Technická univerzita v Liberci
Fakulta strojní

DIPLOMOVÁ PRÁCE

Laboratorní model řízení třídícího automatu

Liberec 2003 Lukáš Petrus

Technická univerzita v Liberci

Fakulta strojní

Studijní obor : 23 – 40 – 8 Automatizované systémy řízení ve strojírenství
Zaměření : Automatizace inženýrských prací

Katedra aplikované kybernetiky

Laboratorní model řízení třídícího automatu

A Prototype of Controlling an Automatic Classifier

Lukáš Petrus

Vedoucí diplomové práce : Ing. Slavomír Němeček
Konzultant diplomové práce : Ing. Jan Klobouček

Anotace:

Cílem diplomové práce je navrhnout a realizovat systém pro řízení laboratorního modelu třídiče výrobků na kovové a nekovové a jejich odebírání angulárním manipulátorem. Třídič obsahuje tři lineární pneumatické motory. Manipulátor je poháněn čtyřmi krokovými motory. Manipulátoru i třídiče lze použít nezávisle. Ovládání řídícího systému je možné po sériové lince z PC.

Annotation:

The graduate work aims to design and materialize a control system for the laboratory model of a metal/non-metal product-sorting machine, and subsequent product removal done by an angular manipulator. The sorting machine has three linear pneumatic motors. The manipulator is powered by four step motors. Both the manipulator and the sorting machine may work separately. The whole process may be controlled from a PC via a serial line.

Prohlášení

„Místopřísežně prohlašuji, že jsem diplomovou práci vypracoval samostatně s použitím uvedené literatury."

V Liberci 25. 5. 2003

.....

Lukáš Petrus

Poděkování

Na tomto místě bych chtěl poděkovat Ing. Janu Kloboučkovi za realizaci hardwaru třídiče a za podporu při konzultacích. Také děkuji Ing. Slavomíru Němečkovi za návrh tématu pro mou diplomovou práci. Dále bych chtěl poděkovat Ditě Filipové za návrhy ikon a kurzorů pro ovládací program. Děkuji také rodičům Heleně Petrusové a Kamilu Petrusovi za podporu při mých studiích.

Obsah

1 Úvod

V rámci modernizace laboratoří na Katedře aplikované kybernetiky jsem dostal možnost vytvořit řídící systém k ovládání angulárního manipulátoru. Jádrem řídícího systému měl být jednočipový mikropočítač řady 8051. Protože jsem se již delší dobu těmito mikroprocesory zabýval, rozhodl jsem se své dosavadní zkušenosti zhodnotit a prohloubit v diplomové práci. Nakonec bylo zadání mé diplomové práce ještě rozšířeno o ovládání třídiče materiálu, který však nevyžaduje tak složité řízení jako manipulátor.

Pohon manipulátoru je tvořen čtyřmi bipolárními krokovými motory. Před zadáním práce jsem měl zkušenost s motory pouze unipolárními, když jsem se však blíže seznámil s vlastnostmi bipolárních motorů, zjistil jsem, že jejich řízení je v podstatě totožné.

Původně byl školní manipulátor řízen programovatelným logickým automatem (PLC). Program pro PLC byl řešen jako PTP (Point To Point - narážkové řízení). Polohy, do nichž se manipulátor mohl přesunout byly pevně určeny snímači. V reálném čase se pohyboval pouze jeden krokový motor. Nebylo možné ovládat více motorů najednou. Koncepce s těmito nedostatky je pro plné využití mechanismu robota zcela nevyhovující. Proto jsem navrhl jiné řešení, které je však pro svou hardwarovou náročnost na daném PLC nerealizovatelné.

Mnou navržený řídící systém je schopen v reálném čase ovládat všechny motory najednou a manipulátor je schopen zastavit v jakékoli poloze svého pracovního prostoru. Tolerance této polohy je závislá na velikosti úhlového přírůstku na jeden krok krokového motoru a na mechanických tolerancích celého mechanismu.

Koncepci řízení manipulátoru jsem navrhl tak, že každému motoru je v paměti (RAM) přiřazena struktura. V této struktuře jsou uloženy (mimo jiné) proměnné určující polohu manipulátoru. Těmi jsou *pozice* (sem jsem uložil počet kroků motoru od nulové pozice) a *stoppozice* (sem jsem uložil počet kroků od nulové pozice, kam má manipulátor dojet). Posloupnost proměnných *stoppozice* je po čtveřicích uložena v paměti EEPROM. Program si vždy převezme čtveřici aktuálních *stoppozic*, přiřadí je motorům a posune *index* čtveřice. Proměnná *index* slouží k orientaci v paměti. Pokud je hodnota *stoppozice* příslušející motoru 0 větší než 65520 (tj. 0xFFF0h), pak se nejedná o zadání souřadnic, ale jde o řídící příkaz. Parametry konkrétních řídících příkazů jsou v následujících *stoppozicích* ve vybrané čtveřici.

Manipulátor vykonává přesně definovaný pohyb, který je tvořen pohybovými vektory. Tyto vektory vznikají z rozdílu příslušných hodnot *stoppozic*. Data v EEPROM můžeme editovat tak, že pomocí manuálního ovládání přesuneme manipulátor na požadovanou pozici a ovládacím programem (je spuštěn na PC) zaznamenáme tuto polohu do EEPROM. Jiné možné

řešení je přímo zadat hodnotu polohy v obslužném programu a zapsat ji do EEPROM. Tak postupně vytvoříme trajektorii, po které se bude manipulátor pohybovat.

Jelikož při zadávání pozic nevystačíme s osmibitovou proměnou (tj. 0 až 255), musíme použít šestnáctibitové slovo (word) s rozsahem 0 až 65535. Při zařazení funkce, která zajišťuje současný dojezd všech motorů, pracujeme dokonce s dělením šestnáctibitové proměnné a navíc se celý výpočet provede v čtyřnásobném cyklu. Mikroprocesor 8051 je plně osmibitový a tak při práci s šestnáctibitovými proměnnými musí tyto proměnné rozkládat a skládat na dvě dvojice osmibitových proměnných. Pro čtyři motory a 2ms mezi kroky výpočetní výkon mikroprocesoru ještě dostačuje. Pokud bychom chtěli řídit větší počet motorů nebo by nám nestačila rychlost pohybu manipulátoru, musíme použít výkonnější mikroprocesor. Nejlepším řešením by bylo použít mikroprocesor šestnáctibitový.

Třídící automat je řízen PTP řízením. Pro ovládání třídícího automatu jsem napsal funkci *tridic*. Tato funkce je vyvolána buď řídícím příkazem, který přijde po sériové lince, nebo řídícím příkazem, který byl přečten z EEPROM. Podrobně je funkce *tridic* popsána v kapitole 8.5.

K řídící desce je pomocí sériové linky připojen ovládací počítač PC. Sériová komunikace probíhá rychlostí 57600 kb/s. Po této lince jsou posílány pakety s adresou zařízení, zadaným příkazem, daty a kontrolním součtem CRC. Tak je zajištěna vysoká bezpečnost přenosu, která je u podobných zařízení nepostradatelná.

2 Mikroprocesor 8051

Jádro mikroprocesoru 8051 je relativně staré, avšak díky rozšiřování čipu o další periferie je dodnes velice rozšířené. Firma Intel ho uvedla na trh v roce 1983. Mikroprocesor 8051 je osmibitový s tzv. harwardskou architekturou (tzn. je oddělená paměť programu od paměti dat). Je schopen samostatné činnosti již po připojení krystalu a napájecího napětí 5 V. Čip obsahuje 4 kB programové paměti ROM, nebo EPROM, 128 B paměti dat RAM, sériový kanál UART, dva 16-ti bitové čítače/časovače, čtyři vstupně/výstupní brány a přerušovací systém, který zpracovává 5 zdrojů přerušení (UART, dva čítače/časovače a dvě externí přerušení). Pomocí řídících signálů (/PSEN pro paměť programu, /WR a /RD pro paměť dat) a brány P0 a P2 můžeme přistupovat do vnějších pamětí EPROM a RAM o velikosti až 64 kB.

U mikroprocesoru 8051 se instrukce vykonávají v jednom až dvou strojových cyklech, které se dále dělí na 6 stavů a každý stav má dvě fáze. Během první fáze se vykonávají aritmetické a logické operace a během druhé fáze dochází k přesunům dat mezi registry. Fáze trvá jednu periodu základních hodin. Budeme-li mít krystal na 12 MHz, pak jedna fáze trvá 1/12 MHz a jeden strojový cyklus trvá 1 us. Instrukční soubor obsahuje 58% instrukcí, které se vykonají během 1 strojového cyklu a 40% instrukcí během 2 strojových cyklů. Dvě instrukce MUL a DIV potřebují 4 cykly.

Mikroprocesory „51“ se dnes vyrábějí v různých obměnách. Podle výrobce a typu můžeme na čipu nalézt např. paměti ROM (až 32 kB), EPROM, FLASH, RAM (až 512 B), EEPROM, sériové linky UART, I2C, SPI, CAN, časovače, analogové komparátory, AD převodník, PWM, dekodér MP3, obvod pro rádiový přenos a další.

Pro vývoj jsem použil mikroprocesor AT89S8252, který umožňuje ladění programu přímo v aplikaci pomocí sériové linky SPI. Ve výsledné aplikaci je možno mikroprocesor v patici zaměnit za typ AT89C52, jenž plně zahrnuje instrukční soubor řady „51“. Oproti typu 80C51 má však rozšířen počet speciálních funkčních registrů, které odpovídají přidaným periferiím na čipu.

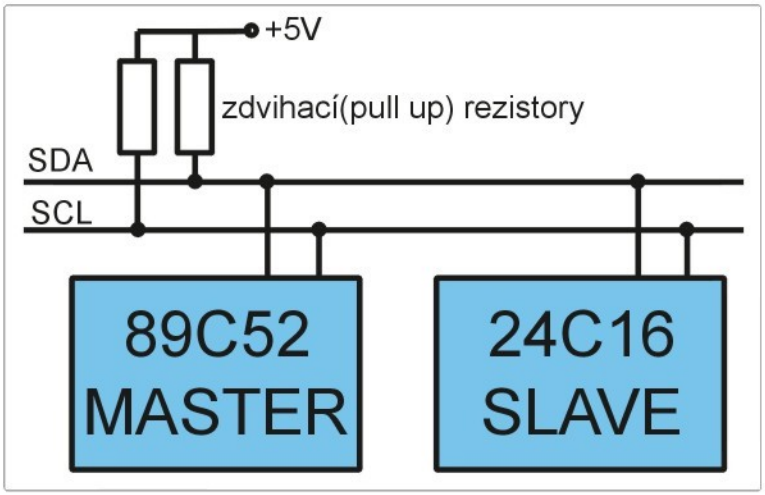
3 Sběrnice I^2C

Sběrnice I^2C *(Inter-IC-Bus)* je dvouvodičové datové propojení mezi jedním nebo několika procesory *(master)* a speciálními periferními součástkami *(slave)*. Všechny součástky jsou připojeny na tutéž sběrnici a jsou cíleně vybírány pod svými adresami. Adresy i data se přenášejí týmiž linkami. Sběrnice umožňuje velmi jednoduché propojení mezi několika integrovanými obvody a bezproblémové dodatečné rozšiřování. Mohou být připojeny všechny integrované obvody, které zvládají speciální protokol sběrnice. Mimo integrovaných obvodů RAM, EEPROM, obvodů pro rozšíření portů, A/D a D/A převodníků a obvodů hodinových signálů existuje ještě řada speciálních integrovaných obvodů, jako například budiče displejů.

Použitý procesor nedisponuje interním obvodovým řešením pro řízení sběrnice I^2C, proto jsem musel I^2C řešit programově. Linka SCL je připojena na P3.4 procesoru a linka SDA je k dispozici na P3.5 procesoru. Pomocí této sériové linky jsem k mikroprocesoru připojil EEPROM paměť 24C16.

3.1 Protokol sběrnice I^2C

Sběrnice I^2C používá sériovou datovou linku SDA a linku hodinového (taktovacího) signálu SCL. Data a adresy se přenášejí podobně jako v posuvných registrech společně s hodinovými impulsy. Obě linky lze používat jako obousměrné. Jsou vybavené zvyšovacím *(pull up)* odporem a mohou být každým účastníkem sběrnice staženy na nízkou úroveň výstupem s otevřeným kolektorem. Vstupy neaktivních účastníků sběrnice mají vysokou impedanci, neustále však vyhodnocují signály na sběrnici. Je-li použit jen jeden master, vydává hodinový signál jen on. Data však může vysílat jak *master*, tak i podřízené zařízení (*slave*).



Obr.1 Princip propojení sběrnice, autor Lukáš Petrus (dále jen LP) podle [6].

Protokol I^2C rozeznává řadu přesně definovaných situací, které každému účastníkovi umožňují rozpoznat začátek a konec přenosu a také své možné adresování:

Klidový stav: SDA i SCL jsou na vysoké úrovni (HIGH) a tím neaktivní.

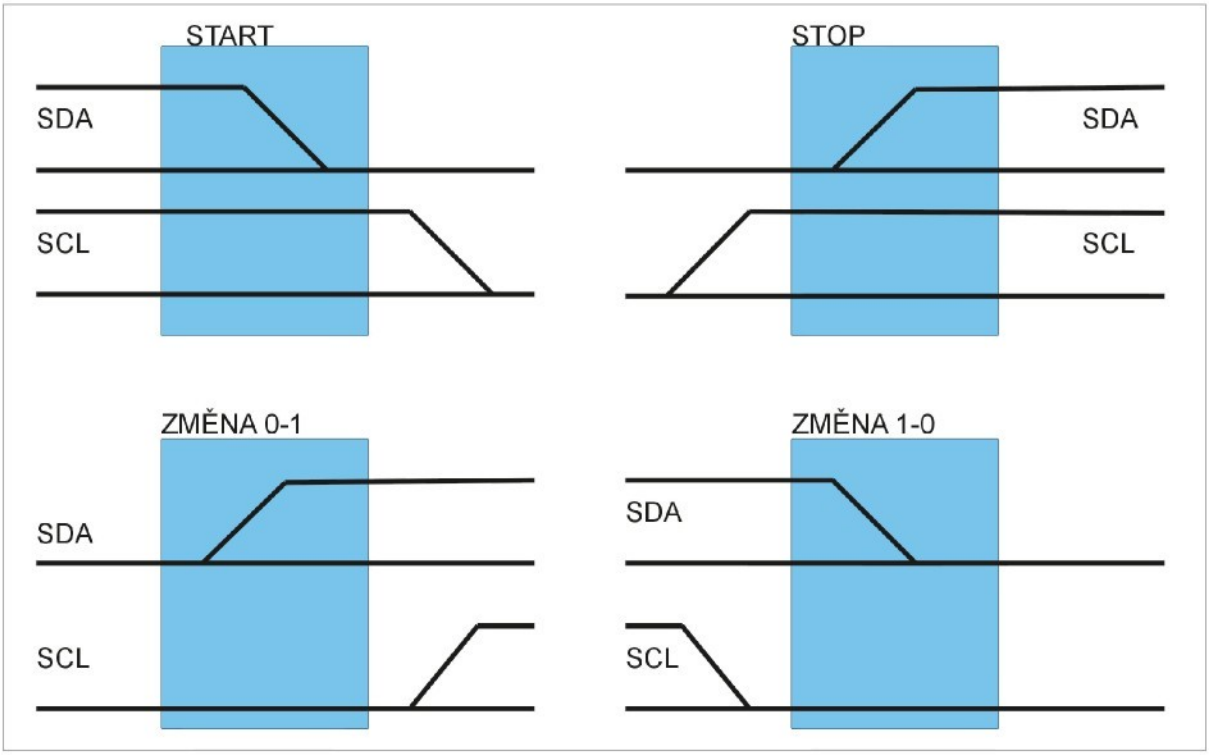
Podmínka startu: SDA je masterem stažena na nízkou úroveň, zatímco SCL zůstává na vysoké úrovni.

Podmínka stopu: SDA přejde z nízké úrovně na úroveň vysokou, SCL zůstává na vysoké úrovni.

Přenos dat: Příslušný vysílač přivede na datovou linku SDA osm datových bitů, které jsou hodinovými impulsy na lince SCL vysílanými masterem posouvány dále. Přenos začíná bitem s nejvyšší váhou.

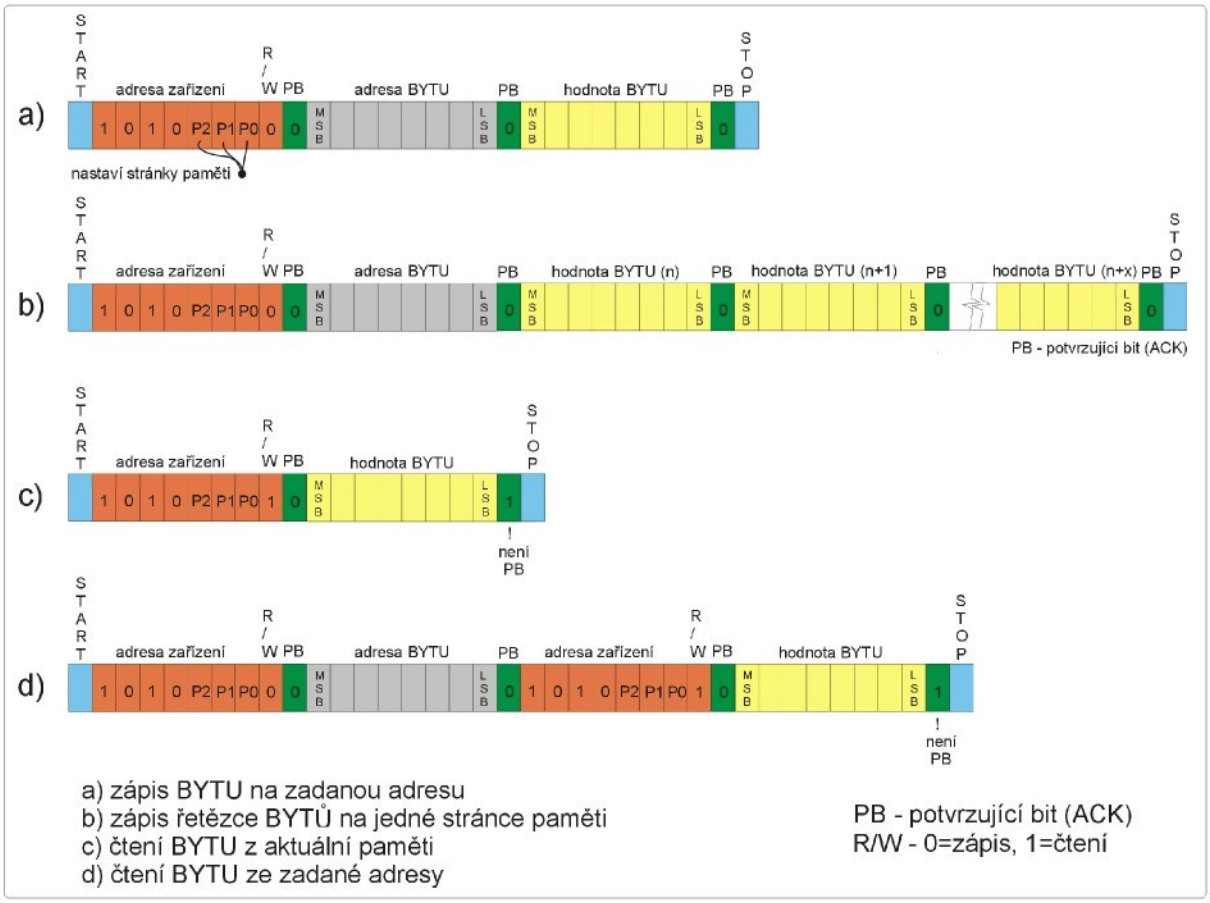
Potvrzení *(acknowledge):*

Příslušný přijímač potvrzuje příjem bytu nízkou úrovní na SDA, dokud *master* nevyšle devátý hodinový impuls na SCL. Potvrzení současně znamená, že se může přenášet další datový byt. Požadované ukončení přenosu se musí ohlásit neexistencí potvrzení. Vlastního ukončení přenosu se dosahuje podmínkou stopu.



Obr.2 Podmínky START, STOP a změna hodnoty bitu na I^2C, LP podle [6].

Přenos a potvrzování adres se provádí přesně stejně jako přenos dat. V nejjednodušším případě přenosu dat od zařízení mastera k podřízenému zařízení (*slave*) probíhají následující děje. Master vyrobí podmínku startu a pak v bitech 7 až 1 přenese adresu zařízení (součástky) a v bitu 0 požadovaný směr přenosu dat, totiž 0 pro zápis. Podřízené zařízení (*slave*) adresu potvrdí. Pak master vyšle datový byt, který rovněž bude potvrzen. Master nyní může spojení přerušit vysláním podmínky stopu nebo může témuž zařízení *slave* posílat další byty.



Obr.3 Přenos po I^2C, LP podle [7].

Mají-li se číst data od zařízení *slave*, musí se adresa přenést s jedničkovým bitem směru přenosu R/W. Master vždy vydá osm hodinových impulsů a dostane osm datových bitů. Potvrdí-li příjem vysláním devátého hodinového impulsu, může přijímat další bajty. Přenos se ukončí tím, že master nevydá podmínku potvrzení, nýbrž ukončení přenosu. Maximální hodinová frekvence pro sběrnici I^2C je 100 kHz.

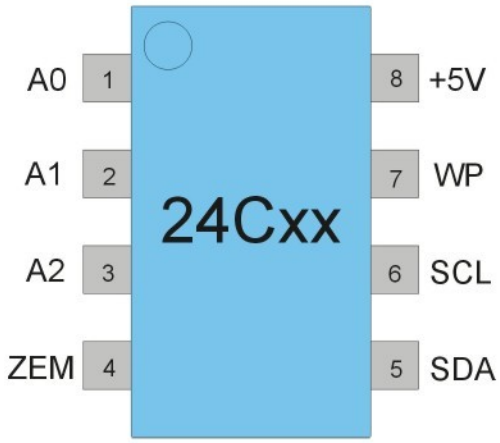
Každý prvek I^2C má pevně stanovenou adresu, jejíž jedna část je charakteristická pro typ prvku a druhá část je proměnná v závislosti na vyvedených linkách adres. To znamená, že například v případě tří vyvedených linek může být na jednu sběrnici připojeno až osm prvků téhož typu.

3.2 Sériové paměti EEPROM I^2C

Sériové paměti EEPROM se v osmivývodovém pouzdru vyrábějí s kapacitou 16 B až 64 kB. Na paměti se lze obracet různě v závislosti na velikosti interního registru adres. Malé paměti EEPROM do 256 B používají 1-bytový čítač adres, větší, mezi 4 kB a 64 kB potřebují 2-bytový čítač adres. Všechny paměti EEPROM používají adresu A0h jako základní (bázovou) adresu.

Tato adresa platí, jsou-li všechny vyvedené linky A0 až A2 v logické nule (připojeny na zemění).

Obr.4 ukazuje rozložení vývodů na pouzdře integrovaného obvodu. Vidíme linky I^2C SCL a SDA a adresní linky A0 až A2. V závislosti na typu bývá k dispozici vstup WP *(write protect,* tj. ochrana proti zápisu), pomocí něhož lze hardwarově potlačit zápisové přístupy. Pro připojení na systém s napájecím napětím 3,3 V by se měly používat nízkonapěťové typy 24LCxx. Pro řídící desku vyrobenou podle mého návrhu jsem použil typ 24C16 při napájecím napětí 5 V. Linky SCL a SDA pracují s úrovněmi 5 V.



Obr.4 Rozmístění vývodů pamětí 24Cxx v pouzdrech PDIP, SOIC, TSSOP, LP podle [7].

Kapacity pamětí 24Cxx :	- 24LC00:	16	B
	- 24LC01:	128	B
	- 24LC02:	256	B
	- 24LC04:	512	B
	- 24LC08:	1	kB
	- 24LC16:	2	kB
	- 24LC32:	4	kB
	- 24LC64:	8	kB
	- 24LC256:	32	kB

Menší paměti EEPROM do 256 B (tj. 2 kilobity) používají interní registr adres o délce 8 bitů. Ten obsáhne maximálně 256 adres. Jednoduchý protokol proto připouští jen paměti s kapacitou do 256 B. Je možno společně použít až 8 těchto součástek, protože lze používat adresy mezi A0h až AEh. Maximální celková kapacita tedy činí 2 kB. Větší EEPROM 24LC04 až 24LC16 obsazují každá několik adres sběrnice I^2C. Obvod 24LC16 se tudíž chová přesně jako 8 obvodů

24LC02 umístěných v tomto adresovém prostoru. Proto nejsou adresní vstupy A0 až A2 obsazeny. Do těchto typů EEPROM se musí číst a zapisovat v blocích po 256 bajtů, přičemž pokaždé musí proběhnout nové adresování. Větší EEPROM s kapacitou nad 2 kB používají pozměněný přenosový protokol s šestnáctibitovým registrem adres. Po sběrnicové adrese se musí vyslat ještě další dva bajty pro zadání interní adresy bajtu. Protokol tedy již není kompatibilní s menšími EEPROM.

4 Krokové motory

Velkou předností krokových motorů je, že pracují bez nákladných snímačů otáček nebo polohy, jsou jednodušší a tím i provozně spolehlivější a levnější než servomechanismy. Řízení se typicky provádí v přímé větvi bez zpětné vazby. Jedinou podmínkou spolehlivé funkce je jejich správné dimenzování ve všech provozních režimech. Mají vysokou životnost a nevyžadující údržbu.

Obr.5 Bipolární krokový motor, LP

V rámci objektivity hned na začátku zmíníme i nevýhody pohonů s krokovými motory. Nejzávažnější je pravděpodobně trvalý odběr proudu i když se motor netočí. V důsledku toho se motor ohřívá. Nepříliš výhodný je i poměr výkonu (kroutícího momentu) vůči hmotnosti motoru.

Motory mají několik možností způsobu vinutí. Jejich charakteristiky se potom liší podle připojení k výkonovému stupni: unipolární, bipolární-sériové a bipolární-paralelní.

4.1 Konstrukce krokového motoru

Na obr.6 a 7 je krokový motor se 100 kroky na otáčku (tj. 3.6 stupně na krok). Stator krokového motoru tvoří sada cívek. Pólové nástavce statoru jsou vroubkovány se stejnou roztečí, jaká je rozteč mezi magnety na rotoru. Jsou tak jednou z částí zvyšujících přesnost motoru při stejném počtu cívek. Rotor je tvořen hřídelí usazenou na kuličkových ložiskách a prstencem permanentních magnetů.

Obr.6 Stator motoru, Internet

Obr.7 Rotor motoru, Internet

4.2 Funkce krokového motoru

Základní princip krokového motoru je jednoduchý. Proud procházející cívkou statoru vytváří magnetické pole, které přitahuje opačný pól magnetu rotoru. Vhodným zapojováním cívek vytvoříme rotujícího magnetické pole, které otáčí rotorem.

Podle požadovaného kroutícího momentu, přesnosti nastavení polohy a přípustného odběru volíme některou z variant řízení (popsány níže).

Kvůli přechodovým magnetickým jevům je omezena rychlost otáčení motoru a to na několik stovek kroků za sekundu (v závislosti na typu motoru a zatížení). Při překročení této maximální rychlosti nebo při příliš velké zátěži začínají motory „ztrácet kroky“.

4.3 Základní definice

Úhel kroku

je jmenovitý úhel, o který se otočí hřídel motorku na jeden řídicí impuls.

Řídicí kmitočet

je kmitočet, kterým je řízen motor. Souhlasí s kmitočtem kroku, jestliže motor běží bez chyby kroku.

Vlastní přídržný moment

Maximální moment, kterým může být staticky zatížena hřídel nevybuzeného motoru, aniž by se začala plynule otáčet.

Přídržný moment

Maximální moment, kterým může být staticky zatížena hřídel vybuzeného motoru, aniž by se začala plynule otáčet.

Zatěžovací moment

Moment charakteru pasivního tření, kterým je zatížena hřídel motoru.

Rozběhový moment

Zatěžovací moment, se kterým se motorek může rozběhnout start-stop bez chyby kroku, bez přídavné vnější setrvačné hmoty při definovaném řídicím kmitočtu.

Rozběhový moment setrvačnosti

Vnější moment setrvačnosti, se kterým se motorek může rozběhnout start-stop bez chyby kroku, bez zatížení zatěžovacím momentem při definovaném řídicím kmitočtu.

Chod naprázdno

Provozní stav, ve kterém motorek není zatížen ani vnějším zatěžovacím momentem ani vnější setrvačnou hmotou.

Maximální rozběhový kmitočet

Největší řídicí kmitočet, při kterém se motorek může rozběhnout start-stop při chodu naprázdno bez ztráty kroku.

Rozběhový kmitočet

Největší řídicí kmitočet, při kterém se může motorek rozběhnout bez ztráty kroku s určitým zatížením sestávajícím z rozběhového momentu a z vnějšího momentu setrvačnosti.

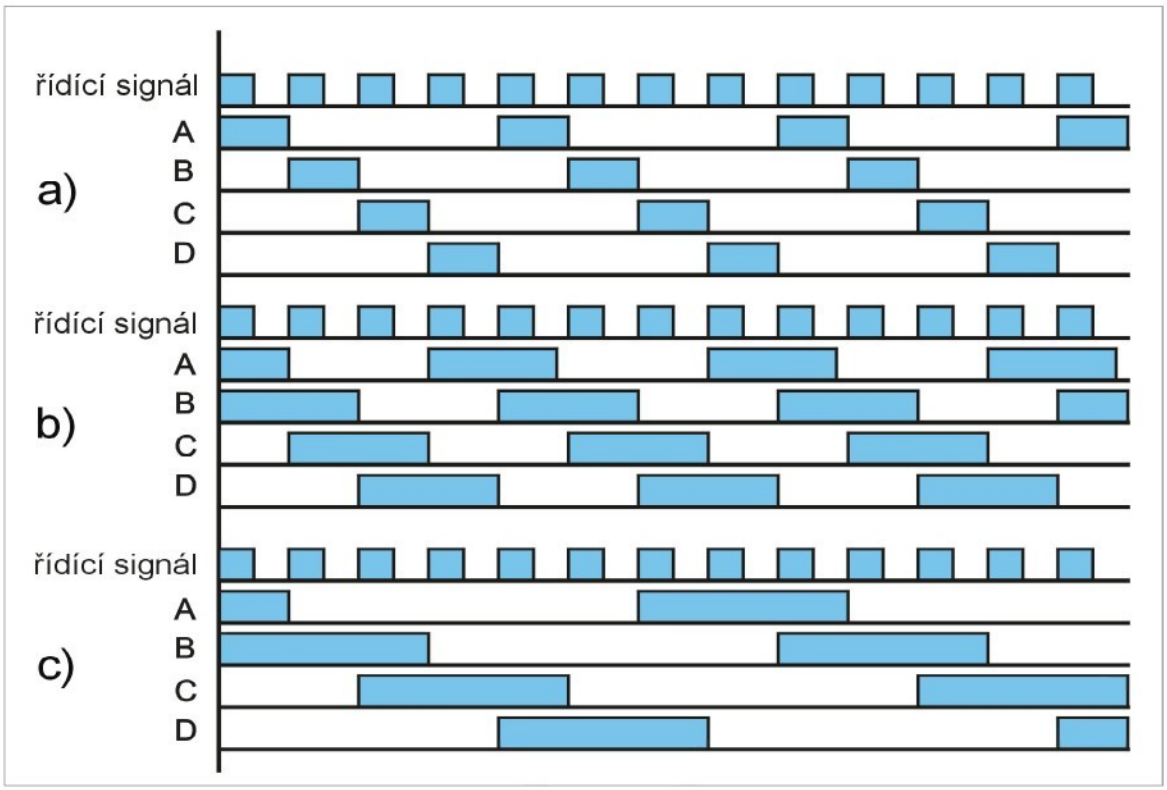
Maximální provozní kmitočet

Nejvyšší řídicí kmitočet, při kterém může být motorek provozován při plynulém zvýšení nebo snížení řídicího kmitočtu při chodu naprázdno.

