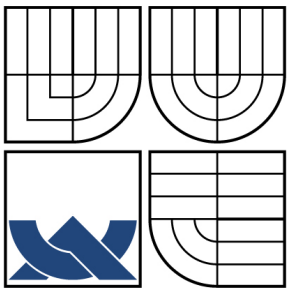
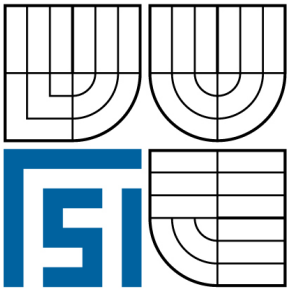


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

REALIZACE ŘÍDICÍHO SYSTÉMU PRO HYDRAULICKÝ MANIPULÁTOR

CONTROL SYSTEM DEVELOPMENT FOR HYDRAULICS MANIPULATOR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MAREK VOTAVA
AUTHOR

VEDOUCÍ PRÁCE — Ing. STANISLAV VĚCHET, Ph.D.
SUPERVISOR

BRNO 2009

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2008/2009

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Marek Votava

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Realizace řídicího systému pro hydraulický manipulátor

v anglickém jazyce:

Control system development for hydraulics manipulator

Stručná charakteristika problematiky úkolu:

Navrhněte strukturu elektronického řídicího sub-systému pro hydraulický manipulátor, který bude zprostředkovávat základní komunikaci mezi nadřazeným PC, hydraulickými aktuátory a sensorickou částí. V další fázi návrhu vytvořte řídicí software v prostředí NI LabView, který bude demonstrovat funkčnost systému jako celku.

Cíle diplomové práce:

Seznamte se s použitými senzory a aktuátory
Navrhněte vhodnou strukturu řídicího hardware
Navržený řídicí systém prakticky otestujte
Vytvořte nadřazený řídicí software v NI LabView

Seznam odborné literatury:

www.hydrocom.cz
www.hydroma.cz
www.atmel.com

Vedoucí diplomové práce: Ing. Stanislav Věchet, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2008/2009.

V Brně, dne 26.11.2008

L.S.

doc. RNDr. Ing. Miloš Šeda, Ph.D.	doc. RNDr. Miroslav Doupovec, CSc.
Ředitel ústavu	Děkan fakulty

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:

Bytem:

Narozen/a (datum a místo):

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta strojního inženýrství

se sídlem Technická 2896/2, 616 69 Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: _____

Vedoucí/ školitel VŠKP: _____

Ústav: _____

Datum obhajoby VŠKP: _____

VŠKP odevzdal autor nabyvateli v*:

- □ tištěné formě – počet exemplářů ……..
- □ elektronické formě – počet exemplářů ……..

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: …….

…….. ……
Nabyvatel Autor

ABSTRAKT

Tato diplomová práce se zabývá realizací řídicího systému pro hydraulický manipulátor. Je rozdělena do tří hlavních částí. První část je věnována senzorům a aktuátorům, které jsou použity na hydraulickém manipulátoru. Další část popisuje návrh, výrobu a programování elektronického řídicího sub-systému. V poslední části je popsán nadřazený řídicí software vytvořený v prostředí LabVIEW.

ABSTRACT

This diploma thesis concernes with the realization of a control system for hydraulic manipulator. It is devided into three main parts. The first part concerns sensors and actuators, which are used at the hydraulic manipulator. The second part describes design, production and programming of electronic control sub-system. The last part describes high level control software, created in the LabVIEW environment.

KLÍČOVÁ SLOVA

Řídicí systém, hydraulický manipulátor, ATmega16, LabVIEW.

KEYWORDS

Control system, hydraulic manipulator, ATmega16, LabVIEW.

PODĚKOVÁNÍ

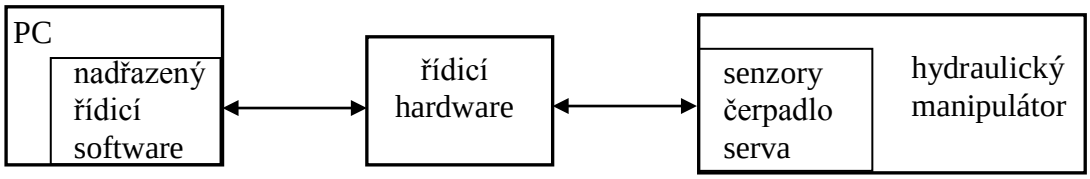
Děkuji panu Ing. Stanislavu Věchetovi, Ph.D., vedoucímu mé diplomové práce, za čas a cenné rady, které mi věnoval. Dále chci tímto poděkovat svým rodičům za podporu během celého mého studia.

Obsah:

1 ÚVOD

Hydraulicky ovládané mechanismy jsou dnes velice rozšířené a mají velké uplatnění především v průmyslu. Použití zvyšuje stupeň mechanizace a automatizace celého systému. Jejich výhoda spočívá v síle, kterou jsou schopny oproti ostatním mechanismům vyvinout. Proto se hydraulika uplatňuje především u manipulační techniky, jako je například vysokozdvižný vozík. V průmyslu je asi nejvíce zastoupena v oboru stavebnictví, dopravě, zemědělství. Podstatou celého mechanismu je využití tlaku přenosového média. Hydraulická zařízení mají médium v podobě kapaliny, nejčastěji oleje. V principu nejjednodušší tekutinový mechanismus je píst ve válci, který je náležitě utěsněn. Síla, kterou disponuje, je závislá nejen na konstrukčním provedení, ale také na ostatních součástech celého systému. Tyto mechanismy se stále vyvíjí a jsou doplňovány o prvky z jiných oblastí [2]. Mezi takové patří především řídicí elektronika v podobě snímačů a čidel, které umožní tyto mechanismy řídit a získávat, popř. zobrazovat, různé údaje pomocí počítačů.

Cílem této práce je návrh a výroba právě takovéto řídicí elektroniky. Vytvořený elektronický sub-systém zprostředkovává základní komunikaci mezi nadřazeným počítačem (PC), hydraulickými aktuátory a senzorickou částí. Dále byl podle zadání vytvořen v prostředí LabVIEW nadřazený řídicí software, který demonstruje funkčnost systému jako celku.



Obr. 1: Schéma komunikace jednotlivých částí systému.

Práce je rozčleněna do několika hlavních kapitol. První kapitolu tvoří úvod, po kterém následuje druhá kapitola, jež seznamuje s prvky hydrauliky a senzory, které jsou použity na hydraulickém manipulátoru, pro který byl řídicí systém navržen. Ve třetí kapitole je pak detailně popsán řídicí hardware a je v ní také uveden postup návrhu a výroby řídicí elektroniky. Dále pak seznámení s mikrokontrolérem ATmega16, který je nejdůležitější součástkou hardwarové části. Do této kapitoly je také zařazen popis programu mikrokontroléru. U návrhu desky plošných spojů a u programování mikrokontroléru je také seznámení s programy, které byly použity. Čtvrtá kapitola pojednává o vytvořeném nadřazeném softwaru a o prostředí LabVIEW, ve kterém byl tento software vytvořen. Poslední kapitolou je závěr, kde jsou shrnuty výsledky práce.

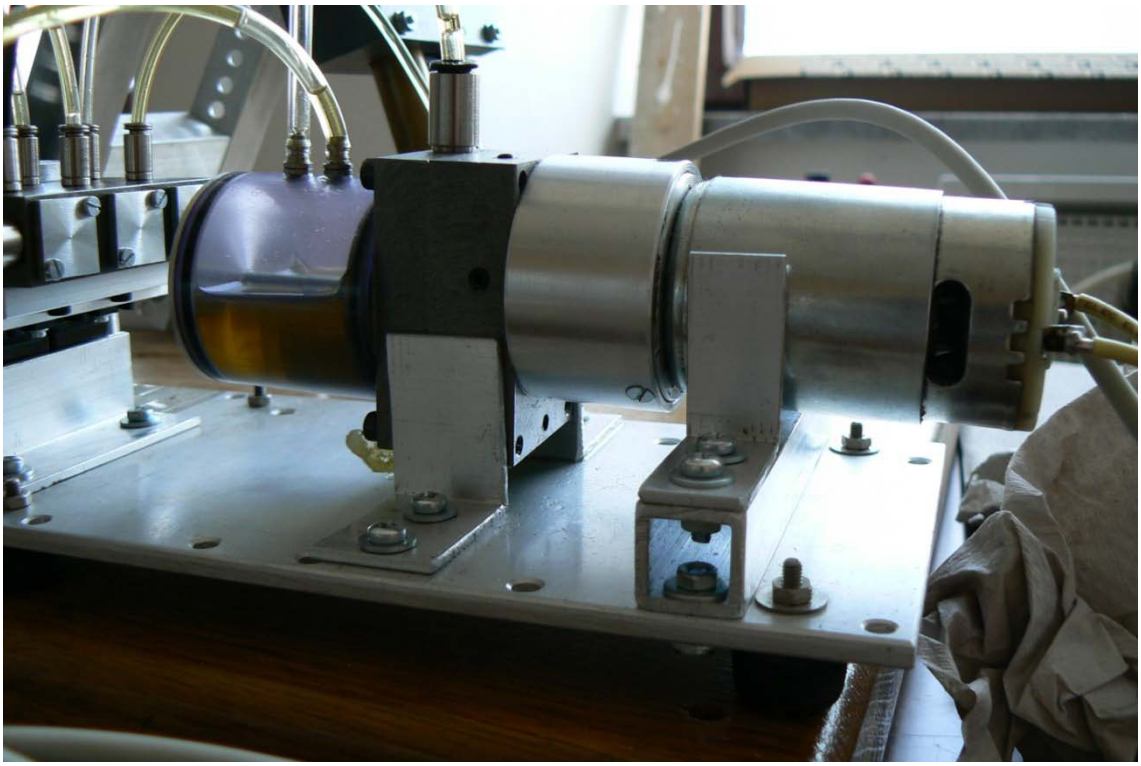
2 HYDRAULICKÁ A SENZORICKÁ ČÁST MANIPULÁTORU

Hydraulická část manipulátoru je tvořena hydraulickými prvky dodanými firmou Š-Hobby z Hradce Králové, která vyráběla vlastní modelářskou hydrauliku. Výroba v současné době již nepokračuje a firma není schopna poskytnout bližší technické specifikace jednotlivých komponent. Dále jsou tedy uvedeny jen základní zjištěné údaje. Schéma celého hydraulického obvodu je v příloze č. 1.

2.1 Hydrogenerátor

Hydrogenerátor neboli čerpadlo slouží k přeměně mechanické energie na tlakovou energii kapaliny. Zdrojem mechanické energie je elektromotor. Hydrogenerátory jsou mechanismy s rotační součástí, která svým pohybem vyvolává tlak v kapalině.

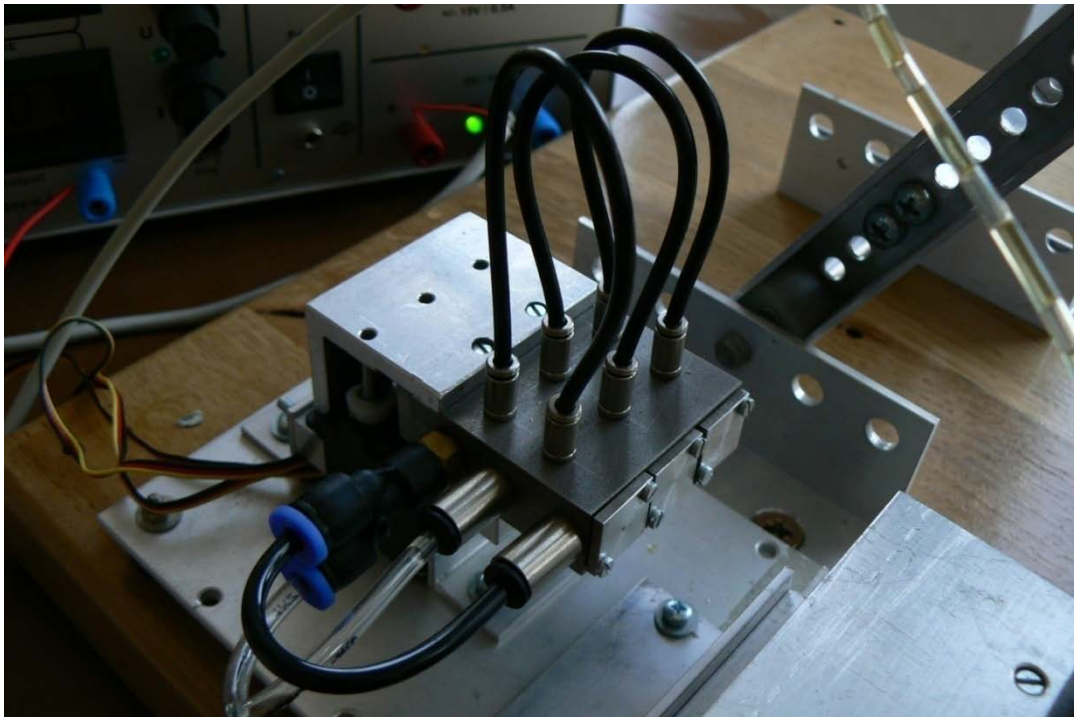
Použité modelářské čerpadlo má při obvyklé konfiguraci průtok 180 cm^3 za minutu. Maximální tlak tohoto čerpadla je 0,36 MPa. Objem nádrže je cca 45 ml.



Obr. 2: Čerpadlo od firmy Š-Hobby [13].

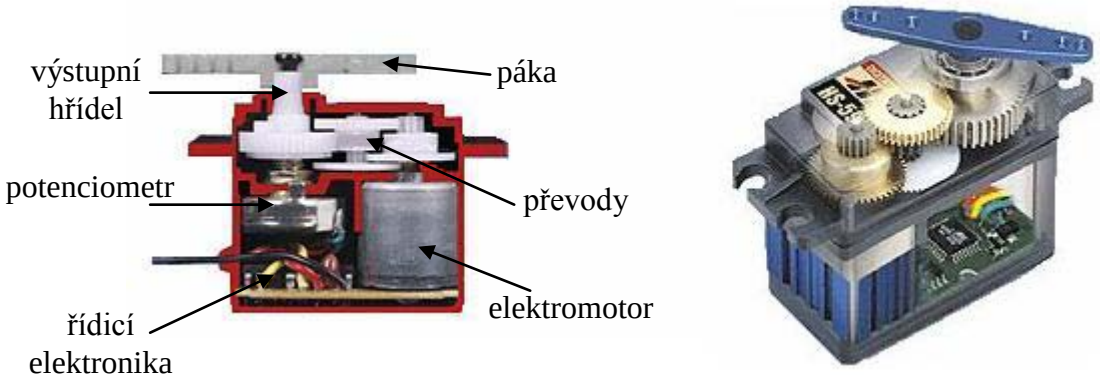
2.2 Ovládací blok hydrauliky

Ovládací blok hydrauliky tvoří proporcionální rozvaděč typu 5/2 (5 přípojů, 2 funkční stavy), který je vyroben z duralu a modelářská serva. Serva slouží k ovládání rozvaděče.



Obr. 3: Ovládací blok hydrauliky [13].

Dnešní modelářská serva obsahují elektromotor s převodovkou a řídicí elektroniku. Do vstupu řídicí elektroniky přichází řídicí impuls, který spustí monostabilní klopný obvod. Ten vygeneruje impuls o délce odpovídající momentální poloze serva a opačné polarity než je vstupní řídicí impuls [3]. Tyto dva impulsy se porovnají a výsledkem je rozdílový impuls, který po zesílení přes můstkový spínač způsobí roztočení elektromotoru jedním nebo druhým směrem. Elektromotor přes převodovku otáčí výstupní hřídelí a současně i potenciometrem, který působí jako zpětná vazba do monostabilního klopného obvodu. Směr otáčení je takový, že impuls generovaný monostabilním klopným obvodem se svojí délkou přibližuje délce vstupního řídicího impulsu a až jsou oba impulsy stejně dlouhé, elektromotor se zastaví. Servo dosáhlo polohy, která odpovídá momentálně přijímanému řídicímu impulsu. Jakým způsobem se generují řídicí impulsy pro serva je popsáno v kapitole 3.6.2.



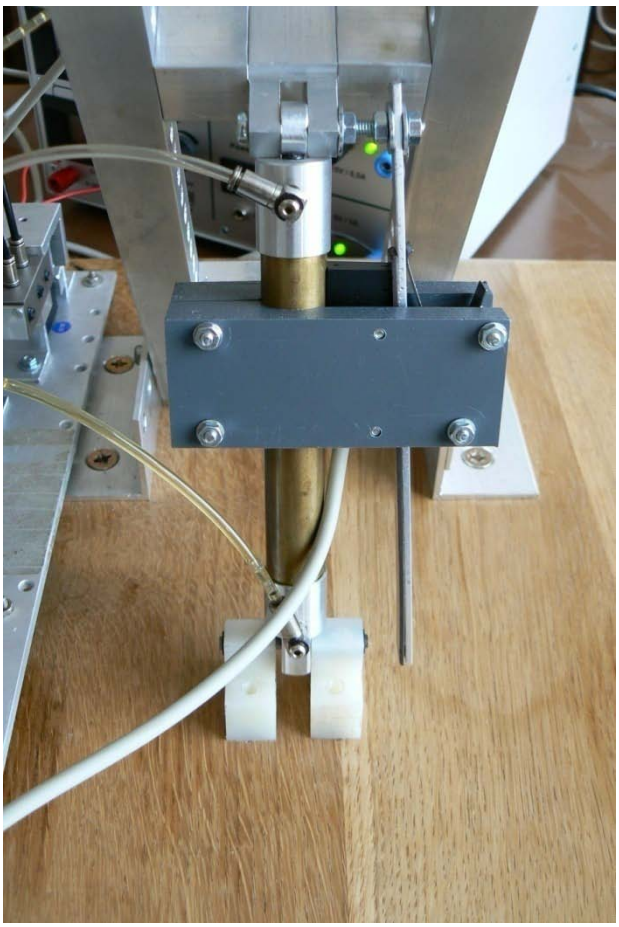
Obr. 4: Modelářské servo [4].

2.3 Hydromotory

Od firmy Š-Hobby byly zakoupeny i modelářské lineární hydromotory 16-145/4 s následujícími parametry:

- délka v zasunutém stavu: 145 mm
- průměr upevňovacích otvorů: 4 mm
- maximální zdvih: 120 mm
- průměr válce: 16 mm
- maximální tlak: 2 MPa
- síla vysunování: 153 N (při tlaku 1 MPa)
- síla zasunování: 125 N (při tlaku 1 MPa)

Konce pro připojení hadic jsou rotační. Přívodní hydraulické trubice se upevňují pouze zasunutím do rotačního pouzdra.

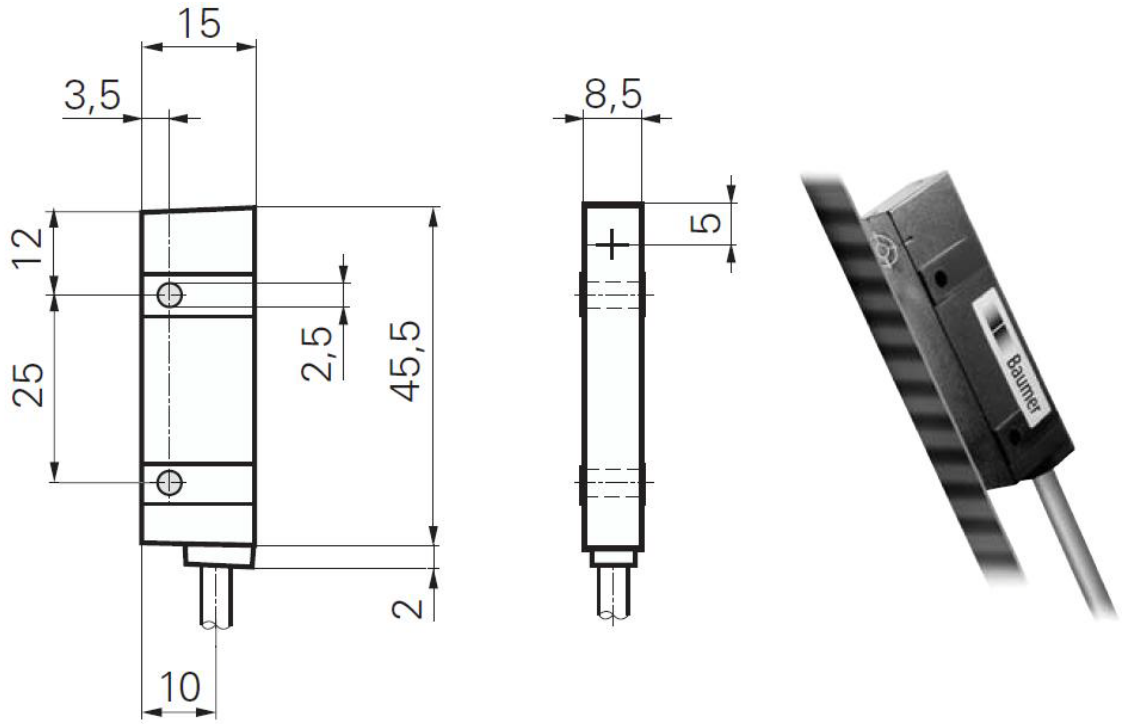


Obr. 5: Hydromotor upevněný na hydraulickém manipulátoru [13].

