ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta Elektrotechnická
Katedra řídící techniky

BAKALÁŘSKÁ PRÁCE

Návrh číslicového řízení stejnosměrného momentového motoru

Vypracoval: Ota Fejfar
Vedoucí práce: Ing. Zdeněk Hurák, Ph.D.

2007

Anotace

Cílem této práce je získat základní zkušenosti z oblasti číslicového řízení pro jejich další využití na jiných projektech Katedry řídící techniky. Práce se zabývá komplexním návrhem řídícího systému stejnosměrného motoru.

V první části této práce je řešena hardwarová část systému. Ta zahrnuje výběr čidla a návrh vhodné elektroniky na nepájecím poli. Elektronika je rozdělena na část zpracování signálu z čidla a část zpracování signálu PWM směrem k motoru. Středem elektroniky je mikroprocesor v němž je implementován příslušný program.

V druhé části je řešen návrh jednoduchého regulátoru. Nejprve je vytvořen číslicový model elektronického systému v prostředí MATLAB/Simulink. Dále je tento číslicový model doplněn v reálný model zpětnovazebního řízení reálného motoru. Pomocí vytvořeného Simulinkového modelu je nakonec provedena experimentální identifikace přenosové funkce reálného motoru a vyzkoušení několika různých regulátorů. Výsledky jsou porovnány s reálným systémem.

Annotation

The goal of this thesis is to acquire basic experience in the digital control area for its following use in some other projects of Department of Control Engineering. The thesis considers a complex concept of control system of direct current motor.

In the first section of this thesis is solved the hardware part of the system. It includes a sensor selection and appropriate electronic concept on non-soldering field. The electronics is devided into the part of sensor signal processing and part of PWM signal processing. In the middle of electronics is situated a processor inside which is appropriate program implemented.

In the second section of this thesis is solved a concept of simple controller. First is created a digital model of electronic system in MATLAB/Simulink environment. This digital model is then completed into a real model of feedback control of a real motor. With the aid of the created Simulink model is finally made an experimental identification of step response of real motor and tested some different controllers. Products are compared with real system.

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, SW, atp.) uvedené v přiloženém seznamu.

V Praze dne……. ……..

Podpis

Poděkování

V prvé řadě bych chtěl poděkovat vedoucímu své bakalářské práce Ing.Zdeňku Hurákovi, Ph.D. za velice pozitivní a zapálený přístup. Dále pak všem, kteří jakkoliv přispěli k dokončení této práce. V neposlední řadě děkuji své rodině a přítelkyni za ohromnou trpělivost a morální podporu.

Obsah

1 Úvod

1.1 Motivace

Již mnoho let člověk naráží na problémy a klade takové požadavky na svůj okolní svět, jejichž řešení nebo realizace je pro něj bez pomůcek přinejmenším zdlouhavý proces. Kvůli tomu vymyslel různá zařízení, postupy a programy, které řeší tyto problémy za něj a on mohl svůj potenciál vrhnout do dalšího pokroku, do tvorby mnohem „větších“ věcí. Do popředí zájmu se dostali pojmy „řízení“, „regulace“ a „automatizace“. K dosažení již zmiňovaných „větších“ věcí se musí i tyto pojmy a především jejich význam dále vyvíjet. Od analogového řízení zprostředkovávaného fyzickou zpětnou vazbou a PID regulátorem se přechází k mnohem flexibilnějšímu číslicovému řízení zajišťované digitálním výpočtem uvnitř PC nebo samotného mikroprocesoru, které je považováno za směr dalšího vývoje oboru. Z tohoto důvodu a také proto, že bych se rád v budoucnu této problematice věnoval, jsem se rozhodl seznámit se s problematikou číslicového řízení prostřednictvím této práce. Pro co největší budoucí samostatnost při řešení složitějších problémů musím a chci návrh provést od píky, tedy včetně volby a sestavení hardwaru pro realizaci řízení.

1.2 Číslicové řízení

Většina dnešních řídících systémů používají digitální počítače (resp. mikroprocesory) pro implementaci regulátoru. To umožňuje snáze propojit regulační systém se stávající (digitální) výpočetní technikou a tím získat větší přehled a kontrolu nad regulačním procesem. Regulace zde není prováděna spojitě, ale diskrétně v pevných časových intervalech daných periodou vzorkování.

Hlavními výhodami je již zmiňovaná kontrola nad systémem, větší odolnost proti rušení, minimální náklady na opravy či modifikace systému, velká flexibilita. Poslední dvě zmiňované výhody vyplývají z možnosti modifikace systému pouhou změnou implementovaného programu. Není zde vždy nutnost měnit hardware.

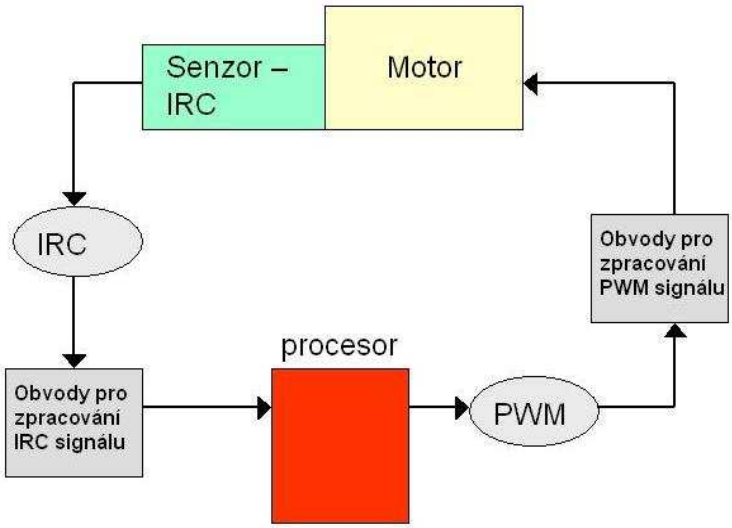
Nevýhodou je bezesporu ztráta informace a možnost vzniku aliasingu v důsledku pevné periody vzorkování. Tyto problémy je možno minimalizovat vhodnou volbou mikroprocesoru a optimalizací kódu programu zajišťujícího regulační výpočet.

1.3 Cíl práce

Cílem je vytvořit číslicový regulační systém pro stejnosměrný momentový motor. Tento systém by měl být co možná nejvíce konečný, tzn. že by měl být schopen řídit všechny stejnosměrné motory podobných elektrických vlastností (jako je například maximální napětí na motoru nebo maximální proud kotvou atp.) s co nejmenšími zásahy do systému. Důraz je kladen především na regulaci polohy, regulace rychlosti je pro účel této práce vedlejším cílem.

2 Koncepce řídícího systému

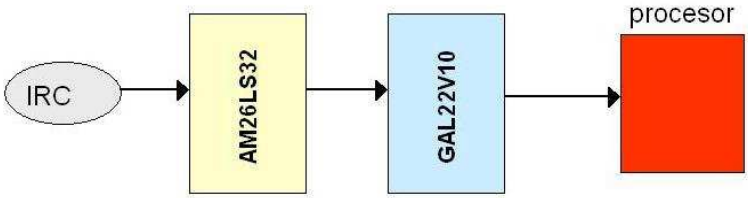
Blokové schéma celé soustavy je zobrazeno na obr.1. Předmětem řízení je stejnosměrný motor, ke kterému je mechanicky připevněn inkrementální senzor polohy (IRC). Výstupem IRC jsou IRC signály, které jsou zpracovány příslušnými logickými obvody a přivedeny jako vstup do mikroprocesoru. Mikroprocesor provádí výpočet v nekonečné smyčce a generuje PWM signál, který prostřednictvím příslušné elektroniky ovládá napájení motoru a tím i jeho chování.



obrázek 1 – blokové schéma řídícího systému

2.1 Obvody pro zpracování IRC signálu

Blokové schéma obvodů pro zpracování IRC signálu je zobrazeno na obr.2. Signály IRC jsou nejdříve zpracovány budičem sběrnice 422/423 AM26LS32 s třístavovým výstupem. Výstupní signály jsou dále převedeny hradlovým polem GAL22V10 na signál, který je očekáván v mikroprocesoru. Oběma obvodům se budu detailněji věnovat v kapitole 3.

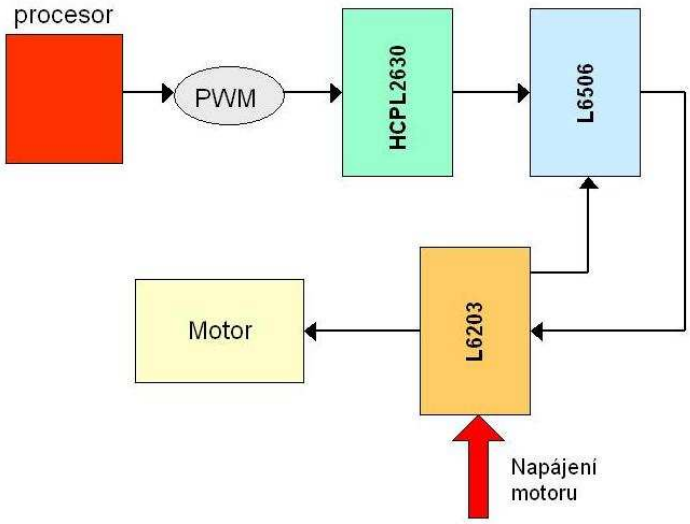


obrázek 2 – zpracování IRC signálu (blokové schéma)

2.2 Obvody pro zpracování PWM signálu

Blokové schéma obvodů pro zpracování PWM signálu je na obr.3. PWM signál je generován mikroprocesorem na základě vnitřního výpočtu mikroprocesoru. Tento signál prochází optickým vazebním členem HCPL2630, který slouží jako galvanické oddělení elektronické části od výkonové, čili jako ochrana elektronické části. Galvanicky oddělený signál je pak přiveden přes obvod L6506 na H-můstek L6203, který umožní regulaci napájení motoru v závislosti na příchozím PWM signálu. Obvod L6506 je zde umístěn jako hardwarová ochrana motoru. Ochrana je zajištěna kontrolou proudu protékajícího motorem a obvodem L6203.

Podrobně budu všechny zmíněné obvody popisovat v kapitole 3.



obrázek 3 - zpracování PWM signálu (blokové schéma)

3 Návrh elektroniky

V kapitole 2 byla nastíněna obecná koncepce návrhu hardwarové části a zmíněny prakticky všechny součástky, které byli použity. Vše však bylo popsáno obecně, pro vytvoření si základní představy. V této kapitole bych se chtěl detailněji věnovat návrhu elektroniky. Chtěl bych zde popsat použité součástky, jejich význam v systému, zapojení a u některých i jakým způsobem přímo spolupracují s ostatními součástkami.

3.1 IRC čidlo

V kanceláři G203 je k dispozici vzorkové IRC čidlo SR 580 od firmy ESSA Praha s 2500 periodami výstupního signálu na otáčku. Rozhodl jsem se této možnosti využít a použít jej pro svůj návrh.

Výrobce k tomuto čidlu uvádí:

SR 580x je optoelektronický inkrementální snímač, určený pro převod rotačního pohybu na elektrické pulsy. Počet těchto pulsů je přímo úměrný úhlu natočení, jejich frekvence rychlosti otáček.

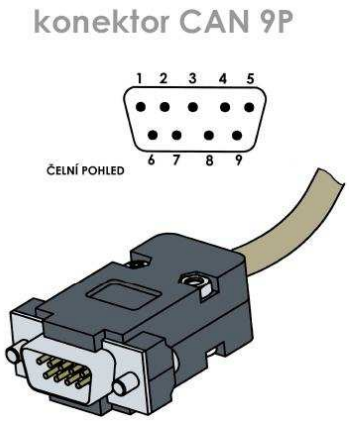
SR 580x tvoří základní těleso z hliníkové slitiny v němž je uložen systém odečítání, skleněný rastrový pár, osvětlovací systém s diodou LED a vyhodnocovací elektronika. Transparentní systém odečítání využívá skleněný kotouč s ryskami a jednou referenční značkou na otáčku. Rotační pohyb nerezového hřídele umožňují oboustranně krytovaná, předepnutá ložiska. Celý snímač je zakryt zatěsněným krytem s výstupním konektorem resp. vývodkou s kabelem v osové či boční poloze.

Důležitá data použitého IRC čidla jsou uvedena v tabulce 1.

rychlost otáčení	10 000 ot./min
počet period na otáčku	2500
napájecí napětí	5 Vss +/- 5%
napájecí proud	100 mA +/- 10%
výstupní frekvence	max. 300 kHz
pracovní teplota	0 ℃ až 60 ℃
skladovací teplota	-20℃ až +70 ℃
Výstup. signál při napájení UN = 5V:	RS 422
konektor	CAN 9P

tabulka 1 – Důležitá data použitého IRC čidla

Konektor CAN 9P je zobrazen na obr.4. Zapojení konektoru a používané signály jsou uvedeny v tab.2.



obrázek 4 – konektor CAN 9P

signál	CAN 9P
+5V	2
0V	1
1	9
1 NON	5
2	8
2 NON	4
3 (referenční)	7
3 NON	3
stínění	6

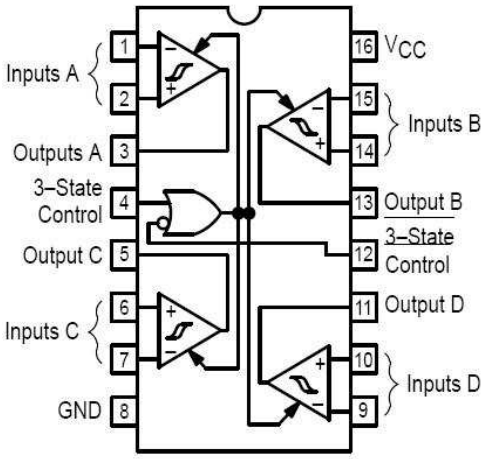
tabulka 2 – zapojení konektoru CAN 9P

3.2 Obvody pro zpracování IRC signálu

Obecné blokové schéma je znázorněno na obr.2. Následující řádky budou věnovány podrobnějšímu popisu obvodu AM26LS32 a hradlovému poli GAL22V10.

3.2.1 Obvod AM26LS32

Jedná se o budič sběrnice RS 422 s třístavovým výstupem. V této aplikaci slouží jako první zpracování signálu z IRC čidla. Na obr.5 je rozložení konektorů a vnitřní zjednodušené schéma obvodu AM26LS32.

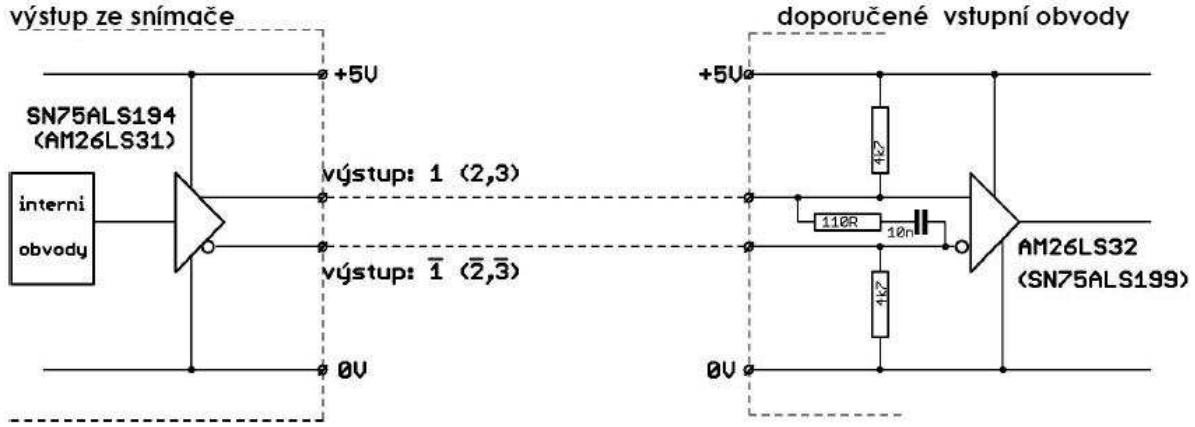


obrázek 5 – rozložení konektorů obvodu AM26LS32

Jak je vidět z obr.5, obvod se skládá ze 4 nezávislých přijímacích řetězců. Na vstup každého řetězce musí být přivedena log.1 na vstup „+“ a log.0 na vstup „-“, aby na výstupu byla log.1. Přivedeme-li na vstup řetězce log.0 na vstup „+“ a log.1 na vstup „-“, bude na výstupu tohoto řetězce log.0. Ovládání třetího stavu není pro tuto aplikaci důležité, proto je vstup třístavové kontroly trvale připojen (pin4 je trvale připojen na log.1 a pin12 na log.0). Tím jsou všechny řetězce neustále aktivní.

Jak již bylo řečeno, obvod AM26LS32 zajišťuje celkem 4 přijímací řetězce, kdy na vstupu každého řetězce je očekáván logický signál a jeho negace. Pohledem na tab.2 je vidět, že výstupem IRC čidla jsou celkem 3 signály a jejich negace. Tento obvod tedy pohodlně zajistí prvotní zpracování všech signálů z IRC čidla.

Obvod AM26LS32 jsem vybral především díky jeho použití v doporučeném zapojení udávaným výrobcem IRC čidla (ESSA Praha). Alternativou k tomuto obvodu je např. obvod MAX3096. Doporučené zapojení, které uvádí výrobce IRC čidla je na obr.6. Toto zapojení jsem také použil v této aplikaci. Schéma zapojení elektronické části je v příloze 1.



obrázek 6 – schéma zapojení výstupních obvodů RS 422

3.2.2 Obvod GAL22V10

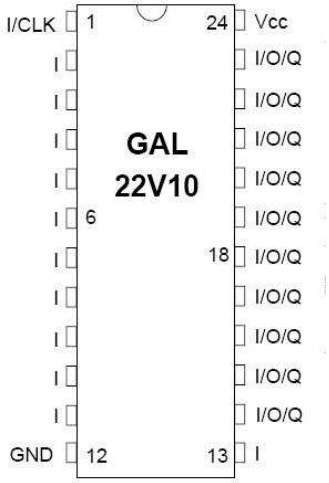
GAL22V10 patří mezi součástky typu PLD (Programmable Logic Device). Je to reprogramovatelné hradlové pole, umožňující provádět mnoho logických operací v rámci jedné relativně malé součástky (v porovnání s objemem součástek, které by bylo nutné použít, kdyby GAL nebyl k dispozici). Tato součástka nahrazuje složité propojování logických obvodů.

V této aplikaci tato součástka slouží k dalšímu zpracování IRC signálů po průchodu obvodem AM26LS32. Začlenění této součástky do regulačního systému je patrné v příloze 1. V tabulce 3 jsou nejdůležitější hodnoty potřebné pro návrh. Z tabulky 3 je vidět, že maximální zpoždění výstupu obvodu GAL22V10 po změně na vstupu je 7,5 ns. Jednoduchým výpočtem získáme, že se může vstupní signál v krajním případě měnit s frekvencí maximálně **130 MHz**.

Maxima (vyšší hodnoty vedou k poškození zařízení)	
Napájecí napětí Vcc	-0,5 až + 7 V
Napětí na vstupních konektorech	-2,5 až Vcc + 1V
Doporučené	
Napájecí napětí Vcc	+4,75 až 5,25 V
Důležitá hodnota	
zpoždění výstupu za vstupem	max. 7,5 ns

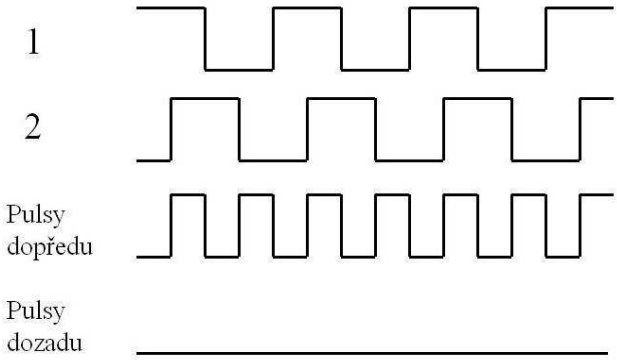
tabulka 3 –důležité hodnoty obvodu GAL22V10

Na obr.7 je znázorněno rozložení konektorů obvodu GAL22V10. ‚I' konektory jsou použitelné pouze pro vstup, ‚I/O/Q' konektory jsou vstupně-výstupní nebo stavové.

