro časovač jsem zvolil přerušení nízké úrovně.

Pro zapnutí a nastavení přerušení existují speciální registry procesoru INTCON, INTCON1, INTCON2, PIR1, PIR2, PIE1, PIE2, IPR1, IPR2, RCON. Některé z nich jsem použil. Například pro nastavení přerušení z časovače jsem ho musel nejprve inicializovat:

```
//------Timer1 Nastaveni Preruseni--------------------------
  INTCONbits.GIEH = 0;          //Vypnout globalni preruseni
  INTCONbits.GIEL = 0;          //Vypnout preruseni nizkeho urovne
  IPR1bits.TMR1IP = 0;          //Nastaveni preruseni timeru1 na
                                //nizky prioritni uroven
  PIR1bits.TMR1IF = 0;          //Smazat flag preruseni timeru1
  PIE1bits.TMR1IE = 1;          //Povolit preruseni od timeru1
  INTCONbits.GIEL = 1;          //Povolit preruseni nizkeho urovne
  INTCONbits.GIEH = 1;          //Povolit globalni preruseni
```

Jak vidíme, pro inicializaci přerušení z nějakého modulu, musíme provést kroky v následujícím pořadí:

- Vypnout všechny přerušení (vysoké a nízké úrovně)
- Vybrat úroveň přerušení (vysoká nebo nízká úroveň)
- Smazat flag přerušení (vynulovat bit, který ukazuje, že došlo k přerušení)
- Povolit přerušení z potřebného modulu
- Zapnout všechny přerušení (vysoké a nízké úrovně)

Vypínáme všechny přerušení kvůli tomu, aby proces inicializace nerušilo nějaké už zapnuté přerušení. Pravděpodobnost, že toto může nastat, je velmi malá, ale pro zvýšení spolehlivosti programu je lepší to udělat.

Nastavení přerušení z modulu EUSART při přijmu signálů z počítače:

```
//---------------USART Nastaveni preruseni----------------------------------
  INTCONbits.GIEH = 0;          //Vypnout globalni preruseni
  IPR1bits.RCIP = 1;            //Nastavit preruseni z USART
                                //na velky prioritetni uroven
  PIR1bits.RCIF = 0;            //Smazat flag preruseni
  RCONbits.IPEN = 1;            //Povolit urovni preruseni
  PIE1bits.RCIE = 1;            //Povlit preruseni od USART
  INTCONbits.GIEH = 1;          //Povolit globalni preruseni
```

Jak je vidět zde jsem použil stejný postup nastavení přerušení. Liší se pouze jednotlivé bity, které závisí na tom, pro jaký modul přerušení zapínáme.

Pro volání funkce přerušení vysoké úrovně jsem použil následující postup příkazů:

```
#pragma code InterruptVectorHigh = 0x000008
void
InterruptVectorHigh (void)
{
  _asm
    goto InterruptHandlerHigh         //Skok na preruseni vysokeho urovne
  _endasm
}
```

Zde je vidět, že pro zpracování přerušení, program skočí na adresu paměti 0x000008, která je začátkem rutiny přerušení vysoké úrovně.

V této rutině program zpracovává informace, které získal z počítače. Například se převádí 16 znaků (bajtů) na 2 bajty informace, kterou potřebuje pro řízení motoru (viz kapitolu 3.4.3). Pro další využití se ukládají data do EEPROM paměti. Také se nastavují rychlosti pro motory. Dále je uvedený přiklad pro konverzi jednoho bajtu dat, které mikroprocesor získává z počítače:

```
for(i=7;i>(-1);i--)             // Cyklus konvertace dat do vhodneho
{                               // pracovniho formatu
chr = read_eeprom(read_address); // Cteme data z EEPROM
num = chr - '0';                // Prevadime data z ASCII kodu, ve kterem ony
                                // prisly na COM port, na numericky format
res = num << i;                 // Posouvame bit doleva o i
sum = buffer | res;             // Pobitova operace OR nad buffrem a vysledkem
buffer = sum;                   // Vysledek zapisujeme do bufru
write_eeprom(write_address,res); // Ulozime vysledek kazde iterace do eepromu
                                // pro kontrolu   (V provozu se da vymazat)
read_address++;                 // Inkrementace adresy datoveho prostoru pro
                                // cteni dat z EEPROM
write_address++;                // Inkrementace adresy datoveho prostoru pro
                                // zapis dat do EEPROM
busy_eeprom();                  // Cekame na ukonceni operaci s EEPROM
}
write_eeprom(0x0F0,sum);        // Zapis celkoveho vysledku konvertace dat do
              //EEPROM (Musime dostat presne to co jsme posilali v jednem bytu)
```

