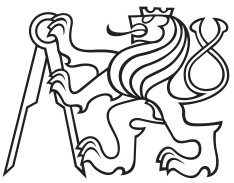
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídicí techniky

Řízení bezkartáčových motorů s deskou Raspberry Pi a Linuxem

Martin Prudek
Studiní program: Kybernetika a robotika.
Obor: Systémy a řízení.

Květen, 2015
Vedoucí práce: Ing. Pavel Píša, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Martin Prudek**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Řízení bezkartáčových motorů s deskou Raspberry Pi a Linuxem**

Pokyny pro vypracování:

Na platformě procesorové desky Raspberry Pi implementujte systém pro řízení bezkartáčových (BLDC/PMSM) motorů.

1. Pro komunikaci procesorového systému s výkonovým hardwarem realizovaným s využitím programovatelného obvodu (FPGA) vyberte vhodný protokol a periferii.
2. Pro vybraný způsob komunikace navrhněte ovladač na straně jádra Linux a obvodový návrh ve VHDL na straně FPGA.
3. Integrujte bloky pro snímání polohy, řízení výkonových stupňů a měření proudu do FPGA návrhu.
4. S využitím navržených periferií realizujte řízení bezkartáčového motoru.
5. Vyžaduje se podrobná technická dokumentace včetně přípravy podkladů pro prezentaci včetně videozáznamu.

Seznam odborné literatury:

[1] https://rt.wiki.kernel.org/index.php/Main_Page
[2] Radek Mečiar: Řízení motorů s deskou Raspberry Pi a Linuxem, bakalářská práce, ČVUT FEL 2014
[3] Dokumentace výkonového stupně Rpi-Mi-1, PiKRON 2014
[4] Martin Meloun: FPGA Based Robotic Motion Control System, diplomová práce, ČVUT FEL 2014
[5] Libero User's Guide, IGLOO Low Power Flash FPGAs DS, Microsemi 2012

Vedoucí: Ing. Pavel Píša, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 23. 1. 2015

Poděkování / Prohlášení

Na tomto místě bych chtěl především velmi poděkovat svému vedoucímu bakalářské práce, panu Ing. Pavlu Píšovi, PhD. za vstřícnost, ochotu a trpělivost. Bez jeho cenných rad a zkušeností by tato práce nemohla vniknout.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne

.....

Abstrakt / Abstract

Bakalářská práce se zabývá řízením bezkartáčových synchronních motorů (PMSM) s deskou Raspberry Pi a Linuxem. K tomuto účelu je použita rozšiřující jednotka integrující část pro výkonové řízení a programovatelné hradlové pole (FPGA).

V první části práce je popsána problematika PMS motorů, včetně jejich matematického popisu, konstrukce a možností řízení. Následuje seznámení s použitým technickým vybavením, Raspberry Pi, rozšiřující jednotkou a motorem.

Práce pokračuje popisem operačního systému GNU/Linux a jeho Real-Time modifikací.

Pro komunikaci mezi rozšiřující jednotkou a nadřazeným sytémem je použit protokol SPI. Jeho implementace v rámci FPGA obvodu a Raspberry Pi je věnována další kapitola.

Měření proudů protekajících fázemi vinutí je řešeno s využitím Hallova efektu. Takto získaná napětí jsou zpracována AD převodníkem. Ke čtení hodnot z AD převodníků je v FPGA obvodu navržen jednoduchý komunikační protokol.

Poslední část práce se zabývá různými možnostmi elektronické komutace. Implementována je komutace využívající pouze výstupu z Hallových sond a jednoduchá komutace realizující posun fází o 120°. V samotném závěru je uvedena možnost vektorového řízení.

Klíčová slova: synchronní bezkartáčový motor; bakalářská práce; FPGA; Raspberry Pi; Linux; Real-Time; řízení; PMSM.

This bachelor thesis is focused on the control of permanent magnet synchronous motors (PMSM) with a single-board computer Raspberry Pi and Linux. For this purpose an expansion unit which integrates power management and a field-programmable gate array (FPFA) is used.

The first part of thesis deals with theory of PMSM, including their construction, mathematical description and possibilities of control. The presentation of the hardware used, Raspberry Pi, expansion unit and motor takes place afterwards.

The thesis continues with a description of the operating system GNU/Linux and its Real-Time modification.

SPI protocol is used for communication between the expansion unit and its superior system. Its implementation in FPGA and Raspberry Pi is described in next chapter.

The solution, which takes advantage of Hall Effect, is used to measure currents in stator windings. Voltages that are obtained this way are then processed with A/D converter. A simple communication protocol to read values from A/D converter is implemented in FPGA .

The last part of the thesis examines multiple possibilities of electronic commutation. Commutation that utilizes only the output of Hall Effect sensors and simple commutation that carry out phase shift by 120° are implemented. The concept of vector control of PMSM is presented at the very end of the thesis.

Keywords: brushless; permanent magnet motor; bachelor thesis; FPGA; Raspberry Pi; Linux; Real-Time; control; PMSM.

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Elektrické motory jsou již dlouhou dobu nedílnou součástí moderní civilizace. Jejich řízení se stalo důležitou disciplínou, potřebnou v mnoha aplikacích. Postupem času se přitom stává výhodnějším i pro menší výkony použít složitější elektroniku než řešit komutaci mechanickými kartáči.

Synchronní bezkartáčový motor přitom nabízí ve srovnáním s konvenčním DC motorem několik výhod. Nedochází k takovému opotřebení mechanických částí, ani ke kolísání točivého momentu v průběhu otáčky. Zvyšuje se efektivita. Více o PMS motorech je uvedeno v kapitole 2.

Cílem práce je především výuka řízení a možnost experimentování. Pro tyto účely má smysl pro řízení použít kompletní systém typu Linuxového stroje i tam, kde se produkční provedení realizují malými MCU. Do takto rozsáhlého systému je možné snadno integrovat i další funkce právě například pro výuku nebo další rozšíření. Výhodou je také pohodlný vzdálený přístup přes SSH nebo HTTP, který malé MCU ve velké míře neumožňují.

Na trhu je dnes celá řada malých jednodeskových počítačů, které nedosahují kvalit potřebných pro průmyslové nasazení. Je ale možné je využít k návrhu řízení a komunikace s výkonovou částí. Takto vytvořená řešení jsou pak použitelná i při nasazení na profesionálnějších systémech.

Z široké nabídky takových zařízení, jako jsou BeagleBone, Raspberry Pi či Banana Pi [1]) bylo vybráno pro tuto práci Raspberry PI model B rev. 2.0.

Raspberry Pi (dále také „RPi“) je počítač realizovaný na jednom plošném spoji, velikosti kreditní karty, který od roku 2006 vyvíjí britská nadace *Raspberry Pi Foundation* [2]). Výhodou je možnost využití univerzálního operačního systému, například výrobcem připraveného sestavení operačního systému GNU/Linux na bázi distribuce Debian. Na takto upravený systém, pojmenovaný Raspbian, je pak možné aplikovat Real-Time modifikaci, která zajistí splnění požadavků řídicí aplikace. Více k Raspberry Pi a RT modifikaci Linuxu uvedu v sekci 3.2 a 3.3.

U tohoto počítače již byla prostudována možnost řízení stejnosměrného motoru s využitím softwarového zpracování impulzů inkrementálního rotačního senzoru polohy [1]. Pro použitý, nepříliš výkonný, hardware však softwarové dekódování impulzů představuje pro vyšší rychlosti otáčení motoru takovou zátěž, kterou již není schopen operační systém bezchybně obsloužit. U RPi tak docházelo ke kritickým časovým prodlevám při zpracovávání přijatých pulsů, čímž se ztrácela informace o poloze. Takové chování nastávalo již od frekvence pulsů 14kHz, což při 500 pulsech na jednu otočku dělá 2100ot/min. Tedy ve chvíli, kdy perioda příchodu pulsů byla srovnatelná s dobou odezvy systému na tento druh události.

Výhodným řeším tohoto problému se stává přesunutí zpracování pulsů IRC do samostatné periferie a zjištěnou polohu motoru posílat v binární podobě. RPi tedy zpracovává

[1]) Root.cz, Srovnání: Raspberry Pi a jeho největší konkurenti `http://www.root.cz/clanky/srovnani-raspberry-pi-a-jeho-nejvetsi-konkurenti/`

[2]) web Raspberry Pi Foundation`https://www.raspberrypi.org/`

jen jednu řídicí smyčku o neměnné frekvenci. Z hlediska výpočetního výkonu tedy již otáčky motoru nehrají roli. Protože RPi nedisponuje periferií vhodnou pro tento účel, byla pro tuto práci použita rozšiřující deska vyvinutá firmou *PiKRON* [1]). Tato deska umožňuje návrh potřebné logiky v FPGA obvodu. Podrobnosti v sekci 3.4.

Pro propojení rozšiřující desky a nadřazeného systému (RPi) bylo třeba vybrat jednoduchý komunikační protokol. Požadavkem byla možnost obousměrné komunikace a dostatečná rychlost, která umožní pro požadovanou frekvenci řídicí smyčky přenést dostatečný objem dat. Bylo také nutné, aby použitý protokol byl, pokud možno, jednoduše realizovatelný v FPGA obvodu a zároveň ho bylo možné pohodlně zpracovat v jádře operačního systému na RPi. Více o výběru a implementaci komunikačního protokolu v sekci 4.3.

Bylo otestováno jednoduché řízení podle kombinace Hallových senzorů popsané v sekci 4.8.1. Napěťové řízení se sinusovým průběhem v sekci 4.8.2 a vektorové řízení polohy a rychlosti, sekce 4.8.3. Průběhy řízení, včetně změřených hodnot proudů protékajících jednotlivými fázemi, jsou uvedeny v kapitole 5.

[1]) PiKRON `http://www.pikron.com/`

Kapitola 2
PMS motory

PMS (Permanent Magnet Synchronous) motory jsou díky vysoké efektivitě a robustní kontrukci bez kartáčů vhodnou volbou v mnoha řídicích aplikacích[2]. Zvláště pak v robotice a všude tak, kde je zapotřebí řízení polohy a rychlosti.

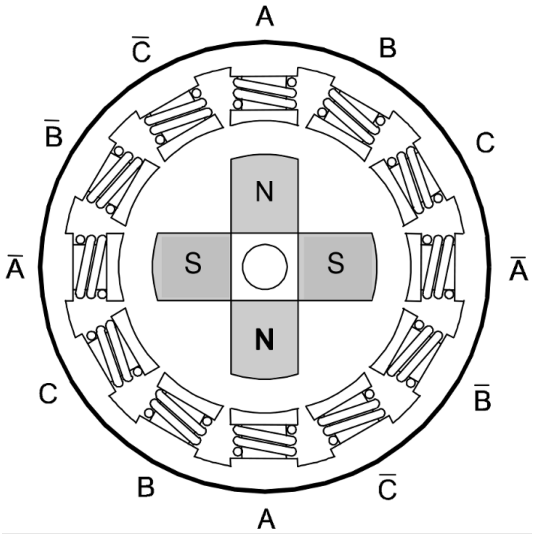
Na rozdíl od kartáčových motorů je komutace PMS motorů řízena elektronicky, což přináší vyšší požadavky na řídicí hardware. Ve chvíli, kdy dnes většina aplikací vyžaduje elektronické řízení jak rychlosti, tak točivého momentu, nepředstavuje ale řídicí elektronika zátěž navíc. [3]. Odměnou je naopak výšší výkon v poměru k váze, stejně tak točivý moment v poměru k příkonu. Výhodou jsou také nížší hlučnost a delší životnost, protože nedochází k opotřebení kartáčů a mechanických částí komutátoru. [4] Elektronická komutace bývá implementována v procesorovém systému, či speciálním obvodu (FPGA / ASIC)[5].

Třífázové synchronní motory s permanentním magnetem se často využívají kromě PMS varianty se sinusovým průběhem zpětného elektromotorického napětí (BEMF) také v BLDC variantě s lichoběžníkovým (trapezoidal) průběhem BEMF. Přičemž výhodou PMSM je konstatní točivý moment v celém rozsahu otáčení, zatímco BLCD motor je snadněji řiditelný a dnes se již využívá převážně z historických důvodů. [3]

2.1 Konstrukce

Základ konstrukce PMS Motoru tvoří rotor s permanentním magnetem a stator, kterým prochází vinutí jednotlivých fází.

Počet vinutí pak závisí na počtu pólů permanentního magnetu umístěného ve statoru. Obvyklá varianta jsou dva páry pólů (polpáry) permanentního magnetu, viz. obrázek 2.1. Celý návrh je přitom optimalizován pro buzení sinusovým průběhem, stejně tak zpětné elktromotorické napětí vykazuje sinusový průběh. [6] [7].



Obrázek 2.1. PMS Motor se dvěma pólovými dvojicemi

2.2 Matematický popis

Pro matematický popis dějů v PMS motorech a vláště pak pro implementaci plného momentového řízení jsou vhodným nástrojem Clarkova a Parkova tranformace. Ty jsou využity k vytvoření modelu použitého třífázového motoru (točivého elektrického stroje) který je uveden dále v této sekci.

2.2.1 Clarkova transformace

Clarkova transformace umožňuje zobrazit proud protékající jednotlivými fázemi jako jeden vektor v komplexní rovině.

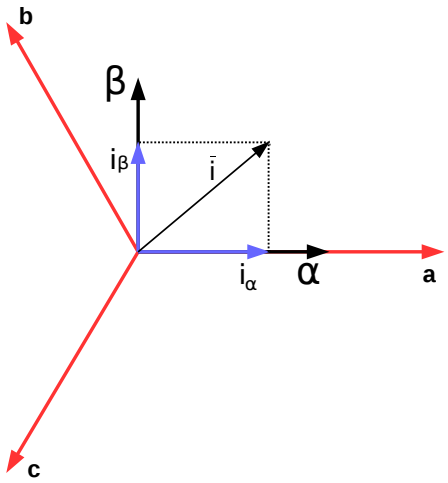
Uvažujme i_a, i_b a i_c proudy procházející vinutím statoru, platí:

$$i_a + i_b + i_c = 0 \tag{1}$$

Toto může být vyjádřeno jako vektor v komplexní rovině, potom:

$$\bar{i} = i_a + \hat{a} i_b + \hat{a}^2 i_c \tag{2}$$

kde $\hat{a}$ a $\hat{a}^2$ jsou operátory posouvající fáze o $\delta = 120° = \frac{2\pi}{3} rad$. Operátory mají podobu: $\hat{a} = e^{j\delta}$ a $\hat{a}^2 = e^{2j\delta}$.



Obrázek 2.2. Komplexní vyjádření vektoru proudu vinutím statoru (Clarkova transformace) v $\alpha\beta$ souřadné soustavě. α je reálná a β imaginární osa.

Po dosazení za $\hat{a}$ je možné rovnici (2) přepsat:

$$\bar{i} = i_a + i_b e^{j\delta} + i_c e^{2j\delta} \tag{3}$$

Použijeme-li Eulerův vzorec $e^{j\phi} = \cos\phi + j\sin\phi$ a Moivreovu větu $(\cos x + j\sin x)^n = \cos(nx) + j\sin(nx)$, dostáváme ekvivalentní vyjádření pomocí goniometrických funkcí:

$$\bar{i} = i_a + i_b(\cos\delta + j\sin\delta) + i_c(\cos 2\delta + j\sin 2\delta) \tag{4}$$

$\bar{i}$ lze interpretovat jako součet jeho reálné a imaginární složky v soustavě $\alpha\beta$:

$$\bar{i} = i_\alpha + j i_\beta \tag{5}$$

Z rovnice (4) je pak možné vyjádřit reálnou a imaginární složku $\bar{i}$ a zapsat maticové vyjádření Clarkovy transformace:

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} 1 & \cos\delta & \cos 2\delta \\ 0 & \sin\delta & \sin 2\delta \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \tag{6}$$

Po dosazení za $\delta = 120°$ do rovnice (6) dostáváme:

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \tag{7}$$

Pro popis PMS motorů je uvažován ideálně symetrický motor se sinusoidně rozloženým vinutím. Pro takovou idealizaci uvažujeme napětí na vinutích u_a, u_b a u_c následující:

$$u_a = R_s i_a + \frac{d}{dt}\psi_a \tag{8}$$

$$u_b = R_s i_b + \frac{d}{dt}\psi_b \tag{9}$$

$$u_c = R_s i_c + \frac{d}{dt}\psi_c \tag{10}$$

kde ψ_a,ψ_b a ψ_c jsou magnetické indukční toky vyvolané proudy odpovídajících vinutí. Vyjádření složek napětí v $\alpha\beta$ souřadné soustavě odpovídá:

$$u_\alpha = R_s i_\alpha + \frac{d}{dt}\psi_\alpha \tag{11}$$

$$u_\beta = R_s i_\beta + \frac{d}{dt}\psi_\beta \tag{12}$$

Přitom složky magnetického indukčního toku statoru budou:

$$\psi_\alpha = L_\alpha i_\alpha + \psi_M cos\theta \tag{13}$$

$$\psi_\beta = L_\beta i_\beta + \psi_M sin\theta \tag{14}$$

kde θ je úhlová pozice rotoru a ψ_M je magnetický indukční tok rotoru. L_α a L_β jsou složky vzájemné indukčnosti rotor-stator.

Úhlové zrychlení takového motoru s momentem setrvačnosti J o zátěži M_L s p póly připadajícími na každou fázi můžeme vyjádřit jako:

$$\frac{d\omega}{dt} = \frac{1}{J}[\frac{3}{2}p(\psi_\alpha i_\beta - \psi_\beta i_\alpha) - M_L] \tag{15}$$

Rovnice (8) az (15) představují model PMS motoru v souřadné soustavě $\alpha\beta$, která je fixována statorem [8].

