VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS

IP CORE PRO ŘÍZENÍ BLDC MOTORŮ
IP CORE FOR BLDC MOTOR CONTROL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Marek Hráček**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Vojtěch Dvořák**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**
Ústav mikroelektroniky

Student: Bc. Marek Hráček ***ID:*** 164288
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

IP core pro řízení BLDC motorů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vektorovým řízením BLDC motorů a synchronních motorů s permanentním magnetem (PMSM). Prostudujte jednotlivé části řídící struktury a definujte zdroje potřebné pro výpočet regulačního algoritmu. V jazyce VHDL navrhněte modul implementující algoritmus řízení BLDC motorů a ověřte jeho funkčnost pomocí simulace na úrovni RTL. Výsledný IP core bude umožňovat nastavení přesnosti interních výpočtů a modifikaci struktury řízení dle potřeby uživatele. Parametry regulátorů budou uloženy v interní konfigurační paměti a budou modifikovatelné uživatelem bez nutnosti zásahu do kódu. Návrh proveďte s ohledem na efektivní využití dostupných zdrojů v obvodech FPGA a maximální pracovní frekvenci.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 21.5.2019

Vedoucí práce: Ing. Vojtěch Dvořák
Konzultant:

doc. Ing. Lukáš Fujcik, Ph.D.
předseda oborové rady

Abstrakt

Tato diplomová práce pojednává o vektorovém řízení synchronních BLDC a PMSM motorů pomocí FPGA. V první části je popsána základní teorie těchto motorů a jejich řízení. Následně je popsáno vektorové řízení a jeho náležitosti jako αβ a Parkova transformace. Zbytek práce se zabývá samotným návrhem univerzálního regulátoru s nastavitelnou přesností v jazyce VHDL. Data jsou oddělena od výpočetní části, které je prováděno specializovanou aritmeticko-logickou jednotkou. V poslední části je návrh ověřen v simulátoru pomocí modelu PMSM motoru.

Klíčová slova

BLDC, PMSM, vektorové řízení, VHDL, FPGA

Abstract

This diploma thesis is about using vector control (or field-oriented control) of synchronous BLDC and PMSM motors on FPGAs. First part describes basic theory of these motors and how to control them. Then vector control is detailed and its parts as αβ (or Clarke) and Park transformation. Rest of the thesis deals with the design of universal controller with adjustable accuracy in VHDL language. Data is separated from computing part which utilizes custom arithmetic-logic unit. In the last part of the thesis the design is tested in simulator using model of PMSM motor

Keywords

BLDC, PMSM, vector control, field-oriented control, VHDL, FPGA

HRÁČEK, Marek. IP core pro řízení BLDC motorů. Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/119412. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce Vojtěch Dvořák.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma IP core pro řízení BLDC motorů jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **21. května 2019** ……
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Vojtěchu Dvořákovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování práce.

V Brně dne: **21. prosince 2019** ……
podpis autora

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Experimentální část této diplomové práce byla realizována na výzkumné infrastruktuře
vybudované v rámci projektu CZ.1.05/2.1.00/03.0072
Centrum senzorických, informačních a komunikačních systémů (SIX)
operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

Obsah

Seznam obrázků a tabulek

Seznam zkratek

ALU aritmeticko-logická jednotka (arithmetic-logic unit)

BLDC bezkartáčový stejnosměrný motor (brushless direct current)

EDAC detekce a oprava chyb (error detection and correction)

FPGA programovatelné hradlové pole (Field Programmable Gate Array)

IP duševní vlastnictví (Intellectual property)

PID označení regulátorů, obsahují proporcionální, integrační a diferenciální složku

PMSM synchronní motor s permanentními magnety (permanent magnet synchronous motor

RAM paměť s náhodným přístupem (random-access memory)

VHDL popisný jazyk určený k tvorbě digitálních obvodů (VHSIC Hardware Description Language)

VHSIC velmi rychlé integrované obvody (Very High Speed Integrated Circuits)

Úvod

V dnešní době BLDC a PMSM motory – synchronní motory s permanentními magnety, mají mnoho využití. Oproti běžným stejnosměrným nebo indukčním motorům mají mnoho předností. Díky permanentním magnetům mohou být menší, než odpovídající indukční motory. Také umožňují jednodušší řízení (to platí hlavně pro BLDC) pomocí pulzně-šířkové modulace.

Pro efektivnější a přesnější řízení je ale výhodné používat pokročilé metody regulace. Jednou z nich vektorové řízení, které skrz transformace motorových elektrických veličin umožňuje motor ovládat s maximální účinností.

Původně byla popsána pro indukční motory, ale nyní se velmi často používá právě pro synchronní motory. Tato metoda je výpočetně náročná, a proto, hlavně v minulosti, její použití nebylo nejjednodušší. Proto, pokud je v systému použitý obvod FPGA, je výhodné implementovat tuto metodu právě tam.

Tato práce se právě zabývá návrhem regulátoru v jazyce VHDL pro použití v obvodech FPGA. Důraz byl kladen na snadnou použitelnost v různých situacích při řízení různých motorů. Při tvorbě bylo taky bráno na zřetel možné použití ve vesmírných aplikacích, proto je tomu struktura designu do určité míry přizpůsobena.

1 TEORIE

V této kapitole je popsána základní teorie o BLDC motoru a PMSM synchronního motoru s permanentními magnety. Vysvětlen je také vektorový způsob řízení, jehož implementace je smyslem celé práce.

1.1 Motory BLDC a PMSM

Motory BLDC a PMSM jsou konstrukcí podobné indukčním motorům, kdy stator obsahuje několik fázových vinutí, nejčastěji 3, které generují otáčivé magnetické pole. Indukční motory mají v rotoru další vinutí či vodivou klec nakrátko, ve kterých se statorovým magnetickým polem indukuje elektrický proud, které zpětně generuje magnetické pole rotoru. Interakce obou polí generuje točivý moment. Obě pole však nemohou být ve fázi, jinak by indukce zanikla, rotor je proto o tzv. skluz pomalejší. PMSM a BLDC naproti tomu v rotoru obsahují permanentní magnety, které generují magnetické pole stále. Rotor se tak otáčí ve fázi s magnetickým polem statoru, jde tedy o synchronní motory. [1]

Pro rychlost otáčení motoru (v otáčkách za sekundu) platí rovnice (1-1), kde f je dodávaná statorová frekvence a p je počet pólů na rotoru. [2]:

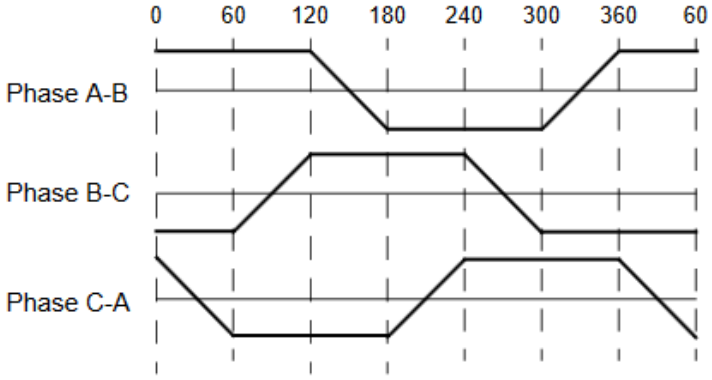
$$N = \frac{2f}{p} \qquad \textbf{(1-1)}$$

Tato rychlost je do určité meze nezávislá na zátěži. Pouze se zvětšuje fázový posun, mezi napětím a magnetickým tokem – tzv. zátěžový úhel. Pokud překročí 90°, dojde ke ztrátě synchronizace a motor se zastaví. [1]

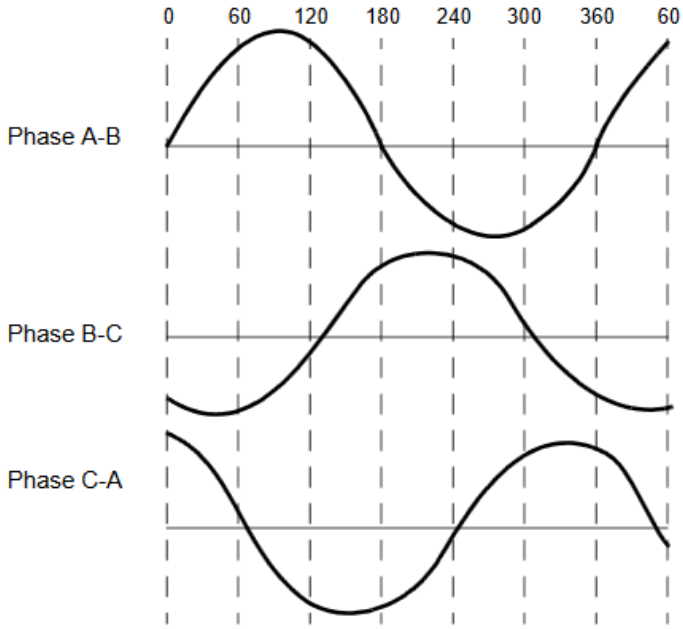
1.1.1 Rozdíl mezi BLDC a PMSM

BLDC a PMSM se liší hlavně způsobem vinutí statoru. BLDC má vinutí soustředěné – každá fáze je koncentrovaná kolem svého magnetického jádra, obvykle plechů z měkké oceli. PMSM má vinutí rozložené, fáze se překrývají.

Důsledkem je jiný průběh zpětně generovaného napětí (BEMF) na statorových cívkách. BLDC má toto napětí lichoběžníkový průběh (Obr. 1-1), PMSM sinusový (Obr. 1-2).