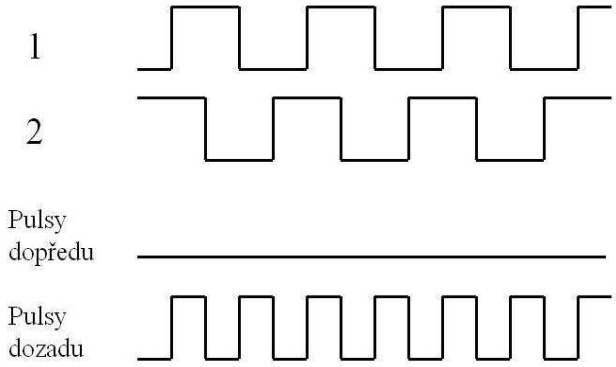


obrázek 7 - rozložení konektorů obvodu GAL22V10

Na dva vstupní konektory je přiveden výstupní signál řetězců obvodu AM26LS32, které zpracovávají signály 1, 1 NON, 2, 2 NON (viz. tab.2 a příloha 1). Vnitřní program obvodu je navržen tak, aby detekoval směr otáčení a generoval signál na jednom z výstupních konektorů „pin 23" nebo „pin 22" v závislosti na směru otáčení IRC čidla. Obr.8 a obr.9 ukazují průběhy vstupních a výstupních signálů před a po zpracování obvodem GAL22V10. Signály „1" a „2" jsou vstupní a „Pulsy dopředu/dozadu" jsou signály výstupní.



obrázek 8 – signály obvodu GAL22V10 při otáčení IRC „vpřed"



obrázek 9 - signály obvodu GAL22V10 při otáčení IRC „vpřed“

Stavové schéma návrhu programu pro obvod GAL22V10 je v příloze 3A. Program pro GAL22V10 je implementován v programu ORCAD_GAL. Zdrojový kód je v příloze 3B. Tímto programem GAL plní funkci popsanou na obr.8 a obr.9 a v příloze 3A. GAL22V10 tímto programem pracuje v asynchronním režimu, tzn. že reaguje teprve až po změně na vstupu.

Ze vstupních signálů (viz. obr.8 a obr.9), které jsou od sebe posunuty o čtvrt periody, je vidět, že se vstup GALu bude měnit s frekvencí **čtyřikrát** vyšší, než je frekvence signálu z IRC čidla. Vezmeme-li tedy v úvahu maximální možnou frekvenci změn vstupu obvodu GAL

f_{max} = 130 MHz;

a počet period IRC signálu na jednu otočku IRC čidla (viz. tab.1)

n = 2500 $^{period}/_{ot.}$

získáme maximální rychlost otáčení IRC čidla, kterou je obvod GAL22V10 schopen zpracovat:

$$\omega_{\max} = \frac{130 \cdot 10^6}{4 \cdot 2500}\left[\frac{ot.}{sek}\right] = 13000\left[\frac{ot.}{sek}\right]$$

Pohlédneme-li opět do tab.1, poznáme, že frekvence výstupních signálů IRC čidla jsou max. 300 kHz, to znamená, že maximální rychlost otáčení IRC čidla, která může být bezpečně zachycena elektronikou tohoto čidla, je 120 $^{ot.}/_{sek.}$ Tím je ukázáno, že by měl obvod GAL22V10 pohodlně zpracovat signály při maximální rychlosti otáčení IRC čidla.

Maximální frekvence výstupního signálu GAL22V10 je (podle obr.8, obr.9, přílohy 3A a 3B) 2-krát větší než maximální výstupní frekvence IRC čidla, tedy

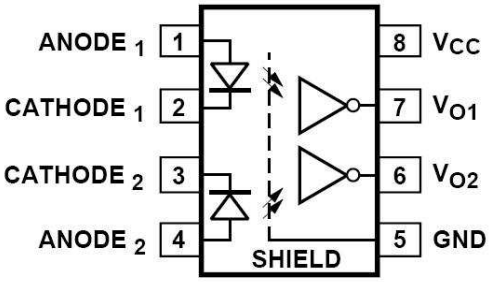
$$f_{GAL\,\max} = 2 \cdot 300kHz = 600kHz$$

3.3 Obvody pro zpracování PWM signálu

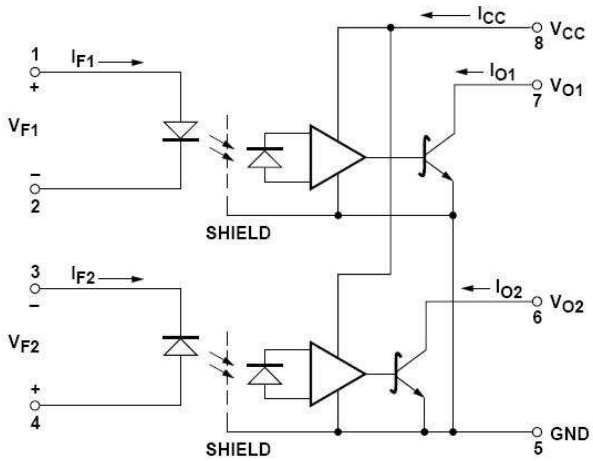
Obecné blokové schéma je znázorněno na obr.3. Následující řádky budou věnovány podrobnějšímu popisu optického vazebního členu HCPL2630, ochrannému obvodu L6506 a H-můstku L6203.

3.3.1 Obvod HCPL2630

Jedná se o optický vazební člen (dále jen optočlen). Funkce tohoto obvodu je naznačena na obr.10 a obr.11.



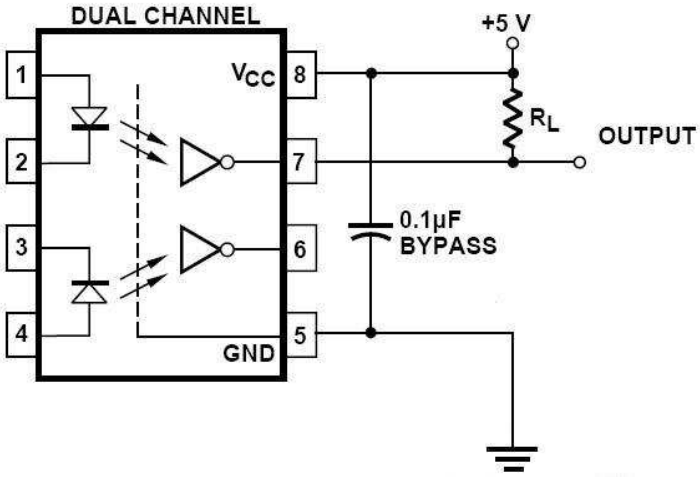
obrázek 10 – zapojení konektorů obvodu HCPL2630



obrázek 11 – vnitřní schéma obvodu HCPL2630

Na vstupu obvodu je fotodioda emitující záření, které je zachycováno fotodetektorem. Zjednodušeně řečeno, intenzita záření fotodiody závisí na rozdílu potenciálů na vstupu diody a rozdíl potenciálů na výstupu fotodetektoru proti zemi je závislý na intenzitě záření fotodiody. V opačném směru není možné, aby prošel nějaký signál a s ním také energie. Proto je tato součástka umístěna mezi elektronickou a výkonovou část, jako ochrana elektronické části před energetickými výkyvy, které mohou vznikat ve výkonové části například ve vinutí motoru.

Signál přivedený na vstup je převeden na výstup ve stejném tvaru, ne však se stejnou amplitudou. K tomu slouží tzv. „zvedací“ (pull-up) rezistor $\mathbf{R_L}$ připojený na výstup proti napájení podle obr.12. Tímto způsobem je „zvednuta“ amplituda výstupního signálu na napájecí napětí optočlenu, tj. +5V.



obrázek 12 – zapojení HCPL2630 v aplikaci

V našem případě HCPL2630 propouští signály PWM a signál ENABLE umožňující nouzové odpojení motorů (viz. příloha 1). V případě velkého výkyvu napětí směrem od výkonové části bude toto v lepším případě „zachyceno“ a nepropuštěno směrem k elektronice. V horším případě dojde ke zničení optočlenu, který pak přestane propouštět PWM signály a dojde k zastavení výkonové části, resp. motoru. Některé důležité hodnoty optočlenu HCPL2630 jsou uvedeny v tab.4.

napájecí napětí Vcc	max. 7 V
napětí na vstupu	max. Vcc + 0.5V
proud vstupem	max. 5mA
mezní závěrné napětí diody na vstupu	max. 5V
odpor izolace	typ. 10^{12} Ω
trvalé napětí na izolaci	max. 630V
výstupní pull-up rezistor R_L	330 až 4kΩ

tabulka 4 – důležité hodnoty optočlenu HCPL2630

*Poznámka: HCPL2630 procházející signál **invertuje**. S tímto faktem se musí počítat při dalším návrhu.*

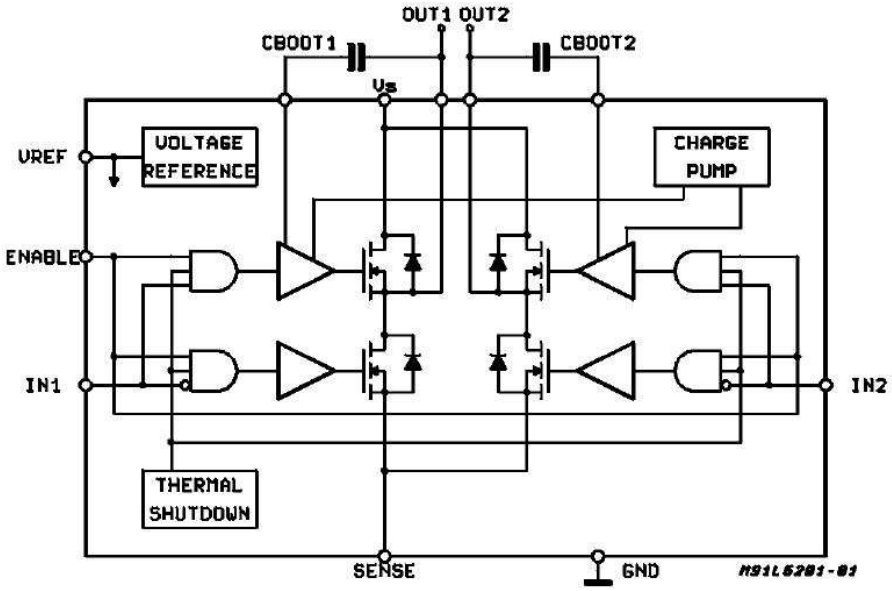
3.3.2 Obvod L6203

Jedná se o H-můstek, který slouží k ovládání napětí přiváděného na řízený motor v závislosti na vstupním PWM (pulsní šířková modulace) signálu. Vnitřní schéma obvodu je na obr.13. Na vstupy IN1 a IN2 jsou přivedeny ovládací PWM signály, které jsou zpracovány vnitřní logikou obvodu. Podle těchto signálů jsou „otevírány a zavírány“ vnitřní tranzistory a tím je měněna orientace napětí mezi výstupy OUT1 a OUT2. Při připojené zátěži na výstupní konektory je tedy také měněn směr proudu protékající zátěží, tedy po ose

V_s → OUT1 (resp. OUT2) → zátěž → OUT2 (resp. OUT1) → SENSE

Tím je měněn i směr sil působících na zátěž, v našem případě motor.

Konektor SENSE uzavírá proudovou smyčku a měl by být připojen na zem, proti které je připojeno napájecí napětí V_s. Signál na konektoru ENABLE umožňuje nouzově odpojit obvod. V mém návrhu je tento konektor připojen konstantně na log.1 (viz. příloha 2) a zpracování signálu ENABLE zajišťuje obvod L6506 (viz. níže).



obrázek 13 – vnitřní schéma obvodu L6203

Důležité hodnoty obvodu L6203 jsou uvedeny v tab.5.

napájecí napětí Vs	max. 52 V
napětí mezi kon. OUT1 a OUT2	max. 60 V
napětí na vstupech IN nebo ENABLE	-0.3 až 7 V
opakovatelný proud výkonovou částí	max. 5 A

tabulka 5 – důležité hodnoty obvodu L6203

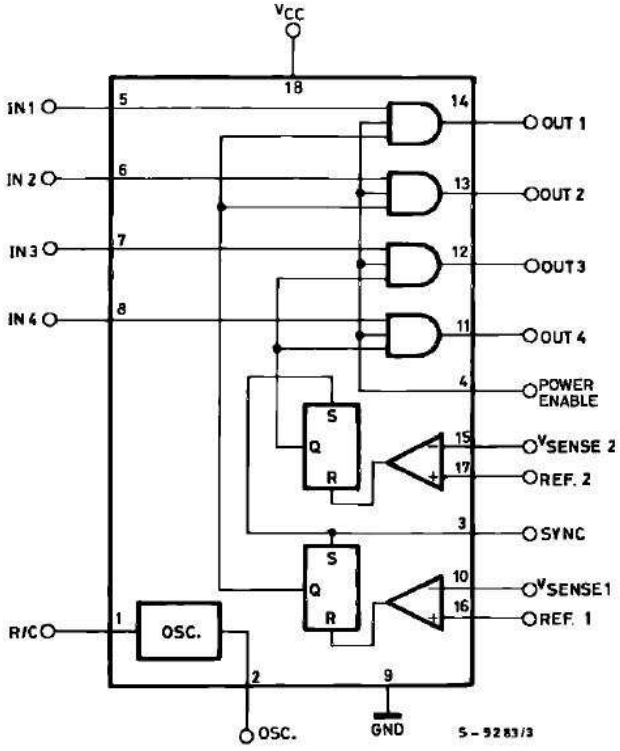
3.3.3 Obvod L6506

Obvod L6506 je doporučeným obvodem výrobce obvodu L6203 (SGS-Thomson Microelectronics) jako ochrana výkonové části před přetížením. Vnitřní schéma obvodu je na obr.14. Obvod propouští PWM signál za splnění daných podmínek. První podmínkou je log.1 na vstupu ENABLE. Druhou podmínkou je nižší napětí na vstupu V_{sense} než na příslušném vstupu V_{ref}. Obvod je zároveň schopen generovat (při správném zapojení příslušných konektorů) hodinový signál.

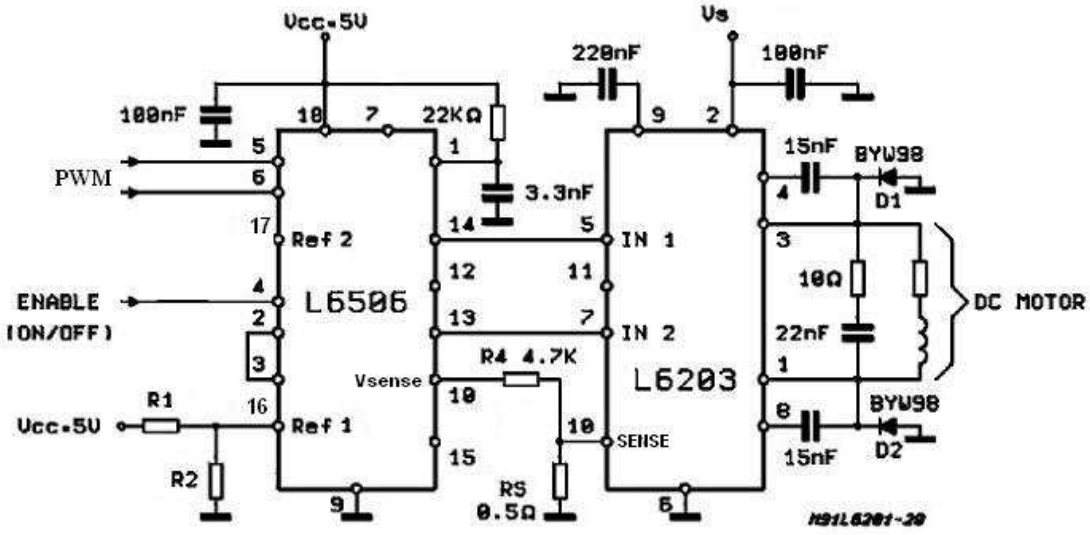
Od obvodu L6203 je možné vytvořit zpětnou vazbu kontrolující velikost proudu protékající tímto obvodem tak, že mezi konektor SENSE, který by měl uzavírat proudovou smyčku, a zem připojíme odpor o relativně malé hodnotě, v našem případě R_s = 0.5 Ω. Proudová smyčka je pak

$V_s \rightarrow$ OUT1 (resp. OUT2) $\rightarrow$ zátěž $\rightarrow$ OUT2 (resp. OUT1) $\rightarrow$ SENSE $\rightarrow R_s \rightarrow$ GND

Protéká-li odporem R_s proud, vzniká na něm napětí. Toto napětí je možné využít ke kontrole proudu protékajícího obvodem L6203 a tedy i zátěží. Na obr.15 je ukázáno vzájemné propojení obvodů L6506 a L6203.



obrázek 14 – vnitřní schéma obvodu L6506



obrázek 15 – zapojení obvodů L6506 a L6203 v této aplikaci

Nyní bych se rád dále věnoval zapojení na obr.15. Jak již bylo řečeno, na odporu R_s vzniká napětí. Toto napětí je přivedeno na konektor V_{sense} obvodu L6506 přes odpor R_4 = 4.7kΩ. To je mnohonásobně více, než R_s = 0.5 Ω. Odpor R_4 má zajistit, že měřený proud půjde cestou nejmenšího odporu, tj. přes odpor R_s a nemůže poškodit obvod L6506. Signál V_{sense} je přiveden na negativní vstup vnitřního komparátoru, na jehož výstupu je log.1 při překročení referenčního napětí. Tato log.1 resetuje vnitřní RS klopný obvod a tím odpojí výstup obvodu L6506 a PWM signál nebude propuštěn (viz. obr.14 a obr.15). Na pozitivní vstup komparátoru je přivedeno zvolené referenční napětí. Např. pro max. proud obvodem L6203 I_{max} = 5A je V_{ref} = 2.5 V.

Další zajímavostí tohoto zapojení je připojení RC oscilátoru na vstup vnitřních obvodů pro tvorbu hodinového signálu. Hodnoty R a C jsou:

R = 22 kΩ
C = 3.3 nF

Frekvence vzniklého oscilačního signálu je dána vztahem

$$f_{osc} = \frac{1}{0{,}69 \cdot RC} \quad pro\ R \geq 10k\Omega$$

$$f_{osc} \cong 20kHz$$

Tento signál je příslušnými vnitřními obvody převeden na hodinový signál na konektoru 2. Konektor 2 je propojen se SET (konektor 3) vstupem vnitřního RS klopného obvodu. Pokud dojde k překročení maximálního proudu obvodem L6203, dojde také k trvalému odpojení výstupu obvodu L6506, který je možné opět připojit buď manuálně nebo celkovým restartem. V době odpojení klesne proud obvodem L6203. Připojením hodinového signálu na konektor 3 je docíleno automatického připojování výstupu L6506 v intervalech daných frekvencí hodinového signálu. Pokud je po připojení výstupu L6506 zjištěn nadměrný proud obvodem L6203, dojde opět k odpojení výstupu L6506 a jeho dalšímu připojení dalším pulsem hodinového signálu, atd.