2.4 Senzory

Jako snímače polohy jsou použity magnetické lineární snímače MLFK 08T7101 od firmy Baumer. Jedná se o relativní snímače s těmito parametry:

- napájecí napětí 5V ± 5%
- ochrana proti zkratu
- maximální rychlost posuvu 40 m/s
- 25 pulsů na 1 mm
- pracovní teplota -25 °C až + 85 °C
- vzduchová mezera max. 1 mm
- vnější rozměry: 45,5x8,5x15 mm
- plastové pouzdro



Obr. 6: Rozměry magnetického snímače MLFK 08T7101 od firmy Baumer[5].

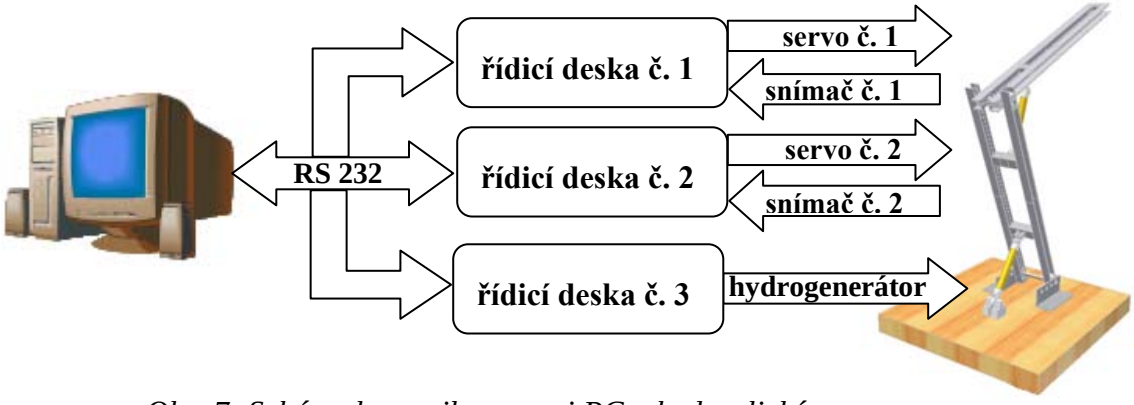
Při pohybu snímače nad magnetickým páskem se začnou na výstupu snímače generovat impulsy. Jak tyto impulsy vypadají a jak se zpracovávají, je popsáno v kapitole 3.6.2.

3 ŘÍDICÍ HARDWARE

3.1 Princip ovládání hydraulického ramene

Nadřazený řídicí software, který bude popsán v kapitole 4, vysílá data z PC po sériové lince. Tyto data přichází do řídicí elektroniky. Řídicí elektronika se skládá ze tří řídicích desek. Přijatá data každý obvod zpracuje. Deska, která vyhodnotí, že přijatá data byla určena jí, provede přijaté instrukce a pošle odpověď zpět do PC. Ostatní desky nijak nereagují. Každá deska obsluhuje konkrétní zařízení a to:

- Deska č. 1 (master) – servo č. 1 a snímač č. 1
- Deska č. 2 (slave) – servo č. 2 a snímač č. 2
- Deska č. 3 (slave) - hydrogenerátor



Obr. 7: Schéma komunikace mezi PC a hydraulickým ramenem.

Master deska se liší od ostatních desek tím, že kromě obsluhování svých zařízení reaguje i na neznámou akci a na chybu v přenosu.

Dále bude popsán návrh a výroba řídicích desek. Poté navazuje kapitola, která se věnuje mikrokontroléru ATmega16, kterým je každá řídicí deska osazena. Na závěr jsou popsány programy nahrané v mikrokontrolérech.

3.2 Návrhový software EAGLE 5.4.0

Pro návrh desky plošných spojů byl použit program EAGLE. Konkrétně EAGLE Light Edition verze 5.4.0. Tuto verzi nabízí firma CadSoft zcela zdarma. EAGLE je uživatelsky přívětivý a výkonný nástroj pro návrh desek plošných spojů (DPS). Název EAGLE je zkratka, pocházející z původního názvu Easily Applicable Graphical Layout Editor [1]. Program se skládá ze tří hlavních modulů:

- Editor schémat
- Editor spojů
- Autorouter

Moduly jsou ovládány z jednoho uživatelského prostředí prostřednictvím *Control Panelu*. *Control panel* dále umožňuje určit umístění souborů a provádět jejich správu dle požadavků uživatele.

3.2.1 Vlastnosti návrhového systému

Společné [6]:

- dopředná a zpětná anotace v reálném čase
- nápověda orientovaná podle obsahu
- žádná hardwarová ochrana programu
- vícenásobná okna pro desku, schéma a knihovnu
- výkonný uživatelský jazyk
- integrovaný textový editor
- dostupný pro Windows 95/98/NT4/2000 a Linux

Editor spojů:

- největší rozměr výkresu 1,6 x 1,6 m (Light verze 100 x 80 mm)
- rozlišení 1/10 000 mm (0,1 mikronu)
- až 16 signálových vrstev (Light verze pouze 2)
- klasické i SMD součástky
- dodává se s plnou sadou knihoven součástek
- snadné vytváření vlastních součástek v plně integrovaném editoru knihoven
- funkce vpřed/vzad pro libovolný editační příkaz, do libovolné hloubky
- pomědění ploch = rozlévání mědi
- funkce kopírovat a vložit pro kopírování kompletních částí výkresu
- kontrola pravidel návrhu

Editor schémat:

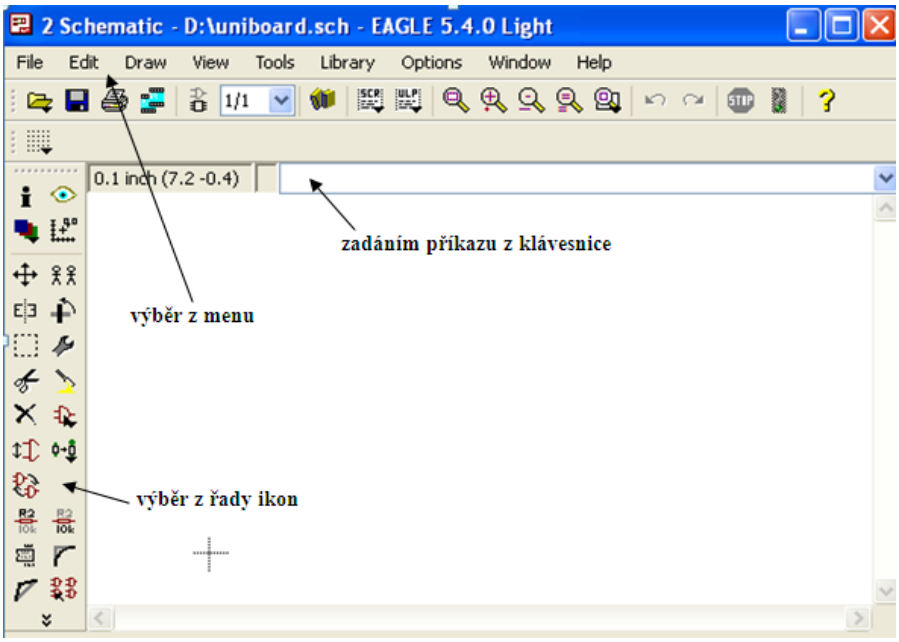
- až 99 listů jednoho schématu (Light verze pouze jeden list)
- kontrola elektrických pravidel zapojení
- prohazování hradel a pinů
- vytvoření desky ze schématu jediným příkazem

Autorouter

- až 16 signálových vrstev
- strategie propojování nastavitelná uživatelem pomocí váhových faktorů

3.2.2 Editor schémat

Program EAGLE umožňuje příkazy zadávat několika způsoby, jak je znázorněno na obrázku 8. Všechny tyto možnosti zadání příkazů jsou naprosto rovnocenné.



Obr. 8: Prostředí editoru schémat.

Před návrhem schématu je potřeba zkontrolovat několik nastavení. Nejdůležitější je zkontrolování nastaveného rastru, který by měl být vždy ponechán v tzv. „defaultním“ nastavení, kdy se rastr rovná 2,54 mm. Pokud by byl rastr nastaven na jinou hodnotu, mohlo by dojít k problému při pájení součástek.

Součástky se vkládají pomocí příkazu *Add*, po jehož zadání se otevře okno s obsahem naposledy použité knihovny. Pokud je potřeba prvek z jiné knihovny, vybere se příkazem *Use*. Z knihovny se pak vybere prvek, který má být umístěn do schématu.

Při propojování jednotlivých součástek může nastat problém, že se součástka propojí s jinou, než s jakou se propojit měla. Kontrolu správnosti propojení lze provést několika způsoby. Součástky se propojují pomocí funkce *Net* a správné propojení lze překontrolovat přesunutím součástky – čáry se přesunují také. K další optické kontrole je určen příkaz *Show*. Po kliknutí na spoje se vše, co je spojeno „prosvítí“, a to včetně pinů součástek. Také je možné provést elektrickou kontrolu, k níž slouží příkaz *Erc*, který zkontroluje zapojení z hlediska zásad správného „elektrického“ návrhu. Každý pin součástky má určitý atribut např. *Pas* - pasivní pin, *Out* - výstupní pin, *Pwr* - napájecí pin, atd. Na základě těchto atributů *Erc* zjišťuje, zda jsou vzájemně spojeny piny, jejichž vlastnosti jim to povolují, dále pak zda ve schématu nejsou některé piny nezapojeny a zda je napájení obvodu provedeno korektně. Na případné prohřešky program upozorní výčtem chyb, který uloží do souboru s příponou *.erc, a automaticky jej zobrazí. Tato kontrola nezohledňuje všechny korektní možnosti zapojení součástek v obvodu, má tedy především informativní charakter a jejím hlavním úkolem je upozornit na možné chyby. Pokud je vše vytvořeno a zkontrolováno, pak ze schematického editoru může být vytvořena deska plošných spojů kliknutím na tlačítko *Board*. Vytvořené schéma řídicí desky je v příloze č. 2.

3.2.3 Editor plošného spoje

Po zadání příkazu *Board* v editoru schémat se automaticky spustí editor desky. Z knihoven se vyberou pouzdra součástek, jejichž plošky se propojí vzdušnými spoji podle sítě spojů definovaných ve schématu a vše se automaticky umístí na pracovní plochu editoru.

Od verze 3.5 a vyšší funguje tzv. zpětná anotace. To znamená, že změna názvu nebo hodnoty součástky provedená v jednom editoru se ihned promítne i do druhého. Pokud je ale nutné provést změnu v zapojení, je třeba se vrátit do schématu, změnu provést a úprava se automaticky přenese i do desky. Změnu zapojení přímo v desce editor nedovolí provést. Program takto hlídá integritu mezi schématem a deskou. Tato vazba samozřejmě funguje pouze tehdy, jsou-li oba editory spuštěny.

První, co je při práci v tomto editoru nutné nakreslit, je rámeček, který definuje velikost desky. Ve verzi, kterou jsem použil, je velikost rámečku omezena na 100 x 80 mm, což pro mnou navrženou desku bylo dostačující. Nakreslení rámečku se provede funkcí *Wire* ve vrstvě *20 Dimension*.

Dále je nutné zkontrolovat pouzdra, která nám automaticky program vybral z knihoven. Často se stalo, že bylo k dispozici jiné provedení součástek, než jsem si zvolil ve schématu. Před rozmístěním součástek je tedy vhodné provést záměnu pouzder na požadované typy. Příkazem *Replace* se zvolí z otevřené knihovny pouzdro (pokud se požadované pouzdro nachází v jiné knihovně, tak se otevře příkazem *Use*). Poté se kliknutím myši na „závěsný“ bod pouzdra provede záměna. Náhradu lze provést pouze tehdy, jsou-li plošky pouzder stejně pojmenovány a je-li počet plošek nového pouzdra stejný nebo vyšší.

Dalším krokem je rozmístění součástek. Součástky se rozmístí na desku pomocí příkazu *Move*. Při rozmisťování je dobré používat příkaz pro optimalizaci vzdušných spojů *Ratsnest*, aby se dalo lépe orientovat podle jejich délky. V knihovnách jsou standardně pouzdra definována pro umístění do vrstvy *1 Top*. Pro umístění pouzdra ze strany spojů (vrstva *16 Bottom*) se používá příkaz *Mirror*. Tohoto příkazu bylo potřeba při umisťování mikroprocesoru, SMD odporů a kondenzátorů.

Záměna vzdušného spoje na spoj se provede pomocí příkazu *Route*. Pak se klikne levým tlačítkem myši na plošku, ze které má spoj vést. Úhel zalomení se mění pravým tlačítkem myši. Spoj se ukončí při dotažení spoje na plošku. Pro „pokládání“ spojů jsou určeny vrstvy 1 až 16. Editor desky tedy umožňuje vytvářet až šestnáctivrstvé plošné spoje. Pro jednostranné desky se používá pouze vrstva *16 Bottom*. Pro dvoustranné desky se používají vrstvy *1 Top* a *16 Bottom*.

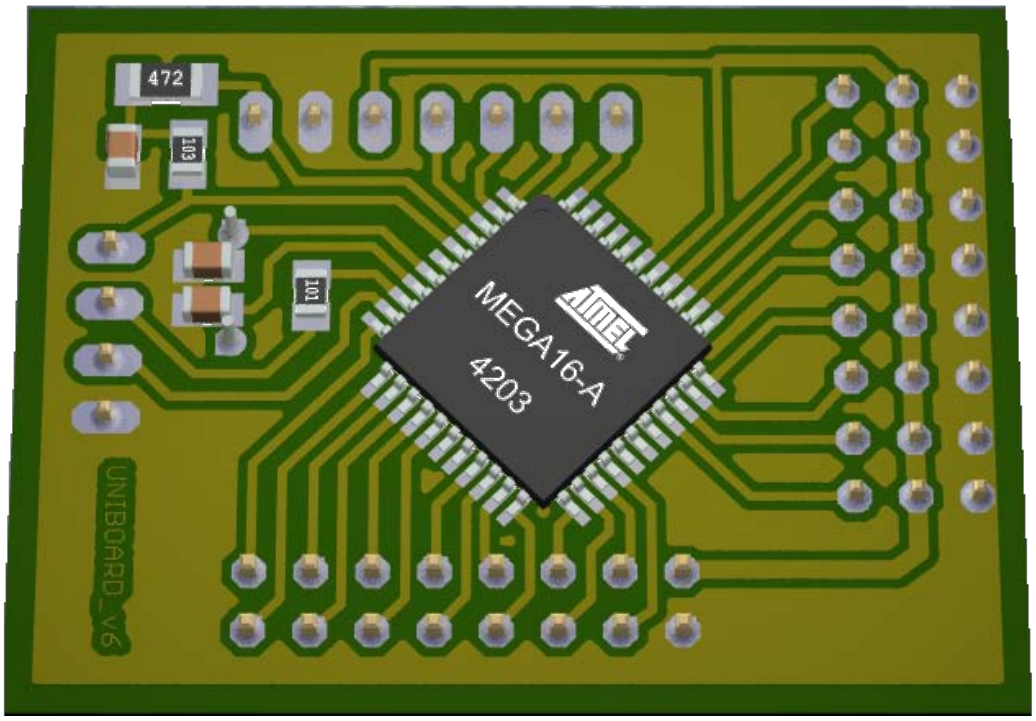
Pomocí polygonu se může vytvořit souvislá plocha, která „obteče“ spoje s rozdílným potenciálem. Po zadání příkazu *Polygon* se nejdříve zvolí vrstva, ve které má být stínění umístěno a vyznačí se obvod plochy, která má být vyplněna. Při této akci není nutné vyhýbat se žádným překážkám (spoje, plošky, pouzdra), neboť program se všem překážkám při vyplnění polygonu vyhne sám. Tvorba polygonu se ukončí kliknutím na bod jeho počátku. V mém případě jsem chtěl, aby měla vyplněná plocha stejný potenciál jako GND. Tudíž jsem polygon pomocí příkazu *Name* pojmenoval stejně jako vodič GND, se kterým má být spojen.

Stejně jako v editoru schémat je i v editoru spojů možnost kontroly. Dodržení technologických parametrů nutných pro výrobu desky je možné zkontrolovat příkazem *Drc*. V dialogovém okně *Drc* je možné volit parametry, které budou kontrolovány. Seznam chyb lze zobrazit pomocí příkazu *Errors*, navíc se vyznačí na desce šrafováním. Po opravě se označení chyb zruší tlačítkem *Clear* v dialogovém okně *Drc*.

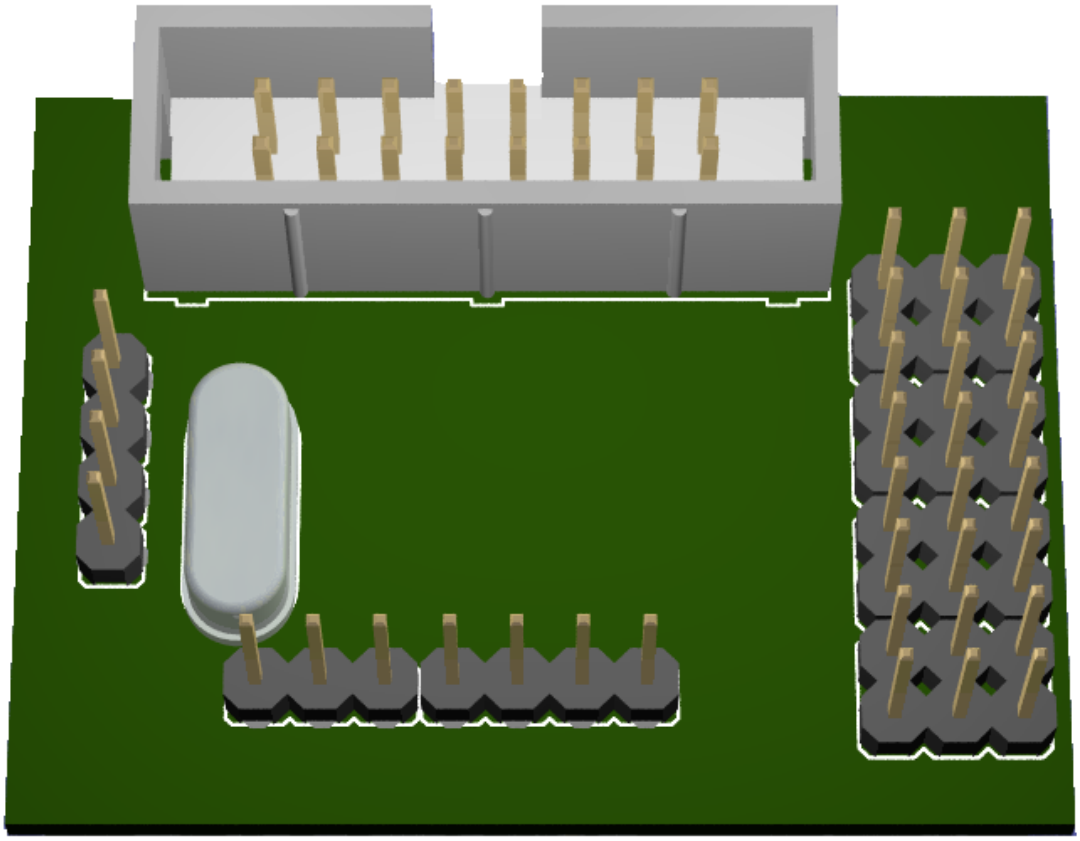
Posledním krokem je výstup z programu. Kromě výstupů, které jsou obsaženy v CAM Procesoru (tiskárna, plotter, vrtačka, TIF-obrázek), se dá použít funkce *Export*, ve které lze vybrat např. *partlist*, který tvoří seznam součástek schématu nebo desky. Seznam použitých součástek je v příloze č. 3.

3.2.4 Autorouter

Poslední ze tří modulů je autorouter, který slouží k automatickému vygenerování všech cest na DPS. Cesty se tedy nevytváří „ručně“, ale automaticky. Pouze se nastaví různé specifikace, například šířka cest, vzdálenost mezi cestami, atd. Já tuto funkci nevyužil a celý návrh jsem udělal sám ručně.



Obr. 9: 3D model DPS ze strany spojů.



Obr. 10: 3D model DPS ze strany součástek.