Obr. 1-1: Průběh zpětně indukovaného napětí BLDC motoru [1]



Obr. 1-2: Průběh zpětně indukovaného napětí PMSM [1]

Dalším důsledkem je průběh řídích napětí. BLDC motor je často řízen několika spínači, kdy vždy dvě fáze jsou připojené na napětí a třetí je vypnutá. Lze jej řídit také sinusovým průběhem, PMSM jej však vyžaduje. Dosahuje však hladšího průběhu točivého momentu. [2]

1.2 Vektorové řízení motorů

Tato metoda, také nazývaná vektorově orientované řízení, k řízení elektrických motorů byla popsaná poměrně nedávno – poprvé byla publikována v roce 1971. Jejímu rozšíření do 80. let ale bránila nedostupnost mikroprocesorů, protože je tato metoda

náročná na výpočetní výkon. Určena byla původně pro asynchronní motory, ale použitelná je také synchronní motory s permanentními magnety i s elektricky buzeným vinutím v rotoru. [2]

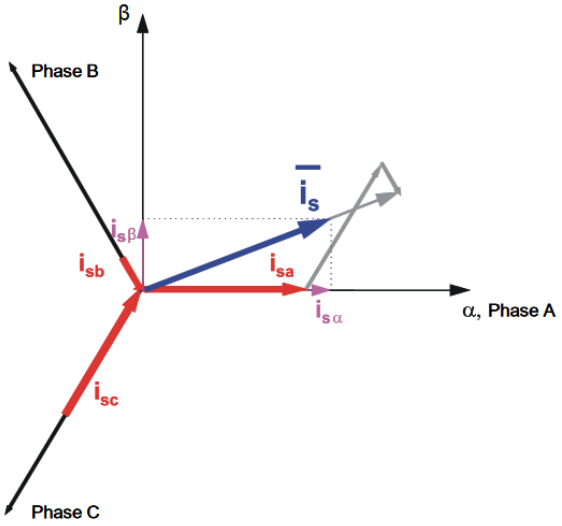
Patří mezi metody řízení změnou napájecího kmitočtu, z nichž pro řízení synchronních motorů je nejpoužívanější skalární řízení, nebo také řízení *U/f*. To vychází z mechanických vlastností motorů, kdy při dodržení konstantního poměru napětí a frekvence daného pro konkrétní motor lze měnit napájecí frekvenci a tím rychlost motoru, aniž by se měnil spřažený magnetický tok ψ, což způsobí, že moment zvratu je konstantní (maximální moment síly, který motor při určitém napětí a frekvenci dokáže vyvinout) a odebíraný proud bude minimální. [3][4]

Vektorové řízení je oproti skalární metodě vždy zpětnovazební systém. Pro správnou funkci vyžaduje měření statorových fázových proudů a dále je nutné dodat orientaci magnetického pole. V původní koncepci byly navrženy Hallovy sondy umístěné u rotoru, které detekovaly toto pole přímo. Tento způsob je však velmi náročný na přesnost a kalibraci sond. Navíc je náročnější údržba. Proto se v dnešní době používá měření rychlosti nebo polohy rotoru, ze které lze spočítat orientaci magnetického pole. Dále jsou tzv. bezsenzorové metody, kdy se orientace magnetického pole počítá z naměřených proudů a sdružených zpětně indukovaných napětí. Výhodou je pak levnější konstrukce, nevýhodou nižší přesnost, hlavně při nižších otáčkách. [2][3]

1.2.1 Princip

Vektorové řízení využívá pojmu prostorový vektor, kdy fázová napětí, proudy a generované magnetické toky lze popsat jako komplexní prostorové vektory, protože rozložení fázových cívek a jejich zapojení je známé. Každou veličinu lze tak popsat jediným vektorem, jenž je daný dvěma souřadnicemi [5]. Např. pro proud tak platí (znázorněno na Obr. 1-3, $i_a = i_{sa}$, $i_b = i_{sb}$, $i_c = i_{sc}$):

$$\boldsymbol{i_s = i_a + i_b + i_c} \qquad \textbf{(1-2)}$$



Obr. 1-3: Proudový prostorový vektor a proudové fáze, ze kterých se skládá [5]

Tento prostorový vektor lze rozdělit na dvě složky, jedna je souhlasná s magnetickým tokem rotoru a druhá je na něj kolmá. Souhlasná složka se podílí na generaci magnetického toku, kolmá generuje točivý moment. [6]

Pro PMSM, pokud jsou zanedbány změny teploty, odporu a indukčnosti fází při provozu, lze napsat pro točivý moment tento vztah:

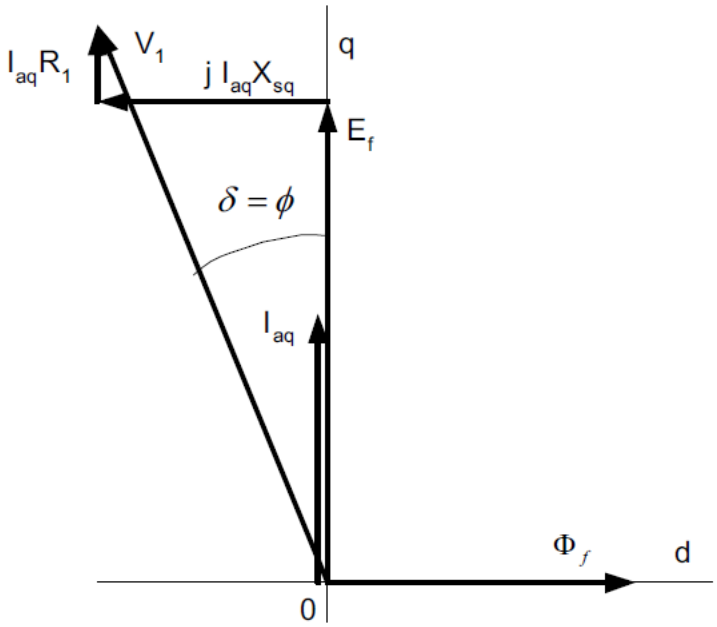
$$T_e = \frac{3p}{2}[\Psi_{PM} i_q + i_d i_q (L_d - L_q)] \qquad \textbf{(1-3)}$$

Kde p je počet pólů, ψ_{PM} je magnetický tok permanentních magnetů a indexy d a q značí směr proudu i a indukčnosti L souhlasně s magnetickým tokem ψ_{PM}, respektive kolmo na něj. Pro motory bez vystouplých pólů navíc platí $L_d = L_q$, proto lze rovnici (1-3) zjednodušit na:

$$T_e = \frac{3p}{2}\Psi_{PM} i_q \qquad \textbf{(1-4)}$$

Z tohoto vztahu vyplývá, že točivý moment závisí pouze na složce proudu i_q. Regulací obou složek zvlášť lze dosáhnout požadovaného točivého momentu při největší efektivitě. Složku i_d lze nastavit pouze na potřebnou hodnotu, aby byl vytvořen potřebný magnetický tok. V případě asynchronních motorů jde o konstantu. U PMSM motorů se i_d reguluje většinou na 0, protože magnetický tok vytvářejí permanentní magnety. Výjimkou jsou motory s vystouplými póly, jak vyplývá z rovnice (1-3) a provoz při vyšších než nominálních otáčkách. [7]

Stav s $i_d = 0$ je zaznačen na Obr. 1-4 (V_1 je napěťový vektor, E_f BEMF, X_{sq} reaktance statoru ve směru q, R_1 odpor statoru, δ zátěžový úhel $I_{aq} = i_q$ a $\phi_f = \psi_{PM}$).



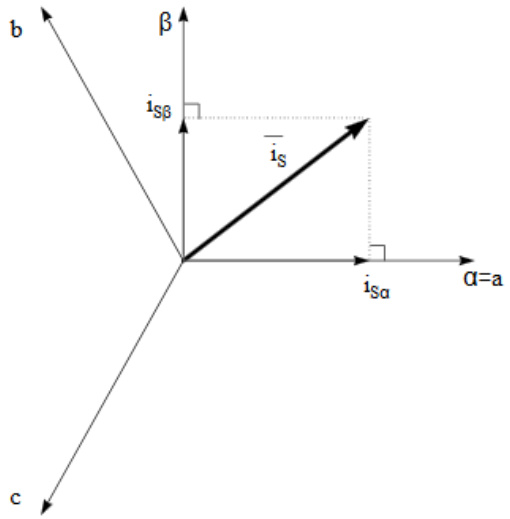
Obr. 1-4: Fázorový diagram statorových veličin pro $i_d = 0$ [1]

Protože prostorové vektory jsou definované na jakémkoli stavu soustavy, platí i pro přechodné stavy, kdy např. dojde k náhlému zatížení. Průběh točivého momentu je plynulejší a do ustáleného stavu se dostane rychleji a efektivněji než v případě skalárního řízení. [2]

Prostorový vektor proudu se při provozu otáčí podle frekvence přiváděných fázových proudů. Složky proudu i_d a i_q však byly definovány podle směru magnetického toku permanentních magnetů. Rámec, ke kterému jsou vztaženy, se otáčí, zatímco tyto složky zůstávají na místě. To je výhodné pro PI regulaci (popsána v kapitole 0), což pro sinusové signály fázových proudů není vhodné. Pro převod mezi stacionárním rámcem v osách *x*,*y* a rotujícím v osách *d*,*q* je ale třeba zavést transformace αβ a Parkovu. [2]

1.3 αβ transformace

Transformace sloužící převedení 3 fázové soustavy do pravoúhlých os *α*,*β* (viz Obr. 1-5). V kontextu motorů si to lze představit jako transformaci 3 fázového motoru na 2 fázový, jehož cívky jsou v pravém úhlu vůči sobě. [8]



Obr. 1-5: αβ transformace [9]

V případě, že osy *a,b,c* mají mezi sebou úhel 120°, platí rovnice:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\frac{2}{3} & -\frac{1}{3} & -\frac{1}{3}\\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\\ c\end{bmatrix} \quad \textbf{(1-5)}$$

Pokud platí podmínka (1-2), lze dále upravit na:

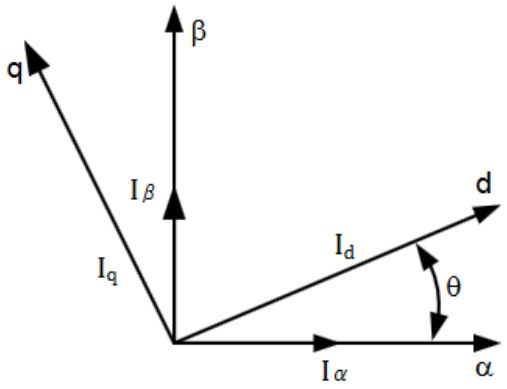
$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}1 & 0\\ \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix} \quad \textbf{(1-6)}$$

Pro transformaci zpět do os *a,b,c* lze použít inverzní αβ transformaci:

$$\begin{bmatrix}a\\ b\\ c\end{bmatrix} = \begin{bmatrix}1 & 0\\ -\frac{1}{2} & \frac{\sqrt{3}}{2}\\ -\frac{1}{2} & -\frac{\sqrt{3}}{2}\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \quad \textbf{(1-7)}$$

1.4 Parkova transformace

Umožňuje převést staticky rámec *α,β* na rotující rámec *d,q*. Lze to znovu znázornit jako transformaci motoru – 2 fázový motor staticky v prostoru se změní na stejný motor, který ale rotuje, čímž se změní průběhy v jeho fázích. Transformace vektorů je znázorněna na Obr. 1-6, θ je úhel mezi oběma rámci.



Obr. 1-6: Parkova transformace [10]

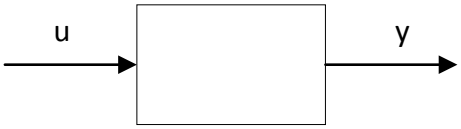
Matematicky jsou dopředná a inverzní Parkova transformace vyjádřeny takto:

$$\begin{bmatrix} d \\ q \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \textbf{(1-8)}$$

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} d \\ q \end{bmatrix} \quad \textbf{(1-9)}$$

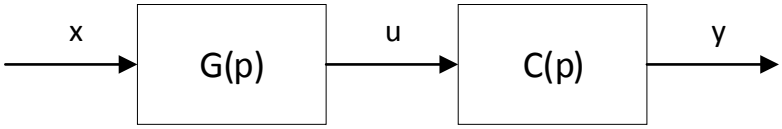
1.5 Regulace PID

Při řízení systému je obvykle požadavkem dosáhnutí určité hodnoty výstupu nastavením vstupů systému. To lze ilustrovat Obr. 1-7, kdy nastavením veličiny *u* se na výstupu objeví určitá hodnota veličiny *y*.