3.4 Mikroprocesor ATmega8515

Na mikroprocesor v této aplikaci jsou kladeny požadavky z tab.6

vlastnost	poznámka
2x externí přerušení	čítání pulsů IRC signálu
1x časovač	nastavení periody vzorkování
1x generátor obdélníkového signálu	generace PWM signálu
rychlost >> 600kHz	600kHz je max. frekvence příchozího IRC signálu po zpracování příslušnou elektronikou

tabulka 6 – nároky na mikroprocesor

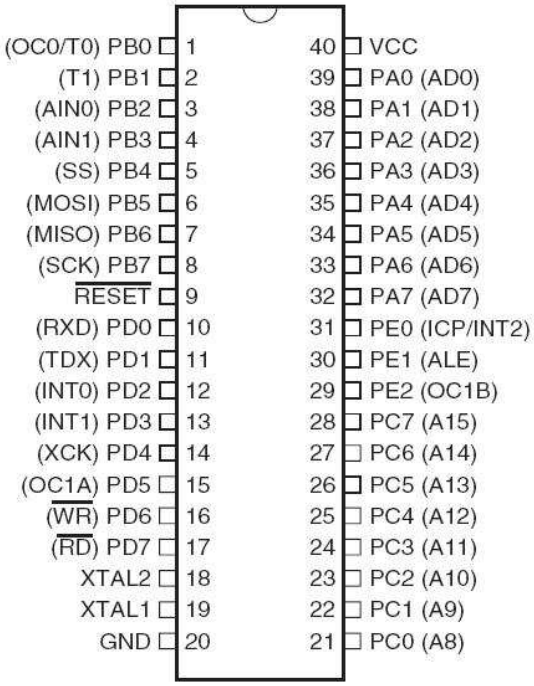
Tyto požadavky splňuje např. mikroprocesor ATmega8515, jehož vlastnosti jsou v tab.7 který jsem vybral pro tuto aplikaci.

vlastnost	poznámka
8-bitový mikroprocesor	
8 kB flash paměti	automaticky programovatelná
2x čítač/časovač	1x 8-bit a 1x 16-bit
3 PWM kanály	
3x externí přerušení	
rychlost 0 až 16 MHz	
operační napájení 4.5 až 5.5 V	

tabulka 7 – vlastnosti ATmega8515

Rozložení konektorů mikroprocesoru ATmega8515 je na obr.16. Názvy konektorů bez závorek označují implicitní funkci konektoru, zatímco názvy v závorkách označují alternativní funkci, která může být aktivována nastavením příslušného registru mikroprocesoru. Například konektor PD2 je implicitně třetím bitem vstupně-výstupního 8-bitového portu PortD, avšak správným nastavením registrů GICR (General Interrupt Control

Register), MCUCR (MCU Control Register) a SREG (Status REGister) je možné tento konektor uvést do stavu vyvolání externího přerušení INT0 při příchodu potřebného signálu (také nastavitelné). Této problematice se budu věnovat dále v kapitole „**Programování mikroprocesoru ATmega8515**“.



obrázek 16 – rozložení konektorů mikroprocesoru ATmega8515

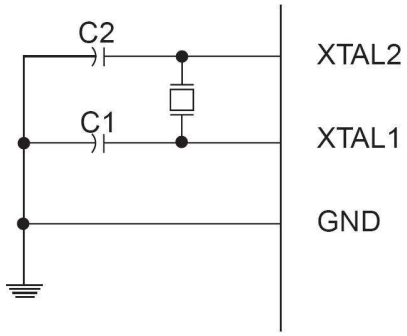
Zapojení mikroprocesoru do systému je nejlépe vidět v příloze 1. Využívané konektory mikroprocesoru ATmega8515 jsou uvedeny v tab.8.

konektor	funkce
INT 0	spuštění obslužné rutiny pro zpracování signálu "Pulsy_dopředu" výstupu obvodu GAL22V10
INT 1	spuštění obslužné rutiny pro zpracování signálu "Pulsy_dozadu" výstupu obvodu GAL22V10
OC1A	výstupní konektor signálu PWM1 generovaného procesorem
OC1B	výstupní konektor signálu PWM2 generovaného procesorem
PE1	výstupní konektor signálu ENABLE pro výkonovou elektroniku
XTAL1 a XTAL1	připojení krystalu udávajícího rychlost vnitřních "hodin" procesoru
TXD	výstupní konektor pro sériovou komunikaci. Zapojeno pouze při tvorbě a testování programového kódu

tabulka 8 – využívané konektory mikroprocesoru ATmega8515

3.5 Krystal

Krystal jsem zvolil takový, abych plně využil rychlost mikroprocesoru ATmega8515, tedy **16 MHz** (viz. tab.7). Krystal má standardní pouzdro HC49/U-S. Zapojení krystalu je na obr.17.



obrázek 17 – zapojení krystalu

Doporučené hodnoty kondenzátorů jsou v tab.9. V praxi jsem použil kondenzátory o hodnotě

$C_1 = C_2 = 18\,pF$.

Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
0.4 - 0.9	–
0.9 - 3.0	12 - 22
3.0 - 8.0	12 - 22
1.0 ≤	12 - 22

tabulka 9 – doporučené hodnoty kondenzátorů pro zapojení krystalu mikroprocesoru ATmega8515

3.6 Motor

Byl použit dostupný zkušební motorek švýcarské výroby firmy MAXON. Sériové číslo bylo bohužel nečitelné a tak byl motor považován za „neznámý“ (nejsou známy žádné elektrické ani mechanické hodnoty). Motorek byl pro jistotu připojen na relativně nízké maximální napětí **12V.** Experimentální identifikace tohoto neznámého motoru je v kapitole 5.1.

4 Programování mikroprocesoru ATmega8515

Jedním z důvodů výběru mikroprocesoru ATmega8515 byla jeho podpora Windows kompatibilním vývojovým programem AVR studio. Tento program je dodáván výrobcem ATmega8515, firmou Atmel, jako softwarová podpora všech jimi vyráběných mikroprocesorů s jádrem AVR. Jako programátor, umožňující naprogramování mikroprocesoru, byl použit starter kit STK500, podporující všechny vyráběné DIP (double in-line package) pouzdra Atmel AVR mikroprocesorů. Program je možné v AVR studiu psát buď v Assembler nebo C kódu. Já si vybral C kód především pro svoji větší zkušenost v jeho používání.

4.1 STK500

STK500 je kompletní souprava, která slouží k seznámení se s AVR mikroprocesory. Zároveň však může být použita k vývoji a testování složitějších elektronických struktur využívajících mikroprocesory AVR a k programování těchto mikroprocesorů po sériové lince. STK500 je plně podporován programem AVR studio. Velkou výhodou použití STK500 je vyvedení všech konektorů mikroprocesoru na signálové kolíky, což umožnilo připojení mikroprocesoru do elektronického systému složenému na nepájivém poli, aniž bych musel vyjmout mikroprocesor z vývojové soupravy. Pro utvoření si základní představy jsem umístil schématický obrázek STK500 do přílohy 4.

4.2 AVR Studio

AVR studio je integrované vývojové prostředí pro psaní a ladění AVR aplikací v prostředích Windows 9x/Me/NT/2000/XP. Poskytuje nástroj projektového manažera, editor zdrojových souborů, chip-simulátor a obvodový emulátor pro AVR 8-bitové mikroprocesory. Zároveň podporuje vývojovou desku STK500, která dovoluje programování všech AVR zařízení.

4.3 WinAVR

WinAVR je souprava open source vývojových nástrojů pro Atmel AVR mikroprocesory pracujících na platformě Windows. Obsahuje GNU GCC překladač pro C a C++ a důležité hlavičkové soubory pro práci s AVR mikroprocesory. WinAVR musí být nainstalován ve Windows předtím, než je spuštěna jakákoliv kompilace programu v AVR studiu, jinak je oznámena chyba a zobrazena výzva k nainstalování WinAVR.

4.4 Fúzní bity

Fúzní bity slouží podobně jako registry k nastavení vlastností AVR mikroprocesoru. Na rozdíl od registrů je nelze měnit v průběhu programu. V našem případě je nutné nastavit fúzní bity určující frekvenční oblast ve které by měl mikroprocesor pracovat a typ zdroje oscilačního signálu (viz. tab.10, 11 a 12).

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

tabulka 10 – fúzní bity pro volbu typu oscilátoru

CKOPT	CKSEL3..1	Frequency Range (MHz)
1	101(1)	0.4 - 0.9
1	110	0.9 - 3.0
1	111	3.0 - 8.0
0	101, 110, 111	1.0 ≤

tabulka 11 - fúzní bity pro volbu módu krystalového oscilátoru

CKSEL0	SUT1..0	Start-up Time from Power-down	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	–	Ceramic resonator, BOD enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

tabulka 12 - fúzní bity pro volbu startovací doby krystalového oscilátoru

Při požadavku využití maximální rychlosti mikroprocesoru (**16MHz**), možnosti použití **krystalu** o frekvenci 16MHz a požadavku bezpečného „nastartování“ krystalu po zapnutí či restartu i za cenu delší doby startu systému, jsem se rozhodl nastavit fúzní bity podle tab.13.

CKOPT	0
CKSEL3..1	111
CKSEL0	1
SUT1..0	11

tabulka 13 – frekvenční nastavení ATmega8515

4.5 Definice typu mikroprocesoru a přístup k I/O registrům

Zavedením hlavičkového souboru

```
#include <avr\io.h>
```

je zpřístupněn základní registr SREG (Status REGister) společný všem AVR mikroprocesorům. Nadefinujeme-li však předtím konstantu __AVR_ATmega8515__, tedy

```
#define __AVR_ATmega8515__
#include <avr\io.h>
```

bude zároveň zahrnut i hlavičkový soubor avr\iom8515.h, který definuje I/O registry a příslušné bity těchto registrů pro konkrétní mikroprocesor ATmega8515. Použité názvy registrů a jejich bitů se shodují s názvy uváděnými v manuálových listech daného mikroprocesoru.

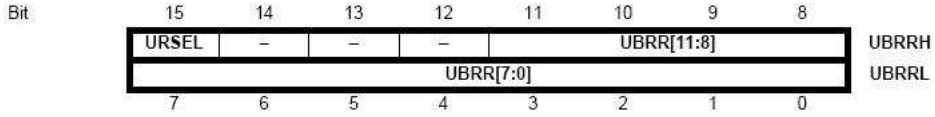
4.6 Nastavení sériové komunikace USART

Zapojení sériové komunikace USART (Universal Synchronous and Asynchronous Receiver and Transmitter) bylo nezbytné před dalším pokračování ve vývoji programu. Prostřednictvím USART bylo možné posílat do PC po sériovém kabelu informace o výpočetním procesu. To značně usnadnilo tvorbu programu, hledání případných chyb a jejich odstraňování.
Nejprve jsem si definoval baudovou rychlost podle tab.14:

```
#define BAUD 103                    //rychlost USART 9600 baudu
```

Tato hodnota je dosazena do UBRR registru (USART Baud Rate Registers), který je rozdělen na UBRRH a UBRRL register podle obr.18, následovně:

```
/* Set baud rate */
UBRRH = (unsigned char)(BAUD>>8);
UBRRL = (unsigned char)BAUD;
```



obrázek 18 – UBRR registr

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. (1)	1 Mbps		2 Mbps	

tabulka 14 – nastavení baudové rychlosti USART ATmega8515 při f_{osc} = 16MHz

Poznámka: Tab.14 také upozorňuje na chybu 0.2% v přenosu při stávajícím použitém krystalu (16MHz). USART komunikace je však použita pouze pro ladění a občasná chyba tedy byla akceptovatelná. Důležitější pro mě bylo vidět chování systému za podmínek,při kterých by měla pracovat konečná verze systému.

Registrem UCSRB (obr.19) je možné nastavit funkci konektoru PD0 jako přijímače USART (RXD) a konektoru PD1 jako vysílače USART (TDX). Registrem UCSRC (obr.20) je nastaven počet data-bitů pomocí bitů UCSZ2..0 (tab. 15) a počet stopbitů pomocí bitu USBS. Není-li nastaven bit UMSEL, pracuje USART v asynchronním režimu.

```
 /* Enable receiver and transmitter */
   UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 2stop bit */
   UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
```

Parita je tímto způsobem vypnuta.

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

obrázek 19 – UCSRB registr

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

obrázek 20 – UCSRC registr

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

tabulka 15 – nastavení bitů UCSZ2..0 registru UCSRC

Dalším důležitým registrem je UDR registr (USART Data Register), ve kterém je uložena aktuálně odesílaná zpráva. To, jestli je UDR registr prázdný, udává UDRE (UDR Empty) bit registru UCSRA. Před samotným odesláním 8-bitové zprávy se dvěma stop-bity je třeba počkat na vyprázdnění UDR registru:

```
while ( !( UCSRA & (1<<UDRE)) );
```

Pak je možné poslat další zprávu opětovným naplněním UDR registru:

```
UDR = data;
```

Poslední dva příkazy jsou prováděny funkcí

void USART_Transmit1(unsigned int data),

jak je také možné vidět v příl.5 / kód 6. Ta je využita ve funkci (viz. příl.5 / kód **7**)

void USART_Transmit2(unsigned int data)

Zde je vstupní číslo rozděleno na maximálně 5 cifer, které jsou postupně za sebou odeslány funkcí USART_Transmit1 v ASCII tvaru, takže je pak na konzoli zobrazena pěticiferná číslice. Zpráva je nakonec odesílána ve tvaru proměnné a k ní příslušné hodnoty pomocí funkce (viz. příl.5 / kód 4)

void tiskni(int_fast32_t cas, int_fast32_t hodnota)

4.7 Externí přerušení

Externí přerušení jsou využívána pro čítání pulsů signálů od obvodu GAL22V10. Na vstup externího přerušení INT0 je přiveden signál „pulsy dopředu“ a na vstup externího přerušení INT1 signál „pulsy dozadu“.

Nejprve je ale opět nutné nastavit příslušné registry. Nastavením I-bitu registru SREG (Status REGister) jsou povolena přerušení. Nastavením INT0 a INT1 bitu registru GICR (General Interrupt Control Register) jsou konektory externího přerušení INT0 a INT1. Bity ISC00..1 a ISC10..1 registru MCUCR (MCU Control Register) je nastaven mód ve kterém externí přerušení pracují podle tab.16.

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

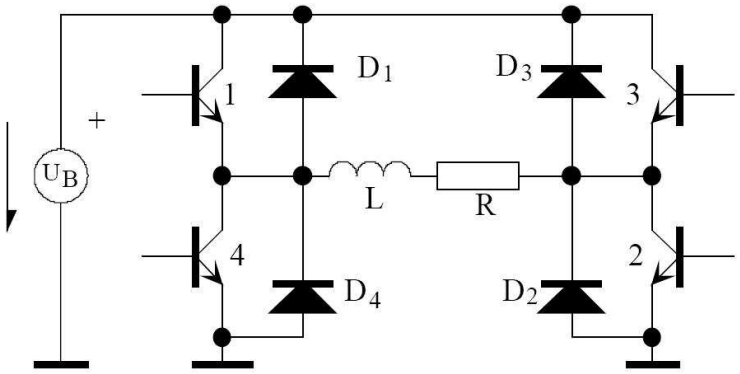
tabulka 16 – nastavení módu externího přerušení INT1

Poznámka: mód externího přerušení INT0 se nastaví obdobně pomocí bitů ISC00..1

Obě použitá externí přerušení jsou nastavena do módu detekce náběžné hrany, resp. při detekci náběžné hrany na konektoru externího přerušení je toto přerušení vyvoláno a spuštěna obslužná rutina. Obslužná rutina INT0 je v příl.5 / kód 1 a rutina pro INT1 je v příl.5 / kód 2. Jedná se prakticky „pouze“ o inkrement globální proměnné krokA o jedničku při přerušení INT0, resp. její dekrement o jedničku při přerušení INT1.

4.8 Tvorba PWM signálu

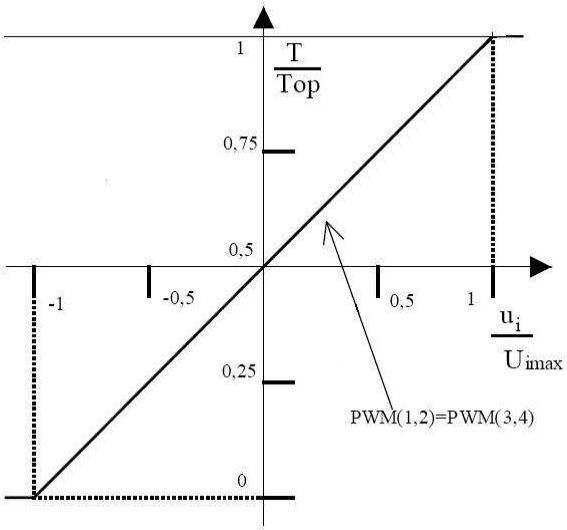
Nejprve je nutné uvědomit si, jaký typ PWM signálu bude potřeba. Rozhodoval jsem se mezi unipolárním a bipolárním čtyřkvadrantovým řízením. S podrobným popisem PWM modulace se odkazuji na [1]. Uvažované zapojení H-můstku je na obr.21. Je vidět, že se shoduje s vnitřní částí obvodu L6203 na obr.13.



obrázek 21 – zapojení pro realizaci PWM řízení (H-můstek)

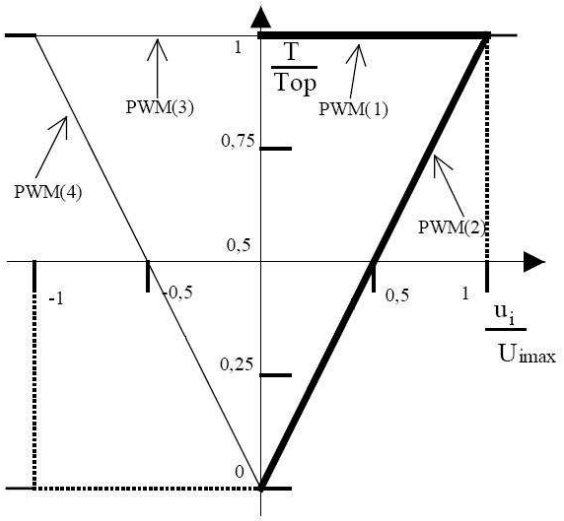
Průběh bipolární PWM modulace je na obr.22. Tato modulace je výhodná při řízení otáček, protože je dodávaná energie převedena na rotační pohyb. Pro řízení polohy však nevyhovuje, protože v klidové poloze je sice střední hodnota napětí na zátěži nulová, ale efektivní hodnota je nenulová. Motor se sice neotáčí, ale protéká jím nezanedbatelný proud,

který se mění na Jouleovo teplo. Motor se pak zahřívá a pokud bychom nedostatečně chladili, mohl by se zničit.



obrázek 22 – graf bipolární PWM modulace

Na obr.23 je průběh unipolární modulace. Ta zajišťuje nulový proud zátěží při klidovém stavu (v našem případě - motor se neotáčí) absolutním odpojením zátěže od napájení. Nevýhodou je, že v tomto stavu při pouhém PWM řízení je možno libovolně otáčet hřídelí motoru (je větší náchylnost na okolní vlivy a hřídel motoru se snáze vychýlí z požadované polohy). Tento problém je ale vyřešen zapojením regulátoru ve zpětné vazbě, který bude hřídel motoru udržovat v pevné pozici. Protože se budu ve větší míře zabývat polohovou zpětnou vazbou, bude další postup tedy směřovat k návrhu unipolární PWM modulace.



obrázek 23 - graf bipolární PWM modulace

PWM modulační signál je možné v ATmega8515 zavést nastavením jednoho z časovačů jako generátoru obdélníkového signálu. Rozhodl jsem se k tomuto účelu nastavit 16-bitový časovač/čítač, protože umožňuje současné generování dvou PWM signálů.

K nastavení slouží registr TCCR1A (Timer/Counter1 Control Register A, obr.24) a registr TCCR1B (obr.25).