3.3 Výroba desky plošných spojů

Existuje několik způsobů jak vyrobit desku plošných spojů (DPS). Já zvolil způsob nažehlení toneru na DPS. Jedná se o rychlý, levný a technicky nenáročný způsob zejména pokud si DPS vyrábíme sami a jen v malém počtu kusů. Skoro vše, co je k výrobě desky potřeba, lze koupit v obchodě s elektrosoučástkami, papírnictví a drogerii:

- barevné lepící papíry
- jednostranný cuprextit tloušťky 1,5 mm
- vrtačka na plošné spoje
- plochá miska
- smirkový papír
- izolepa
- žehlička
- kyselina chlorovodíková
- peroxid vodíku
- laserová tiskárna

Prvním krokem při výrobě je zkušební tisk, který se provede na čistý kancelářský papír. Vytiskne se tedy předloha, která byla navržena v program EAGLE. Tiskárna se před tiskem nastaví na nejvyšší sytost tisku, aby bylo toneru na papíře co nejvíce. Přes tento vytištěný obraz návrhu plošného spoje, s dostatečným okrajem (alespoň 1cm), se přelepí lepicí páskou barevný lepící papír a to lepivou stranou nahoru.

Za sucha tato strana samozřejmě nelepí, v tiskárně se nenatavuje. Poté se znovu vloží papír s nalepeným kusem barevného papíru do tiskárny a opět se vytiskne předloha [7]. Motiv se vytiskne na stejné místo, ale tentokrát na lepící papír. Poté se lepící barevný papír s vytištěným motivem na lepící straně odstřihne nebo odlepí.

Zásadní výhodou použitého papíru je, že natištěný motiv na něm pevně drží a nestírá se, což se při použití fólií stávalo již při průchodu papíru tiskárnou. Asi nejdůležitější částí výroby je nažehlení. Vytištěný motiv na lepícím papíru se položí na rovnou podložku, motivem navrch. Na tento motiv se přesně přiloží očištěná a odmaštěná deska cuprextitu mědí směrem k natištěné předloze. Na plošný spoj se přiloží kancelářský papír a nakonec žehlička. Žehlička se nastaví na nižší hodnotu tak, aby se během přežehlování nespálil papír. Po dobu cca jedné minuty se nechá žehlička bez přítlaku na desce. Pak se po dobu další zhruba jedné minuty na žehličku mírně zatlačí. Při této operaci je třeba dbát na to, aby se nepohnulo s deskou a motiv se nerozmazal. Osvědčilo se také několikrát přejet žehličkou přes papír, aby toner byl opravdu na plošném spoji. Pokud je málo zažehlen, motiv v následujícím kroku z desky „uplave“. Po nažehlení sundáme papír a desku plošných spojů s lepícím papírem necháme vychladnout. Když je deska již studená vloží se do nádoby s obyčejnou vodou a nechá se bez dalšího zásahu volně pod vodou. Po krátké době se papír sám odlepí a na plošném spoji zůstane otištěný motiv.

Před leptáním se musí zkontrolovat, jestli se motiv přežehlil dobře a jestli není někde porušen. Pak se připraví leptací lázeň obr. 11. V mém případě jsem použil peroxid vodíku (technický 30%), kyselinu chlorovodíkovou (solnou) a obyčejnou vodu v poměru přibližně 1:1:1. Někdy se dává více vody, aby bylo více času kontrolovat průběh leptání. Při leptání a probíhající chemické reakci se uvolňuje chlór, je tedy nutné provádět tuto činnost v dobře větrané místnosti nebo volně venku. Doba leptání je závislá na koncentraci jednotlivých chemikálií.

Obr. 11: Leptací lázeň [7].

Po odleptání mědi desku vydrhneme a vyčistíme od nažehleného toneru. Poté se vyvrtají otvory pro nožičky součástek. Nakonec se všechny součástky připájí. Tím je deska hotova.

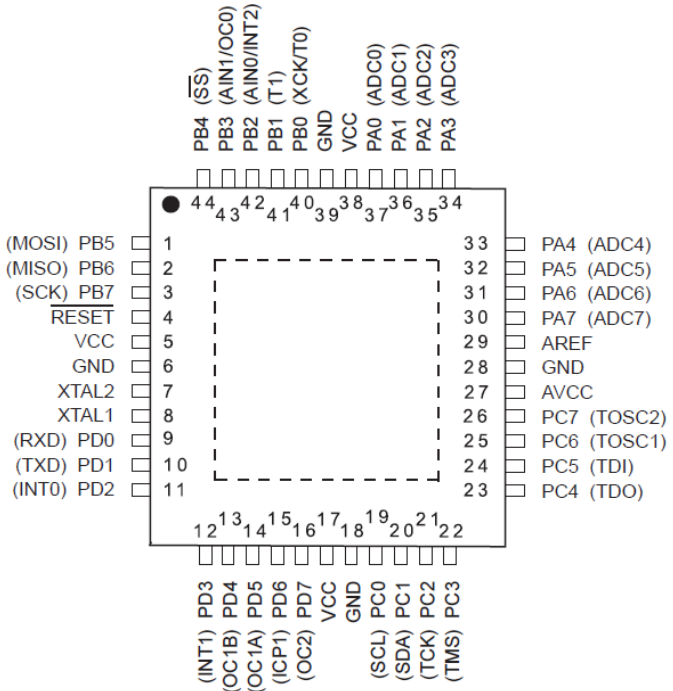
3.4 Mikrokontrolér ATmega16

Pro komunikaci mezi PC a ovládací částí hydraulického ramene byl vybrán mikrokontrolér ATmega16. Mikrokontrolér ATmega16 je nízkopříkonový osmibitový mikrokontrolér založený na rozšířené architektuře AVR RISC. Tím, že provádí výkonné instrukce v jediném hodinovém cyklu, dosahuje 1MIPS na 1MHz [8].

3.4.1 Základní vlastnosti mikrokontroléru ATmega16:

- instrukční soubor obsahuje 131 instrukcí
- 32 registrů délky osmi bitů
- čtyři osmibitové vstupně/výstupní porty (celkem tedy 32 vstupů/výstupů)
- hodinový kmitočet 0 až 16 MHz, maximální výpočetní výkon až 16 MIPS
- paměť programu je tvořena zabudovanou Flash, kapacita je 16 KB ; počet přeprogramování je 1000 cyklů
- datová paměť RAM kapacity 1 KB
- datová paměť E^2PROM kapacity 512 B; počet přeprogramování je 100000 cyklů
- Flash a E^2PROM jsou programovatelné přímo v systému pomocí rozhraní SPI nebo JTAG
- dva osmibitové čítače/časovače; jeden šestnáctibitový (dokonalejší) čítač/časovač
- čtyři PWM kanály
- analogový komparátor, desetibitový A/D převodník
- jednotky USART, SPI, TWI (podpora I^2C)
- jednotky WDT, Power-on reset
- pouzdra DIP 40, TQFP 44

Kvůli dosažení minimálních velikostí řídicích desek, bylo vybráno menší pouzdro typu TQFP44.



Obr. 12: Rozložení vývodů mikrokontroléru ATmega16 na pouzdru TQFP [9].

3.4.2 Podrobnější pohled na některé části mikrokontroléru

Mikrokontrolér ATmega16 je vybaven jednoduchým osmibitovým čítačem/časovačem. Také je k dispozici další osmibitový asynchronní čítač/časovač jehož hodinový kmitočet není odvozen od kmitočtu jádra, ale z „hodinkového“ krystalu 32,768 kHz. Nejdokonalejší je šestnáctibitový čítač/časovač (režimy Out Compare, Input Capture, PWM a další).

Dále je vybaven obvodem WDT (Watch-Dog Timer; hlídač správného běhu programu) a analogovým komparátorem. Také je k dispozici zabudovaný synchronní a asynchronní seriový kanál USART. SPI kanál a TWI rozhraní zajišťuje komunikaci s periferními obvody buď ve stylu sběrnice Microware nebo I^2C. SPI rozhraní se také používá pro programování přímo v aplikaci.

Zajímavostí je zabudovaný RC oscilátor. V případě jeho použití není třeba připojovat krystal nebo vnější zdroj hodinového signálu. JTAG rozhraní zajišťuje především podporu pro ladění aplikace přímo na čipu.

Mikrokontrolér ATmega16 má k dispozici zabudovaný desetibitový A/D převodník s osmi vstupními kanály. Jedná se o vývody ADC0 až ADC7. Pokud se daný kanál nepoužije, jedná se o prostý digitální vstup/výstup. Vstupy A/D převodníku lze navíc konfigurovat do tří režimů:

- osm vstupů SE (Single-Ended) – osm vstupů vztažených vůči AGND (analogová zem, která potlačí rušení pronikající z digitální části)
- sedm diferenčních vstupů – sedm vstupů vztahujících napětí diferenčně proti vývodu ADC1
- dva diferenční vstupy – dva nezávislé diferenční vstupy (ADC1, ADC0 a ADC3, ADC2) s volitelným ziskem 1×,10× a 200×

Mikrokontrolér disponuje čtyřmi úplnými osmibitovými vstupně/výstupními porty, které jsou označeny PA až PD. Všechny porty mohou pracovat jako obousměrné s možností připojit/odpojit zabudované zvyšovací odpory. Vstupní proud je až 20 mA. Vzhledem k omezenému počtu vývodů mikrokontroléru, je mnoho bitů daných portů sdíleno se zabudovanými periférierniemi. Pokud se použije některá z periférií, pak se samozřejmě ztratí možnost použít příslušný vývod portu. Alternativní význam jednotlivých vývodů je uveden v tabulce 1.

Vývod	Význam
$\overline{\text{RESET}}$	nulovací vstup
AIN0	vstup analogového komparátoru (+)
AIN1	vstup analogového komparátoru (-)
$\overline{\text{SS}}$	Slave Select kanálu SPI
MOSI	Master Out/Slave In kanálu SPI
MISO	Master In/Slave Out kanálu SPI
SCK	hodinový signál kanálu SPI
RxD	vstup USART
TxD	výstup USART
XCK	hodinový signál pro synchronní režim USART
INT0	vstup vnějšího přerušení 0
INT1	vstup vnějšího přerušení 1
INT2	vstup vnějšího přerušení 2
T0	hodinový vstup čítače/časovače 0
T1	hodinový vstup čítače/časovače 1
OC0	Output Compare čítače/časovače 0
OC1A	Output Compare čítače/časovače 1 (kanál A)
OC1B	Output Compare čítače/časovače 1 (kanál B)
OC2	Output Compare čítače/časovače 2
ICP	Input Capture čítače/časovače 1
TDI	vstupní data JTAG rozhraní
TDO	výstupní data JTAG rozhraní
TMS	výběr režimu JTAG rozhraní
TCK	hodinový signál JTAG rozhraní
SDA	datový signál TWI (I^2C) rozhraní
SCL	hodinový signál TWI (I2C) rozhraní
ADC0 až ADC7	kanály A/D převodníku

Tab. 1: Alternativní význam jednotlivých vývodů [8].

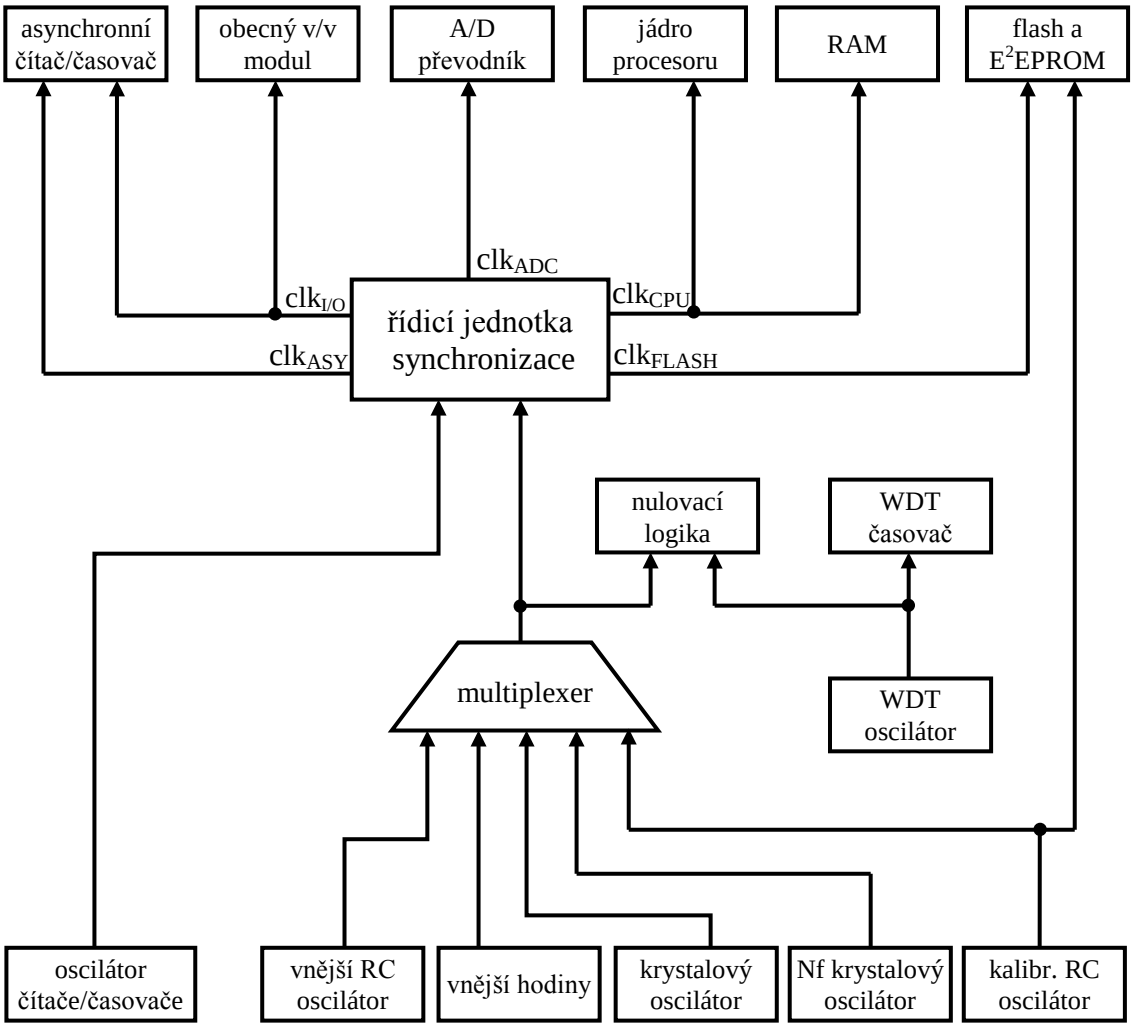
3.4.3 Systémové hodiny a volby synchronizace

Na obr. 13 je znázorněna distribuce hodinového signálu v mikrokontroléru ATmega16. Hodiny jádra procesoru clk_{CPU} řídí časování AVR jádra (registry, SREG, SRAM, SP). Výběr zdroje synchronizace pro clk_{CPU} lze provést pomocí programovatelných propojek. Na výběr je pět druhů zdrojů:

- vnější krystal nebo rezonátor
- vnější nízkofrekvenční krystal
- vnější RC oscilátor
- kalibrovaný vnitřní RC oscilátor
- vnější hodiny

V mém případě byl jako zdroj synchronizace vybrán vnější oscilátor s frekvencí 11,059Mhz.

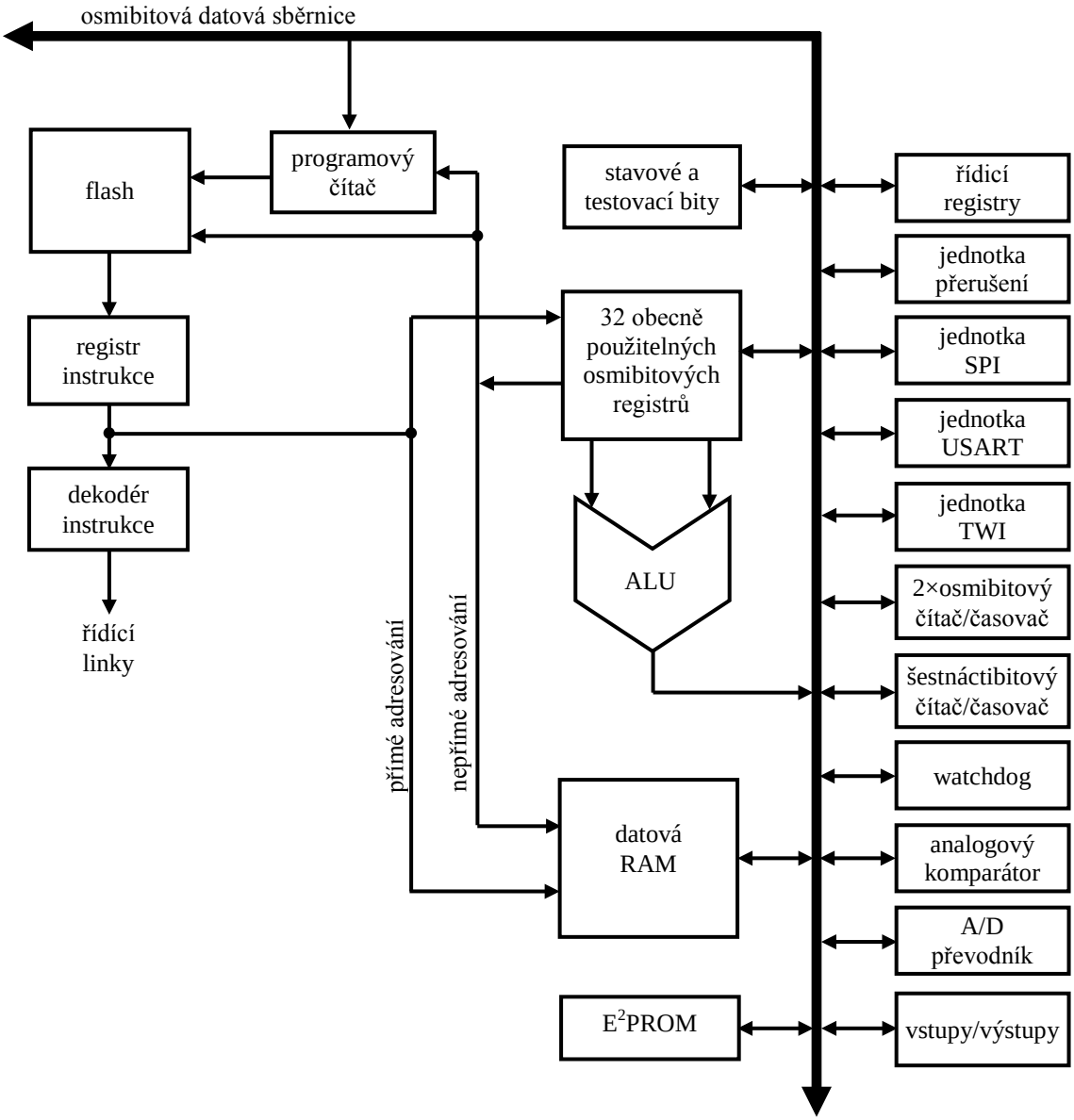
Hodiny vstupně/výstupních jednotek $clk_{I/O}$ řídí vstupně/výstupní jednotky (čítač/časovač, SPI, USART), také se používá pro vnější přerušení. Hodiny pro flash clk_{FLASH} řídí operace nad rozhraním flash (obvykle se aktivují současně s clk_{CPU}). Hodiny asynchronního čítače/časovače clk_{ASY} představují synchronní signál 32,768 kHz pro osmibitový asynchronní čítač/časovač. Tento čítač/časovač se obvykle používá jako hodiny reálného času a to i pro případ, že mikrokontrolér přejde do režimu spánku. Poslední z hodinových signálů jsou hodiny A/D převodníku clk_{ADC}. Ty synchronizují A/D převody a tak dovolují zastavit clk_{CPU} hodiny a $clk_{I/O}$ pro snížení rušení vyzařovaného z digitálního jádra do analogové části A/D převodníku.



Obr. 13: Distribuce hodinového signálu [8].

3.4.4 AVR architektura

AVR architektura vychází z koncepce rychle přístupného registrového pole, které obsahuje 32 obecně použitelných registrů délky osm bitů. Přístup do registrového pole je proveden v jediném strojovém cyklu. To znamená, že během jednoho strojového cyklu lze vykonat jednu aritmeticko – logickou operaci. Oba operandy aritmeticko – logické instrukce jsou načteny z registrového pole, operace je provedena a výsledek směřuje opět do registrového pole. Toto vše se provede v jediném strojovém cyklu. Tato technika dává procesorům ATmega velký výpočetní výkon.



Obr. 14: Zjednodušený výklad AVR architektury [8].