Obr. 1-7: Systém se vstupem a výstupem

Jedním ze způsobů, jak řídit tento systém, je připojení regulátoru k systému v otevřené smyčce. Viz Obr. 1-8, nastavením vstupu regulátoru *x* je aplikován jeho výstup *u* na vstup systému, čímž je obdržen výstup *y*.



Obr. 1-8: Systém s regulátorem v otevřené smyčce

Vyjádřeno v Laplaceově tranformaci, pokud je přenos regulátoru G(p) a přenos regulovaného systému C(p) a platí rovnice (1-10), a tedy výsledný přenos celé soustavy

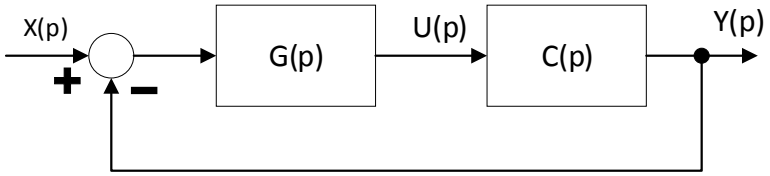
je 1, pak nastavením požadované hodnoty na vstup je okamžitě obdržena stejná hodnota na výstupu. [12]

$$F(p) = G(p)C(p) = 1 \tag{1-10}$$

Takový stav je ideální a prakticky nedosažitelný jak z pohledu charakterizace regulovaného systému, tak implementace regulátoru. Tímto řešením také nelze reagovat na změny regulovaného systému v čase. Proto se to používá v aplikacích, kde není problémem menší přesnost a pomalejší časová odezva. Příkladem může být dříve zmíněné skalární řízení *U/f* (viz 1.2).

Pro překonání těchto nedostatků se proto často používá zpětnovazební zapojení (viz Obr. 1-10), kdy jsou oba bloky provázány tak, že regulátor svůj výstup neodvozuje pouze z požadované hodnoty, ale také z výstupu regulovaného systému. Přenosová funkce je daná rovnicí:

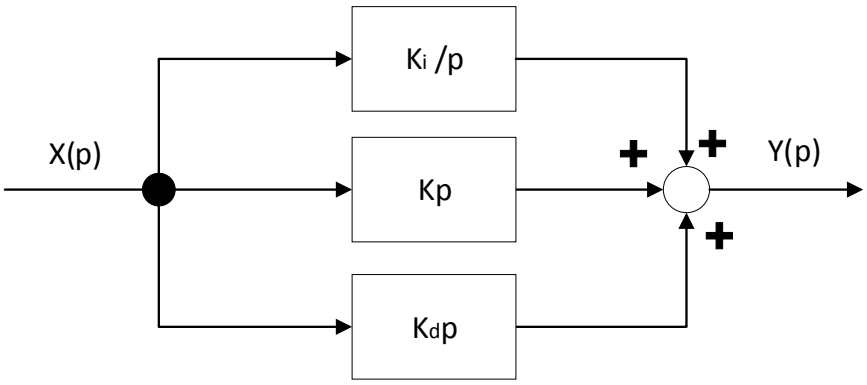
$$F(p) = \frac{C(p)}{1 + C(p)G(p)} \tag{1-11}$$



Obr. 1-9: Systém s regulátorem ve zpětné vazbě

Nejpoužívanější z nich je PID regulátor, který pracuje pouze nad odchylkou mezi požadovanou a skutečnou hodnotou výstupu regulovaného systému. Na tu aplikuje kombinaci 3 členů – proporciálního, integrálního a derivačního (viz (1-12) a Obr. 1-10). Správným nastavením zesílením jednotlivých prvků lze docílit rychlého ustálení s nízkým překmitem, nesprávným se soustava může rozkmitat. [11][12]

$$H(p) = K_p + \frac{K_i}{p} + K_d p \tag{1-12}$$

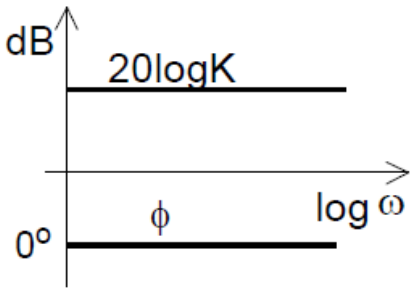


Obr. 1-10: Schéma obecného PID regulátoru

1.5.1 Proporcionální

Zesilovací člen – přenosová funkce je pouze konstanta, viz rovnici (1-13) a frekvenční a fázovou charakteristiku (Obr. 1-11). Pro reálné zesilovače by mělo platit, že jejich časová konstanta je mnohem menší než zbytku soustavy, a můžeme ji zanedbat. Přímo reaguje na chybu – čím větší zesílení, tím větší reakce. Tím se soustava stává více kmitající. Problémem tohoto členu je nulová reakce při nulové chybě. Díky tomu proporcionální člen nedokáže nikdy doregulovat přesně požadovanou hodnotu. [11][12]

$$H(p) = K \tag{1-13}$$

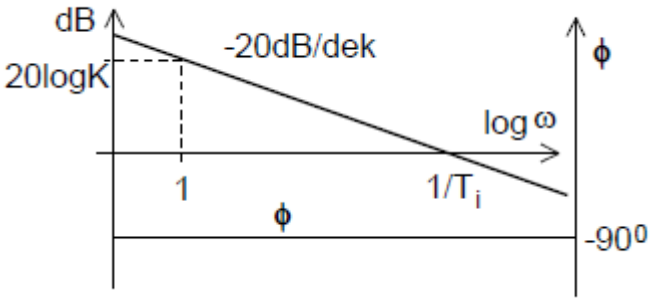


Obr. 1-11: Frekvenční a fázová charakteristika proporcionálního členu [11]

1.5.2 Integrální

Integrační člen, viz přenosová funkce (1-14) (T_i je integrační konstanta) a frekvenční a fázová charakteristika Obr. 1-12, sčítá chybu a reaguje tedy podle uplynulého průběhu. Obecně reaguje na změnu pomaleji než proporcionální člen, ale na druhou stranu dokáže chybu vykompenzovat úplně. [12]

$$H(p) = \frac{K}{p} = \frac{1}{T_i p} \tag{1-14}$$

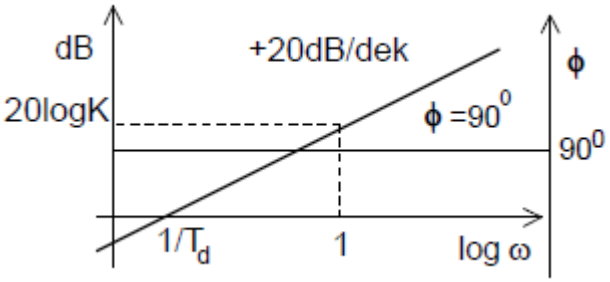


Obr. 1-12: Frekvenční a fázová charakteristika integrálního členu [11]

1.5.3 Derivační

Derivační člen, viz přenosová funkce (1-15) (T_d je derivační časová konstanta) a frekvenční a fázová charakteristika Obr. 1-13, jak je podle názvu patrné, derivuje vstupní chybu. Reaguje tedy na rychlost její změny. Pomáhá tak tlumit oscilace. Sám o sobě však nedokáže vykompenzovat chybu. Navíc je velmi náchylný na vysokofrekvenční šum. Používá se spíše v méně dynamických systémech. [12]

$$H(p) = pK = pT_d \tag{1-15}$$

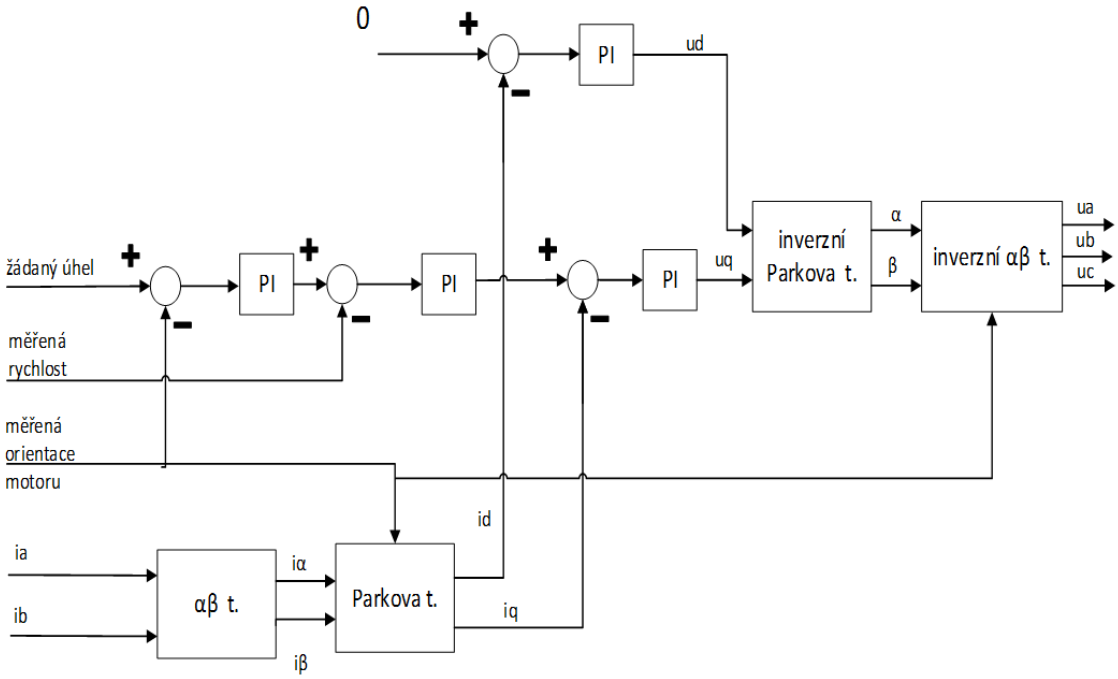


Obr. 1-13: Frekvenční a fázová charakteristika derivačního členu [11]

2 POPIS REGULAČNÍ SOUSTAVY

V této kapitole je popsána řídící struktura, která vychází z teorie vektorového řízení představeného v předchozí části. Jsou použity αβ a Parkova transformace pro převody vstupů na hodnoty, které lze ovládat PI regulátorem.

Podrobně je celý algoritmus znázorněn na Obr. 2-1. Regulační soustava zahrnuje tři vnořené smyčky pro řízení proudu, rychlosti a pozice motoru. Vstupy regulačního algoritmu je úhel natočení motoru, rychlost otáčení a trojfázové proudy motoru.



Obr. 2-1: Schéma regulačního algoritmu

2.1 Proudová smyčka

Proudové řízení probíhá v *d*,*q* doméně (popsaná v kapitole 1.2). Měřené třífázové proudy jsou transformovány pomocí αβ transformace a Parkovy (dq) transformace.