Tyto definice jsou dostupné na [8].

4.4 Metody řízení krokových motorů

Časový průběh buzení fází čtyřfázového motorku v závislosti na řídicím signálu je na následujícím obrázku.



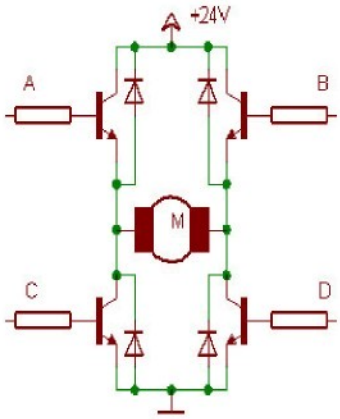
Obr.8 Průběhy proudu při různých způsobech ovládání a) čtyřtaktní po jedné fázi, b) čtyřtaktní po dvou fázích, c) osmitaktní, LP podle [9].

Na obr.8a je vidět, že v každé periodě řídicího signálu je magnetické pole buzeno pouze jednou ze čtyř fází vinutí. Druhý způsob řízení krokového motorku je zobrazen na obr.8b. Při tomto způsobu řízení se budí současně vždy dvě blízké fáze: (1, 0, 0,1), (1,1, 0, 0), (0,1,1, 0), (0, 0, 1, 1). Tímto způsobem řízení se realizuje stejná velikost kroku v předchozím příkladě, neboť se pouze změní klidová magnetická poloha, která je mezi statorovými póly. Nevýhodou je dvojnásobná proudová a tím i výkonová náročnost, která má za následek větší ohřátí motoru. Výhodou je zvětšení statického momentu motoru, který je v porovnání s předchozím způsobem buzení asi dvakrát větší. Oběma dosud uvedeným způsobům řízení říkáme „čtyřtaktní", protože se v nich střídají dokola čtyři možné kombinace.

Třetí způsob řízení vznikne sloučením prvního a druhého způsobu řízení tak, že vložíme mezi kombinace jednofázové kombinace dvoufázové a vznikne tak „osmitaktní" způsob řízení. Časový průběh osmitaktního řízení vidíme na obr.8c. Výhoda tohoto způsobu řízení je v tom,

že zmenšíme základní velikost kroku na polovinu a získáme tak větší úhlové rozlišení. Protože se střídá při řízení zapojení jedné a dvou fází motoru, nutně se nám i v tomto rytmu mění velikost provozního momentu motoru, takže celkový výsledný moment bude menší než v případě čtyřtaktního řízení po dvou fázích.

Při unipolárním řízení prochází proud v jednom okamžiku právě jednou cívkou. Motor s tímto buzením má nejmenší odběr, ale také poskytuje nejmenší kroutící moment. Výhodou tohoto řešení je jednoduché zapojení řídící elektroniky - v podstatě stačí jeden tranzistor na každou cívku. Pro menší motory lze s výhodou použít integrovaný obvod ULN2803. V jednom pouzdře je dostatek budičů pro řízení dvou motorů.



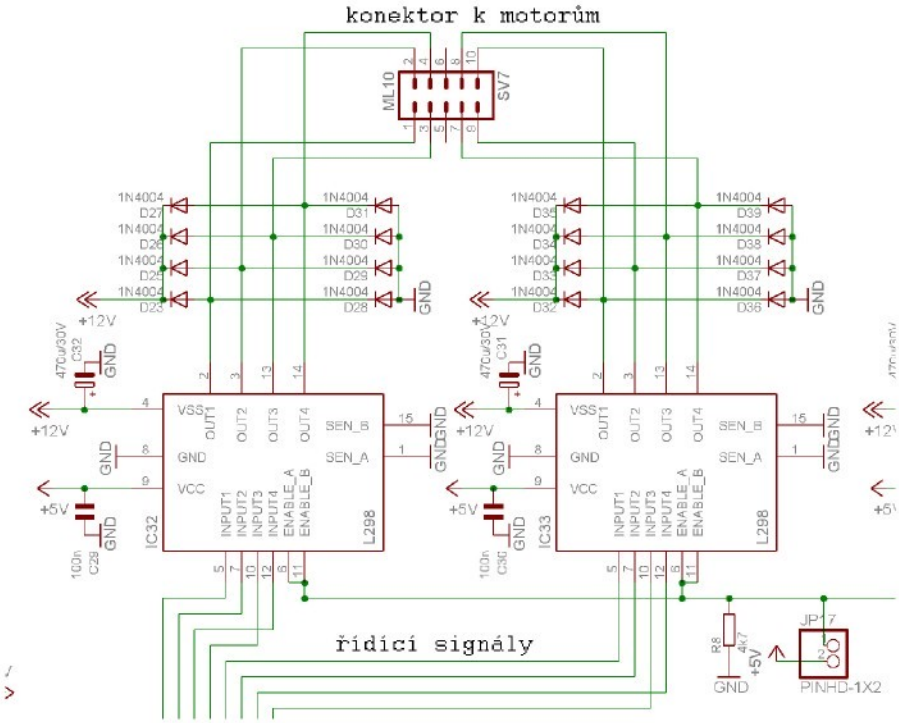
Obr.9 Zapojení H-můstku, LP.

Při bipolárním řízení zapojujeme cívky krokových motorů do tzv. H-můstků (viz obr.9). Pokud řídící signály A a D budou v logické jedničce a B a C budou v logické nule, proud poteče do cívky jedním směrem. Je-li A a D log. nula a B a C log. jednička, proud poteče do cívky směrem opačným.

Proud prochází vždy dvěma protilehlými cívkami tak, že mají navzájem opačně orientované magnetické pole. Pro řízení jednoho motoru jsou tedy zapotřebí dva H-můstky, každý pro jednu cívku. To ve výsledku znamená složitější zapojení než při použití motorů unipolárních. Vhodným integrovaným obvodem pro bipolární řízení menších motorů je dvojice H-můstků L298. Na výstup těchto můstků musíme paralelně připojit ochranné diody, ty nejsou v L298 obsaženy.

V praktické části diplomové práce jsem řídil bipolární krokové motory, které byly hnacími jednotkami pro manipulátor. Technické vlastnosti těchto motorů jsem bohužel neměl k dispozici, proto jsem je musel přibližně stanovit z provedených pokusů.

Tyto motory jsou napájeny napětím 24 V. Mají 100 kroků na otáčku, tzn. úhel 3,6° na jeden krok. Maximální řídící kmitočet jsem stanovil na 500 Hz, což jsou 2 ms mezi kroky. Při vyšším kmitočtu motor zajišťující rotaci ztrácel kroky (viz výše). Taktéž jsem musel provést řadu experimentů kvůli vyřešení bezpečného rozjezdu a dojezdu. Velikost přídržného a vlastního přídržného momentu v tomto případě nebyla důležitá. Šroubové mechanismy použité na rameno, zvedák a kleště jsou samosvorné a správnost polohy rotace je ohrožena pouze dynamickými silami, ale protože robot před vykonáním pohybu vždy provede rozjezdovou a na konci pohybu dojezdovou sekvenci, jsou dynamické síly minimalizovány.



Obr.10, Zapojení H-můstků L298 pro dva krokové motory; LP

5 Návrh obvodového řešení – popis schématu

Při návrhu řídící desky bylo třeba uvažovat o začlenění systému do modulární koncepce školní laboratoře. Proto jsem vytvořil návrh poněkud složitější, který je však velice univerzální.

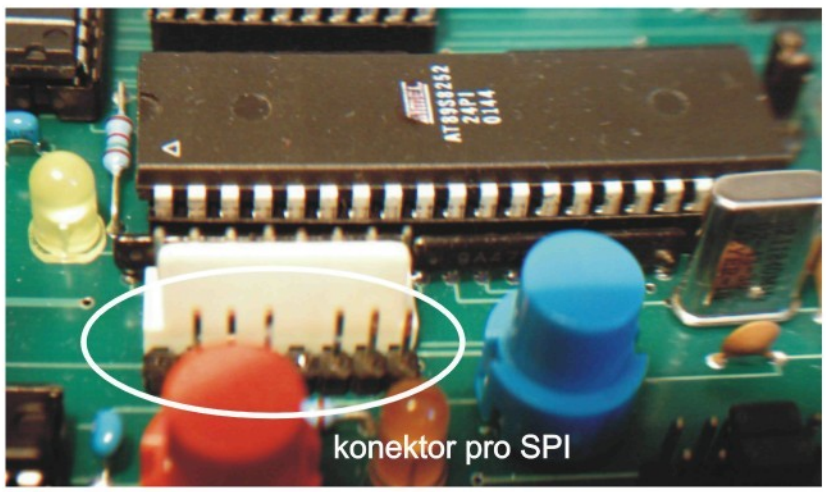
Všechny obvody jsou řešeny v rámci jedné desky plošných spojů, která obsahuje jak logickou, tak výkonovou část systému. (Pro kompletní schéma zapojení viz.přílohu č. 8).

5.1 Mikroprocesor

Řídící jádro tvoří jednočipový mikropočítač AT89C52. Přesné taktování vnitřního oscilátoru mikroprocesoru je zajištěno krystalem 22,118400 MHz. Původně jsem pracoval s krystalem 11,0592 MHz, ale po zařazení náročných funkcí (počítání s šestnáctibitovými čísly) bylo nutno zvýšit výpočetní výkon procesoru.

Na všechny čtyři brány procesoru jsem paralelně připojil odporové sítě (RN1 až RN4) se zvedacími odpory 4k7.

Při návrhu desky bylo třeba zohlednit také možnost zvyšování výkonu celého systému, a tak je možno patici osazovat různými mikropočítači, které jsou vývodově kompatibilní s mikroprocesorem 80C51. Nyní je osazen typ 89C52. Ten je vybaven integrovanou pamětí ROM o kapacitě 8 kB, pamětí RAM o kapacitě 256 B a třetím časovačem, který však můj program nepoužívá. Při vývoji programu jsem používal procesor typu 89C8252, který navíc obsahuje rozhraní SPI, které umožňuje přeprogramovat interní paměť programu (FLASH). Další integrované periferie (EEPROM, hlídací obvod tzv. *watchdog*) jsem nevyužíval.

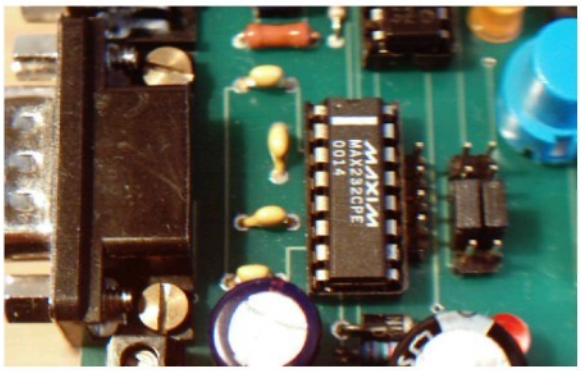


Obr.11 Procesor AT98C8252 a konektor pro připojení programovacího kabelu, LP

V současné době je zřejmě nejvýkonnějším použitelným procesorem řady x51 typ DS80C320. Vykonání jednoho strojového cyklu u něj zabere jen čtyři hodinové cykly, zatímco u klasického typu 80C51 je potřeba dvanácti cyklů. Navíc je schopen pracovat s krystalem o frekvenci až 33 MHZ ! Dosahujeme tak rychlosti srovnatelné se šestnáctibitovými procesory. DS80C320 přináší i další výhody kromě vyšší rychlosti. Přidává druhý plně duplexní hardwarový sériový port, sedm dalších zdrojů přerušení, programovatelný *watchdog*, přerušení a reset při poklesu napájení. Dále nabízí dva datové pointry (DPTR), což zrychluje blokové přesuny dat v paměti. Může též nastavovat rychlost přístupu k pamětem mimo procesor od dvou do devíti strojních cyklů.
Sériové linky UART mohou při správném propojení na řídící desce pracovat současně. Jedna jako RS232 a druhá jako RS485. DS80C320 neobsahuje interní paměť programu, ale na řídící desce je patice pro externí paměť ROM typu 27C256 s kapacitou 32 kB, nebo vývodově kompatibilní.

5.2 Sériové linky

Systém obsahuje linku RS232 a linku RS458. Jako převodník napěťových úrovní je pro RS232 užito obvodu MAX232, a pro RS485 obvodu 7517B. Propojkami ve schématu označenými jako JP2 je možno přesměrovat signál buď na MAX232, nebo na 7517B. Propojením vývodů 5-7 a 6-8 vybereme linku RS485, propojením 5-3 a 6-4 vybereme RS232. Při použití procesoru DS80C320 pro připojení obou linek propojíme buď vývody 9-7, 10-8, 5-3 a 6-4 , nebo vývody 1-3, 2-4, 5-7, 6-8 tím linky prohodíme.



Obr.12 Při tomto nastavení propojek je vybrána linka RS232, LP.

Vývod procesoru P1.0 ovládá směr přenosu (vysílání/poslech). Při vysílání svítí žlutá kontrolní dioda, na schématu označená jako LED3.

5.3 Paměť programu

Protože jsem použil mikroprocesor s interní pamětí programu (typu *flash*) a můj program se do této paměti vešel, nevyužil jsem možnost externí paměti. Tu můžeme po naprogramování vložit do patice pro obvod (ve schématu označený jako IC5). Po vynulování procesoru (reset) si procesor zjistí logickou úroveň na vývodu EA, Je-li 0, tvoří programovou paměť pouze paměť vnější, je-li 1, čtou se instrukce v adresovém prostoru 0000h až 0FFFh z vnitřní paměti, instrukce mimo tento prostor (tj. 1000h až FFFFh ze zbývajících 60 kB) z vnější paměti programu. Logickou hodnotu na vývodu EA lze přepínat pomocí konektoru označeného jako JP4. Při spojení vývodu 1 a 2 tohoto konektoru bude EA=1. Při spojení 2 a 3 bude EA=0. Mechanismus pro přístup do vnější paměti je podrobně popsán v oddíle *Přístup do vnější paměti* v knize [1].

5.4 Paměť dat – RAM

Pro práci s pamětí RAM má procesor řady x51 různé instrukce pro adresování vnitřního a vnějšího paměťového prostoru. Do vnitřní paměti jsem ukládal všechny proměnné. Vnější paměť jsem nepoužil a tudíž jsem nemusel osazovat řídící desku integrovaným obvodem IC6. Ten představuje statickou RAM paměť typ 62256 o velikosti 32 kB. Tato paměť se nachází na adresách od 0000h do 7FFFh. Nad tímto prostorem na adresách 8000h až FFFFh se nacházejí vstupní a výstupní periferie, ke kterým přistupujeme jako k vnější RAM paměti. Sem jsem umístil ovládání krokových motorů manipulátoru, ovládání pneumatických ventilů třídiče, čidla a snímače manipulátoru i třídiče a přepínače manuálního ovládání manipulátoru.
Mechanismus pro přístup do vnější paměti je opět podrobně popsán v [1].

5.5 Přístup k periferiím v paměťovém prostoru

Řídící deska disponuje osmi výstupními a šesti vstupními periferiemi. Pokud použijeme zápis nebo čtení z určité adresy (větší než 7FFFh), na bráně procesoru P0 se objeví nižší část šestnáctibitové adresy a ta je signálem ALE přepsána pomocí záchytného registru IC3 na sběrnici AB. Brána P2 udává vyšší část adresy sběrnice AB. Nastavený adresový bit A15 odpojí vnější paměť RAM (IC6) a povolí současně při nulovém bitu A3 dekodéry adresy IC10 a IC11. Dále už rozhodují pouze signály RW a RD, zda budeme hodnotu číst, nebo zapisovat. Hradla IC7A a IC7C slouží pouze k negaci logických hodnot.

Pokud provádíme zápis, signál WR vybere dekodér adresy pro zápis (IC11) a zároveň nastaví obousměrnou bránu (obvod IC4 – 74245) jako výstupní. Tak se přepíše hodnota z brány

procesoru P0 (DATA) na sběrnici DB a z této sběrnice si tuto hodnotu převezme příslušný záchytný registr, který přímo ovládá vybranou periferii.

Pokud provádíme čtení, signál RD vybere dekodér adresy pro čtení (IC10) a zároveň nastaví obousměrnou bránu IC4 jako vstupní. Hodnota kterou žádáme je na sběrnici DB přepsána vybraným registrem. Brána P0 pak přečte hodnotu bajtu ze sběrnice DB.

Výstupní periferie se ovládají tak, že na konkrétní adresu zapíšeme požadovaný bajt a ten je držen záchytným registrem do té doby, než je opět přepsán. Záchytný registr, do kterého se má hodnota ze sběrnice přepsat je vybrán v závislosti na adrese obvodem IC11 - 74138. Jako záchytný registr jsem použil obvod 74574.

Vstupní periferie se ovládají tak, že z konkrétní adresy čteme požadovaný bajt. Ten na sběrnici předá právě vybraný registr tvořený obvodem 74541. Registr vybírá v závislosti na adrese obvod IC10 – 74138.

5.6 Adresy periferií

výstupní periferie: 8000h 1. a 2. krokový motor manipulátoru
8001h 2. a 3. krokový motor manipulátoru
8002h pneumatické ventily třídiče
8003h až 8007h nevyužito

vstupní periferie: 8000h čidla a spínače krajních poloh manipulátoru (vyvolávají přerušení)
8001h čidla a spínače krajních poloh manipulátoru (vyvolávají přerušení)
8002h spínače manuálního ovladače manipulátoru
8003h nevyužito
8004h čidla na třídiči
8005h až 8007h nevyužito

SVI a SV2 jsou konektory typu ML20. Pomocí těchto konektorů můžeme připojit přídavná zařízení k vstupním bránám na adresách 8002h až 8005h. Konektory SV3 a SV4 slouží pro výstupní brány na adresách 8003h až 8007h. Nad adresou 8007h nejsou na desce žádné další periferie.

5.7 Externí přerušení

Přerušovací systém mikroprocesoru x51 obsahuje dvě externí přerušení. První externí přerušení INT0 je možno aktivovat spínačem „STOP“ umístěným přímo na řídící desce (modré tlačítko). Tento spínač můžeme nahradit jiným spínačem, který připojíme na konektor JP20.

Druhé externí přerušení INT1 je aktivováno kterýmkoli čidlem koncového dorazu. Logické signály z těchto čidel, jsou svedeny přes osmivstupová hradla (IC13 a IC15) do dvou signálů K1 a K2. Nad těmito signály je buď provedena funkce NOR pomocí hradla IC7B a výsledek ovládá přímo INT1. Na konektoru JP3 musí být propojen vývod 2 a 3. Jiná alternativa je propojit na JP3 vývody 1-2 a 3-4. Pak je signál K2 přiveden na vývod mikroprocesoru P1.4, na kterém má procesor DS80C320 další externí přerušení INT2 a K1 ovládá INT1.

Čidla spínající na logickou jedničku jsou přivedena konektorem JP12 do vstupů osmivstupového hradla OR. Výsledkem je signál K1. Pokud se změní logická hodnota výstupu z jakéhokoliv snímače na úroveň 1, hradlo změní stav na 1 (K1=1). Integrovaný obvod ve funkci osmivstupového hradla OR je typu 744078. Původně jsem chtěl použít obvod 4078, ale ten nemá na vývod 1 připojen výstup a je k dispozici pouze negovaný výstup na vývodu 13.

Čidla spínající na logickou nulu jsou přivedena konektorem JP15 do vstupů osmivstupového hradla NAND. Výsledkem je signál K2. Pokud se změní logická hodnota výstupu z jakéhokoliv snímače na úroveň 0, hradlo změní stav na 1 (K2=1). Integrovaný obvod ve funkci osmivstupového hradla NAND je typu 7430.

Propojky JP13 a JP16 slouží k odpojení signálu od hradla. Toho využijeme, pokud nechceme, aby konkrétní čidlo nevyvolávalo přerušení. Integrované obvody IC12 a IC16 (74541) slouží jako oddělovače a zároveň budiče indikačních diod (LED). Ty jsou umístěny přímo na řídící desce. Propojkami JP11 a JP14 můžeme indikaci vyřadit.

Po vyvolání přerušení INT1 musíme programově zjistit, které čidlo přerušení vyvolalo. Hodnotu brány, kde máme hodnoty čidel spínající na logickou 1 máme na adrese 8000h a čidla spínající na logickou nulu máme na adrese 8001h.

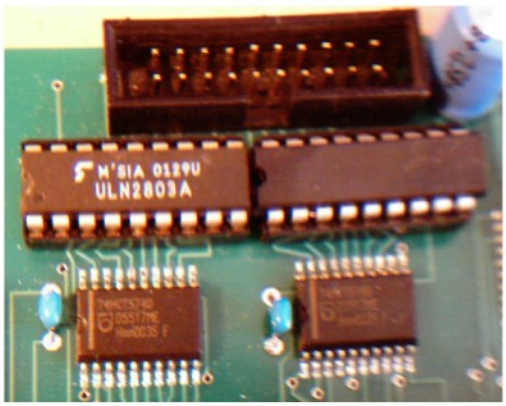
5.8 Výkonové obvody

Na řídící desce jsou umístěny čtyři obvody L298. Těmito obvody budíme cívky krokových motorů. Obvod L298 obsahuje dvojici výkonových H-můstků, které mohou ve špičkách dodávat proud od cívek až 4 A. Maximální přípustné napětí pro tento obvod je 46 V. Obvod musí být doplněn ochrannými diodami, aby naindukované napětí na cívkách po změně polarity nezničilo výkonové tranzistory. V zapojení těchto obvodů nesmí chybět kondenzátor

s kapacitou nejméně 470 μF připojený mezi napájecí vývod (4) a zemění. Při mých pokusech s tímto obvodem jsem tento kondenzátor zapomněl připojit a jelikož jsem neměl zdroj dostatečně odolný proti proudovým špičkám, docházelo k resetu procesoru. Výstupy z těchto H-můstků jsou přivedeny na konektory ML10 (SV7 a SV 8), kam připojíme cívky krokových motorů.

Dalšími výkonovými obvody jsou IC28 a IC29. Jsou to pole osmi výkonových darlingtonových tranzistorů.

IC28 je typ ULN2803A. Vstupy tohoto obvodu můžeme ovládat logikou TTL a výstupy nám zemní se spínacím proudem až 0.5 A. Logická hodnota výstupu je oproti vstupu negovaná. Maximální povolené napětí na tomto obvodu je 50 V. Obvod má již ochranné diody integrované přímo na čipu. Tento obvod jsem využil ke spínání cívek elektricky ovládaných pneumatických ventilů.



Obr.13 Registry a výkonové obvody pro ovládání elektromagnetických ventilů, LP.

IC29 je typ UDN2981A. Parametry má stejné jako ULN2803A, ale jeho výstup není negovaný. Na řídící desce jsem tento obvod z důvodů finanční nedostupnosti nakonec nahradil přibližně desetkrát levnějším vývodově kompatibilním obvodem o srovnatelných vlastnostech TD62783AP.

5.9 Napájení

Svorkovnice X1 slouží k připojení napájecího napětí. Pro výkonovou část zapojení potřebujeme napětí 24 V. Proudová spotřeba je díky krokovým motorům přes 2 A. Napájecí zdroj by měl i s rezervou být schopen dodat proud 3 A. Pro logickou část zapojení potřebujeme napětí 5 V. Toto napětí můžeme na desku přivést přímo, nebo můžeme využít stabilizačního obvodu na desce. Tento obvod zařadíme propojením vývodů na konektoru JP6.

Pro stabilizační obvod je použit stabilizátor 7805T, ochranná dioda a nezbytné kondenzátory.

5.10 Ostatní

Nulovací obvod je tvořen pouze RC článkem a spínačem pro aktivaci. Reset je vyvolán při sepnutí vývodu RST na +5 V.

Nezmínil jsem se zde o zapojení EEPROM paměti I^2C (IC2), zapojení je popsáno v kapitole o I^2C sběrnici.

6 Laboratorní manipulátor

Manipulátor, ke kterému jsem navrhoval řídící systém, je angulární (antropomorfní, mnohaúhlový) typ se třemi stupni volnosti. Robot angulárního typu je definován jako mechanismus se třemi rotačními jednotkami s takovým rozvržením, že je možná jedna rotace kolem svislé osy a další dvě rotace jsou kolem vodorovných a rovnoběžných os.

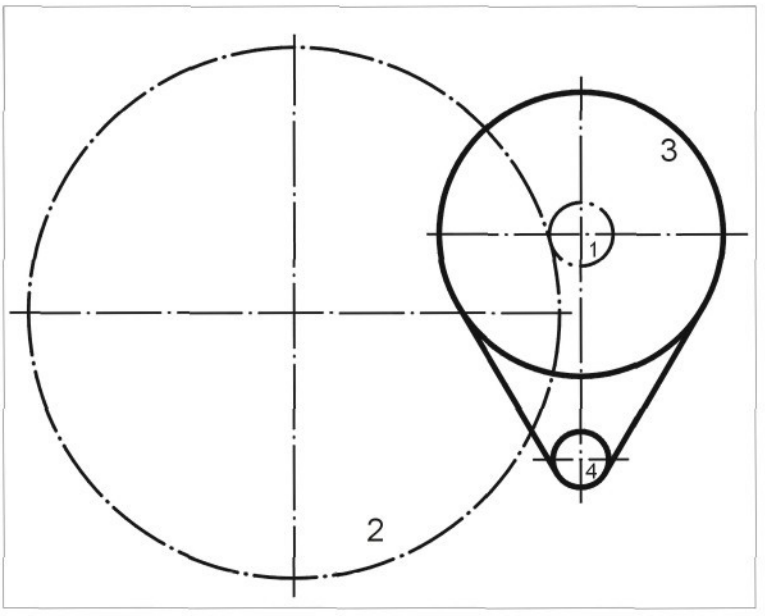
Pohyb mechanismu zajišťují tři pohonné jednotky. Jedna pohání rotaci kolem svislé osy a zbývající dvě pohánějí zvedák a rameno.

Vyobrazení modelu manipulátoru je v příloze č. 7.2. Jako pohonné jednotky jsou použity čtyři bipolární krokové motory, přičemž čtvrtý motor ovládá úchopnou hlavici manipulátoru. Typové označení motorů je MAE HY 100-1713-020 A4. Ačkoli jsem kontaktoval výrobce, nepodařilo se mi bohužel získat autorizovanou technickou specifikaci vlastností motorů.

Pracovní prostor manipulátoru je znázorněn v ovládacím programu na PC.

6.1 Rotace kolem svislé osy

Kroutící moment vyvolaný krokovým motorem je předán pomocí převodu ozubeným řemenem pastorku, který pohání ozubené kolo. Mechanismus je schématicky naznačen na obrázku 14.



Obr.14 Převod rotace: 1 - pastorek, 2 - ozubené kolo, 3 - hnaná řemenice, 4 - hnací řemenice, LP.

Průměr hnací řemenice je 10,5mm (d_4), průměr hnané řemenice je 53,5mm (d_3). Tyto hodnoty jsem naměřil posuvným měřidlem přímo na manipulátoru, je tedy nutno počítat

s určitou nepřesností. Počet zubů na pastorku je 12 (z_1=12), na ozubeném kole pak 100 (z_2=100).

Výpočet převodového poměru *i*:

$$i_{34} = \frac{d_3}{d_4} = \frac{53,5}{10,5} = 5,095 \qquad i_{12} = \frac{z_2}{z_1} = \frac{100}{12} = 8,\overline{33}$$

$$i = i_{12} \cdot i_{34} = 42,460$$

úhel o který se pootočí hnací řemenice (4) při jednom kroku motoru α_4 = 3,6°

$$\alpha_2 = \frac{\alpha_4}{i} = \frac{3,6}{42,460} = 0,084785°$$ (Kvůli chybám měření a mechanickým vůlím jde pouze o teoretickou hodnotu)

Při kalibraci rotace se začne manipulátor otáčet proti směru hodinových ručiček, dokud indukční čidlo nesignalizuje dosažení koncové polohy. Tam má rotace nulovou polohu. Pokud se chce robot dotočit až k pravému koncovému dorazu, musí krokový motor rotace vykonat 3467 kroků, což odpovídá úhlu 293°.