3.5 Programovací jazyk a vývojové prostředí

K programování mikrokontrolérů se v minulosti většinou používal assembler. Výjimku tvořily velké aplikace, kde se zapojovaly i vyšší programovací jazyky (VPJ) jako C nebo Pascal. V posledních letech se však od assembleru upouští a hojně se využívají VPJ [10]. Důvod je takový, že assembler je méně přehledný, hůře upravovatelný a špatně přenositelný kód oproti jeho nástupcům. Dnes se již nemusíme příliš ohlížet na kódem zabírané místo v mikrokontroléru, protože cena paměťového prostoru dostatečně klesá a je tedy výhodnější zaplatit větší paměť a programátora VPJ než ušetřit na paměti a připlatit za zdlouhavou práci na vývoji v assembleru. Dalším důvodem rozšíření VPJ je fakt, že výrobci mikrokontroléru začali před několika lety uvádět na trh nové čipy uzpůsobené pro snadnější programování ve vyšších jazycích. K tomu, že mikrokontrolér ATmega16 budu programovat ve VPJ *ANSI-C*, nepřispěly jen výše uvedené důvody, ale hlavně to, že mám blíže k programování v jazyku C než k programování v assembleru.

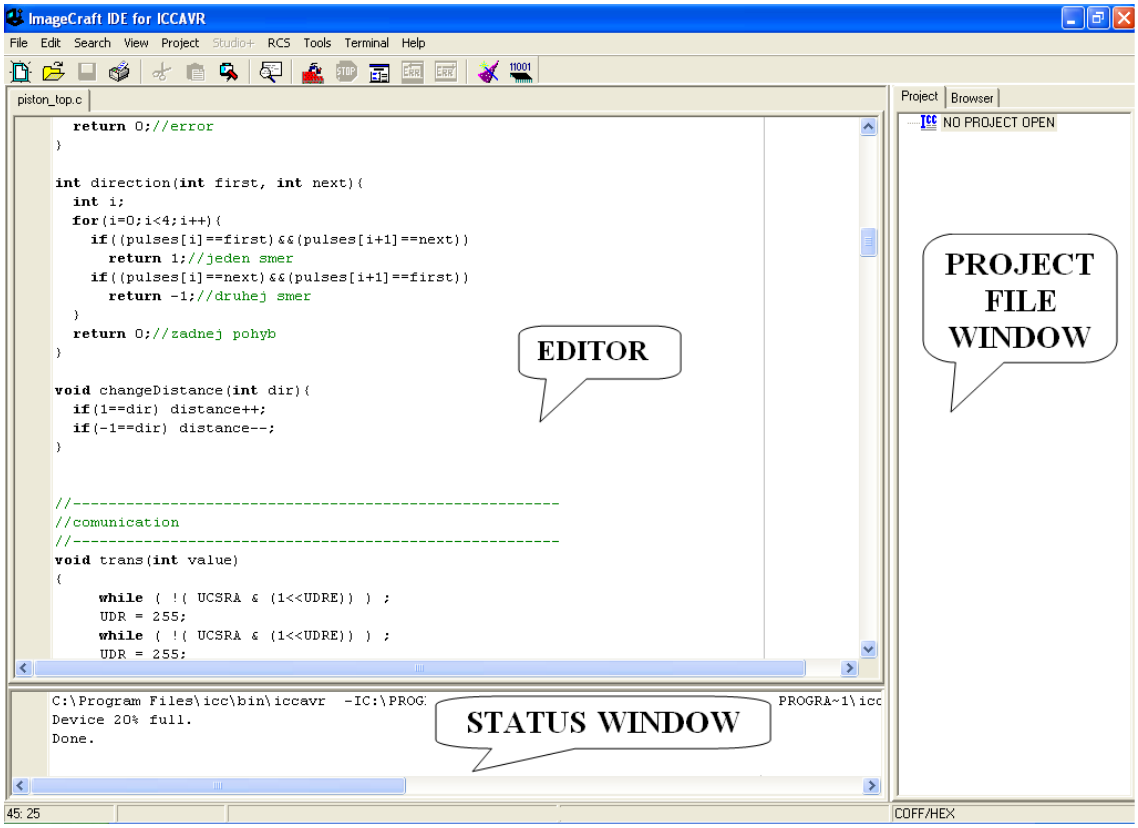
3.5.1 ImageCraft ICC AVR

Jako kompilátor VPJ byl zvolen poměrně kvalitní ICC AVR od firmy ImageCraft. Použita byla plně funkční, ale časově omezená demoverze. ICC AVR je moderní vývojové prostředí, které obsahuje rychlý C kompilátor s plnou podporou ANSI C. Nabízí optimalizaci pomocí:

- přepínání řízení
- specifických instrukcí pro daný procesor
- algebraického zjednodušení
- pokročilého přiřazování registrů
- eliminování bloků společných podprogramů
- odstranění redundantních instrukcí

ICC AVR dále obsahuje velké množství podpůrných knihoven s předpřipravenými makry a funkcemi, které ušetří práci programátorovi, neboť se ve většině případů může oprostit od práce s hexadecimálními čísly, například místo 0x15 lze zapisovat PORTC, což je snáze zapamatovatelné. Adresy přerušení, porty i specifické registry každého AVR procesoru mají obšírně zpracovaná makra uložená většinou v souboru ioxxx.h, kde x odpovídá typu mikrokontroléru. Pro jejich použití stačí pouze připojit patřičný hlavičkový soubor ke konkrétnímu typu mikrokontroléru.

Samotný program je rozdělen do tří oblastí: „editor“, „project file window“ a „status window“, podle obrázku 15. V okně editoru se píše vlastní program. Tento editor má podporu zvýraznění slov správně syntakticky zapsaných. V okně „project file window“ jsou zobrazeny všechny soubory použité při vývoji programu. Stavové okno zobrazuje zprávy o překladu zdrojového kódu. Tyto dvě poslední okna lze skrýt, aby bylo editorové okno co největší.



Obr. 15: Prostředí ICC AVR.

3.5.2 Nastavení projektových vlastností

Předtím, než se začne psát vlastní program, je potřeba nastavit vlastnosti překladače. Každý projekt může mít své vlastní nastavení, které se ukládá do adresáře tohoto projektu. Jedná se o okno „*Project → Options*“, které obsahuje tři záložky. V záložce „*Paths*“ se nastavují cesty k hlavičkovým souborům, knihovnám a adresář kam se zkompiluje program. V záložce „*Compiler*“ se nastavují vlastnosti ovlivňující vlastní překlad. Například výstupní formát nebo nastavení optimalizace. Tuto záložku jsem nechal v původním tzv. defaultním nastavení. A nakonec v záložce „*Target*“ se nastavují vlastnosti jako typ mikroprocesoru, pro který je program určen. V mém případě jsem tedy vybral ATmega16. Typ funkce PRINTF jsem neměnil, zůstala nastavena na *small*. Přesným výběrem mikroprocesoru podle jeho označení okamžitě *compiler, assembler* a *linker* ví, jakou velikost paměti lze využívat, jaké periferie atd., což velmi usnadňuje práci oproti jiným překladačům, kde se všechny tyto parametry nastavují zvlášť.

3.6 Popis programů řídicích desek

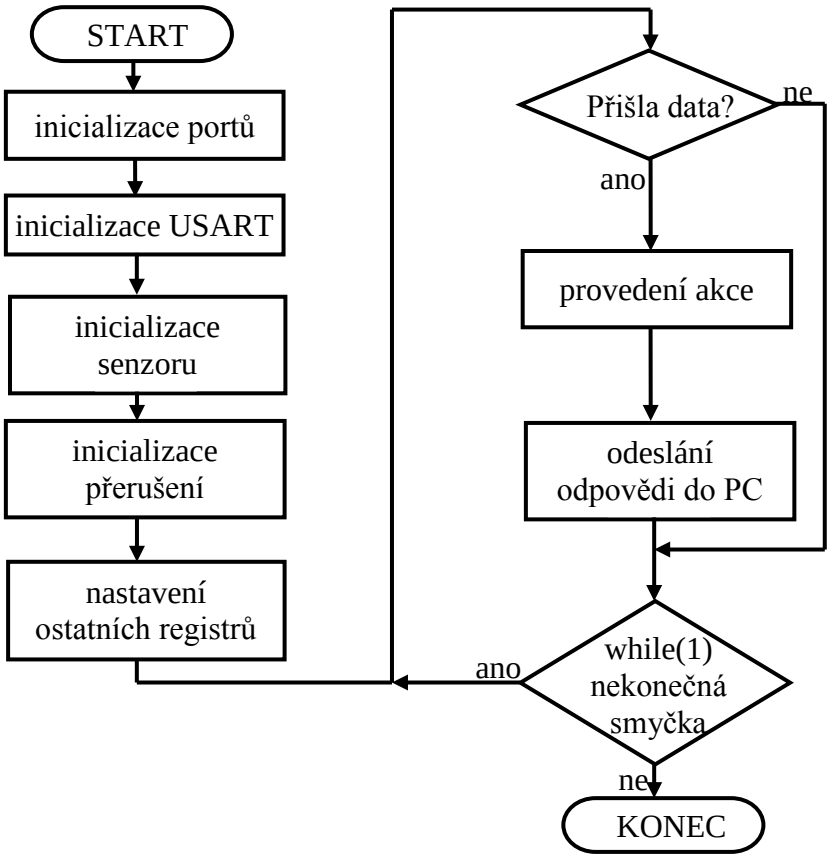
Program zajišťuje oboustrannou komunikaci mezi PC a prvky ovládající hydraulické rameno a senzorickou částí. Program bude popsán postupně. Programy v jednotlivých mikrokontrolérech vypadají dosti podobně. Rozdíly jsou pak v hlavní smyčce, kde probíhá obsluha daných zařízení. Kostra programu vypadá podobně jako v bodech navržená osnova níže.

Inicializace procesoru:

1. inicializace portů
2. inicializace USART
3. inicializace senzoru
4. inicializace přerušení
5. nastavení ostatních registrů na potřebné výchozí hodnoty

Hlavní nekonečná smyčka programu:

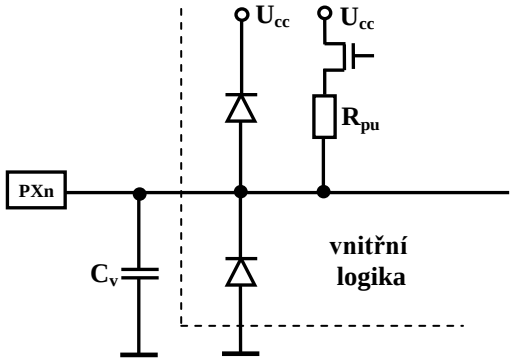
6. testování, zda nepřišla data z PC
7. provedení akce dle obdržených dat
8. vyslání informace do PC o provedení akce



Obr. 16: Vývojový diagram programu.

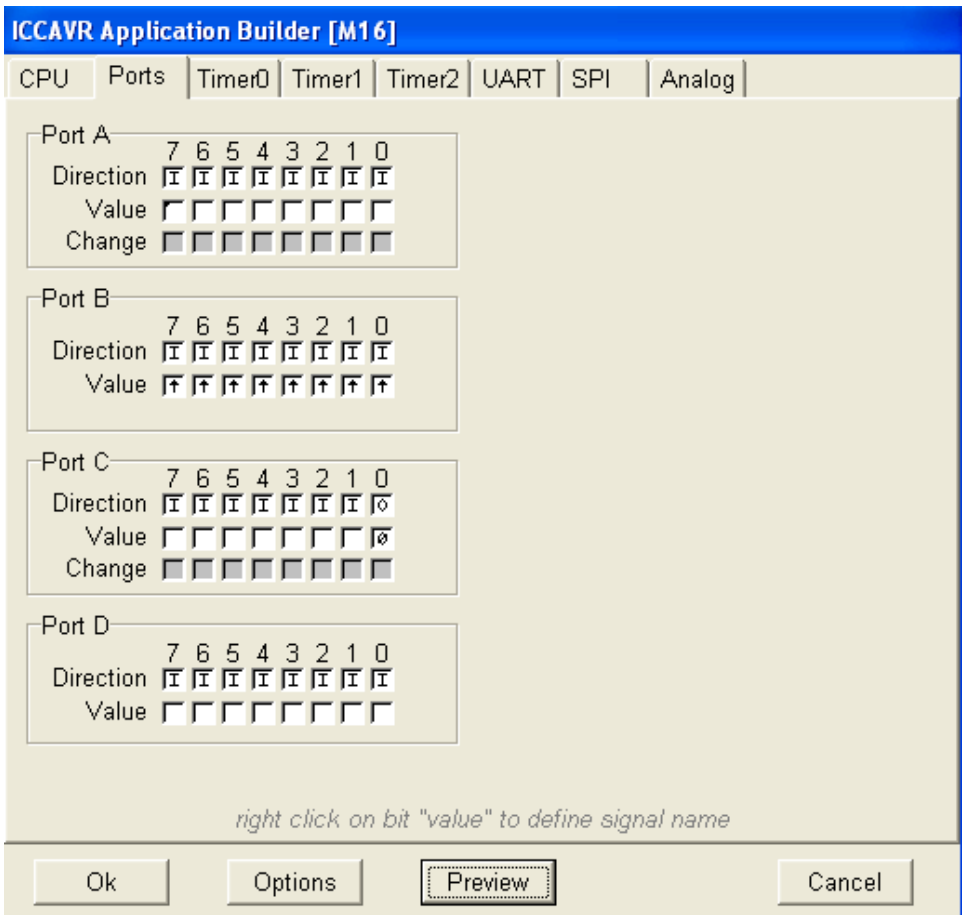
3.6.1 Inicializace

Program musí nejprve nadefinovat výchozí nastavení portů. Všechny vývody mají individuálně aktivovatelný pull-up rezistor R_{PU} (zdvihací rezistor tažený k napájecímu napětí) a ochranné diody dle obrázku 17.



Obr. 17: Ekvivalentní zapojení vstupu/výstupu PXn [8].

Kapacitu vývodu zajišťuje kondenzátor C_V. Symbol PXn zastupuje libovolný vývod portu. X je název (A až D), n je pak číslo bitu (0 až 7). Mnoho vývodů má zaveden multiplex s alternativními funkcemi vývodů, jak bylo popsáno dříve v tabulce 1.



Obr. 18: Nastavení portů mikroprocesoru ATmega16 v programu ICC AVR.

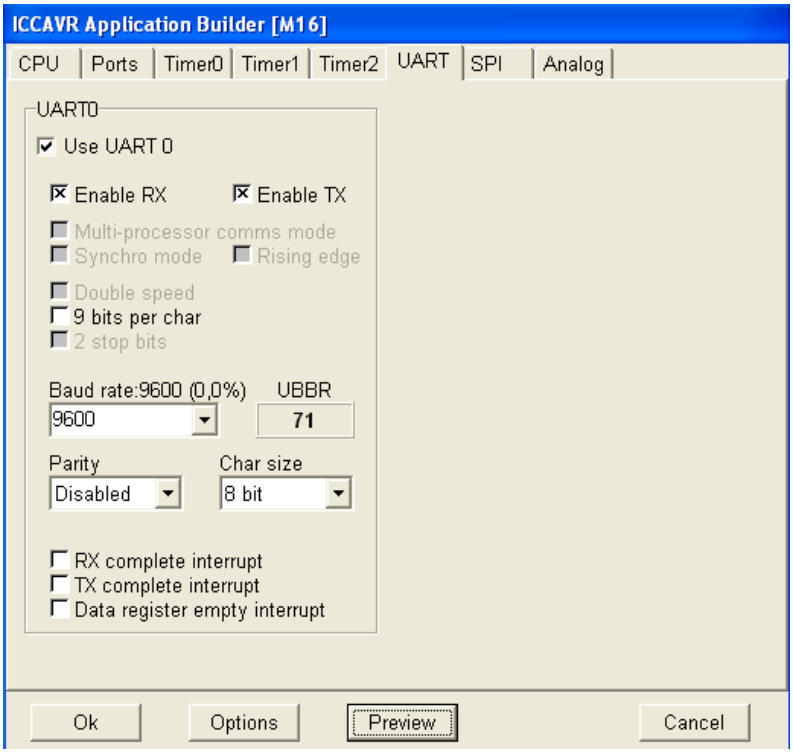
Vývody portů použité jako obecné číslicové vstupy/výstupy jsou obousměrné a lze u nich konfigurovat pull-up rezistory. Každý port je ovládán třemi registry (X zastupuje název portu):

- Registr DDRX (Data Direction Registr) určuje směr toku dat (zda se vývody chovají jako vstupy nebo výstupy).
- Registr PORTX je datovým registrem portu, odpovídá hodnotě zapsané do vyrovnávacího registru portu.
- Registr PINX (Pins Input) je určen pouze pro čtení a odpovídá hodnotě přečtené z jednotlivých vývodů.

Toto nastavení se v kompilátoru ICC AVR provede pomocí nástroje *Application Builder* v kartě *Ports* jak je vidět na obrázku 18.

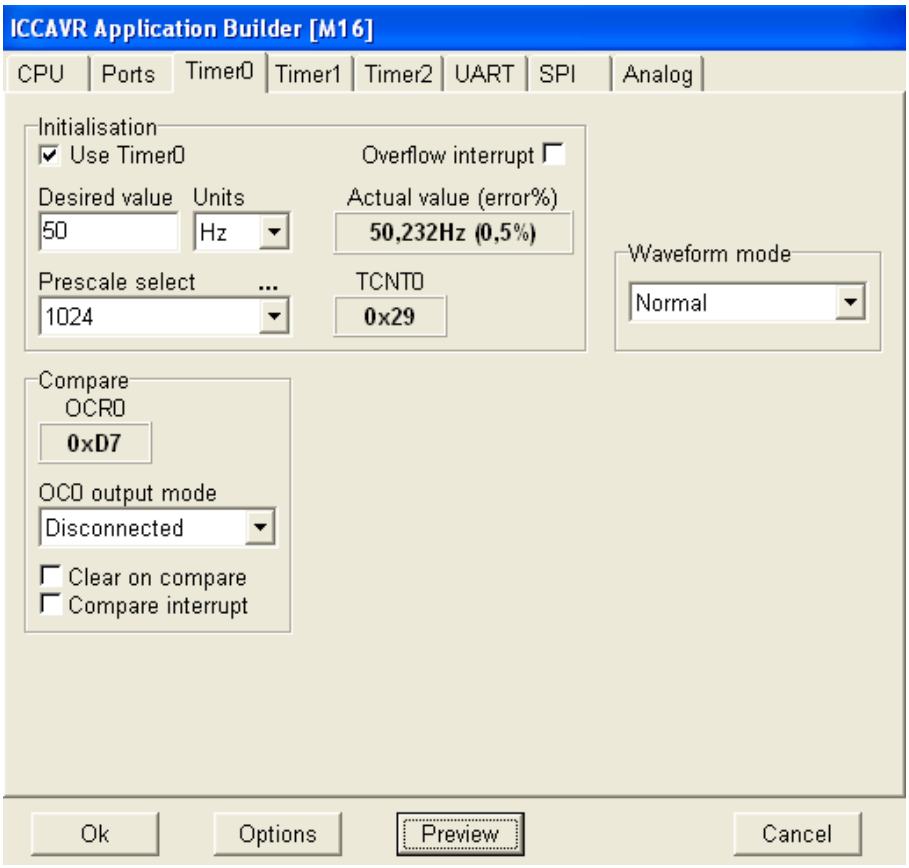
Jako další se musí zinicializovat univerzální synchronní a asynchronní sériový přijímač a vysílač (USART). Jedná se o vysoce flexibilní zařízení pro sériovou komunikaci. Jednotka USART zakomponovaná do mikrokontroléru ATmega16 je plně kompatibilní s jednotkami UART, které byly k dispozici ve starších modelech mikrokontrolérů AVR. Zinicializování této jednotky sestává z povolení funkce přijímače a vysílače, nastavení přenosové rychlosti, formátu rámce, parity. V mém případě jsem nastavoval tyto hodnoty:

- Povolení přijímač a vysílač – enable RX, enable TX
- Přenosová rychlost - Baud rate 9600
- Formát rámce – Char size 8 bit
- Bez parity – Parity disabled



Obr. 19: Nastavení USART v programu ICC AVR.

Nakonec se v *Application Builder* zinicializují časovače. Časovač TIMER0 musí být nastaven tak, aby každých 20 ms vyvolal přerušení. To odpovídá hodnotě 50 Hz. Přerušení je vyvoláno s předděličkou 1024. S tímto nastavením se nedosáhne zcela přesné hodnoty, ale hodnoty 50,232 Hz což odpovídá chybě 0,5%. Tato chyba však nemá na funkčnost programu žádný vliv.



Obr. 20: Nastavení Timeru0 v programu ICC AVR.