Vstupem Iq regulace (výstup rychlostní smyčky) je požadovaný proud i_q, který regulátor proudu Iq přepočítá na napětí u_q. Složka *d* je řízena samostatným PI regulátorem na referenční konstantu (viz kapitola 1.2.1) a generuje hodnotu u_d. Pro přepočet mezi rotujícím rámcem *d*,*q* a 3fázovou soustavou jsou použity zpětná αβ a Parkova transformace. Výstupem proudové smyčky je pak třífázové napětí u_a, u_b a u_c, která jsou aplikována na motor.

2.2 Rychlostní smyčka

Řízení rychlosti otáčení je provedeno samostatnou rychlostní smyčkou. Jejím vstupem je požadovaná rychlost (výstup z regulátoru pozice) a aktuální rychlost otáčení.

Stejně jako v proudové smyčce, regulátor zahrnutý v rychlostní smyčce je typu PI. Výstupem rychlostního regulátoru je požadovaný proud i_q, který je úměrný požadovanému momentu, resp. zrychlení.

2.3 Poziční smyčka

Řízení motoru na pozici je primárním úkolem popsaného regulační algoritmu. Poziční smyčka je tedy vnější smyčka celé soustavy a jejím vstupem je požadovaná pozice a informace o aktuálním úhlu natočení motoru. Regulátor poziční smyčky je taktéž typu PI a je vhodný především pro řízení motoru při měnící se požadované pozici. Poziční regulátor upravuje hodnotu požadované rychlosti pro dosažení požadované pozice v co nejkratším čase dle regulačních parametrů.

2.4 PI regulátor

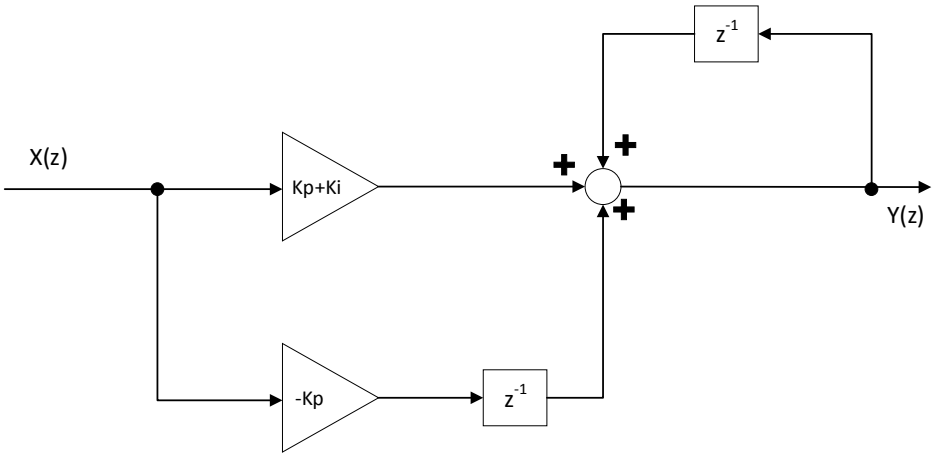
Jak bylo popsáno v předchozích kapitolách, v tomto návrhu bude použit pouze regulátor typu PI. Derivační složka je pro rychlé a dynamické systémy obtížně použitelná, je náchylná na vysokofrekvenční šum, který je proto obvykle nutné filtrovat [12].

Přenosová funkce PI regulátoru v Z-transformaci je daná takto [11]:

$$H(z) = \frac{Y(z)}{X(z)} = K_p + \frac{K_i}{1 - z^{-1}}$$
$$K_p = K - \frac{KT}{2T_i} \qquad K_i = \frac{KT}{T_i} \tag{2-1}$$

Kde K je zesílení, Ti integrační časová konstanta a T je vzorkovací frekvence. Úpravou lze získat rovnici pro výstup (2-2), ze které lze již nakreslit schéma obvodu – Obr. 2-2.

$$Y(z) = X(z)\left(K_p + K_i - K_p z^{-1}\right) + Y(z)z^{-1} \tag{2-2}$$

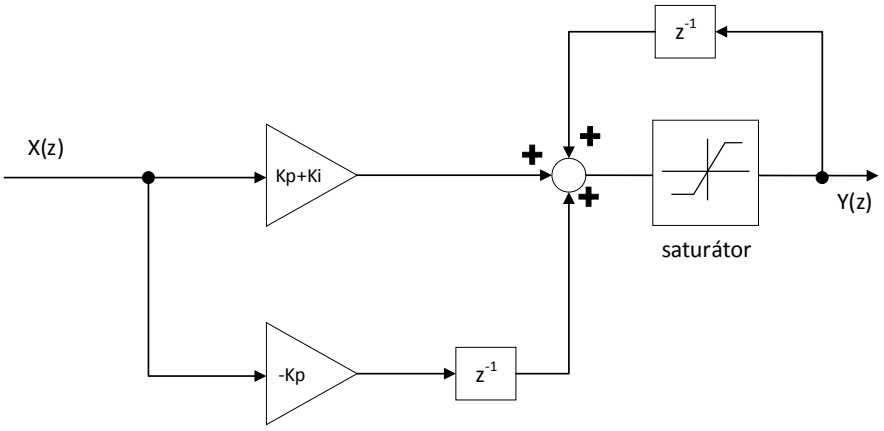


Obr. 2-2: Schéma PI regulátoru

Reálné vstupní hodnoty řízeného systému jsou omezeny (napájecí napětí, mechanická pevnost apod.), proto je třeba omezovat výstup regulátoru, aby nedošlo k šíření nebezpečných hodnot do řízeného systému.

Při zavedení této nelinearity se ale projevuje tzv. integrační windup. Pokud je výstup regulátor omezen saturátorem, rychlost změny vstupní chyby se změní. Integrátor při tom však stále akumuluje chybu. Po návratu do lineární oblasti pak dojde k výraznému zpoždění reakce regulátoru, kdy se musí nahromaděná chyba nejprve vyprázdnit. Mezi běžná řešení patří podmíněná integrace, nebo zpětná vazba upravující vstup integrátoru. [13]

V navržené struktuře je použitá další možnost, jak zabránit tomuto nežádoucímu jevu. Saturátor je vložen před odbočku zpětné vazby integrátoru (Obr. 2-3).



Obr. 2-3: Schéma PI regulátoru se saturátorem

3 NÁVRH IMPLEMENTACE

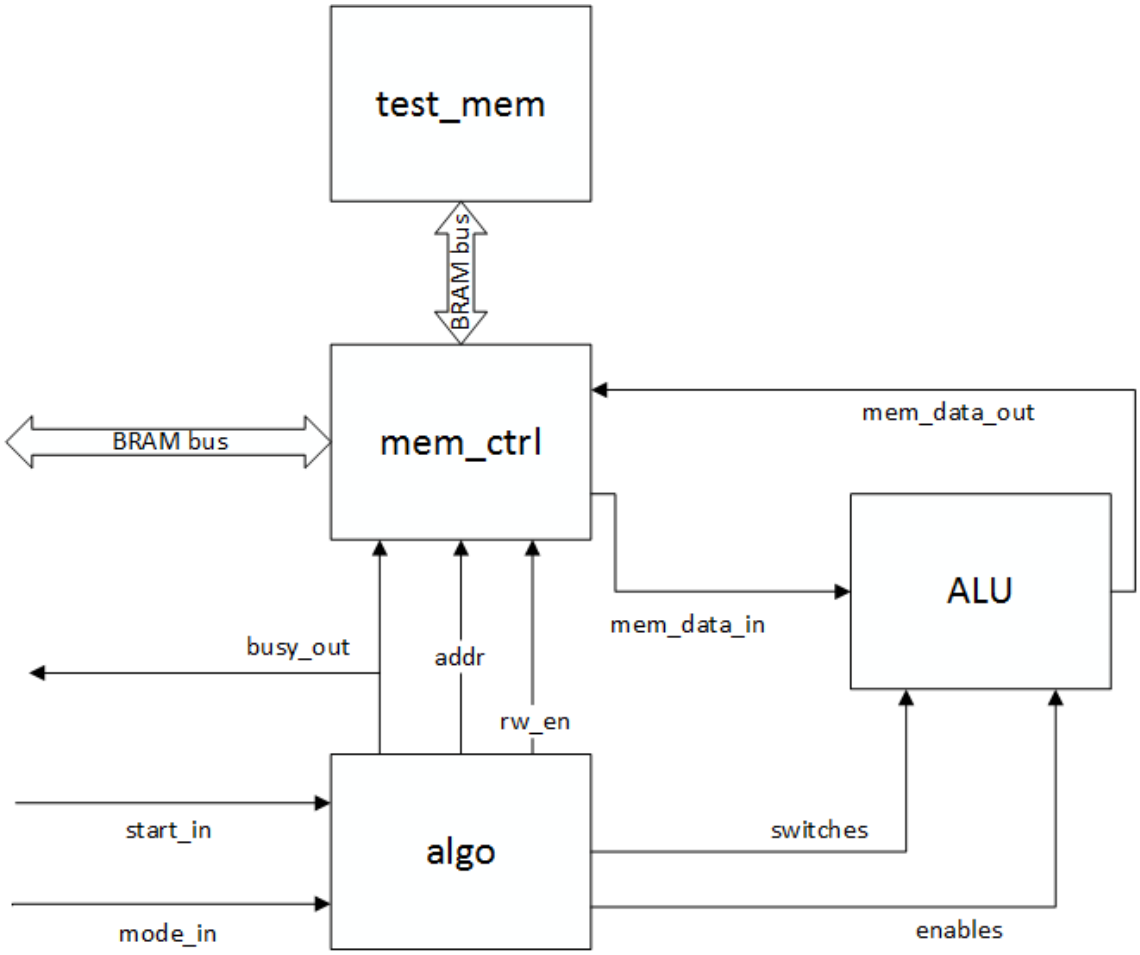
V této kapitole je popsán návrh celého regulátoru, využívající vektorové řízení. Z důvodu požadavků na univerzalitu použití je návrh široce parametrizovatelný. Proměnné i konstanty jsou uloženy v interní paměti, nejsou přímo součástí programovatelné logiky a lze je případně měnit i při běhu FPGA. Popis samotného výpočetního postupu je zhotoven tak, aby jeho případná změna nepodmiňovala zásah do ostatních částí designu.

Celková struktura vychází z podmínek vektorového řízení – regulační frekvence řízení se obvykle pohybuje kolem 10 až 15 kHz [2], zatímco frekvence FPGA v desítkách MHz. Díky tomu je možné priorizovat nižší využití prostředků hradlového pole před rychlostí výpočtu.

Není tak třeba každý krok algoritmu implementovat jako vlastní část komponenty, ale lze sdílet prostředky mezi sebou. Proto bylo zvoleno použití vlastní aritmeticko-logické jednotky, která obsahuje pouze operace potřebné pro tento výpočet. Samotný průběh výpočtu je řízen dalším blokem, který také generuje adresy pro čtení a zápis dat z paměti (bloková RAM v FPGA). Pokud algoritmus neběží, je do této paměti umožněn přístup z nadřazeného modulu. Je tak možné zapisovat vstupní data a vyčítat výsledky.