6.2 Zvedák a rameno

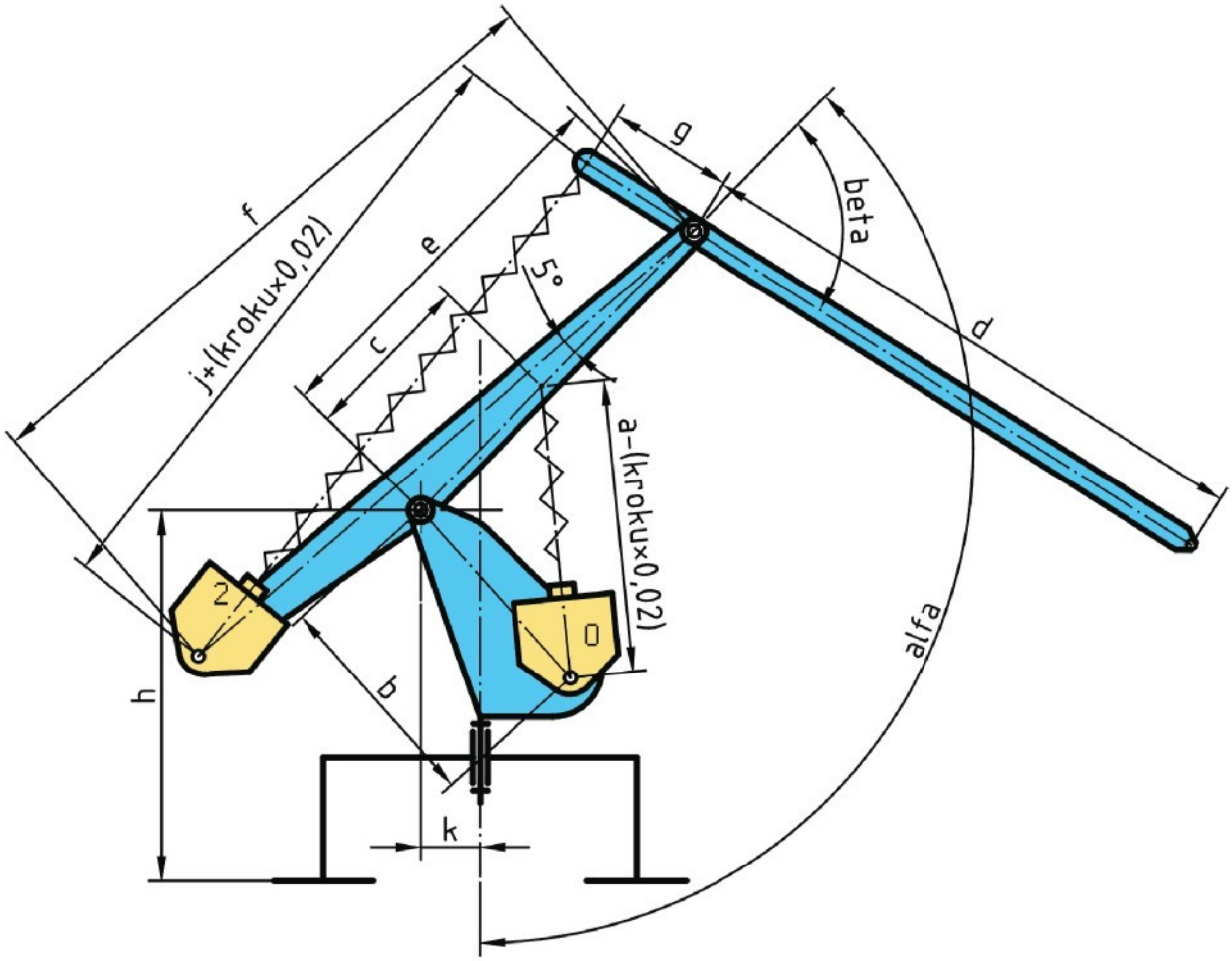
Na obrázku 15 je zobrazen mechanismus zvedáku a ramene. Zvedák je ovládán motorem 0 a rameno je ovládáno motorem 2. Číslování motorů vychází z programu řídícího systému.

Naměřené hodnoty:	a = 114,7mm	e = 158mm
	b = 91mm	f = 264mm
	c = 70mm	g = 51mm
	d = 238mm	h = 150mm
	j = 213,8mm	k = 24mm

Kvůli vlastnostem použitých měřidel a technik měření jsou tyto hodnoty pouze orientační. Vykoná-li motor (tzn. provede-li jednu otáčku) posune pohybový šroub matici o 2 mm, na jeden krok tedy připadá 0,02 mm.

Při kalibraci stoupá zvedák až do polohy, v níž sepne koncový spínač; tuto polohu zvedáku tedy řídící systém interpretuje jako nulovou polohu, při níž $\alpha = 130°$. Má-li zvedák dojet na opačný koncový doraz, vykoná motor 0 1503 kroků; v této poloze $\alpha = 102°$.

Rameno se při kalibraci zvedá tak dlouho, dokud nesepne příslušný koncový spínač. Tuto polohu ramen interpretuje řídící systém jako nulovou polohu, v níž $\beta = 15°$. Má-li rameno dojet na opačný koncový doraz, vykoná motor 2 4082 kroků.



Obr.15 Schéma zvedáku a ramene, LP.

6.3 Souřadnice manipulátoru

Protože použitý mikroprocesor nepočítá explicitně s goniometrickými funkcemi, pracuje řídící systém výhradně se strojními souřadnicemi. Tyto strojní souřadnice chápeme jako počet kroků příslušného motoru od nulové polohy. To se při pohybu projevuje inkrementací nebo dekrementací proměnné *pozice* ve struktuře krokového motoru.

Strojní souřadnice lze po sériové lince předávat nadřazenému systému, který je (při využití hodnot rozměrů manipulátoru) přepočítá na souřadnice válcové. Nadřazený systém může řešit

také opačnou úlohu. Ze zadaných válcových souřadnic vypočítá souřadnice strojní, které odešle řídící desce manipulátoru. Souřadnice se pak buď uloží do paměti EEPROM na desce, nebo dojde k přesunu manipulátoru na tyto souřadnice.

Výpočet válcových souřadnic:

p-počet kroků od nulové polohy na motoru 1 (rotace)

q-počet kroků od nulové polohy na motoru 0 (zvedáku)

r-počet kroků od nulové polohy na motoru 2 (ramena)

Rotace: $\varphi = \alpha_2 \cdot p$ α_2 viz. výše

α: $\cos\alpha = \frac{a^2 - b^2 - c^2}{-2bc} \Rightarrow \alpha = \arccos\frac{a^2 - b^2 - c^2}{-2bc}$

β: $\cos\beta = \frac{g^2 - j^2 - f^2}{2gf} \Rightarrow \beta = \arccos\frac{g^2 - j^2 - f^2}{2gf}$

Výška $H = h + e \cdot \cos(alfa) + d \cdot \sin(90 - (alfa + beta))$

Rádius: $R = -k + e \cdot \sin(alfa) + d \cdot \cos(90 - (alfa + beta))$

6.4 Úchopná hlavice

Manipulátor má univerzální aktivní úchopnou hlavici se silovým charakterem uchopení. Úchopná hlavice je poháněna krokovým motorem. Motor otáčí pohybovým šroubem, který otevírá nebo zavírá klešťový mechanismus hlavice. Držení materiálu je navrženo jako mechanické s třecími silami. Úchopná síla hlavice je závislá na kroutícím momentu od pohonné jednotky, který bohužel není znám. V případě běžných pokusných těles však tato síla byla vždy dostatečná.

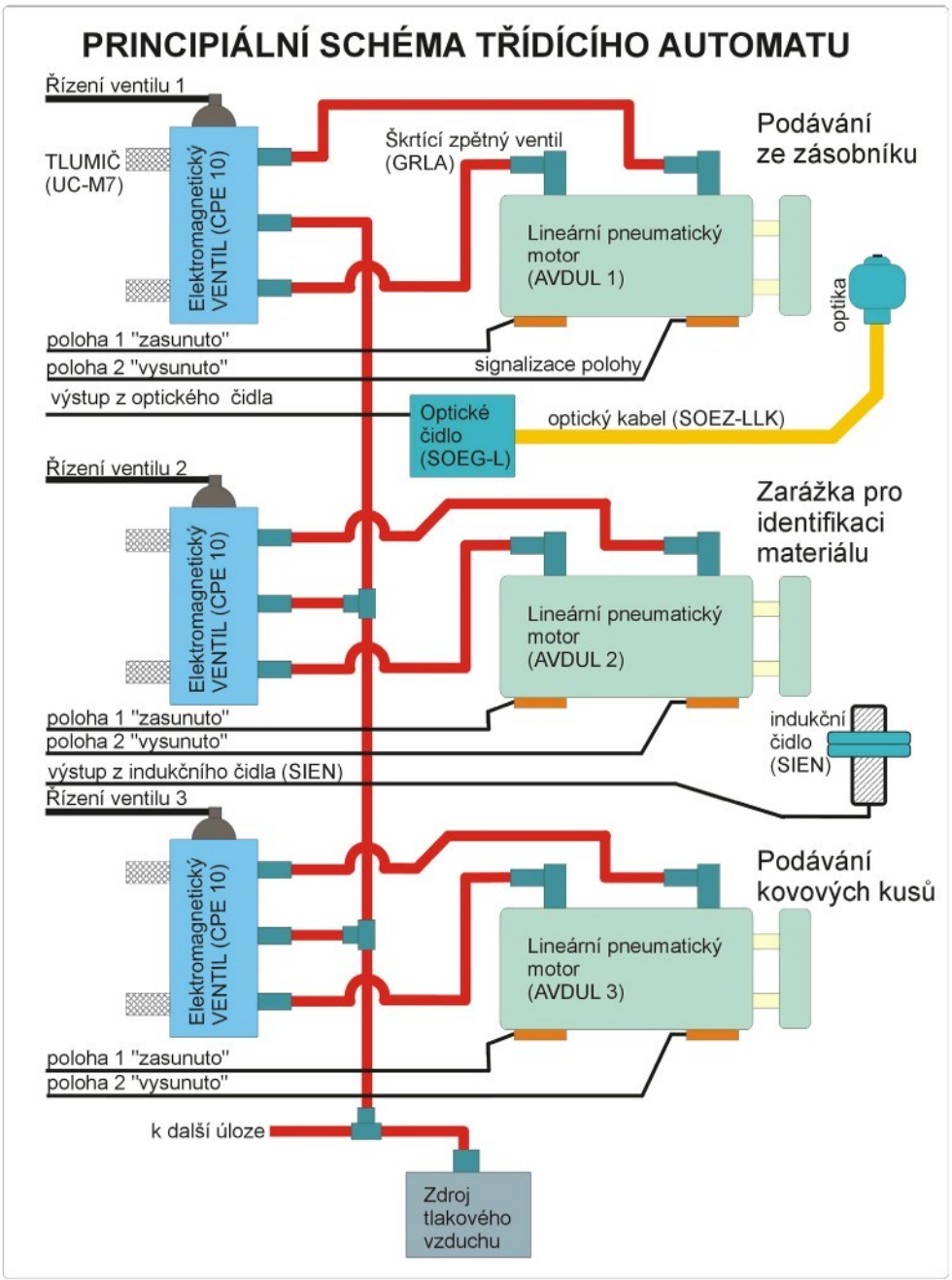
S úkonem uchopení (stisku) materiálu je spojen problém, jak stanovit, kdy má pohonná jednotka přestat přibližovat čelisti. Mají-li uchopovaná tělesa různé rozměry, je pokaždé jiná i poloha čelistí, tedy i úhel otočení motoru. Pokud by byl rozměr tělesa větší než předpokládá řídící systém, motor by začal po maximálním stisku tělesa ztrácet kroky. Ztrácí-li motor kroky, přenesou se do konstrukce robota nežádoucí vibrace a řídící systém ztratí kontrolu nad polohou mechanismu. Tento problém jsem vyřešil tak, že jsem doplnil čelisti o mikrospínač mechanicky

spojený s jehlou, která po stisku tělesa mikrospínač sepne. Sepnutí mikrospínače řídící systém vyhodnotí a zastaví motor. Toto řešení je pro mou úlohu postačující, protože materiál má mít vždy tvar válce. Lepším řešením, které by nebylo závislé na tvaru předmětu, by bylo např. použití tenzoru, jehož signál by vyhodnocoval komparátor. Protože čelisti úchopné hlavice jsou relativně pružné, bylo by možné připevnit tenzor na bok čelistí a připojit ke komparátoru jako proměnný odpor. Nastavení komparační hodnoty by záviselo na požadované úchopné síle, motor by se vypínal po překlopení komparátoru.

Při kalibraci robota se kleště otevírají, dokud nestisknou dorazový spínač, což indikuje otevření kleští, tj. jejich nulovou polohu. Aby motor kleště zavřel, musí vykonat 647 kroků.

7 Model třídiče výrobků

Automat je tvořen třemi lineárními pneumatickými motory, které jsou ovládány přepínacími elektromagnetickými pneumatickými ventily. Každý motor má dvě čidla indikující polohu. Stabilní polohy jsou dvě (vysunuto/zasunuto). Zásobník třídiče obsahuje optické čidlo pro zjištění přítomnosti materiálu. Indukční přibližovací čidlo analyzuje, zda je tříděný předmět kovový či nekovový. Pro ovládání postačí dvoustavová logika doplněná čekacími smyčkami.



Obr.16 Funkční schéma třídiče materiálu, LP.

První motor vysouvá materiál ze zásobníku. Ten se pak zarazí o zarážku ovládanou druhým motorem. Je-li předmět z kovu, je zarážka motorem odstraněna a materiál propadne na pozici 1. Je-li ovšem předmět z nekovového materiálu, postrčí třetí motor předmět tak, že propadne na pozici 2.

7.1 Zapojení svorkovnic na třídiči

Na třídícím automatu jsou dvě svorkovnice. Na svorkovnici SV1 jsou připojena všechna čidla a na svorkovnici SV2 jsou připojeny elektromagnetické ventily. Napájecí napětí je přivedeno na obě svorkovnice.

SV1	1	+5V !zaslepeno!
	2	Zem
	3	Snímač – motor1 (třídič) vysunuto
	4	Snímač – motor1 (třídič) zasunuto
	5	Snímač – motor2 (zarážka) vysunuto
	6	Snímač – motor2 (zarážka) zasunuto
	7	Snímač – motor3 (zásobník) vysunuto
	8	Snímač – motor3 (zásobník) zasunuto
	9	Indukční snímač – kov/nekov
	10	Optický snímač přítomnosti předmětu
SV2	1	+5V !zaslepeno!
	2	Zem
	3	+12 V
	4	Ventil 1 – zarážka
	5	Ventil 2 – třídič na 2. pozici
	6	Ventil 3 – vysunutí ze zásobníku
	7	Nezapojeno
	8	Nezapojeno
	9	Nezapojeno
	10	Nezapojeno

Tab.1. Zapojení svorkovnic na třídiči, LP.

Řídicí systém čte čidla třídiče na adrese 8005h.

7	Motor1 - vysunuto
6	Motor1 – zasunuto
5	Motor1 – vysunuto
4	Motor1 – zasunuto
3	Motor1 – vysunuto
2	Motor1 – zasunuto
1	Optické čidlo
0	Indukční čidlo

Tab.2. Význam přečtené hodnoty z adresy 8005h, LP.

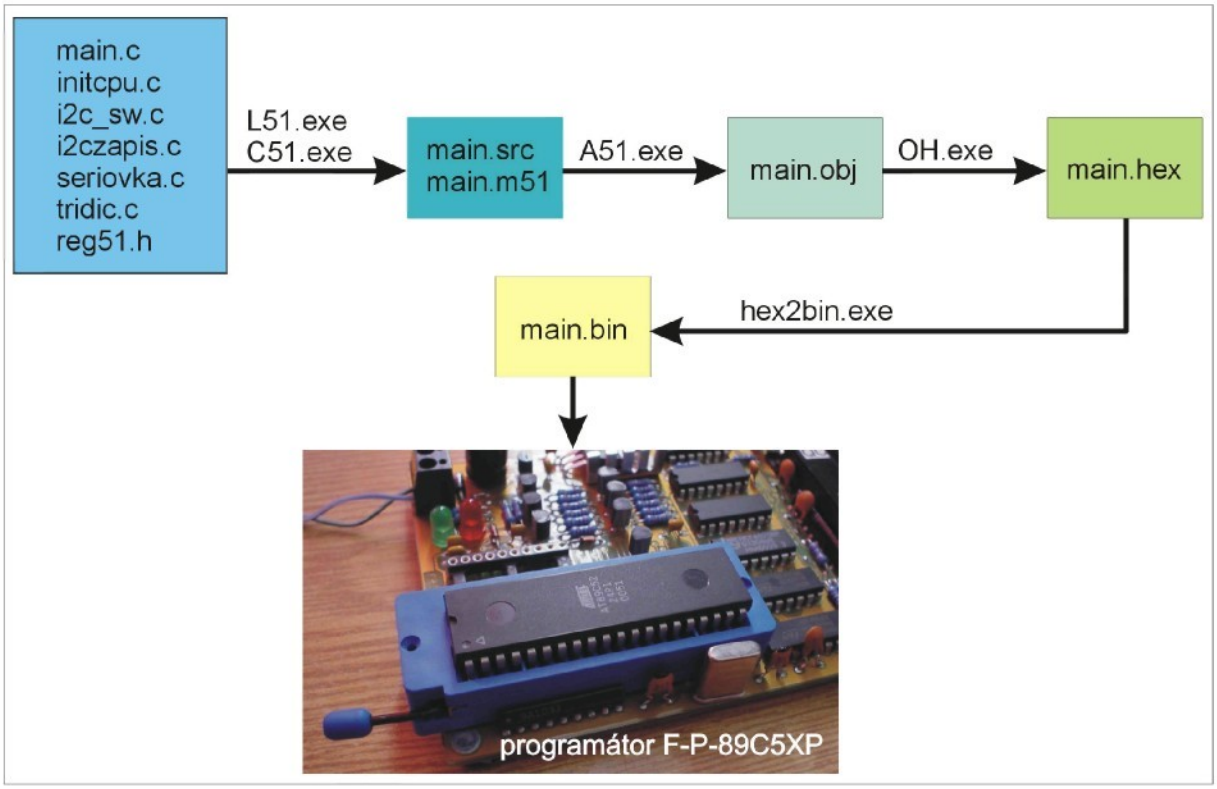
Ventily jsou řídicím systémem ovládány na adrese 8002h.

7	Nic
6	Motor3 (zásobník)
5	Motor1 (třídič)
4	Motor2 (zarážka)
3	Nic
2	Nic
1	Nic
0	Nic

Tab.3. Význam zapisované hodnoty na adresu 8002h, LP

8 Popis programu mikroprocesoru

Zdrojový program pro mikroprocesor je napsán v jazyce C. Tento kód je uložen do souborů s příponou c (*.c). Soubor *main.c* je sestaven *linkerem* L51.exe a přeložen překladačem C51.exe. Výsledek překladu vytvoří mj. soubor *main.src*, který obsahuje program převedený do assembleru. Následně A51.exe přeloží *main.src*, a vytvoří tak *main.obj*. Ten se dále převádí programem oh.exe na *main.hex*, z něhož pak program hex2bin.exe vytvoří soubor *main.bin*. Výsledný soubor *main.bin* se zavádí pomocí hardwarového programátoru do programové paměti mikroprocesoru. Pro názornost je celý postup naznačen na obrázku 17.



Obr.17 Převod formátů, LP.

Pro kompletní zdrojový kód programu viz. přílohy 1 až 6.

8.1 Soubor main.c

V souboru main.c je uložena většina funkcí programu. Zde je uložena funkce *main* a dále se zde nacházejí definice většiny globálních proměnných a konstant. Do tohoto souboru jsou pomocí direktivy *#include* vloženy další části programu. Ty jsou uloženy v souborech *initcpu.c*, *i2c_sw.c*, *i2czapis.c*, *seriovka.c*, *tridic.c* a *reg51.h*. Jejich obsah bude vyložen dále.

8.1.1 Konstanty a globální proměnné

Konstanty definované direktivou #define

VERZE :	vrací po sériové lince verzi programu. Je-li např. 101, verze programu je 1.01
MOTORU:	počet motorů, které má program ovládat
ROZJEZD:	délka rozjezdové (dojezdové) sekvence
AKCELERACE:	roztahuje rozjezdovou (dojezdovou) sekvenci Délka = ROZJEZD * AKCELERACE
ROTACEMAX, KLESTEMAX, RAMENOMAX, ZVEDAKMAX:	hodnoty strojních souřadnic koncových dorazů
IC1R, IC2R, IC3R, IC4R:	hodnoty strojních souřadnic indukčních čidel, při nájezdu zprava
IC1L, IC2L, IC3L, IC4L:	hodnoty strojních souřadnic indukčních čidel, při nájezdu zleva
XTAL:	hodnota použitého krystalu
TIMER:	hodnota udávající čas, za jaký má časovač vyvolávat přerušení (1 ms)
BAUDDIV:	hodnota udávající přenosovou rychlost sériové linky (57600)

Konstanty uložené v paměti programu

Aby překladač identifikoval konstanty, které má umístit v paměti programu, musíme použít direktivu *code*. Deklarace tedy vypadá takto:
unsigned char code promenna;

krokcas:	jde o pole, v němž jsou uloženy časy v ms mezi kroky krokových motorů
kombcivek:	pole s konstantami, které vyjadřují různé kombinace řídících signálů, ovládající cívky krokových motorů

Bitové proměnné

Direktivou bit překladači oznámíme, že jde o bitovou proměnnou v bitové oblasti. Deklarace pak vypadá takto: *bit promenna;*

Pokud chceme, aby proměnná ovládala nějaký bit speciálního funkčního registru, použijeme direktivu *sbit*. Deklarace potom vypadá takto: *sbit Dioda=P1^7;*

Dioda:	rozsvěcí a zhasíná oranžovou diodu, ve schématu označenou jako LED3
P11:	povoluje a zakazuje výstup registrům ovládajícím výstupní periferie
P32:	vývod externího přerušení INT0
automat_enable:	povoluje provádění funkce automatického provozu (viz.níže)
napln_enable:	povoluje provádění funkce, která vybírá koncové souřadnice pohybového vektoru, nebo řídícího příkazu z EEPROM paměti.
kalibruje:	proměnná je nastavena na 1 po vstupu do funkce *nulovapoloha*. Před ukončením této funkce se proměnná nastaví na 0.
pocitani_enable:	povoluje funkci současného dojezdu všech motorů.
kalibr:	tato proměnná je použita při kalibraci rotace na indukčních čidlech

Proměnné uložené ve vnější paměti dat

V programu jsem vnější paměť nepoužil. Použil jsem však periferie umístěné v adresovém prostoru vnější paměti dat. Pro umístění do vnější paměti dat použijeme direktivu *xdata* a pro zadání konkrétní adresy použijeme direktivu *_at_*.

Deklarace potom vypadá takto: *xdata unsigned char promemna _at_ 0x8000;*

adresa8000, adresa8001 atd.:

Externí vstupy a výstupy jsou popsány v kapitole *Přístup k periferiím v paměťovém prostoru*.

Proměnné uložené ve vnitřní paměti dat

tickfurt:	tuto proměnnou časovač inkrementuje po jedné milisekundě. Slouží tedy k odměřování času.
brzdi:	hodnota udává, po kolika krocích se má začít brzdit (tj. prodlužovat čas mezi kroky)
kmax:	maximální počet kroků, který se má vykonat v aktuálním pohybovém vektoru
poradi:	udává, kolikátý krok se právě vykonává v aktuálním pohybovém vektoru
index_tab:	index řádku v pohybové tabulce uložené v EEPROM
akcelerace, kolik:	tyto proměnné jsou používané v rozjezdové a dojezdové sekvenci
cas:	hodnota času mezi kroky. Tuto hodnotu časovač dekrementuje a po vynulování této hodnoty může motor provést další krok.
cas_index:	index do pole *krokcas*

Struktura příslušející krokovému motoru

Každý krokový motor má svou strukturu, v níž jsou uloženy proměnné určující stav motoru. Tyto struktury jsou uloženy ve vnitřní datové paměti jako pole.

Definice struktury vypadá takto:

```
typedef struct
{
 uchar zapnuty;      povolení motoru
 uint  pozice;       aktuální pozice (strojní souřadnice) od nulové polohy
 uint  stoppozice;   kam se má motor dotočit - pozice od nulové polohy
 uint  kroku;        kolik vykoná motor kroků v aktuálním vektoru pohybu
 uint  m_last;       poslední hodnota výpočtu algoritmu pro současné dojetí motorů
 uchar krok;         index do pole kombcivek
 char  smer;         zadání směru otáčení: 1 = tam, -1 = zpet, 0 = nikam
 uchar toc;          je-li 1, bude se při dalším volání funkce automat točit motor
} MOTOR;
MOTOR data motor[MOTORU];
```

Pro některé proměnné by stačilo použít bitovou proměnnou, neumožňuje to však použitý překladač C51.exe.

8.1.2 Hlavní program - main

Hlavní program v C se vždy jmenuje ***main*** a musí se v programu vždy uvést. Je to první funkce volaná po spuštění programu. Hned na začátku musíme nastavit registry speciálních funkcí (SFR). V těchto registrech jsou uloženy všechny informace důležité pro činnost mikroprocesoru a jeho periferních obvodů integrovaných na čipu procesoru, jako jsou čítače, sériový kanál, přerušovací systém aj. Nastavení SFR provedeme voláním funkce *initcpu*, kterou nalezneme v souboru *initcpu.c*. Význam jednotlivých položek SFR nalezneme v katalogových listech výrobce příslušného mikroprocesoru (pro AT98C52 viz. [10]).

Dalšími příkazy nastavíme sériovou linku RS485 na příjem (DTR=1), zhasneme diodu (Dioda=0) a povolíme globální přerušení (EA=1). Musíme také zajistit nulování komunikačních obvodů I^2C EEPROM paměti. To provedeme voláním funkce *I2Creset*, která je uložena v souboru *i2c_sw.c*. Následující funkce *zablikej* rozsvítí a za 250 ms zhasne diodu (označenou ve schématu jako LED3). Tak se ujistíme, že program nastartoval. Příkazem P11=0 aktivujeme registry, které ovládají výstupní periferie (budiče krokových motorů a elektromagnetických pneumatických ventilů). Zapsáním nuly na adresu 8002h zasuneme všechny pneumatické motory. Dále voláme funkci, která zajišťuje kalibraci manipulátoru (*nulovapoloha*) a po ní zajistíme výběr pozice, nebo řídícího příkazu. K tomu je určena funkce *napln*.

Výše uvedená inicializace systému se provede pouze jednou, a to po startu programu. Poté program skočí do nekonečné smyčky, v níž se opakuje obsluha třech různých stavů běhu programu. V prvním stavu zkontrolujeme, zda přišel příkaz po sériové lince. Pokud ano, bude proměnná *delka* nenulová. Přišlý příkaz vykonáme, vynulujeme proměnnou *delka* a přejdeme na kontrolu druhého stavu. Tím je ošetření manuálního ovládání manipulátoru. Pokud je smáčknuté jakékoliv tlačítko na ovladači a čas mezi kroky motoru již překročil zadaný interval, systém vyhodnotí o které jde a pootočí příslušným motorem o jeden krok zadaným směrem. Příkaz *continue* přesune běh programu na začátek smyčky. Nebylo-li žádné tlačítko stisknuté, obslouží se třetí stav. Tím je automatický provoz, který zajišťuje pohyb robota po trajektorii zapsané v EEPROM, nebo volání funkce obsluhující třídič. Aby se automatický provoz mohl provést, musí být proměnná *automat_enable* nenulová. Je-li tomu tak, pak při nenulové

proměnné *pocitani_enable* voláme funkci *pocitej*, která počítá algoritmus současného dojezdu motorů. Dále se provede volání funkce *automat*. Ta obslouží krokové motory a vrátí nulovou hodnotu, pokud všechny motory nejsou na konci aktuálního pohybového vektoru. Pokud funkce *automat* vrátí nenulovou hodnotu, vybere se (pokud to není zakázáno) další koncová poloha, nebo řídící příkaz z EEPROM. K tomu slouží funkce *napln*. Následuje funkce *zablikej*, která problikne diodou (LED3). Tím se indikuje výběr z EEPROM.
Odtud program skočí na začátek smyčky a vše se opakuje.

8.1.3 Obsluha přerušení

Čítač 0

V obsluze tohoto přerušení nastavíme registry TH0 a TL0 tak, aby se další přerušení vždy vyvolalo po jedné milisekundě. S přerušením inkrementujeme proměnnou *tickfurt* a dekrementujeme proměnnou *cas*, není-li nulová.

Čítač 1 nevyvolává přerušení. Je využit pro generování přenosové rychlosti sériové linky.

Externí 0

Tímto přerušením spustíme nebo zastavíme manipulátor. Negujeme zde proměnnou *napln_enable*.

Externí 1

Toto přerušení je vyvoláno po aktivaci nějakého koncového čidla. V obsluze zjistíme, které čidlo přerušení vyvolalo. Pak zajistíme takový pohyb, aby manipulátor odjel z čidla, a nastavíme proměnnou *pozice* ve struktuře motoru na pozici, odpovídající konkrétnímu čidlu.

Sériová linka

Po vyvolání přerušení od sériové linky zjistíme, zda je přerušení vyvoláno kvůli tomu, že byl odeslán poslední bit z registru SBUF (byl odeslán byt), nebo byl přijat poslední bit přijímaného bytu. Bylo-li vyvoláno přerušení kvůli odeslání bytu, hned se z podprogramu přerušení vrátíme do hlavního programu.

Pokud přišel po sériové lince byt, zjišťuje se, zda je proměnná „délka“ nulová, aby nedošlo k přepsání dosud nevybrané zprávy novou informací. Dále je nutné rozlišovat mezi začátkem přenosu, koncem přenosu, příznakem *escape* a vlastními daty. Začátek přenosu je definován

znakem 0xA7, konec znakem 0xA8. Příznaku *escape* odpovídá znak 0x27. Pokud se v přenášených datech objeví znak 0xA7, 0xA8 nebo 0x27, musí tento znak předcházet příznak *escape*. Zpráva pak může vypadat například takto:

0xA7, 0x15, 0xD8, 0x27, 0xA7, 0xFF, 0xA8

Pokud bychom vynechali příznak *escape* (0x27), byl by obsahem zprávy pouze znak 0xFF. Přijaté znaky se ukládají do pole *zpráva*. Po přijetí konce přenosu, uloží obslužný program přerušení uloží do proměnné *délka* počet přijatých znaků.

8.1.4 Popis dalších funkcí v souboru main.c

_cekej: Jde o časovou smyčku, jíž se využívá nelze-li užít kvůli zakázanému přerušení časování od časovače 0. Vstupní parametr udává čas v µs.V této funkci je použita direktiva *#pragma asm* a *#pragma endasm*. Tyto direktivy umožňují použít v programu kód napsaný v assembleru.

toc: Tato funkce zajišťuje zápis hodnoty do registru, který řídí budiče cívek motoru. Zapsaná hodnota je závislá na proměnných *smer*, *zapnuty* a *krok*. Vstupním parametrem je číslo motoru, jímž chceme otáčet.

pocitej: Počítá algoritmus současného dotočení motorů. Uloží-li tato funkce do proměnné *toc* hodntu 1, může motor v následujícím cyklu udělat krok. Krok se neprovede, uloží-li se do *toc* hodnota 0. Výpočet se provádí v cyklu pro každý motor.

automat: Po vstupu do této funkce se zjišťuje, zda již uběhl interval pro vykonání dalšího kroku motoru. Pak se plní proměnná *cas*. Rozjezd a dojezd motoru se provádí tak, že se index (proměnná *cas_index*) ukazující do pole *krokcas* posouvá. Tím se mění intervaly mezi kroky. Dále se v této funkci ošetřují různé možné situace: např. dojezd všech motorů do zadané pozice, sepnutí čidla v čelistech úchopného mechanismu nebo aktivace některého indukčního čidla při rotaci.

zablikej: Rozsvítí a po uplynutí 250 ms zhasne diodu (LED3). V čekací smyčce kontroluje, zda přišel příkaz po sériové lince

napln: Zajišťuje výběr souřadnic, nebo řídícího příkazu z EEPROM paměti. Je zde zakázáno přerušení. Vybraný řídící příkaz se okamžitě provede.

nulovapoloha: Po zavolání této funkce začne kalibrace manipulátoru. Svírá-li manipulátor cokoli v úchopné hlavici, kalibrace se neprovede. Při kalibraci manipulátor maximálně zvedne rameno, dotočí se proti směru hodinových ručiček až na koncový doraz a rozevře kleště. Protože jsou v této poloze všechny strojní souřadnice nulové, nazýváme ji nulová poloha manipulátoru.

manual: Otočí o jeden krok zadaným motorem. Vstupními parametry jsou směr rotace a číslo motoru.