2.2.2 Parkova transformace

Kromě soustavy spojené se statorem bývá někdy výhodné vyjádřit proudy a další veličiny v soustavě spojené s rotorem. Pro dosažení maximálních momentových účinků, je totiž nutné, aby vektor magnetické indukce, magnetického pole vyvolávaného proudy protékající vinutím statoru, svíral pravý úhel s vektorem mg. indukce mg. pole permanentního magnetu rotoru.

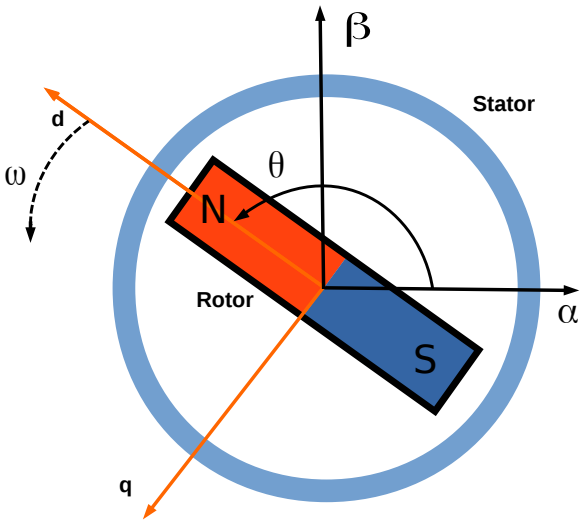
Pro tyto účely je tedy vhodné veličiny fixovat k osám spojeným právě s rotorem. Osy jsou nazývané d a q a jejich poloha je ilustrována na obrázku 2.3.

V soustavě spojené se statorem byly veličiny s úspěchem vyjádřeny pomocí Clarkovy transformace. K jejich vyjádření v soustavě rotující společně s rotorem, je silným nástrojem transformace Parkova (dq0 transformace).

Transformace je realizována podobně, jako v předchozím případě, pomocí operátoru otočení. Symbol $\bar{i}_r$ představuje komplexní vektor proudu vztažený k osám d a q.

$$\bar{i}_r = \bar{i}e^{-j\theta} \tag{16}$$

Kde θ je úhlová pozice rotoru, viz obrázek 2.3.



Obrázek 2.3. Osy d(direct) a q(quadrature) jsou voleny vzhledem k rotoru.

Vztah (16) je možné podobně jako rovnicí (2) přepsat pomocí Eulerova vzorce do následujícího tvaru:

$$\bar{i}_r = \bar{i}(\cos\theta - j\sin\theta) \tag{17}$$

Složky komplexního vektoru $\bar{i}_r$ v *dq0* sostavě souřadné pak můžeme po dosazení z výrazu (4) zapsat jako:

$$\bar{i}_r = i_d + ji_q = (i_\alpha + ji_\beta)(\cos\theta - j\sin\theta) \tag{18}$$

Pomocí rovnice (17) je možné vyjádřit také maticový tvar Parkovy transformace:

$$\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \tag{19}$$

Složky napětí v *dqo* souřadném systému jsou následující:

$$u_d = R_s i_d + \frac{d}{dt}\psi_d - \omega\psi_q \tag{20}$$

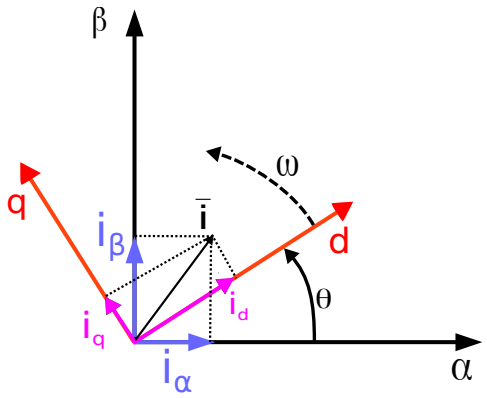
$$u_q = R_s i_q + \frac{d}{dt}\psi_q - \omega\psi_d \tag{21}$$

Symbol ω je úhlová rychlost rotoru. ψ_d a ψ_q jsou složky magnetického indukčního toku statoru vyjádřeného v *dq0* souřadném systému.

Jednotlivé složky je možné vyjádřit:

$$\psi_d = L_d i_d + \psi_M \tag{22}$$

$$\psi_q = L_q i_q \tag{23}$$



Obrázek 2.4. Parkova transformace

Úhlové zrychlení motoru s momentem setrvačnosti J o zátěži M_L s p póly připadajícími na každou fázi můžeme v v *dq0* souřadném systému vyjádřit jako[8]:

$$\frac{d\omega}{dt} = \frac{1}{J}[\frac{3}{2}p(\psi_d i_q - \psi_q i_d) - M_L] \tag{24}$$

Po dosazení za ψ_d a ψ_q z rovnic (22) a (23) do (24) dostáváme:

$$\frac{d\omega}{dt} = \frac{1}{J}[\frac{3}{2}p(\psi_M i_q + (L_d - L_q) i_q i_d) - M_L] \tag{25}$$

Točivý moment M motoru tak vypadá následovně:

$$M = \frac{3}{2}p(\psi_M i_q + (L_d - L_q) i_q i_d) \tag{26}$$

Za předpokladu $i_d = 0$, pak dojde ke zjednodušení:

$$M = \frac{3}{2}p\psi_M i_q \tag{27}$$

Z rovnice (27) vyplývá, že točivý moment motoru je možné libovoně řídit, jen nastavením hodnoty i_q [9] [10]. Procesem, jak z i_q vypočítat proudy jednotlivými fázemi se zabývá sekce 2.3. Implementace možného řešení je v sekci 4.8.3.

2.3 Řízení

PMS Motory se vyznačují plynulou rychlostí v celém rozsahu otáčení a schopností plně řídit točivý moment i při nulové rychlosti. K tomu se využívaji techniky vektorového řízení.

S jejich pomocí je možné rozložit proud protékající vinutím statoru na složky generující magnetické pole a točivý moment. Tyto složky pak můžeme řídit na sobě nezávisle a přiblížit se tak v jednoduchosti řízení standartního kartáčového DC motoru [8].

V této práci je implementovávo poziční řízení s možností rozšíření na plné momentové řízení. Z rovnice (27) vyplývá, že pro poziční a momentové řízení je nutné modifikovat pouze složku i_q komplexního vektoru proudu $\bar{i}$.

Matematickým nástrojem, jak přepočítat i_q na statorové proudy i_a, i_b a i_c jsou inverzní Parkova a Clarkova transformace, kterým je věnován zbytek této sekce. Implementace možného řešení je popsána v sekci 4.8.3. Pro názornost jsou zde obě transformace uváděny v maticovém tvaru.

2.3.1 Inverzní Parkova transformace

Maticový tvar dopředné Parkovy transformace je uveden v rovnici (19). Transformační matice je regulární a je tedy možné jednoduše vytvořit její inverzi:

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} \tag{28}$$

V případě, že je úhel odečítán opačným směrem, než je v matematice obvyklé (obvyklý směr je proti chodu hodinových ručiček), je možné do rovnic dosadit $\theta = -\vartheta$.

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} \cos\vartheta & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} \tag{29}$$

Tento případ se týká i řešení použitého v této práci.

2.3.2 Inverzní Clarkova transformace

Nyní jsou již k dispozici hodnoty i_α a i_β, k jejich přepočtení na jednotlivé fáze slouží inverzní Clarkova transformace.

Vytvořit inverzi k transformační matici v rovnici (6) není možné. Je tedy nutné rozšířit transformaci o osu γ. Proud v této ose označme i_γ a definujme $i_\gamma = z(i_a + i_b + i_c)$.

$$\begin{bmatrix} i_\alpha \\ i_\beta \\ i_\gamma \end{bmatrix} = k \begin{bmatrix} 1 & \cos\delta & \cos 2\delta \\ 0 & \sin\delta & \sin 2\delta \\ z & z & z \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \tag{30}$$

Tímto způsobem byla transformační matice rozšířenana velikost 3x3. Matice je pro $z \neq 0$ a $k \neq 0$ regulární, je tak možné vytvořit matici inverzní a celé inverzní zobrazení.

$$\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \frac{2}{3k} \begin{bmatrix} 1 & 0 & \frac{2}{z} \\ \cos\delta & \sin\delta & \frac{2}{z} \\ \cos 2\delta & \sin 2\delta & \frac{2}{z} \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \\ i_\gamma \end{bmatrix} \tag{31}$$

Pokud je transformační konstanta rovna $k = \frac{2}{3}$, nedochází k zesílení požadovaných proudů. Konstanta z bývá z estetických důvodů volena $z = \frac{1}{2}$. Po dosazení do takto zvolených konstant vyčíslení $\delta = 120° = \frac{2\pi}{3} rad$, bude rovnice vypadat:

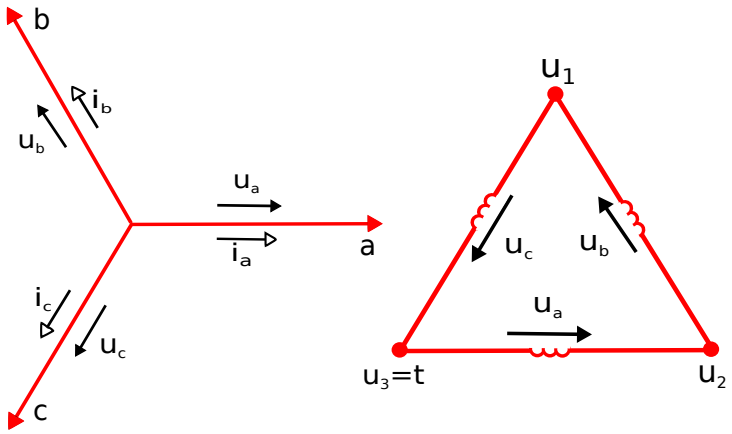
$$\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & \frac{-\sqrt{3}}{2} & 1 \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \\ i_\gamma \end{bmatrix} \tag{32}$$

Zde je nutné připomenout, že z rovnice (1) vyplývá: $i_\gamma = z(i_a + i_b + i_c) = 0$.

2.3.3 Transformace napětí při delta zapojení

S využitím inverzní Clarkovy transformace lze vypočítat jednotlivé statorové proudy. Z rovnic (8) až (10) pak vyplývá, že proudy je možné řídit napětím na vinutí.

Nyní je tedy třeba napětí u_a, u_b a u_c, které představují akční zásahy pro řízení proudů, přepočítat na napětí u_1, u_2 a u_3, která jsou již přímo úměrná šířce plnění PWM. Pro výpočet je uvažováno zapojení typu „delta“, které je použito i při konstrukci použitého motoru, viz. sekce 3.1. Schématicky problém postihuje obrázek 2.5.



Obrázek 2.5. Delta zapojení

Pro vyjádření napětí u_a, u_b a u_c je možné využít následující vztahy:

$$u_a = u_3 - u_2 \tag{33}$$
$$u_b = u_2 - u_1 \tag{34}$$
$$u_c = u_1 - u_3 \tag{35}$$

Maticový zápis:

$$\begin{bmatrix} u_a \\ u_b \\ u_c \end{bmatrix} = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{36}$$

Soustava je přeurčená, matice soustavy je tedy singulární a není možné vytvořit inverzní matici. Při parametrizaci jednoho z napětí, vyberme $u_3 = t$, je ale možné vyjádřit u_1 až u_3, tak, aby jejich rozdíly (napětí u_a až u_c) zůstaly zachovány:

$$u_1 = t + u_c \tag{37}$$
$$u_2 = t + u_c + u_b \tag{38}$$
$$u_3 = t \tag{39}$$

Parametr t je volen libovolně. Implemetací se zabývá sekce 4.8.3.

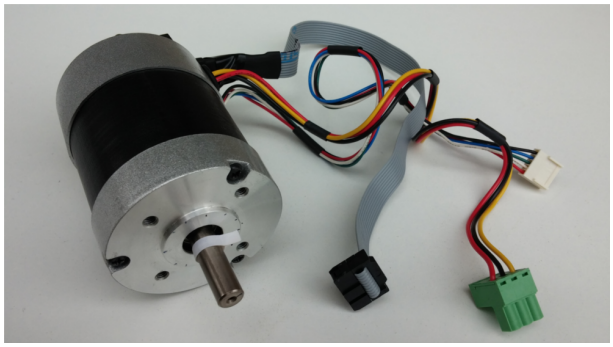
Kapitola 3
Popis hardware

Minulá kapitola shrnula teorii týkající se konstrukce bezkartáčových motorů a jejich řízení. Nyní bude následovat bude popis konkrétního technického vybavení použitého pro tuto práci.

Budou tak popsány parametry použitého motoru a Raspberry Pi. Prostor bude věnován problematice OS Linux a jeho Real-Time modifikaci. Následně dojde na téma FPGA obvod a jeho naprogramování. Kapitola je ukončena sekcí o rozšiřující jednotce, která integruje všechny tyto komponenty do jedoho řešení.

3.1 Použitý motor

Z široké nabídky bezkartáčových PMS motorů byl vybrán model BLWR233D-36V-4000 od společnosti Aneheim Automation [1]) viz. Obrázek 3.1. Jedná se o motor s výkonem 92W, pracující při maximálním napětí 36V. Vinutí statoru tvoří 6 polpárů. Vinutí každé z fází tedy tvoří právě dvě pólové dvojice. Analogicky má stator motoru právě dva polpáry, viz. Obrázek 2.1.



Obrázek 3.1. Použitý motor

V motoru jsou integrovány Hallovy sondy, které snímají absolutní polohu motoru ve 12 úsecích mechanického cyklu. Při přechodu mezi dvěma úseky se mění právě jeden z výstupních signálů těchto senzorů. V místech přechodu je tedy možné určit pozici motoru relativně přesně.

Do motoru je přivedeno celkem 8 vodičů ve dvou svazcích. Ve svazku o třech vodičích jsou přivedeny fáze, v ostatních je pak napájení a výstup Hallových senzorů, viz. Tabulka 3.2. V Tabulce 3.1 jsou pak uvedeny technické parametry motoru.

[1]) Aneheim Automation BLWR23 `http://www.anaheimautomation.com/products/brushless/brushless-motor-item.php?sID=148&pt=i&tID=96&cID=22`

Veličina	Uvedené hodnoty / jednotky	Metrický systém
Max. napětí	36 V	36 V
Max. výkon	92 W	92 W
Max. točivý moment	31.2 oz-in	0.0219 Nm
Max. otáčky	4000 rpm	4000 rpm
Torque constant	8.5 oz-in/A	0.06 Nm/A
BEMF konst.	4.45 V/krpm	4.45 V/krpm
Odpor mezi fázemi (line-to-line)	0.64 Ω	0.64 Ω
Vzájemná indukčnost fází	2.1 mH	2.1 mH
Moment setrvačnosti rotoru	0.00106 oz-in-sec^2	$7.485 * 10^{-6}$ kg$*m^2$
Délka	2.9 in	73.7 mm
Váha	1.65 lbs	0.75 kg

Tabulka 3.1. Parametry použitého motoru

Svazek	Barva	Funkce
1	Žlutá	Fáze A
1	Červená	Fáze B
1	Černá	Fáze C
2	Červená	Napájení hallů
2	Modrá	Hall senzor A
2	Zelená	Hall senzor B
2	Bílá	Hall senzor C
2	Černá	Uzemění hallů

Tabulka 3.2. Vodiče přivedené do motoru

K motoru je též z jedné strany namontován inkrementální rotační senzor (dále jen IRC), který měří přírůstky polohy motoru vůči pozici při startu řízení. Jeho výhodou je vysoká přesnost. Zatímco Hallovy sondy dokáží rozlišit jen mezi 12 polohami na jednu mechanickou otáčku, použitý IRC rozezná 2000 samostatných pozic (běžné hodnoty rozlišení tohoto typu senzorů se pohybují v rozmezí 16 až 10000 bodů na otáčku). K IRC vedou vodiče napájení a vodiče přenášející informaci o poloze. Dva z těchto vodičů nesou signály, z jejichž aktuálních stavů a jejich změn je možné vypočítat rychlost pohybu motoru a jeho směr. Třetí signál pak vyšle puls vždy jen v jedné pozici za jednu mechanickou otáčku motoru. Poskytuje tak referenční, absolutní polohu.

3.2 Raspberry Pi

Raspberry Pi je jednodeskový počítač založený na rodině architektury ARM, který se v současnosti dodává v několika variantách.

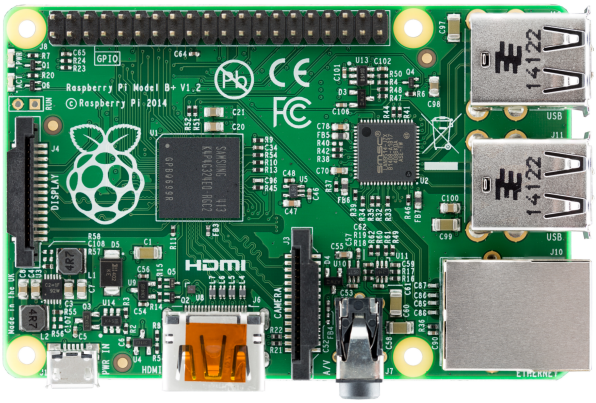
Základem první verze tohoto minipočítače je SoC BCM2835 [1]), který obsahuje centrální procesor ARM1176JZF-S s taktem 700 MHz, grafický procesor VideoCore IV a 256 MB nebo 512 MB paměti RAM. Neumožňuje však přímé připojení pevného disku pomocí SATA a neobsahuje ani jinou trvalou integrovanou velkokapacitní paměť (MTD). Operační systém a data, která mají být uchována i po restartu zařízení je tak třeba uložit na SD kartu, jejíž slot je k dispozici.