Dále je počítáno s možným použitím ve vesmírných aplikacích. Při takových podmínkách může vlivem dopadu nabitých částic docházet ke změnám logických úrovní a tedy chybám. Nejnáchylnější jsou na toto obvykle právě BRAM bloky [16]. Proti tomu lze použít různé protiopatření, jako např. opravná logika EDAC, ta však zabírá další prostředky FPGA. Proto tyto paměti byly použity pouze na nutné věci a samotný průběh algoritmu se generuje pomocí kombinační logiky.

Struktura celého řadiče je znázorněna na Obr. 3-1. Blok *algo* je komponenta, řídící průběh výpočtu, *ALU* je aritmeticko-logická jednotka a *mem_ctrl* je přepínač přístupu do paměti, podle toho, zda probíhá nebo neprobíhá výpočet. *Test_mem* je, jak název napovídá, testovací zástupce blokové RAM. Při konkrétním použití zde bude dosazena vhodná implementace.



Obr. 3-1: Blokový diagram struktury regulátoru

3.1 Styl a struktura kódu

Použitý jazyk designu je podle zadání VHDL, konkrétně standard 2002, z důvodu zaručení kompatibility se všemi syntetizéry. Pro simulaci je však použita knihovna VUnit (viz kapitola 4), která vyžaduje verzi 2008 (či alespoň některé funkce tohoto standardu).

Jednotlivé bloky jsou rozděleny do souborů, které jsou spojeny v bloku *vec_ctrl.vhd*. Dále jsou použity dvě package, obsahující definice typů a některých konstant, které nebylo vhodné přiřazovat konkrétním souborům skrz generika.

Samotné bloky jsou pak popsány pomocí 2 a více procesů, kdy poslední implementuje klopné obvody a předchozí procesy pak kombinační logiku. Vstupy a výstupy do komponent nesou příponu *_in*, respektive *_out*. To neplatí pouze pro signály hodin (*clk*), synchronní reset (*rst*) a asynchronní reset (*arst*). U interních signálů komponent pak platí, že přípona *_r* značí klopný obvod přidružený ke kombinační cestě signálu bez přípony.

Pro podpůrné účely, testování goniometrických algoritmů a také běh již zmíněného VUnit je použit jazyk Python 3.

3.2 Parametrizovatelnost

Komponentu lze při syntéze několika způsoby přizpůsobit pro konkrétní použití. Pomocí generik lze nastavit logické úrovně aktivního synchronního resetu (RST_LVL), asynchronního resetu (ARST_LVL), aktivní úroveň povolovacího signálu pro klopné obvody (CLKEN_LVL) a pro zápis a čtení do BRAM (MEM_WRITE_LVL, respektive MEM_READ_LVL).

Dále je důležité určení šířky čísel používaných při výpočtu (OP_WIDTH). Jak bylo zmíněno, všechny proměnné a konstanty procházející aritmetickou jednotkou a ukládané do BRAM mají stejnou šířku. Druhý parametr, popisující formát čísel, je FRAC_POINT_POS, který určuje pozici pevné čárky. Všechny operace tedy probíhají při stejné přesnosti. Při paralelní implementaci by to bylo neefektivní, ale v tomto případě vše musí projít přes stejnou ALU. Při použití je tak třeba zvážit minimální šířku před a za čárkou, aby výsledky v celé části nepřetékaly a zároveň zlomková část měla dostatečnou přesnost pro správnou funkci algoritmu.

Další generika se týkají funkce aritmetické jednotky. ADD_PIPELINE a MULT_PIPELINE určuje počet úrovní klopných obvodů vložených do sčítačky, respektive násobičky, přičemž poslední z nich je výstup konkrétní operace. Minimální hodnota je 1 v případě násobičky, v případě sčítačky 2. Vyšší číslo znamená vyšší zpoždění, ale možný vyšší dosažitelný kmitočet. Je však třeba poznamenat, že to závisí na syntetizéru (je třeba navíc aktivovat funkci označovanou jako *register rebalancing* nebo *retiming*) podle toho, jak dokáže přemisťovat klopné obvody v kombinační cestě. Příliš vysoká hodnota tedy nemusí mít odpovídající přínosný výsledek.

Další důsledek změny tohoto parametru je nutnost úpravy řídícího algoritmu, protože, jak bude popsáno v kapitole 3.4, aritmetická jednotka je vysoce paralelní a jednotlivé instrukce počítají s přesným načasováním při výkonu operací. V základní podobě je algoritmus načasován pro hodnotu 2 obou parametrů. Také je počítáno se zpožděním jednoho taktu při čtení a maximálně 2 takty při zápisu (přičemž je to pipelinované).

Posledními parametry jsou ADD_OVERFLOW_EN a MULT_OVERFLOW_EN, které aktivují indikaci přetečení ve sčítačce a násobičce.

3.3 Komunikace s nadřazenou komponentou

Pro ovládání je vyvedeno několik individuálních portů. Logická úroveň, daná parametrem CLKEN_LVL, na vstupu *start_in* spustí celý výpočet. Pomocí vstupu *mode_in* lze nastavit, které PI smyčky se použijí. Průběh je pak signalizován výstupem *busy_out*. Jakmile je průchod algoritmem dokončen, vrátí se do logické 0. Pro čtení a

zápis do BRAM jsou zde standardní porty pro přístup do paměti. Při aktivní úrovni ve vstupu povolujícím zápis (*write_en_in*) dojde k zápisu hodnoty v *data_in* na adresu *addr_in*. Při aktivní úrovni *read_en_in* pak dojde k vyčtení dat z adresy v *addr_in* na výstup *data_out*. Posledními výstupy jdou indikace přetečení sčítačky (*add_overflow_out*) a násobičky (*mult_overflow_out*). Podrobněji jsou popsány v kapitole 3.4.

Obvyklé postup použití je pak takové, kdy se naplní vstupními hodnotami (požadované hodnoty veličin a výstupy motorových senzorů) příslušná místa v paměti a nastaví se vstup *mode_in* podle toho, které PI smyčky je zamýšleno pouštět. Následně se nastaví *start_in*. Vyčká se, než se provede výpočet nad vloženými daty pomocí signalizace *busy_out*. Poté se vyčtou z paměti výsledky algoritmu a předají se dále v řetězci směrem k motoru. V další iteraci se postup opakuje.

Komponenta není připojena k žádnému rozhraní komunikační sběrnice. Díky tomu ji lze snadněji integrovat do vlastního designu. Pokud by bylo potřeba, lze před ní připojit rozhraní na libovolnou sběrnici.

Kromě toho jsou zde také běžné vstupy pro hodinový signál (*clk*), synchronní (*rst*) a asynchronní reset (*arst*). Není třeba využívat oba dva, podle připojení signálu nebo konstanty dojde k přizpůsobení syntézy. Při použití asynchronního resetu je však třeba počítat, že není přítomen synchronizér vypnutí, proto je ho třeba použít někde v nadřazené úrovni, jinak při deaktivaci *arst* se klopné obvody mohou dostat do nedefinované úrovně. Důvod pro jeho nezařazení je větší flexibilita použití komponenty.

3.4 Aritmeticko-logická jednotka

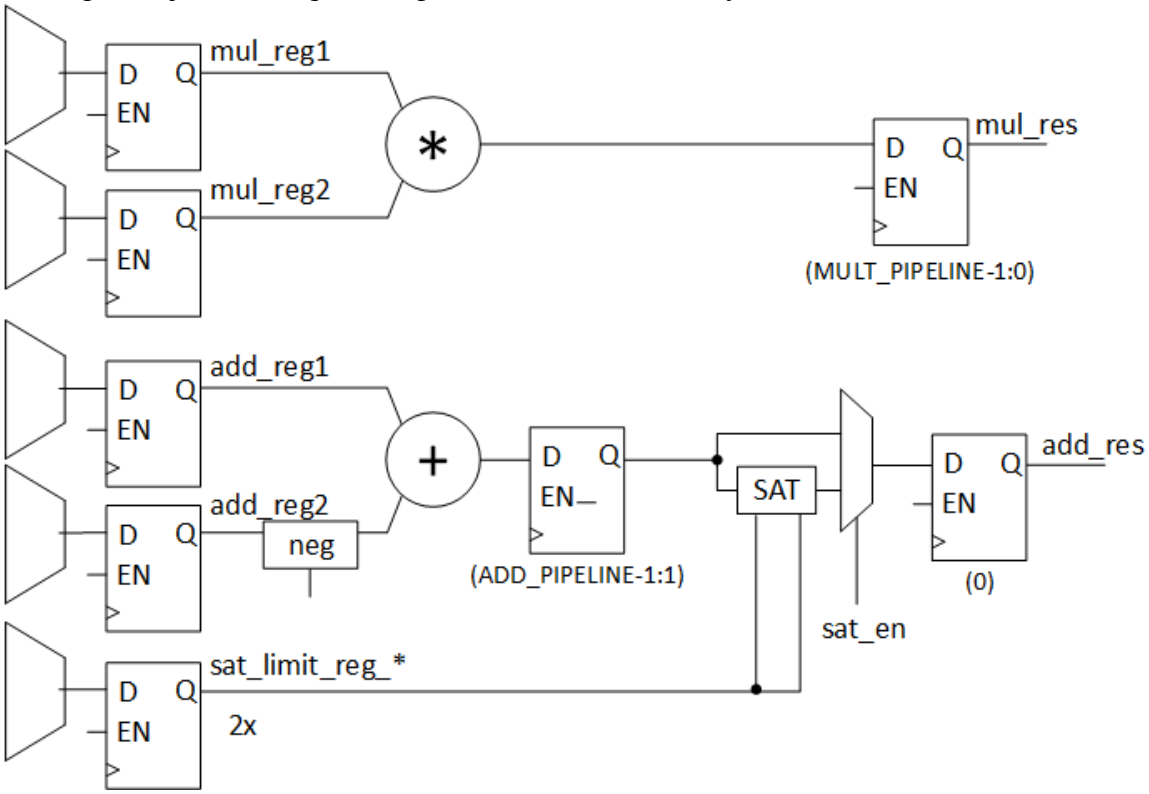
Jak již bylo zmíněno, celý výpočet probíhá zde, proto jednotka musí podporovat všechny použité operace (pouze určité bitové operace při výpočtu goniometrických funkcí byly přesunuty do bloku Algo – viz 3.5). Protože se ve všech krocích dělí pouze konstantou, lze to nahradit násobením převrácenou hodnotou. Dále je potřeba sčítání, odčítání a pro PI regulátory také saturace na minimální a maximální hodnotu.

Na Obr. 3-2 je znázorněno jejich provedení. Protože saturace je prováděna pouze po sčítání, umístěna je za sčítačku. K dosažení vyššího kmitočtu je vždy oddělena od sčítačky aspoň jednou úrovní klopných obvodů. Zvýšením hodnoty pipeline jsou tak další úrovně vkládány pouze za sčítačku. Do zpoždění se však počítají všechny úrovně zpoždění, proto minimální hodnota generika ADD_PIPELINE je 2. Přítomnost saturátoru je také využita k pasivní saturaci výsledků, pokud není aktivováno použití vlastních limitů. Při vhodné volbě šířky operandů by k tomu nemělo docházet, ale pokud se tak

přesto stane, výsledek se omezí na maximální nebo minimální hodnotu, kterou je konkrétní šířka čísla schopná pojmout a nedojde tak k přetečení nebo podtečení. Pokud je to aktivováno generikem (ADD_OVERFLOW_EN), dojde přitom k signalizaci na výstupu zmíněném v předchozí kapitole.