8.2 Soubor i2c_sw.c

V souboru *i2c_sw.c* jsou uloženy funkce pro práci se sběrnicí I^2C. Linka SCL je zde přiřazena vývodu mikroprocesoru P3.4, linka SDA pak vývodu P3.5. Protože pracujeme s pamětí I^2C typu 24C16, musíme bázovou adresu nastavit na 0xA0. Do proměnné „_i2c_error" ukládáme případné chyby v komunikaci.

Popis nejdůležitějších funkcí

I2CSendAddr: Vytvoří na sběrnici podmínku startu a přenese adresu zařízení spolu s informací, zda budeme z paměti číst, nebo do ní zapisovat. Vstupními parametry jsou adresa a směr přenosu.

I2CSendByte: Posílá po I^2C byte, který je funkci předán vstupním parametrem.

_I2CGetByte: Čte byt z I^2C. Je-li vstupní parametr 1, předáme po I^2C informaci, že se bude číst poslední byte,jinak je vstupní parametr nulový.

I2CSendStop: Ukončuje přenos.

I2CReset: Nuluje komunikační obvody EEPROM paměti. Pro navázání komunikace se opakovaně čte první řádek v pohybové tabulce a cyklus se ukončí, shodují-li se přečtené hodnoty se zadanými. Programově je ošetřeno, aby průchod cyklem mohl nastat nanejvýš 255-krát. Pokud bychom zaměnili paměťový modul v patici jiným, neobsahoval by první řádek tabulky správné hodnoty a program by se zacyklil. Takto program opustí cyklus po 255tém průchodu a hodnoty lze do prvního řádku zapsat.

8.3 Soubor i2czapis.c

V souboru *i2czapis.c* je uložena pouze funkce pro zapsání řádku do pohybové tabulky. Název této funkce je *zapis_radek*. Po zapsání řádku proběhne jeho zpětné čtení pro kontrolu správnosti záznamu. Vstupními parametry jsou index řádku v pohybové tabulce a ukazatel na pole, ve němž jsou připraveny hodnoty k zapsání. Funkce vrátí hodnotu 1 v případě správného zápisu. Pokud se řádek nepodařilo do paměti zapsat, vrátí funkce hodnotu 0.

8.4 Soubor seriovka.c

V tomto souboru jsou uloženy všechny potřebné funkce týkající se vysílání a přijímání dat po sériové lince (RS232 nebo RS485) včetně funkce podprogramu přerušení. Zde je také definováno globální pole, do něhož ukládáme došlé znaky. Toto pole se ukládá do rozšířené vnitřní paměti dat pomocí direktivy *idata*: *unsigned char idata zprava[DELKA];*

Popis nejdůležitějších funkcí

serial:	Spouští se po vyvolání přerušení od sériové linky. Funkce byla popsána v kapitole Obsluha přerušení.
CRCni:	Počítá osmibitový kontrolní součet.
vysli:	Naplní odesílací registr SBUF a počká, dokud není byte odeslán.
vyslipaket:	Sestavuje odesílání paketu. Funkce odešle data z pole *zprava[]*. Vstupními parametry jsou posílaný příkaz a délka dat v paketu.
paketok:	Kontroluje, zda má přijatý paket v pořádku adresu, *magic* a kontrolní součet. Funkce vrací hodnotu 1, není-li paket v pořádku.
vyber:	Podle čísla příkazu v přijaté zprávě pokračuje program na příslušném procesu (*case*) a vykoná příkaz v něm zapsaný.

8.5 Soubor *tridic.c*

V tomto souboru jsou uloženy funkce potřebné pro obsluhu třídiče. Je zde definována globální proměnná *stav_motoru*, jejíž poslední čtyři bity zaznamenávají stav všech pneumatických motorů (1-vysunutý, 0-zasunutý).

Význam bitů v proměnné *stav_motoru*:
0 - není přiřazen
1 - motor obsluhující zásobník
2 - motor obsluhující třídění na druhou pozici
3 - motor obsluhující zarážku

Použité funkce:

tridic

Po zavolání této funkce připraví systém pneumatické motory do výchozích pozic. Vždy při vysunutí nebo zasunutí se program dotazuje snímačů koncových poloh pístů pneumatických motorů. Pokud se při změně polohy píst nepřesune do pěti sekund, funkce vrátí hodnotu 3 (viz. výše).

Tato funkce nemá žádný vstupní parametr.

Význam návratové hodnoty: 0 – v zásobníku není materiál
1 – materiál je kovový
2 – materiál je nekovový
3 – chyba hardwaru (např. není přiveden tlakový vzduch)

Pneumotor

Tato funkce zapisuje hodnotu *stav_motoru* do registrů, které ovládají budiče elektromagnetických pneumatických ventilů.

Vstupními parametry funkce jsou číslo ovládaného motoru a stav (vysunout/zasunout).

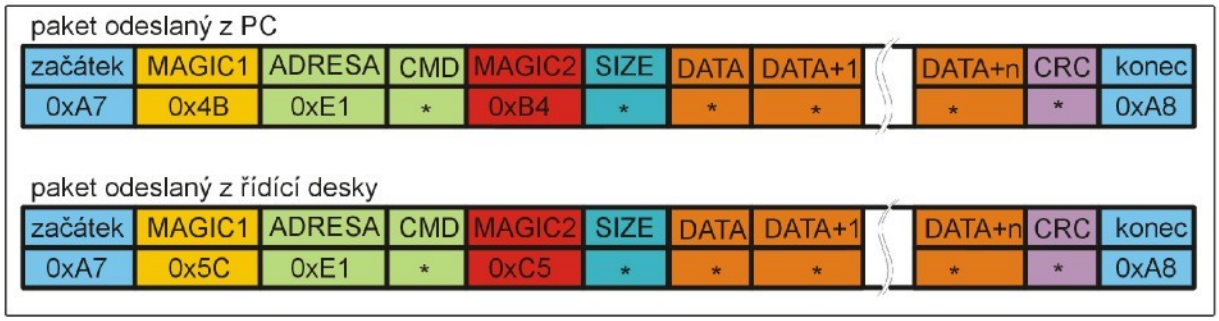
9 Ovládací program

Ovládací program pro PC jsem napsal v prostředí C++ Builder verze 5.0. Protože standardní instalace neobsahuje knihovnu pro práci se sériovou linkou, doplnil jsem ji o volně šiřitelnou knihovnu Cport (verze 2.63) od autorů Dejana Crnila a Paula Dolanda. Tuto knihovnu a její případné nové verze je možno získat na internetu (http://www2.arnes.si/~sopecrni).

Knihovna Cport obsahuje funkce pro otevření a zavření sériového portu, nastavení přenosové rychlosti, čtení a zápis dat a mnoho dalších. Autoři této knihovny mysleli i na tvorbu a příjem paketů, avšak tuto podporu jsem nemohl použít, protože při příjmu jednoho znaku není možné tento znak okamžitě převzít a použít ho do výpočtu kontrolního součtu CRC. Dále je zde předpoklad, že paket začíná a končí řetězcem. Paket, který jsem použil pro komunikaci s řídící deskou, je však odstartován a zakončen znakem (0xA7 a 0xA8), a proto je nutné vždy testovat příznak *escape*. Funkce vhodné pro práci s paketem jsem tedy musel vytvořit sám. Když byly napsané a odzkoušené, zjistil jsem, že komunikace funguje pouze pod operačním systémem Windows 2000, ale už ne pod systémem Windows 98. Chyba byla v použití funkce *TransmitChar*. Tato funkce má po sériové lince odeslat znak (byte). Komunikace probíhá bez problémů, pokud tuto funkci nevoláme rychle za sebou. V opačném případě ohlásí systém Windows 98 chybu komunikace. Nakonec jsem problém vyřešil tak, že jsem celou zprávu sestavil do pole, které jsem odeslal pomocí funkce *WriteStr* (zapiš řetězec).

Program obsahuje všechny příkazy, kterými je možno řídící desku ovládat a získávat od ní potřebné informace. Jednotlivé příkazy jsou vysvětleny v nápovědě programu. Po spuštění se program snaží načíst celou paměť EEPROM z řídící desky. Podaří-li se mu to, uloží data do tabulky, jinak ohlásí chybu. Tabulku lze kdykoli obnovit. Komunikace PC s řídící deskou probíhá rychlostí 57600 b/s. Posloupnost a význam bytů v paketu je naznačena na obrázku č.18.

Pokud se v paketu vyskytne 0xA7, 0xA8 nebo 0x27, vkládá se před tento byte znak 0x27. DATA je první datový byte v paketu. Následující datové byty jsou označeny jako DATA+1, DATA+2, …., DATA+n. Těmito byty jsou předávány parametry příkazů a vrácené hodnoty. Hodnota SIZE určuje počet datových bytů v paketu. Hodnotou CMD zadáváme číslo příkazu.



Obr.18 Paket v sériové komunikaci, LP.

9.1 Příkazy posílané po sériové lince

příkaz 0 (číslo verze) CMD=0

Je dotaz na číslo verze programu řídící desky. Odešle příkaz bez dalších parametrů (SIZE=0). Řídící deska vrátí dva datové byty. Významem prvního datového bytu jsou jednotky, druhého jsou setiny. Příklad: deska odpoví DATA=1 a DATA+1=1, číslo verze tedy bude 1.01.

příkaz 1 (hodnota řádku v EEPROM)

Je dotaz na hodnoty zapsané na konkrétním řádku v paměti EEPROM. V bytu DATA se odesílá číslo řádku. Řídící deska vrátí osm bytů, z nichž ovládací program složí čtyři požadované šestnáctibitové hodnoty.

příkaz 2 (stop)

Je příkaz pro okamžité zastavení manipulátoru. Řídící deska po vykonání příkazu odpoví. Při tomto příkazu se nepředávají žádné parametry.

příkaz 3 (kalibruj)

Po přijetí tohoto příkazu provede manipulátor kalibraci, a pak pokračuje v činnosti, od níž byl odvolán. Při tomto příkazu se nepředávají žádné parametry.

příkaz 4 (dojezd na pozici)

Tímto příkazem lze manipulátor přesunout na zadané (strojní) souřadnice. Souřadnice desce předáváme v osmi datových bytech (DATA až DATA+7). Řídící deska po vykonání příkazu odpoví (bez parametrů).

příkaz 5 (start)

Po přijetí tohoto příkazu je vybrán další řádek z paměti EEPROM a povolí se automatický pohyb. Manipulátor se začne pohybovat po zadané trajektorii. Řídící deska po vykonání příkazu odpoví. Při tomto příkazu se nepředávají žádné parametry.

příkaz 6 (max SP)

Tento příkaz není v ovládacím programu zahrnut. Pro uživatele systému není důležitý. Po přijetí příkazu vrátí řídící deska maximální výšku zásobníku v paměti RAM. Využil jsem ho při kontrole přetečení zásobníku.

příkaz 7 (zapiš řádek do EEPROM)

Příkaz pro zapsání hodnot souřadnic nebo řídícího příkazu do EEPROM paměti. Byt DATA je rezervován pro práci s pamětí, která je schopna pojmout větší počet řádků než 256. V bytu DATA+1 zadáváme řádek tabulky a v dalších osmi bytech předáváme zapisované hodnoty. Řídící deska po zapsání hodnot odpoví (bez parametrů).

příkazy 8 a 9 nejsou implementovány.

příkaz 10 (pootočení motorem)

Příkaz slouží pro otáčení konkrétním motorem o zadaný počet kroků. V parametru DATA je číslo motoru který chceme ovládat. Hodnota bytu DATA+1 předává směr otáčení. Byty DATA+2 a DATA+3 tvoří šestnáctibitový údaj o počtu kroků. Řídící deska po vykonání příkazu odpoví.

příkaz 11 (dotaz na aktuální pozici)

Tento příkaz nemá vstupní parametry. Řídící deska po přijetí příkazu odešle v osmi bytech (DATA až DATA+7) aktuální hodnoty strojních souřadnic.

příkaz 12 (zapnutí/vypnutí funkce pro současný dojezd motorů)

Zařadí nebo vyřadí funkci pro současný dojezd motorů. Řídící deska po vykonání příkazu odpoví (bez parametrů).

příkaz 13 (přesun indexu řádku v pohybové tabulce)

Přesune index řádku v tabulce. Byt DATA je rezervován pro práci s pamětí, která je schopna pojmout větší počet řádků než 256. V bytu DATA+1 zadáváme, na který řádek tabulky chceme index přesunout.

příkaz 14 (setřiď materiál)

Po přijetí příkazu je obsloužen třídič. V tomto příkazu neposíláme řídící desce žádné parametry. Po obsloužení třídiče řídící deska vrátí v parametru DATA výsledek třídění.
0 – není materiál, 1 – materiál je nekovový, 2 - materiál je kovový, 3 – chyba hardwaru.

příkaz 15 (dotaz na aktuální index řádku v pohybové tabulce)

Řídící deska po přijetí tohoto příkazu odešle v parametru DATA+1 aktuální řádek v pohybové tabulce. Parametr DATA je rezervován pro práci s větší pamětí.

Pokud řídící deska přijme příkaz s číslem jiným, než jaké má předdefinované, odpoví paketem s číslem příkazu 0xFF.

10 Závěr

Zadaný řídící systém jsem kompletně realizoval na jedné desce plošných spojů. Navržené schéma zapojení logických i výkonových obvodů je plně funkční.

Programové vybavení, které jsem při realizaci vyvinul, je odladěné a také plně funkční. V přiložených zdrojových kódech jsou vyřešeny velmi často potřebné části programů jako např. obsluha jednotlivých přerušení, nastavení časovače a sériové linky komunikace po I^2C sběrnici a další.

Uvedený přepočet strojních souřadnic na válcové jsem použil v ovládacím programu. Tam jsou jak strojní, tak válcové souřadnice zapsány do tabulky. Hodnoty válcových souřadnic jsou pouze informativní. Prakticky jsem je zkoušel měřit a odchylky od teoretických hodnot jsou v toleranci asi ± 10mm, což je způsobeno zadáním nepřesných rozměrů manipulátoru.

Manipulátor může být vizualizován pomocí sériového rozhraní, které předává příslušnému programu potřebná data. Jeden ze způsobů vizualizace by mohlo být zobrazení manipulátoru na monitoru se stejnými strojními souřadnicemi jako u fyzického modelu v pohybu. Počítač by se dotazoval řídící desky na aktuální souřadnice a podle nich by mohl zobrazovat virtuální model.

Řídící deska je univerzální a snadno použitelná i pro jiné aplikace, protože při návrhu byl brán ohled na její začlenění do koncepce školní laboratoře. Je tedy možné ji snadno využít i k výukovým účelům.

Na přiloženém CD jsou mj. videa manipulátoru v naprosto odlišných úlohách. Tím jsem chtěl naznačit široké možnosti nasazení systému díky schopnosti jednoduše měnit pohybové vektory a řídících příkazy.

Při zpětném ohlédnutí si uvědomuji, že tato práce byla pro mě velkým přínosem. Prohloubil jsem si znalosti především o mikroprocesorech řady 51 a sběrnici I^2C, o které jsem se již delší dobu zajímal a získal jsem praktické zkušenosti v oblasti mikroprocesorové techniky. Přál bych si, aby tato práce byla přínosem i dalším studentům při jejich hledání.

Použitá literatura:

[1] Skalický, P. – Mikroprocesory řady 8051.

[2] Janeček, J. – Projektování mikropočítačových systémů.

[3] Novák, P. – Krokové motory a jejich využití, Automa, 0/1994.

[4] Krokové motory (Step motors), firemní literatura AEG.

[5] Firemní literatura fy. FESTO (pneumatické servopohony a ventily).

[6] Philips Semiconductors – The I^2C-BUS SPECIFICATION, version 2.1, 2000.

[7] Atmel - AT24C01A/02/04/08/16, 2000, (http://www.atmel.com).

[8] http://www.regulace.cz/cz_vyroba-krokove_motory.htm - Krokové reverzační pohonné jednotky SMR, leden 2003.

[9] Maule, P. – Krokové motory, Amatérské rádio 8/1992.

[10] Atmel - AT89C52 8-Bit MCU with 8K Bytes Flash, 1998, (http://www.atmel.com).

Prohlášení o užití diplomové práce a poskytnutí licence třetím osobám

Autor, Lukáš Petrus, si je vědom toho, že jeho diplomová práce (dále jen dílo) je školním dílem ve smyslu § 60 zákona č.121/2000 (autorský zákon) (dále jen zákon), to mj. znamená, že Technická univerzita Liberec (dále jen TUL) je oprávněna k nevýdělečnému užití díla k vnitřní potřebě TUL ve shodě s odstavcem (3) § 35 zákona.

TUL smí dílo výdělečně užít či poskytnout licenci k užití třetí osobě pouze na základě písemné smlouvy s autorem díla, která upraví zejména podíl autora na zisku z užití či prodeje nebo jiný způsob stanovení odměny pro autora.

Užije-li autor díla výdělečně, nebo poskytne-li třetí osobě za úplatu licenci k užití díla, uvědomí o tom bez zbytečného prodlení TUL a nabídne TUL doporučeným dopisem úhradu nákladů vynaložených TUL na vytvoření díla ve shodě s ustanoveními odstavce (3) § 60 zákona. Neodpoví-li TUL na takovouto nabídku do jednoho měsíce od doručení nabídky, je autor oprávněn považovat své závazky k TUL vyplývající z nákladů TUL na vytvoření díla za vyrovnané.

V Liberci 25. 5. 2003

.....

Lukáš Petrus

Příloha 1 – zdrojový kód v souboru main.c

```
//Lukas Petrus
//Diplomova prace 2003
//main.c

//motor[0] - zvedak
//motor[1] - rotace
//motor[2] - rameno
//motor[3] - kleste


#pragma ObjectExtend db
#pragma Optimize (6, Speed)
#pragma NoIntPromote

#include <reg51.h>

#define VERZE           101      // verze 1.01
#define MOTORU          4        //kolik ma robot motoru
#define ROZJEZD         20
#define AKCELERACE      1

#define ROTACEMAX       3467
#define KLESTEMAX       670
#define RAMENOMAX       4082
#define ZVEDAKMAX       1503
#define IC1R            1077
#define IC2R            1613
#define IC3R            1896
#define IC4R            2420
#define IC1L            1144
#define IC2L            1691
#define IC3L            1954
#define IC4L            2486

#define XTAL            22118400 //11059200 //
#define TIMER           (0x10000-XTAL/12/1000)

#if XTAL==11059200
  #define BAUDDIV       0xFF
#else
  #define BAUDDIV       0xFE //0xE8//-pro 4800//0xFE-pro 57600
#endif

typedef unsigned long ulong;
typedef unsigned int  uint;
typedef unsigned char uchar;

sbit  Dioda=P1^7;
sbit  P11  =P1^1;
sbit  P32  =P3^2;
bit   automat_enable;
bit   napln_enable;
bit   kalibruje=0;
bit   pocitani_enable=1;
bit   kalibr=0;

xdata uchar adresa8000 _at_ 0x8000;    //uchar na kterem jsou motory 1a2
xdata uchar adresa8001 _at_ 0x8001;    //uchar na kterem jsou motory 3a4
xdata uchar adresa8002 _at_ 0x8002;    //uchar na kterem jsou civky ventilu
```

```
xdata uchar adresa8003 _at_ 0x8003;    //uchar na kterem je manualni
ovladani
xdata uchar adresa8004 _at_ 0x8004;    //uchar na kterem jsou cidla od
tridice

uint  tickfurt;
uint  brzdi=0;
uint  kmax;                //nejvetsi pocet kroku pri jednom vektoru pohybu
uint  poradi;              //poradi kroku (v automat)

uint  index_tab=0;
uint  index_tab_zaloha=0;
uchar akcelerace=AKCELERACE;
uchar kolik=AKCELERACE;
uchar cas=0;
uchar cas_index=5;
uchar maxSP=0;

uchar code
krokcas[ROZJEZD]={2,2,2,3,3,3,3,4,4,4,5,5,5,6,6,6,7,7,7,8};//,9,10};
                            //11,12,13,14,15};//16,17,18,19,20};
                            //rozjezdove casy mezi kroky (v ms)
uchar code kombcivek[4]    ={5,6,10,9}; //pro 4-takt
//uchar code kombcivek[8]={10,8,9,1,5,4,6,2}; //pro 8-takt

typedef struct
{
  uchar zapnuty;           //povoleni motoru
  uint  pozice;            //aktualni pozice od nul polohy
  uint  stoppozice;        //kam ma dojet-pozice od nul polohy
  uint  kroku;             //kolik vykona kroku v aktualnim vektoru pohybu
  uint  m_last;            //posledni hodnota vypoctu algoritmu
  uchar krok;              //index do pole kombinaci na civkach
  char  smer;              //1=tam, -1=zpet, 0=nikam
  uchar toc;               //je-li 1, bude se pri dalsi akci motor tocit
} MOTOR;
MOTOR data motor[MOTORU];



void _cekej(uint kolik)
{
//  while(kolik !=0) {kolik--;}
    #pragma asm
              mov     A,R7
              clr     C
              rrc     A
              mov     R7,A
              djnz    R7,$
    hopla:    mov     R7,#080h
              djnz    R7,$
              djnz    R6,hopla
    #pragma endasm
    kolik++;
}


void napln(void);
void toc  (uchar);
bit automat(void);
void nulovapoloha(void);
```

```
#if  PREPISEEPROM==1
     #include "tabpohyb.c"
#endif

#include "initcpu.c"
#include "i2c_sw.c"
#include "I2Czapis.c"
#include "tridic.c"
#include "seriovka.c"



void citac0 (void) interrupt 1 using 2
{
  TH0=TIMER/256;
  TL0=TIMER;
  tickfurt++;
  if(SP>maxSP) maxSP=SP;
  if (cas) cas--;
}



void  externi0 (void) interrupt 0 using 2
{
  uchar i;

  if (napln_enable==0) napln_enable=1;
  else
  {     for (i=0; i<MOTORU; i++) motor[i].stoppozice=motor[i].pozice;
       napln_enable=0;
       brzdi=0;
  }
  while (P32==0) {}
}



void  externi1 (void) interrupt 2 using 2
{
  uchar a;

  if(~adresa8000)
  {
    a=~adresa8000;

    if((a&1)==0)
    {
      motor[0].smer=-1;
      motor[0].zapnuty=1;
      while((~adresa8000&1)==0)
      {
        toc(0);
        tickfurt=0;
        while(tickfurt++<3000) {}
      }
      motor[0].pozice=ZVEDAKMAX;
      motor[0].stoppozice=ZVEDAKMAX;
      return;
    }
```

```
if((a&2)==0)
{
  motor[0].smer=1;
  motor[0].zapnuty=1;

  while((~adresa8000&2)==0)
  {
     toc(0);
     tickfurt=0;
     while(tickfurt++<3000) {}
  }
  motor[0].pozice=0;
  motor[0].stoppozice=0;
  return;
}

if((a&4)==0)
{
  motor[2].smer=-1;
  motor[2].zapnuty=1;
  while((~adresa8000&4)==0)
  {
     toc(2);
     tickfurt=0;
     while(tickfurt++<3000) {}
  }
  motor[2].pozice=RAMENOMAX;
  motor[2].stoppozice=RAMENOMAX;
  return;
}

if((a&8)==0)
{
  motor[2].smer=1;
  motor[2].zapnuty=1;
  while((~adresa8000&8)==0)
  {
     toc(2);
     tickfurt=0;
     while(tickfurt++<3000) {}
  }
  motor[2].pozice=0;
  motor[2].stoppozice=0;
  return;
}


if((a&0x10)==0)
{
  motor[3].smer=-1;
  motor[3].zapnuty=1;
  while((~adresa8000&0x10)==0)
  {
     toc(3);
     tickfurt=0;
     while(tickfurt++<3000) {}
  }
  motor[3].pozice=KLESTEMAX;
  motor[3].stoppozice=KLESTEMAX;
  return;
```

```
    }

    if((a&0x20)==0)
    {
      motor[3].smer=1;
      motor[3].zapnuty=1;
      while((~adresa8000&0x20)==0)
      {
         toc(3);
         tickfurt=0;
         while(tickfurt++<3000) {}
      }
      motor[3].pozice=0;
      motor[3].stoppozice=0;
      return;
    }

  }

  if(adresa8001)
  {
    a=adresa8001;

    if((a&1)==0)
    {
      motor[1].smer=1;
      motor[1].zapnuty=1;
      while((adresa8001&1)==0)
      {
         toc(1);
         tickfurt=0;
         while(tickfurt++<3000) {}
      }
      motor[1].pozice=0;
      motor[1].stoppozice=0;
      return;
    }

    if((a&2)==0)
    {
      motor[1].smer=-1;
      motor[1].zapnuty=1;
      while((adresa8001&2)==0)
      {
         toc(1);
         tickfurt=0;
         while(tickfurt++<3000) {}
      }
      motor[1].pozice=ROTACEMAX;
      motor[1].stoppozice=ROTACEMAX;
      return;
    }

  }
}


void toc(uchar a)
{
  uchar civky;
```

```
  motor[a].pozice+=motor[a].smer;
  motor[a].krok  +=motor[a].smer;
  motor[a].krok  &=0x03;           // pro 4-takt,  &=0x07 pro 8-takt

  if(motor[a].zapnuty) civky=kombcivek[motor[a].krok];
  else civky=0;

  if(a==0)
  {
    if(motor[1].zapnuty) adresa8000=(civky | (kombcivek[motor[1].krok]<<4));
    else                 adresa8000= civky;
  }

  if(a==1)
  {
    if(motor[0].zapnuty) adresa8000=( (civky<<4) |
kombcivek[motor[0].krok]);
    else adresa8000=civky<<4;
  }

  if(a==2)
  {
    if(motor[3].zapnuty) adresa8001=(civky | (kombcivek[motor[3].krok]<<4));
    else                 adresa8001= civky;
  }

  if(a==3)
  {
    if(motor[2].zapnuty) adresa8001=( (civky<<4) |
kombcivek[motor[2].krok]);
    else                 adresa8001=  (civky<<4);
  }
}



void pocitej(void)      //algoritmus na soucasny dojezd motoru
{
   uchar i;
   ulong m;

   poradi++;

   for(i=0;i<MOTORU;i++)
   {
      //m=poradi;
      //m*=motor[i].kroku;
      //m/=kmax;
      m=poradi*motor[i].kroku;
      m/=kmax;
      if(m>motor[i].m_last)   motor[i].toc=1;
      else motor[i].toc=0;
      motor[i].m_last=m;
   }
}



bit automat(void)
{
  uchar i, vracim=0;
```

```
if (cas==0)
{
   cas=krokcas[cas_index];

   //-------rozjezd a brzdeni---------------------
   if (brzdi) brzdi--;

  if (kolik==0)
   {
     kolik=akcelerace;
     if (cas_index && brzdi)                 cas_index--;
     if ((cas_index<ROZJEZD-1)&&(brzdi==0)) cas_index++;
   }
   else kolik--;
   //-----------------------------------------------
   if(kalibruje) cas=krokcas[5];

   for(i=0;i<MOTORU;i++)
   {
     if (motor[i].pozice==motor[i].stoppozice)
     {
       vracim++;
       motor[i].zapnuty=0;//vypni civky !!!
       motor[i].smer=0;
       toc(i);
       continue;
     }

     if (motor[i].toc || pocitani_enable==0)   toc(i);

     //--------jehla na celistech--------------------------
     if (adresa8000&0x40)
       if (motor[3].smer==1)
           motor[3].stoppozice=motor[3].pozice;
     //--------osetreni kalibrace na indukcnich cidlech---

     if (i==1)
     {
      if (~adresa8001&0xFC)
      {
       if (kalibr==0)
       {
         kalibr=1;
         //zprava[DATA]   = motor[1].pozice>>8;
         //zprava[DATA+1] = motor[1].pozice;
         //vyslipaket(20,2);

         if (motor[1].smer==1)
         {
           if (~adresa8001&0x10)  motor[1].pozice=IC1R;
           if (~adresa8001&0x20)  motor[1].pozice=IC2R;
           if (~adresa8001&0x40)  motor[1].pozice=IC3R;
           if (~adresa8001&0x80)  motor[1].pozice=IC4R;
         }
         if (motor[1].smer==-1)
         {
           if (~adresa8001&0x10)  motor[1].pozice=IC1L;
           if (~adresa8001&0x20)  motor[1].pozice=IC2L;
           if (~adresa8001&0x40)  motor[1].pozice=IC3L;
           if (~adresa8001&0x80)  motor[1].pozice=IC4L;
```

```
            }
            if (motor[1].stoppozice>motor[1].pozice) motor[1].smer=1;
            if (motor[1].stoppozice<motor[1].pozice) motor[1].smer=-1;

          }
         }
         else kalibr=0;
        }
        //-------------------------------------------------

      }

      if (vracim==MOTORU) return 1;
      else                return 0;
   }
   else return 0;
}


void zablikej(void)
{
  uchar i;
  for(i=0;i<2;i++)
  {
    tickfurt=0;
    while(tickfurt<250)
    {
     if (delka)               //osetreni seriove linky
     {
         vyber();             //vybere postu (pole naplnene seriovkou)
         delka=0;
     }

    }
    if(Dioda==0) Dioda=1;
    else         Dioda=0;
  }
}


void napln(void)
{
 uchar i, stranka, radek;

 brzdi=1;
 EA=0;
 stranka=index_tab/32;
 stranka<<=1;
 radek=index_tab%32;
 radek*=8;

 I2CSendAddr(I2CADDR+stranka,WRITE);  //!!!musi tu byt
 //I2CSendByte(0);           //pro 24C64
 I2CSendByte(radek);
 I2CSendAddr(I2CADDR+stranka,READ);

 for(i=0;i<MOTORU;i++)
 {
    motor[i].stoppozice=I2CGetByte();
    motor[i].stoppozice<<=8;
    if(i==MOTORU-1) motor[i].stoppozice|=I2CGetLastByte();
```

```
    else            motor[i].stoppozice|=I2CGetByte();

    if(i==0 && motor[0].stoppozice==0xFFFF)      //PRESUN INDEXU NA POZICI
    {
      index_tab=I2CGetByte();                   //dalsi uint je radek, kam
to
                                                    //ma skocit
      index_tab<<=8;
      index_tab|=I2CGetLastByte();
      I2CSendStop();
      EA=1;
      motor[0].stoppozice=motor[0].pozice;
      return;
    }

    if(i==0 && motor[0].stoppozice==0xFFFE)      //KALIBRUJ
    {
      //I2CGetByte();
      i=I2CGetLastByte();                       //ukonceni komunikace
      I2CSendStop();
      EA=1;
      motor[0].stoppozice=motor[0].pozice;
      nulovapoloha();
      index_tab++;
      return;
    }

    if(i==0 && motor[0].stoppozice==0xFFFD)      //obsluz tridic
    {
      i=I2CGetLastByte();                       //ukonceni komunikace
      I2CSendStop();
      EA=1;
      motor[0].stoppozice=motor[0].pozice;

      switch (tridic())
      {
        case 0: //neni material
                break;
        case 1: //zaton je nenov
                index_tab_zaloha=index_tab+1;
                index_tab=75;
                break;
        case 2: //zeton je kov
                index_tab_zaloha=index_tab+1;
                index_tab=50;
                break;
        case 3: //tridic porucha
                //zapiskej();
                break;

        default: break;
      }
      return;
    }
    if(i==0 && motor[0].stoppozice==0xFFFC)      //vrat index_tab
    {
      i=I2CGetLastByte();                       //ukonceni komunikace
      I2CSendStop();
      EA=1;
      motor[0].stoppozice=motor[0].pozice;
      index_tab=index_tab_zaloha;
```

```
      return;
    }

    motor[i].smer=0;
    if(motor[i].pozice < motor[i].stoppozice)
    {
      motor[i].zapnuty=1;
      motor[i].smer=1;
      motor[i].kroku=motor[i].stoppozice-motor[i].pozice;
      if(motor[i].kroku>brzdi) brzdi=motor[i].kroku;
    }

    if(motor[i].pozice > motor[i].stoppozice)
    {
      motor[i].zapnuty=1;
      motor[i].smer=-1;
      motor[i].kroku=motor[i].pozice-motor[i].stoppozice;
      if(motor[i].kroku>brzdi) brzdi=motor[i].kroku;
    }
 }
 I2CSendStop();
 EA=1;
 index_tab++;
 //if (index_tab>=RADKU) index_tab=0;

 kmax=brzdi;
 cas_index=ROZJEZD-1;
 brzdi-=ROZJEZD*akcelerace;
 poradi=0;
}



void nulovapoloha(void)
{
  uchar i;
  for(i=0;i<MOTORU;i++) motor[i].stoppozice=motor[i].pozice;

  //!ZKONTROLUJ ZDA JE NECO V KLESTICH
  //POKUD ANO, PREPNI NA MANUAL A ZAPIS TO NA DISPLAY
  if(adresa8000&0x40)                  //jehla
  {
     napln_enable=0;
     brzdi=0;
     return;
  }
  //---------------
  kalibruje=1;
  for(i=0; ;)
  {
    motor[i].pozice=0xEEFF;
    motor[i].stoppozice=0;
    motor[i].smer=-1;
    motor[i].zapnuty=1;

    while(automat()==0)
    {
      motor[i].toc=1;
      tickfurt=0;
      while(tickfurt<3) {}
    }
```

```
    if(i==3) break;
    if(i==1) i=3;
    if(i==2) i=1;
    if(i==0) i=2;
  }
  kalibruje=0;
  return;
}


void manual(uchar ktery, char smer)
{
    motor[ktery].smer=smer;
    motor[ktery].zapnuty=1;
    toc(ktery);
    motor[ktery].stoppozice=motor[ktery].pozice;
}



void main(void)
{

  init_cpu();
  DTR=1;                //Nastav DTR na prijem
  Dioda=0;
  EA=1;
  I2CReset();
  zablikej();
  P11=0;                //zapne registry!!!
  adresa8002=0;         //zasune pneumotory na tridici

  zablikej();
  automat_enable=1;
  nulovapoloha();
  tickfurt=0;
  napln_enable=0;
  automat_enable=1;

  napln();

  while(1)
  {
     if (delka)               //osetreni seriove linky
     {
         vyber();             //vybere postu (pole naplnene seriovkou)
         delka=0;
     }
    //---------------------------- manual provoz
     if(adresa8003)
     {
       if(cas==0)
       {
         cas=krokcas[7];
         if(adresa8003&0x01)    manual(0,1);
         if(adresa8003&0x02)    manual(0,-1);
         if(adresa8003&0x04)    manual(1,1);
         if(adresa8003&0x08)    manual(1,-1);
         if(adresa8003&0x10)    manual(2,1);
         if(adresa8003&0x20)    manual(2,-1);
         if(adresa8003&0x40)    manual(3,1);
```

```
        if(adresa8003&0x80)    manual(3,-1);
       }
       continue;
     }
   //---------------------------- automaticky provoz
  if (automat_enable)
  {
       if(pocitani_enable)   pocitej();  //pocita algoritmus soucasneho
                                         //dojeti motoru
       if(automat())
       {
         if(napln_enable) napln(); // v teto fci muze volat  tridic(void)
         zablikej();               // !pokud je volana fce tridic(), presune
se
                                   // index_tab na 50, nebo 75
       }
     }
  }
}
```