Časovač Timer1 má být nastaven tak, aby vyvolal přerušení každých 1,5 ms. To odpovídá frekvenci 666,667 Hz. Nastavení se provede v ICC AVR na příslušné kartě podobně jak tomu bylo u předchozího časovače. Tentokrát se použije předdělička 1. O nastavení příslušných registrů se postará ICC AVR sám na základě zvoleného nastavení.

Poslední časovač Timer2 je nastaven tak, že každých 15µs vyvolá přerušení. Odpovídající frekvence pro tento čas je 64 KHz. Po nastavení v ICC AVR s předděličkou 8 dostaneme hodnotu 65, 827 KHz a chybu 2,8 procenta.

Před inicializací časovačů musí ještě proběhnout inicializace senzoru, která už se nedá nastavit v *Application Builderu* jako předchozí inicializace. Jako první se v této inicializaci nastaví proměnná *distance* na nulu. Tato proměnná uchovává vzdálenost od nuly v počtech impulsů. Pak se zavolá funkce *readSensor*, která přečte hodnotu ze snímače, jež se pak přiřadí do proměnné *first*, která uchovává první hodnotu ze snímače. Nakonec se do proměnné *next* přiřadí hodnota uložená v proměnné *first*. Tím je vše zinicializováno.

3.6.2 Přerušení

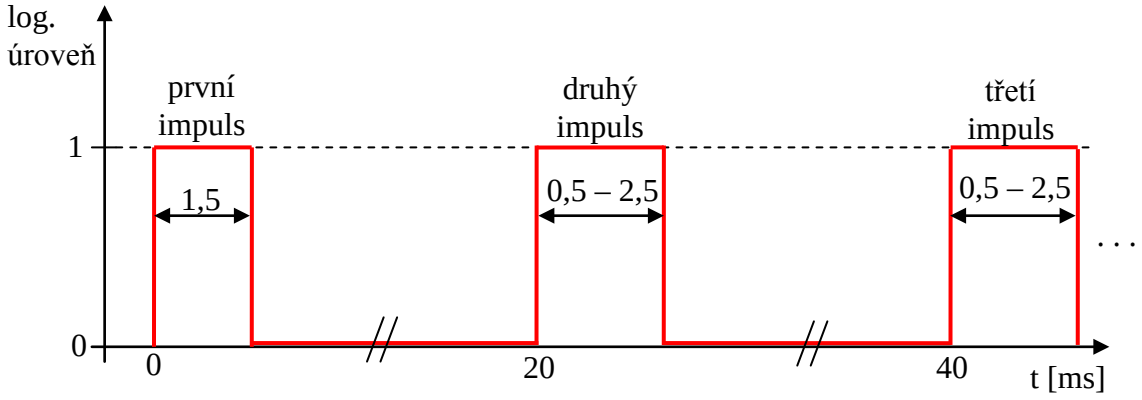
Výše uvedená inicializace časovačů určuje, po jaké době bude vyvoláno přerušení od příslušného časovače. Co se děje v těchto přerušeních je popsáno níže:

- *Přerušení od Timeru0*

  V tomto přerušení se nastavuje hodnota proměnné *count* na hodnotu 1.

- *Přerušení od Timeru1*

Toto přerušení má na starost šířku impulsu posílaného na servo. Zvolená hodnota 1,5 ms odpovídá střední hodnotě polohy serva. Krajní polohy jsou 0,5 a 2,5 ms. Pokud je proměnná *count* nastavena na hodnotu 1 přiřadí se do registru PORTC hodnota 0x01. Přiřazením této hodnoty nastavíme logickou úroveň vývodu na logickou 1. Tím se nám na vývodu, na kterém je připojeno servo objeví impuls. Poté nastavíme proměnnou *counter* na požadovanou hodnotu, která přišla z PC. Tato hodnota nám udává šířku impulsu. Dále nastavíme proměnnou *count* na hodnotu 2. Při dalším přerušení se vyhodnotí podmínka count == 2 a proměnné PORTC se přiřadí 0x00 čímž, se na výstupu objeví logická nula. Tím se ukončil impuls odeslaný na servo. Jelikož servo zpracovává impulsy zhruba s frekvencí 50 Hz, je potřeba, aby se na vývodu po 20 ms objevil další impuls. To zajišťuje právě přerušení od Timeru0, který vrací do proměnné *count* hodnotu 1.



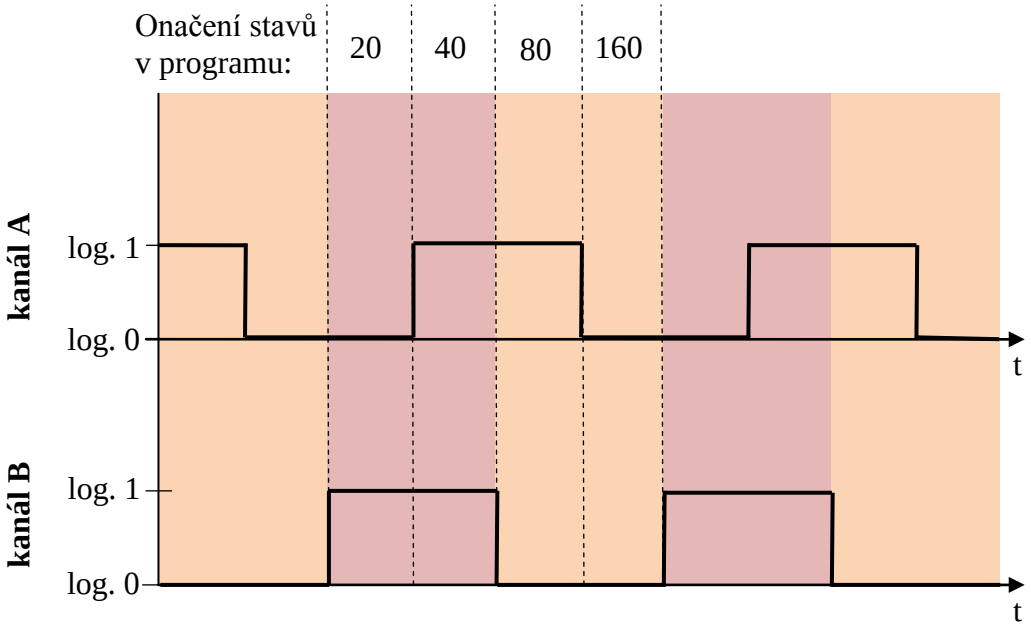
Obr. 21: Graf řídicích signálů posílaných na servo.

- *Přerušení od Timeru2*

Toto přerušení slouží ke čtení impulsů z magnetického snímače. Zde nebyly použity žádné funkce kvůli rychlosti kódu. Pokud se totiž vše nestihne provést včas, tak nefunguje výstup na servo.

Při pohybu magnetického snímače nad magnetickým páskem začne snímač vysílat impulsy na kanálech A a B. Impulsy na těchto kanálech jsou od sebe posunuty o $\pi/2$. To zajišťuje větší přesnost snímače. První věc v tomto přerušení je zjištění, v jakém stavu se nachází oba impulsy, tj. zdaly jsou v logické jedničce či nule. Stavy jsou v programu označeny hodnotami 20, 40, 80, 160. Která hodnota odpovídá jakému stavu je vidět na obrázku 22.

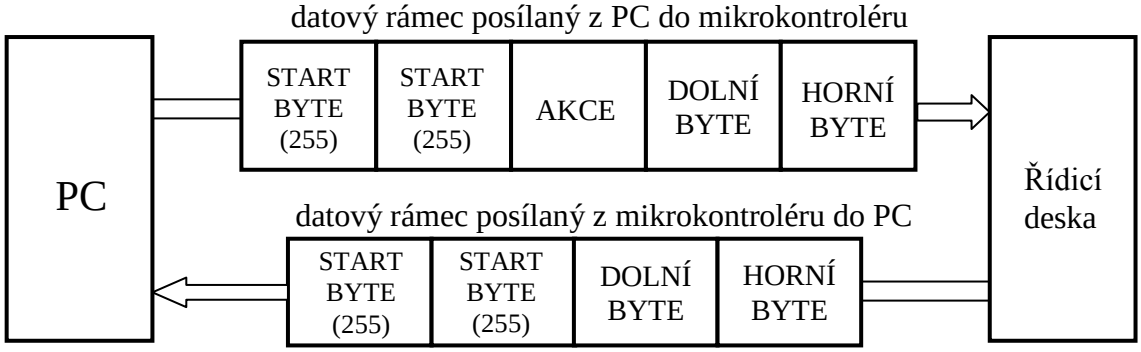
Když je zjištěno v jakém vzájemném stavu se nachází impulsy, tak se porovnají s předchozím stavem. Tím se určí směr pohybu. Směr pohybu v jednom směru určuje řada po sobě jdoucích hodnot: 20 – 80 – 40 – 160 – 20. Opačný směr je dán hodnotami: 20 – 160 – 40 – 80 – 20. Podle směru změníme hodnotu, která je v proměnné *distance.* V jednom směru se proměnná inkrementuje v druhém dekrementuje. Proměnná *distance* tedy uchovává počet impulsů od počáteční hodnoty. Z této hodnoty se pak určí posun snímače vůči magnetickému pásku.



Obr. 22: Výstup z obou kanálů snímače a označení stavů v programu.

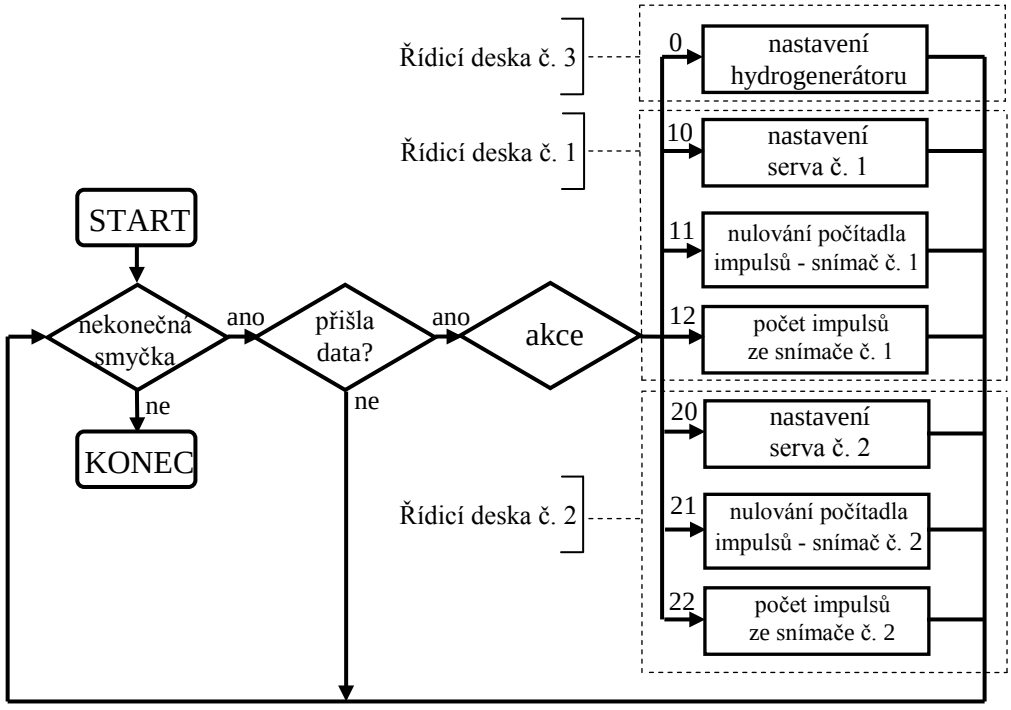
3.6.3 Hlavní program

Po inicializacích následuje hlavní funkce programu funkce *main*. Vývojový diagram této funkce je na obrázku 24. V ní běží nekonečná smyčka. Uvnitř této smyčky se přes funkci *receive* testuje, jestli přišla nějaká data z PC. Formáty datových rámců jsou znázorněny na obrázku 23.



Obr. 23: Formáty datových rámců.

Uvnitř funkce *receive* se testuje, jestli přišel odpovídající datový rámec. Čeká se na dvě kontrolní hlavičky, které jsou hodnoty 255. Ty slouží k odfiltrování jiných signálů, které by mohli narušit komunikaci. Třetí bajt obsahuje číslo akce. Tato hodnota se ukládá do proměnné *action*. Další dva bajty obsahují hodnotu, která určuje změnu šířky impulsu odesílaného na servo nebo na hydrogenerátor. Tato hodnota je ukládána do proměnné *timer_value*.



Obr. 24: Vývojový diagram hlavní funkce main.

Pokud tedy byl obdržen korektní datový rámec, rozebere se a uloží do patřičných proměnných a pokračuje se v další instrukci hlavní funkce. V ní se vyhodnocuje obsah proměnné *action*. Zde se program v mikrokontrolérech jednotlivých řídicích desek liší. Každá má naimplementován jen kód pro konkrétní zařízení, které obsluhuje. To, kterou část kódu mikrokontrolér desky obsahuje, je znázorněno ve vývojovém diagramu hlavní funkce na obrázku 24. Dále budou jednotlivé akce postupně popsány. Pro přehlednost jsou čísla akcí a jim odpovídajících návratových hodnot uvedeny v tabulce 2.

- Nastavení hydrogenerátoru.

Hodnota uložená v proměnné *timer_value* se zkontroluje, jestli spadá do povoleného rozsahu hodnot. Pokud ano, přiřadí se hodnota do proměnné *pozice*. S touto proměnnou se pak pracuje v přerušení od *Timer1*, kde udává šířku impulsu posílaného k hydrogenerátoru. Pokud vše proběhne v pořádku, odešle se zpět do PC příslušná návratová hodnota. Pokud se hodnota v *timer_value* vyhodnotí, že je mimo rozsah, odešle se návratová hodnota viz. tabulka 2.

Datový rámec posílaný z mikrokontroléru do PC má velikost čtyři bajty. První dva bajty jsou opět kontrolní hlavičky. Další dva bajty obsahují příslušnou návratovou hodnotu.

popis akce	číslo akce	příjemce	návratová hodnota	popis návratové hodnoty
nastavení hydrogenerátoru	0	deska č. 3	102	nastavení generátoru - OK
			103	špatná hodnota pro nastavení hydrogenerátoru
nastavení serva č. 1	10	deska č. 1	110	nastavení serva č. 1 - OK
			111	špatná hodnota pro nastavení serva č. 1
vynulování počítadla impulsů - snímač č.1	11	deska č. 1	112	vynulování počítadla impulsů snímače č. 1 - OK
počet impulsů – snímač č. 1	12	deska č. 1	počet impulsů	vrací počet impulsů ze snímače č. 1
nastavení serva č. 2	20	deska č. 2	120	nastavení serva č. 2 - OK
			121	špatná hodnota pro nastavení serva č. 2
vynulování počítadla impulsů - snímač č. 2	21	deska č. 2	122	vynulování počítadla impulsů snímače č. 2 - OK
počet impulsů – snímač č. 2	22	deska č. 2	počet impulsů	vrací počet impulsů ze snímače č. 2
Deska č. 1 reaguje také na neznámou akci a na chybu v přenosu. Pokud některá z těchto situací nastane, tak pošle návratovou hodnotu 100 resp. 101.				

Tab. 2: Akce a návratové hodnoty.

- Nastavení serva.

  Zde je princip úplně stejný jako v případě řízení hydrogenerátoru.

- Vynulování počítadla impulsů.

Zde se volá funkce *sensor_init*, ve které se proměnná *distance* nastaví na nulu. V tomto případě je v datovém rámci, který přišel z PC, na pozicích horního a dolního bajtu nula.

- Zjištění počtu impulsů ze snímače.

Zde se volá funkce *trans*, která odešle do PC hodnotu proměnné *distance*. Mikrokontrolér vrací do PC na posledních dvou bajtech datového rámce počet impulsů odeslaných snímačem do mikrokontroléru.

4 NADŘAZENÝ ŘÍDICÍ SOFTWARE

Aby se dalo pomocí počítače hydraulické rameno ovládat, bylo nutné naprogramovat nadřazený řídicí software. Ten byl podle zadání diplomové práce naprogramován ve vývojovém prostředí NI LabVIEW 8.6, na které vlastní škola licenci. Kapitola 4.1 pojednává obecně o programu LabVIEW. Kapitola 4.2 seznamuje s prostředím a programováním v LabVIEW. V dalších kapitolách je pak detailně popsán řídicí software pro ovládání hydraulického ramene.

4.1 LabVIEW

LabVIEW je zkratkou pro Laboratory Virtual Instrument Engineering Workbench od firmy National Instruments. Jedná se o moderní výkonný systém speciálně vhodný pro programování komunikace osobního počítače s různými periferními zařízeními. LabVIEW lze chápat jako vývojové prostředí nad jazykem G. Jazyk G značí grafický programovací jazyk [11]. Narozdíl od Object Pascalu, kde zdrojovým kódem programu je text, zdrojovým kódem programu v jazyce G je obrázek. Místo aby byly programy psány, budou kresleny. Grafické programování je technika neobvyklá a nová. Byla patentována společností National Instruments teprve v roce 1990. Pro jazyk G existuje kompilátor, který produkuje samostatné spustitelné programy. Tvůrci LabVIEW prohlašují, že programy vytvořené v jazyce G běží po přeložení srovnatelně rychle, jako programy napsané v jazyce C, který je obecně považován za velmi efektivní. V jazyce G má programátor k dispozici jak rychlé funkce nízké úrovně, tak i hotové komplikované podprogramy pro matematickou analýzu, statistiku, komunikaci se standardizovanými periferiemi a podobně. V současné době neexistuje česká lokalizace LabVIEW, pouze anglická, německá, španělská a japonská. Naprostá většina akcí se provádí myší.

4.2 Úvod do programování v LabVIEW

LabVIEW vyvíjí společnost National Instruments. Uživatelské rozhraní programu LabVIEW se podobá skutečným měřicím přístrojům. Proto se mu také říká virtuální přístroj. Anglický ekvivalent je „Virtual instruments“ a jeho zkratka je VI. Tuhle zkratku National Instruments používá dosti často ve svém textu v příručkách k LabVIEW.

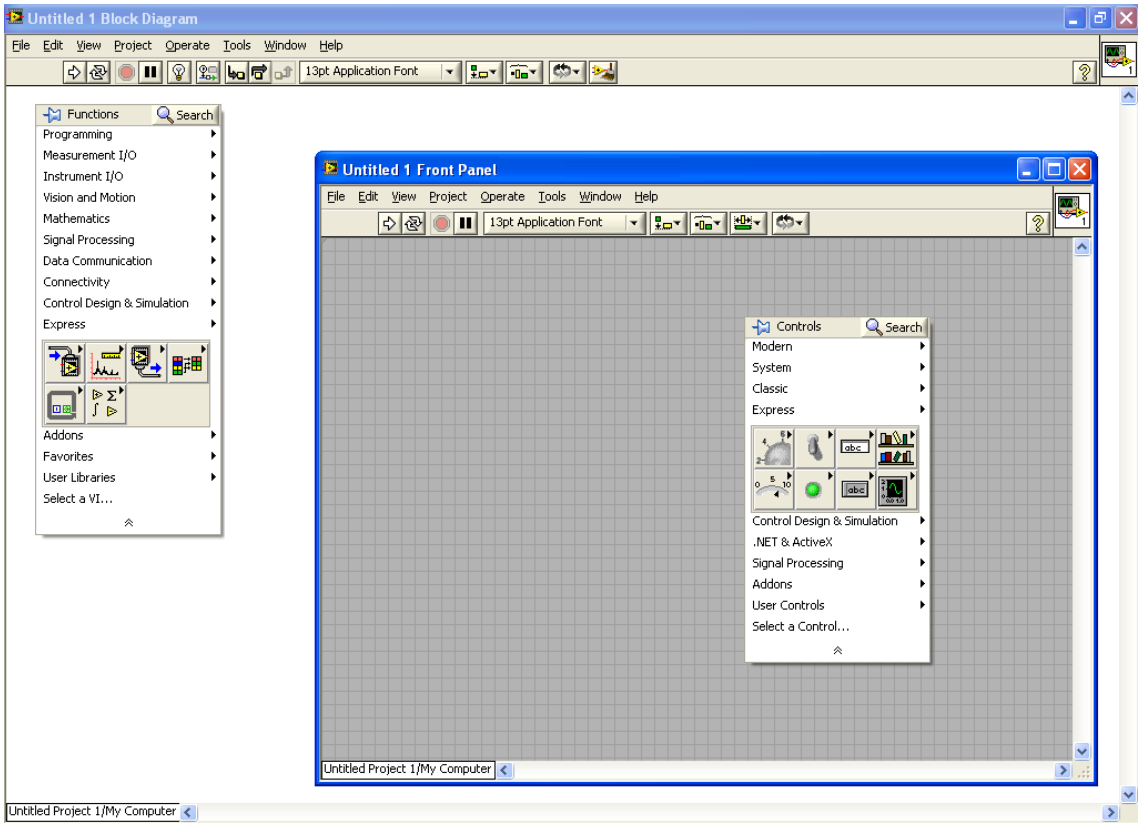
K vytváření VI slouží vývojové prostředí LabVIEW. Po otevření programu LabVIEW, se na monitoru objeví startovací nabídka tzv. „*Getting Started*“. Startovací nabídka je rozdělená na soubory „*Files*“ a zdroje „*Resources*“. Soubory jsou dále rozděleny na vytvoření nového VI, projektu nebo VI ze šablony a na otevření VI ze souboru. Ve zdrojích se nachází různé nápovědy.