5.3 Funkce

V programu pro řízení robota se dají najít některé funkce. Níže můžete vidět jejich prototypy.

```
void spi(char);                          //SPI Funkce pro posilani dat
void speed(int);                         //SPI Funkce pro vyber odporu potenciometru
void write_eeprom( int badd, char edata );  //Funkce pro zapis do EEPROM pameti
char read_eeprom( int badd );               //Funkce pro cteni z EEPROM pameti
void busy_eeprom ( void );                  //Funkce pro zjisteni stavy EEPROM
void move(int);                             //Funkce pro pohyb motoru
```

Funkce SPI se používá pro převod dat typu char do registru buffru spi, odkud se postupně předávají přes spi sběrnici.

```
//------Funkce posilani dat po SPI sbernice-------------------
void spi (char myData){

    SSPBUF = myData;         //Zapsat data v bufr SPI BUS
    while(!SSPSTATbits.BF); //Pockat na prenos vsech dat
}
```

Bit BF v registru SSPSTAT ukazuje na to, že přenos dát ještě trvá. Jakmile se bit BF bude rovnat 1, program ukončí cyklus, a to bude znamenat ukončení přenosu dat. [3]

Funkce SPEED je určena pro zadání rychlosti motorů. Kvůli digitálním potenciometrům, které se ve skutečnosti skládají ze dvou potenciometrů, jsem musel udělat funkci, která bude posílat hodnoty současně pro dva motory. Přestože chceme změnit rychlost pouze jednoho motoru, stejně posíláme hodnoty pro dva. Je to způsobené architekturou digitálních potenciometrů (viz kapitolu 3.3.2).

Pro vyřešení tohoto problému jsem to udělal tak, že ukládám hodnoty žádaných rychlostí do EEPROM. Toto ukládání se provádí, když se generuje přerušení ze sériového portu. A když se provádí funkce spi, program posílá hodnoty z EEPROM přes SPI sběrnici do registrů digitálních potenciometrů.

```
//------Funkce rizeni rychlosti motoru-------------------
void speed(int ds ){

    char coxa_speed;
    char femur_speed;
    char tibia_speed;
    char claws_speed;

       coxa_speed  = read_eeprom(0x080);   //Cteni rychlosti COXA  z EEPROM
       femur_speed = read_eeprom(0x090);   //Cteni rychlosti FEMUR z EEPROM
       tibia_speed = read_eeprom(0x0A0);   //Cteni rychlosti TIBIA z EEPROM
       claws_speed = read_eeprom(0x0B0);   //Cteni rychlosti CLAWS z EEPROM
```

V této části funkce speed čte hodnoty z EEPROM a zapisuje je do proměnných.

```
    if (ds == 0){           //Vyber Potenciometru c.0 a c.1
        PORTEbits.RE0 = 1;  //Zacit posilani dat
//       Stack Registr (bit 0)
        spi(0b00000000);
//       Potentiometr #1 (bits 1-8)
        spi(coxa_speed);
//       Potentiometr #0 (bits 9-16)
        spi(femur_speed);
        PORTEbits.RE0 = 0;  //Ukoncit posilani dat
    }
    else{                   //Vyber Potenciometru c.2 a c.3
        PORTEbits.RE1 = 1;  //Zacit posilani dat
//        Stack Registor (bit 0)
        spi(0b00000000);
//        Potentiometr #3 (bits 1-8)
        spi(claws_speed);
//        Potentiometr #2 (bits 9-16)
        spi(tibia_speed);
        PORTEbits.RE1 = 0;  //Ukoncit posilani dat

    }
}
```

Zde je vidět, že pro výběr digitálního potenciometru používám vstupy z mikroprocesoru RE0 a RE1. Je to kvůli tomu, že mám dvě součástky na jedné sběrnici a musím zapínat ten digitální potenciometr, na kterém chceme změnit odpor.

Když je povolený zápis do registru potenciometru, začínám posílat tři bajty dat. První bajt je nula, protože nepoužívám možnost zapojení dvou potenciometrů do jedné řady. Druhý a třetí bajt už jsou hodnoty pro dva fyzické potenciometry, které jsou uvnitř.