Příloha 2 – zdrojový kód v souboru i2c_sw.c

```
//Lukas Petrus
//Diplomova prace 2003
//i2c_sw.c

#ifndef READ
    #define READ   1
#endif

#ifndef WRITE
    #define WRITE  0
#endif

#ifndef I2CADDR                     //adresa zarizeni
    #define I2CADDR 0xA0
#endif

#ifndef SCL
    sbit SCL  =P3^4;
#endif

#ifndef SDA
    sbit SDA  =P3^5;
#endif

unsigned char _i2c_error;

void          _I2CWait(void);
void          _I2CSCLHigh(void);
void           I2CSendAddr(unsigned char addr, unsigned char rd);
void           I2CSendByte(unsigned char bt);
unsigned char _I2CGetByte(unsigned char lastone);
void           I2CSendStop(void);

#define I2CGetByte()     _I2CGetByte(0)
#define I2CGetLastByte() _I2CGetByte(1)

void _I2CWait(void)           // ceka cca 4.7uS
{                             // pocitam 11.0592MHz krystal
  // asm nelze -neumi prekldac !!!
  //asm("   NOP");
  unsigned char i;
  i++; i++; i++; i++; i++; i++; i++; i++; i++; i++; i++; i++;
  return;
}



void _I2CSCLHigh(void)        // nastavi SCL a kontroluje, jestli to tak
je
{
  register int err;
  SCL = 1;
  while (! SCL)
  {
    err++;
    if (!err)
    {
      _i2c_error &= 0x02;     //zaznam
      return;
    }
  }
```

```
}

void I2CSendAddr(unsigned char addr, unsigned char rd)
{
  SCL = 1;
  _I2CWait();
  SDA = 0;                          // generuje start
  _I2CWait();
  SCL = 0;
  _I2CWait();
  I2CSendByte(addr+rd);             // posle adresovy byte
}

void I2CSendByte(unsigned char bt)
{
  register unsigned char i;
  for (i=0; i<8; i++)
  {
    if (bt & 0x80) SDA = 1;         // posle prvni MSB
    else SDA = 0;
    _I2CSCLHigh();
    _I2CWait();
    SCL = 0;
    _I2CWait();
    bt = bt << 1;
  }
  SDA = 1;                          // sleduje ACK
  _I2CSCLHigh();
  _I2CWait();
  if (SDA)
    _i2c_error &= 0x01;             // jestlize neprisel ACK -zaznam
  SCL = 0;
  _I2CWait();
}

unsigned char _I2CGetByte(unsigned char lastone) // lastone == 1 kdyz ctu
posledni
{
  register unsigned char i, res;
  res = 0;
  for (i=0;i<8;i++)                 // prvni MSB
  {
    _I2CSCLHigh();
    _I2CWait();
    res *= 2;
    if (SDA) res++;
    SCL = 0;
    _I2CWait();
  }
  SDA = lastone;
  _I2CSCLHigh();
  _I2CWait();
  SCL = 0;
  SDA = 1;
  _I2CWait();
  return(res);
}

void I2CSendStop(void)              //ukonceni prenosu
{
  SDA = 0;
```

```
  _I2CWait();
  _I2CSCLHigh();
  _I2CWait();
  SDA = 1;
  _I2CWait();
}



void I2CReset(void)
{
  uchar i=0;
  SDA=1;
  do
  {
    i++;
    _i2c_error=0xFF;
    I2CSendStop();
    if(_i2c_error!=0xFF) continue;

    _i2c_error=0xFF;
    I2CSendAddr(0xA0,WRITE);
    if(_i2c_error!=0xFF) continue;

    _i2c_error=0xFF;
    I2CSendByte(0);
    if(_i2c_error!=0xFF) continue;

    //I2CSendByte(0);

    _i2c_error=0xFF;
    I2CSendAddr(0xA0,READ);
    if(_i2c_error!=0xFF) continue;

    if(I2CGetByte()==0xFF)
      if(I2CGetByte()==0xFF)
          if(I2CGetByte()==0)
              if(I2CGetLastByte()==1)
              {
                I2CSendStop();
                break;
              }
  }while(i<255);
}
```

Příloha 3 – zdrojový kód v souboru i2czapis.c

```
//Lukas Petrus 9.12.2002
//-------zapis do EEPROM I2C - jeden radek--------------------------
//!!!pozor vypina preruseni!!!
// casovano pro  krystal 22,1184MHz

bit zapis_radek (uchar radek, uchar *sloupec)
{
  uchar i, stranka;
  uchar koneccc=0;
  uint  aaa;

  stranka=radek/32;
  stranka<<=1;
  radek%=32;
  radek*=8;
  EA=0;

  //----------------zapise radek do tabulky--------------

  I2CSendAddr(I2CADDR+stranka,WRITE);     //I2CADDR+stranka pameti
                                          //adresa zarizeni+stranky
  I2CSendByte(radek);                     //adresa bytu

  for(i=0;i<8;i++)
  {
    I2CSendByte(sloupec[i]);
    _cekej(20000);                        //pro krystal 22,xxxMHz
//    I2CSendByte(sloupec[i]);
//    _cekej(20000);
  }
  I2CSendStop();

  //------------zpetne cteni (kontrola)----------------------
/*
  I2CSendAddr(I2CADDR+stranka,READ);
  i=I2CGetByte();
  I2CSendStop();
*/
  _cekej(20000);

  I2CSendAddr(I2CADDR+stranka,WRITE
  I2CSendByte(radek);                     //adresa bytu  0
  I2CSendAddr(I2CADDR+stranka,READ);

  for(i=0;i<8;i++)
  {
    if(i==7)   aaa=I2CGetLastByte();
    else       aaa=I2CGetByte();
    _cekej(15000);
    if(aaa==sloupec[i]) koneccc++;
  }

  EA=1;

  if(koneccc>=8) return 1;
  else           return 0;
}
```

Příloha 4 – zdrojový kód v souboru initcpu.c

```
//Lukas Petrus
//Diplomova prace 2003
//initcpu.c

void init_cpu(void)
{
  IE=0x17; //1A;  //povoluje preruseni
      //IE.7=EA  = 0 vsechna prer.
      //IE.6     = 0 reserved
      //IE.5=ET2 = 0 (8052)
      //IE.4=ES  = 1 serial
      //IE.3=ET1 = 1 Timer1
      //IE.2=EX1 = 0 Externi1
      //IE.1=ET0 = 1 Ttimer0
      //IE.0=EX0 = 0 ext0
  IP=0x02;  // nastavuje priority preruseni
      //IP.5=PT2 = 0 (8052)
      //IP.4=PS  = 0 serial
      //IP.3=PT1 = 0 timer1
      //IP.2=PX1 = 0 ext1
      //IP.1=PT0 = 1 timer0
      //IP.0=PX0 = 0 ext0
  TMOD=0x21;  // nastavuje:
      //    000
      //TMOD.7 GATE= 0 ...cas.1 rizen TR1
      //TMOD.6 C/T = 0 ...vyber=casovac
      //TMOD.5 M1  = 1 ...cas.1 v rezim 2
      //TMOD.4 M0  = 0
      //TMOD.3 GATE= 0 ...cas.0 rizen TR0
      //TMOD.2 C/T = 0 ...vyber=casovac
      //TMOD.1 M1  = 0 ...cas.0 v rezim 1
      //TMOD.0 M0  = 1
  TCON=0x51;  //nastavuje:
      //TCON.7 TF1 = 0 ...preteceni casovace 1
      //TCON.6 TR1 = 1 ...povol beh casovace 1
      //TCON.5 TF0 = 0 ...preteceni casovace 0
      //TCON.4 TR0 = 1 ...povol beh casovace 0
      //TCON.3 IE1 = 0 ...preruseni 1 ne
      //TCON.2 IT1 = 0 ...urovni 0
      //TCON.1 IE0 = 0 ...preruseni 0 ano
      //TCON.0 IT0 = 1 ...hranou
  PCON=0x80;  //nastavuje:
      //PCON.7 SMOD= 1 ...dvojnasobna rychlost prenosu
  SCON=0x50; //nastavuje:
      //SCON.7 SM0 = 0 ...rezim ser. portu
      //SCON.6 SM1 = 1 ...
      //SCON.5 SM2 = 0 ...povoleni viceprocesorove komunikace 1
      //SCON.4 REN = 1 ...povoleni prenosu 1
      //SCON.3 TB8 = 0 ...devaty datovy bit vysilani
      //SCON.2 RB8 = 0 ...devaty datovy bit prijmu
      //SCON.1 TI  = 0 ...priznak preruseni - vysilani
      //SCON.0 RI  = 0 ...prizank preruseni - prijem

//  P3=0xCF;  //NASTAVUJE
      //P3.7 RD    = 1 ...cteni z externi pameti
      //P3.6 WR    = 1 ...zapis do externi pameti
      //P3.5 T1    = 0 ...vnejsi vstup cit/cas1
      //P3.4 T0    = 0 ...vnejsi vstup cit/cas0
      //P3.3 INT1  = 1 ...vnejsi preruseni
      //P3.2 INT0  = 1 ...vnejsi preruseni
      //P3.1 TXD   = 1  ...seriovy vystupni port
      //P3.0 RXD   = 1  ...seriovy vstupni port
```

```
  P0=P1=P2=P3=0xFF;
  TH1=BAUDDIV;
  TH0=TIMER/256;
  TL0=TIMER;
//  TI=0;
//  RI=0;
}
```

Příloha 5 – zdrojový kód v souboru seriovka.c

```
// Lukas Petrus 7.12.2002
// funkce pro prijimani a vysilani po seriove lince
// pocitani CRC atd.

#ifndef ADDR
  #define ADDR            0xE1
#endif

#ifndef MAGIC1
  #define MAGIC1          0x4B   //na ten slysi
#endif

#ifndef MAGIC2
  #define MAGIC2          0xB4
#endif

#ifndef MAGIC1OUT
  #define MAGIC1OUT       0x5C   //ten posila
#endif

#ifndef MAGIC2OUT
  #define MAGIC2OUT       0xC5
#endif

#ifndef DELKA
  #define DELKA           35        //kolik se ma prijmout max znaku po ser
                                    //lince+3 rezerva.
#endif

#ifndef MINMSG
  #define MINMSG          5        //minimalni delka z prijimanych paketu
#endif

enum { MAG1=0, ADRESA, CMD, MAG2, SIZE, DATA};

sbit DTR       =P1^0;               // nastavuje DTR !je to NON

uchar index_msg=0;                  //index do pole zpravy
uchar idata zprava[DELKA];          //pole pro prijimane zpravy
uchar sendCRC=0;
uchar esc=0;                        //priznak pro A7, A8, 27 pri prijmu
uchar delka=0;                      //delka prisle zpravy


void serial (void) interrupt 4 using 3
{
  if (RI)
  {
    RI=0;
    if (delka==0)
    {
      if (index_msg>=DELKA) index_msg=0;

      if (esc==1)
      {
        zprava[index_msg++]=SBUF;
        esc=0;
      }
      else
        switch (SBUF)
        {
          case   0x27: esc=1; break;
```

```
          case   0xA7: index_msg=0; break;
          case   0xA8: if (index_msg>=MINMSG) delka=index_msg; break;
          default:   zprava[index_msg++]=SBUF; break;
        }
    }
  }
  if (TI) {}
}


uchar CRCni (uchar Bajt, uchar CRC)      //vypocet kontrolniho souctu CRC
{
  #define Poly            0x8C
  uchar i, Bit;

  for (i=0; i<8; i++)
  {
    Bit = CRC & 1;
    CRC >>= 1;
    if (Bajt & 1)  CRC |= 0x80;
    if (Bit)       CRC ^= Poly;
    Bajt >>= 1;
  }
  return CRC;
}


void vysli(uchar dato)
{
  SBUF=dato;
  while (TI==0) {}//{automat();}
  TI=0;
}


void vysliCRC (uchar dato)
{
  if (dato==0xA7 || dato==0xA8 || dato==0x27)  vysli(0x27);
  vysli(dato);
  sendCRC= CRCni (dato, sendCRC);
}


void vyslipaket (uchar command, uchar sizedata)
{
  uchar i=0;
  zprava[MAG1]=MAGIC1OUT;
  zprava[MAG2]=MAGIC2OUT;
  zprava[ADRESA]=ADDR;

  zprava[CMD]=command;
  zprava[SIZE]=sizedata;
  sendCRC=0;
  sizedata+=5;

  DTR=0;
  vysli(0xA7);
  for (i=0; i<sizedata; i++) vysliCRC (zprava[i]);
```

```
  vysliCRC(CRCni (0x00, sendCRC));
  vysli(0xA8);
  DTR=1;
}



bit paketok(void)
{
  uchar CRCbyte=0;
  uchar i=0;

  if ((zprava[MAG1]!=MAGIC1) || (zprava[MAG2]!=MAGIC2) ||
(delka!=6+zprava[SIZE])) return 1;
  else
  {
    for (i=0; i<delka; i++)
    {
      CRCbyte=CRCni(zprava[i], CRCbyte);
    }
    if (CRCbyte!=0) return 1;
    else return 0;
  }
}



void vyber()
{
  uchar i, stranka, radek;

  if (paketok() || zprava[ADRESA]!=ADDR) return;
  else
  {
    switch (zprava[CMD])
    {
    case 0:  //------vrat cislo verze---------
             zprava[DATA]=(VERZE / 100);
             zprava[DATA+1]=(VERZE % 100);
             vyslipaket(0,2);
             break;
    case 1:  //------vrat stav EEPROM-----------
               //I2CSendAddr(I2CADDR,READ);
               //I2CSendStop();

               EA=0;
               stranka=zprava[DATA]/32;
               radek  =zprava[DATA]%32;
               stranka<<=1;
               radek*=8;
               I2CSendAddr(I2CADDR+stranka,WRITE);
               I2CSendByte(radek);
               I2CSendAddr(I2CADDR+stranka,READ);
               zprava[DATA]    =I2CGetByte();
               zprava[DATA+1]  =I2CGetByte();
               zprava[DATA+2]  =I2CGetByte();
               zprava[DATA+3]  =I2CGetByte();
               zprava[DATA+4]  =I2CGetByte();
               zprava[DATA+5]  =I2CGetByte();
               zprava[DATA+6]  =I2CGetByte();
               zprava[DATA+7]  =I2CGetLastByte();
               I2CSendStop();
```

```
            EA=1;
          vyslipaket(1, 8);
          break;

case 2:   //------STOP---------------
          for(i=0;i<MOTORU;i++) motor[i].stoppozice=motor[i].pozice;
          napln_enable=0;
          brzdi=0;
          vyslipaket(2,0);
          break;

case 3:   //------Kalibruj------
          vyslipaket(3,0);
          nulovapoloha();
          break;

case 4:   //------dojed na pozici--------------
          automat_enable=1;
          napln_enable=0;
          brzdi=0;

          for(i=0;i<MOTORU;i++)
          {
            motor[i].zapnuty=1;
            motor[i].stoppozice=zprava[DATA+(i*2)];
            motor[i].stoppozice<<=8;
            motor[i].stoppozice|=zprava[DATA+1+(i*2)];
            motor[i].smer=0;

            if (motor[i].stoppozice > motor[i].pozice)
            {
              motor[i].smer=1;
              motor[i].kroku=motor[i].stoppozice-motor[i].pozice;
            }

            if (motor[i].stoppozice < motor[i].pozice)
            {
              motor[i].smer=-1;
              motor[i].kroku=motor[i].pozice-motor[i].stoppozice;
            }

            if (motor[i].kroku > brzdi) brzdi=motor[i].kroku;
          }

          kmax=brzdi;
          cas_index=ROZJEZD-1;
          brzdi-=ROZJEZD*akcelerace;
          poradi=0;
          vyslipaket(4,0);
          break;
case 5:   //-----start auto----------------
          automat_enable=1;
          napln_enable=1;
          index_tab--;
          napln();
          vyslipaket(5,0);
          break;

case 6:   //-----maxSP----------------
          zprava[DATA]=maxSP;
          vyslipaket(6,1);
          break;
```

```
 case 7:  //-----zapis radek do EEPROM----------------
          zapis_radek (zprava[DATA+1], zprava+7);
          vyslipaket(7,0);
          break;

case 10: // posun motoru o inkrement
          automat_enable=1;
          napln_enable=0;
          if(zprava[DATA+1])
          {
               motor[zprava[DATA]].stoppozice=motor[zprava[DATA]].pozic
               e+(zprava[DATA+2]*256)+zprava[DATA+3];
               motor[zprava[DATA]].smer=1;
               motor[zprava[DATA]].kroku=motor[zprava[DATA]].stoppozice
               -motor[zprava[DATA]].pozice;
          }
          else
          {
               motor[zprava[DATA]].stoppozice=motor[zprava[DATA]].pozic
               e-(zprava[DATA+2]*256)-zprava[DATA+3];
               motor[zprava[DATA]].smer=-1;
               motor[zprava[DATA]].kroku=motor[zprava[DATA]].pozice-
               motor[zprava[DATA]].stoppozice;
          }
          motor[zprava[DATA]].m_last=0;
          motor[zprava[DATA]].zapnuty=1;
          brzdi=0xFFFF;
          cas_index=2;
          vyslipaket(10,0);
          for(i=0; i<MOTORU; i++)
                  if(motor[i].kroku>kmax) kmax=motor[i].kroku;
          poradi=0;
          break;
 case 11: // vrat aktualni pozici
            zprava[DATA]   = motor[0].pozice>>8;
            zprava[DATA+1] = motor[0].pozice;
            zprava[DATA+2] = motor[1].pozice>>8;
            zprava[DATA+3] = motor[1].pozice;
            zprava[DATA+4] = motor[2].pozice>>8;
            zprava[DATA+5] = motor[2].pozice;
            zprava[DATA+6] = motor[3].pozice>>8;
            zprava[DATA+7] = motor[3].pozice;
            vyslipaket(11,8);
            break;
 case 12: // zapnout/vypnout funkci soucasneho dojezdu motoru
            pocitani_enable^=1;
            vyslipaket(12,0);
            break;
 case 13: //presun index_tab na polohu
            index_tab=zprava[DATA];
            index_tab<<=8;
            index_tab+=zprava[DATA+1];
            vyslipaket(13,0);
            break;
 case 14: //volej funkci tridic()
            zprava[DATA]=tridic();
            vyslipaket(14,1);
            break;
 case 15: //vrat aktualni index_tab
            zprava[DATA]=index_tab>>8;
            zprava[DATA+1]=index_tab;
```

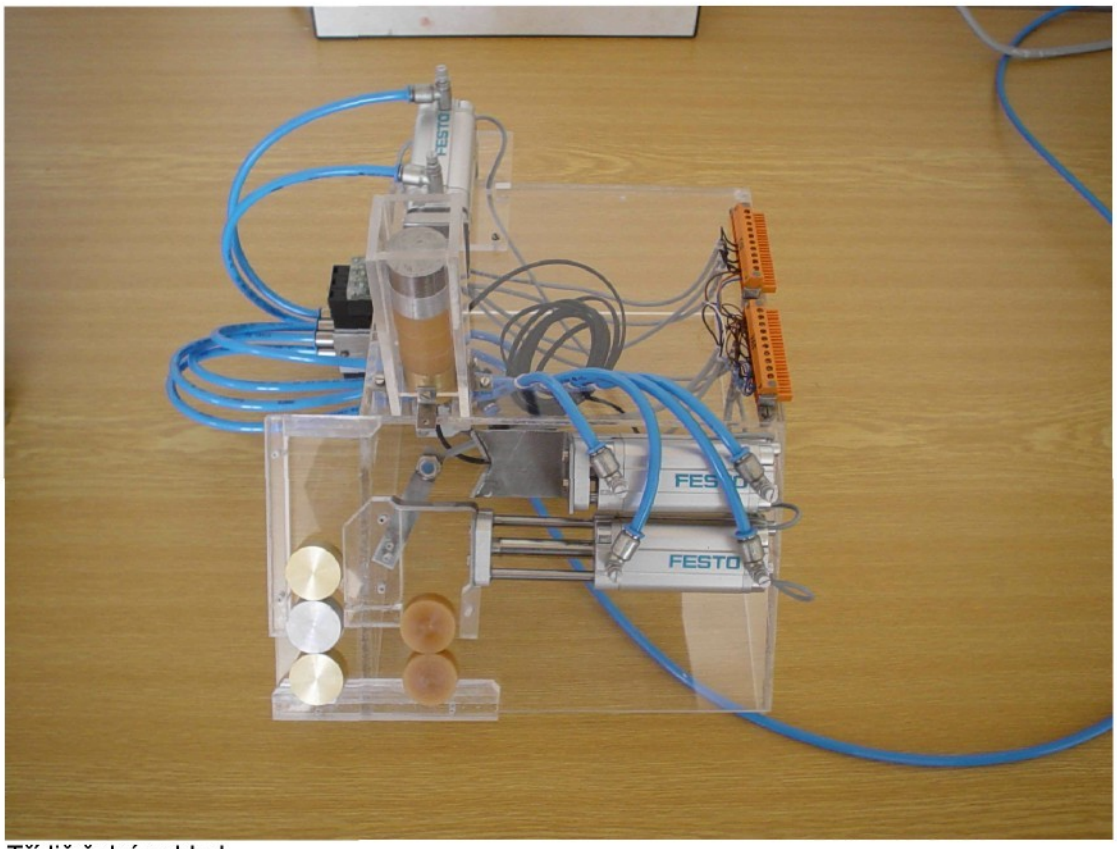
```
                vyslipaket(15,2);
                break;

    default: //odpovi, ze slysi
             vyslipaket(0xFF,0);
             break;
    }
    zprava[SIZE]=0;
  }
}
```

Příloha 6 – zdrojový kód v souboru tridic.c

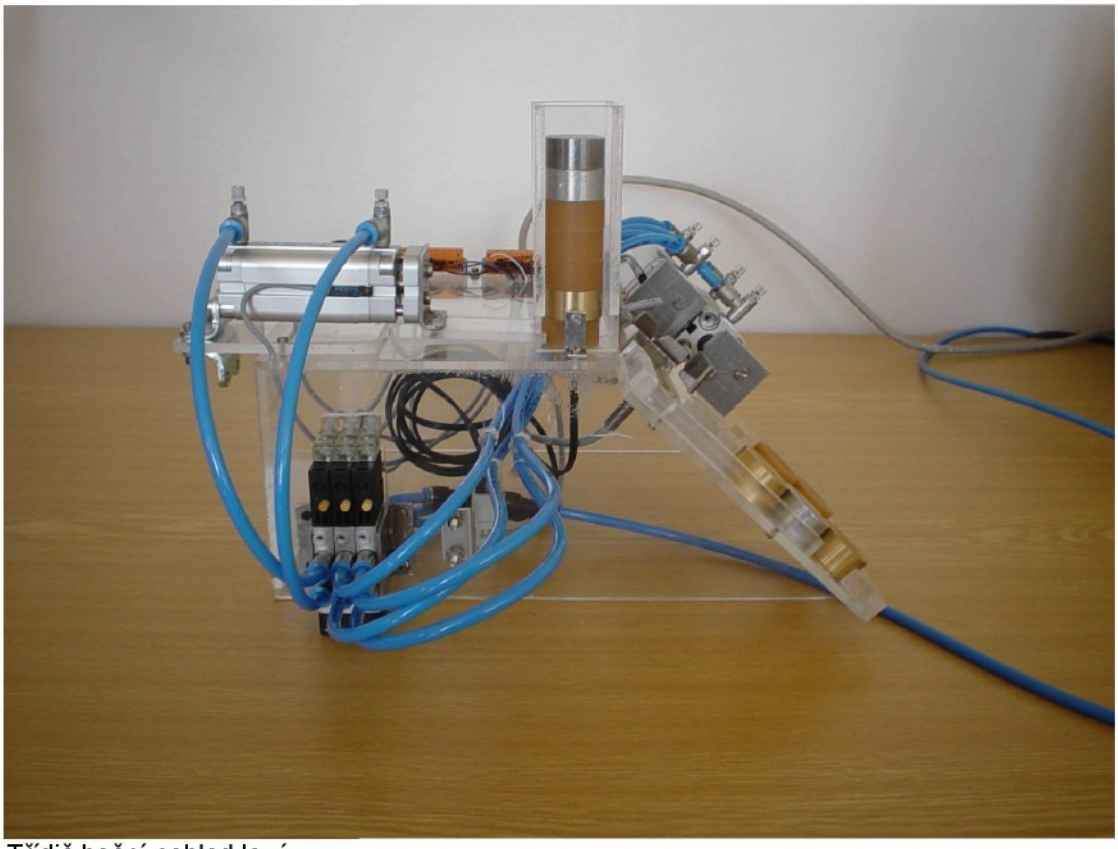
```
//-------obsluha tridice----------------------------
// Lukas Petrus 10.4.2003
// casovano pro  krystal 22,1184MHz
// xdata uchar adresa8002 _at_ 0x8002; uchar na kterem jsou civky ventilu
(wr2)
// xdata uchar adresa8005 _at_ 0x8005; uchar na kterem cidla od tridice
(rd5)
// funkce vraci :       0 - neni material
//                      1 - nekov
//                      2 - kov
//                      3 - chyba HW

uchar stav_motoru=0;
void pneumotor (uchar, uchar);

uchar tridic(void)
{
  if((adresa8005 & 0x02)==0) return 0;
                                        //zjisti, je-li v zasobniku zeton
  pneumotor (2, 1);                     //je-li, pripravi zarazku
  pneumotor (1, 0);                     //zasune tridici motor

  tickfurt=0;
  while((adresa8005 & 0x40)==0 || (adresa8005 & 0x20)==0)
    if(tickfurt>5000) return 3;

                                        //ceka na cidlo vysunuti zarazky a
                                        //a cidlo zasunuti tridiciho motoru

  pneumotor (0, 1);                     //vysune zeton ze zasobniku
  tickfurt=0;
  while((adresa8005 & 0x08)==0)         //ceka na cidlo vysunuti
    if(tickfurt>5000) return 3;


  tickfurt=0;                           //ceka na presun zetonu
  while(tickfurt<1000) {}


  if((adresa8005 & 0x01)==0)            //je-li zeton kovovy
  {
    pneumotor (2, 1);                   //zasune zarazku
    tickfurt=0;
    while((adresa8005 & 0x10)==0)       //ceka na cidlo zasunuti
      if(tickfurt>5000) return 3;
    return 2;
  }
  else                                  //je-li zeton nekovovy
  {
    pneumotor (1, 1);                   //vysune tridici motor
    tickfurt=0;
    while((adresa8005 & 0x80)==0)     //ceka na cidlo vysunuti
      if(tickfurt>5000) return 3;
    tickfurt=0;                           //ceka na presun zetonu
    while(tickfurt<1000) {}
    pneumotor (1, 0);                   //zasune tridici motor
    return 1;
  }

}
```

```
//-------------------------------------------------------------
//funkce ovlada pneumotory.
//  motor urcuje ktery (0-zasobnik, 1-tridic, 2-zarazka)
//  on_off (0- zasunout, 1 vysunout)
void pneumotor (uchar motor, uchar on_off)
{
  if(motor==0)
     if(on_off==0)  adresa8002=stav_motoru & 0xFD;
     else           adresa8002=stav_motoru | 0x02;
  if(motor==1)
     if(on_off==0)  adresa8002=stav_motoru & 0xFB;
     else           adresa8002=stav_motoru | 0x04;
  if(motor==2)
     if(on_off==0)  adresa8002=stav_motoru & 0xF7;
     else           adresa8002=stav_motoru | 0x08;
}
```