Procesorová jednotka typu ARM11 využívá poměrně zastaralou architekturu ARMv6 se statší verzí jednotky pro výpočty v plovoucí řádové čárce VFPv2 [11] [2]),která již není

[1]) anthill inside

[2]) ARM11 Online Technical Reference Manual `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Cegdejjh.html`

podporovaná sestavením systému Debian pro moderní zařízení s procesorem ARM. Proto je nutná rekompilace distribuce systému Debian/Raspbian. Oficiální port této distribuce ARMhf totiž vyžaduje alespoň architektutu ARMv7 s koprocesorem pro výpočty v plovoucí řádové čárce nejméně ve verzi VFPv3-D16. [1])



Obrázek 3.2. Raspberry Pi verze 1 model B+

Druhá verze Raspberry Pi přinesla zvýšení výpočetního výkonu s růstem taktu procesoru na 900Mhz a využitím čtyř výpočetních jader. To vše pod modernější architekturou ARMv7-A s procesorem ARM Cortex-A7 (podpora VFPv4 [12] [2])) v čipu BCM2836. Tato verze počítače je tak se současnými distribucemi plně kompatibilní. Kromě vyššího výpočetního výkonu došlo i k nárustu hlavní paměti na 1GB. Momentálně je k dispozici jen v modelu B, který navazuje na model B+ verze 1. Na rozdíly jednotlivých variant odkazuje tabulka 3.3

Nevýhodou použitých SoC v obou verzích je chybějící integrovaná podpora rozhraní ethernet. Pro připojení do sítě toho typu je použit na desku integrovaný převodník USB-Ethernet.

Model	A	A+	B	B+	Bv2
Počet pinů	26	40	26	40	40
RAM paměť [MB]	256	256	256/512	512	1024
USB porty	1	1	2	4	4
RJ45	Ne	Ne	Ano	Ano	Ano
Slot na kartu	SDHC	MicroSD	SD	MicroSD	MicroSD
Příkon [W]	1.5	1.0	3.5	3	4
Takt CPU [MHz]	700	700	700	700	900
Jádra CPU [MHz]	1	1	1	1	4
CPU Arch [W]	ARMv6	ARMv6	ARMv6	ARMv6	ARMv7-A

Tabulka 3.3. Seznam modelů Raspberry Pi

3.3 Jádro Linux a jeho použitelnost pro RT aplikace

Linux je víceuživatelský, víceúhlový operační systém založený na stejnojmenném jádře, které vyvinul v roce 1991 Linus Thorvadls. Po aplikaci úprav pro zaručení kontrolovaného času odezvy na vnější události (RT-rozšíření) se stává zajímavou volbou i pro některé řídicí aplikace.

[1]) Debian, List of official ports `https://www.debian.org/ports/`
[2]) ARM Cortex-A7 Online Technical Reference Manual `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.cortexa.cortexa7/index.html`

Systém, který byl původně myšlen spíše jako koníček se postupem času vyvinul z pouhého emulátoru terminálu v jeden z nejpoužívanějších na světě. Dnes mají Linuxové distribuce 97% zastoupení mezi největšími superpočítači [1]) a například systém Android, s Linuxovým jádrem, běží na 47% všech smartphonů a tabletů [2]). Mohutný rozmach systému byl zapříčiněn mimo jiné zveřejněním jeho zdrojového kódu pod svobodnou licencí. To umožnilo na jeho vývoji pracovat tisícům dobrovolníků a mnoha korporacím, mezi které patří například Red Head, Intel, Samsung či Google [3]).

Pro široké možnosti svého využití a jednoduché vzdálené správy si systém brzy získal oblibu mezi vyvojáři. Doménou řídicích aplikací však dále zůstavaly menší systémy. Linux se tak stával součástí jen větších celků, kde za odezvu k kritických částech byl zodpovědný RT OS a Linux plnil spíše administrativní funkce. Vývoj plikací pro menší RT OS však přinaší mnoho komplikací navíc, přikladem je spíše minimalistické API a často nedostatečná podpora některých komunikačních protokolů. Vznikl tak projekt, jehož cílem bylo upravit linuxové jadro způsobem, který umožní včasné přeplánování a zajistí tak splnění časových požadavků vyžadovaných řízením.

Projekt KURT (Kansas University Real Time) se stal průkopníkem, když využil podporu Linuxu pro víceprocesorové systémy (SMP) a rozšířil možnost souběhu procesů při zpracování systémových volání v jádře, což jádro dříve v plné míře nedovolovalo. Serializace takových úseků, ve kterých se v danou chvíli směl nacházet jen jeden procesor je pak řešena zámky s aktivním čekáním (spin-lock). Ostatní úseky kódu s nutností vyloučení vzájemného souběhu namísto spin-locků začaly využívat RT-mutex. Další snaha byla minimalizovat či přepracovat části kódu, které neumožňují preempci, jako například obslužné rutiny přerušení. Důležitá je implementace dědění priorit. [13] [14]

Vzhledem k rozsáhlosti zdrojového kódu systému Linux není možné analyticky spočítat veškerá zpoždění a vyhodnotit tak maximální latenci systému a časy potřebné k přeplánování. Podobný výpočet by navíc nebyl možný při použití rozmanitých vyrovnávacích pamětí a víceprocesorových systémů. Pro mnoho aplikací je však dostačující znát průběhy zpoždění měřeného na zatíženém systému v delším časovém období - řádově v měsících. Těmito testy a také dalším vývojem RT vlastností Linuxu se zabývá laboratoř OSADL [4]).

Jedna z posledních verzí RT-varianty jádra (3.18.7-rt2), tak po měsících testování reakcí na vnější události nevykázala byť jediné zpoždění přesahující 100μs. Přičemž testy probíhaly na výkonném HW s architekturou x86. Pro některé další architektury, například ARM pak zpožnějí pro vybrané modely nepřesahuje 200μs. [13].

3.4 FPGA obvod

FPGA obvody (Field Programmable Gate Array - Programovatelná hradlová pole) jsou speciální číslicové integrované obvody obsahující různě složité programovatelné bloky, dále násobičky či různé druhy pamětí. Tyto bloky jsou propojené konfigurovatelnou maticí spojů. FPGA obvody se odlišují od naprosté většiny integrovaných obvodů možností přeprogramování - při změně požadavků lze jednoduše náhrát novou konfiguraci.

[1]) Linux dominates supercomputers `http://www.zdnet.com/article/linux-dominates-supercomputers-as-never-before`

[2]) Mobile/Tablet Operating System Market Share `http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8&qptimeframe=M&qpcustomd=1`

[3]) Who actually develops Linux? `http://www.extremetech.com/computing/175919-who-actually-develops-linux-the-answer-might-surprise-you`

[4]) OSADL `http://www.osadl.org/`

FPGA se v současnosti využívají v mnoha aplikacích. Těží tak svých vlastností, mezi které patří poměrně snadný návrh v některém z HDL jazyků či grafických nástrojů. Flexibilita, neustále klesající cena ale i zmenšující se spotřeba elektrické energie patří mezi další pozitiva. Jsou tak zajímavou alternativou pro návrh a výrobu integrovaného obvodu na míru, jehož zavedení do výroby bývá velmi nákladné s návratností při výrobě miliónů kusů. Další běžnou oblastí použití je prototypování složitějších zařízení či tvorba periferií pro procesorové jednotky - tzv. „Glue logic“. Složitější FPGA obvody je dokonce možné použít k implementaci procesoru.[15] [16] [17]

Pro tuto práci byl vybrán FPGA obvod od společnosti *Microsemi Corporation* [1]) z rodiny IGLOO. Konkrétně čip AGL125 v pouzdře VQ100. Tato jednotka obsahuje 125 tisíc hradel a nabízí mimo jiné 36 kilobitů RAM. Podporuje In System Programming (ISP) a lze naprogramovat prostřednictvím rozhraní JTAG [18].

Na rozdíl od větších a výkonnějších obvodů Xilinx a Altera zde dochází k uložení konfigurace přímo do obvodu a není ji tak třeba nahrávat z externí FLASH či z CPU. Taktéž poměr ceny a kvality produktu je příznivý. Pro inicializaci hodnot v obvodu při zapnutí napájení pak může být využit externí signál. Výhodným řešením může být i využití resetu z bloku fázového závěsu (PLL), které je implementováno i v této práci. Bez implementace tohoto mechanismu je ale třeba brát na zřetel hrozbu inicicializace na nevhodné hodnoty. Příkladem jsou signály pro ovládání polovičních H-můstků, které, při inicializaci na logickou jedničku mohou zapříčinit přehřátí a zničení motoru.

Návrh byl vytvořen v HDL jazyce VHDL. Jazyk VHLD slouží jak k popisu a následné syntéze obvodu v programovatelném hradlovém poli, tak k simulacím. Používá se také pro popis obvodů, které se podle návrhu ve VHDL později přímo vyrobí. Standardizován byl v r. 1987. Jako jazyk vyvinutý z přísně typového jazyka Ada si pak ponechává některé jeho vlastnosti, které se s výhodou využijí při popisu hardware.

Mezi tyto výhody patří zabudovaná možnost souběžného výkonávání kódu. Tato funkce se totiž plně využije právě při navrhování hardware, kde je souběžné vykonávání elementární záležitostí. Vznikají tak bloky *Process*, které po příchodu spouštěcího signálu pracují zcela paralelně.

3.5 Syntéza VHDL kódu a programování FPGA

Pro syntézu kódu je používán balíček aplikací *Libero* (viz. Manuál [19]) vyvíjený výrobcem obvodu. Program je možné používat i zdrama. Jedinou podmínkou je registrace a získání „FREE Licence“. Před každým spuštěním aplikace pak musí běžet licenční daemon. Jeho spouštěcí příkaz je následující:

```
/opt/microsemi/licensing_daemon/lmgrd ...
-c /opt/microsemi/license/license.dat ...
-l /home/user/license.log
```

Samotná syntéza je pak řízena skripty v jazyce TCL. Zvlášť je použit skript pro analýzu a syntézu logiky a skript pro vlastní umístění logiky do jednotlivých elementů obvodu. Oba kroky syntézy jsou pak společně volané shellovým skriptem `synthetize-agl.sh`. Všechny soubory VHDL, včetně nejvyšší entity, musí být uvedené v souboru `syn.tcl` a při syntéze musí být umístěné v spolu s shellovým skriptem v jednom adresáři.

Pro naprogramování FPGA obvodu je využívána open-source aplikace *UrJTAG* hostovaná na serveru *Sourceforge* [2]). Pro běh aplikace je nutné mít nainstalovány knihovny

[1]) Microsemi Corporation `http://www.microsemi.com/`
[2]) UrJTAG, Sourceforge.net `http://sourceforge.net/projects/urjtag/`

`libusb` a `libftdi`. Poté je možné FPGA obvod naprogramovat spuštěním skriptu `program-agl.sh` ze stejného adresáře, ve kterém proběhla syntéza.

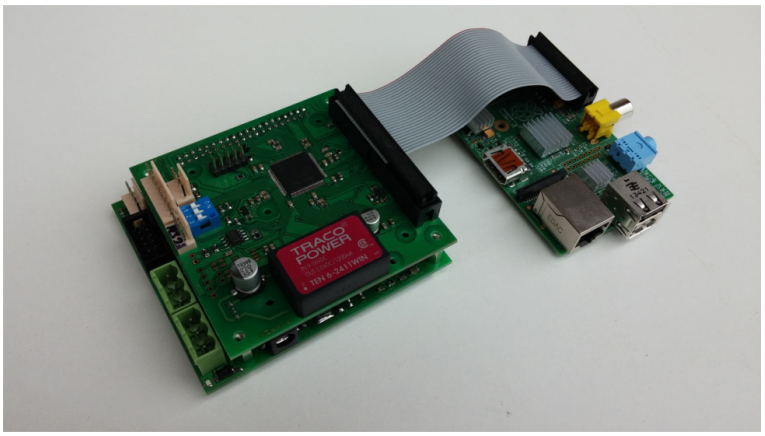
3.6 Rozšiřující jednotka

Rozšiřující jednotka k RPi je tvořena dvojicí plošných spojů vyvinutých firmou *PiKRON* pro účely experimentování s řízením motorů s využitím FPGA na Katedře řídicí techniky. Jsou zde integrovány bloky pro výkonové řízení motoru a měření proudů protékajících jeho vinutím. V tomto místě také dochází ke zpracování výstupů z IRC a Hallových senzorů. Pro ochranu řidíci elektroniky a potlačení rušení je výkonová část kompletně galvanicky oddělena od napájení RPi a FPGA obvodu pomocí dvoukanálových číslicových izolátorů [1]).

Výkonové řízení motoru realizuje trojice polovičních H-můstků tvořených vždy dvojicí výkonových N-MOS trazistorů. Každá dvojice/půlmůstek je řízená integrovaným obvodem řady LT1158 [2]). Vstupy těchto budičů jsou již ovládány přímo z FPGA.

Pro měření proudů protékajících jednotlivými fázemi motoru je použita sada senzorů využívajících Hallova efektu. Napětí je pak převedeno na digitální hodnotu 12 bitovým A/D převodníkem[3]). Tento integrovaný obvod je vybaven čtyřmi kanály, z nichž zde jsou použity tři. Pro přepínání mezi nimi je v převodníku integrován multiplexor. ADC může pracovat při napájecím napětí v rozsahu od 2.7V do 5V. Od napájecího napětí, které je současně i referenčním (viz. Příloha D), se následně odvíjí frekvence hodinového signálu, kterou je obvod časován a slouží i k přenosu dat. Při U_{ref}=2.7V je maximální frekvence 2Mhz, pro U_{ref}=5 pak f_{CLK}=5Mhz [20].

Jedná se o převodník s postupnou aproximací. Tento druh převodníku vyžaduje pro každý převod pevně stanovený počet hodinových cyklů. Při každém hodinovém cyklu pak teoreticky dochází k zpřesnění o jeden bit. Realizován bývá obvykle s využitím několika komponent. Jednou z nich je jednoduchý analogový vzorkovač, který udrží měřenou veličinu neměnnou po čas nutný pro konverzi. Dalšími součástmi jsou DA převodník a aproximační registr (SAR). Komunikace s AD převodníkem je podrobně probrána v sekci 4.4.



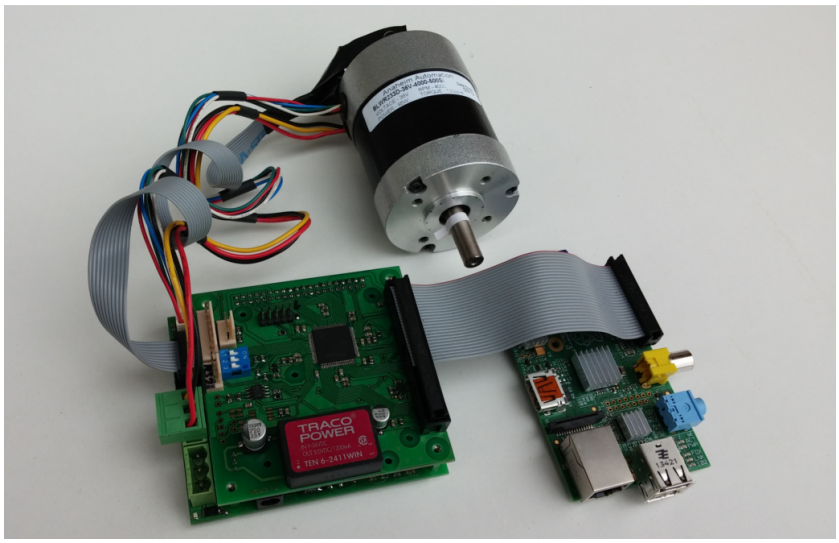
Obrázek 3.3. Propojení Raspberry Pi s výkonovým HW

[1]) ADuM1200 Dual-Channel Digital Isolators `http://www.analog.com/media/en/technical-documentation/data-sheets/ADuM1200_1201.pdf`

[2]) Half Bridge N-Channel Power MOSFET Driver `http://cds.linear.com/docs/en/datasheet/1158fb.pdf`

[3]) ADS7841 `http://www.ti.com/lit/ds/symlink/ads7841.pdf`

Celá rozšiřující jednotka může být napájena v poměrně velkém rozsahu napětí pohybujím se mezi 9V až 30V. Toto napětí je poté konvertováno pomocí DC-DC měniče [1]) na hodnotu 5V, které je vyžadováno RPi. Protože logika na RPi využíva 3.3V napěťovou úroveň, dojde na RPi ještě ke konverzi na 3.3V, kterými je napájen i FPGA obvod. Mechanické propojení rozšiřující jednotky s Raspberry Pi je realizováno prostřednictvím plochého 26-žilového kabelu viz. Obrázek 3.3. Propojení motoru s rozšiřující jednotkou a RPi je na zachyceno na obrázku 3.4. Schéma obou plošných spojů rozšiřující jednotky se nachází v příloze D.



Obrázek 3.4. Propojení motoru s rozšiřující jednotkou a RPi

[1]) TEN 6-2411WIN Traco Power DC/DC Converters `http://mediaserver.voxtechnologies.com/FileCache/Traco%20Power-TEN%206WIN%20Series-datasheet1-1243850956.pdf`

Kapitola 4
Použité řešení

Předcházející kapitola byla věnována použitému hardware a jeho specifikacím. Nyní následuje popis, průběhu práce samotné.

První sekce diskutuje volbu frekvence řídicí smyčky. Následuje debata k objemu dat, který je třeba přenášet mezi Raspberry Pi a rozšiřující jednotkou. Další sekce se zabývá výběrem a implementací vhodného komunikačního protokolu, který slouží k propojení rozšiřující jednotky s nadřazeným hardware. Následuje část, která se věnuje čtení a částečnému zpracování dat z A/D převodníku. Ten je připojen k senzorům snímajícím proudy protékající fázemi motoru. Budou probrány požadavky RT rozšíření Linuxu a postupu pro jeho přípravu. Tématem se stane i generování PWM signálů a dekódování IRC. Nakonec bude popsána elektronická komutace a implementace vektorového řízení v jazyce C.

4.1 Frekvence řídicí smyčky

Ve chvíli, kdy přejdeme od spojitého řízení k diskrétnímu, je nutné se zabývat frekvencí řídicí smyčky.