Násobička tuto pasivní saturaci neobsahuje, protože by byly spotřebovávány prostředky FPGA na něco, co je opatření proti nežádoucímu stavu, ke kterému by nemělo docházet při správném nastavení šířek operandů. K chybě by mohlo dojít kvůli působení záření, nicméně výpočet by byl pravděpodobně narušen i při saturaci. V této implementaci je alespoň zaručeno, že při přetečení nedojde ke změně znaménka výsledku. Opět, pokud je povoleno generikem (MULT_OVERFLOW_EN), je přetečení signalizováno příslušným výstupem.

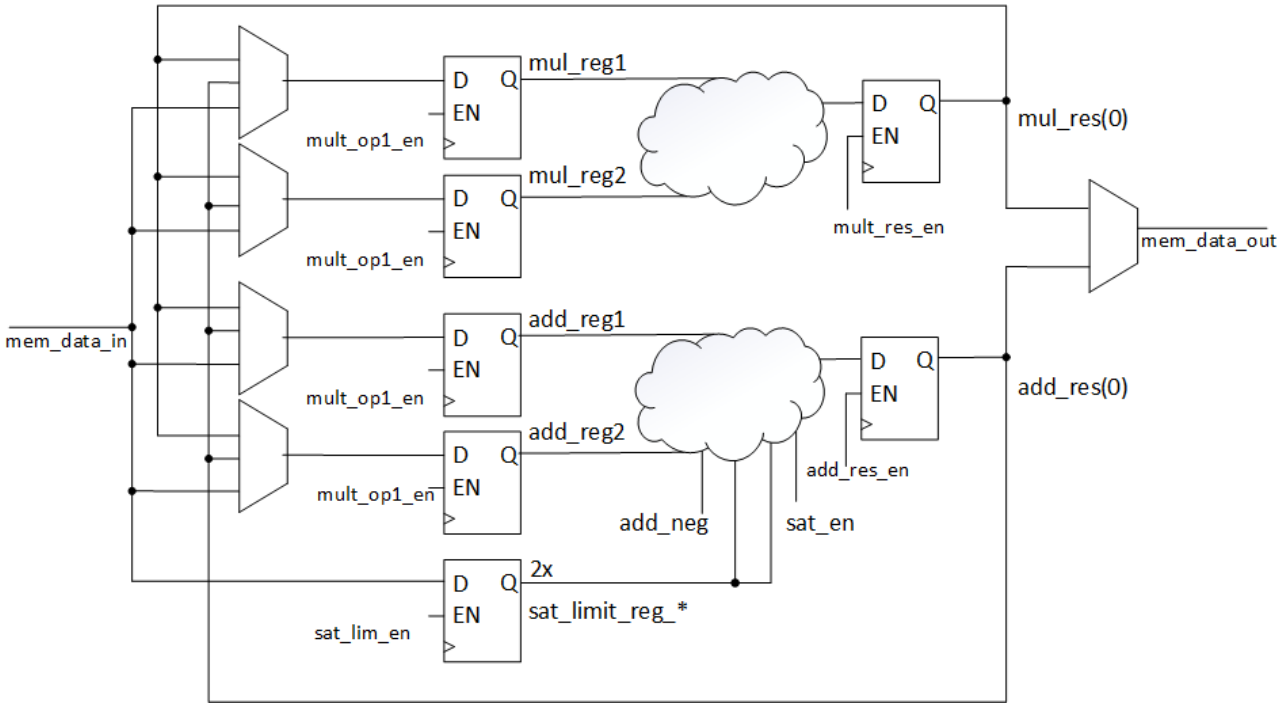
Samotné operace jsou popsané ve VHDL pomocí matematických operátorů. Optimální implementace je ponechána syntetizéru. Ten může použít náhledové tabulky, či pokud je má k dispozici, specializované DSP bloky.



Obr. 3-2: Zapojení matematických operací v ALU

Na Obr. 3-3 jsou znázorněny multiplexery (jejich řídící signály zaznačeny nejsou) rozvádějící signály mezi operacemi. Je vidět, že je možné libovolně předávat výsledky obou operací do libovolného operandu, nebo na výstup do paměti. Vstupy do klopných obvodů se saturačními operandy jsou vedeny pouze z paměti. Další podstatná věc je nezávislost přepínání multiplexerů a povolovacích signálů k uložení hodnot do operandů

nebo výsledků. Lze tak řídit průchod dat jednotkou podle potřeby a využívat tyto registry jako dočasné úložiště hodnot. Není potřeba přidávat další pomocné registry nebo příliš často data vždy po jednom výpočtu ukládat zpět do paměti.



Obr. 3-3: Zapojení multiplexů v ALU pro vedení dat

Důsledkem tohoto řízení průchodu dat je ale nemožnost určovat, kdy jsou načtené oba operandy konkrétní operace, a proto signalizace přetečení není nutně směrodatná, zda právě k němu dochází, a slouží spíše k odladění konkrétního nastavení.

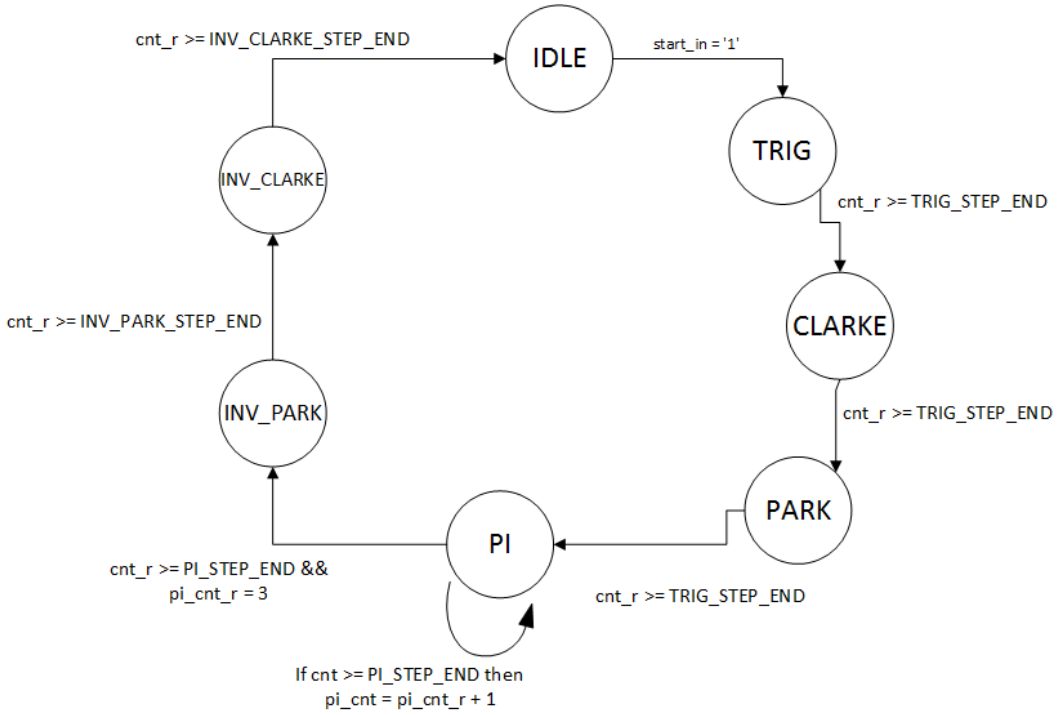
3.5 Algo

Tento blok řídí provádění regulačního algoritmu – čte a zapisuje do paměti a ovládá jednotlivé multiplexery a klopné obvody. Z důvodu již zmiňovaného předpokládaného použití ve vesmíru, kdy je omezováno použití RAM bloků v FPGA, je celé řízení algoritmu implementováno v programovatelné logice.

Základem je stavový automat se stavy reprezentujícími jednotlivé kroky výpočtu. Na Obr. 3-4 je zjednodušený diagram se stavy a přechody mezi nimi.

Při nečinnosti je setrváváno ve stavu IDLE, kdy je výstup *busy_out* nastaven do log. 0, což signalizuje právě nečinnost a zároveň to umožňuje externí přístup do paměti (viz kapitola 3.6). Také je možné hodnotou vstupu *mode_in* zvolit použité PI smyčky. To se provede uložením této hodnoty do čítače *pi_cnt_r*. Šířka těchto signálů je 2 bity, možnosti startovních smyček jsou 3, proto v případě zadání hodnoty 3 dojde k omezení na číslo 2.

Při spuštění výpočtu vstupem *start_in* pak dojde k přechodu do stavu TRIG a nastavení výstupu *busy_out* do log. 1.



Obr. 3-4: Zjednodušený diagram stavového automatu v algo

Stav TRIG je první z fází a jsou při něm počítány goniometrické funkce sinus a kosinus vstupního úhlu změřeného z motoru. Dále následují stavy CLARKE (αβ transformace), PARK (Parkova transformace), PI (PI smyčky), INV_PARK (zpětná Parkova transformace) a INV_CLARKE (zpětná αβ transformace). Poté se automat vrátí do stavu IDLE a nadřazený blok může opět začít komunikovat s pamětí.

Každá fáze algoritmu se skládá z několika kroků, instrukcí pro aritmeticko-logickou jednotku. K jejich počítání je použit čítač *cnt*, který je sdílen mezi všemi stavy. Aby zápis kódu byl přehlednější a umožnil snazší úpravy, jednotlivé kroky jsou uloženy v pomocné package, která obsahuje příslušný počet polí konstant. Již zmíněný čítač se pak použije jako index pole k dekódování aktuální instrukce. Jakmile v určitém stavu čítač napočítá hodnotu určenou další konstantou, přejde do dalšího stavu.

Úprava nějaké fáze pak spočívá pouze v úpravě příslušného pole konstant. Pokud je potřeba přidat fázi úplně novou, stačí zapsat další stav automatu a přiřadit mu jeho pole. Struktura package samotné a pomocný skript na její generaci jsou více popsány v 3.5.1.

Většina kroků je takto stejná, výjimky bylo třeba udělat u výpočtu goniometrických funkcí (podrobněji viz 3.5.2) a u PI smyček. Ty se opakují 4krát po sobě stejně, pouze je třeba měnit adresy operandů a výsledků. Proto je použito stejné pole. Přítomen je ale čítač

pi_cnt_r, jehož hodnota se používá jako předpona adres uložených v poli. Tím každá smyčka dostane své adresy. Vstupy jsou ale načítány z předchozí smyčky nebo z jiných algoritmů, proto je pro první dva kroky použita hodnota čítače o 1 menší (předchozí smyčka) anebo je změněn bit na o 1 vyšší pozici, než do které je vkládán *pi_cnt_r*.