Funkce WRITE_EEPROM se používá pro zápis dat do EEPROM paměti. Je to velmi důležitá funkce, protože ji používám docela často v celém programu. Základ této funkce jsem převzal ze standardních funkcí kompilátoru od Microchipu. Pak jsem ji upravil pro své potřeby.

```
void write_eeprom( int badd,char edata )    //Funkce zapisu do EEPROM
{
        EEADR = badd;               // Registr adresy
        EEDATA = edata;             // Registr dat
        EECON1bits.EEPGD = 0;       // Vyber EEPROM pameti
        EECON1bits.CFGS = 0;        // Vyber EEPROM nebo FLASH
        EECON1bits.WREN = 1;        // Dovolit zapis v cyklu
        INTCONbits.GIE = 0;         // Zabranit preruseni
        EECON2 = 0x55;              // Neni fizicky registr
        EECON2 = 0xAA;              // Neni fizicky registr
        EECON1bits.WR = 1;          // Iniciace zapisu
        while(EECON1bits.WR);       // Pockat ukonceni zapisu
        INTCONbits.GIE = 1;         // Povolit preruseni
        EECON1bits.WREN = 0;        // Zabranit zapis do EEPROM
}
```

Na vstup funkce musíme přidat adresu a data, které chceme z této adresy uložit. Paměť pro uživatele začíná od adresy 080h. Paměť do uvedené adresy je vysokorychlostní paměť, kterou občas používá sám procesor. Ale při programování jsem používal i vysokorychlostní paměť pro rozběhnutí časovače. Vůbec to nepřekáželo procesoru v práci a přistup do této paměti je povolen.

Funkci READ_EEPROM používám pro čtení dat z EEPROM paměti

```
char read_eeprom( int badd )
{
        EEADR = badd;           // Registr adresy
        EECON1bits.CFGS = 0;    // Vyber EEPROM nebo FLASH
        EECON1bits.EEPGD = 0;   // Vyber EEPROM pameti
        EECON1bits.RD = 1;      // Iniciace cteni
        Nop();                  // Pro velke rychlosti
        Nop();                  // Pro velke rychlosti
        return ( EEDATA );      // Vraceni hodnoty ulozene v pameti
}
```

Na vstup funkce musíme přidat adresu, ze které chceme číst data. Výstupem funkce budou data, které se nacházejí na této adrese v datovém typu char. Pokud mikroprocesor pracuje ve velké rychlosti, je potřeba ponechat operace NOP k tomu, aby se data stihly zapsat do bufferu.

Funkce BUSY_EEPROM je pro ověření toho, zda jsou ukončené operace zápisu do EEPROM paměti.

```
void busy_eeprom ( void )
{
        while(EECON1bits.WR);   // Pockat ukonceni zapisu
}
```

Funkce MOVE je základní funkce pro řízení motoru. Hlavni princip funkce spočívá v tom, že procesor čte data, která byla předaná přes RS-232 a uložená v paměti EEPROM a uloží je do proměnných. Pak mikroprocesor nastavuje výstupní porty pro řízení směru, režimu a jiných věcí v L297. Pokud motor najede na nějaký koncový spínač, tak změní směr otáčení a odjede z něj, tak, aby spínač nebyl sepnutý.

Níže je uvedena část kódu pro řízení jednoho motoru:

```
void move(int motor){

    char coxa_direction;            //Smer
    char coxa_half;                 //Pulkrokovy rezim
    char coxa_unreset;              //Reset
    char coxa_enable;               //Povolovaci vstup
```

```
coxa_direction = read_eeprom(0x081);     //Precist smer
coxa_half =  read_eeprom(0x082);         //Precist pulkrokovy rezim
coxa_unreset = read_eeprom(0x083);       //Precist reset
coxa_enable = read_eeprom(0x084);        //Precist povolovaci vstup
```

```
if (motor == 1) //COXA
{
    PORTAbits.RA0 = coxa_direction;      //Smer
    PORTAbits.RA1 = coxa_half;           //Pulkrokovy rezim
    PORTAbits.RA2 = coxa_unreset;        //Reset
    PORTAbits.RA3 = coxa_enable;         //Povolovaci vstup
    if (steps_coxa == 0x00)              //Overujeme pocet kroku
    {
        write_eeprom(0x084,0x00);        //Zastavi motor
        write_eeprom(0x086,steps_coxa); //Nuluje pocet kroku
    }
       if (PORTAbits.RA4 == 0){          //Sepnuti IC1
        femur_speed = read_eeprom(0x090);
        //Snizeni rychlosti
        PORTEbits.RE0 = 1;   //Zacit posilat data
        //       Stack Registor (bit 0)
        spi(0b00000000);
        //        Potentiometr #1 (bits 1-8)
        spi(0x80);
        //        Potentiometr #0 (bits 9-16)
        spi(femur_speed);
        PORTEbits.RE0 = 0;       // Ukoncit posilat data
        PORTAbits.RA0 = 1;       // Zmena smeru
        Delay10KTCYx(10);        // Pockat 100Kcyklu
        write_eeprom (0x084,0x00);

    }
    }
    if (PORTAbits.RA5 == 0){             // Sepnuti IC2
        femur_speed = read_eeprom(0x080);
        //Snizeni rychlosti
        PORTEbits.RE0 = 1;   //Zacit posilat data
        //       Stack Registor (bit 0)
        spi(0b00000000);
        //        Potentiometr #1 (bits 1-8)
        spi(0x80);
        //        Potentiometr #0 (bits 9-16)
        spi(femur_speed);
        PORTEbits.RE0 = 0;       // Ukoncit posilat data
        PORTAbits.RA0 = 0;       // Zmena smeru
        Delay10KTCYx(10);        // Pockat 100Kcyklu
        write_eeprom (0x084,0x00);
    }
}
```