Řídicí smyčka je proces, při kterém dochází k periodickému vzorkování hodnot nutných pro řízení a následnému nastavení velikosti akčního zásahu. Její frekvence je tak závislá na časových konstantách řízeného systému, či zjednodušeně, jen na té dominantní. V případě elektrických motorů je přitom dominantní časovou konstantou časová konstanta mechanická.

U malých elektrických motorů se mechanická časová konstanta pohybuje v rozmezí od 1 do 20ms. Je ale obvyklé, že i moderní motory s lehkou kotvou nemají menší hodnoty mechanické časové konstanty než 3ms. [13]. Pro přesné určení časových konstant je třeba brát v úvahu parametry motory uvedené v manuálu nebo v tabulce 3.1. V případě reálné aplikace pak i transformovanou hodnotu momentu setrvačnosti zátěže, která výslednou časovou konstantu zvyšuje.

Model motoru, kde je implementováno poziční řízení má charakter integrátoru. Pokud vyžadujeme od regulátoru kvalitní výsledky, především krátkou dobu ustálení, je tak třeba volit frekvenci regulační smyčky alespoň dvojnásobnou v porovnání s vlastní frekvencí motoru, tedy převrácenou dominantní časovou konstantou.

V mezním případě může vlastní frekvence dosahovat hodnot $f = \frac{1}{0.003} = 333\text{Hz}$. S dostatečnou rezervou tak byla zvolena frekvence regulační smyčky o velikosti 1kHz.

4.2 Objem dat přenášených mezi RPi a rozšiřující jednotkou

K úspěšnému řízení potřebuje řídící jednotka (RPi) data získaná ze senzorů proudu, IRC a Hallových sond. Rozšiřující jednotka pak vyžaduje hlavně šířku plnění PWM a

řídicí signály pro poloviční H-můstky. Je také nutné určit přesnost, s jakou se budou potřebná data přenášet.

Z IRC je posíláno 32 bitů dat s informací o aktuální poloze motoru. Dále je posílána 12-bitová informace o vzdálenosti od indexu. Index je přesně určené místo v rozsahu otáčení, maximální hodnota této vzdálenosti tak může být 1999 rozlišitelnýcch pozic. Tři bity jsou obsazeny výstupy z Hallových sond. Následuje 3x16 bitů informace o změřených proudech. Celkem tak v tomto směru dojde při každém průchodu řídicí smyčkou k výměně 128 bitů dat.

Ke výměně stejného objemu dat dojde z důvodu použitého komunikačního protokolu i v opačném směru. Z RPi do rozšiřující jednotky přichází tři 11-bitové informace o šířce plnění PWM. Tyto hodnoty jsou ale pro snadnější manipulaci zarovnány na 16 bitů. Jen dolních 11 z nich je ale využito. Součástí přenosu jsou ještě 3 bity odemykající poloviční H-můstky a 3 bity určené k jejich úplné deaktivaci. Poslední využitý bit pak při změně z logické 0 do 1 povoluje resetování AD převodníku. Celkem je v tomto směru komunikace využito jen 40 ze 128 bitů.

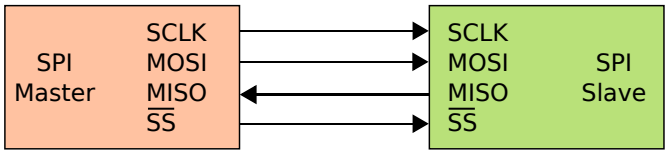
4.3 Komunikační protokol

Výběr komunikačního protoku mezi RPi a FPGA obvodem se stal jednou z prvních otázek a bylo ji třeba zodpovědět již na začátku práce.

Existuje celá řada známých a hojně využívaných protokolů pro komunikaci ať již mezi periferiemi a procesorovou jednotkou či mezi procesory navzájem. Příkladem je I^2C, SSI, USART a jeho synchronní varianta SPI. Požadavkem přitom bylo, aby komunikační protokol byl dostatečně rychlý a aby při frekvenci minimálně 1kHz dokázal přenést alespoň 128bitů. A to oběma směry. Je také nutné, aby byl dostatečně jednoduchý pro efektivní realizaci v FPGA obvodu.

Protokol I^2C se zdál příliš složitý, vzhledem ke svým módům. Adresace, kterou poskytuje by mohla být využita v případě ovládání více os, je zde zbytečná. Hlavní nevýhodou je nedeterministické chování.

Asynchronní verze protokolu USART nemá v tomto případě smysl, protože vybraný FPGA obvod nemá vnitří zdroj hodinového signálu. Hlavní doménou tohoto protokolu je využití pro konzoli a jeho složitá implementace na straně operačního systému opět nezaručuje determinismus.



Obrázek 4.1. SPI s jedením uzlem typu slave (Zdroj:wikipedia.org)

Jako příhodné varianty se jeví protokoly SSI a SPI. Výhodou SSI protokolu je, že jím generovaný hodinový signál, posílaný po samostatném vodiči, není zastaven s koncem komunikačního cyklu. Protokol SPI hodinový signál v této chvíli zastavuje a je tedy nutné do FPGA obvodu přivést hodinový signál jinou cestou. Hodinový signál je totiž nutný pro funkci všech synchronních komponent obvodu. Na rozdíl od protokolu SSI je ale SPI velmi pohodlně přístupný z uživatelské aplikace. Není tedy třeba tvořit vlastní

ovladač na straně jádra OS. Výhodou SSI ale zůstává časté použití při připojování obvodů k DSP a pro motion control. Pro práci byl nakonec vybrán protokol SPI.

SPI je deterministický komunikační protokol využívající model komunikace „master-slave“. Pro komunikaci mezi dvěma zařízeními jsou vyžadovány obvykle čtyři datové vodiče.

- `SCLK` přenaší hodinový signál, jehož zdrojem je zařízení typu „master“.
- `MOSI` (Master Out Slave In) je datová linka nesoucí sekvenci bitů směrem od „master“ ke „slave“.
- `MISO` (Master In Slave Out) pak vede data opačný směrem.
- `SE` (Slave Enable, někdy značeno `CE` - Chip Enable, na „slave“ zařízeních `SS` - Slave Select), je čtvrtý vodič, obvykle řízený inverzní logikou (Active-Low), kterým master oznamuje slave, že spustí hodinový signál a dojde k výměně dat. V případě více uzlů typu „slave“ vede do každého zvláštní vodič `SE`, zatímco ostatní tři jsou společné.

Existují čtyři módy tohoto protokolu, které lze volit nastavením dvou konfiguračních bitů [21].

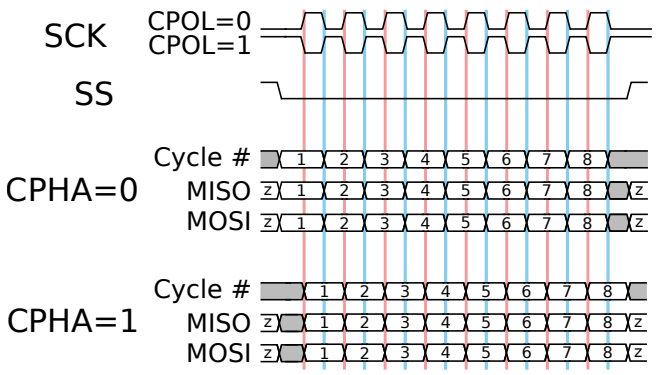
- `CPOL`, první bit, definuje, jaký je stav hodin (signálu `SCLK`) na počátku komunikace - ve chvíli, kdy `SE` přejde do stavu `Low`. Při `CPOL`=0 je počáteční stav `SCLK`=0 a první hrana je tedy nástupná. Při `CPOL`=1 začínají hodiny ve stavu `High` a první hrana je sestupná.
- `CPHA`, druhý bit, indikuje, zda jsou data vzorkována či zapisována při nástupné nebo sestupné hraně. Při `CPHA`=0 jsou data vzorkována při nástupné hraně `SCLK` a zapisována při hraně sestupné. Pokud je `CPHA`=1, je tomu naopak.

Platí přitom, že zápis dat, stejně jako jejich vzorkování, probíhá v obou zařízeních současně. Přiřazení režimů k nastavení jednotlivých bitů shrnuje Tabulka 4.1.

Mód	CPOL	CPHA
0	**0**	**0**
1	0	1
2	1	0
3	1	1

Tabulka 4.1. Módy protokolu SPI

Při použití módů 0 a 2, kdy `CPHA`=0 je nutné data připravit ještě před příchodem první hrany hodin. Ta, ať už je sestupná či nástupná, totiž data již vzorkuje. Data jsou tedy vystavena s poklesem signálu `SE`. Průběhy komunikace při jednotlivých kombinacích `CPOL` a `CPHA` ukazuje Obrázek 4.2.



Obrázek 4.2. Průběh komunikace po SPI při různých módech (Zdroj:wikipedia.org)

Protože jednotlivé módy jsou kvalitativně rovnocené, výběr jednoho z nich není nijak rozhodující. Přesto se dá říci, že mód 0 je asi nejpoužívanější, proto byl zvolen právě ten. Dále v textu nebude tato problematika více zmiňována a bude předpokládáno, že je používán právě mód 0.

4.3.1 Zdroj hodinového signálu pro FPGA

Jak bylo uvedeno výše, hodinový signál generovaný při použití protokolu SPI trvá jen po dobu přenosu dat. Zdrojem hlavního hodinového signálu pro FPGA obvod se tak stal generátor hodinových pulzů na RPi, jehož výstup je možné přivést na jeden ze vstupně/výstupních pinů. Jeho frekvence byla stanovena na 50MHz. Dále v textu bude tento signál označen stejně, jako ve VHDL kódu, tedy `gpio_clk`. Nastavení generátoru je implementováno v souboru `rpi_hw.c`.

4.3.2 SPI slave v FPGA obvodu

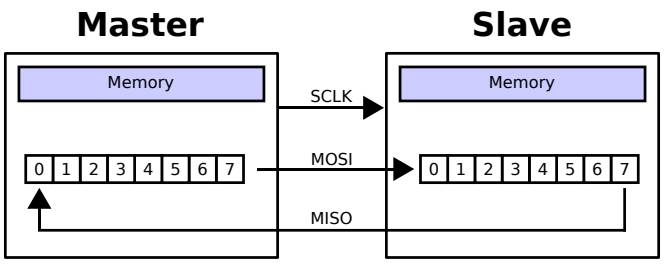
FPGA obvod je v tomto řešení uzlem typu slave. V opačném případě by mohlo docházet k problémům se zajištěním trvalé připravenosti RPi v roli slave. Pokud je ale RPi uzlem typu master, tak se případné problémy s dostatečně rychlým plnění a čtení SPI neprojeví, protože se SPi periferie zastaví a zastaví i hodiny přenosu. Implementace obvodového návrhu se nachází souboru `rpi_pmsm_control.vhdl`.

Komunikace je pak součástí jednoho Procesu v jazyce VHDL. Proces byl vytvořen jako synchronní, spínaný jen nástupnou hranou vnitřního hodinového signálu (`gpio_clk`). Spínání tohoto procesu signály `SCLK` a `SE` by nebylo výhodné, protože by vedlo k vytvoření nové hodinové domény. V takovém případě by bylo nutné řešit přenos již širokých paralelních dat mezi hodinovými doménami. Je tedy nutné signály `SE` a `SCLK` synchronně vzorkovat. V kódu Procesu se tak objeví následující část:

```
wait until (gpio_clk'event and gpio_clk='1');
  --SCLK edge detection
  spiclk_old(0)<=spi_clk;
    spiclk_old(1)<=spiclk_old(0);

  if (spiclk_old="01") then --rising edge, faze cteni
  ...
  práce s daty
  ...
  end if;
```

Protokol SPI je v FPGA obvodu obvykle implementován jako posuvný registr viz. Obrázek 4.3.



Obrázek 4.3. Realizace SPI pomocí posuvného registru (Zdroj:wikipedia.org)

Průběh komunikace ilustruje obrázek 4.2. S aktivací signálu `SE`, tedy s jeho sestupnou hranou, jsou připraveny data do posuvného registru. Zároveň je třeba do vodiče `MISO`

zapsat první bit přenosu. První příchozí hrana signálu SCLK je nástupná, tedy vzorkovací. Po jejím příchodu je posuvný registr posunut o jeden bit doleva a na uprázdněné místo je zapsán aktuální stav vodiče MOSI.

```
dat_reg(127 downto 0) <= dat_reg(126 downto 0) & spi_mosi;
```

Po příchodu druhé hrany signálu SCLK, tedy sestupné, je nutné vystavit do vodiče MISO nový bit.

```
spi_miso <= dat_reg(127);
```

Tento postup se opakuje až do deaktivace signálu SE.

4.3.3 SPI master na Raspberry Pi

Pro uživatelské aplikace je periferie SPI v OS Linux zpřístupněna jako znakové zařízení (obvyklé řešení pro Unix nebo Windows NT). Po otevření určeného souboru/inode reprezentujícího SPI je možné přenosy realizovat voláním řídicí funkce (systémového volání IOCTL) s určenými parametry. Implementace znakového zařízení je umístěna v driveru `spidev.c`, který je součástí jádra OS Linux.

V přiloženém kódu je pak pužití tohoto driveru součástí souboru `rp_spi.c`. V úvodu je třeba vložit hlavičku `#include <linux/spi/spidev.h>`. Klíčovým prvkem je zde volání zmíněné funkce `int ioctl(int d, int request, ...)`. První argument, argument `d`, je file descriptor znakového zařízení. Zařízení je zpřístupněno voláním funkce `int open(const char *pathname, int flags)`, jejíž návratovou hodnotou je deskriptor. Argument `pathname` je cesta k souboru. Argument `flags` určuje mód otevření. Příkaz tedy může vypadat:

```
1  int fd = open("/dev/spidev0.1", O_RDWR);
```

Makro `O_RDWR` specifikuje otevření pro čtení i psaní.

Druhým argumentem funkce `int ioctl(int d, int request, ...)` je request kód. Jedná se o kód specifický pro každou operaci s daným zařízením zvlášť. Posledním argumentem je ukazatel (pointer) na datovou strukturu specifickou pro daný příkaz. Datový typ ukazatele není striktně určen. Konvence však, tam, kde není uvedeno jinak, doporučuje volit `char * argp` [22].

Jako první je třeba nastavit mód SPI, to je provedeno s kódem operace definovaným makrem `SPI_IOC_WR_MODE` následovně:

```
2   unsigned char mode = 0;
3   int ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
```

Dále je nutné stanovit délku slova, využijeme `SPI_IOC_WR_MODE`:

```
4  unsigned char bits = 8;
5  ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
```

Poslední část konfigurace vyžaduje nastavit frekvenci hodin `SCLK`, použijeme kód operace `SPI_IOC_WR_MAX_SPEED_HZ`. Frekvence je očekávána v jednotkách Hz.:

```
6  unsigned int speed = 500000;
7  ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
```

Před samotným přenosem je ještě nutné nastavit parametry specifické pro daný přenos, vyplnit ukazatel na úsek paměti s výstupními daty (`tx`) a ukazatel na oblast připravenou pro uložení vstupních dat (`rx`).

```
8     unsigned char rx[16];
9     unsigned char tx[16];
```

Parametry jsou uloženy ve struktuře `spi_ioc_transfer`:

```
10  struct spi_ioc_transfer tr =
11  {
12      .tx_buf = (unsigned long)tx,
13      .rx_buf = (unsigned long)rx,
14      .len = 16,
15      .delay_usecs = 0,
16      .speed_hz = speed,
17      .bits_per_word = bits,
18  };
```

Přenos je spuštěn opět voláním IOCTL, s kódem operace `SPI_IOC_MESSAGE(1)`.

```
19  ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
```

Po jeho úpěšném průběhu je možné již používat doručená data v paměti odkazované ukazatelem `rx`.

4.3.4 Pořadí dat v přenosu

Prvním přenášeným bitem komunikace je bit s nejvyšší hodnotou (MSB), tedy bit `dat_reg(127)` na straně FPGA. Za předpokladu, že data pro přenos jsou v Raspberry uchovávána v poli charů a odpadnou tak problémy s endianitou (instrukce ARMv6 jsou defaultně little-endian [1])), bude prvním přenášeným bitem MS bit v `tx[0]`. Pozice a funkce bitů v přenosech z Raspberry Pi do FPGA a v opačném směru jsou zdokumentované v tabulkách 4.2 a 4.3.

Bity	Byte přenosu	Funkce
127	tx[0]	Aktivace resetu AD převodníku
126 .. 124	tx[0]	Enable PWM1 .. PWM3
123 .. 121	tx[0]	Shutdown můstku pro PWM1 .. PWM3
120 .. 43		-
42 .. 32	tx[10] tx[11]	Šířka plnění PWM1
31 .. 27		-
26 .. 16	tx[12] tx[13]	Šířka plnění PWM1
15 .. 11		-
10 .. 0	tx[14] tx[15]	Šířka plnění PWM1

Tabulka 4.2. Přenos z Raspberry Pi do FPGA obvodu

[1]) ARMv6 Support for mixed-endian data `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Cdfbbchb.html`

Bity	Byte přenosu	Funkce
127 ..96	rx[0] .. rx[3]	IRC
95	rx[4]	Hall 1
94	rx[4]	Hall 2
93	rx[4]	Hall 3
92 .. 81	rx[4] rx[5]	Pozice indexu
80 .. 72	rx[5] rx[6]	Počet sečtených proudů
71 .. 48	rx[7] .. rx[9]	Součet proudů, kanál 2
47 .. 24	rx[10] .. rx[12]	Součet proudů, kanál 0
23 .. 0	rx[13] .. rx[15]	Součet proudů, kanál 1

Tabulka 4.3. Přenos z FPGA obvodu do Raspberry Pi

4.4 Čtení hodnot z AD převodníku

Měření statorových proudů je nutné k plnému momentovému řízení nebo k řízení bez polohových senzorů (sensorless control).