Všechny tyto adresy z konstant a čítačů jsou pak posledním prefixovým bitem posazeny o 128 pozic výše, protože spodní pozice je vyhrazena pro náhledovou tabulku sinusů – viz 3.5.2, adresní prostor je popsán v 3.7.

3.5.1 Package vec_ctrl_cmds_pkg.vhd

Je to soubor obsahující definice jednotlivých výpočetních fází. Předávat komponentě tyto informace přes generika by bylo nepraktické, zvolena byla proto package, na kterou je možné se pak odkázat v každém souboru, kde je to potřeba.

Jednotlivé kroky jsou uloženy jako kompozitní typ obsahující 7bitovou spodní část adresy (*low_addr*), čtecí a zapisovací signál (*rw_en*), přepínací a povolovací signály (*switches* a *enables*), které jsou samy kompozitní typy definované v package vec_ctrl_types_pkg.vhd, zahrnující kontrolní prvky v ALU.

Dále následují konstanty jednotlivých paměťových pozic, které jsou použity při zápisu instrukcí, což jednak zpřehledňuje čtení tohoto souboru, a také je možné tyto adresy použít v simulačních souborech. Nakonec jsou přítomny pole s instrukcemi pro jednotlivé fáze výpočtu spolu s konstantou udávající počet instrukcí.

Pro snadnější generaci tohoto souboru byl také vytvořen pomocný skript pkg_print.py v jazyce Python 3, který dokáže vytisknout celou package. Výstup tohoto programu je pak třeba pouze přesměrovat do cílového souboru.

Skript ulehčuje zápis algoritmu tím, že pro *switches* a *enables* jsou definovány vlastní třídy s implicitními hodnotami, stačí tedy vyplnit pouze hodnoty, které se mění. Jednotlivé instrukce jsou vkládány jako položky seznamu. Adresní konstanty jsou čteny z definičního souboru addr_list.py. Hodnoty adresy jsou zapsány v desítkové soustavě. Tento skript je automaticky převede do binárního zápisu ve VHDL.

3.5.2 Goniometrické funkce

Protože se v Parkově transformaci, viz (1-8), vyskytují goniometrické funkce sinus a kosinus, je třeba do implementace zařadit jejich výpočet. Kritéria pro výběr jsou vhodnost použití s čísly s pevnou čárkou, doba trvání, množství spotřebovaných prostředků FPGA.

Nejběžnější způsoby jsou algoritmy typu CORDIC a metody využívající Taylorovu řadu. [14][15]

Metody CORDIC (COordinate Rotation Digital Computer) pracují na principu rotace vektoru, kdy rotaci o úhel ϕ lze vyjádřit takto [14]:

$$x' = xcos\varphi - ysin\varphi$$
$$y' = ycos\varphi + xsin\varphi \quad \textbf{(3-1)}$$

Vektor o známém úhlu lze iterativně otáčet o malé úhly, jejichž siny a kosiny jsou známé, dokud velikost otočení neodpovídá vstupnímu úhlu. Hlavní výhoda CORDIC je však ta, že vhodnou volbou pomocných úhlů lze získat součinitele o hodnotě 2x. Násobení je tak možné nahradit bitovým posunem. Rovnice jsou upraveny na tvar:

$$x' = cos\varphi(x - ytan\varphi)$$
$$y' = cos\varphi(y + xtan\varphi) \quad \textbf{(3-2)}$$

Počet iterací odpovídá požadované přesnosti. Na konci hodnoty x a y odpovídají spočítaným velikostem kosinu a sinu. Počáteční hodnoty a hodnoty arctan(2^{-i}) lze předpočítat a uložit do paměti v FPGA [14].

Nevýhoda tohoto řešení je však poměrně dlouhá doba výpočtu, zvláště v již představené ALU, které neobsahuje 2 sčítačky ani programovatelný bitový posun. I když ten by šel nahradit násobičkou, právě její přítomnost anuluje jednu z hlavních výhod CORDICu. Proto bylo vybíráno z aproximací využívajících Taylorovu řadu, které na rozdíl od něj v tomto designu budou mnohem rychlejší.

Porovnávány byly 2 metody. Při první jsou nejprve vrchní bity vstupního úhlu použity jako adresa do náhledové tabulky sinusů a kosinusů, kdy z tohoto nalezeného místa je zapsán Taylorův rozvoj:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots \quad \textbf{(3-3)}$$

Derivace sinu a kosinu je snadná, proto vzorec lze zapsat takto pro 2. řád:

$$\sin(x) = \sin(a) + \cos(a)(x-a) - \frac{1}{2}\sin(a)(x-a)^2$$
$$\cos(x) = \cos(a) + \sin(a)(x-a) - \frac{1}{2}\cos(a)(x-a)^2 \quad \textbf{(3-4)}$$

Člen *(x-a)* lze jednoduše získat jako spodní bity vstupního úhlu nepoužité pro adresaci.

Při druhé metodě je stejně nalezena hodnota z tabulky. Pro spodní bity je ale použita aproximace Taylorovým rozvojem z 0, protože zbylé bity tvoří malý úhel. Druhý řád má tvar tento:

$$\sin(x) = x$$
$$\cos(x) = 1 - x^2/2 \quad \textbf{(3-5)}$$

Tyto dva siny/kosiny pak lze pomocí rovnosti součtu úhlů spojit do jednoho:

$$\sin(a + x) = \sin(a)\cos(x) + \cos(a)\sin(x)$$
$$\cos(a + x) = \cos(a)\cos(x) - \sin(a)\sin(x) \quad \textbf{(3-6)}$$

Obě varianty byly simulovány pomocným skriptem test_trig_algo.py, který využívá knihovnu *spfpm* pro výpočty s pevnou čárkou. Postupně je jako argument použito 100 úhlů od 0 do 2π.

Z porovnání vyplynulo, že první metoda je nejen méně přesná, ale také nedodržuje monotónnost ve správných úsecích sinusové křivky. To by zvláště při pomalém otáčení motoru mohlo velmi zhoršit stabilitu regulace. Vybrána byla proto metoda druhá. Při nastavení 16 bitů za pevnou čárkou bylo dosaženo největší absolutní chyby 0,00018.

Při implementaci bylo nutné přidat do algo.vhd výjimku pro práci s adresou náhledové tabulky. Nejprve je vstupní úhel vynásoben škálovací konstantou tak, aby celý kruh namapoval na hodnoty [- 1;1). Tím je možné adresu z úhlu vypočítat pouhým bitovým výběrem. Adresa je uložena do speciálního registru právě v algo.vhd na specifickou hodnotu čítače *cnt_r*. Přitom je naškálovaný úhel v bloku *mem_ctrl*, ořezán na spodní bity tvořící zbytek úhlu (viz 3.6). Tento zbytek je potřeba ještě vynásobit π a je možné jej použít jako argument v rovnicích (3-5).

Protože kosinus je sinus posunutý o π/4, je možné použít pouze jednu náhledovou tabulku se sinusy a kosinusy vybírat dalším adresním registrem. K původní adrese je přičtena čtvrtina velikosti tabulky (v tomto případě 32). Kdy se tyto adresy mají použít je určováno podmínkami na hodnotu *cnt_r,* v tom případě dojde k přepnutí adresního prefixu, protože sinusy jsou uloženy v nižší polovině paměti (viz kapitola 3.7).

3.6 Paměťový přepínač

Komponenta, která řídí přístup do BRAM podle toho, zda je algoritmus v běhu nebo ne. Pokud běží, což je signalizováno vstupem *busy_in*, jsou porty pro externí přístup odpojeny, datový výstup vždy vrací 0 a nedochází k zápisu do paměti. Po skončení výpočtu jsou tyto porty opět připojeny.

Také je zde implementována již zmiňovaná úprava datového vstupu pro účely goniometrických funkcí. Tato funkce se aktivuje vstupem *diff_en_in*. Při té jsou ponechány pouze spodní bity, které nejsou použity k adresaci náhledové tabulky sinusů. Ostatní jsou vynulovány. Výsledek této operace je v rámci algoritmu okamžitě uložen do RAM.

Komponenta obsahuje pouze kombinační logiku.

3.7 Paměť

Pro potřeby testování je přiložena základní implementace synchronní jednoportové RAM paměti v FPGA. Při reálném nasazení bude nahrazena požadovanou implementací. Šířka adresy je 8 bitů, obsahuje tedy 256 slov, jejíž šířka se nastavuje podle šířky čísel použité ve zbytku designu. Algoritmus současné čtení a zápis nyní nevyužívá, nicméně tato paměť to podporuje, přičemž nejprve je buňka přečtena a poté dojde k jejímu přepsání.

Pro inicializace je připraven skript mem_data_print.py, který vytiskne pole s hodnotami pro všechny buňky. Skript automaticky spočítá hodnoty sinů pro náhledovou tabulku a vloží je do příslušných adres. Ostatní konstanty jsou definovány instancemi třídy MemCell, kdy první argument konstruktoru je adresa, druhý argument je inicializační hodnota. Jako základní adresy je použit soubor addr_list.py, přičemž k těmto hodnotám je přičten odpovídající prefix. Všechny hodnoty jsou převedeny do bitové šířky nastavené proměnnou na začátku skriptu (*data_width*). V souboru jsou uvedeny také všechny vstupní adresy včetně zanořených PI smyček.

Zde jsou uvedeny rozsahy adres jednotlivých sekcí:

- 0-127 – náhledová tabulka sinusů
- 128-135 – poziční PI smyčka
- 136-143 – rychlostní PI smyčka
- 144-151 – Iq PI smyčka
- 152-159 – Id PI smyčka
- 160-255 – ostatní

4 TESTOVÁNÍ A SYNTÉZA

Správnost návrhu byla ověřena sérií funkčních testů zkoušejících postupně více prvků regulátoru. Pro spouštění testů a nějaké další funkce byla použit nástroj VUnit, který sestává ze spolupracujících knihoven ve VHDL a Pythonu. Pomocí skriptu run.py pak lze snadno spouštět jeden čí více testů. Skript automaticky najde testovací soubory se správnou předponou (*tb_*) nebo příponou (*_tb*) a které mají jako generikum znakový řetězec *runner_cfg*. Pomocí parametru *-l* lze pak například vypsat všechny nalezené testy. Konkrétním jménem testu nebo použitím znaků * jako divokých karet lze vyvolat pouze některé:

*python run.py *loop**

Tímto příkazem jsou např. spuštěny všechny testy, v jejichž názvu se vyskytuje slovo „loop“.

Z volně dostupných simulátorů, které jsou podporovány VUnit je použit program GHDL a jako zobrazovač průběhů signálů GTKWave.

4.1 Aritmeticko-logická jednotka

Nejprve byla otestována správnost samotné aritmeticko-logické jednotky. Byly vytvořeny testy sčítání („Adder test“), násobení („Multiplier test“), zda saturace funguje správně („Saturation test“). Byl proveden test multiplexerů („Interchange test“) a nakonec ověření správnosti pipeliningu obou operací („Pipeline test“).