ZÁVĚR

Jako zadání pro diplomovou práci jsem dostal za úkol navrhnout řídící systém a vytvořit prototyp desky plošných spojů pro řízení angulárního robota. Tento systém by měl být řízený z nějakého vnějšího přístroje, podle zadaní.

Realizoval jsem zadaný řídící systém a vyzkoušel jsem ho. Vytvořil jsem program pro mikroprocesor a odzkoušel jsem jeho funkčnost. Po zkouškách jsem zjistil, že tento program a celý systém jsou plně funkční.

Pro vytvoření programu jsem použil programovací jazyk C. Není to nejrychlejší řešení, ale pro řídicí systém rychlosti stačí. Tento program se dá zrychlit použitím Asembleru, ale pro tyto účely je to zbytečně náročné.

Realizoval jsem komunikaci s vnějším zařízením přes sériovou linku na základě rozhraní RS-232. Navrhnul jsem vlastní rozhraní pro zadání jednotlivých příkazů motorům. Způsob komunikace a rozhraní, které jsem navrhnul, může být použito pro práci se všemi přístroji, které mohou přenášet symboly přes sériovou linku. Pro ukázku funkčnosti komunikace jsem použil program MATLAB, přes který se dá robot ovládat.

Z důvodu, že už jsou krokové motory a mechanické části robota poměrně staré, musel jsem občas některé z nich opravovat. Nyní má robot dvě nové plastové spojky a opravené vodiče k motorům a čidlům.

Řídící systém dovoluje řídit všechny motory najednou s tolerancí cca 10 mm. Tato chyba je způsobena hlavně mechanickou vůlí angulárního robota.

POUŽITÁ LITERATURA

[1] ЕМЕЛЬЯНОВ, А.В – ШИЛИН, А.Н.: *Шаговые двигатели.* Темплан, 2005

[2] anon.: *L297 Data Sheet – Stepper Motor Controllers.* ST Microelectronics, 2001

[3] anon.: *PIC18F2420/2520/4420/4520 Data Sheet - 28/40/44 – PinEnhanced Flash Microcontrollers with 10-Bit A/D and nano Watt Technology.* Microchip Technology, Inc., 2004

[4] anon.: *L298 Data Sheet - Dual Full-Bridge Driver*. ST Microelectronics, 2000

[5] NOVOTNÝ, F.: *Přednášky z předmětu „Základy robotiky“*. TUL, 2011

[6] anon.: *Dual Digital Potentiometer Chip DS1267.* DALLAS Semiconductor

[7] anon.: *MAX220-MAX249 +5V-Powered, Multichannel RS232 Drivers/Receivers.* Maxim Integrated Products, Inc., 2010

[8] PETRUS, L.: *Laboratorní model řízení třídícího automatu.* TUL, 2003

[9] anon.: *L6210 Data Sheet – Dual Schottky Diode Bridge.* ST Microelectronics, 1994

[10] anon.: *HlpC18Lib*. Microchip Technology, Inc., 2008

SEZNAM PŘÍLOH

A. Zdrojový kód programu pro mikroprocesor PIC18F4420 (pouze CD)

B. Fotografie robota (pouze CD)

C. Video pohybu robota (pouze CD)