Plošný spoj výkonového stupně *3P-MOTOR-DRIVER-1* proto obsahuje čtyřkanálový, 12-bitový A/D převodník ADS7841. Na vstup převodníku jsou připojeny napěťové výstupy senzorů proudů protékajících jednotlivými fázemi, které využívají Hallova jevu. Převodník může pracovat buď ve 12-bitovém nebo jen v 8-bitovém režimu [20].

Důležité je zvolit správnou hodinovou frekvenci, která bude přivedena na hodinový vstup převodníku. Její maximální hodnota se pro daný typ převodníku liší podle velikosti napájecího napětí obvodu, viz sekce 3.6. Obvod je stejně, jako celé FPGA, napájen napětím 3.3V, které je přivedeno z RPi. Hodinová frekvence tak byla stanovena na hodnotu 2.08Mhz.

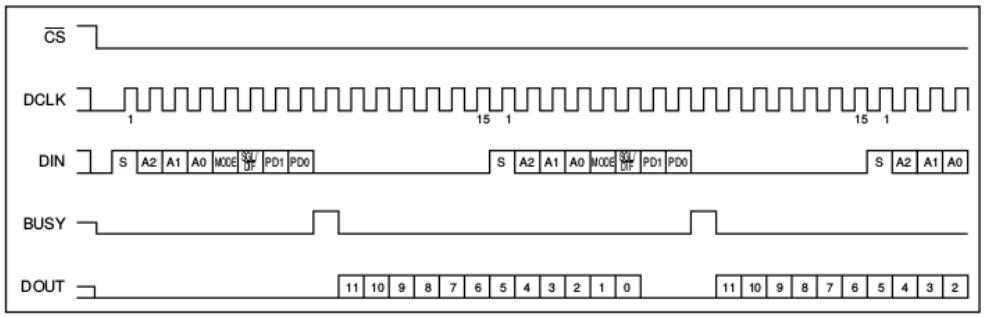
Tato frekvence je vytvořena kombinací frekvenčních děliček implementovaných v komponentách `divider.vhdl` a `adc_reader.vhdl`. V komponentě `dvider.vhdl` dochází k vydělení frekvence 50MHz dvanácti a vzniká tak frekvence o velikosti 4.17MHz. Tato frekvence je následně vydělena dvěma v komponentě `adc_reader.vhdl`, o které je více napsáno níže. Aby v rámci komponenty `adc_reader` nedošlo k vytvoření nové hodinové domény, je ke spouštění Procesu ve VHDL určen hlavní 50MHz hodinový signál. Tento signál pak pouze vzorkuje pomalejší 4.17Mhz hodiny. Podobné řešení je popsáno již v sekci 4.3.2.

AD převodník je schopen pracovat v několika módech, které se liší počtem hodinových cyklů nutných k jednomu převodu. V základním módu je třeba 24 hodinových pulzů na jednu konverzi. Tento mód předpokládá komunikaci se zařízením vyžadujícím přenos dat po SPI protokolu s pevnou délku slova 8 bitů. Je určený především pro nepravidelné odečítání většinou z mikrokontroléru. Pokud je třeba odečítat hodnoty pravidelně a co nejrychleji, je k dispozici 16-pulzový převod.

Je-li možnost použít FPGA obvod, není nutné se omezovat 8-bitovou šířkou slova a je možné využít nejrychlejší variantu. Tedy konverzi dokončenou za 15 hodinových pulzů. V použitém řešení byla zvolena právě tahle varinta. Průběh 15-cyklového převodu z hlediska datového přenosu ukazuje Obrázek 4.4.

Komponenta `adc_reader`, kromě frekvenční děličky, která je spíše vedlejším produktem, realizuje celou komunikaci mezi AD převodníkem a FPGA. Ta se skládá z přenosu řídícího slova z FPGA do ADC a z následného vyčtení hodnot. Převodník je připojen pomocí čtyř vodičů, analogických k vodičům použitých v SPI komunikaci: `CS`(SE), `DCLK`(SCLK), `DIN`(`MOSI`) a `DOUT`(`MISO`).

Řídící slovo je tvořeno osmi bity. Prvním z nich je startbit, který při delším provozu s nepravidelným vyčítáním, při kterém není vypnutý hodinový signál, definuje počátek komunikace. Následují tři bity adresy určené pro výběr jednoho ze čtyř kanálů. Další bity definují výběr buď 8 či 12-bitové konverze a volbu mezi „Single-Ended“ nebo diferenčním zapojením. Poslední dva bity slouží k nastavení úspornéro režimu v době mezi konverzemi.



Obrázek 4.4. Průbeh konverze v režimu 15 hodinových pulzů na převod (Zdroj:Texas Instruments)

Komponenta `adc_reader` pak implementuje komunikaci pomocí stavového automatu. Každý ze stavů generuje buď logickou úroveň 1 či 0 do signálu `DCLK`. Pokud tak nastala při přechodu mezi dvěma stavy sestupná hrana, je zapsán bit řídícího slova na vodič `DIN`. Ve stejné chvíli také dochází k zápisu hodnoty na vodič `DOUT` na straně ADC. Při nástupné hraně jsou naopak hodnoty na vodičích `DIN` a `DOUT` vzorkovány. Protože k přechodu mezi dvěma stavy dochází vždy synchronně s nástupnou hranou hodinového signálu, má signál `DCLK` vysílaný do převodníku právě poloviční frekvenci.

Nutnost přecházet mezi stavy však neexistuje jen u konfigurace převodníku a vyčítání hodnot. Je třeba také přepínat i mezi jednotlivými kanály. Výše zmíněný stavový automat je tak logicky vnořen do dalšího, který přepíná právě mezi odečítáním proudů z jednotlivých fází.

Konfiguraci řídicího slova a adresy kanálů všech tří fází ukazují tabulky 4.4 a 4.5.

Bit	Hodnota	Funkce
7 (MSB)	1	Start bit
6	Bit 2 adresy	Bit 2 adresy
5	Bit 1 adresy	Bit 1 adresy
4	Bit 0 adresy	Bit 0 adresy
3	0	12-bitový převod
2	1	Single-Ended zapojení
1	0	Power Down mód
0 (LSB)	0	Power Down mód

Tabulka 4.4. Konfigurace řídícího slova ADC

Kanál	Bit 2 adresy	Bit 1 adresy	Bit 0 adresy
ch0	0	0	1
ch1	1	0	1
ch2	0	1	0

Tabulka 4.5. Adresace kanálů ADC

4.4.1 Sčítání více hodnot proudu

Po první implementaci měření proudů bylo patrné, že nedochází k ustálení hodnot přibližně na posledních čtyř bitech. Částečným zpřesněním může být použití výběrového průměru namísto jen jedné změřené hodnoty.

K vyčtení hodnoty z AD převodníku dochází v porovnání s frekvencí řídicí smyčky (1kHz) poměrně rychle. Při 15 hodinových cyklech nutných pro jednu konverzi a hodinové frekvenci 2.08Mhz, může dojít až ke 138 převodům. Pro každou měřenou fázi se jedná přibližně o 46 převodů. Byla provedena následující úvaha:

Změřenou hodnotu proudu procházejícího jednou z fází za neměnných podmínek označme jako náhodnou veličinu X. Za předpokladu, že tato náhodná veličina má normální rozložení pravděpodobnosti, platí, že skutečná hodnota proudu je rovna její střední hodnotě. Tedy $i_p = EX$, $p \in \{a, b, c\}$.

Realizujme tedy náhodný výběr z této veličiny a spočtěme jeho výběrový průměr. Pro výběrový průměr náhodné veličiny platí následující vztahy[23]:

Výběrový průměr z náhodného výběru $\mathbf{X} = (X_1, ..., X_n)$ je

$$\overline{\mathbf{X}} = \frac{1}{n} \sum_{j=1}^{n} X_j \tag{1}$$

Potom platí, že střední hodnota výběrového průměru se rovná střední hodnotě náhodné veličiny X.

$$E\overline{\mathbf{X}} = EX \tag{2}$$

Pro jeho směrodatnou odchylku $\sigma_{\overline{\mathbf{x}}_n}$ pak platí:

$$\sigma_{\overline{\mathbf{x}}_n} = \frac{1}{\sqrt{n}} \sigma_x \tag{3}$$

Za předpokladu, že $n = 46$ tedy platí, že $\sigma_{\overline{\mathbf{x}}_{46}} \approx \frac{1}{6.8}\sigma_x$. Směrodatná odchylka výběrového průměru se oproti odečtení samostatných měření zmenšila 6.8x. A protože $\log_2 6.8 \approx 2.7$, je možno říci, že pokud byl původní signál snímán s dostatečným šumem, byly získány téměř tři bity informace navíc.

4.5 Generování PWM

Generováním signálů PWM se zabývá komponenta `mcpwm`. Její návrh implementuje 11-bitovou PWM.

Základem funkce je nastavení příslušného výstupu na hodnotu logická 1 při průchodu čítače `count` nulou a ukončení plnění PWM cyklu při shodně čítače `count` se zadanou hodnotou plnění, uloženou v registru `match_reg`. Hodnota `match` je odesílána ve finální podobě již z Raspberry Pi. Čítač `count` je inkrementován hlavním hodinovým signálem `gpio_clk` o frekvenci 50MHz, frekvence PWM je tedy $50000/2^{11} \approx 24.4$kHz.

Průchod čítače `count` nulou je mimo jiné detekován nastavením signálu `sync` po dobu jednoho hodinového cyklu na hodnotu 1.

```
process (sync, count, match_reg, q)
begin
  if count = match_reg then
    next_q <= '0';
  elsif sync = '1' then
```

```
      next_q <= '1';
    else
      next_q <= q;
    end if;
  end process;
```

Jeden hodinový cyklus před nastavením signálu `sync` na hodnotu 1, je prováděna kontrola, zda je k dispozici nová hodnota signálu `match`. V takovém případě je v dalším cyklu PWM používána již aktualizovaná hodnota `match_reg<=match`.

Protože potřebujeme generovat tři signály PWM, je nutné bloky `mcpwm` vytvořit právě tři. Vystup čítače je pak rozveden na všechny tři bloky.

4.6 Dekódování IRC

Dekódování výstupů IRC senzoru je realizováno v komponentě `qcounter.vhdl`.

Vstupem komponenty jsou signály `irc_A` a `irc_B`. Podle pořadí, v jakém dochází ke změnám těchto dvou signálů,je možné odvodit, v jakém směru se motor otáčí.

Signály `irc_A` a `irc_B` jsou nejprve synchronně vzorkovány DFF obvodem spouštěným hlavní hodinovou frekvencí `gpio_clk`. Aby došlo ke ztrátě informace o poloze, muselo by tak dojít ke dvěma změnám jednoho signálu během jednoho hodinového cyklu. Motor by se tedy musel za tuto dobu pootočit o tři rozlišitelné pozice IRC. To by při frekvenci `gpio_clk` 50MHz znamenalo frekvenci otáčení nejméně 75KHz, tedy 4.5 miliónu otáček za minutu, což není z konstrukčních důvodů zdaleka možné.

Nejmenší dva bity výsledné pozice jsou vytořeny čistě kombinačně z navzorkovaných `irc_A` a `irc_B`:

```
  qcount(0) <= a xor b;
  qcount(1) <= b;
```

Přechod do vyššího řádu je pak řešen pomocí 30-bitového čítače `count`:

```
  if (a_prev = '0') and (b_prev = '1') and (a = '0') and (b = '0') then
    count <= count_prev + 1;
  elsif (a_prev = '0') and (b_prev = '0') and (a = '0') and (b ='1') then
    count <= count_prev - 1;
  else
    count <= count_prev;
  end if;
```

Veličiny s příponou `_prev` jsou pak veličiny zpožděné o jeden hodinový cyklus oproti těm bez přípony. Čítač je nakonec do výstupu komponenty promítnut následovně:

```
    qcount(31 downto 2) <= count;
```

4.7 Implementace požadavků RT rozšíření v uživatelském kódu

Aby bylo možné plně využívat možností, které nabízí RT-rozšíření jádra Linux, je nutné v uživatelském kódu dodržovat některá základní pravidla. K těmto pravidlům patří například použití RT plánovací strategie nebo správné nastavení priorit vláken v rámci plánovače [24]. Implementaci těchto požadavků se věnuje právě tato sekce.

4.7.1 Nastavení priority vlákna a plánovací strategie

Jádro OS Linux implementuje pět různých plánovacích strategií. Strategie `SCHED_FIFO`, `SCHED_RR` a `SCHED_OTHER` jsou definovány standartem POSIX, `SCHED_BATCH` a `SCHED_IDLE` jsou pak pro systém specifické. Pro plánování každého vlákna aplikace může být zvolena libovolná z nich.

Strategie `SCHED_OTHER` přiděluje úlohám poměrnou část strojového času podle nastavení *nice*.

Pro RT aplikace jsou využívány strategie `SCHED_FIFO` a `SCHED_RR`, které mají v okamžiku přeplánování přednost před všemi ostatními. Strategie `SCHED_FIFO` a `SCHED_RR` využívají plánování s pevnými prioritami. Pokud je připraveno k běhu vlákno s vyšší prioritou, než má vlákno právě běžicí, je běžící vlákno v nebližší možné době nahrazeno vláknem z vyšší prioritou.

Omezou dobu odezvy, za kterou dojde k přeplánování, pak garantuje pouze jádro s aplikovaným RT rozšířením, tím že minimalizuje nepřerušitelné úseky kódu ve službách a servisních vláknch jádra OS.

Priority vláken v rámci strategií `SCHED_FIFO` a `SCHED_RR` je možné explicitně stanovit v uživatelském kódu. Rozsah jejich hodnot se pohybuje mezi 99 a 1. Hodnota 50 je výchozí priorita úloh obsluhujících v RT variantě jádra přerušení od periférií. Platí, že čím vyšší hodnota, tím vyšší priorita.

V rámci prioritního plánování je ještě nutné stanovit, k jakému chování dojde, budou-li k běhu připravena dvě vlákna se stejnou prioritou. V tomto bodě se strategie `SCHED_FIFO` a `SCHED_RR` následovně odlišují:

- `SCHED_FIFO`: V případě, že právě beží jiné `SCHED_FIFO` nebo `SCHED_RR` vlákno se stejnou prioritou, je nové vlákno zařazeno na konec fronty úloh, se stejnou prioritou, čekajících na přidělení procesorového času. Ve chvíli, kdy je novému vláknu procesor přidělen, není mu odejmut, dokud není zablokováno, ukončeno nebo nahrazeno vláknem s vyšší prioritou.
- `SCHED_RR`: Sdílí všechny vlastnosti `SCHED_FIFO`. Běžící vlákno je ale přeplánováno vždy po uplynutí pevně stanoveného časového kvanta. Respektive je zařazeno na konec fronty úloh, se stejnou prioritou, čekajících na přidělení procesorového času.

Pro nastavení priority a plánovací strategie v rámci vlákna slouží systémové volání: `int sched_setscheduler(pid_t pid, int p, const struct sched_param *s)`. Jeho použití je následující:

```
struct sched_param param;
param.sched_priority = PRIORITY;
if(sched_setscheduler(0, SCHED_FIFO, &param) == -1) {
     perror("sched_setscheduler failed");
     exit(-1);
}
```

Pro nastavení priority a plánovače vlákna, které teprve bude vytvořeno je nutné nejprve povolit nastavení plánovače prostřednicvím atributu, který je předáván při tvorbě vlákna (dále jen „atribut“). Poté je třeba nastavit prioritu a plánovací strategii. Nakonec je vytvořeno i samotné vlákno. Celý proces je uveden níže: (Pro zvýšení přehlednosti zde byla odstraněna detekce chybových stavů)

```
pthread_attr_t attr;         /*atribut předávaný při tvorbě vlákna*/
struct sched_param schparam; /*struktura pro nastavení priority*/

pthread_attr_init(&attr);    /*inicializace atributu*/

/*povolení nastavení planovací strategie prostřednictvím atributu*/
(pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);

pthread_attr_setschedpolicy(&attr, SCHED_FIFO); /*nastaveni planovace*/

schparam.sched_priority = PRIORITY;             /*nastaveni priority*/

/*ulozeni priority podle hodnoty 'schparam' do atributu*/
pthread_attr_setschedparam(&attr, &schparam);

pthread_create(thread, &attr, start_routine, arg); /*vytvori vlakno*/

/*uvolneni struktury, nema vliv na vlakna jiz vytvorena*/
pthread_attr_destroy(&attr);
```

Celá implementace se nachází v souboru `misc.c`.

4.7.2 Řídicí smyčka

Frekvence řídicí smyčky byla stanovena v sekci 4.1. Nyní je třeba zajistit, aby byla tato frekvence skutečně dodržena.

Pro periodického vykonávání kódu o stanovené frekvenci je možné využít cyklu `while`, v jehož těle je volána funkce realizující čekání. Nejjednodušší variantou takové funkce jsou knihovní volání `unsigned int sleep(unsigned int seconds)` a `int usleep(useconds_t usec)`. Tato volání však neberou v úvahu čas, po který je vykonávána samotná periodická činnost. Prodleva, která je takto realizovaná se také může prodlužovat v závislossti na zatížení systému nebo na délce vykonávání samotného volání [25].

Je tedy nutné použít „čekací“ funkci s přesně definovaným chováním. Takovou funkcí je `int clock_nanosleep(...)`. Při jejím volání, na místo čekání po stanovenou dobu, dojde k pozastavení vykonávání kódu do časového okamžiku, definovaného ve struktuře `timespec`. Na počátku je tedy nutné získat aktuální čas, k tomu slouží funkce `int clock_gettime(clockid_t clk_id, struct timespec *tp)`. Její použití je následující:

```
1 struct timespec t_spec;
2 clock_gettime(CLOCK_MONOTONIC ,&t_spec); /*zjisteni pocatecniho casu*/
```

`CLOCK_MONOTONIC` označuje použití systémových hodin, které pracují podobně jako `CLOCK_REALTIME` nezávisle na zatížení systému. Hodnotu `CLOCK_MONOTONIC` navíc není možné za běhu modifikovat například administrátorským zásahem.