Pokud bude otevřen nový VI, tak se na monitoru objeví dvě okna. *Front panel* a *Block diagram* viz. obrázek 25. Pokud by tyto okna byly přirovnány k nějakému přístroji, tak *Front panel* je zevnějšek přístroje, kde je veškeré ovládání a indikace a *Block diagram* je veškerá elektronika a řízení, které je ukryto uvnitř přístroje. Tedy *Block diagram* slouží k programování VI a jako výsledný produkt vznikne ve *Front panelu*.

Dále jsou na obr. 25 znázorněny dvě palety. Paleta *Function* patří k blokovému diagramu a jsou zde funkce a proměnné potřebné k sestavení programu. Má stromovou strukturu. Zobrazí se, pokud se pokliká pravým tlačítkem myši do blokového diagramu. Po zobrazení se u názvu palety objeví nepřipíchnutý připínáček, pokud se pokliká

na něj, tak paleta zůstane po celou dobu u blokového diagramu. Jinak po každém použití funkce se musí paleta znovu vyvolávat.

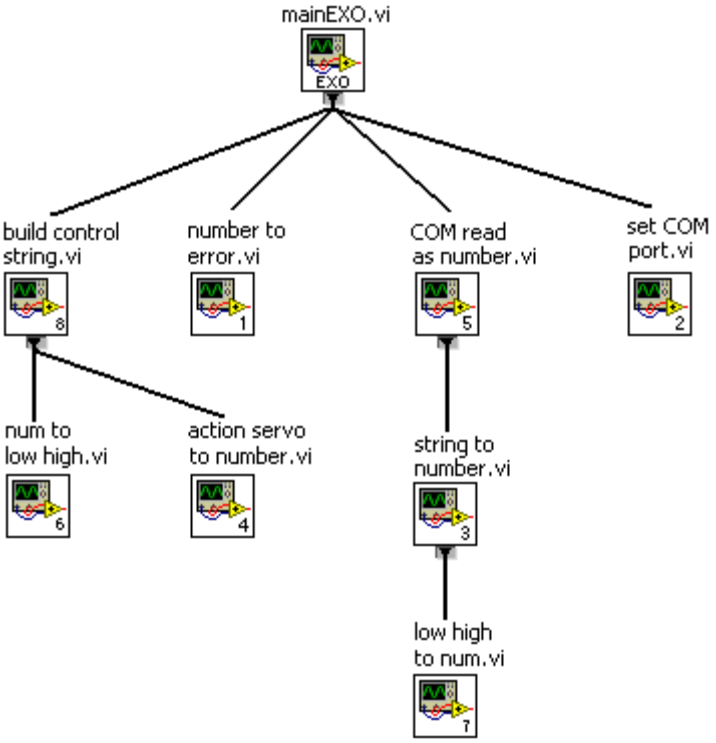
Paleta *Controls* patří k čelnímu panelu. Obsahuje veškeré indikátory a řídící součásti. Má také stromovou strukturu a stejně tak i připínáček u názvu palety ze stejného důvodu jako paleta *Function*. Další zajímavostí je, že jednotlivé palety jsou viditelné, pouze když je aktivní dané okno buď blokového diagramu, nebo čelního panelu. To znamená, že nemohou být aktivní obě palety současně.



Obr. 25: Prostředí programu LabVIEW.

4.3 Popis jednotlivých částí řídicího systému

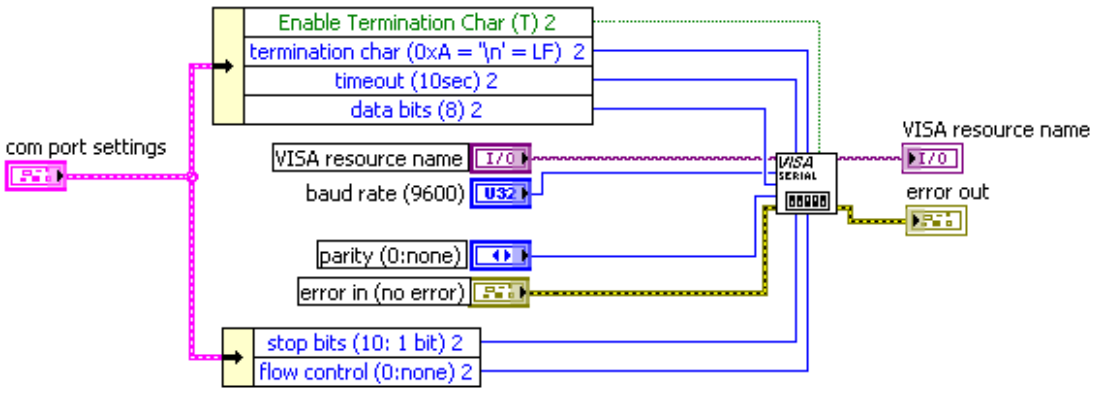
Samotný řídicí software není jen jedno VI. Skládá se z několika dílčích podprogramů, které se nazývají podřízené virtuální přístroje tzv. subVI. Každé subVI provádí nějakou dílčí úlohu. Používání subVI má velkou výhodu pokud děláme nějaké rozsáhlé blokové schéma, které obsahuje několik identických částí. V případě nalezení chyby nebo úpravy programu pak stačí upravit jen dané subVI. Další výhodou je přehlednost nadřazeného VI. Součástí každého virtuálního přístroje je jeho ikona, kterou je prezentován v blokovém schématu, a konektor s přípojnými místy pro vstupní a výstupní signály. V dalších kapitolách budou jednotlivá vytvořená subVI popsána. Na závěr bude popsáno nadřazené VI s názvem *mainEXO*.



Obr. 26: Hierarchická struktura VI.

4.3.1 Nastavení COM portu

Jako první bude popsáno VI s názvem *set COM port*. Jak už název napovídá, jedná se o nastavení sériové komunikace. V LabVIEW verze 8.2 a vyšší jsou pro sériovou komunikaci vytvořeny bloky. Nalézají se na paletě *Function → Instrument I/O → Serial.* Nejdříve je vložen blok *VISA Configure serial port*. Je to vlastně před vytvořené VI vložené jako blok. Přes konfiguraci portu lze nastavit několik vlastností portu. Jednotlivé vlastnosti budou nastavitelné v hlavní aplikaci v záložce *COM port settings.*



Obr. 27: Nastavení vlastností sériového přenosu v LabVIEW.

Vstupy:

- *VISA resource name* – zde se volí port, na kterém bude sériová komunikace probíhat
- *Baud rate* – je nastavení rychlosti komunikace, která je nastavena defaultně na 9600 baudů
- *Data bits* – jsou přednastaveny na osmi bitovou komunikaci
- *Parita* – parita komunikace není žádná
- *Stop bits* – je přednastaven na jeden stop bit
- *Flow control* – slouží k nastavení typu kontroly v mechanismu odesílání
- *Timeout* – zde se nastavuje maximální hodnota pro odesílání a zápis. Defaultně je nastavena na 10000 milisekund.
- *Error in* – do programu vstupuje hlášení o chybách z předešlého bloku. Jedná se o *cluster*, což je struktura obsahující položky různého typu. V tomto případě obsahující proměnnou typu *boolean* (je – není chyba), proměnnou typu *integer* s kódem chyby a proměnnou typu *string* s popisem zdroje chyby.

Výstupy:

- *VISA resource name* – zvolený zdroj komunikace
- *Error out* - obsahuje tytéž proměnné co *Error in*

Tvar a barva čáry reprezentující definovanou signálovou cestu rozlišuje typ proměnné, která prochází daným místem.

4.3.2 Zpracování návratové hodnoty

VI s názvem *Number to error* zpracovává návratovou hodnotu z řídící desky. Návratová hodnota je číslo, kterým mikrokontrolér odpovídá nadřazenému softwaru viz. tabulka 2.

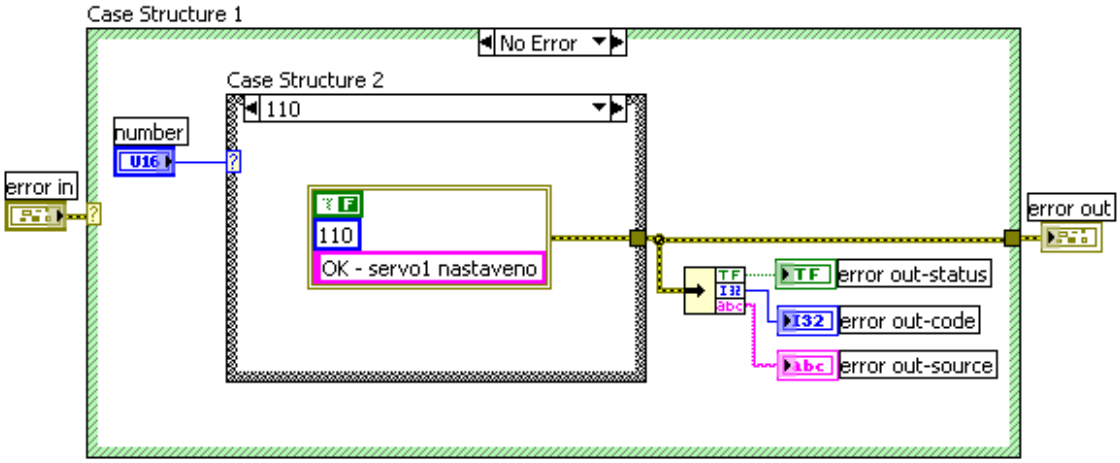
Vstupy:

- *error in* – do programu vstupuje hlášení o chybách z předešlého bloku
- *number* – číslo návratové hodnoty

Výstupy:

- *error out* – obsahuje tytéž proměnné co *error in*
- jednotlivé proměnné clusteru *error out*

Vstup *error in* vstupuje do *Case structure 1*, která vyhodnocuje, jestli přišla chyba z předchozího bloku či nikoliv. Pokud chyba přišla, tak se přepošle na výstup *error out*. Pokud žádná chyba nepřišla, tak se vyhodnotí návratová hodnota, která přichází na vstup *number*. *Case structure 2* vyhodnotí číslo na vstupu. Podle tohoto čísla se sestaví *cluster*, který pak vede na výstup *error out*. Obrázek 28 zachycuje zpracování návratové hodnoty číslo 110. Proměnná typu *boolean* clusteru *error out* se v tomto případě naplní hodnotou „*false*“, proměnná typu *integer* číslem „110“ a proměnná typu *string* řetězcem „*OK – servo1 nastaveno*“.



Obr. 28: Zpracování návratové hodnoty 110.

4.3.3 Převedení horního a dolního bajtu na šestnáctibitové číslo.

Pro nastavování čerpadla a serv nestačí osmibitová hodnota, ale je potřeba šestnáctibitová. Při komunikaci mezi počítačem a řídicí deskou se toto číslo posílá ve dvou bajtech označených jako *low byte* a *high byte*. VI s názvem *low high to num* převádí horní a dolní bajt ze vstupu zpět na šestnáctibitové číslo.

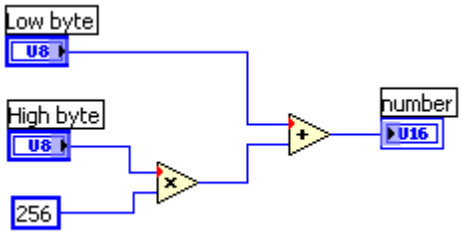
Vstupy:

- *Low byte* – dolní bajt
- *High byte* – horní bajt

Výtup:

- *number* – šestnáctibitové číslo složené z horního a dolního bajtu

Vlastní převod probíhá tak, že se horní bajt vynásobí číslem 256 a tato hodnota se pak sečte s hodnotou uloženou v dolním bajtu.



Obr. 29: Blokové schéma převodu.

4.3.4 Převedení šestnáctibitového čísla na horní a dolní bajt.

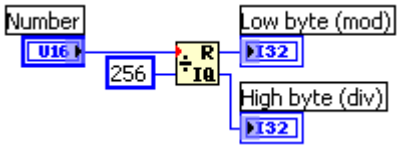
VI s názvem *num to low high* provádí opačný převod než předchozí popisované VI.

Vstup:

- *Number* – šestnáctibitové číslo

Výstupy:

- Low byte – dolní bajt
- High byte – horní bajt



Obr. 30: Převod šestnáctibitového čísla na dva bajty.

Šestnáctibitové číslo ze vstupu *Number* se podělí číslem 256. Celočíselný výsledek jde pak na výstup *High byte* a zbytek po dělení jde na výstup *Low byte*.

4.3.5 Získání čísla akce

Ve VI s názvem *action servo to number* získáváme číslo akce, které je součástí datového rámce při komunikace s řídící deskou.

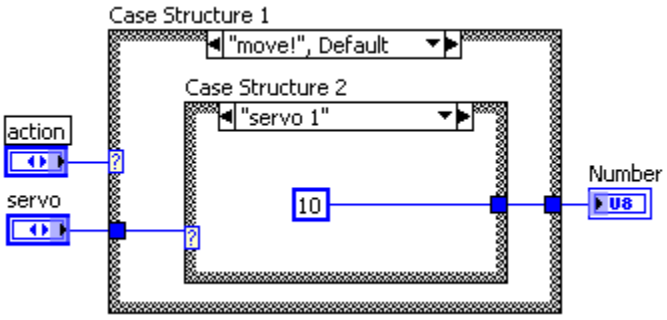
Vstupy:

- *action* – výběr druhu akce
- *servo* – výběr serva

Výstup

- *Number* – číslo akce

Vstup *action* je rozhodovací podmínka pro *Case structure 1* a vstup *servo* je rozhodovací podmínka pro *Case structure 2*. Obě struktury vyhodnotí podmínku a na základě těchto podmínek se odešle na výstup *Number* správné číslo akce.



Obr. 31: Získání čísla akce.

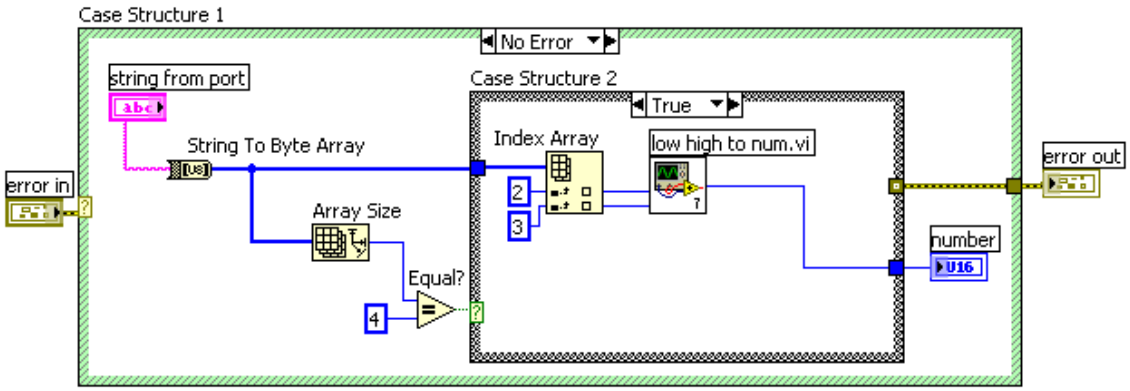
4.3.6 Převedení vstupního řetězce na pole čtyř bajtů

VI s názvem *string to number* převádí hodnotu ze vstupu, která je typu *string* na pole čtyř bajtů. Z posledních dvou bajtů pak složí číslo, které pošle na výstup *Number*.
Vstupy:

- *Error in* – do programu vstupuje hlášení o chybách z předešlého bloku
- *String to port* – vstup řetězce

Výstupy:

- *Error out* – obsahuje tytéž proměnné co *error in*
- *Number* – výstupní číslo



Obr. 32: Zpracování vstupního řetězce.

Vstup *error in* vstupuje do *Case structure 1*, která vyhodnocuje, jestli přišla chyba z předchozího bloku či nikoliv. Pokud chyba přišla, tak se přepošle na výstup *error out*. Pokud žádná chyba nepřišla, tak se načte hodnota ze vstupu *string from port*. Tato hodnota dále pokračuje do bloku *String To Byte Array*, kde se provede konverze řetězce na pole ASCII hodnot. Blok *Array Size* zjistí velikost pole. Tato hodnota se porovná s číslem čtyři, což je velikost datového rámce, který vysílá řídicí deska.

Pokud se hodnoty rovnají, je podmínka v *Case Structure 2* vyhodnocena jako *true*. Řetězec vstoupí do bloku *Index Array* v *Case Structure 2*. Zde se z pole hodnot

vezmou poslední dvě hodnoty a ty pokračují na vstup subVI *low high to num*. Toto subVI převede tyto dva bajty na číslo a to jde pak na výstup *number*.

Pokud se podmínka v *Case Structure 2* vyhodnotí jako *false*, tak se na výstup *number* pošle číslo nula a na výstup *error out* se pošle kód chyby nula a popis chyby: „Zkonvertovany retezec ma neplatnou delku“.

4.3.7 Sestavení výstupního řetězce

Řetězec pěti ASCII znaků se vytváří ve VI s názvem *build control string.*

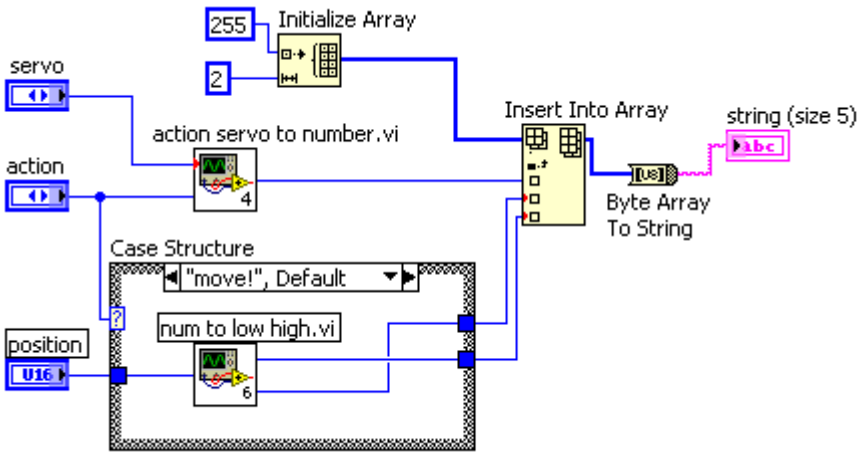
Vstupy:

- *servo* – výběr serva
- *action* – výběr druhu akce
- *position* – hodnota udávající pozici serva

Výstup:

- string – řetězec pěti ASCII znaků

Vytvoření prvních dvou bajtů se děje v bloku *Initialize Array*. Na vstup *element* se dá číslo 255 a na vstup *dimension size* číslo 2. Tím se vytvoří pole o dvou prvcích, kde má každý prvek hodnotu 255. Toto pole se pošle na vstup *array* bloku *Insert Into Array*. Dále se v subVI *action servo to number* na základě vstupu *action* a *servo* získá třetí bajt, což je číslo akce. Toto číslo se pošle na vstup *new element* bloku *Insert Into Array*.



Obr. 33: Vytvoření výstupního řetězce.

Vstup *action* je rozhodovací podmínkou pro *Case structute*. Pokud je vybrána akce „*move!*“, tak se načte hodnota ze vstupu *position.* Tato hodnota se v subVI *num to low high* převede na dolní a horní bajt a pošle se na výstup *Case structure*. Pokud je akce „*clear buffer*“ nebo „*get pulse number*“, tak se na výstup z *Case structure* pošlou dvě nuly. Tím se získaly poslední dva bajty, které jsou také poslány na vstup *new element* bloku *Insert Into Array*. Tento blok tedy sestaví nové pole o pěti bajtech a pošle je na svůj výstup *output array*. Odtud jde do bloku *Byte Array To String*, kde se pole převede na řetězec pěti ASCII znaků, které se objeví na výstupu *string* tohoto VI.