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

obrázek 24 – TCCR1A registr

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

obrázek 25 – TCCR1B registr

Nastavení typu PWM je provedeno podle tab.17 na Fázově a frekvenčně korektní PWM signál (vysvětlení viz.[3]), kdy horní hodnota čítače (při které se mění směr čítání na dekrementaci) je dána v ICR1 registru a spodní hodnota (při které se mění směr čítání na inkrementaci) je 0. Délka pulsu PWM je pak nastavována přímo v programu zapsáním hodnoty do registru OCR1A (Output Compare Register1 A), resp. OCR1B podle toho, který ze dvou poskytovaných PWM kanálů chceme změnit. Pokud se aktuální hodnota čítače/časovače ukládaná do registru TCNT1(Timer/Counter1 register) rovná hodnotě v registru OCR1A (OCR1B), dojde ke změně logické úrovně na příslušném konektoru. Nastavením bitu CS10 spustíme čítač/časovač, kdy frekvence, se kterou se inkrementuje (resp. dekrementuje) TCNT1 registr je rovna frekvenci oscilátoru použitého pro obvod ATmega8515.

```
TCCR1B = (1<<WGM13)|(1<<CS10);
ICR1 = 100;
```

Při tomto nastavení registrů bude frekvence PWM signálu rovna

$$f_{PWM} = \frac{f_{CPU}}{2 \cdot ICR1} = \frac{16MHz}{2 \cdot 100} = 80kHz$$

Vzhledem k tomu, že rozsah slyšitelných frekvencí je 20Hz – 16kHz, „pískání“ motoru spojené právě s PWM řízením bude odstraněno.

Nyní je ještě nutné nastavit výstupní PWM konektory OC1A a OC1B. Nejprve musí být nastaveny jako výstupní v příslušném DDR registru (Data Direction Register). Dále je podle tab.18 nastavena jijich funkce a mód.

```
DDRD = (1<<PD5);            //nastaveni PINu PD5 (resp. OC1A) jako vystup
DDRE = (1<<PE2);            //nastaveni PINu PE2 (resp. OC1B) jako vystup
TCCR1A = (1<<COM1A1)|(1<<COM1B1);
```

OC1A a OC1B jsou takto nastaveny na přechod do log.0 při shodě TCNT1 a OCR1A(B) registru při inkrementaci TCNT1 registru a na přechod do log.0 při shodě TCNT1 a OCR1A(B) registru při dekrementaci TCNT1 registru.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

tabulka 17 – nastavení módu, ve kterém bude Timer/Counter1 pracovat

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 14: Toggle OC1A on Compare Match, OC1B disconnected (Normal port operation). For all other WGM1 setting, Normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when downcounting.
1	1	Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when downcounting.

tabulka 18 - nastavení výstupních PWM konektorů

Funkce **int pwm (int_fast32_t voltage),** která nastavuje PWM signál je v příl.5 / kód 5. Vstupem je požadované napětí v rozsahu $-V_{max}$ a $+V_{max}$. Pokud je vstup roven nule, pak jsou odpojeny oba PWM kanály (motorem neteče proud), jinak je z absolutní hodnoty vstupu vypočítána délka pulsu PWM a nastavena pro oba kanály stejně (OCR1A a OCR1B registr). Nakonec je podle znaménka vstupu rozhodnuto, který PWM kanál bude připojen a tím určen i směr otáčení motoru.

4.9 Vnitřní časovač

Abychom byli vůbec schopni zrealizovat nějaké zpětnovazební řízení, je třeba odečítat hodnoty senzoru a provádět výpočet v určitých známých časových intervalech. K tomu nám poslouží zbývající 8-bitový časovač/čítač (Timer/Counter0). Lze jej nastavit tak, aby při shodě aktuální hodnoty v registru TCNT0 (Timer/Counter0 register) s OCR0 (Output Compare Register0) vyvolal přerušení, v jehož obslužné rutině je možné provést celý potřebný výpočet. Nastavením WGM01 bitu registru TCCR0 (Timer/Counter0 Control Register) uvádím čítač/časovač do módu čítání do hodnoty uložené v OCR0 registru (CTC mód – Clear Timer on Compare match). Bity CS02 a CS00 téhož registru zavádějí inkrementaci hodnoty v TCNT0 registru o frekvenci 1024-krát menší než je vnitřní frekvence mikroprocesoru daná připojeným krystalem (16 MHz). Bit OCIE0 registru TIMSK (Timer Interrupt Mask) společně s I-bitem SREG (Status registru), umožňuje vyvolání přerušení při rovnosti registrů TCNT0 a OCR0.

```
TCCR0 = (1<<WGM01)|(1<<CS02)|(1<<CS00);   //nastaveni casovace, 1puls/1024(pulsu/O)
TIMSK = (1<<OCIE0);                       //preruseni pri shode registru TCNT0 s OCR0
OCR0 =  250;                              //hodnota, se kterou budeme porovnavat
```

Tímto nastavením získávám délku periody vzorkování výpočetního procesu

$$t_{vz} = \frac{OCR0 \cdot 1024}{f_{cpu}} = \frac{250 \cdot 1024}{16MHz} = 16ms$$

Obslužná rutina přerušení od časovače Timer/Counter0, ve které je realizováno vzorkování údaje z IRC čidla a výpočet regulace, je tedy prováděno v intervalech 16ms. Tato rutina je v příl.5 / kód 3.

Nejprve je uložena aktuální hodnota času a údaje z čidla pro kontrolní výpis. Pak jsou spočteny hodnoty regulace a akční zásah, který je omezen maximální hodnotou použitelného napětí. Zásah je pak zpracován funkcí **pwm** na reálný PWM signál.

4.10 Volba regulátoru

Jak již jsem se zmiňoval, práce se soustředí v první řadě na regulaci polohy (úhlu natočení) motoru. Pro tento systém jsem se rozhodl vyzkoušet, porovnat a zvolit lepší z regulátorů typu **P** a **P s rozdílovou složkou** (rozdíl hodnot ve dvou po sobě jdoucích krocích). Samotný regulátor tedy obsahuje pouze proporcionální nebo proporcionální a rozdílovou složku, které zajišťují rychlost regulace, ne však nulovou regulační odchylku. Rozdílovou složku jsem se rozhodl vyzkoušet, protože by mohla urychlit přechodovou část regulace. Pro zajištění nulové regulační odchylky musí být v systému přítomna integrační složka, v některých případech (jako je právě tento), nemusí být integrační složka v případě malé **konstantní** zátěže přímo součástí regulátoru. Pohledem na obr.28 je vidět, že náš systém obsahuje integrátor, díky němuž je získán úhel natočení motoru ze samotného výstupu motoru – úhlové rychlosti. Díky tomu je možné teoreticky dosáhnout nulové regulační odchylky bez přítomnosti integrační složky v samotném regulátoru. Daný systém musí být ale pro návrh kvalitně identifikován a jeho vlastnosti musí být v čase neměnné. I malý rozdíl od ideální identifikace pak způsobí nenulovou regulační odchylku. Tento problém je možné vyřešit právě zařazením integrační složky do regulátoru. Tento fakt je také v této práci později ukázán.

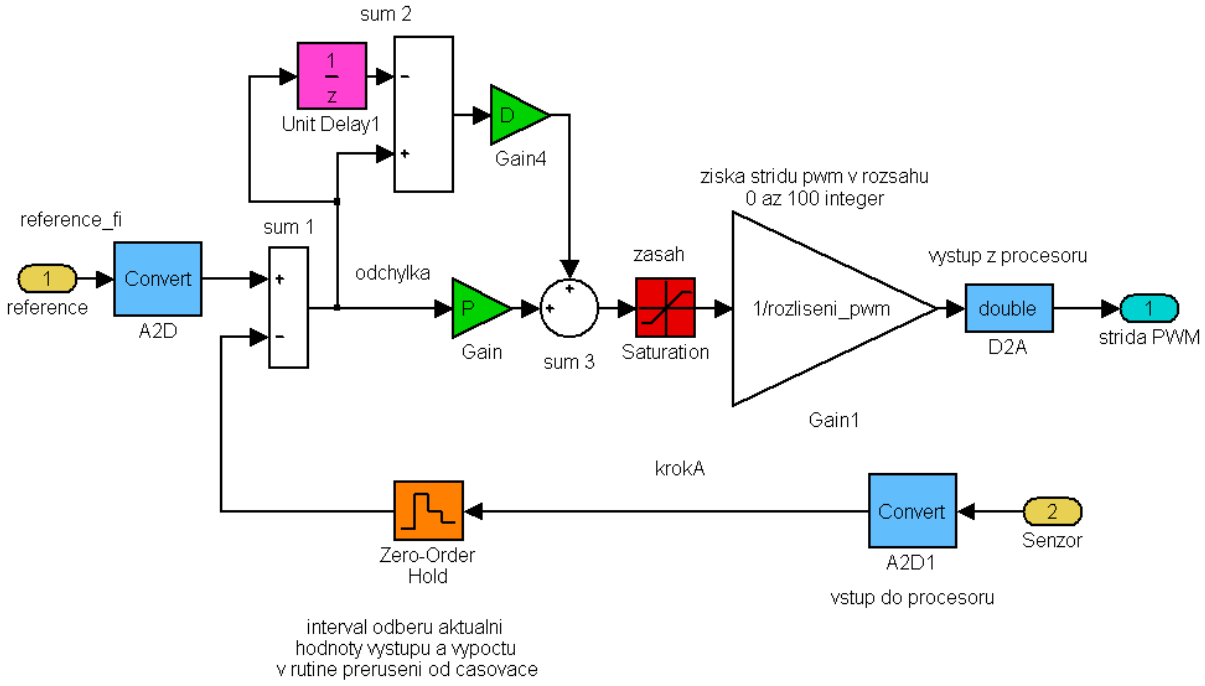
4.11 Implementace regulátoru

Jak již bylo řečeno výše, regulační výpočet je prováděn v rámci obslužné rutiny přerušení od vnitřního časovače (viz. příl.5 / kód 3). Regulace je typu PD.

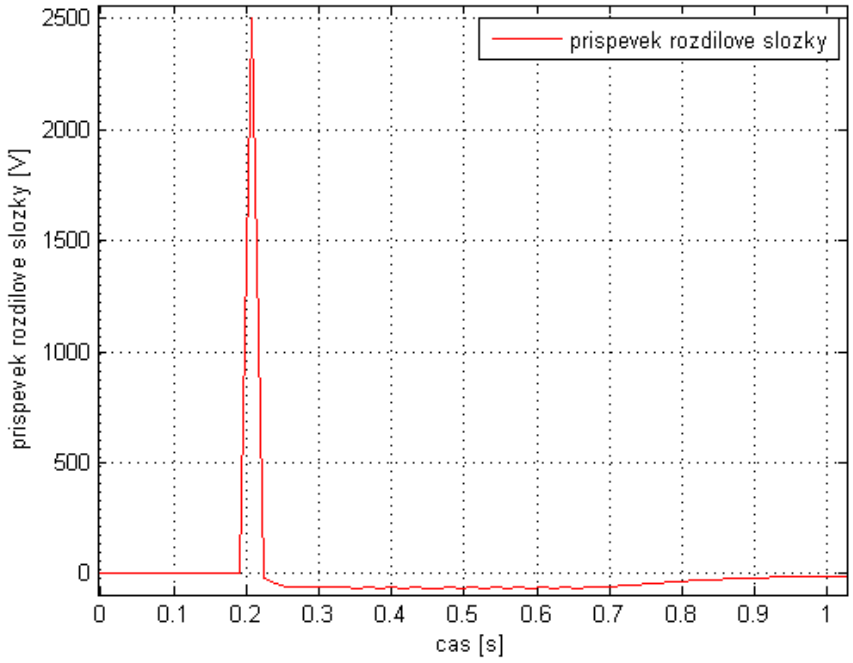
Nejprve je spočtena regulační odchylka odchylkaA jako rozdíl reference reference_fi a aktuálního počtu pulsů od IRC čidla krokA. Proporcionální část je rovna regulační odchylce vynásobené konstantou P:

```
odchylkaA = reference_fi - krokA;
stav_P = P * odchylkaA;
```

Rozdílová část v tomto případě není vypočítávána ze změny regulační odchylky (viz obr. 26). V takovém případě by totiž při skokové změně reference (např. na 2500) vypadaly výsledky „rozdílového základu“ odchylkaA – odchylkaB (rozdíl aktuální odchylky a odchylky v minulém vzorku) podle grafu 1. Tento postup vede k nastavení maximálního akčního zásahu v prvním vzorku a v dalších vzorcích (jak se regulační odchylka zmenšuje) působí jako „brzda“ a celkovou regulaci zpomaluje.



obrázek 26 – klasické zapojení rozdílové složky



graf 1 – průběh příspěvku rozdílové složky při klasickém zapojení

Řešením je výpočet rozdílového základu přímo z údaje senzoru (obr.29). V takovém případě není možné získat tak vysoký rozdílový základ v prvním kroku a každý další krok je větší než předcházející, a proto bude rozdílová složka působit ve směru regulace a proces se bude urychlovat. Rozdílový základ je pak vynásoben konstantou D. Nakonec je uložena aktuální hodnota čidla jako hodnota minulého výpočtu a připravena tak pro další výpočet.

```
stav_D = D * (krokA - krokB);
krokB = krokA;
```

Akční zásah se počítá jako součet rozdílové a proporcionální složky. Výsledek je pak ještě omezen, aby nedošlo k překročení maximálního dovoleného napětí na motoru.

```
zasah = stav_P + stav_D;
if(zasah > VOLTAGE_MAX){
        zasah = VOLTAGE_MAX;
  }
  if(zasah < -VOLTAGE_MAX){
        zasah = -VOLTAGE_MAX;
  }
```

Protože číselný typ *float* výrazně zpomaloval výpočetní proces a bylo pro zpřesnění řízení potřeba počítat s desetinnými čísli, rozhodl jsem se při použití typu *int_fast32_t* (32-bit integer) vynásobit maximální hodnotu použitelného napětí stem a tím získat dva řády napravo od desetinné čárky. Zároveň doba přerušení časovače je používána v celých milisekundách, a tak případné měření rychlosti by bylo v pulsechIRC/ms. Tento krok velice zpřesní výpočet za dosažení velké rychlosti výpočtu. Otázkou ještě zůstává, zdali nedojde k přetékání integerovske hodnoty. Použití 32-bitového Integeru nám dává rozsah přibližně

-2 147 483 648 až +2 147 483 647

a při posunu o 2 desetinná místa:

-21 474 836,48 až +21 474 836,47.

Při všech výpočtech se žádná hodnota k přetečení nepřiblížila ani zdaleka. To bylo ověřeno pomocí MATLAB Fixed-point Toolbox, který poskytuje funkci v Simulinku na monitorování přetečení používaných proměnných.

Přesto tu existuje omezení:

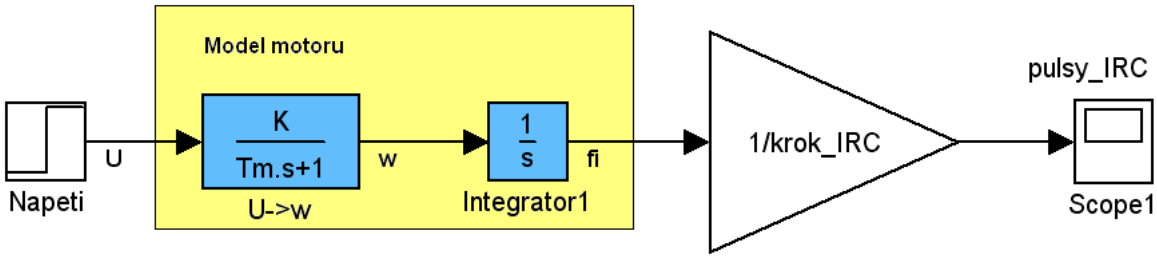
- Hodnota reference nesmí přesáhnout $+2^{31}-1$ a -2^{31} (pulsů IRC čidla), což je asi 429 000 otáček v obou směrech.

5 Matematické modelování

V této kapitole se budu věnovat stavbě modelu soustavy, který jsem sestavil v programu MATLAB/Simulink.

5.1 Experimentální identifikace motoru

Simulinkový model polohové odezvy motoru na skok je na obr.27. U motorů jsou při identifikaci uvažovány elektrická T_e a mechanická T_m časová konstanta. Protože je však elektrická časová konstanta většinou mnohokrát menší než konstanta mechanická, je možné ji při identifikaci preciznějších motorů zanedbat, zvláště pak při návrhu diskrétního řízení, kdy by musela být nastavena extrémně malá perioda vzorkování, aby byla elektrická časová konstanta vůbec zachycena.

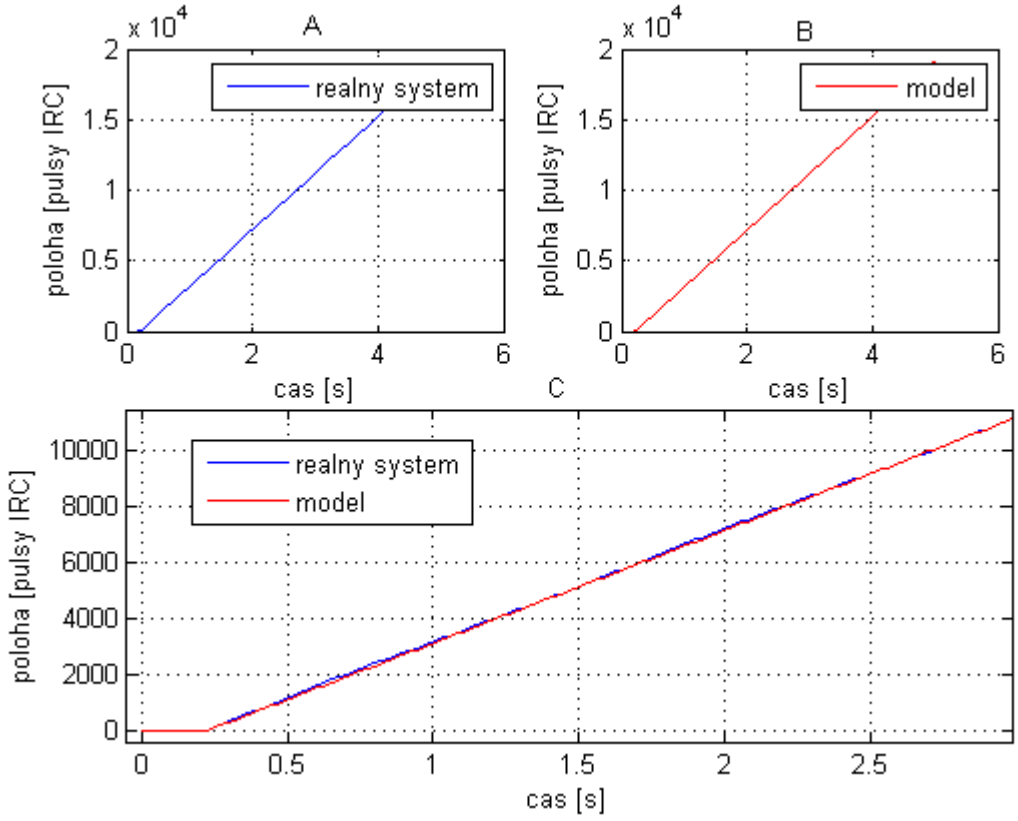


obrázek 27 – model polohové odezvy motoru na skok

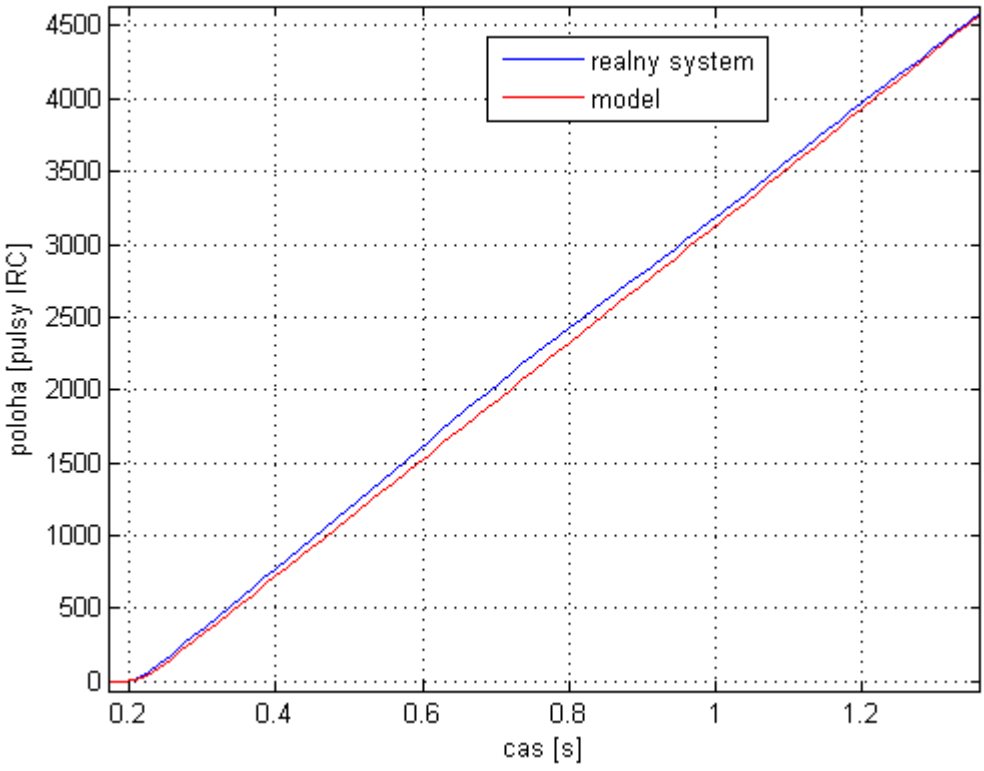
Odezva reálného motoru na skok je v grafu 2A. Přenos motoru byl experimentálně identifikován jako