Pokud je to nutné, je poté možné inkrementovat `timespec` a nastavit tak okamžik opětovného spuštění kódu. Tato možnost může být využita například při ladění.

```
3   t_spec.tv_sec++; /*pocatecni prodleva - 1s*/
```

Nyní je již možné spustit samotnou smyčku. Při každém jejím průchodu je hodnota uložená ve struktuře `timespec` zvětšena o hodnotu rovnou periodě řídicí smyčky. V totmto případě `1000*1000 ns`.

```
4   while(1) {
5       /* čekání na stanovený časový okamžik */
6       clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &t_spec, NULL);
7
8       /* vlasntni kod ridici smycky */
9
10      t_spec.tv_nsec += PERIODA_NS; /*nastavení nového spouštěcího času*/
```

Aby nedošlo k přetečení počtu nanosekund ve struktuře `timespec`, je nutné průběžně přepočítávat nanosekundy na sekundy.

```
11      while (t_spec.tv_nsec >= NSEC_PER_SEC) {
12          t_spec.tv_nsec -= NSEC_PER_SEC;
13          t_spec.tv_sec++;
14  }   }
```

Implementace v souboru `main_pmsm.c`.

4.7.3 Uzamčení aplikace v RAM

Při vyšším zatížení systému může docházet k odkládání a výměně procesů (Swapping) mezi hlavní (RAM) a sekundární pamětí počítače.

Odložení procesu aplikace, vede ke zvýšení reakčních časů na vnější události. U magnetických disků může tímto způsobem narůstat zpoždění o až několik desítek milisekund.[26] Z tohoto důvodu je nutné, aby proces řídicí aplikace zůstal uzamčen v primární paměti. K tomuto účelu je využíváno systémové volání `int mlockall(int flags)`, které zajistí uzamčení stránek v hlavní paměti.

Parametr `int flags` definuje, zda budou v hlavní paměti uzamčeny jen aktuálně namapované stránky (makro `MCL_CURRENT`) nebo stránky namapované později (makro `MCL_FUTURE`). Tato makra je možno použít současně. Příklad použití je uveden níže:

```
if(mlockall(MCL_CURRENT|MCL_FUTURE) == -1) {
    perror("mlockall failed");
    exit(-2);
}
```

Takto použité volání zajistí, že nedojde v výpadku žádné stránky, která byla nebo bude namapováva do primární paměti počítače. Celý kód se nachází v souboru `misc.c`.

4.8 Elektronická komutace

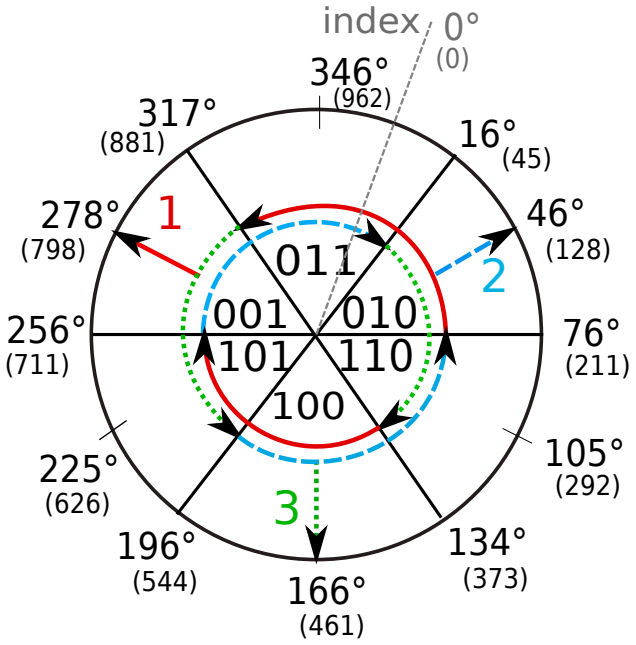
Jak bylo uvedeno výše, PMS motory nevyužívají ke komutaci mechanické kartáče, Komutace je zde řízena elektronicky. Komutaci je možné provádět několika způsoby v závislosti na informacích o stavu motoru, které máme k dispozici. Popisem těchto variant se zabývá právě tato sekce. Všechny zde popsané motody jsou implementovány v souboru `commutators.c`.

4.8.1 Komutace s využitím pouze Hallových sond

Pokud není k dispozici absolutní poloha motoru, není možné řídit komutaci pomocí výstupu z IRC senzoru. V takovém případě je realizován nejjednodušší způsob komutace, pouze za použití výstupu z Hallových sond.

Výstup z Hallových sond umožňuje lokalizovat polohu motor ve dvanáctí úsecích mechanického cyklu. Vzhledem ke konstrukci motoru odpovídají jednomu mechanickému cyklu dva cykly elektrické. Je tedy možné určit polohu motoru v rámci jednoho ze šesti úseků elektrického cyklu. Rozložení úseků elektrického cyklu je na obrázku 4.5.

Z takto zjištěné polohy je následně možné řídit komutaci tak, aby bylo napětí přivedeno jen na jednou z fází a docházelo tak k natočení rotoru do požadované pozice.



Obrázek 4.5. Rozložení výstupů z Hallových sond v elktrickém cyklu (trojice bitů uprostřed [Hall1 Hall2 Hall3]). Závorkované hodnoty při vnějším okraji jsou přibližné vzdálenosti od indexu vypočténé pomocí IRC, kalibrované pro tento motor.

V obrázku 4.5 jsou pomocí šipek na barevných soustředných kružnicích vyznačeny dva směry pohybu rotoru. Pro každý ze směrů různě vybarvéné úseky dané kružnice značí, jaká fáze musí být při daném natočení napájena, aby došlo k pohybu žádaným směrem. Barevnými čísly u šipek směřujících od středu kružnic jsou pak označeny pozice, ve kterých dojde k ustálení motoru při napájení pouze dané fáze.

Pokud je vyžadován pohyb ve směru hodinových ručiček, bude kód komutátoru následující:

```
if (hal2 && !hal3){
    rps.pwm1=0;
    rps.pwm2=0;
    rps.pwm3=duty;
}else if (hal1 && !hal2){
    rps.pwm1=duty;
    rps.pwm2=0;
    rps.pwm3=0;
}else if (!hal1 && hal3){
```

```
    rps.pwm1=0;
    rps.pwm2=duty;
    rps.pwm3=0;
}
```

Velkou nevýhodou této komutace je kolísání točivého momentu v závislosti na úhlu natočení rotoru. Dochází tak k nepříjemným otřesům motoru.

4.8.2 Komutace pomocí přičítání 120 stupňů

Pokud je k dispozici absolutní poloha rotoru v rámci celého elektrického cyklu s dostatečnou přesností, je možné provádět účinnější komutaci.

Jednou z možností je simulovat sinusový průběh napětí na jednotlivých fázích. Fáze přitom budou oproti sobě posunuty o 120°. Pro pohyb ve směru hodinových ručiček může být ve zdrojovém kódu taková operace provedena následovně:

```
sin = pxmc_sin_fixed_inline(pos+DEGREE_240,10);
pwm1=sin*duty/1024;
if (pwm1<0) pwm1=0;

sin = pxmc_sin_fixed_inline(pos+DEGREE_120,10);
pwm2=sin*duty/1024;
if (pwm2<0) pwm2=0;

sin = pxmc_sin_fixed_inline(pos,10);
pwm3=sin*duty/1024;
if (pwm3<0) pwm3=0;
```

Kde `pxmc_sin_fixed_inline` je funkce z knihovny *PXMC*, která celočíselně, se specifikovanou přesností 10 bitů, počítá hodnotu sinu. Proměnná `pos` udává úhlovou pozici rotoru. Po otestování této komutace je ale patrné, že ke kolísání momentu v průběhu otáčky dochází i nadále. Není totiž provedena transformace napětí pro delta zapojení uvedená v sekci 2.3.3.

4.8.3 Vektorové řízení

Vektorové řízení využívá pro výpočet statorových proudů Clarkovu a Parkovu inverezní transformaci. S jejich pomocí je možné řídit směr pohybu rotoru pouze řízením složky proudu rovnoběžné s osou q v $dq0$ souřadném systému. Tato metoda je popsána v sekci 2.3.

Implementace vektorového řízení ve zdrojovém kódu zahrnuje především vytvoření funkcí pro obě inverzní transformace.

Úhel natočení rotoru je inkrementován při pohybu po směru hodinových ručiček, algoritmus pro výpočet inverzní Parkovy transformace proto vychází z rovnice (29):

```
void dq2alphabeta(int32_t *alpha, int32_t *beta,
                  int d, int q,
                  int32_t sin, int32_t cos){
    *alpha=cos*d+sin*q;
    *beta=-sin*d+cos*q;
}
```

Clarkova inverzní transformace je zde vytvořena pomocí (32), její kód je následující:

```
void alphabeta2pwm3(int32_t * ia, int32_t * ib, int32_t *ic,
                    int32_t alpha, int32_t beta){
```

```
    *ia=alpha;
    *ib=-alpha/2+beta*887/1024;
    *ic=-alpha/2-beta*887/1024;
}
```

Pro výpočet hodnot $\sin\vartheta$ a $\cos\vartheta$ je zde použita funkce z knihovny *PXMC*, `void pxmc_sincos_fixed_inline(int32_t *sn, int32_t *cs, uint32_t x, int bit)`. Jejím parametrem je, kromě počtu bitů vysledku, úhel ϑ (parametr `x`). Pro potřeby funkce je tak výstup IRC senzoru, který udává natočení motoru, převeden na cyklickou `uint32_t` logiku. Plný úhel 2π rad je zde ekvivalentní hodnotě 2^{32}.

Transformace napětí při delta zapojení vychází z rovnic (37) až (39). Parametr t je zde volen tak, aby nejmenší z napětí u_1 až u_3 bylo rovno nule. Její kód je následující:

```
void transDelta(int32_t * u1, int32_t * u2, int32_t *u3,
                int32_t ub , int32_t uc){
    int32_t t;

    /*vypocte napeti tak, aby odpovidaly rozdily*/
    *u1=uc;
    *u2=uc+ub;
    *u3=0;

    /*najde zaporne napeti*/
    t=min(*u1,*u2,*u3);

    /*dorovna zaporna napeti na nulu*/
    *u1-=t;
    *u2-=t;
    *u3-=t;
}
```

Všechny tři kroky jsou provedeny takto:

```
1  {
2      uint32_t pos;
3      int32_t sin, cos;
4      int32_t alpha, beta;
5      int32_t ua,ub,uc;
6      int32_t ia,ib,ic;
7      int32_t u1,u2,u3;
8      pos=rps.index_dist;
9
10     pos+=960; /*zarovnani faze 'a' s osou 'alpha'*/
11
12     /*pro výpočet sin a cos je pouzita 32-bit cyklicka logika*/
13     pos*=4294967;
14     pxmc_sincos_fixed_inline(&sin, &cos, pos, 16);
15
16     dq2alphabeta(&alpha, &beta,0,duty, sin, cos);
17     alpha>>=16;
18     beta>>=16;
19
20     alphabeta2pwm3(&ia,&ib, &ic,alpha,beta);
```

Na tomto místě by mělo být implementováno zpětnovazební řízení jednotlivých proudů za použití jejich naměřených hodnot. Pro jednoduchost je ale použita jen dopřednovazební regulace vycházející z rovnic (8) až (10):

```
21        ua=ia;
22        ub=ib;
23        uc=ic;
```

Nyní zbývá přepočítat fázová napětí a podle jejich hodnot nastavit šířky plnění PWM:

```
24        transDelta(&u1,&u2, &u3,ub,uc);
25
26        rps.pwm1=(uint16_t)u1;
27        rps.pwm2=(uint16_t)u2;
28        rps.pwm3=(uint16_t)u3;
29    }
```

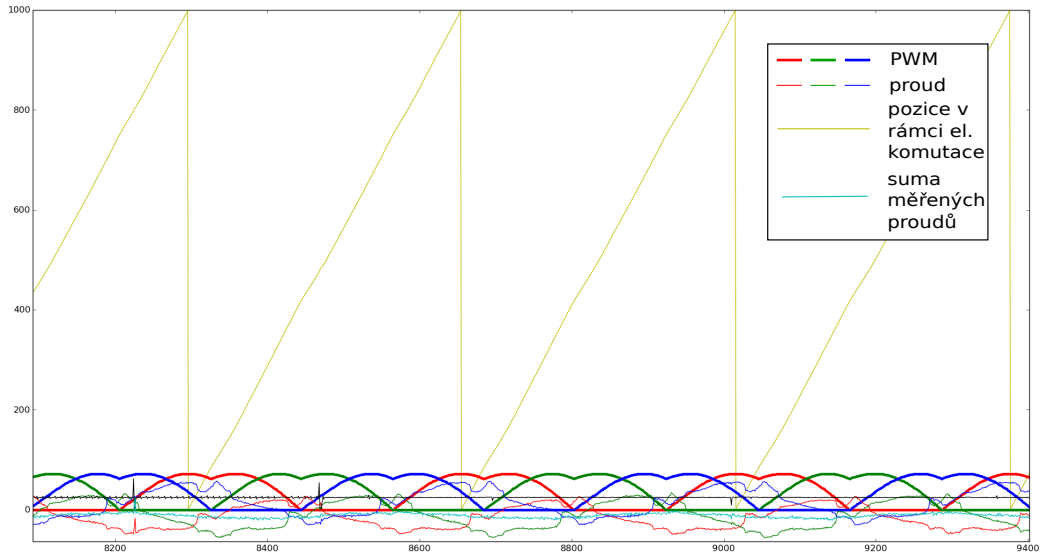
4.9 Řízení rychlosti

Pro regulaci rychlosti motoru je používáno vektorové řízení. V rámci testovaci aplikace byl navržen jednoduchý PI regulátor pomocí nástroje *rltool* z prostředí MATLAB. Zdrojový kód regulátoru se nachází v soubru `controllers.c`. Naměřené průběhy této regulace jsou uvedeny v sekci 5.2. Jako vhodná rychlost pro testování byla zvolena rychlost 100 IRC za 50 ms a aktuální rychlost je také zobrazená jako počet pulzů za shodnou dobu.

Kapitola 5
Závěr

5.1 Záznamy průběhů PXMC

S využitím knihovny PXMC byly zachyceny některé průběhy změřených proudů jednotlivých fází a dalších veličin. Průběh při nastavení pevné šířky plnění příkazem `PWMA:1000` je na obrázku 5.1.



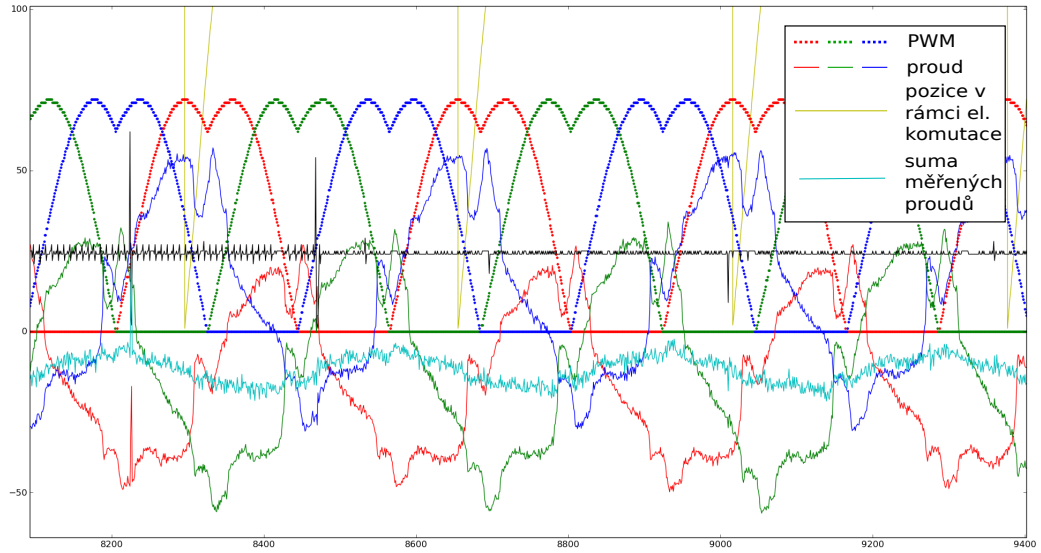
Obrázek 5.1. Záznam průběhu při konfiguraci PWMA:1000 bez zatížení. Na svislé ose je vynesena pozice IRC, v grafu je vyznačena žlutozelenou barvou.

Detail průběhů PWM a proudů z obrázku 5.1 je na obrázku 5.2. Z tohoto obrázku je také patrné, že jeden z měřených proudů má o něco menší zesílení (suma změřených proudů není nulová). Následná analýza ukázala jako pravděpodobné, že při měření proudů dochází k nežádoucímu magnetickému rušení.

Senzory obsahují pro měření proudu jen necelý jeden závit zařazený mezi výkonovými piny. Obvody sice jsou odstíněné podle manuálu proti elektrostatickému rušení, ale ne proti magnetismu. Zde nastává problém, že proudy dalších dvou výkonových výstupů prochází v blízkosti obvodu příslušného měřené fázi . Magnetické pole kolem „přímého“ vodiče se tak značně promítá do blízkého obvodu. Přitom podle geometrie jednotlivých vodičů je ovlivnění různé.