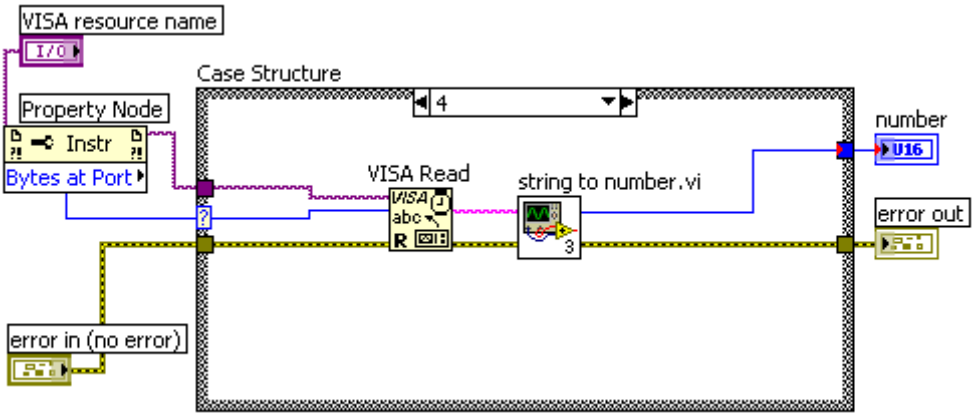
4.3.8 Získání čísla z COM portu

VI s názvem *COM read as number* přečte z COM portu datový rámec a vrátí číslo složené z posledních dvou bajtů.
Vstupy:

- *VISA resource name* – jméno zdroje, který má být otevřen pro komunikaci
- *error in* – do programu vstupuje hlášení o chybách z předešlého bloku

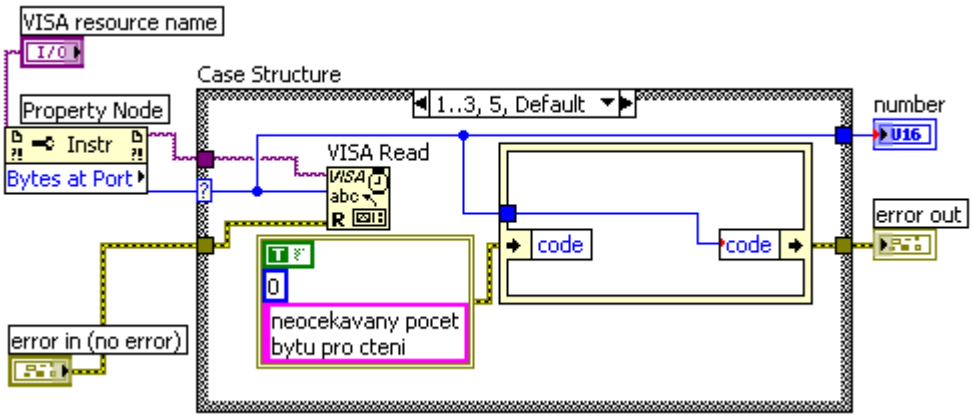
Výstup:

- *number* – číslo složené z posledních dvou bajtů
- *error out* – obsahuje tytéž proměnné co *error in*



Obr. 34: Načtení a zpracování správného řetězce z komunikačního portu.

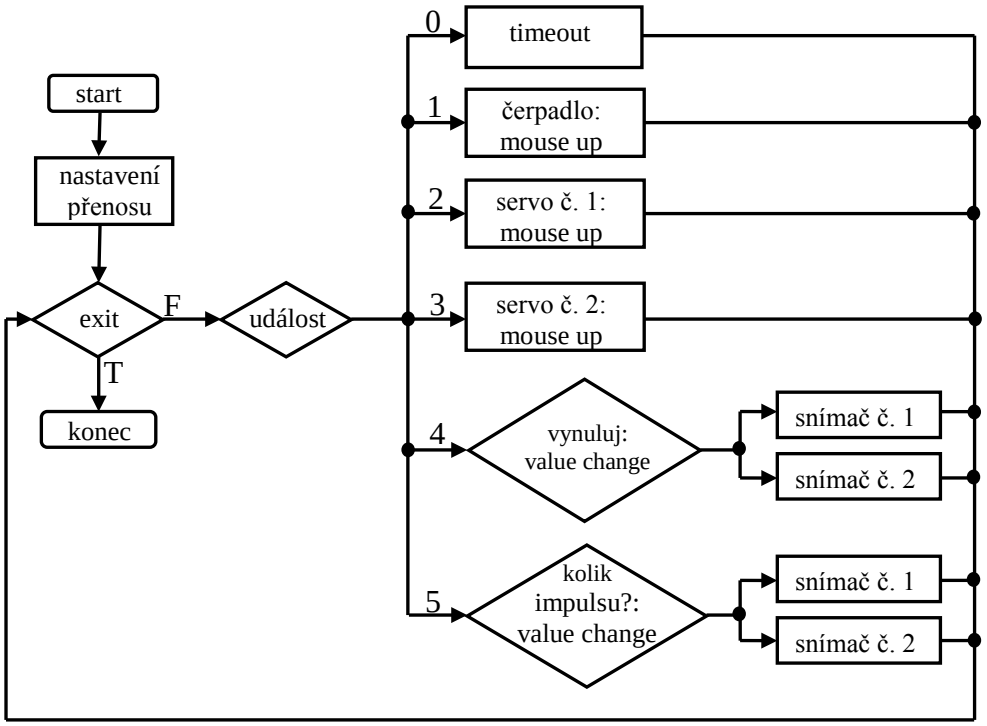
Jako podmínka do *Case structure* vstupuje počet bajtů na portu, jehož jméno se získá ze vstupu *VISA ressource name*. Pokud přišly čtyři bajty, tak se v bloku *VISA Read* načtou a pošlou se jako řetězec do subVI *string to number*. V tomto subVI se pak z posledních dvou bajtů složí číslo, které se pošle na výstup *number*. Pokud se podmínka v *Case structure* rovná nule, tak se sestaví *cluster*, který jde na výstup *error out*. Proměnná typu *boolean* clusteru *error out* se v tomto případě naplní hodnotou *„true“*, proměnná typu *integer* číslem „0“ a proměnná typu string řetězcem *„není co číst“*. Pokud má řetězec jinou velikost, vypadají proměnné *clusteru* takto: proměnná typu *boolean* je *„true“*, proměnná typu *integer* obsahuje číslo charakterizující velikost řetězce a proměnná typu string obsahuje hlášku *„neocekavany pocet bytu pro cteni“*.



Obr. 35: Zpracování neplatné velikosti řetezce.

4.3.9 Hlavní program

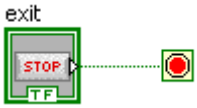
Z hlavního VI s názvem *mainEXO* se bude nastavovat komunikace a ovládat hydraulické rameno. Vývojový diagram hlavního VI je na obrázku 36. *Front Panel* se skládá ze dvou záložek. Na záložce *COM port settings* se nastavují vlastnosti komunikace. Na záložce *Control* jsou ovládací prvky pro ovládání hydraulického ramene. Jak vypadají karty aplikace *Com port settings* a *Control* je zobrazeno v příloze č. 5 a 6.



Obr. 36: Vývojový diagram hlavního VI.

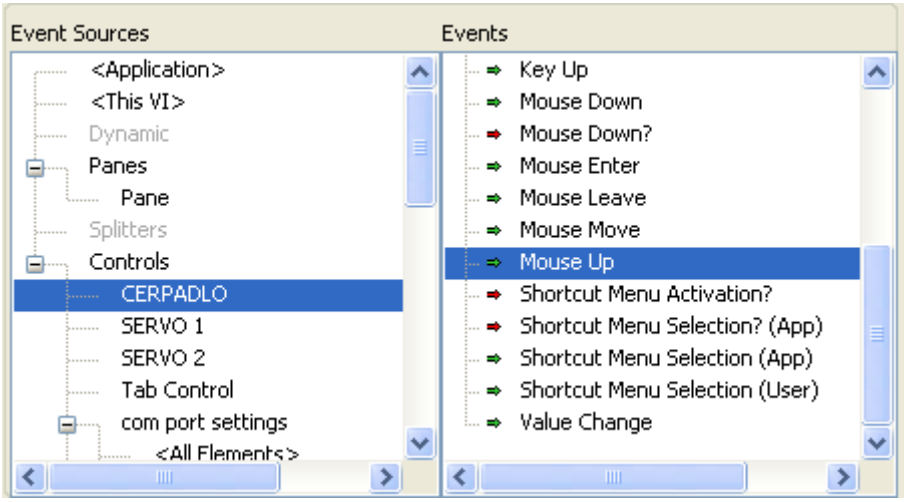
Vybrané vlastnosti komunikace jdou do subVI *set COM port*. Zde se provedou příslušná nastavení. Fialový výstup *VISA resource name* ze subVI vede do cyklu typu *while*. Tento cyklus provádí příkazy uvnitř smyčky, dokud není splněna zastavovací

podmínka. Ta je splněna pokud se klikne na tlačítko EXIT, které je umístěné na hlavním formuláři.



Obr. 37: Zastavovací podmínka cyklu while v hlavním programu.

Uvnitř *While Loop* je další řídící struktura tentokrát však ne *While* nýbrž *Event Structure.* Jedná se o jakési větvení programu. Tato struktura reaguje na určitou událost. Událost je zpráva, kterou vysílá aktivní objekt celému programu. V levém horním rohu struktury je nastavena hodnota *Timeout* na 300. Ta udává, kolik milisekund bude struktura čekat, než nastane událost. Události se nastavují tak, že se klikne pravým tlačítkem myši na strukturu *Event.* Zobrazí se menu, kde se vybere *Add Event Case.* Zde se pak nastaví vlastnosti události.



Obr. 38: Přidání události v Event Structure.

Tímto způsobem se nastaví celkem šest událostí, na které bude program reagovat:

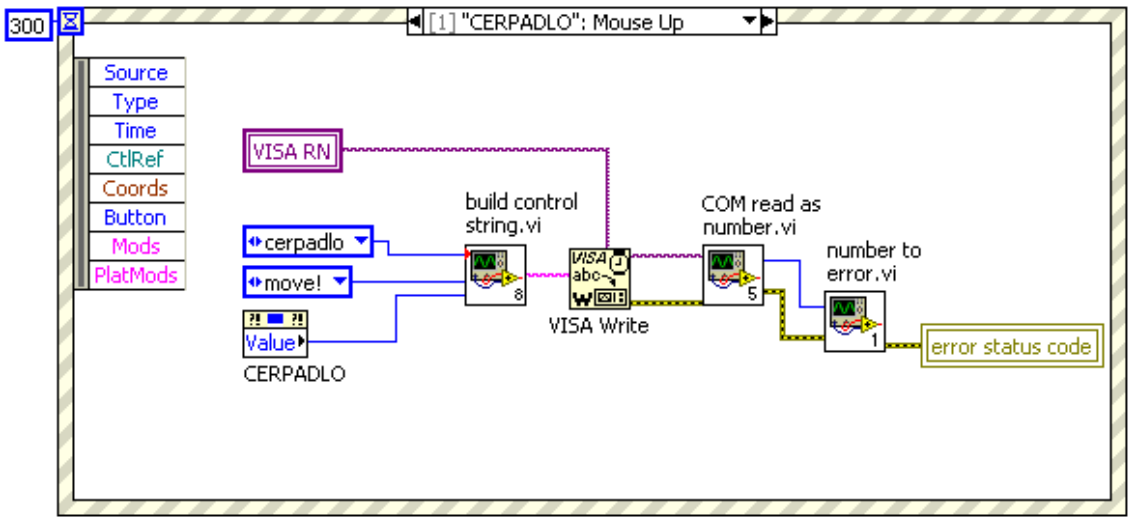
1) Událost „*[0]Timeout*“

Tato událost nastane po uplynutí času nastaveného v levém horním rohu struktury. Struktua pro tuto událost je prázdná a program nic nedělá.

2) Událost „*[1]CERPADLO: Mouse Up*“

Tato událost nastane po uvolnění tlačítka z komponenty pro ovládání čerpadla. Znamená to, že byla nastavena nová hodnota pro čerpadlo. V subVI *build kontrol string* se vytvoří řetězec o pěti znacích. Tento řetězec se pošle na vstup *write buffer* bloku *VISA Write*. Ten ho pošle na rozhraní definované na vstupu *VISA resource name*. V subVI *COM read as number* se přečte z portu definovaného ve *VISA ressource name* datový rámec a vrátí se číslo složené z posledních dvou bajtů. Toto číslo jde na vstup dalšímu subVI *number to error*. Zde se na základě příchozího čísla vyhodnotí, o jakou

návratovou hodnotu se jedná a její popis se pošle na komponentu *error status code*, kde se zobrazí uživateli.



Obr. 39: Událost pro nastavení čerpadla.

3) a 4) Událost „*[2]SERVO 1: Mouse Up*“ a událost „*[3]SERVO 2: Mouse Up*“

Princip je stejný jako při nastavení čerpadla. S tím rozdílem, že událost nastane po uvolnění tlačítka myši nad komponentou pro ovládání serva č. 1 respektive serva č. 2.

5) Událost „*[4]Vynuluj: Value Change*“

Tato událost nastane při stisknutí tlačítka „*Vynuluj*“. Jedná se o pokyn pro mikrokontrolér, aby vynuloval čítač impulsů. Na základě volby serva a akce *clear buffer* se sestaví v subVI *build kontrol string* řetězec, ve kterém je informace o čítači, který se má vynulovat. Řetězec se přes blok *VISA Write* zapíše na COM port. Stejně jako v předchozích případech se v subVI *COM read as number* zpracuje datový rámec, který poslal mikrokontrolér. V subVI *number to error* se vyhodnotí návratová hodnota, která se zobrazí i s popisem v komponentě *error status code*.

6) Událost „*[5]Kolik pulsu?: Value Change*“

Poslední z událostí nastane, když chce uživatel vědět počet impulsů ze snímače a klikne na tlačítko „*Kolik impulsu?*“. Tak jako v předchozích případech se i tady vytvoří v příslušném subVI řetězec, který se pošle na zpracování mikrokontroléru. Ten ho vyhodnotí a pošle nazpět odpovídající datový rámec, kde poslední dva bajty obsahují počet impulsů z příslušného čítače impulsů. Tento odpovědní datový rámec se zpracuje v subVI *COM read as number* a počet impulsů se uživateli zobrazí v příslušném okně.

ZÁVĚR

Cílem této diplomové práce byla realizace řídicího systému pro hydraulický manipulátor. Pro splnění tohoto cíle bylo nutné seznámit se s použitými prvky hydrauliky a senzory. Dále pak získat znalosti s programováním mikrokontrolérů v jazyce C a programováním v NI LabVIEW.

Zpracování návrhu řízení hydraulického mechanismu bylo pro mě osobně velmi přínosné. Umožnilo mi nahlédnout do problematiky návrhu a následné výroby elektronických zařízení. Kromě jiného jsem se naučil programovat mikrokontrolér ATmega16 v jazyku C a zdokonalil jsem se v programování v NI LabVIEW. Věřím, že získané poznatky, nejen v oblasti programování, ale také v pochopení přínosu dílčích projektů pro jejich následné zařazení do komplexního řešení složité problematiky, využiji ve své budoucí praxi.

První část práce se zabývá použitými hydraulickými prvky a senzory. Seznámení se s jejich funkcí a parametry bylo nezbytné pro návrh řídicího systému. Jak již bylo řečeno v úvodu, firma, u které byly zakoupeny hydromotory, čerpadlo a rozvaděč, již přestala modelářskou hydrauliku vyrábět a nebyla schopna poskytnout bližší technické specifikace jednotlivých komponent. Přesto základní zjištěné parametry stačily k vytvoření představy o jejich možnostech a funkci.

V dalším kroku byl navržen a vyroben elektronický řídicí sub-systém. Ten se skládá ze tří desek. Každá deska má na starost nějaké zařízení a stará se o základní komunikaci mezi tímto zařízením a nadřazeným PC. Všechny desky jsou osazeny mikrokontrolérem ATmega16. Bylo tedy nutné se s tímto mikrokontrolérem dobře seznámit. Programy pro mikrokontroléry byly vytvořeny v jazyce C.

Jako poslední byl vytvořen nadřazený řídicí software. Ten byl podle zadání naprogramován v prostředí NI LabVIEW. Pomocí vytvořeného softwaru lze ovládat hydromotory a čerpadlo a tím nastavovat polohu manipulátoru a rychlost jeho pohybu. Dále je možné zobrazit údaj o počtu impulsů vyslaných ze senzorů.

Práce bude sloužit pro výukové účely a studenti se jejím prostřednictvím budou moci seznámit s fungováním hydraulického mechanismu a s možností jeho řízení. Výsledek práce lze využít jako podklad pro další modernější verzi hydraulického ramene, kde budou stávající relativní snímače nahrazeny absolutními snímači polohy a modelářská hydraulika bude vyměněna za mikro-hydrauliku.

SEZNAM POUŽITÉ LITERATURY

[1] ZEMAN, Václav. *Konstrukce elektronických zařízení*: Počítačový *návrh plošných spojů – cvičení*. 1. vyd. Brno: VUT v Brně, Fakulta elektrotechniky a informatiky, Ústav telekomunikací, 1999. 29 s. Dostupný z WWW: <http://www.vabo.cz/stranky/biolek/vyukaVUT/skripta/Eagle.pdf>.

[2] VODRÁŽKA, Jakub. Návrh konstrukce hydraulicky ovládaného ramene. [s.l.], 2008. 51 s. FSI VUT v Brně, Ústav automatizace a informatiky. Vedoucí bakalářské práce Ing. Stanislav Věchet, Ph.D. Dostupný z WWW: <http://uai.fme.vutbr.cz/szz/2008/BP_Vodrazka.pdf>.

[3] Bastlíř & spol.. Jak fungují modelářská serva? [online]. [2000] , Datum poslední aktualizace 15.10.2008 [cit. 2009-04-23]. Dostupný z WWW: <http://vlastikd.webz.cz/bastl/serva.htm>.

[4] Pelikán. Serva : Popis serva [online]. [2008] [cit. 2009-04-23]. Dostupný z WWW: <http://www.rcm-pelikan.cz/index.php?sec=list&storage=71#5>.

[5] AJP-tech spol. s r.o.. *Magnetic sensors*. [s.l.] : [s.n.], [2005?]. 22 s. Dostupný z WWW: <http://www.baumerelectric.com/usa/news/2008sensor%20_solution_%20catalog/magnetic.pdf>.

[6] JURÁNEK, Antonín. Výuka návrhového systému EAGLE. 1. vyd. [s.l.] : [s.n.], 2004. 107 s. Dostupný z WWW: <http://om0tm.nagano.cz/down/eaglemean.pdf>.

[7] *Výroba plošných spojů* [online]. 2008 [cit. 2009-04-20]. Dostupný z WWW: <http://ok1kvk.cz/web/index.php/technika/55-navody/425-vyroba-plonych-spoj>.

[8] MATOUŠEK, David. *Práce s mikrokontroléry ATMEL AVR - ATmega16* 1. vyd. Praha : BEN – technická literatura, 2006. 320 s. , 1 CD-ROM. 4. díl - uP a praxe ISBN 80-7300-174-8.

[9] Atmel. *ATmega16*. [s.l.] : [s.n.], 2008. 358 s. Dostupný z WWW: <http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf>.

[10] FIEDLER, Jindřich. *Jednotka pro měření větru a globální radiace*. [s.l.], ČVUT FEL katedra počítačů 2006. 73 s. Vedoucí bakalářské práce Ing. Martin Novotný. Dostupný z WWW: <https://dip.felk.cvut.cz/browse/pdfcache/fiedlj1_2006bach.pdf >.

[11] ČERMÁK, Martin. *Moderní měřicí přístroje*. Brno : Přírodovědecká fakulta Masarykovy univerzity v Brně, 2003. 85 s. Vedoucí diplomové práce RNDr. Zdeněk Bochníček, Dr. Dostupný z WWW: <http://lsd.linux.cz/pub/muni.cz/physics/education/diploma/cermak/dp.pdf>.

[12] National Instruments. *Manuals : LabVIEW* [online]. [2006] [cit. 2009-05-03]. Dostupný z WWW: <http://digital.ni.com/worldwide/czech.nsf/sb/Download?OpenDocument&node=201494_cs>.

[13] PITAŠ, Martin. *Realizace hydraulického obvodu pomocí systému mikro-hydrauliky*. FSI VUT v Brně, Ústav automatizace a informatiky, 2009. 39 s. Vedoucí bakalářské práce Ing. Stanislav Věchet, Ph.D.