$$P_{mot} = \frac{0{,}42}{0{,}019s + 1}$$

Graf 2B pak ukazuje odezvu tohoto modelu na tentýž skok. Pohledem na Graf 2C si můžeme povšimnout jistých odchylek modelu od reálného systému. To je dáno velice jednoduchým a osově nepřesným připevněním IRC čidla k motoru. Obě charakteristiky v pozdější fázi simulace však rostou v průměru se stejným sklonem, proto si myslím, že je určená konstanta K velmi podobná skutečnosti. Detailním pohledem na počátek obou průběhů v grafu 3 je vidět rozdíl průběhu počátečního kolena po připojení napájení. Tento fakt by mohl nasvědčovat odchylce mechanické časové konstanty T_m od reálného systému. Důvodem je jednak již zmiňované mechanické propojení motoru s IRC čidlem a také fakt, že jsem při simulaci držel jak motor tak IRC čidlo mých rukou (nebyl totiž k dispozici žádný nástroj pro tento účel), kde byla zvláště při rozjezdu patrná určitá vůle. Každé měření pak dávalo z globálního hlediska nepatrně rozdílné výsledky, kdy hodnota $T_m = 0{,}019$ je přibližně střední hodnotou.



graf 2 – odezva motoru na skok napájecího napětí



graf 3 – detail počátku odezvy motoru a modelu

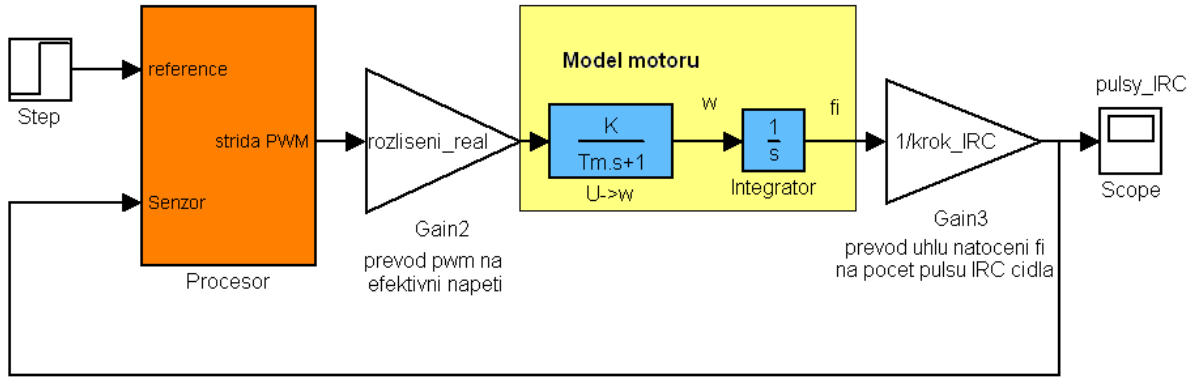
5.2 Model systému

Cílem bylo vytvořit takový model, který co nejvěrněji kopíruje danou situaci. MATLAB/Simulink se totiž implicitně snaží přiblížit se k reálnému světu. Proto jsou pro výpočty používány číselné typy s pohyblivou řádovou čárkou, floating-point typy (*float a double*), které umožňují vysokou přesnost systému a minimální periodu vzorkování, kdy se namodelované systémy blíží spojitým.

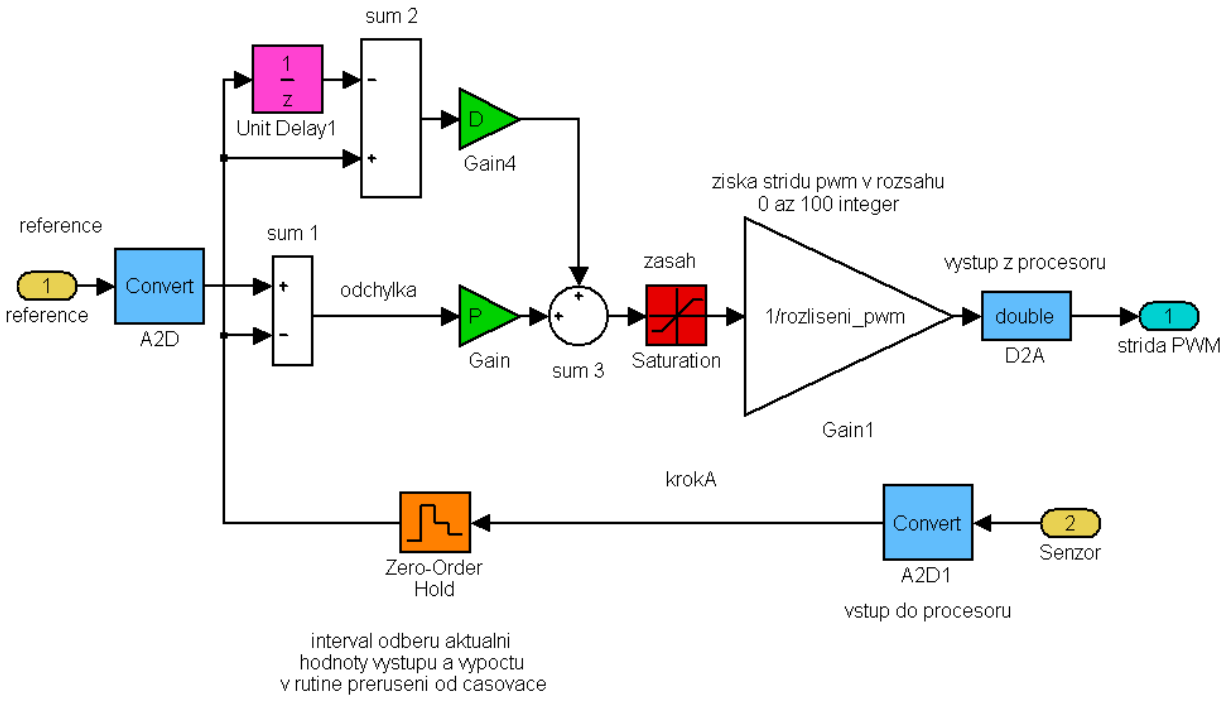
Náš mikroprocesor však pracuje s číselnými typy s pevnou řádovou čárkou (fixed-point), 32-bitový Integer, kde jsem pro zvýšení přesnosti při inicializaci systému vynásobil hodnotu maximálního napětí číslem 100, a od této doby je počítáno s přesností na setiny voltu, přestože se stále využívá výpočetně rychlý 32-bitový číselný typ Integer. Dále je v mikroprocesoru využívána nezanedbatelná perioda vzorkování (0,016 s), která činí tento systém diskrétním. Pro optimalizaci a zrychlení kódu je tato doba vynásobena tisícem a dále je veškerý časový údaj uvažován v celých násobcích 16 ms.

Tyto všechny vlastnosti je nutné nastavit pro jednotlivé bloky programu MATLAB/Simulink, aby byl namodelován daný systém. Navíc model musí uvažovat přechod mezi diskrétní částí mikroprocesoru a spojitou částí motoru.

Komplexní model systému je na obr.28. Jedná se o model motoru, jehož výstup je převeden přes integrátor a příslušnou konstantu na počet pulsu IRC, zatím typu double, tzn. že tento údaj není celočíselný. To je napraveno na vstupu bloku „procesor“, kde je *double* hodnota počtu pulsů IRC převedena na 32-bitový celočíselný typ. Blok „Procesor“ představuje diskrétní část systému. Blok „Step“ představuje skokovou změnu referenčního údaje.



obrázek 28 – komplexní Simulinkový model



obrázek 29 – Simulinkový model algoritmu implementovaného v mikroprocesoru

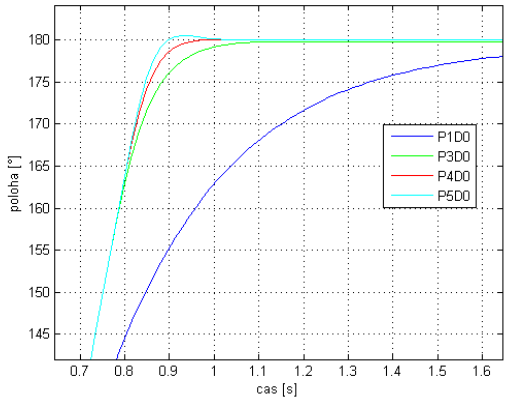
Model algoritmu výpočtu v mikroprocesoru (obr.29) jako diskrétní části systému kopíruje kód uvedený v příl.5. Oba vstupy (reference a údaj senzoru) jsou před vstupem do výpočtu převedeny na 32-bitový celočíselný typ. Údaj od senzoru je zadržován v intervalech 16ms blokem „Zero-Order Hold“. Tím je simulována perioda vzorkování údaje IRC čidla. Dále je tento údaj použit k regulačnímu výpočtu. V případě proporcionální složky je nejprve vypočítána regulační odchylka jako rozdíl reference a údaje senzoru a ta je pak vynásobena konstantou P. V případě rozdílové složky je blokem „Unit Delay1“ ukládána hodnota IRC čidla v minulém odměru. Pak je proveden rozdíl mezi aktuálním údajem IRC čidla a posledním změřeným údajem a to celé je vynásobeno konstantou D. Akční zásah je spočten jako součet proporcionální a rozdílové složky regulace a omezen saturačním blokem na +/- maximální hodnotu akčního zásahu. Blok „Gain 1“ pak převede akční zásah na délku pulsu PWM signálu, kdy maximální hodnota je shodná s délkou periody PWM reálného mikroprocesoru (100). Výstup mikroprocesoru je převeden na *double* číselný typ modelu reálného světa.

Konstanty modelu jsou uvedeny v příloze 6.

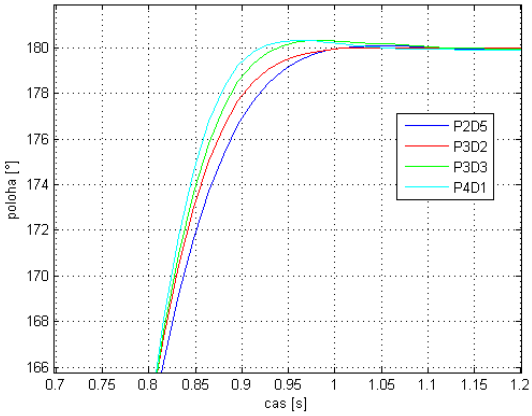
5.3 Nejlepší regulace

Pomocí navrženého modelu jsem se pokusil teoreticky najít nejlepší regulátor, za cíl jsem si dal co možná nejrychlejší náběh při nulovém překmitu. Nejdříve jsem nastavil několik P-regulátorů (graf 4) a PD-regulátorů (graf 5). Vybral jsem nejlepší P-regulátor (P = 4) a P-regulátor s rozdílovou složkou (P = 3, D = 2) a ty jsem porovnal mezi sebou (graf 6). Z porovnání je vidět, že nejlepším regulátorem je „pouhá“ proporcionální složka s konstantou P = 4.

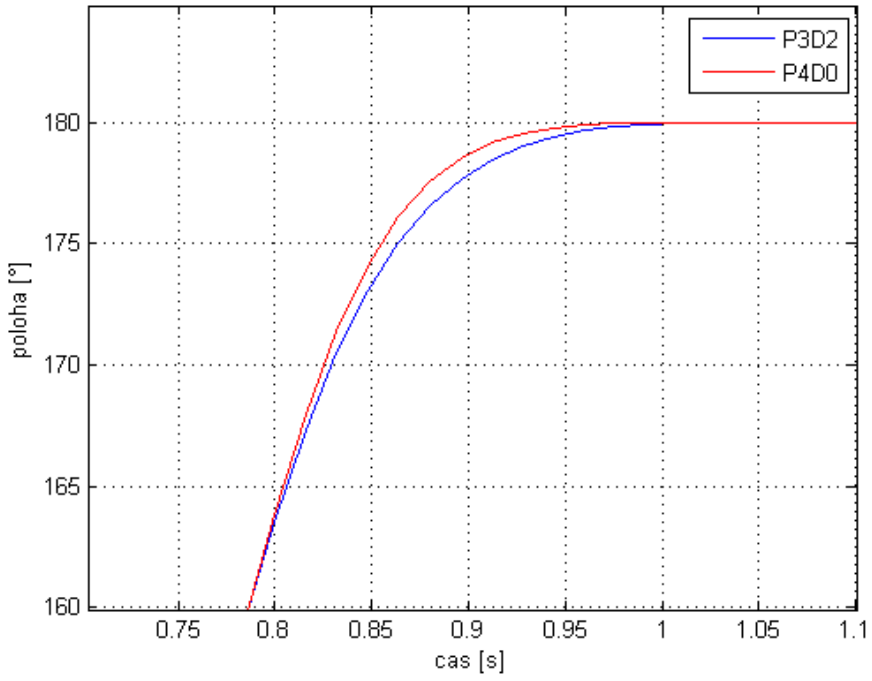
Podobné porovnávání regulátorů bohužel nebylo možné provést v praxi na reálném systému z důvodů, které jsem zmiňoval výše (nekvalitní mechanický spoj motoru s čidlem a držení motoru a čidla v rukou). Rozdíly mezi regulacemi reálných systémů byly patrné až při použití větších rozdílů mezi regulačními konstantami. Některé vybrané nastavení regulátorů jsou v příl.7.



graf 4– vybraná nastavení P-regulátoru



graf 5- vybraná nastavení P s rozdíl. Složkou



graf 6 – výběr nejlepší varianty regulátoru

6 Experimentální výsledky

Některé experimentální výsledky jsou v příl.7. Pohledem na ně je možné všimnout si několika hlavních zvláštností.

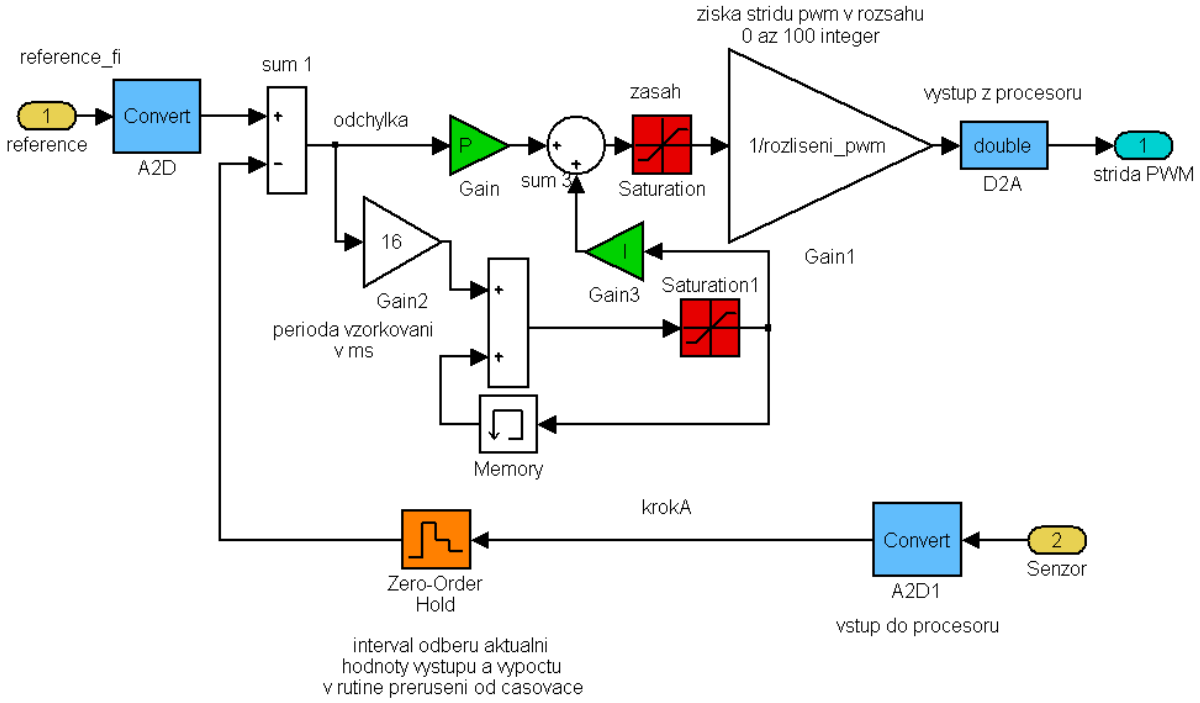
Za prvé je to zvláštní „hrbolek“ v charakteristice všech reálných systémů, jejichž průběh je bez překmitu (např. P = 1, D = 0). Jedno možné vysvětlení je následující.

Pokud chceme motor rozjet, musíme na něj působit větší silou (dodat větší energii), než je síla, která jej pak udrží v pohybu (překonáváme tzv. statické tření). Co když se bude ale motor blížit k nulové rychlosti; mohla by na něj začít působit podobná brzdná síla. To je také předpoklad této úvahy. Tato síla by pak působila jako zátěž na motoru, tím by zvýšila odpor motoru a při stejném napětí by motorem protékal větší proud. Protože je však motor brzděn, ne všechna dodaná energie se přemění v rotační pohyb a motor se nepatrně ohřeje. Tím se změní vlastnosti (např. vlastnosti tření působících na motor) a ten se pak najednou otočí o těch pár pulsů IRC signálu dál, než ukazuje simulace. Ve skutečnosti se jedná asi o 7 pulsů IRC signálu což je asi 0,5°.