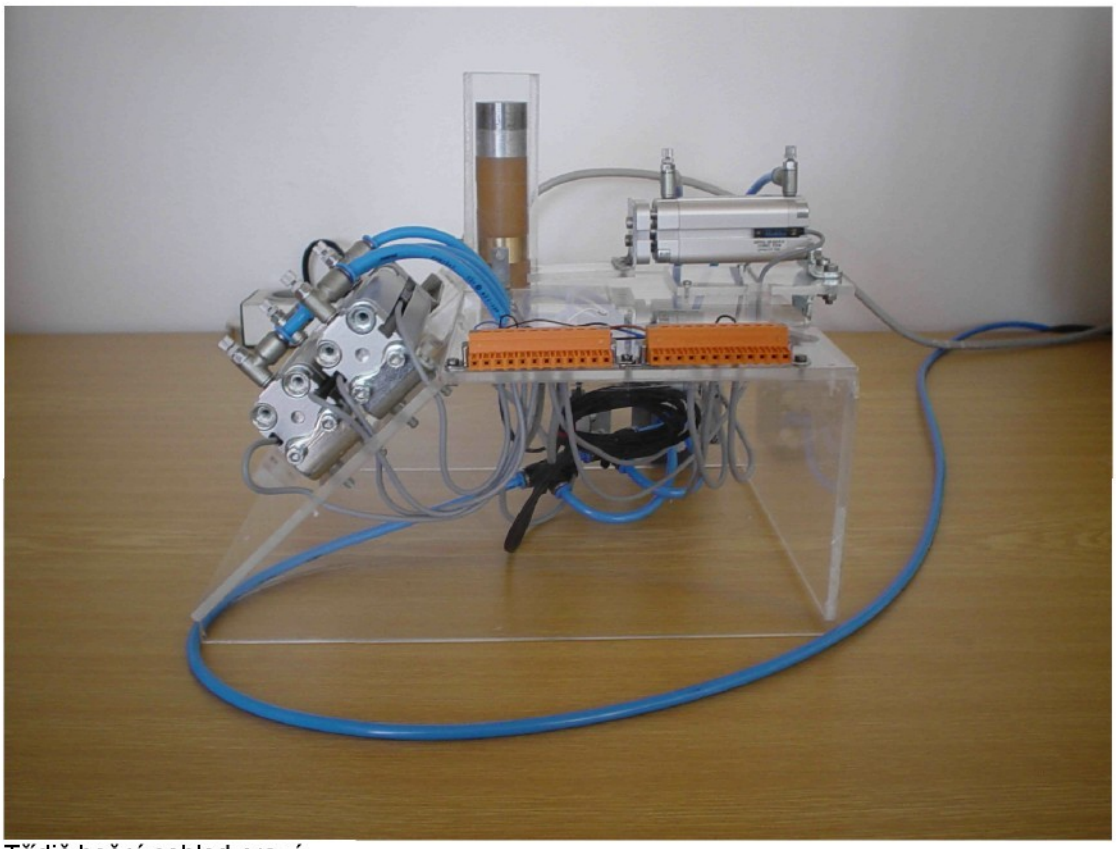
Příloha 7.1 – fotodokumentace - třídič



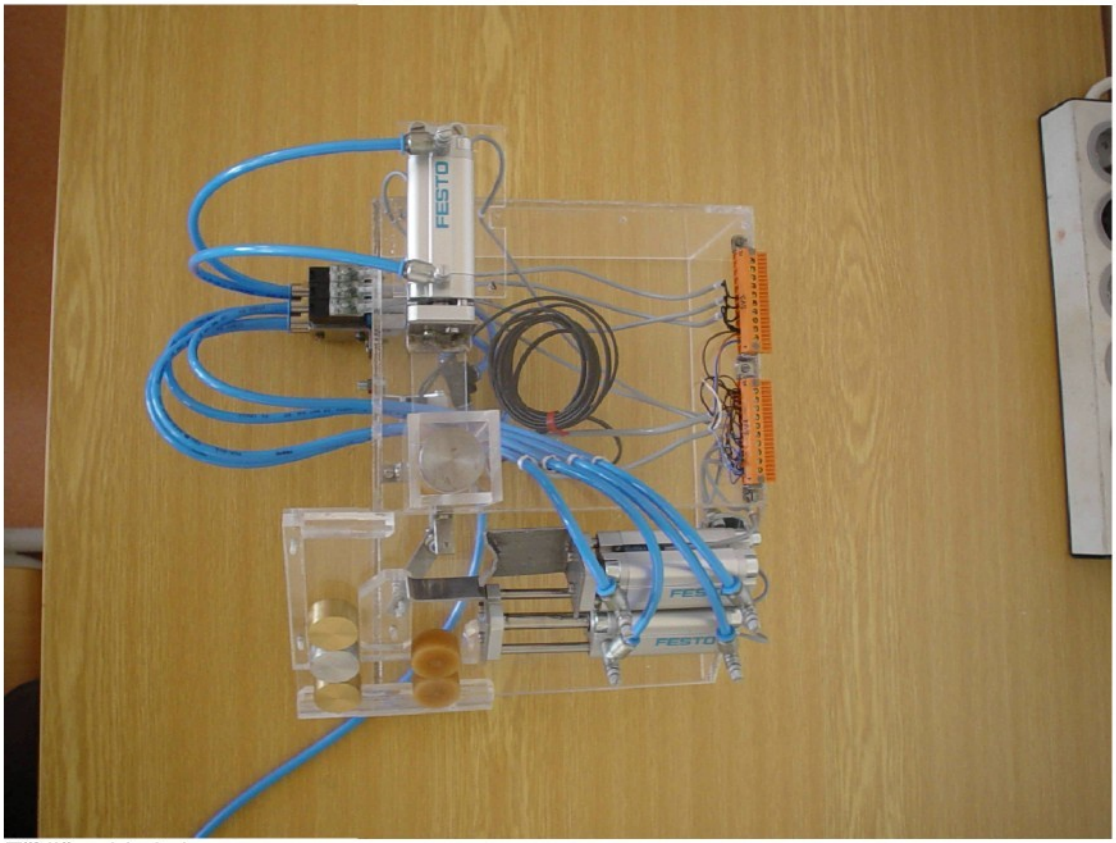
Třídič čelní pohled



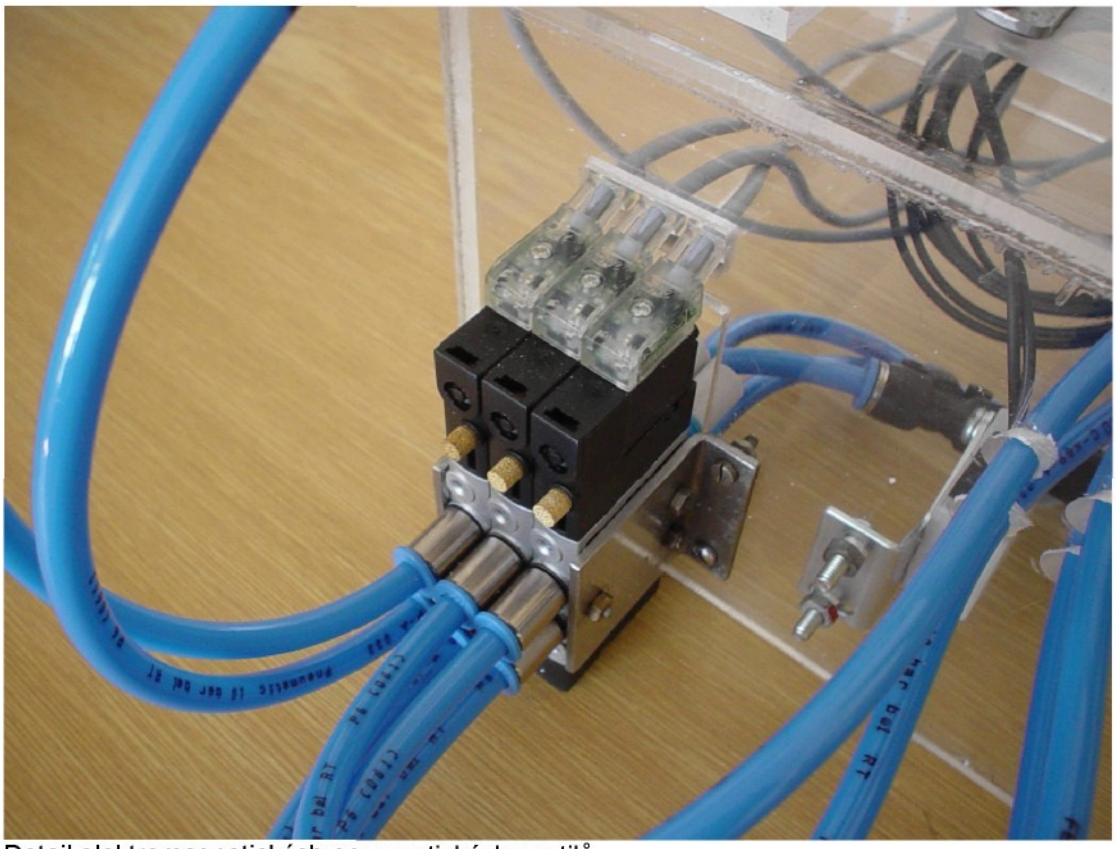
Třídič boční pohled levý



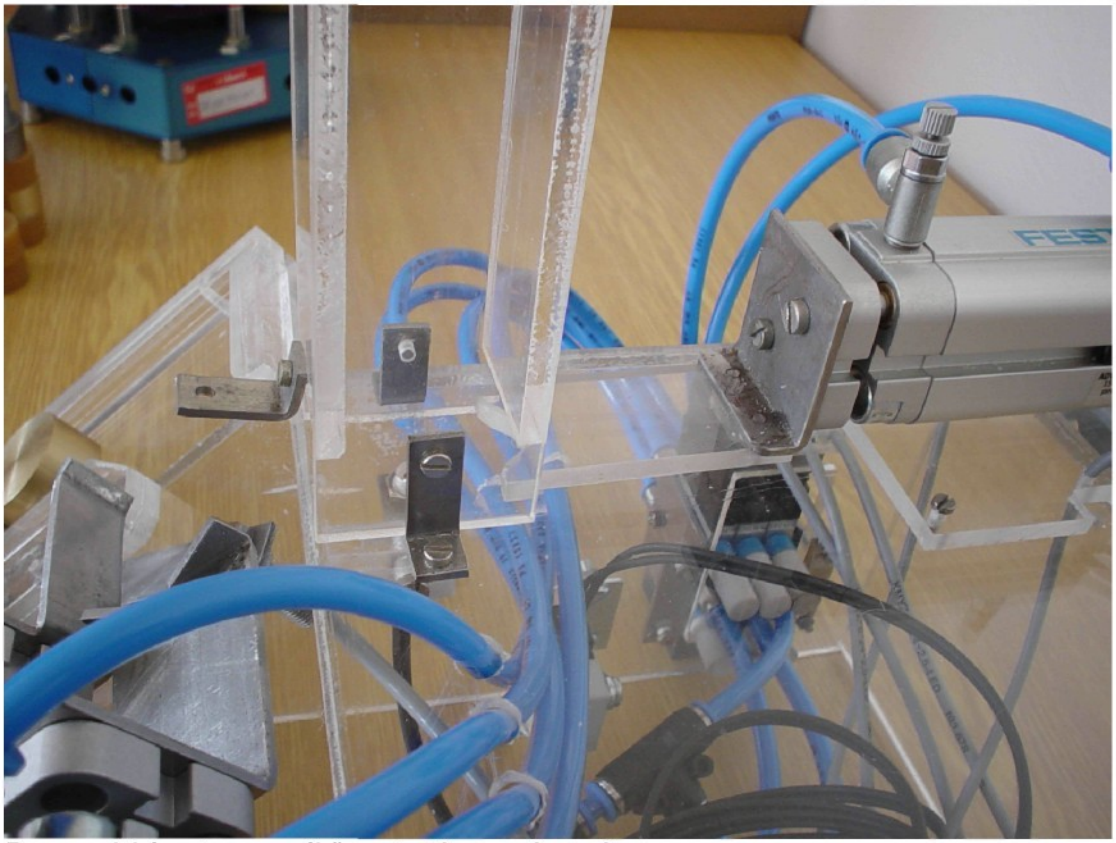
Třídič boční pohled pravý



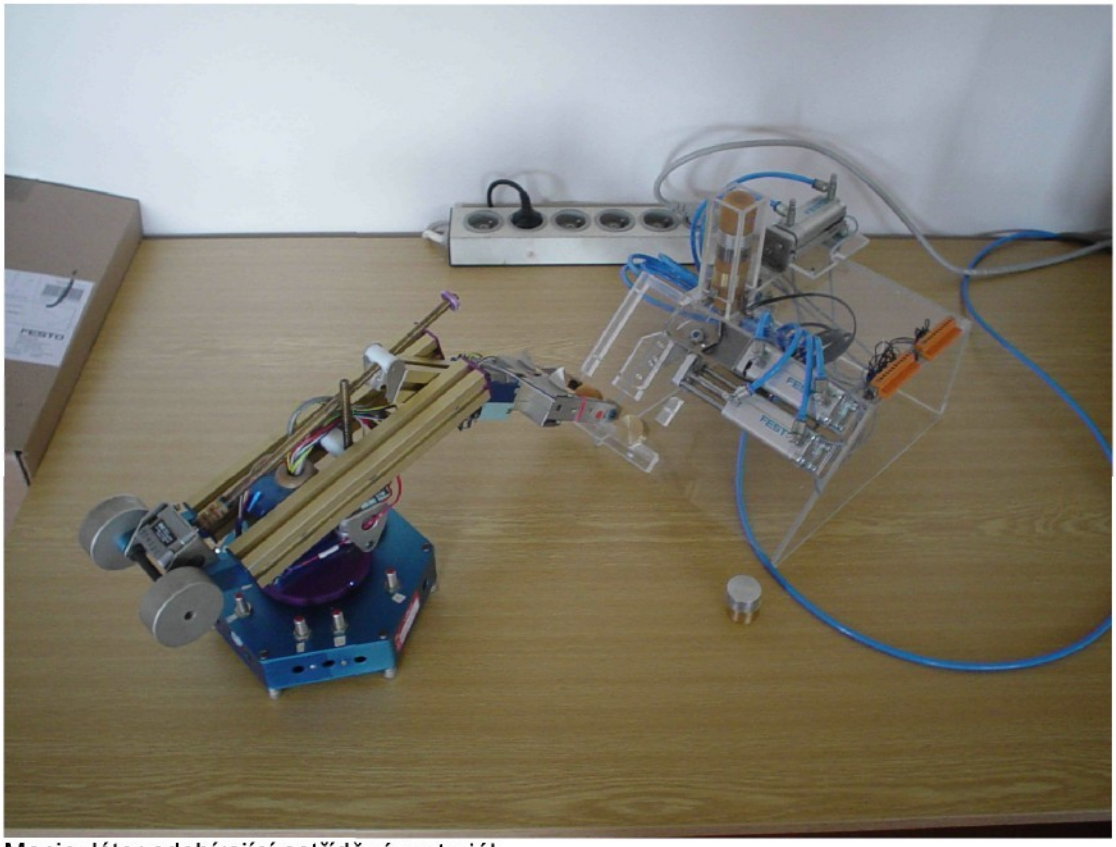
Třídič pohled shora



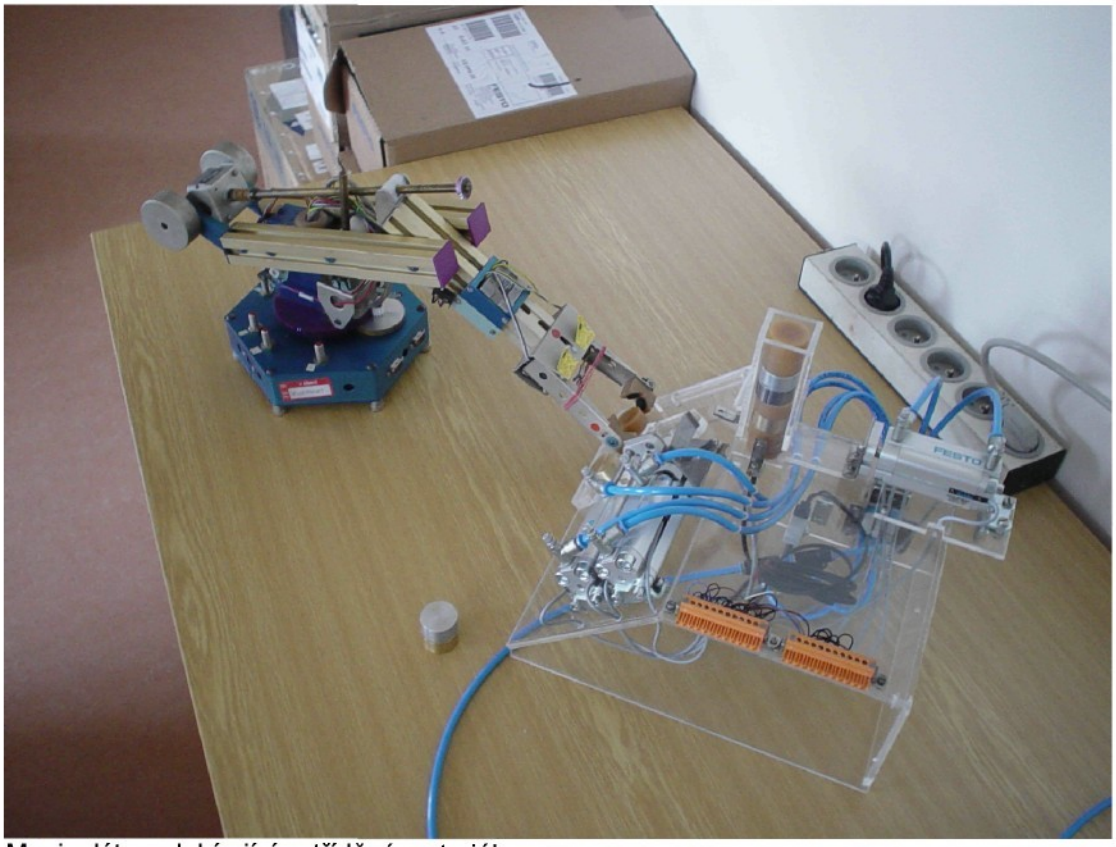
Detail elektromagnetických pneumatických ventilů



Pneumatický motor pro výběr materiálu ze zásobníku

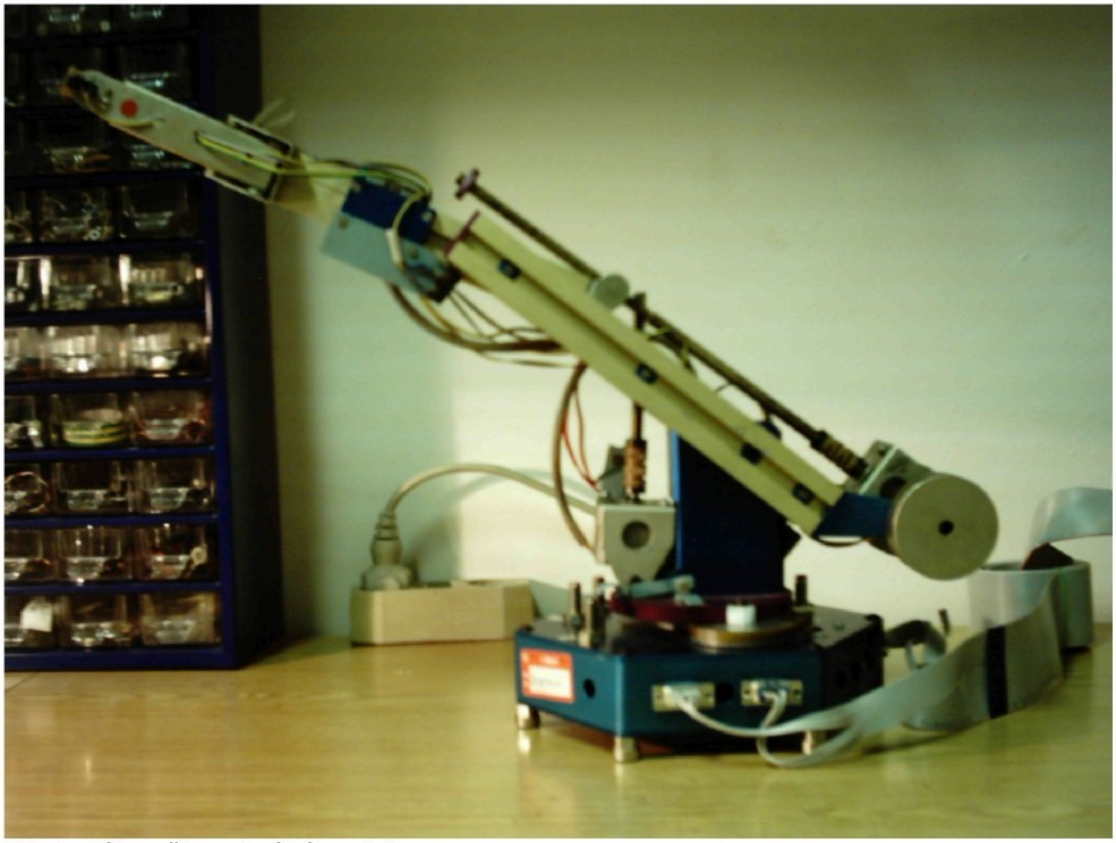


Manipulátor odebírající setříděný materiál

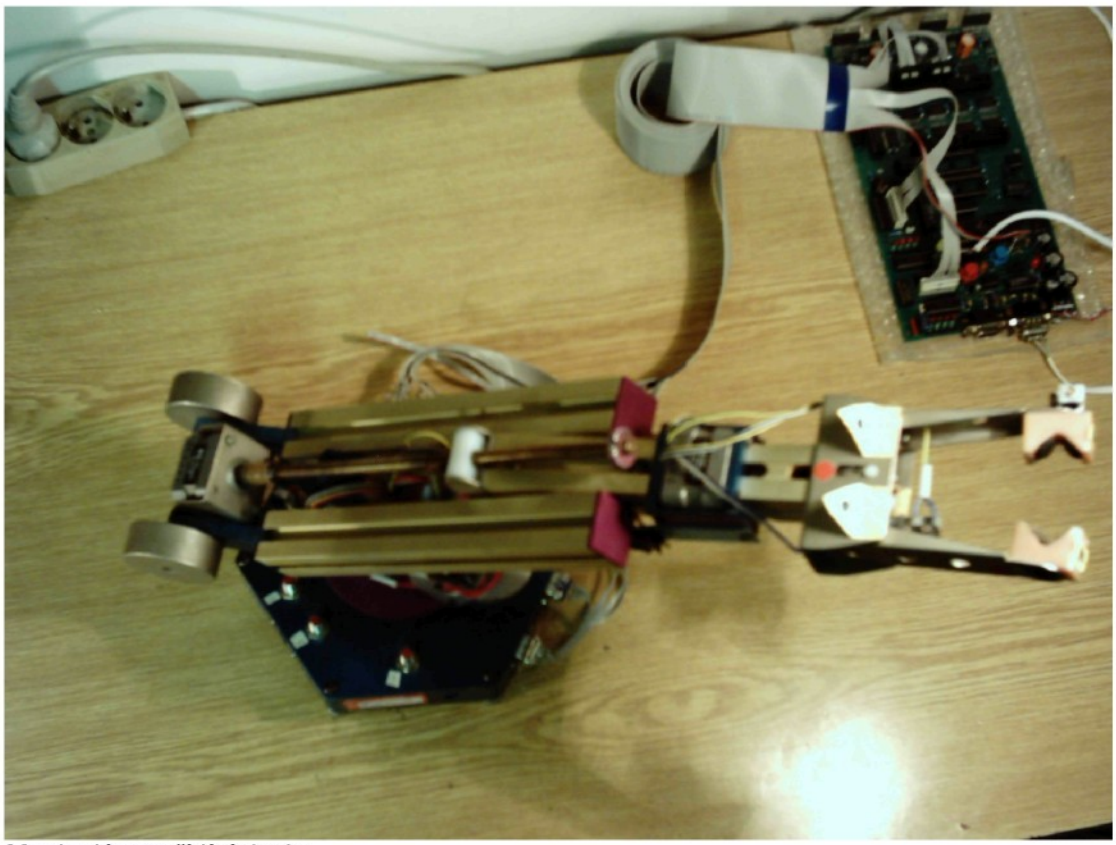


Manipulátor odebírající setříděný materiál

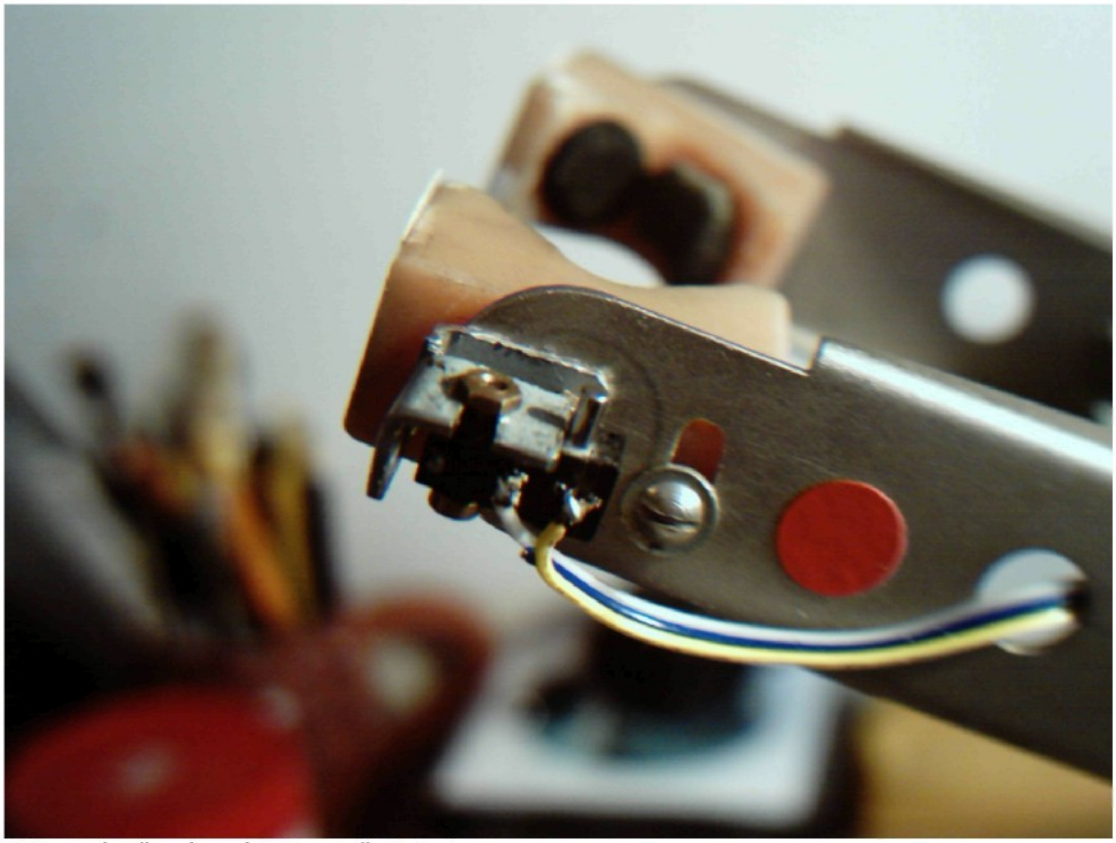
Příloha 7.2 – fotodokumentace - manipulátor