SEZNAM PŘÍLOH

Příloha č. 1: Schéma hydraulického obvodu

Příloha č. 2: Schéma řídicí desky

Příloha č. 3: Seznam součástek

Příloha č. 4 : Kód programu řídicí desky č. 1

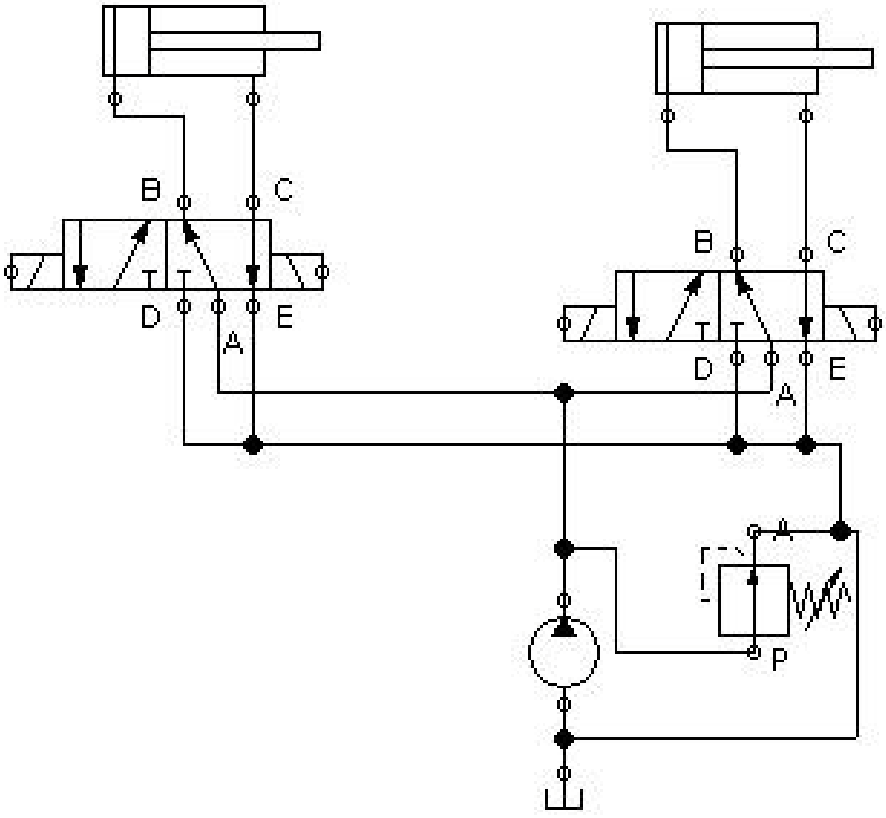
Příloha č. 5: Ovládací panel – karta Control

Příloha č. 6: Ovládací panel – karta Set COM port

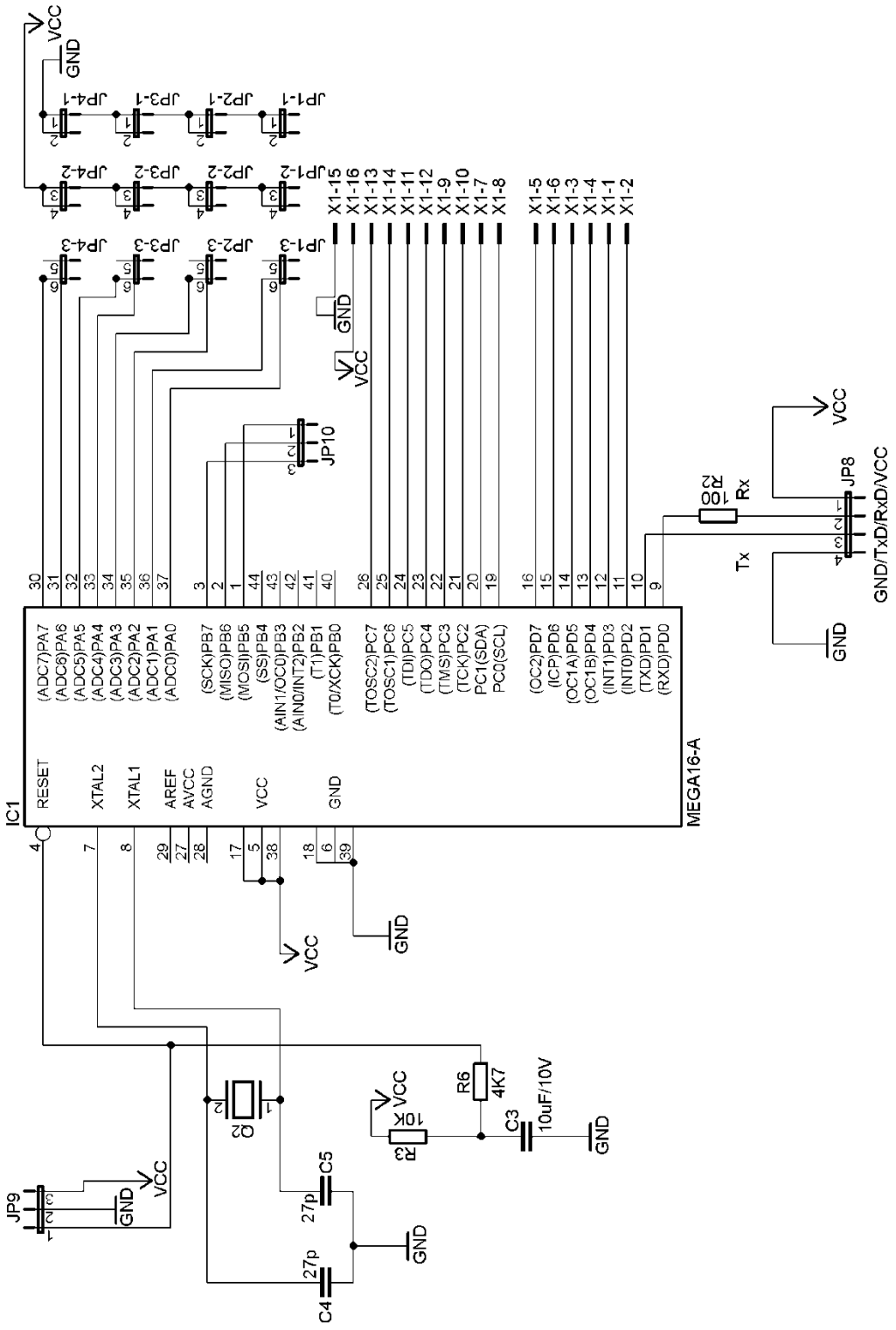
Příloha č. 7: Disk CD-ROM s obsahem:

- Elektronická podoba této práce (DP_VOTAVA.pdf)
- Software pro řídicí desky vytvořený v ICC AVR
- Nadřazený řídicí software vytvořený v NI LabVIEW

Příloha č.1: Schéma hydraulického obvodu



Příloha č. 2: Schéma řídicí desky



Příloha č. 3: Seznam součástek

Kód (GM elektronics)	Název	Cena (s DPH)	Počet kusů	Celková cena (s DPH)
900 - 179	R1206 10K 1%	2 Kč	3	6 Kč
901 - 248	R1206 4K7 1%	2 Kč	3	6 Kč
902 - 192	R1206 100R 1%	2 Kč	3	6 Kč
905 - 069	CK 1206 27P/50V	2,50 Kč	6	15 Kč
906 - 117	CK 0805 10M/10V	4 Kč	3	12 Kč
131 - 077	QM 11,059MHz	10 Kč	3	30 Kč
958 - 122	Atmega16 - 16AU	65 Kč	3	195 Kč
800 - 006	MLW 16G	7 Kč	3	21 Kč
800 - 171	PSH02-04WG	2 Kč	3	6 Kč
832 - 023	S2G10	4 Kč	3	12 Kč
832 - 017	S1G20	8 Kč	5	40 Kč
celková cena všech součástek				349 Kč

Příloha č. 4: Kód programu řídicí desky č.1

```
//ICC-AVR application builder
// Target : M16
// Crystal: 11.059Mhz

#include <iom16v.h>
#include <macros.h>

 //TCNT1 = 60007;            //0.5ms
 //TCNT1 = 48948;            //1.5ms
 //TCNT1 = 37889;            //2.5ms

#define MIN_PULSE_WIDTH       60007
#define MAX_PULSE_WIDTH       37889
#define MIDDLE                 11059

int timeoutcount = 0;
int count=0;                                                    //pro timer 0
int counter = MAX_PULSE_WIDTH + MIDDLE;                         //pro timer 1
int serpos[1] = {MIDDLE};
//obdrzene informace
int timer_value = MIDDLE;
int action = 0;

int distance = 0;                   //namerena vzdalenost od nuly v poctech impulsu
int first = 0;                      //prvni hodnota ze snimace
int next = 0;                       //nasledujici hodnota
int pulses[5] = {20,80,40,160,20};  //impulsy ze snimace v jednom smeru

//================================== SENSOR UTILS ==================================
int readSensor(void){
  //channel A PINC 0x02
  //channel B PIND 0x40
  //output 20 - A
  //      40 - B
  //      80 - both
  //      160 - none
  //direction 20 - 80 - 40 - 160 - 20
  //                    20 - 160 - 40 - 80 - 20
  if( ((PINC&0x02)==0)&&((PIND&0x40)==0x40) ){
```

```
   return 20;
  }
  if( ((PINC&0x02)==0x02)&&((PIND&0x40)==0) ){
   return 40;
  }
  if( ((PINC&0x02)==0)&&((PIND&0x40)==0) ){
   return 80;
  }
  if( ((PINC&0x02)==0x02)&&((PIND&0x40)==0x40) ){
   return 160;
  }
  return 0;  //error
}

//====================================== COMUNICATION ==============================
void trans(int value)
{
    while ( !( UCSRA & (1<<UDRE)) ) ;
    UDR = 255;
    while ( !( UCSRA & (1<<UDRE)) ) ;
    UDR = 255;
    while ( !( UCSRA & (1<<UDRE)) ) ;
    UDR = (char)value;
    while ( !( UCSRA & (1<<UDRE)) ) ;
    UDR = (char)(value>>8);
}

int receive(void)
{
 int result=1;//error
 int val=0;
    while ( !( UCSRA & (1<<RXC)) ) ;                    // cekani na prichozi data
              val = UDR;
              if(val == 255)
              {
             while ( !( UCSRA & (1<<RXC)) ) ;           // cekani na prichozi data
                   val = UDR;
                   if(val==255)
                   {
                   while ( !( UCSRA & (1<<RXC)) )       // servo
                     action = UDR;
```

```
                    while ( !( UCSRA & (1<<RXC)) )            // low
                       val = UDR;
                        while ( !( UCSRA & (1<<RXC)) ) ;      // high
                           timer_value = val + UDR*256;
                              result = 0;                      //ok
                    }
                }
 return result;
}
//================================== SENSOR INIT ==================================
void sensor_init(void)
{
 distance = 0;
 first = readSensor();
 next = first;
}
// ================================== PORTS INIT ==================================
void port_init(void)
{
 PORTB = 0xFF;
 DDRB  = 0x00;
 PORTC = 0x00;    //m103 output only
 DDRC  = 0x01;    //vystupni je pouze pin0 ADC0
 PORTD = 0x00;
 DDRD  = 0x00;
}

//====================================== UART ======================================
//UART0 initialisation
// desired baud rate: 9600
// actual: baud rate:9600 (0.0%)
// char size: 8 bit
// parity: Disabled
void uart0_init(void)
{
 UCSRB = 0x00; //disable while setting baud rate
 UCSRA = 0x00;
 UCSRC = 0x86;
 UBRRL = 0x47; //set baud rate lo
 UBRRH = 0x00; //set baud rate hi
 UCSRB = 0x18;
```

```
}

// ============================ TIMERS ============================
//TIMER0 initialisation - prescale:1024
// WGM: Normal
// desired value: 50Hz
// actual value: 50.232Hz (0.5%)
void timer0_init(void)
{
 TCCR0 = 0x00;     //stop
 TCNT0 = 0x29;    //set count
 TCCR0 = 0x05;    //start timer
}

#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
  //TIMER0 has overflowed
  //with of pulse is 20ms
  TCNT0 = 0x29; //reload counter value

  count = 1;
  timeoutcount++;
}

//TIMER1 initialisation - prescale:1
// WGM: 0) Normal, TOP=0xFFFF
// desired value: 666.663Hz
// actual value: 666.687Hz (0.0%)
void timer1_init(void)
{
 TCCR1B = 0x00; //stop
 TCNT1H = 0xBF; //setup
 TCNT1L = 0x34;
 OCR1AH = 0x40;
 OCR1AL = 0xCC;
 OCR1BH = 0x40;
 OCR1BL = 0xCC;
 ICR1H  = 0x40;
 ICR1L  = 0xCC;
 TCCR1A = 0x00;
```

```
 TCCR1B = 0x01; //start Timer
}

#pragma interrupt_handler timer1_ovf_isr:9
void timer1_ovf_isr(void)
{
  //TIMER1 has overflowed
 //TIMER1 has overflowed
 //TCNT1 = 60007;//0.5ms
 //TCNT1 = 48948;//1.5ms
 //TCNT1 = 37889;//2.5ms

  TCNT1 = counter;

  if (count==2){ PORTC = 0x00; count++;}
  if (count==1){ PORTC = 0x01; counter = MAX_PULSE_WIDTH+serpos[0];       count++;} //servo 1
}

//TIMER2 initialisation - prescale:8
// WGM: Normal
// desired value: 64KHz
// actual value: 65.827KHz (2.8%)
void timer2_init(void)
{
 TCCR2 = 0x00; //stop
 ASSR  = 0x00; //set async mode
 TCNT2 = 0xEB; //setup
 OCR2  = 0x15;
 TCCR2 = 0x02; //start
}

#pragma interrupt_handler timer2_ovf_isr:5
void timer2_ovf_isr(void)
{
  int dir;
  int sensor;
  TCNT2 = 0xEB; //reload counter value

  //precteni senzoru
  if( ((PINC&0x02)==0)&&((PIND&0x40)==0x40) ){
    first = 20;
```

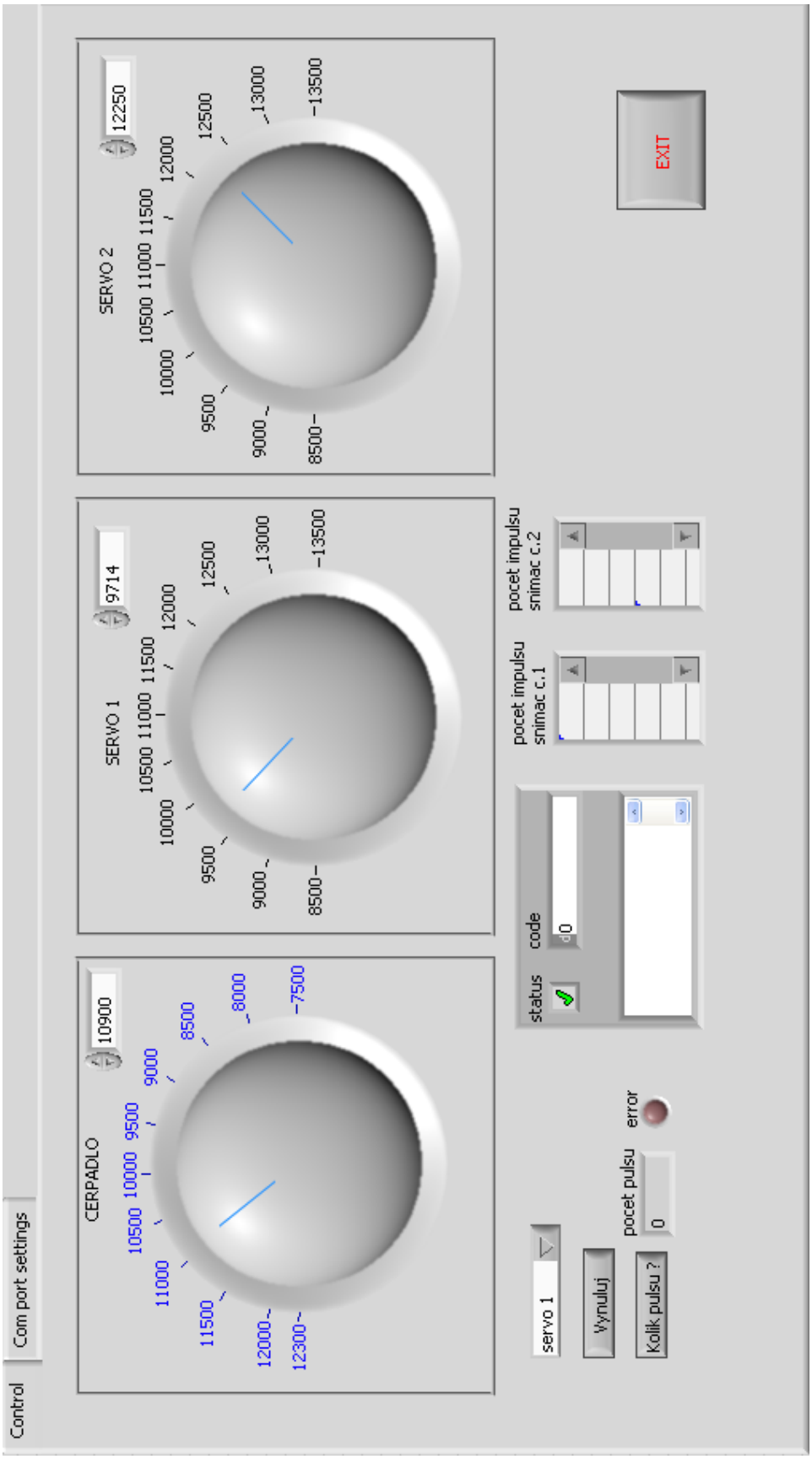
```
 }
 else if( ((PINC&0x02)==0x02)&&((PIND&0x40)==0) ){
  first = 40;
 }
 else if( ((PINC&0x02)==0)&&((PIND&0x40)==0) ){
  first = 80;
 }
 else if( ((PINC&0x02)==0x02)&&((PIND&0x40)==0x40) ){
  first = 160;
 }
 else
  first = 0;//error

  //urceni smeru pohybu
  if((20==first)&&(80==next))
            dir = 1;                                    //jeden smer
          else if((20==next)&&(80==first))
            dir = -1;                                   //druhy smer
  else if((80==first)&&(40==next))
            dir = 1;                                    //jeden smer
          else if((80==next)&&(40==first))
            dir = -1;                                   //druhy smer
  else if((40==first)&&(160==next))
            dir = 1;                                    //jeden smer
          else if((40==next)&&(160==first))
            dir = -1;                                   //druhy smer
  else if((160==first)&&(20==next))
            dir = 1;                                    //jeden smer
          else if((160==next)&&(20==first))
            dir = -1;                                   //druhy smer
          else
            dir = 0;

 //vypocet vzdalenosti

 if(1==dir) distance++;
 if(-1==dir) distance--;
 next = first;
}
```

```
// ================================ INIT DEVICES ==================================

//inicializace vsech periferii
void init_devices(void)
{
 CLI();
 port_init();
 uart0_init();
 sensor_init();    //musi probehnout po inicializaci AD, ale pred timerem
 timer0_init();
 timer1_init();
 timer2_init();

 MCUCR = 0x00;
 GICR  = 0x00;
 TIMSK = 0x45;
 SEI();
}

======================================== MAIN ================================
void main(void)
{
 int val;
 init_devices();

 while (1)
 {
   if(0==receive()){
            switch (action) {

            case 0:
                //nastaveni cerpadla – obsluhuje ridici deska c.3
                /*
                if ((timer_value>=10500)&(timer_value<=12300)){
                   serpos[0] = timer_value;
                   trans(102);    //nastaveni OK
                }
                else{
                    trans(103);       //error - spatna hodnota pro timer cerpadla
                 }
                   */
```

```
      break;

 case 10:
   //nastaveni serva c.1
              if((timer_value>=0)&(timer_value<=22118)){
                  trans(110); //nastaveni OK
              serpos[action] = timer_value;
              }
              else{
                   trans(111);     //error - chybna hodnota pro timer serva c.1
              }
break;

case 11:
     //nastaveni nuly pocitadla impulsu snimace c.1
      sensor_init();
      trans(112); //nastaveni nuly OK
 break;

case 12:
       //pocet impulsu - snimac c.1
        trans(distance);
 break;

case 20:
         //nastaveni serva c.2 - obsluhuje ridici deska c. 2
            /*
         if((timer_value>=0)&(timer_value<=22118)){
            trans(120); //nastaveni OK
            serpos[action] = timer_value;
          }
             else{
                trans(121); //error -spatna hodnota pro timer serva c.2
              }
           */
 break;

case 21:
  //nastaveni nuly pocitadla impulsu snimace c.2 -  obsluhuje ridici deska c. 2
             /*
                    sensor_init();
```

```
                        trans(122);   //nastaveni nuly - OK
                    */
          break;


        case 22:
              //pocet pulsu - snimac c.2 - obsluhuje ridici deska c. 2
                // trans(distance);
          break;

       default:
              trans(100); //error - spatna akce
        break;
        } //konec switch
         } //konec if
       else{
          trans(101); //error - chyba v prenosu
         }
   }//konec while
}
```

Příloha č. 5: Ovládací panel – karta Control



Příloha č. 6: Ovládací panel – karta Set COM port