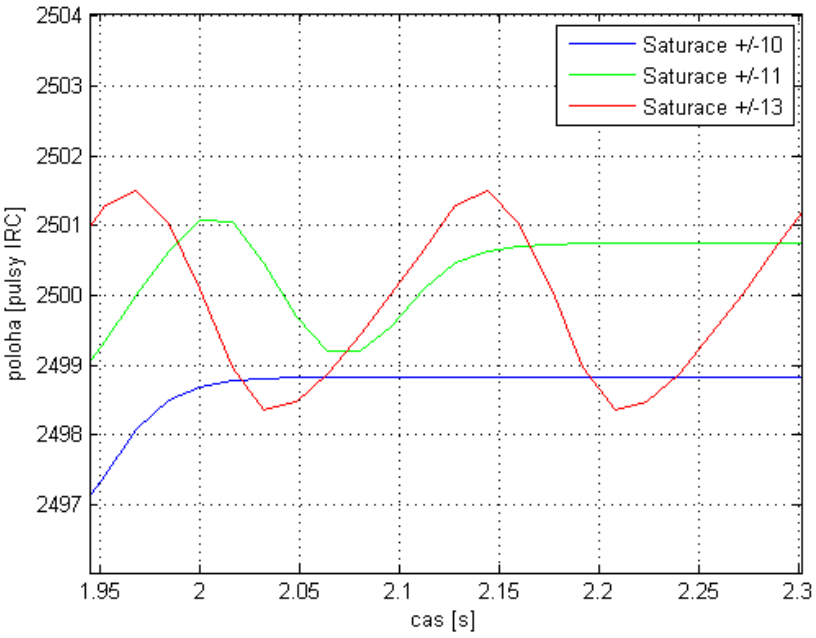
Druhá zvláštnost je, že při některých ustálených nastaveních regulátor nedosáhne nulové regulační odchylky. To je způsobeno rozlišením PWM, které je závislé na celočíselném typu Integer. Délka PWM je celočíselně rovna stu a proto lze napájení na motoru regulované právě tímto PWM měnit po setinách maximálního napájení. Pokud se výpočet dostane do stavu, kdy je akční zásah spočten na méně, než je právě hodnota jedné setiny maximálního napětí, dostáváme se do pásma necitlivosti, kdy je délka pulsu PWM zaokrouhlena na 0, čili dojde k odpojení motoru, i když je nenulová regulační odchylka. Řešením je buď precizní návrh PD regulátoru, nebo začlenění integrační složky do regulátoru, přestože v obvodu už jeden integrátor figuruje (viz. kapitola 4.10). Simulinkové schéma výpočtu s PI regulátorem je na obr.30. Regulační odchylka je vynásobena časem, po který bude platná (perioda vzorkování, t = 16ms). Dále je přičtena k celkovému integračnímu součtu a omezena, aby nedocházelo k překmitu, dokud nebude integrační součet vynulován díky záporné regulační odchylce. Hodnota součtu je jednak uložena pro další výpočet a také je ihned použita k výpočtu regulačního zásahu. Simulace však ukazují, že je použití integrační složky velmi omezené. Pokud chci i nadále zůstat u použití výpočetně rychlého Integeru, musí být i hodnota integrační konstanty I a saturačního omezení celočíselného typu. Pokusem na modelu jsem zjistil, že nejvyšší (a také jediná) v daném případě použitelná hodnota konstanty I je 1. Hodnota saturace musí být při I = 1 buď +/-11 nebo +/-12 (graf 7). Pouze při těchto dvou nastaveních jsem dosáhl nulové regulační odchylky. Pokud by byla hodnota saturace menší, nemusíme opustit pásmo citlivosti, neboť hodnota 11 vyvolá společně s nejmenší proporcionální složkou (P = 1) PWM signál o nejmenší možné střídě. Pokud je saturace zvolena vyšší než +/-12, soustava více či méně kmitá kolem referenční polohy. To je dáno spojením setrvačnosti motoru a velice jemným rozlišením IRC čidla. Integrační složka v tomto případě prostě vyvolá v čase tak dlouhý akční zásah (při dosažení referenční hodnoty přestane mít váhu proporcionální složka, ale Integrační složka má stále takovou váhu, že se motor stále otáčí. Integrační složka se sníží až při překmitnutí a začne působit opačným směrem.).

Poslední zajímavostí je fakt, že systém tak jak je postaven nemůže být nestabilní. To dosvědčuje nastavení P = 100, D = 0 a P = 250, D = 0. I přes obrovský rozdíl mezi konstantami P, výstupy obou systémů kmitají kolem referenční polohy a rozkmit kmitů je prakticky totožný. To je způsobeno omezeným akčním zásahem (viz. příl.5/kód 3). I při

minimálním překmitu je ihned nastaven akční zásah rovnající se maximálnímu napětí působícímu v opačném směru, než je pohyb motoru.



obrázek 30 - Simulinkový model algoritmu PI regulátoru



graf 7 – rozdíly v ustálených stavech při různých nastaveních saturace I složky, reference je 2500 pulsu IRC

7 Závěr

V této práci je detailně rozpracován postup návrhu řídícího systému pro stejnosměrný motor s pomocí mikroprocesoru . Nejprve je řešena hardwarová část, kde uvádím základní zapojení, jaké součástky jsem použil a proč, jejich hodnoty potřebné pro návrh systému a jak součástka pracuje. Použité součástky jsou rozděleny na součástky na zpracování signálu z IRC čidla, které upravují signál do takové formy, která je očekávána procesorem, a na součástky zpracování signálu PWM na cestě od mikroprocesoru k výkonové části motoru. Součástky IRC části skutečně signál upravují, zatímco součástky PWM části jsou víceméně bezpečnostní před H-můstkem, pro ochranu motoru a výpočetní elektroniky. Uprostřed veškeré elektroniky je řídící mikroprocesor ATmega8515.

V další části návrhu jsem se věnoval nastavení mikroprocesoru, což obnášelo nastavení tzv. fúzních bitů, které určují v průběhu programu neměnitelné vlastnosti mikroprocesoru (např. operační rychlost), a nastavování vnitřních registrů, abych zpřístupnil funkce potřebné pro vykonávání požadovaných operací. K oběma předchozím úkolům bylo nutné prostudovat potřebné manuálové listy.

V poslední fázi jsem navrhnul model kopírující reálný navržený systém v programu MATLAB/Simulink. V této fázi jsem měl již připraveny všechny potřebné nástroje pro alespoň přibližnou experimentální identifikaci zkušebního motoru. Dále jsem pomocí navrženého modelu zjistil nejlepší použitelný regulátor a nakonec jsem otestoval několik různých nastavení použitého regulátor a analyzoval rozdíly od namodelovaného systému.

Problémy se vyskytli především při experimentálním testování motoru, kdy jednak nebyla přesně propojena hřídel motoru s hřídelí IRC čidla a motor i IRC čidlo jsem držel v rukou, neboť by shánění a montáž nějakého preciznějšího vybavení byly pro účely této práce značně neefektivní. Dalším problémem byla přesnost výpočtu, kdy první verze programu využívala číselné typy float a double (typy s pohyblivou řádovou čárkou). 8-bitový mikroprocesor ATmega8515 sice umí s těmito typy pracovat, ale výpočet je velice pomalý. Když k tomu připočteme použití těchto datových typů ve složitějších výpočtech obslužné rutiny nějakého přerušení, snadno se stane, že je procesor natolik zaneprázdněn, že nestihne vykonat rutiny jiných přerušení. Tak se stalo i v tomto případě, kdy byl značně zkreslen údaj od IRC čidla. V hlavní části optimalizovaného kódu jsou využívány pouze celočíselné 32-bitové typy. Pro zvýšení přesnosti při zachování celočíselných typů jsou na počátku výpočtu některé údaje vynásobeny konstantou 10^x a tím je vlastně posunuta desetinná o x čísel doleva a v dalším výpočtu už je daná veličina uvažována v x-tinách (např. při vynásobení veličiny číslem 100 dále počítáme v setinách této veličiny). Zjistil jsem, že 32-bitový typ i přes vynásobení konstantou umožňuje využít plného rozsahu číselných hodnot, které jsou systémem dosažitelné. Myslím, že toto řešení je dobrým kompromisem mezi rychlostí výpočtu, přesností a maximální použitelnou hodnotou.

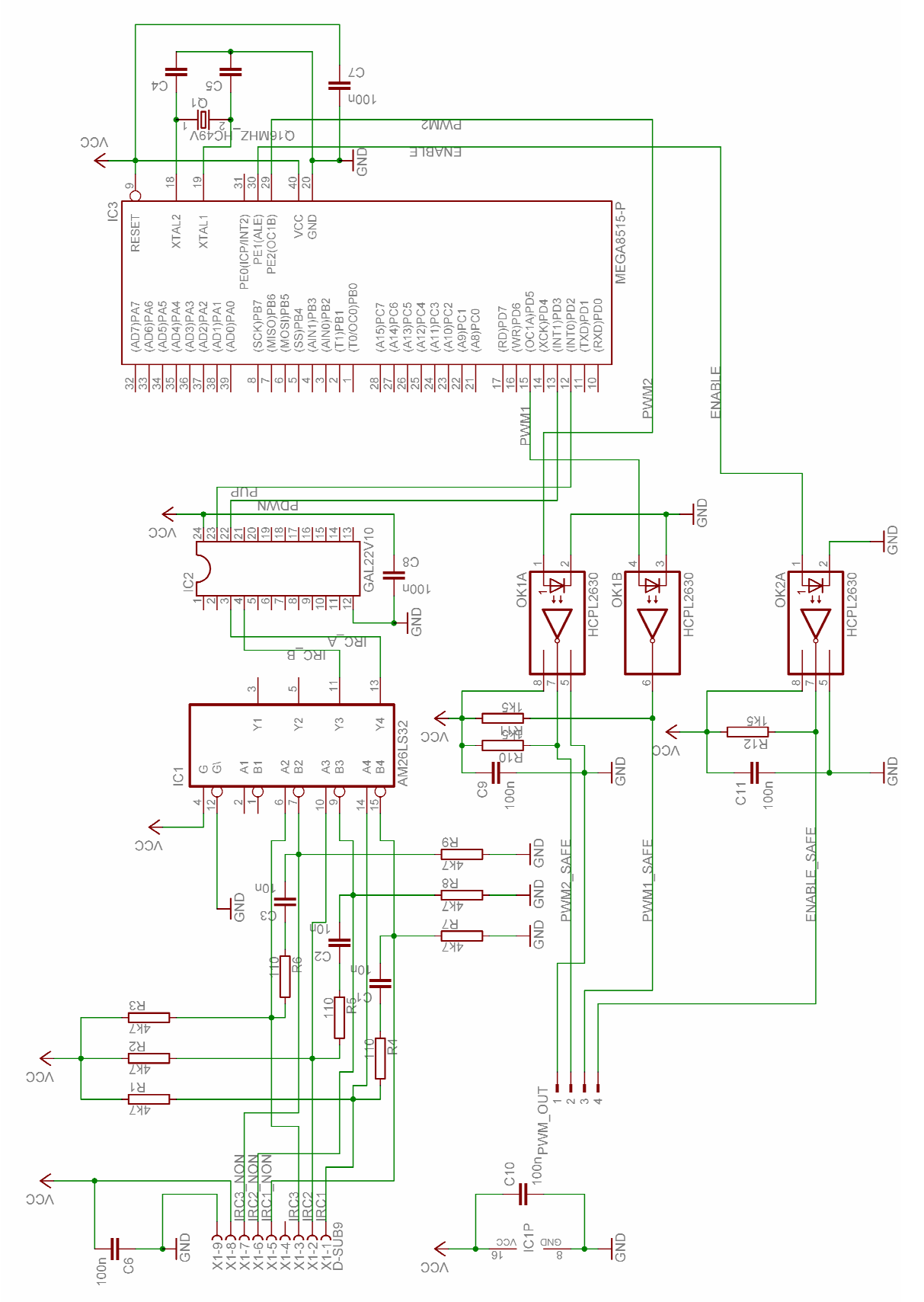
Během práce jsem se naučil především hledat potřebné informace, zacházet s elektrickými součástkami a jejich dokumentacemi, získal jsem základní rozhled v oblasti hardwaru, naučil jsem se základy návrhu desky plošných spojů, začal jsem chápat a využívat pojmy z oblasti mikroprocesorů, naučil jsem se zřídit sériovou komunikaci mezi PC a mikroprocesorem. Dále jsem si utvrdil dosavadní znalosti v oblasti řízení a v programování v jazyce C.

Literatura

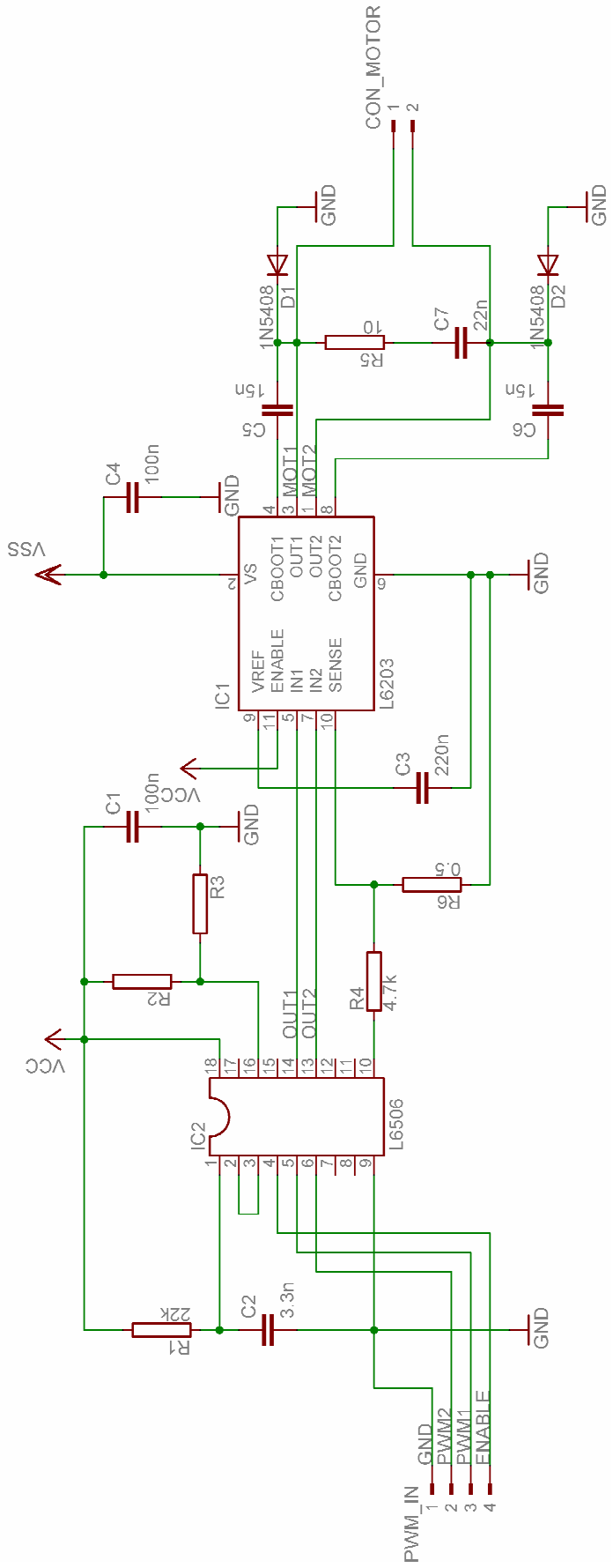
[1] Vysoký O.: Elektronické systémy II. Praha: ČVUT. l997
[2] Gene F. Franklin, J. D. Powell, Abbas Emami-Naeini: Feedback Control of Dynamic Systems, USA, 2006
[3] ATmega8515-16PU, katalogový list, Atmel
[4] 1N5408, katalogový list, DC Components
[5] GAL22V10, katalogový list, Lattice
[6] L6203, katalogový list, SGS-Thomson Microelectronics
[7] L6506, katalogový list, SGS-Thomson Microelectronics
[8] HCPL-2630, katalogový list, Agilent Technologies
[9] AM26LS32, katalogový list, Motorola
[10] avr-libc Reference Manual 1.4.5, 2006
[11] Šusta R., Šivič J.: Programování obvodů PLD v ORCADu: ČVUT-FEL Praha, http://dce.felk.cvut.cz/lsy/cviceni/pdf/galnavod3.pdf, 2001
[12] Nápověda programu MATLAB R2006a
[13] Nápověda programu AVR studio 4
[14] Nápověda programu EAGLE 4.16r2
[15] http://www.alldatasheet.com/
[16] http://www.datasheetcatalog.com/
[17] http://www.ite.tul.cz/form/bzt.html, pomůcka pro identifikaci rezistorů
[18] http://www.embedded.com/2000/0010/0010feat3.htm
[19] http://www.essapraha.cz/
[20] http://cs.wikipedia.org

Přílohy

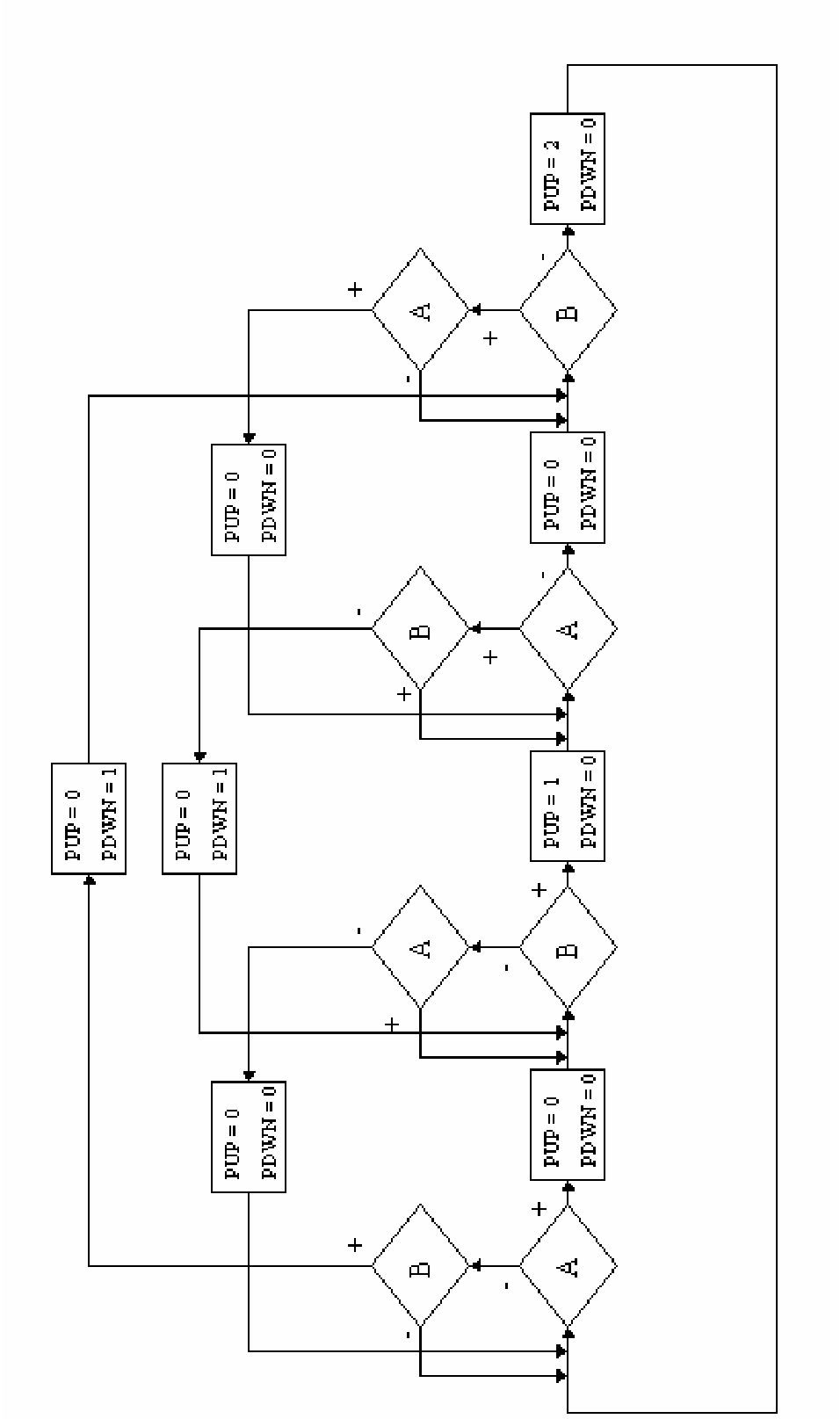
Příloha 1 – schéma zapojení elektronické části