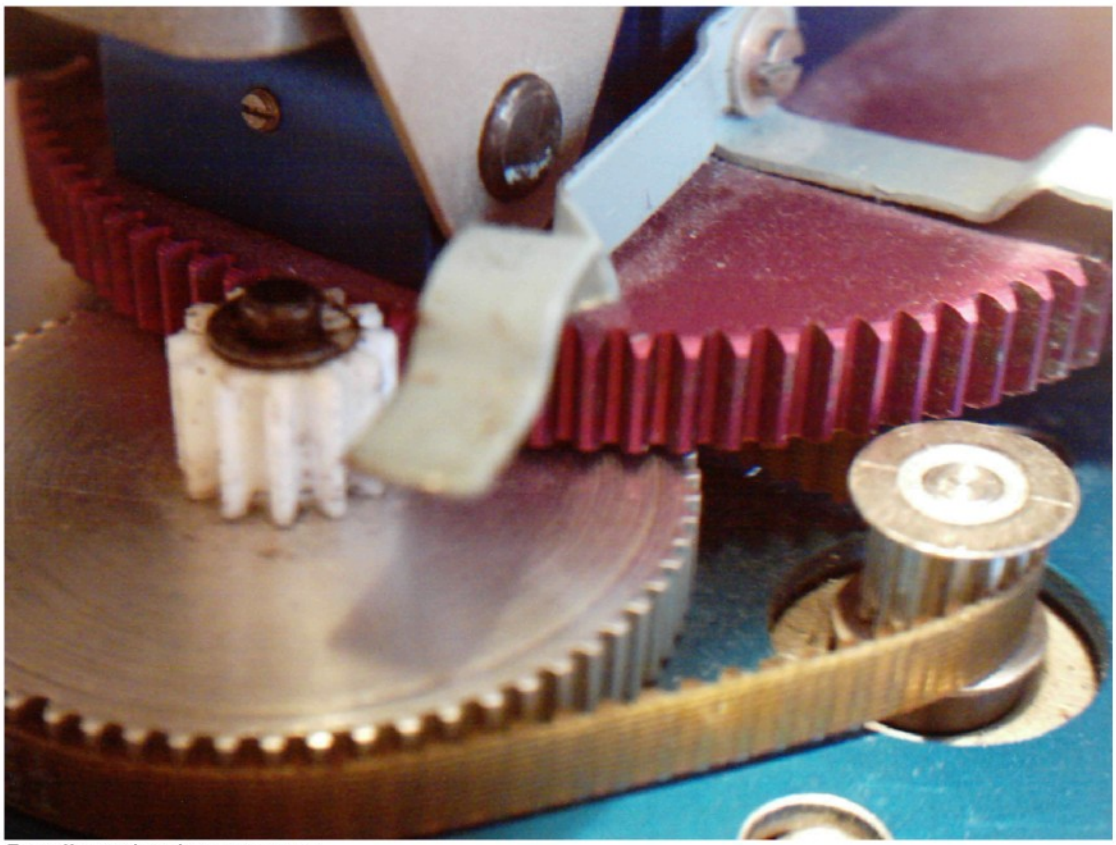
Manipulátor při maximálním zdvihu



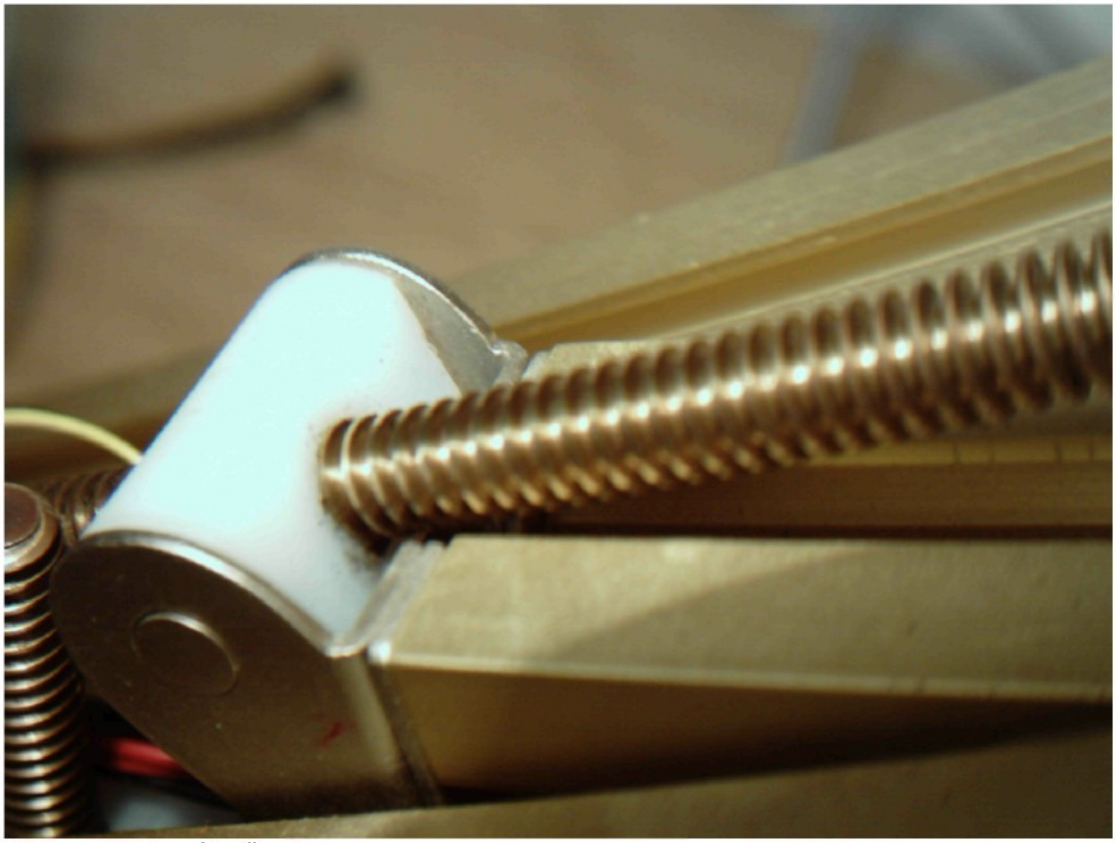
Manipulátor a řídící deska



Mikrospínač spínaný jehlou v čelistech



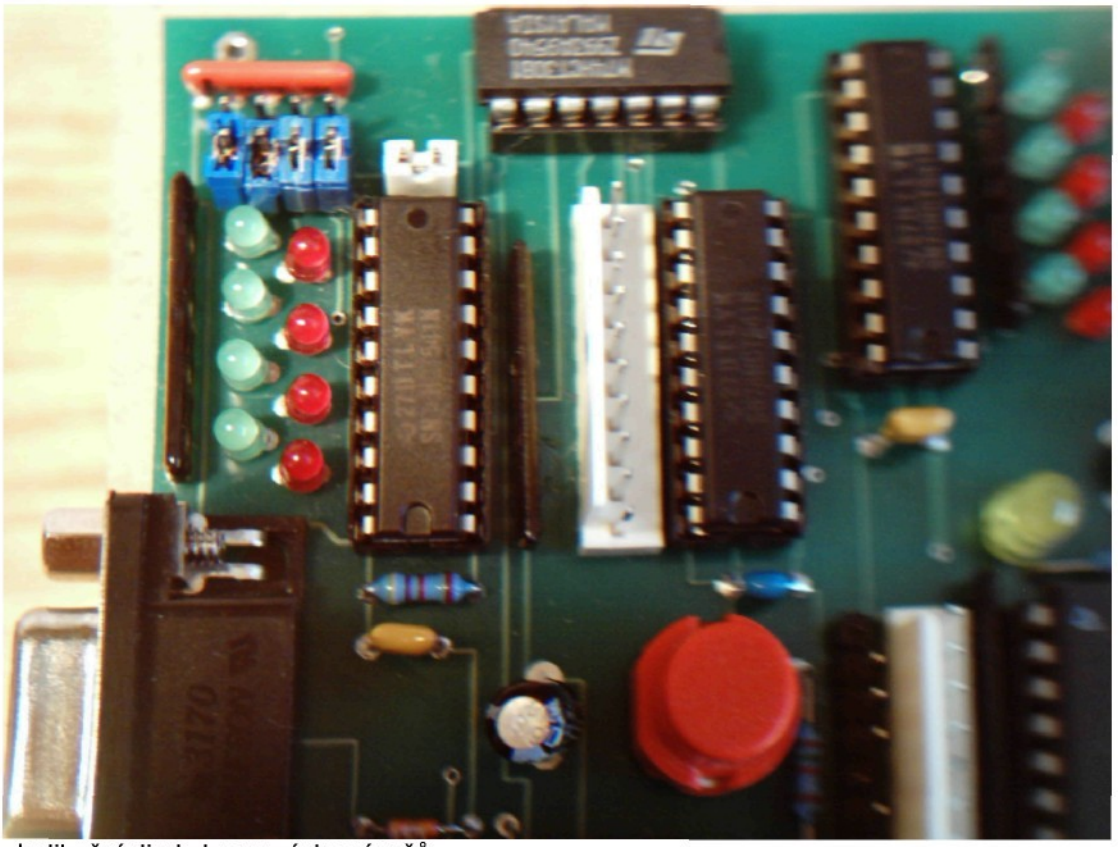
Detail mechanismu rotace



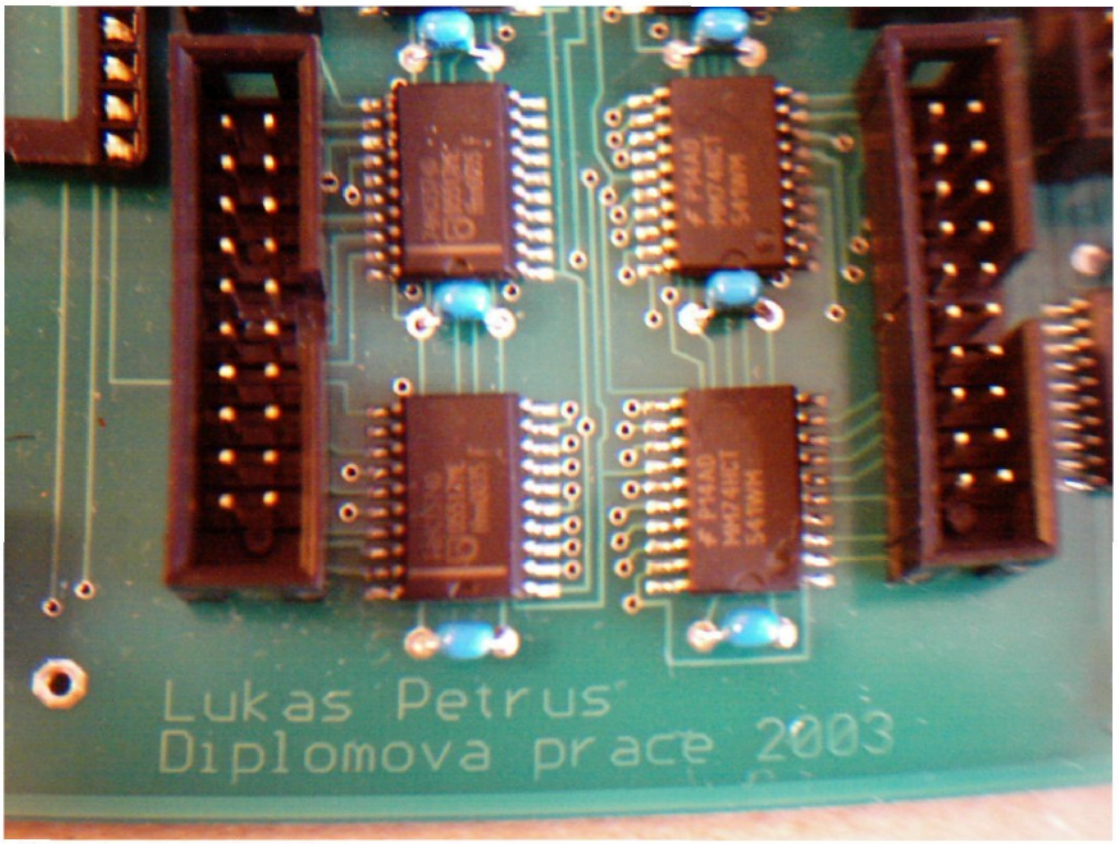
Matice pohybového šroubu ramene

Matice pohybového šroubu zvedáku

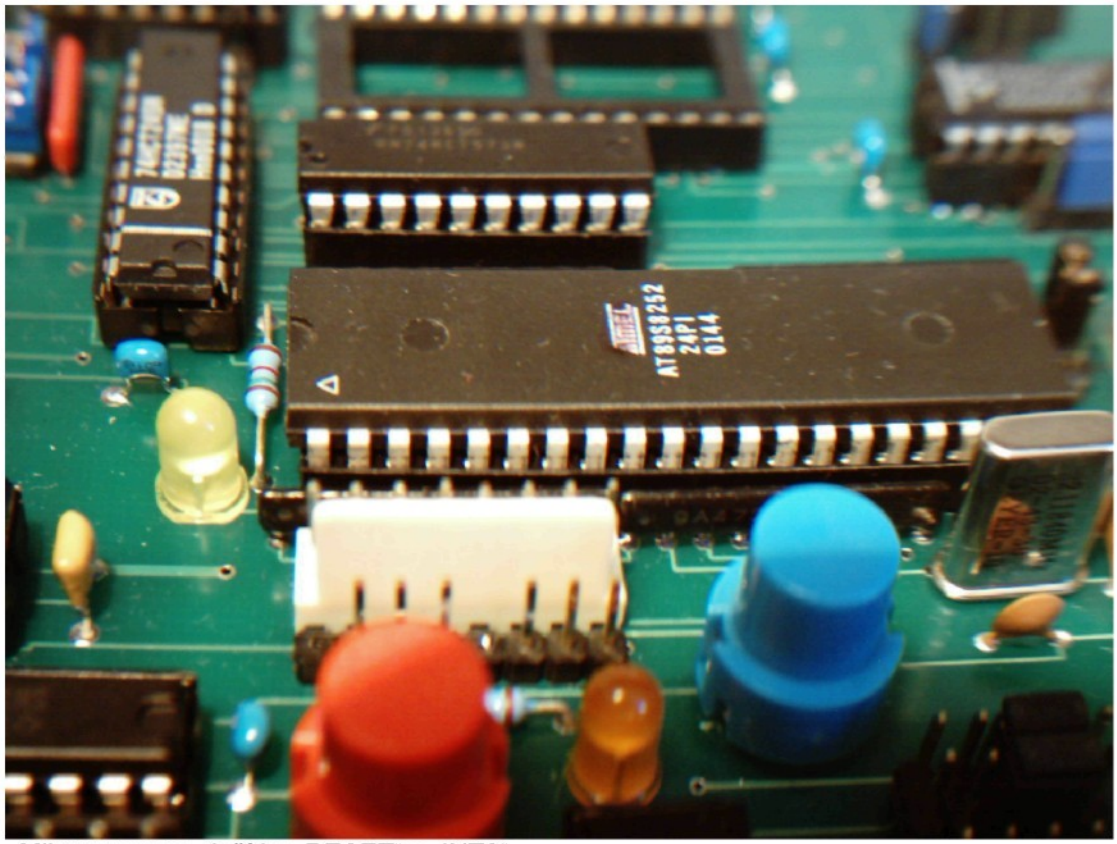
Příloha 7.3 – fotodokumentace – řídící deska



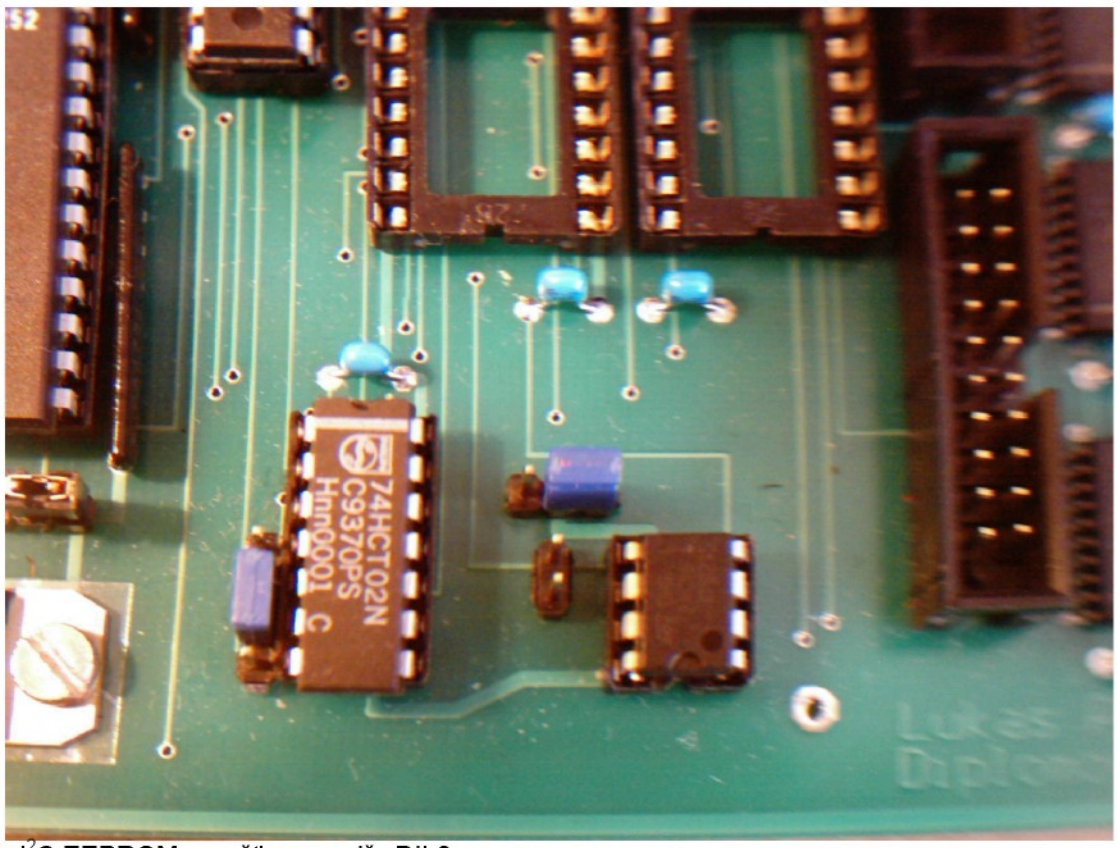
Indikační diody koncových spínačů



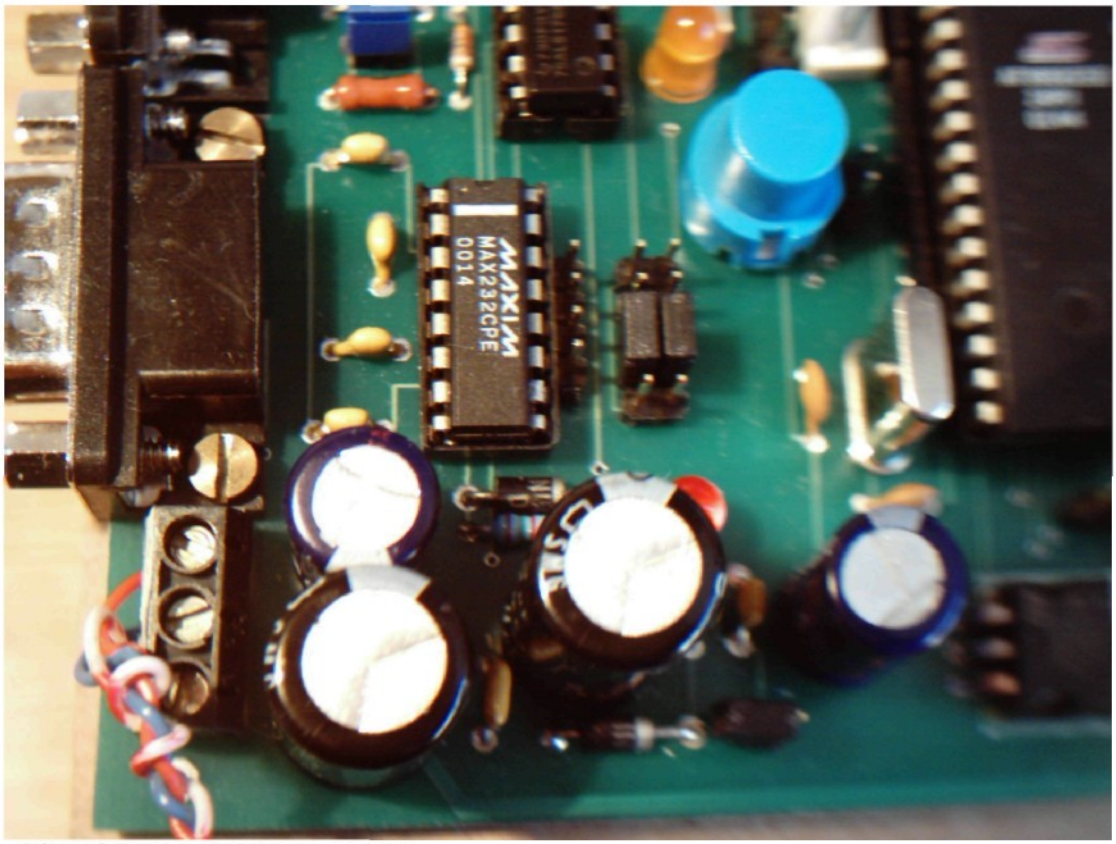
Vstupní a výstupní registry



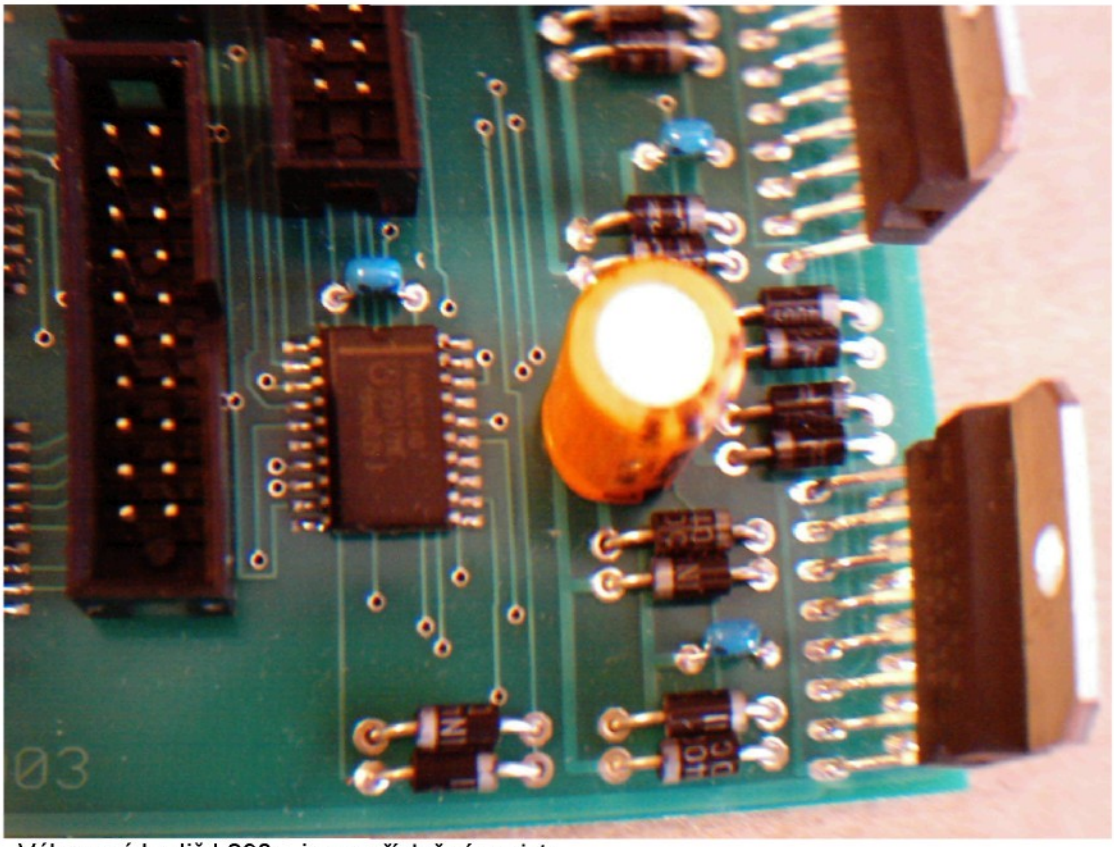
Mikroprocesor, tlačítka „RESET“ a „INT0“