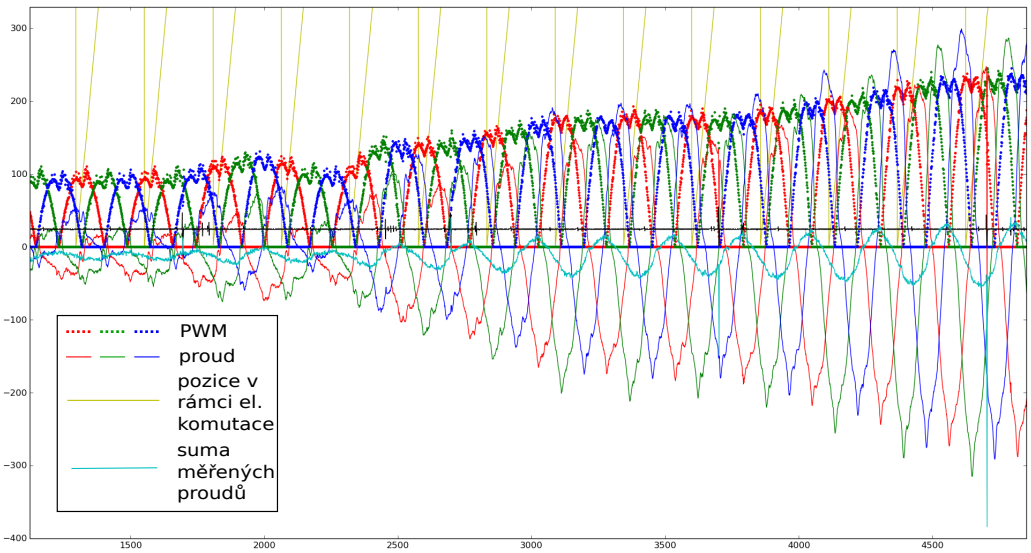
Aby byly takto naměřené hodnoty použitelné pro regulaci, bylo by nutné provést analýzu metody kalibrace, která by eliminovala vzájemné ovlivnění. Přitom vzhledem k tomu, že soustava tří proudů je přeurčená, viz rovnice (1), postačují pouze dvě přesně

měřené hodnoty. Třetí hodnotu, byť změřenou nepřesně, je možné za jistých předpokladů, použít ke zpřesnění výpočtu. V rámci této práce nebyla kalibrace z časových důvodů provedena.



Obrázek 5.2. Záznam průběhu při konfiguraci PWMA:1000 bez zatížení. Tečkované hodnoty jsou šířky plnění PWM. Slabší, modré, červené a zelené průběhy jsou naměřené proudy.

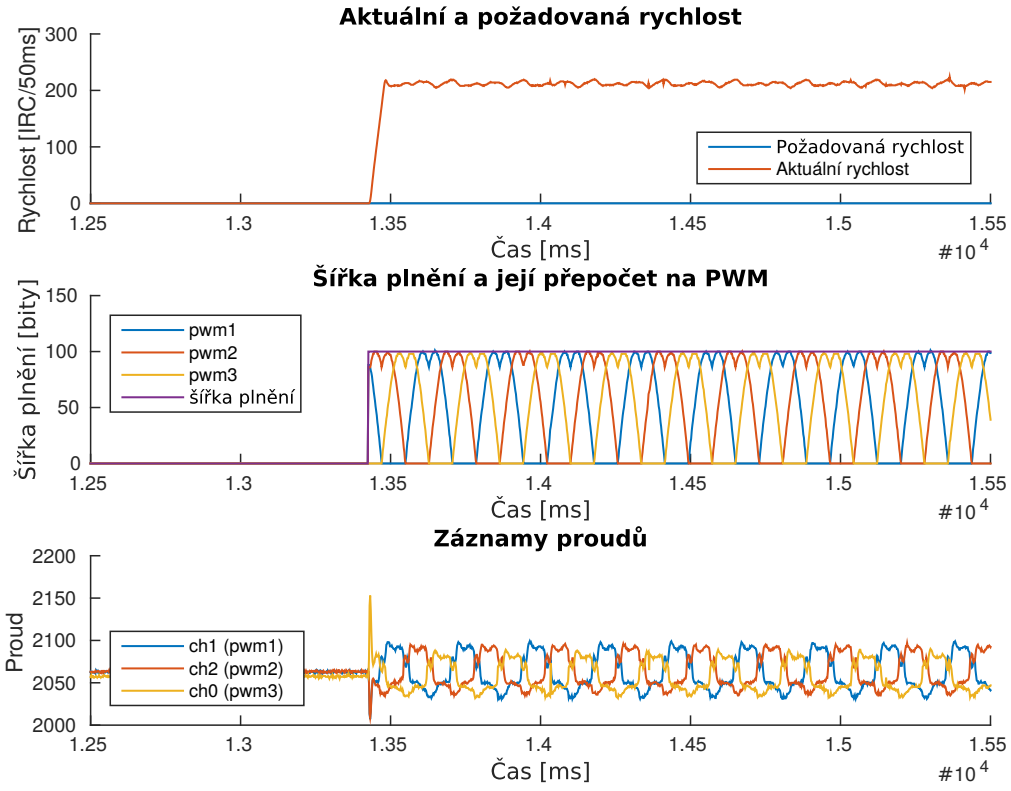
Na obrázku 5.3 je zachycen průběh při řízení rychlosti. Byl použit příkaz `SPDA:1000`. Do hodnoty 1500 na vodorovné, časové ose je motor nezatížen. Dále od této hodnoty pak dochází k postupnému narůstání zátěže.



Obrázek 5.3. Záznam průběhu při konfiguraci SPDA:1000 a následném zatížení

5.2 Záznamy průběhů testovací aplikací

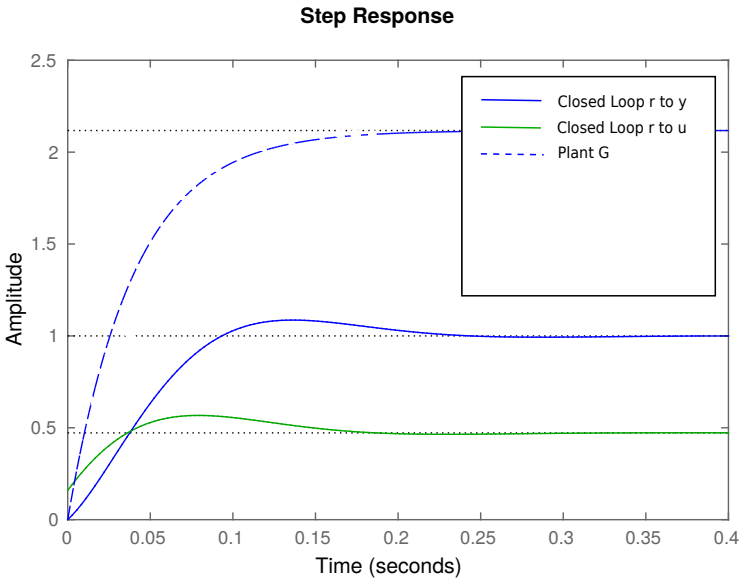
Možnost vytvořit záznam o průběhu řízení je implementována i v testovací aplikaci, v souboru `logs.c`. Průběžné hodnoty rychlosti, měřených proudů a šířek plnění PWM jsou vzorkovány jednou za 2 cyly řídicí smyčky. Následně jsou uloženy do souboru `logs.log` a v textové formě zpracovány v programu MATLAB. Na obrázku 5.4 je odezva systému na změnu pevně volené šířky plnění z 0 na 100.



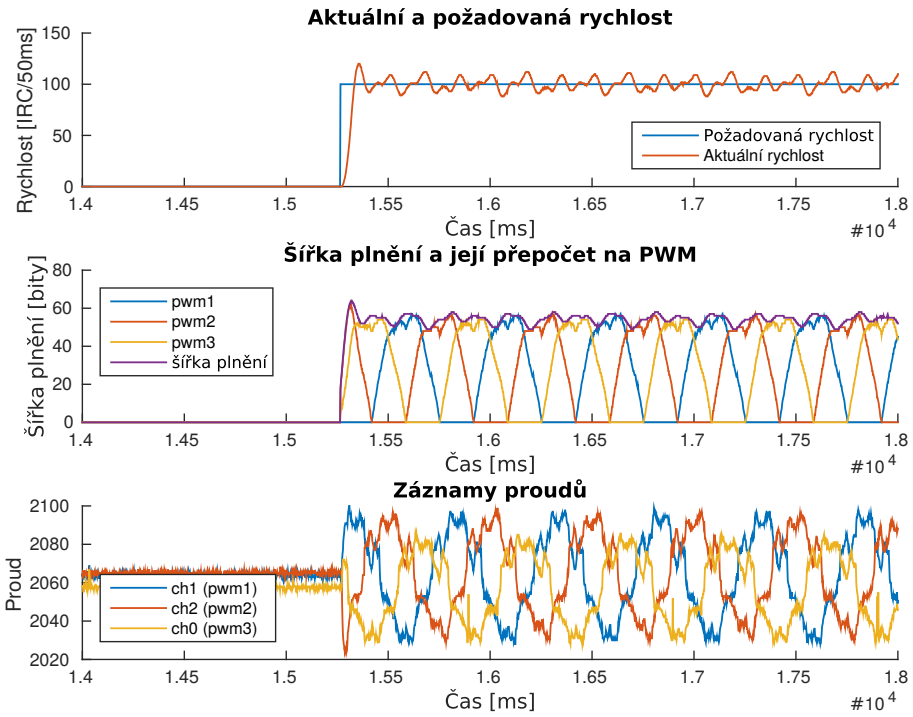
Obrázek 5.4. Záznam průběhu při odezvě na změnu pevné šířky plnění. Konfigurace duty:100.

V rámci testovací aplikace je implementováno poziční řízení s P regulátorem a řízení rychlosti s využitím PI regulátoru.

Pro návrh jednoduchého PI regulátoru rychlosti je použit nástroj *rltool* z prostředí MATLAB. Jako model systému je pro první přiblížení zvolena aproximace systémem prvního řádu, tedy proložení exponenciálou. Odezva na jednotkový skok nástroje *rltool* je na obrázku 5.5. Záznam reálného průběhu řízení z požadované rychlosti 0 na 100 je na obrázku 5.6.

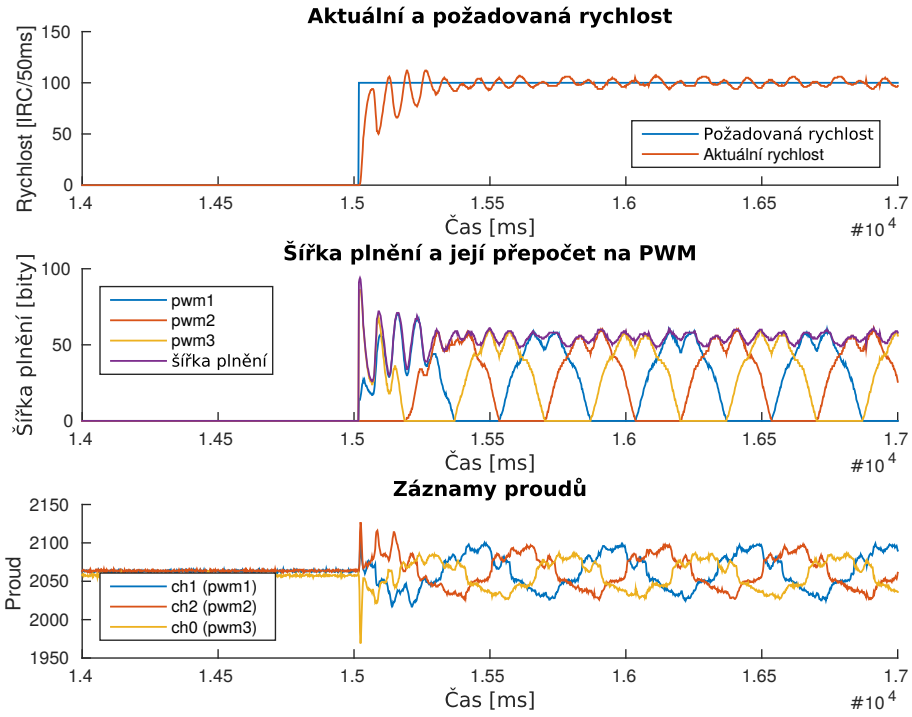


Obrázek 5.5. Grafický výstup nástroje *rltool* prostředí MATLAB. Odezva na jednotkový skok.

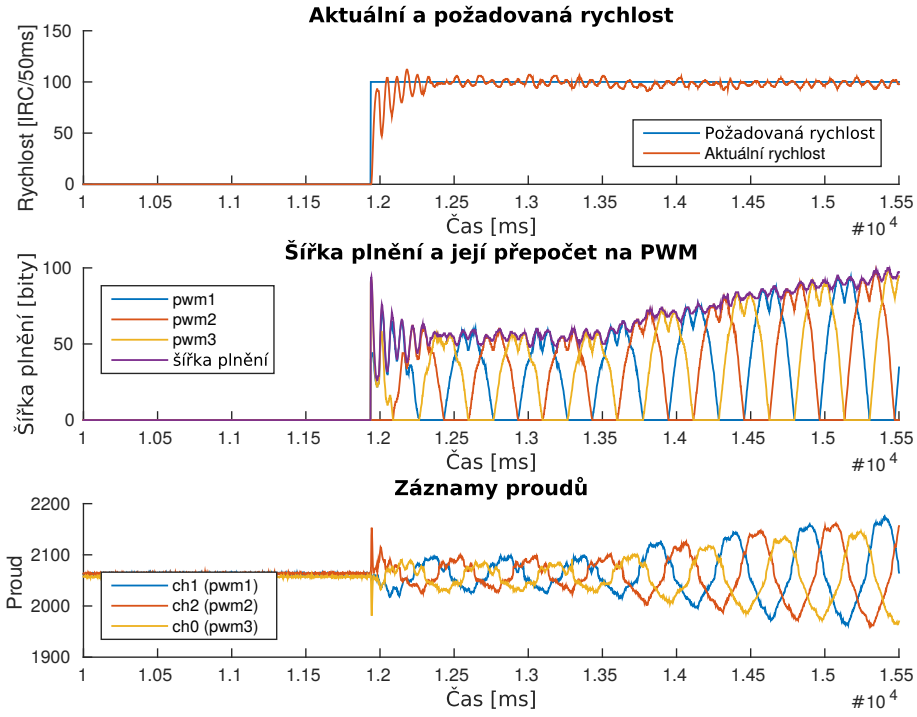


Obrázek 5.6. Záznam průběhu řízení rychlosti při odezvě na změnu požadované rychlosti z 0 na 100. Konfigurace spd:100. Regulutátor navržený pomocí *rltool*.

Hodnoty PI regulátoru jsou následně upraveny tak, aby bylo zmírněno kmitání systému v „ustáleném“ stavu. Na obrázku 5.7 je zaznamenán průběh řízení rychlosti upraveného regulátoru. Konkrétně pak odezva na změnu požadované rychlosti z 0 na 100. Obrázek 5.8 pak zachycuje řízení rychlosti na hodnotu 100 s postupně se zvyšující zátěží.



Obrázek 5.7. Záznam průběhu řízení rychlosti při odezvě na změnu požadované rychlosti z 0 na 100. Konfigurace spd:100. Upravený regulátor.



Obrázek 5.8. Záznam průběhu řízení rychlosti při zvyšujícím se zatížení. Konfigurace spd:100. Upravený regulátor.

5.3 Závěrečné hodnocení

Cílem práce je realizovat řízení pro synchronní bezkartáčové motory s využitím jednodeskového počítače Raspberry Pi a Linuxu. To vše při použití rozšiřující jednotky intergrující výkonový hardware a FPGA obvod.

V první části práce jsou prezentovány základy vlastní konstrukce synchronního bezkartáčového motoru, princip jeho činnosti a základní matematický popis. Jako vhodné matematické nástroje jsou zde zvoleny Clarkova a Parkova transformace.

Další kapitola podrobně popisuje použitý hadrware. Je zde věnován prostor motoru a jeho parametrům, rozšiřující jednotce nebo RPi a jeho vlastnostem. Zvláštní pozornost se dostala nasazení víceúčelového operačního systému GNU/Linux, a jeho RT-rozšíření, pro realizaci zpětvovazebního řízení. Bylo tak potvrzeno, že tento systém je použitelný a vhodný pro zadanou práci a je tak možné ho doporučit k podobným účelům.

Následující kapitola se již plně věnuje realizaci projektu. V úvodu je uvedeno, jaké požadavky klade řízený motor na frekvenci řídicí smyčky. S ohledem na možnosti procesorové jednotky a vlastnosti řízené soustavy je vybrána fekvence 1kHz. Realizace projektu prokázála, že řízení za této frekvence je možné.

Jako minimální objem dat přenášených mezi RPi a rozšiřující jednotkou v každém cyklu řídicí smyčky bylo stanoveno 128 bitů. Pro jejich přenos byl vybrán protokol SPI. Pro budoucí práci je možné doporučit využití právě tohotu protokolu. Zajímavou alternativou je ale použití protokolu SSI, který v sobě integruje přenos hodinového signálu.

Pro možnou realizaci plného momentového řízení a řízení bez polohových senzorů (sensorless control) je nezbyná znalost proudů, protékajících vinutím statoru. Jejich snímaní je prováděno s využitím Hallova jevu. Napětí, takto získané je převáděno 12-bitovým AD převodníkem. Zde bylo provedeno zpřesnění snímání tohoto napětí. Nyní je místo jedné odečtené hodnoty posílán řídicí jednotce jejich výběrový průměr. Toto řešeni vedlo k redukci šumu a tím krozšíření přesnosti naměřených hodnot.

Vlastní sekce je věnována implementaci požadavků a programovým konstrukcím, které musejí být dodrženy v řídicí aplikaci využívající RT-rozšíření jádra Linux. Lze konstatovat, že bez jejich naplnění nelze uživatelské aplikaci garantovat deterministické chování. Samotná aplikace RT-rozšíření by tak vedla pouze k prodloužení průměrného času odezvy na vnější událost, aniž by přinášela jakékoli výhody.

Pro výukové účely jsou implementovány dvě jednoduché metody elektronické komutace. Při použití komutace využívající jen výstup z Hallových sond dochází k nežádoucím vibracím motoru. Jedná se ale o jedinou možnost, pokud nemáme k dispozici přesnou hodnotu natočení rotoru v rámci celého rozsahu otáčení nebo dostatečně přesné měření proudu.