Příloha 2 – schéma zapojení výkonové části



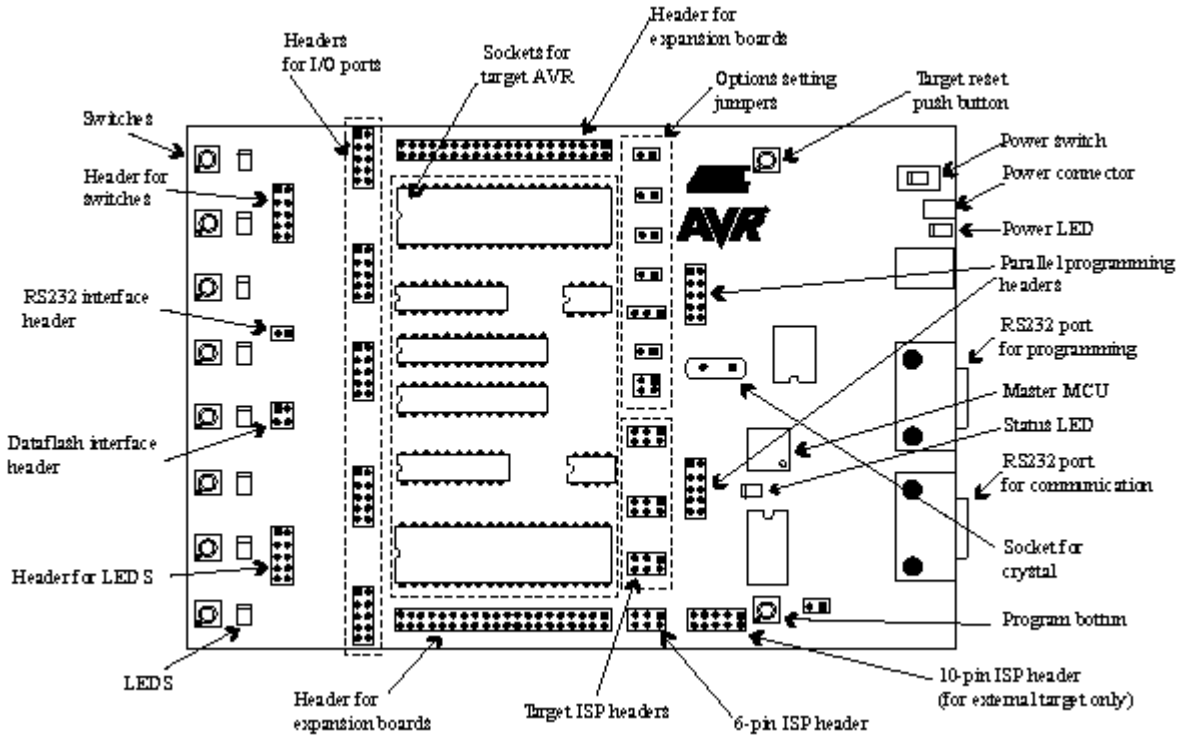
Příloha 3A – stavové schéma návrhu programu pro GAL22V10



Příloha 3B – zdrojový kód programu pro GAL22V10

```
|GAL22V10
| 2..4:(RESET,A,B), 23..22:(PUP,PDWN), 21..19:(Q[0..2])

| Low: RESET

|
| Procedure: RESET, Q[0..2]
| {
| States:
Stav00Forward=0,Stav10Forward=1,Stav11Forward=2,Stav01Forward=3,Stav00Backward=4,Stav01Backward=5,Stav11Ba
ckward=6,Stav10Backward=7

| Stav00Forward.PUP = 1
|                       PDWN = 0
|                       A ? -> Stav10Forward
|                       B ? -> Stav01Backward
|                          -> Stav00Forward

| Stav10Forward.PUP = 0
|                       PDWN = 0
|                       B ? -> Stav11Forward
|                       A ? -> Stav10Forward
|                          -> Stav00Backward

| Stav11Forward.PUP = 1
|                       PDWN = 0
|                       A' ? -> Stav01Forward
|                       B  ? -> Stav11Forward
|                           -> Stav10Backward

| Stav01Forward.PUP = 0
|                       PDWN = 0
|                       B' ? -> Stav00Forward
|                       A  ? -> Stav11Backward
|                           -> Stav01Forward

| Stav00Backward.PUP = 0
|                        PDWN = 0
|                        A ? -> Stav10Forward
|                        B ? -> Stav01Backward
|                           -> Stav00Backward

| Stav01Backward.PUP = 0
|                        PDWN = 1
|                        B' ? -> Stav00Forward
|                        A  ? -> Stav11Backward
|                            -> Stav01Backward

| Stav11Backward.PUP = 0
|                        PDWN = 0
|                        A' ? -> Stav01Forward
|                        B  ? -> Stav11Backward
|                            -> Stav10Backward

| Stav10Backward.PUP = 0
|                        PDWN = 1
|                        B ? -> Stav11Forward
|                        A ? -> Stav10Backward
|                           -> Stav00Backward
|}
```

Příloha 4 – STK500



Příloha 5 – celkový kód

```
#define __AVR_ATmega8515__
#define __OPTIMIZE__

/*nastaveni pro f_osc 16000000*/
#define F_CPU 16000000
#define BAUD 103                        //rychlost USART 9600 baudu

#define NASOBNOST 100
#define DELKA_PERIODY_PWM 100           //delka periody PWM
#define VOLTAGE_MAX (12 * NASOBNOST)    //maximalni napeti na motoru posurute o rad dany nasobnosti
#define KROKU_NA_OT  5000               //pocet prijatych impulsu z IRC cidla a pomocnych
                                        //obvodu behem jedne otacky
#define PRESCALE 1024                   //hodnota, kterou bude TIMER0 delit F_CPU
#define OCR0_value 250                  //hodnota OCR0

/*regulator*/
#define P 10            //proporcionalni slozka regulatoru
#define D 0             //rozdilova konstanta regulatoru


#include <stdlib.h>
#include <avr\io.h>
#include <avr\interrupt.h>
#include <util\Delay.h>


int_fast32_t voltage_pom;       //pomocna promenna pouzivana ve funkci pwm
int_fast32_t rozliseni_pwm;     //rozliseni pwm [V*10^-2/pwm_element]
int_fast8_t delka_pulsu;        //delka pulsu typu potrebnem pro prirazeni do registru
uint_fast32_t pom1;             //pomocna promenna pouzivana ve fci USART_Transmit2
uint_fast32_t pom2;             //pomocna promenna pouzivana ve fci USART_Transmit2
unsigned int i;                 //pomocna promenna pouzivana ve fci USART_Transmit2
int_fast32_t reference_fi;      //reference uhlu natoceni
int_fast32_t odchylkaA;         //aktualni regulacni odchylka
int_fast32_t odchylkaB;         //regulacni odchylka v minulem vypoctu
int_fast32_t zasah;             //regulacni zasah
int_fast32_t stav_P;            //prispevek P slozky
int_fast32_t stav_D;            //prispevek D slozky

int_fast32_t krokA;             //pocet prijatych impulsu od IRC v obou smerech (jejich rozdil)
int_fast32_t krokB;             //predchozi stav poctu kroku
int_fast32_t krok_timer;        //pocet prijatych impulsu od IRC v obou smerech v dobe odecteni timerem
int_fast32_t t_inter;           //doba behu casovace TIMER0 mezi jednotlivymi prerusenimi (resp.
                                //doba mereni poctu kroku) v milisekundach
float v_ot_min;                 //rychlost v otackach za minutu
int_fast32_t aktual_cas;        //doba mereni v milisekundach
int_fast32_t aktual_hodnota;    //hodnota za aktual_cas


void init(void);
int pwm (int_fast32_t voltage);
void USART_Init( unsigned char baud );
void USART_Transmit1( unsigned int data );
void USART_Transmit2( unsigned int data );
void TIMSK_time_init(void);     //spocte a nastavi t_inter
void tiskni(int_fast32_t cas, int_fast32_t hodnota);
```

```
/*-------------------------------------------------------------------------------------------------------------------------------*/
ISR(INT0_vect)          //rutina externiho preruseni INT0
{
        krokA++;
}
```
kód 1

```
/*-------------------------------------------------------------------------------------------------------------------------------*/
ISR(INT1_vect)          //rutina externiho preruseni INT1
{
        krokA--;
}
```
kód 2

```
/*-------------------------------------------------------------------------------------------------------------------------------*/
ISR(TIMER0_COMP_vect)       //rutina preruseni od vnitrniho casovace
{
        aktual_cas += t_inter;
        aktual_hodnota = krokA;

        odchylkaA = reference_fi - krokA;
        stav_P = P * odchylkaA;
        stav_D = D * (krokA - krokB);

        krokB = krokA;

        zasah = stav_P + stav_D;

        /*omezeni akcniho zasahu*/
        if(zasah > VOLTAGE_MAX){
                zasah = VOLTAGE_MAX;
        }
        if(zasah < -VOLTAGE_MAX){
                zasah = -VOLTAGE_MAX;
        }
        pwm(zasah);
}
```
kód 3

```
/*-------------------------------------------------------------------------------------------------------------------------------*/
int main(void)
{
        /*init*/
        init ();                    //inicializace pouzivanych promennych
        USART_Init(BAUD);   //inicializace seriove linky na rychlost 9600 baud pri fosc = 16 MHz
        TIMSK_time_init();

        /*nastaveni registru*/
        /*externi preruseni*/
        GICR = (1<<INT1)|(1<<INT0);             //povoleni externiho preruseni INT1 a INT0
        MCUCR = (1<<ISC00)|(1<<ISC01)|(1<<ISC10)|(1<<ISC11);  //INT1:0 reaguji na nabeznou hranu

        /*nastaveni casovace*/
        TCCR0 = (1<<WGM01)|(1<<CS02)|(1<<CS00);   //nastaveni casovace, 1puls/1024(pulsuI/O)

        TIMSK = (1<<OCIE0);                             //preruseni pri shode registru TCNT0 s OCR0
        OCR0 = OCR0_value;                              //hodnota, se kterou budeme porovnavat

        /*Phase and frequency correct PWM, no prescaler*/
        TCCR1A = (1<<COM1A1)|(1<<COM1B1);
        TCCR1B = (1<<WGM13)|(1<<CS10);

        DDRD = (1<<PD5);                //nastaveni PINu PD5 (resp. OC1A) jako vystup
        DDRE = (1<<PE2)|(1<<PE1);   //nastaveni PINu PE2 (resp. OC1B) a PE1 (sig. ENABLE) jako vystup
```

```
  ICR1 = DELKA_PERIODY_PWM;         //delka periody PWM
  OCR1A = 0;
  OCR1B = 0;
  PORTE = 0b010;                    //log.1 na pinu PE1 slouzi jako ENABLE pro PWM
  sei();                            //povoleni preruseni
  /*program*/
  USART_Transmit1( '\n' );

  /*zpozdeni pred skokem referencni hodnoty*/
  tiskni(aktual_cas,aktual_hodnota);
  _delay_ms(100);
  _delay_ms(100);
  _delay_ms(100);
  _delay_ms(100);
  _delay_ms(100);
  tiskni(aktual_cas,aktual_hodnota);

  reference_fi = 2500;      //skok reference

        for (;;) {
              tiskni(aktual_cas,aktual_hodnota);
        }       /* loop forever */
}

void init(void){      //inicializace pouzivanych promennych
  rozliseni_pwm = (VOLTAGE_MAX/DELKA_PERIODY_PWM);
  voltage_pom = 0;
  delka_pulsu = 0;
  t_inter = 0;
  krokA = 0;
  krokB = 0;
  krok_timer = 0;
  v_ot_min = 0;
  zasah = 0;
  odchylkaA = 0;
  odchylkaB = 0;
  stav_P = 0;
  stav_D = 0;
  i = 0;
  aktual_cas = 0;
  aktual_hodnota = 0;
}
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
void tiskni(int_fast32_t cas, int_fast32_t hodnota){  //vypis potrebnych udaju pres seriovou linku na konzoli
        USART_Transmit1( '\n' );
        USART_Transmit2( cas );
        USART_Transmit1( ' ' );
        USART_Transmit2( hodnota );
        USART_Transmit1( ';' );
        USART_Transmit1( '\r' );
}
```

kód 4

```
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
int pwm (int_fast32_t voltage)
{
  if (voltage == 0){
        TCCR1A = 0x00;          //odpojeni PINu OC1A a OC1B
        return 0;
  }
```

```
        /*vypocet delky pulsu pwm*/
          voltage_pom = abs(voltage);
          delka_pulsu = (int_fast8_t)(voltage_pom / rozliseni_pwm);
          OCR1A = delka_pulsu;
          OCR1B = OCR1A;

          if (voltage > 0){
                  TCCR1A = (1<<COM1A1);       //pripojeni PINu OC1A
          }
          else {
                  TCCR1A = (1<<COM1B1);       //pripojeni PINu OC1B
          }
          return 0;
        }
```

kód 5

```
/*-------------------------------------------------------------------------------------------------------------------------------*/

        void USART_Init( unsigned char baud )
        {
          /* nastaví baudovou rychlost */
          UBRRH = (unsigned char)(baud>>8);
          UBRRL = (unsigned char)baud;

          UCSRB = (1<<RXEN)|(1<<TXEN);                    //povolí receiver a transmitter
          UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);        //nastaví formát posílaných dat: 8data, 2stop bit
        }
/*-------------------------------------------------------------------------------------------------------------------------------*/
        void USART_Transmit1( unsigned int data )
        {
          while ( !( UCSRA & (1<<UDRE)) )         // čekej na prázdný UDR registr
          ;
          UDR = data;                             // pošli data
}
```

kód 6

```
/*-------------------------------------------------------------------------------------------------------------------------------*/
          void USART_Transmit2( unsigned int data )  //rozebere cisla na cislice a odesila zvlast pomoci
                                                     //USART_transmit1 (do peti cifer)
        {
          i=10000;                          //posílání pěticiferného čísla
          pom1 = data;

          while (i > 0){                    //rozebrani cisla na cislice
                  pom2 = pom1 / i;          //získá číslici daného řádu
                  if (pom2 > 0){
                          pom1 = pom1 - ( pom2 * i);      //"umazání" nejvyšší číslice z celkového čísla
                  }
                  USART_Transmit1(pom2 + 48);           //číslice převedená do ASCII kódu
                  i = i/10;
          }
        }
```

kód 7

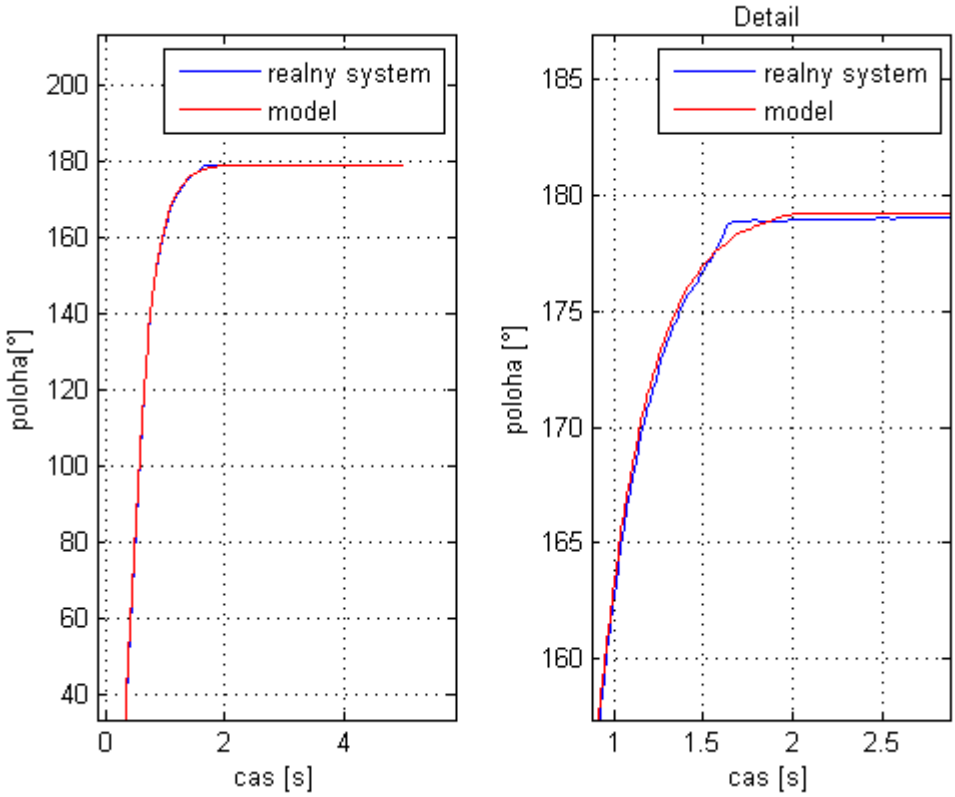
```
/*-------------------------------------------------------------------------------------------------------------------------------*/
        void TIMSK_time_init(void)          //spocte a nastavi t_inter
        {
          float f_presc;        //frekvence prescaled casovace s uvahou citani casovace do hodnoty OCR0_value
          float t_element;      //doba elementarniho inkrementu casovace TIMER0 v milisekundach

          f_presc = F_CPU / PRESCALE / OCR0_value;
          t_element = (1/f_presc) * 1000;

          t_inter = (int_fast32_t)t_element;
        }
```

Příloha 6 – M-file pro Simulinkový model nastaveni.m

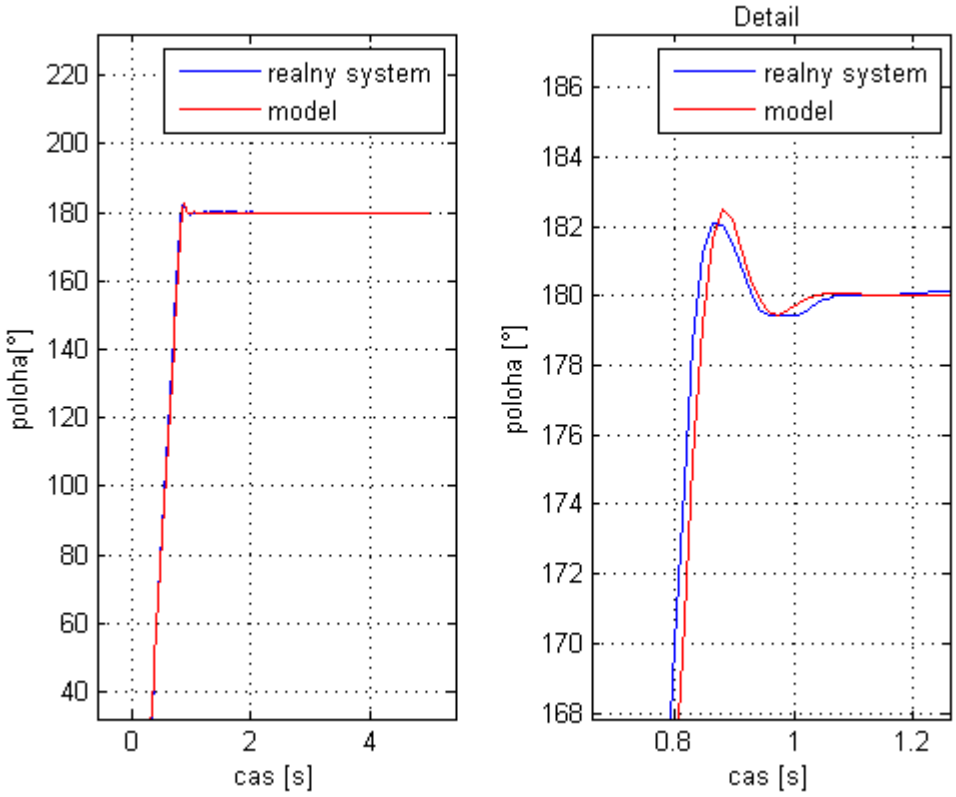
```
NASOBNOST = 100;                    %posun v desetinne carce
MAX_VOLTAGE = 12 * NASOBNOST;       %maximalni napeti v setinach voltu
DELKA_PERIODY_PWM = 100;            %dano nastavenym PWM generatorem
KROKU_NA_OTACKU = 5000;             %dano prichozim signalem od IRC cidla

F_CPU = 16000000; %rychlost mikroprocesoru
PRESCALE = 1024;
OCR0_value = 250;

%konstanty regulatoru
P = 1;  %P slozka
D = 0;  %D slozka

%konstanty motoru
K = 0.42;
Tm = 0.019;

rozliseni_pwm = MAX_VOLTAGE / DELKA_PERIODY_PWM;  %rozliseni pwm v setinach
                                                  %voltu
rozliseni_real = rozliseni_pwm / NASOBNOST;     %rozliseni pwm ve voltech
krok_IRC = 2*pi / KROKU_NA_OTACKU;        %uhel,o ktery se IRC pootoci, aby
                                          %doslo ke vzniku pulzu
timer = (PRESCALE / F_CPU) * OCR0_value; %perioda vzorkovani
```