I²C EEPROM paměť v pouzdře DIL8

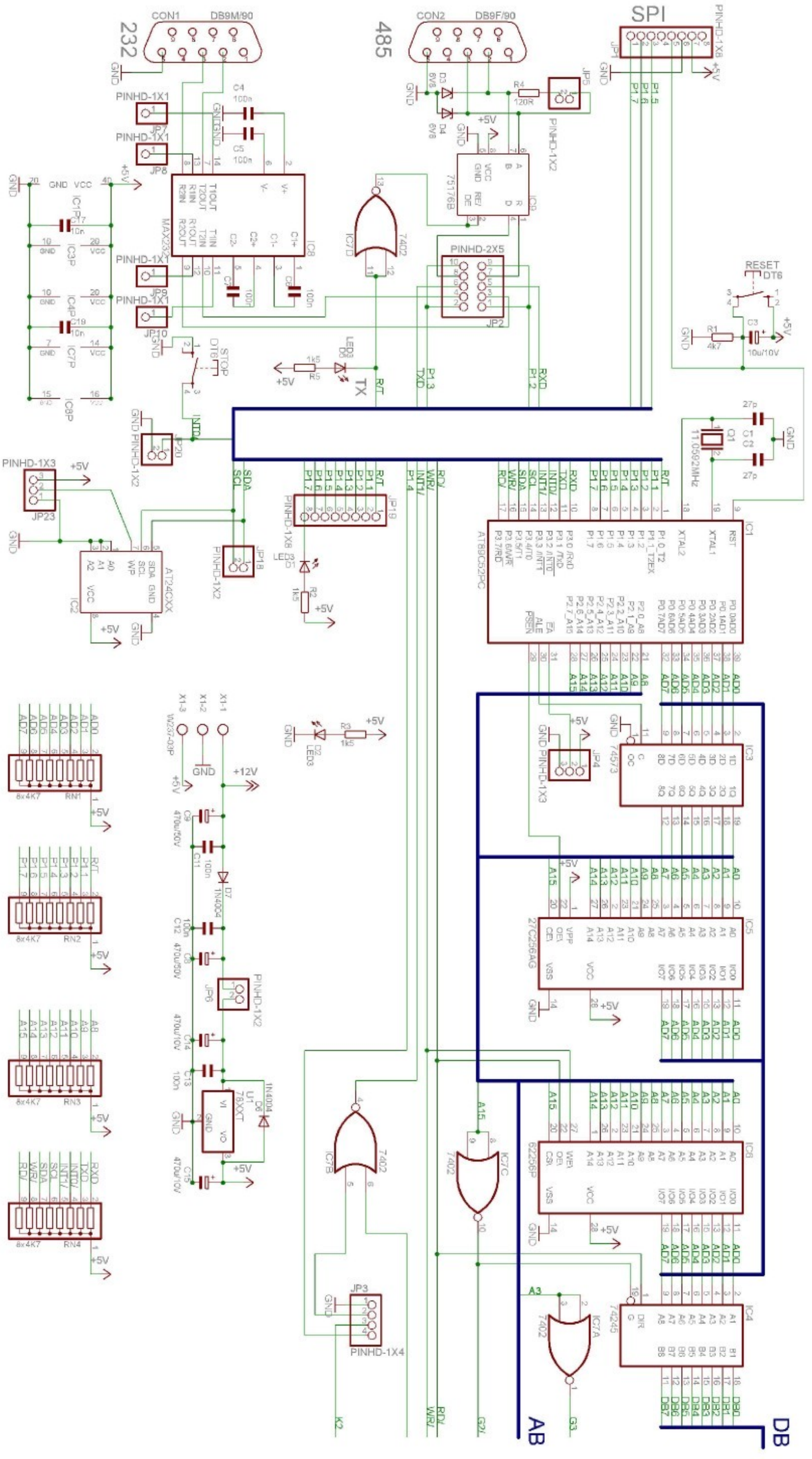


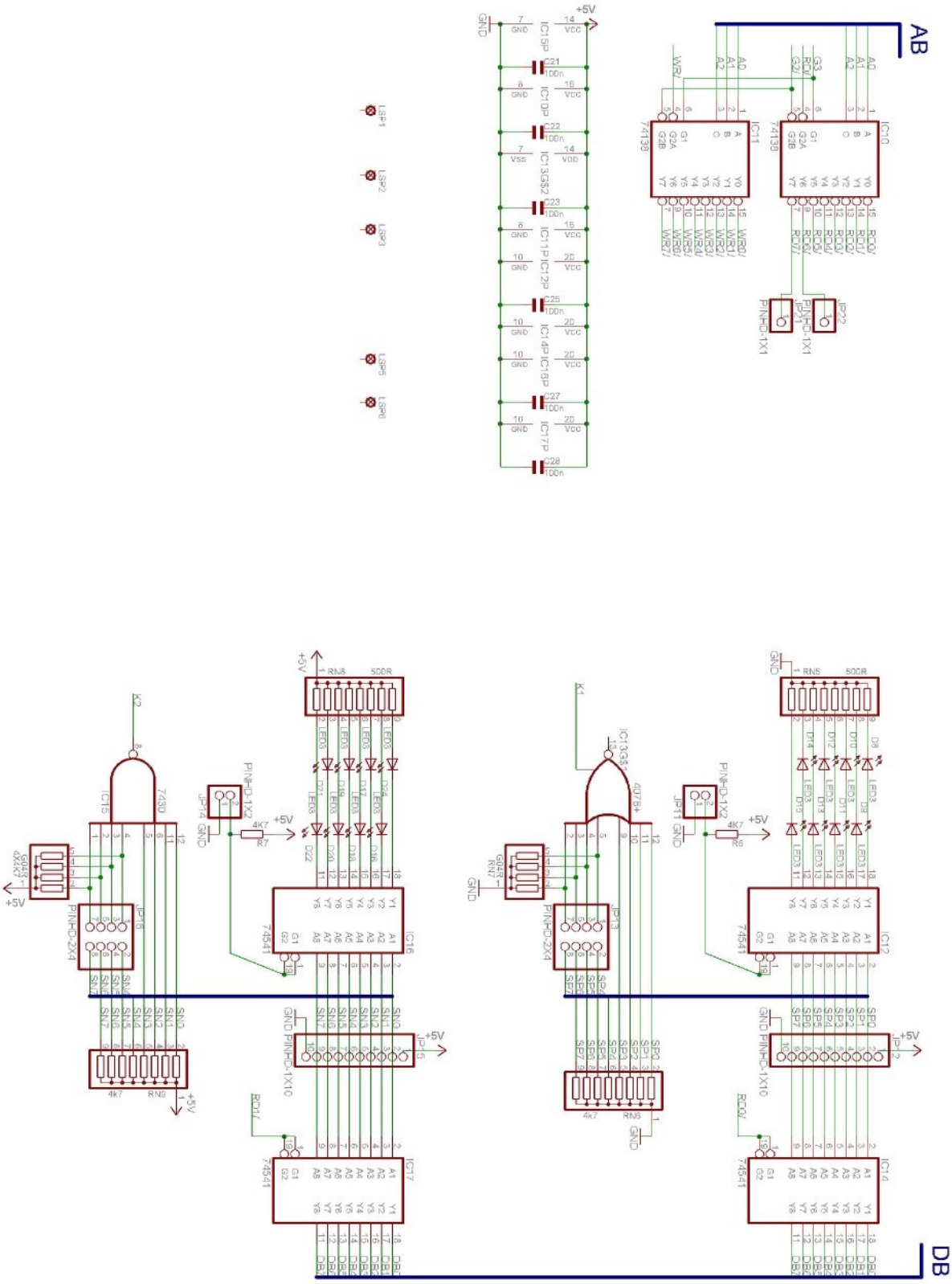
Sériová linka a stabilizace napětí

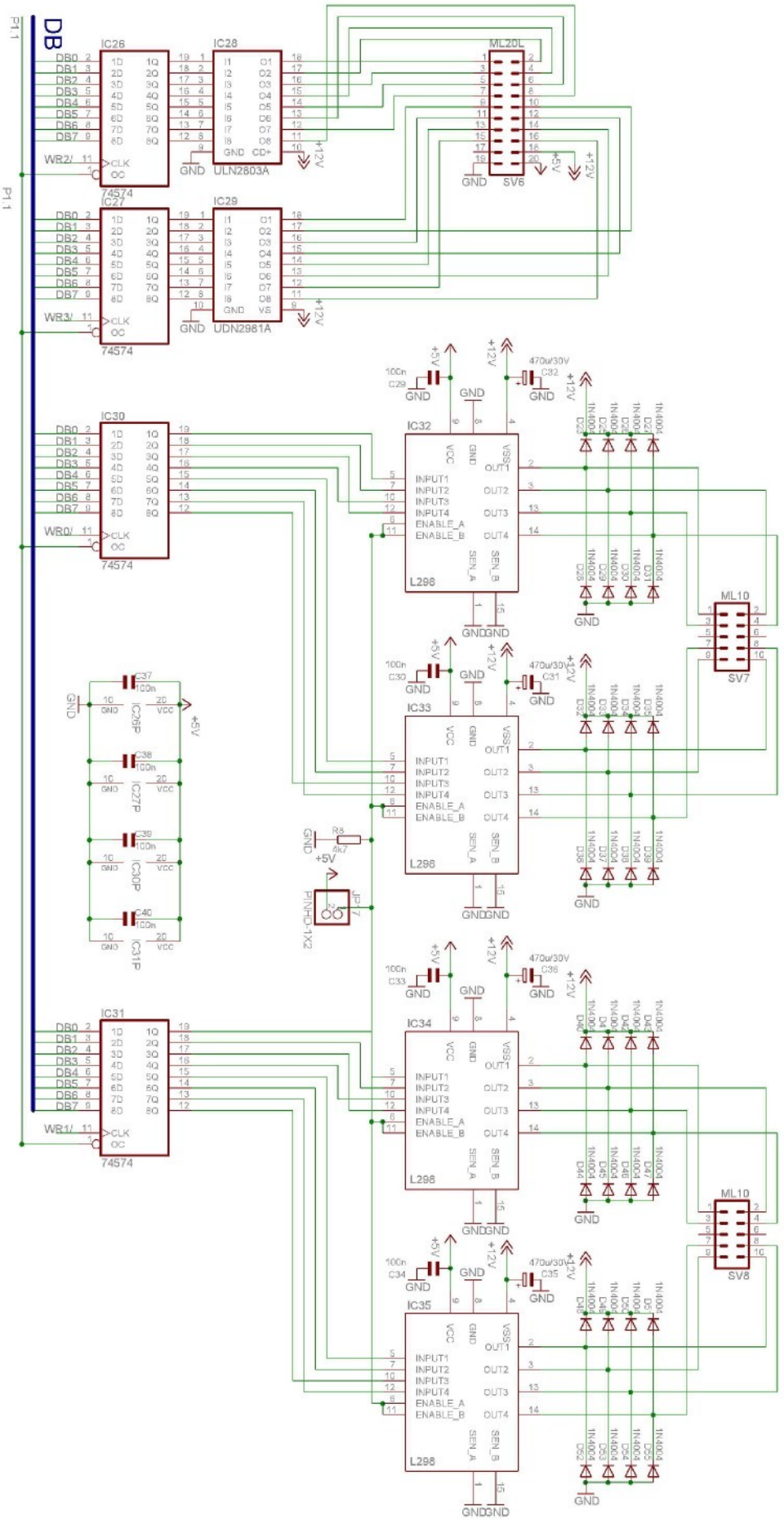


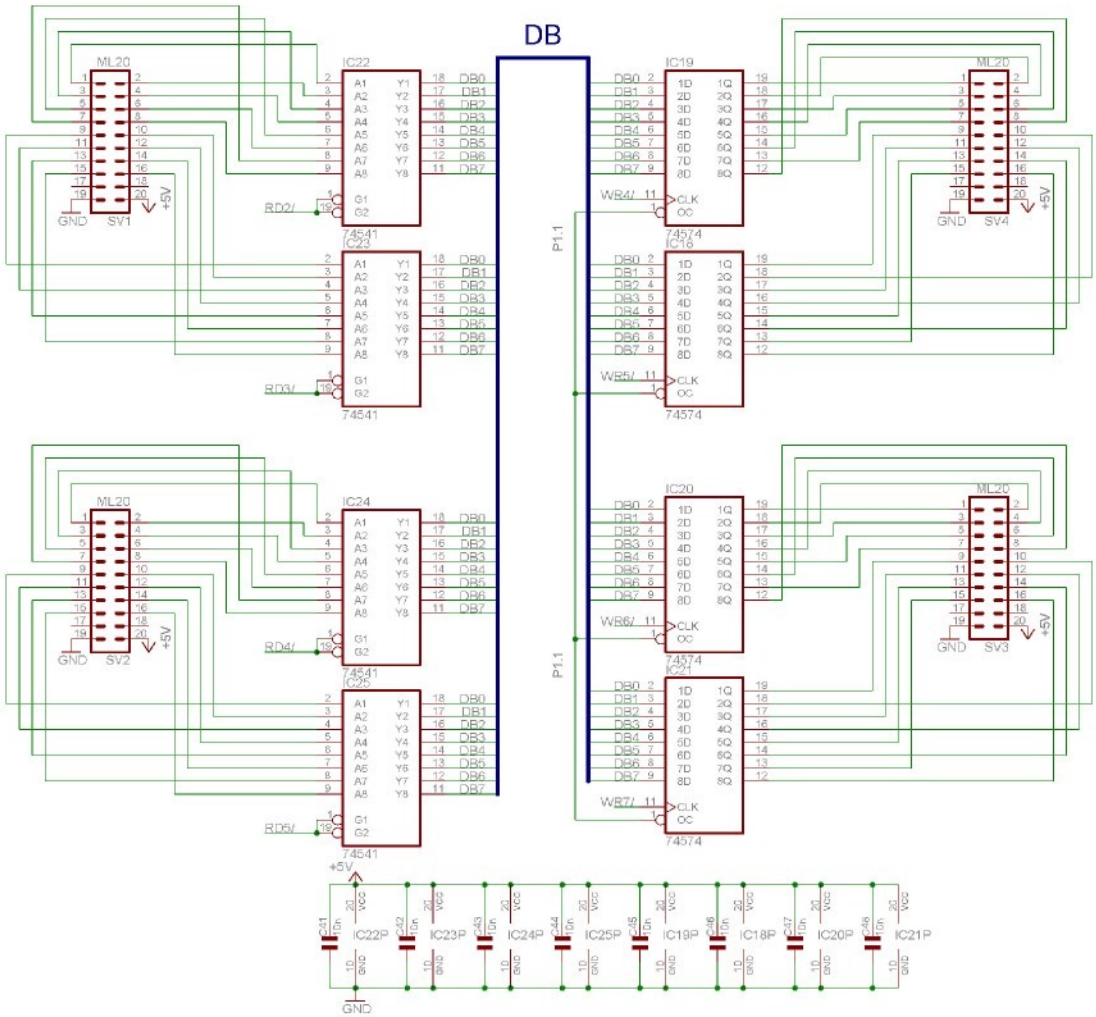
Výkonový budič L298 a jemu příslušný registr

Příloha 8 – schéma zapojení

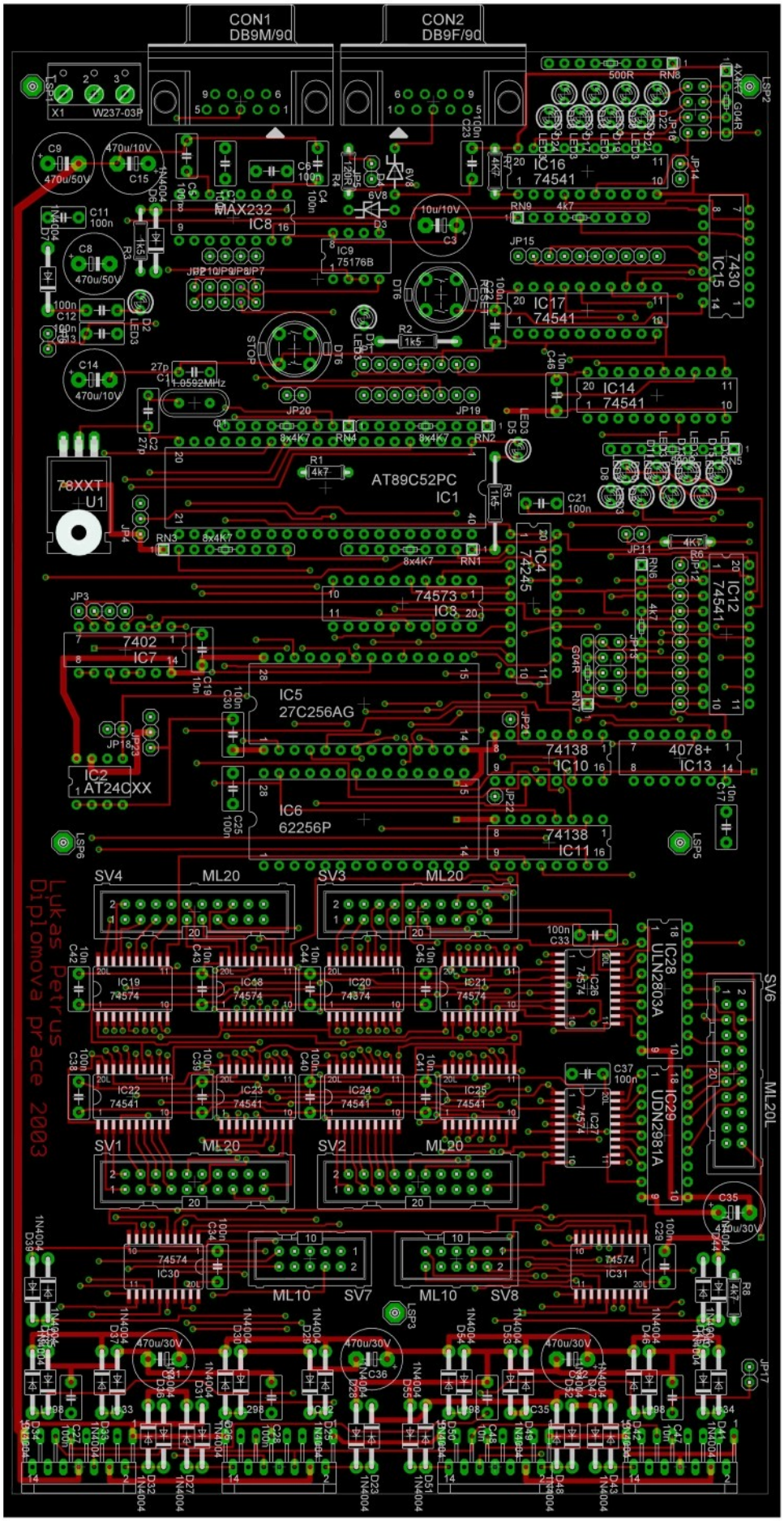

232
485
SPI
CON1
DB9M/90
CON2
DB9F/90
MAX232
AT89C52PC
AT24CXX
74573
27C256AG
62256P
74245
7402
78XXT
RESET
STOP
TX
AB
DB

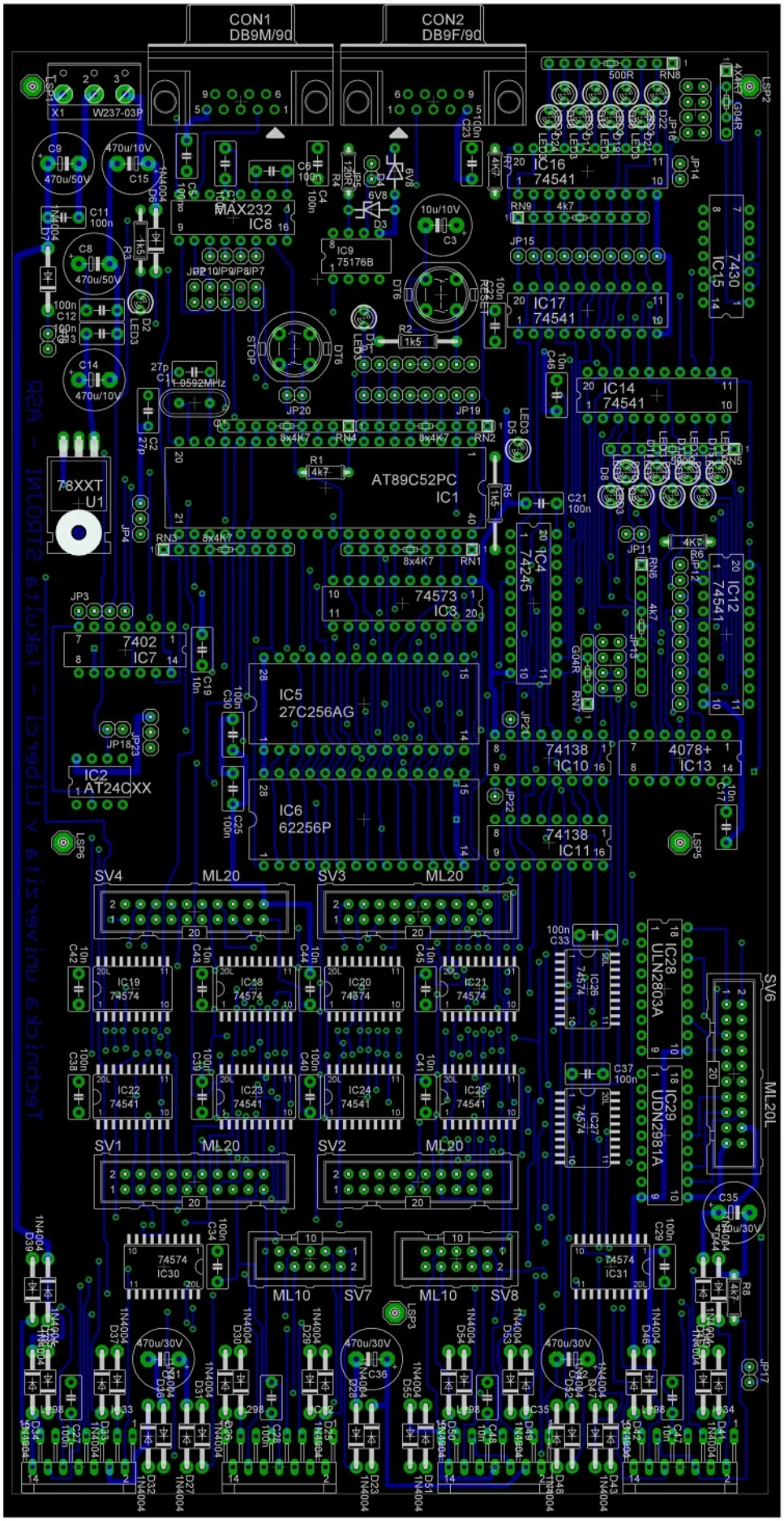

AB
DB
+5V
GND
74138
74541
4k7
PINHD-1X1
PINHD-1X2
PINHD-1X10
PINHD-2X4

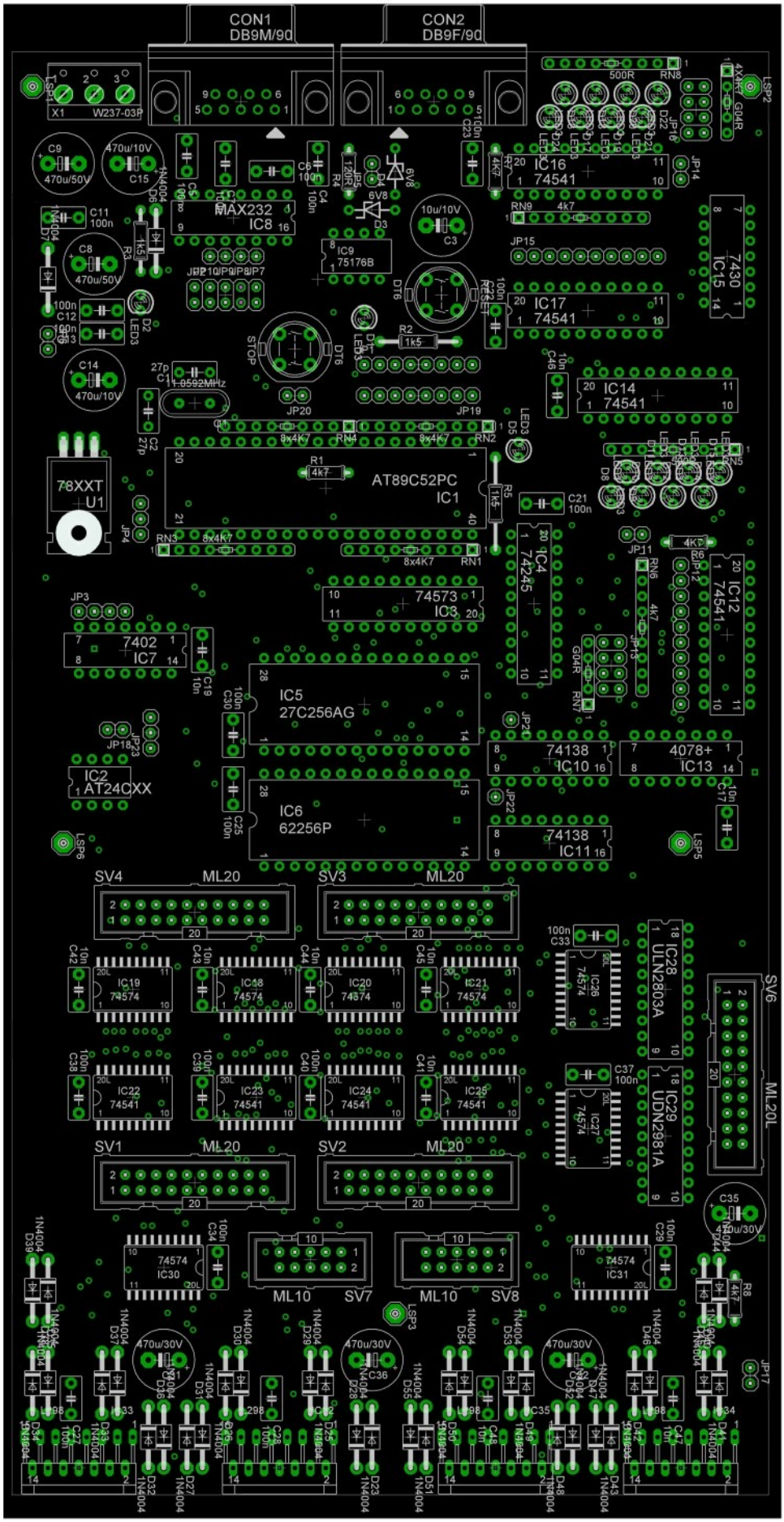

DB
P1.1
IC26
IC27
IC28
IC29
IC30
IC31
74574
ULN2803A
UDN2981A
ML20L
SV6
SV7
SV8
ML10
WR2/
WR3/
WR0/
WR1/
IC32
IC33
IC34
IC35
L298
INPUT1
INPUT2
INPUT3
INPUT4
ENABLE_A
ENABLE_B
OUT1
OUT2
OUT3
OUT4
SEN_A
SEN_B
VCC
GND
+5V
+12V
100n
470u/30V
1N4004
R8
6k7
JP17
PINHD-1X2
IC26P
IC27P
IC30P
IC31P

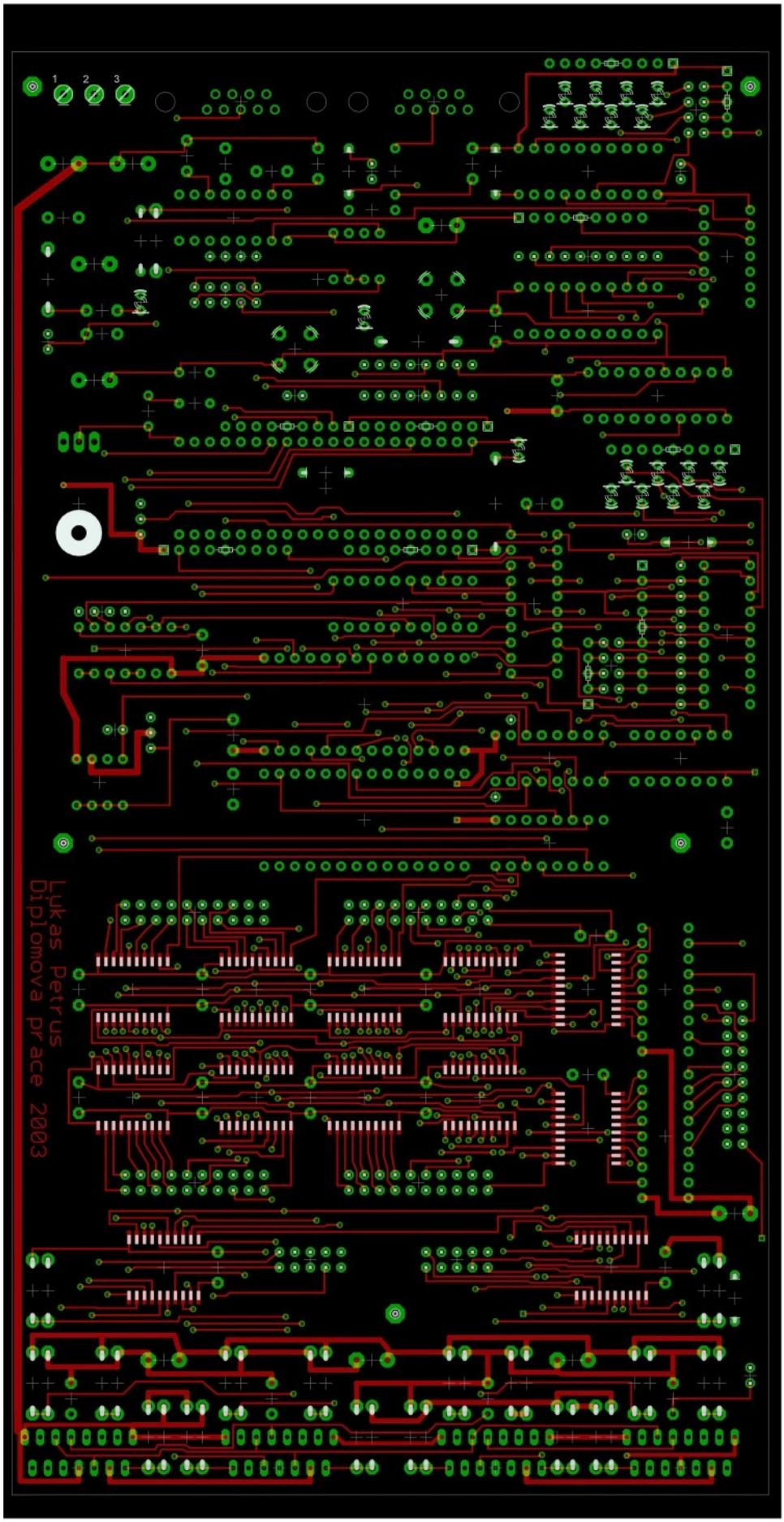

DB
ML20
SV1
SV2
SV3
SV4
GND
+5V
IC22
IC23
IC24
IC25
74541
RD2/
RD3/
RD4/
RD5/
IC19
IC18
IC20
IC21
74574
WR4/
WR5/
WR6/
WR7/
P1.1
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
IC21P
IC22P
IC23P
IC24P
IC25P
IC19P
IC18P
IC20P

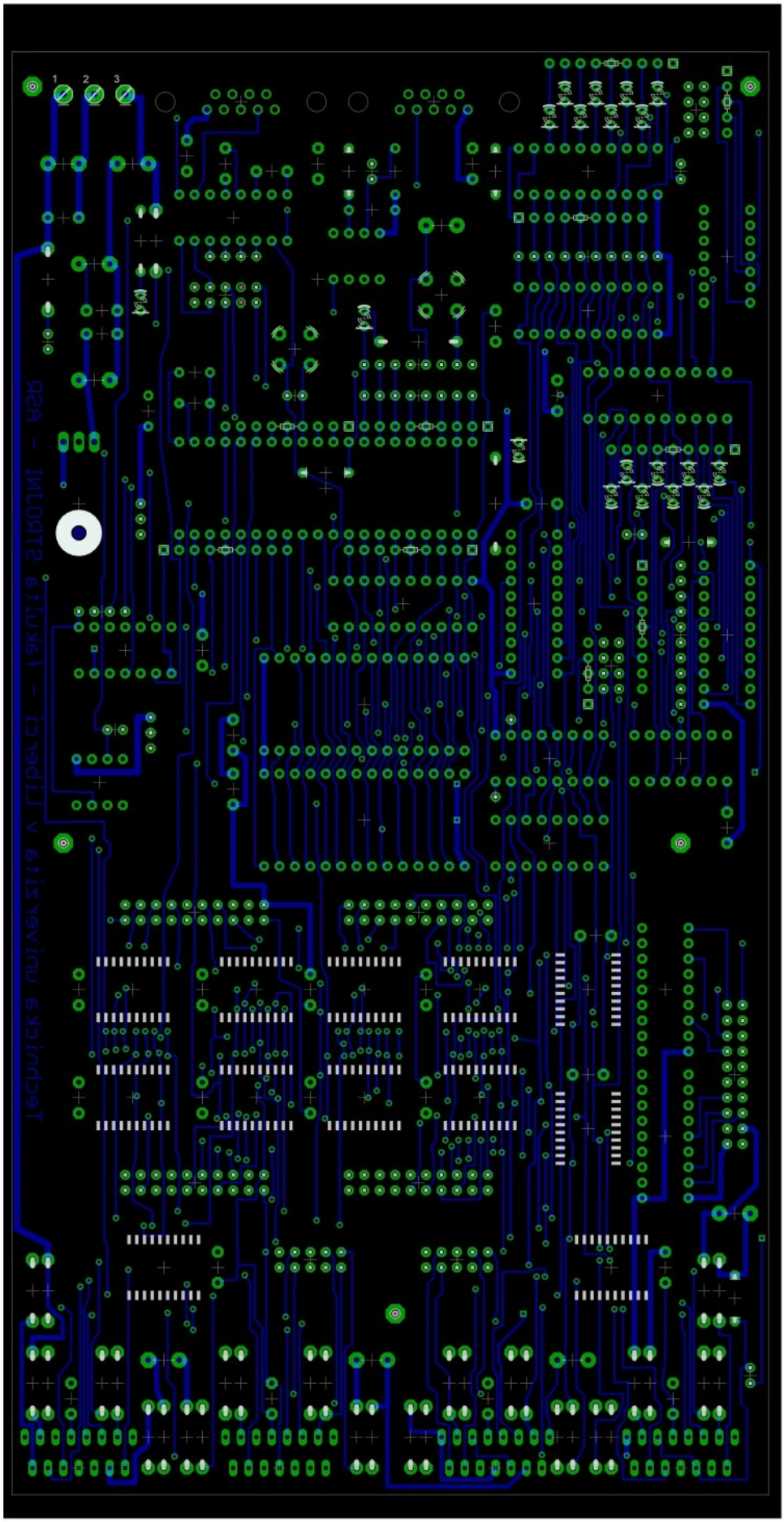
Příloha 9 – Řídící deska


CON1
DB9M/90
CON2
DB9F/90
X1
W237-03P
MAX232
IC8
IC9
75176B
AT89C52PC
IC1
78XXT
U1
IC16
74541
IC17
74541
IC14
74541
IC12
74541
IC15
7430
IC4
74245
74573
IC3
7402
IC7
IC5
27C256AG
IC6
62256P
IC2
AT24CXX
74138
IC10
74138
IC11
4078+
IC13
IC28
ULN2803A
IC29
UDN2981A
SV1
SV2
SV3
SV4
SV6
SV7
SV8
ML20
ML10
ML20L
Lukas Petrus
Diplomová práce 2003


CON1
DB9M/90
CON2
DB9F/90
X1
W237-03P
MAX232
IC8
IC9
75176B
IC16
74541
IC17
74541
IC14
74541
IC15
7430
IC12
74541
IC4
74245
IC3
74573
IC7
7402
IC5
27C256AG
IC6
62256P
IC10
74138
IC13
4078+
IC11
74138
IC2
AT24CXX
IC1
AT89C52PC
78XXT
U1
11.0592MHz
IC28
ULN2803A
IC29
UDN2981A
IC19
74574
IC18
74574
IC20
74574
IC21
74574
IC22
74541
IC23
74541
IC24
74541
IC25
74541
IC26
74574
IC27
74574
IC30
74574
IC31
74574
SV1
SV2
SV3
SV4
SV6
SV7
SV8
ML20
ML10
ML20L
STOP
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
LED9
LED10
LED11
LED12
LED13
LED14
LED15
LED16
LED17
LED18
LED19
LED20
LED21
LED22
LED23
LED24
470u/50V
470u/10V
470u/30V
10u/10V
1N4004
500R
4x4k7
8x4k7
4k7
1k5
120R
6V8
100n
10n
27p


CON1
DB9M/90
CON2
DB9F/90
X1
W237-03P
MAX232
IC8
IC9
75176B
AT89C52PC
IC1
78XXT
U1
11.0592MHz
IC16
74541
IC17
74541
IC14
74541
IC12
74541
IC4
74245
74573
IC3
7402
IC7
IC5
27C256AG
IC6
62256P
IC2
AT24CXX
74138
IC10
4078+
IC13
74138
IC11
7430
IC15
500R
RN8
4x4k7
RN9
4k7
RN4
8x4k7
RN2
RN3
RN1
RN6
RN7
IC28
ULN2803A
IC29
UDN2981A
IC19
74574
IC18
74574
IC20
74574
IC21
74574
IC22
74541
IC23
74541
IC24
74541
IC25
74541
IC26
74574
IC27
74574
IC30
74574
IC31
74574
SV1
SV2
SV3
SV4
SV6
SV7
SV8
ML20
ML10
470u/50V
470u/10V
10u/10V
470u/30V
1N4004
STOP
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
LED9
LED10
LED11
LED12
LED13
LED14
LED15
LED16
LED17
LED18
LED19
LED20
LED21
LED22
LED23
LED24


1
2
3
Lukas Petrus
Diplomová práce 2003


1
2
3