S přesnou znalostí polohy je možné provádět komutaci založenou na fázovém posunu všech fází o 120° a realizovat tak jednoduché napěťové řízení. Toto je pak možné rozšířit a s využitím matematických postupů, popsaných výše, implementovat vektorové řízení. V rámci vektorového řízení bylo realizováno řízení rychlosti a polohy s využitím PI a P regulátorů.

Literatura

[1] Radek Mečiar. *Řízení motorů s deskou Raspberry Pi a Linuxem*. 2014. http://support.dce.felk.cvut.cz/mediawiki/images/1/10/Bp_2014_meciar_radek.pdf.

[2] T. Brown Forbes. *Engineering System Dynamics*. Edition 2 vydání. Taylor and Francis , 2007 . ISBN 0-8493-9648-4.

[3] Freescale Semiconductor Inc. *Sensorless PMSM Field-Oriented Control*. 2014. http://cache.freescale.com/files/microcontrollers/doc/ref_manual/DRM148.pdf.

[4] Freescale Semiconductor Inc. *PMSM Vector Control with Quadrature Encoder on Kinetis*. 2012. http://cache.freescale.com/files/microcontrollers/doc/ref_manual/DRM128.pdf.

[5] Martin Meloun. *FPGA Based Robotic Motion Control System*. 2014. https://dspace.cvut.cz/bitstream/handle/10467/23347/F3-DP-2014-Meloun-Martin-prace.pdf?sequence=3.

[6] Pehong Chen. *Performance Comparison of Permanent Magnet Synchronous Motor and Induction Motor for Cooling Tower Application*. 2012. http://ijetae.com/files/Volume2Issue8/IJETAE_0812_27.pdf.

[7] Erwan Simon a Texas Instruments Inc. *Implementation of a Speed Field Oriented Control of 3-phase PMSM Motor using TMS320F240*. 1999. http://www.ti.com/lit/an/spra588/spra588.pdf.

[8] Freescale Semiconductor Inc. *Permanent Magnet Synchronous Motor Control*. 2012. http://cache.freescale.com/files/industrial/doc/brochure/BBPRMMAGSYNART.pdf.

[9] Ali Ahmed Adam a Kayhan Gulez. *Torque Control of PMSM and Associated Harmonic Ripples* . 2011. http://cdn.intechopen.com/pdfs/13716/InTechTorque_control_of_pmsm_and_associated_harmonic_ripples.pdf.

[10] M. Ouassaid, M. Cherkaoui, A. Nejmi a M. Maaroufi. *Nonlinear Torque Control for PMSM: A Lyapunov Technique Approach* . http://waset.org/publications/12093/nonlinear-torque-control-for-pmsm-a-lyapunov-technique-approach.

[11] ARM Holdings plc. *ARM1176JZF-S Technical Reference Manual*. 2009. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf.

[12] ARM Holdings plc. *Cortex-A7 Floating-Point Unit*. 2012. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0463d/DDI0463D_cortex_a7_fpu_r0p3_trm.pdf.

[13] Pavel Píša Ing. Ph.D. *GNU/Linux, rychlost odezvy a výuka řízení*. 2015.

[14] Paul E. McKenney. *A realtime preemption overview*. 2005.
http://lwn.net/Articles/146861/.

[15] Ayeh E., K Agbedanu, Y. Morita, O. Adamo a P. Guturu. *FPGA Implementation of an 8-bit Simple Processor*.
http://ee.unt.edu/public/adamo/2910_spring_2009/eric.pdf.

[16] Inc. Xilinx. *What is a FPGA?*
http://www.xilinx.com/fpga/.

[17] Bartosiński Roman Ing. Ph.D. *Implementation Methods of LD-RLS with Directional Forgetting for Embedded Systems on a Chip*.
http://support.dce.felk.cvut.cz/mediawiki/images/e/e9/Diz_2011_bartosinski_roman.pdf.

[18] Microsemi Corporation. *IGLOO Low Power Flash FPGAs Datasheet*. 2014.
http://www.microsemi.com/document-portal/doc_download/130694-igloo-low-power-flash-fpgas-datasheet.

[19] Microsemi Corporation. *Libero SoC v11.5 User's Guide*. 2015.
http://www.microsemi.com/document-portal/doc_download/130850-libero-soc-v11-5-user-s-guide.

[20] Texas Instruments Inc. *12-Bit, 4-Channel Serial Output Sampling ANALOG-TO-DIGITAL CONVERTER*. 2008.
http://www.ti.com/lit/ds/symlink/ads7841.pdf.

[21] Mark Brownand David Brownell, Russell King, Grant Likely, Dmitry Pervushin, Stephen Street, Mark Underwood, Andrew Victor, Linus Walleij a Vitaly Wool. *Overview of Linux kernel SPI support*.
https://www.kernel.org/doc/Documentation/spi/spi-summary.

[22] *Linux Programmer's Manual IOCTL(2)* . 2015 .
http://man7.org/linux/man-pages/man2/ioctl.2.html.

[23] Mirko Navara. *Pravděpodobnost a matematická statistika*. 2014.
http://cmp.felk.cvut.cz/~navara/psi/PMS_print.pdf.

[24] Theodore Ts'o, Darren Hart a John Kacur. *Real-Time Linux Wiki* .
https://rt.wiki.kernel.org/index.php/Main_Page.

[25] *Linux Programmer's Manual USLEEP(3)* . 2015 .
http://man7.org/linux/man-pages/man3/usleep.3.html.

[26] Scott Lowe. *Calculate IOPS in a storage array* . 2010 .
http://www.techrepublic.com/blog/the-enterprise-cloud/calculate-iops-in-a-storage-array/.

Příloha A
DVD

Součástí přiloženého DVD jsou následující položky:

- Text této práce `Bp_2015_prudek_martin.pdf`.
- Kód VHDL a skripty pro syntézu a programování FPGA obvodu v adresáři `pmsm-control/`.
- Zdrojové kódy testovací aplikace v adesáři `pmsm-control/test_sw/`.
- Skript pro vizualizaci průběhů naměřených testovací aplikací, uložený v souboru `pmsm-control/test_sw/tools/readLogs.m`

Text této práce ve formě zdrojového kódu je dále volně přístupny na serveru *GitHub*. Práci je možné stáhnou příkazem `git clone http://github.com/wargarw/bp`. Vysázení je prováděno příkazem `make`.

Příloha B
Zkratky

- AC — Alternating current - Střídavý proud
- ADC — Analog-to-Digital Convertor
- API — Application Programming Interface - Rozhraní OS pro programování aplikací
- ASIC — Application Specific Integrated Circuit
- BEMF — Back Electromotive Force
- BLDC — Brushless DC electric Motor
- CE — Chip Enable
- CPU — Central processing unit
- DAC — Digital-to-Analog Convertor
- DC — Direct current - Stejnosměrný proud
- DFF — D Flip-Flop
- FPGA — Field Programable Gate Array - Programovatelné hradlové pole
- FPU — Floating-Point Unit
- GPCLK — General Purpose Clock
- GPIO — General Purpose Input Output
- HDL — Hardware description Language - Jazyk pro popis hardware
- HTTP — Hypertext Transfer Protocol
- HW — Hardware - Technické vybavení
- IRC — Incremental Rotary Encoder - Inkrementální rotační snímač
- ISP — In System Programming
- JTAG — Joint Test Action Group
- LSB — Least Significant Bit - Bit s nejnižší hodnotou
- MCU — Microcontroller - Jenočipový počítač
- MISO — Master In Slave Out
- MOSI — Master Out Slave In
- MSB — Most Significant Bit - Bit s nejvyšší hodnotou
- MTD — Memory Technology Device
- OS — Opperating System - Operační systém
- PLL — Phase-locked loop - Fázový závěs
- PMSM — Permanent Magnet Synchronous Motor
- PWM — Pulse-Width Modulation - Pulzně šířková modulace
- RAM — Random Access Memory - Paměť s libovolným přístupem
- RT — Real Time - Aplikace reálného času
- SAR — Successive Approximation Register
- SATA — Serial ATA
- SE — Slave Enable
- SMP — Symmetric Multiprocessing - Symntrický multiprocesing
- SoC — System on a chip
- SPI — Serial Peripheral Interface
- SS — Slave Select

SSH	Secure Shell
SSI	Synchronous Serial Interface
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus - Univerzální sériová sběrnice
VFP	Vector Floating-Point
VHDL	VHSIC Hardware Description Language - Jazyk pro popis velmi rychlých integrovaných obvodů
VHSIC	Very-High-Speed Integrated Circuit - Velmi rychlý integrovaný obvod

Příloha C
Příkazy testovací aplikace

Testovací aplikace po spuštění začne s frekvencí přibližně 1Hz vypisovat dostupné informace o průběhu řízení a stavu motoru i aplikace. Současně je možné zadávat příkazy uvedené v této příloze. Plná verze informace o příkazech se zobrazí po zadání příkazu `help`.

- `start` - Odemkne enable bity v H-můstcích.
- `stop` - Vypne komutaci, pwm a řizení.
- `0` - Vypne komutaci, pwm a řízení.
- `ga:[hodnota]` - Zapne řízení na zvolenou absolutní polohu.
- `duty:[hodnota]` - Nastaví pevnou šířku plnění.
- `spd:[hodnota]` - Zapne řízení na danou rychlost.
- `log` - Spustí nebo uloži záznam hodnot.
- `ao:[hodnota]` - Přenastaví alpha offset.
- print - Zapne nebo vypne pravidelne vypisování hodnot.
- help - Vypne vypisování hodnot a zobrazí tuto nápovědu.
- exit - Bezpečně ukončí program.

Pokud je vyžadováno řízení polohy na hodnotu 10000 IRC, může sekvence příkazů vypadat následovně:

```
start
ga:10000
```

Nastavení pevné šířky plnění 100. Později pak řízení na rychlosti -100:

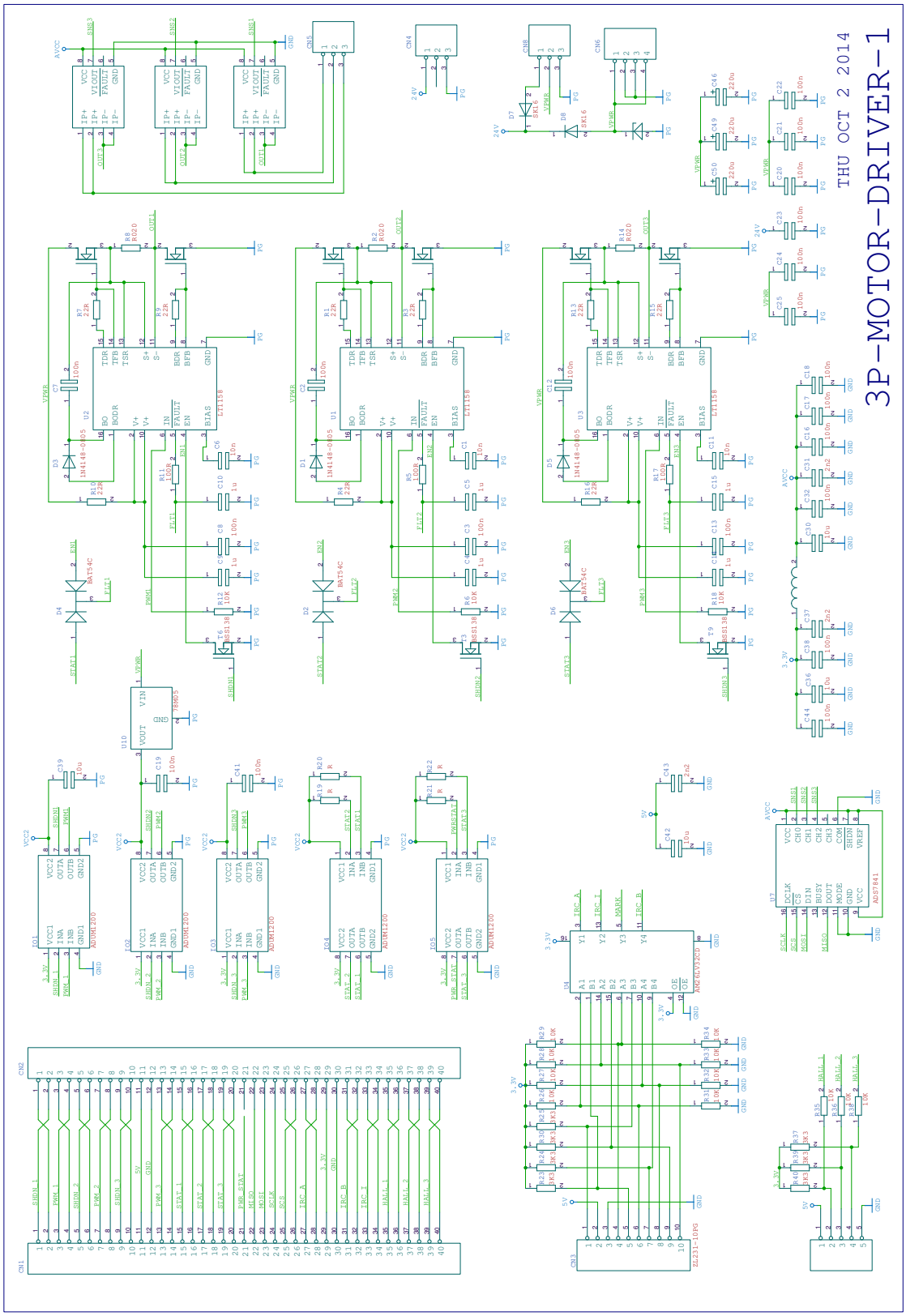
```
start
duty:100
spd:-100
```

Program je ukončen příkazem:

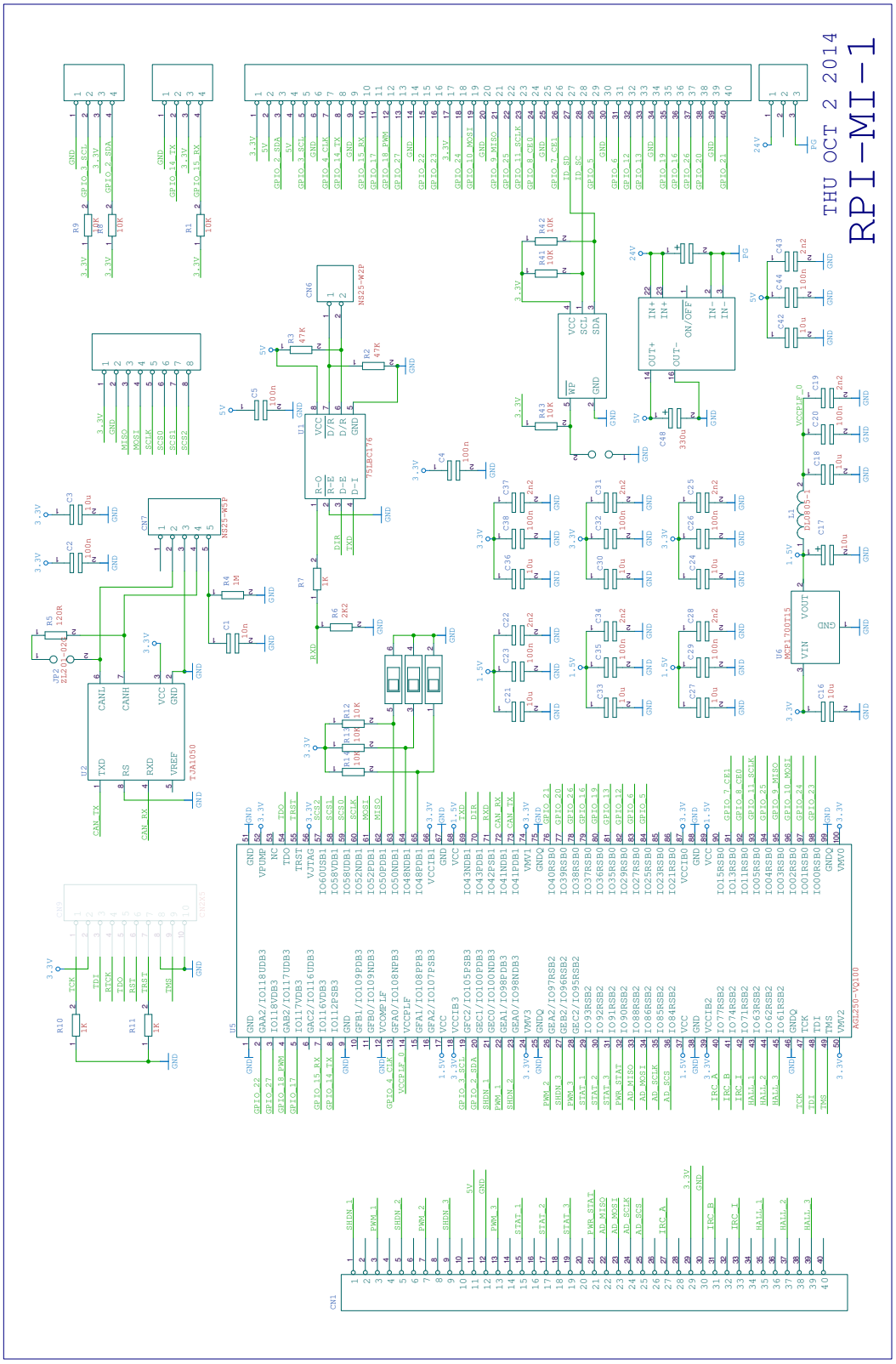
```
exit
```

Poznámka: příkaz `start` je nutné zadávat jen při novém startu aplikace.

Příloha D
Dokumentace výkonového stupně Rpi-Mi-1, PiKRON 2014



Obrázek D.1. Schéma driveru motoru



Obrázek D.2. Schéma propojení RPi a FPGA obvodu