Příloha 7 – Průběhy vybraných regulací polohy

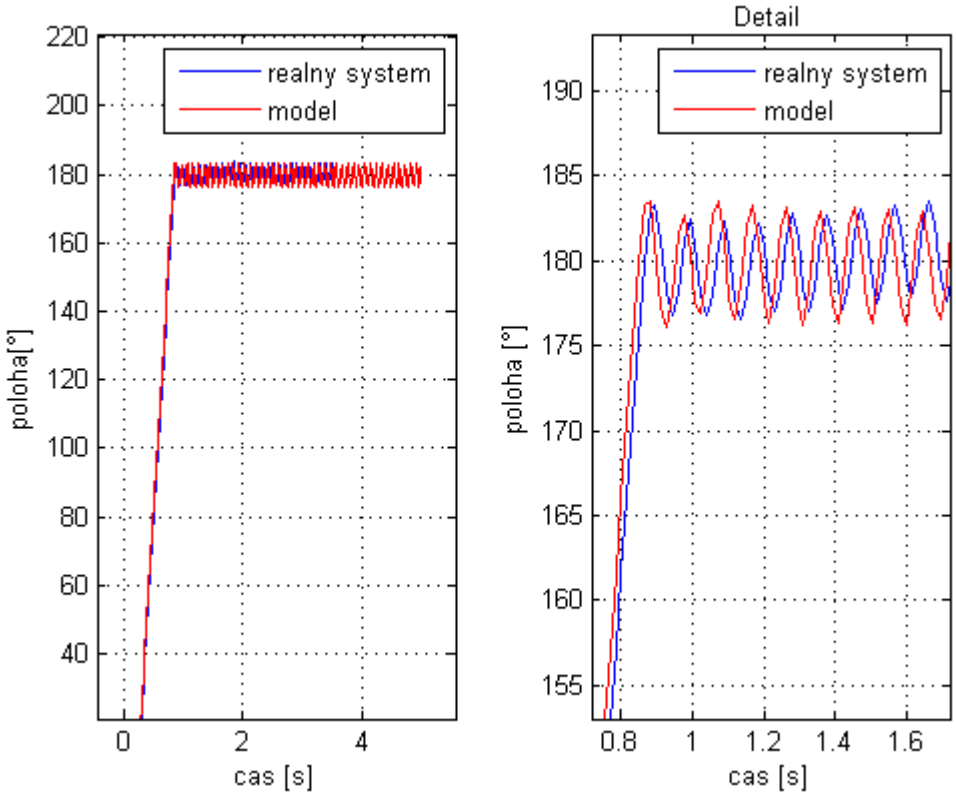
Regulátor P = 1, D = 0



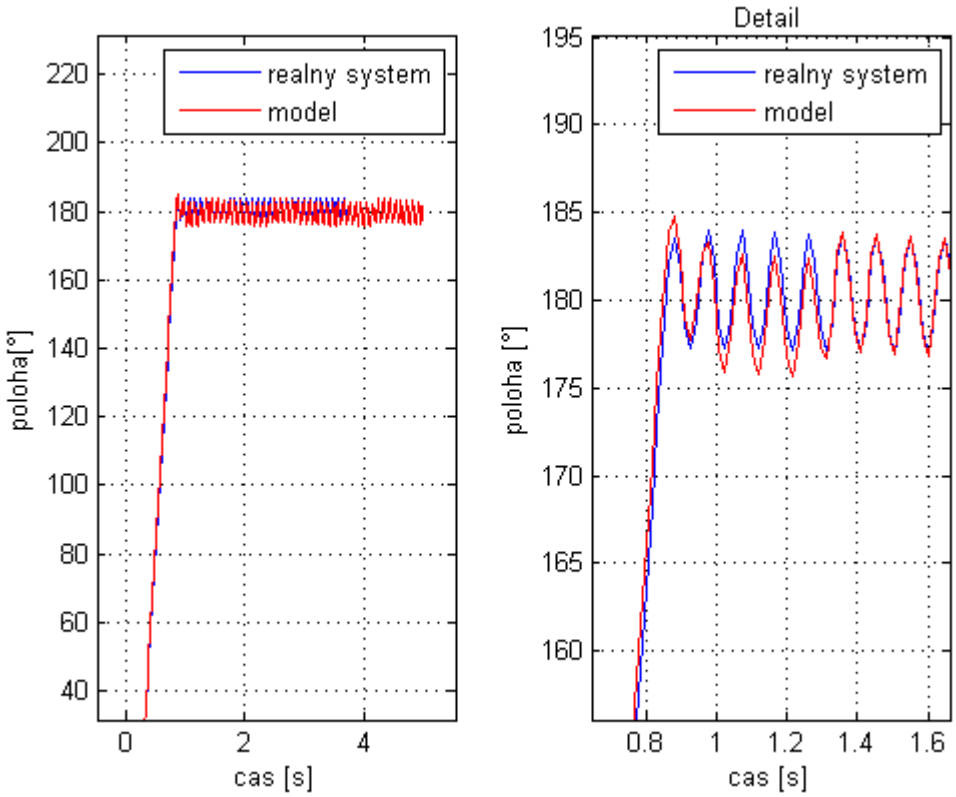
Regulátor P = 10, D = 0



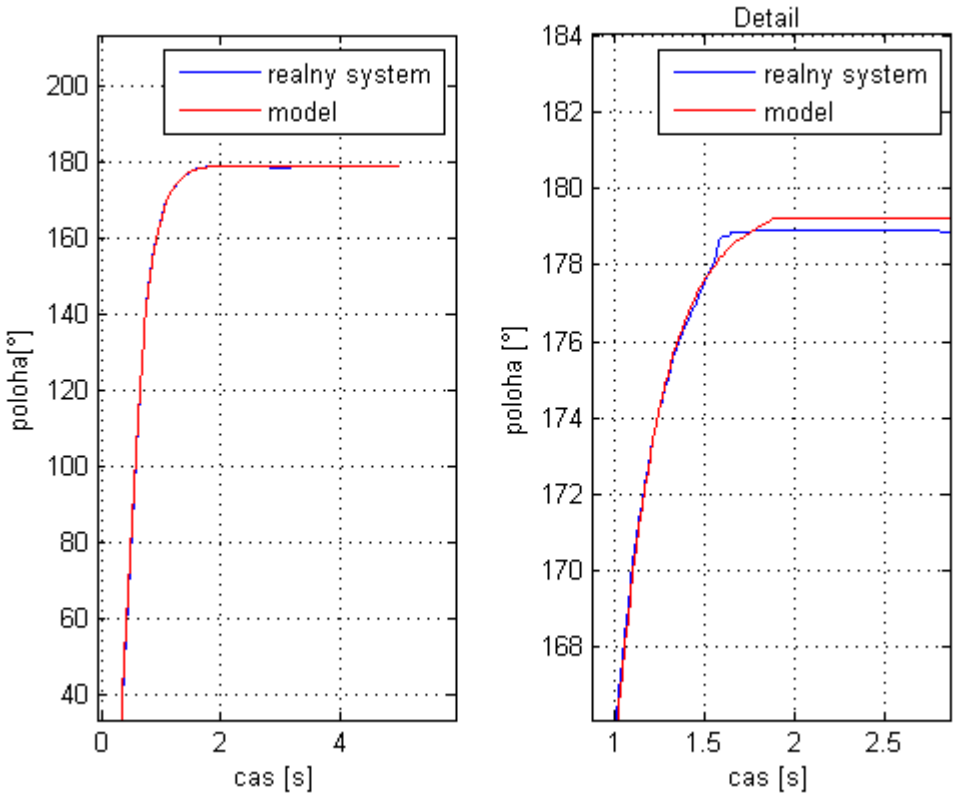
Regulátor P = 100, D = 0



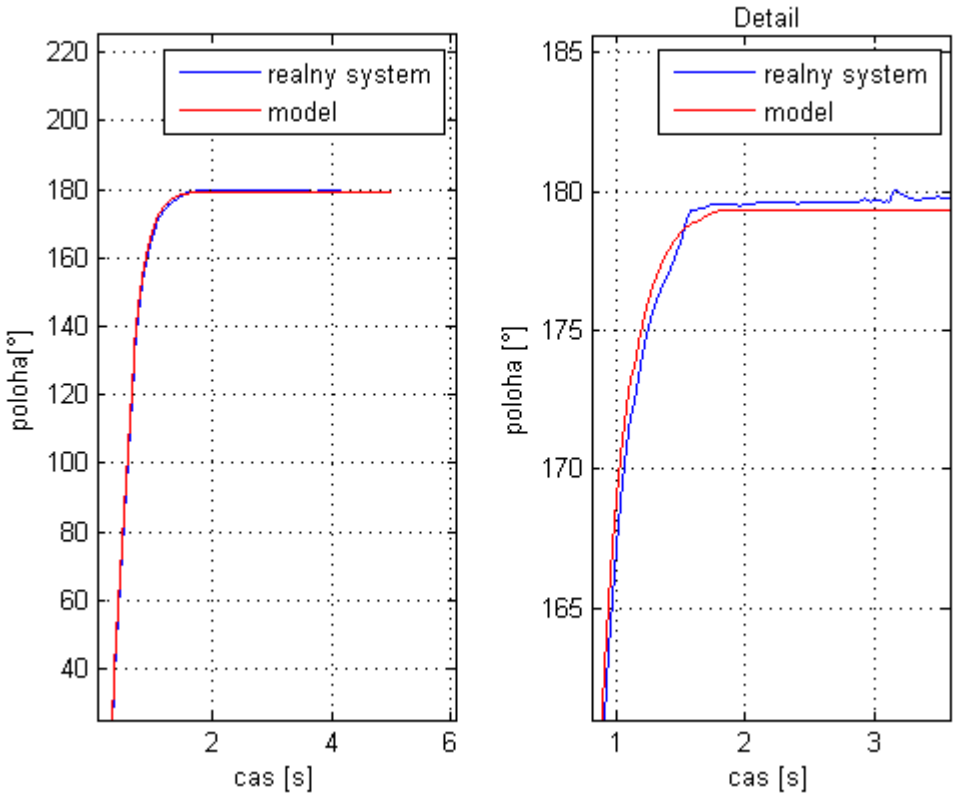
Regulátor P = 250, D = 0



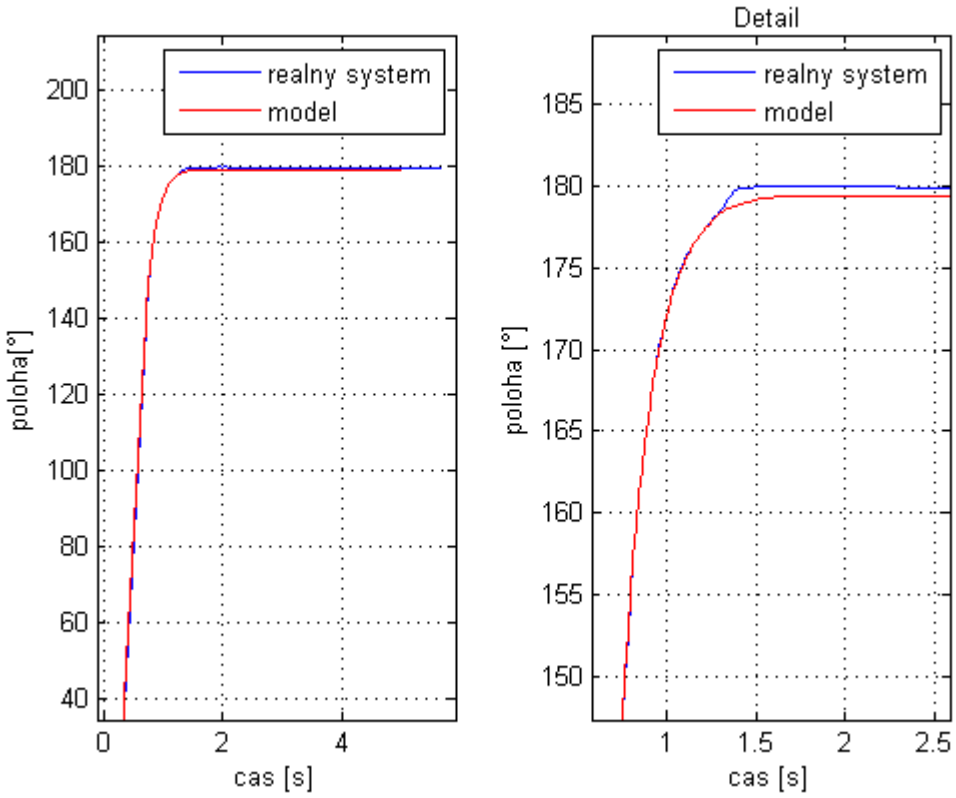
Regulátor P = 1, D = 1



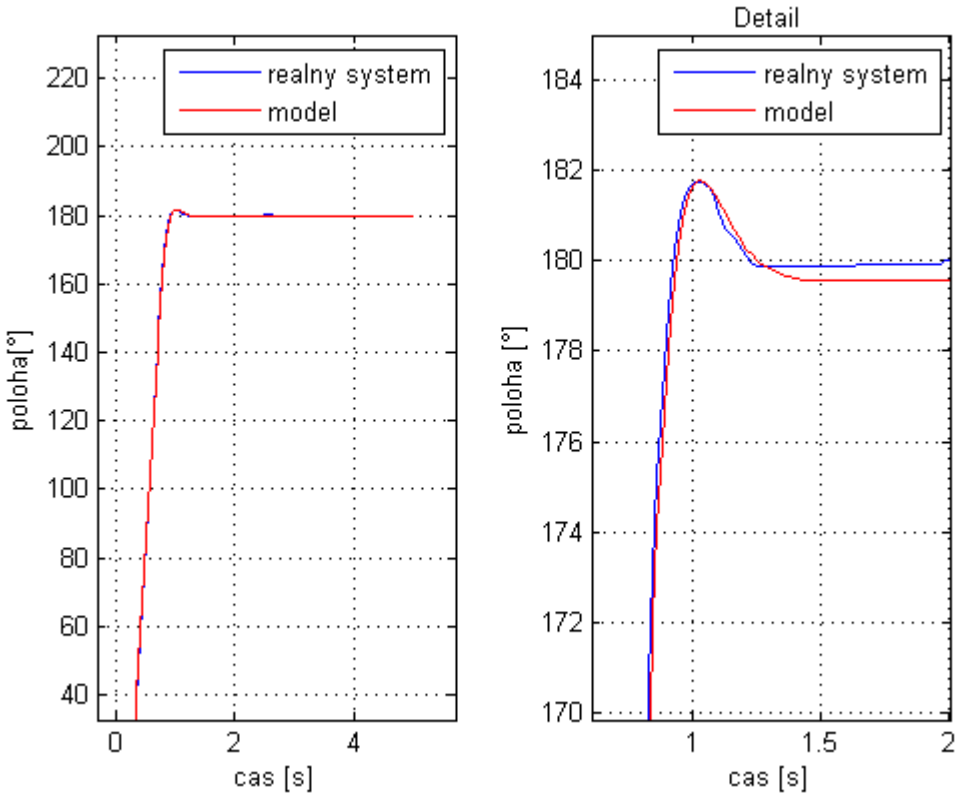
Regulátor P = 1, D = 3



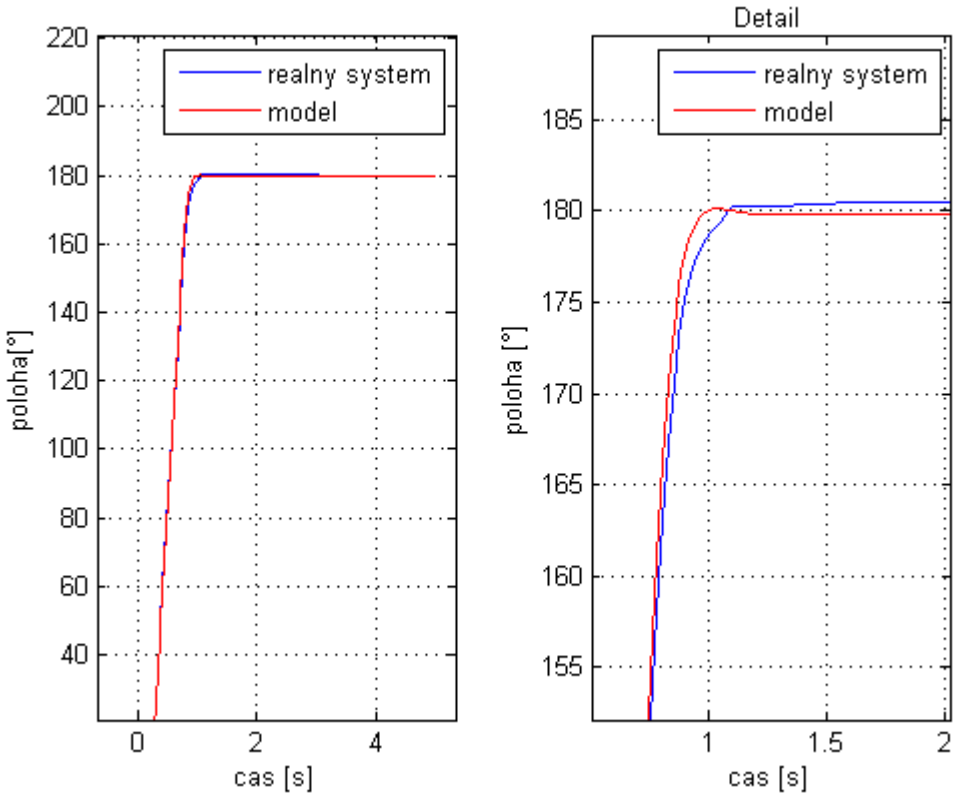
Regulátor P = 1, D = 5



Regulátor P = 1, D = 10



Regulátor P = 2, D = 4



Příloha 8 – Obsah přiloženého CD

Na přiloženém CD se nachází tyto soubory a adresáře:

/BP_Ota_Fejfar_2007.pdf	tato bakalářská práce ve formátu pdf
/Katalogove_listy	katalogové listy všech použitých elektronických součástek
/Programy	instalace WinAVR - podpůrné knihovny a kompilátor C pro mikroprocesory Atmel AVR. Freeware verze programu EAGLE 4.16r2 pro kreslení schémat a návrh desek plošných spojů. Program ORCAD_GAL, který slouží ke kompilování programů pro hradlová pole GAL.
/Zdrojove_kody	Zdrojový kód programu pro ATmega8515 a programu pro GAL22V10
/Dokumentace	avr-libc-manual-1.4.5.pdf – manuál knihoven a funkcí obsažených ve WinAVR. Dále jsem zde umístil dokumentaci C-kódu pro ATmega8515.