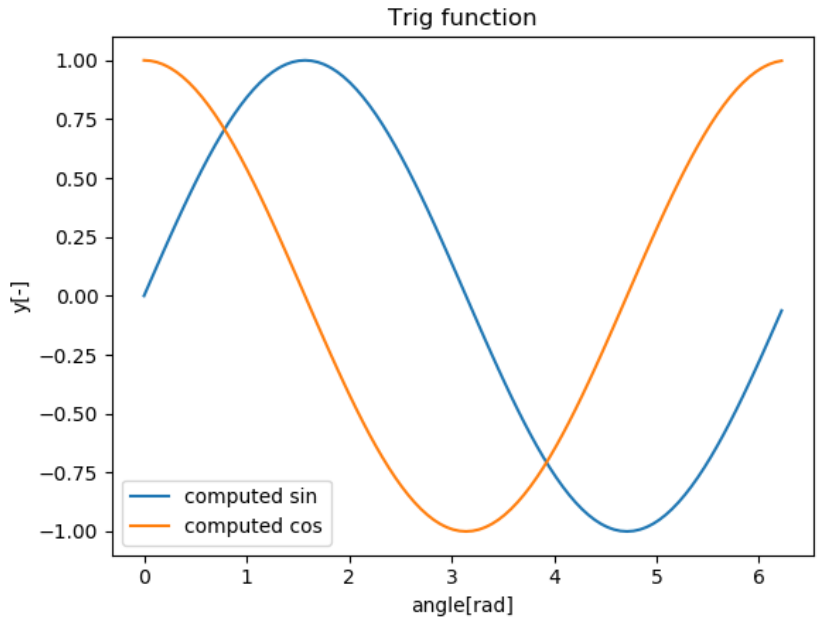
Tyto testy jsou přímé, s automatickou kontrolou správnosti výsledků.

4.2 Goniometrické funkce

Další možnosti regulátoru již bylo vhodnější testovat v rámci celého designu. První byly ověřeny goniometrické funkce. Ty jsou v rámci algoritmu počítány jako první krok. Nemělo by se tedy stát, že dojde k přepsání vložených vstupních testovacích dat (v průběhu algoritmu do paměti již přistupovat nelze).

Test je pojmenovaný „Trig test“ a uvnitř souboru vec_ctrl_tb.vhd. Postupně jsou vkládány násobky úhlu $2\pi/100$ do regulátoru, ten je spuštěn a poté jsou vyčteny výsledné hodnoty sinu a kosinu. Ty jsou ukládány do pomocných souborů tb_sin_out.txt a tb_cos_out.txt. Nad nimi lze spustit další pomocný skript trig_draw.py, který vykreslí vypočítané hodnoty do grafu a určí maximální absolutní chybu a celkové harmonické zkreslení.

Pro polohu pevné čárky 16 je maximální absolutní chyba sinu 0,000415 a kosinu 0,000340. Celkový průběh obou funkcí je vykreslen na Obr. 4-1.



Obr. 4-1: Graf výstupu aproximace goniometrických funkcí

4.3 Testy s modelem motoru

Další testy byly provedeny s pomocí modelu synchronního motoru s permanentními magnety napsaného ve VHDL – motor_model.vhd. Model pracuje pouze s reálnými čísly, vstupy jsou 3 fáze napětí, výstupy pak 3 fáze proudů, orientace motoru a jeho rychlost. Vždy, když je na řídícím vstupu *trig* zaznamenána náběžná hrana, model spočítá s nynějšími hodnotami vstupů nové velikosti výstupů.

Všechny následující testy jsou uloženy ve vec_ctrl_tb.vhd. Navolena je šířka operandů 32 a poloha čárky 16. Pro zrychlení simulace je model spouštěn rychleji, než je jeho vzorkovací kmitočet, proto časová osa v simulaci neodpovídá realitě. Ověření funkci to ale nebrání.

4.3.1 αβ transformace, Parkova transformace

Model interně používá stejné transformace jako v regulátoru – αβ a Parkovu. Nejprve jsou vstupní napětí převedena na napětí U_d a U_q. Jsou vypočítány nové proudy I_d a I_q, orientace motoru a jeho rychlost. Následně jsou proudy I_d a I_q převedeny zpětnými transformacemi na výstupní 3fázové proudy. Jelikož transformace jsou symetrické, dopřednými a následně zpětnými transformacemi by mělo vyjít stejné číslo, jako na začátku. Toho je využito při ověřování správnosti.

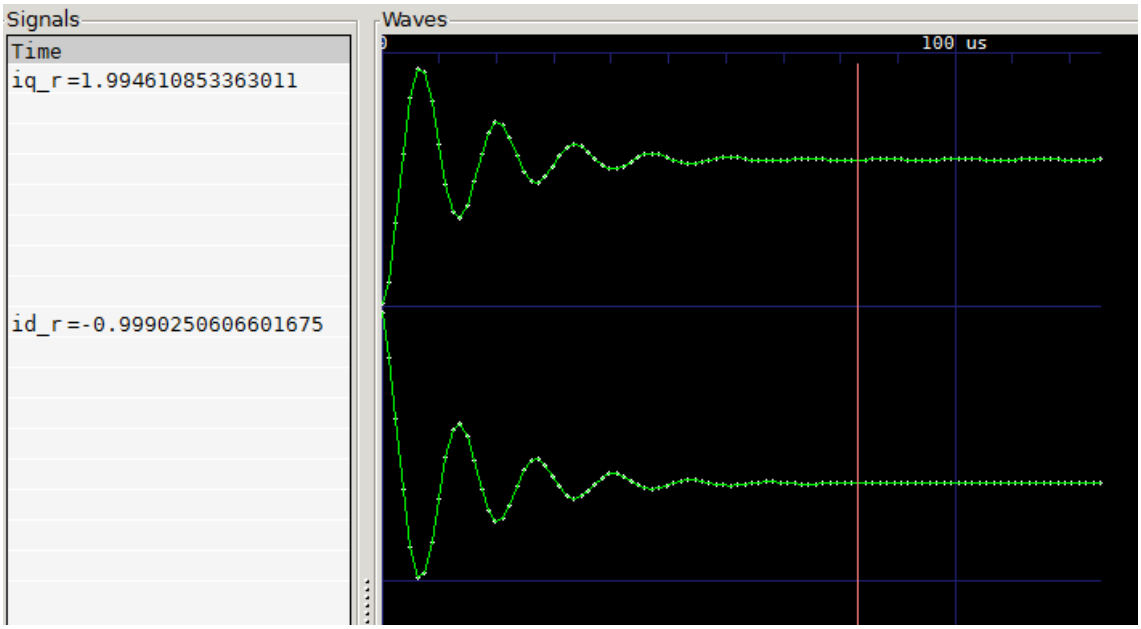
Ověření dopředných transformací je realizováno v testu „Forward Clarke/Park“. Úpravou modelu motoru je možné nastavovat vnitřní proudy I_d a I_q. Ty jsou pak porovnány s hodnotami vzniklými dopřednými transformacemi v regulátoru. Test vyzkouší postupně 200 kombinací hodnot a vždy vytiskne na řádek nejprve relativní chybu I_d a poté I_q.

Ověření zpětných transformací je provedeno identicky testem „Inverse Park/Clarke“. Pouze je třeba před jeho spuštěním změnit v algo.vhd přechod ze stavu TRIG do INV_PARK místo do CLARKE. Důvodem je, že jinak dojde k přepsání vložených testovacích dat výsledky proudových PI smyček.

4.3.2 Proudová smyčka

Jakmile je ověřeno, že transformace pracují správně, je možné začít testovat regulátory. První PI smyčky jsou proudové. Jejich test je pojmenován „Current loop“. Pomocí vstupu *mode_in* jsou deaktivovány poziční a rychlostní smyčky. Následně jsou do regulátoru vloženy konstanty. Celá soustava se nechá běžet dostatečný počet iterací a sleduje se, zda se vnitřní I_d a I_q proudy v motorovém modelu ustálí na hodnotách nastavených v regulátoru.

Na Obr. 4-2 je vidět průběh těchto proudů pro referenční hodnoty -1.0 (I_d) a 2.0 (I_q).

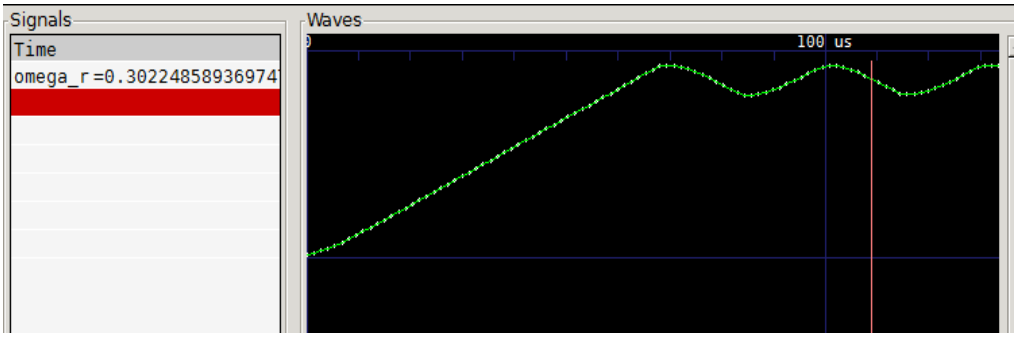


Obr. 4-2: Průběh proudů I_q a I_d v modelu motoru při proudovém řízení

4.3.3 Rychlostní smyčka

Test rychlostní smyčky („Speed loop“) je analogický. Vstupem *mode_in* je tentokrát deaktivována pouze poziční smyčka.

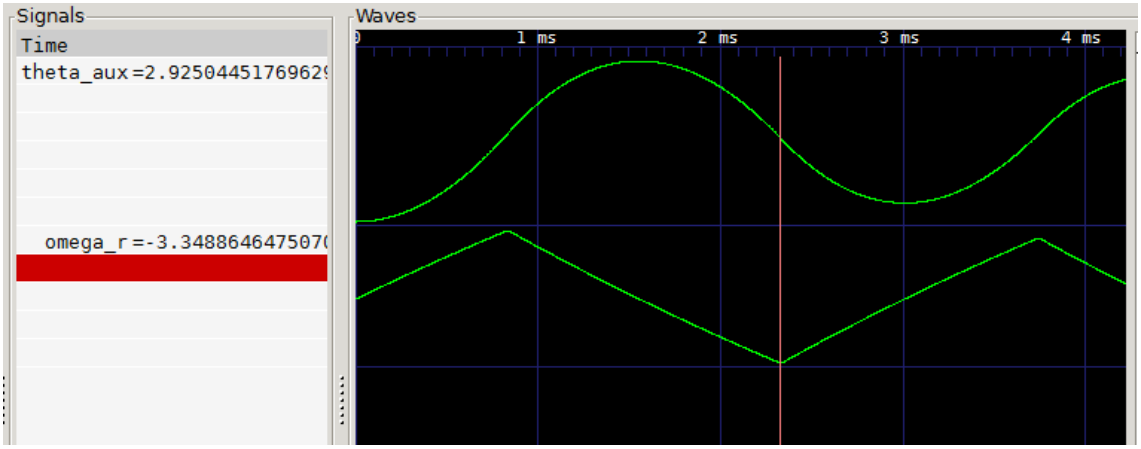
Na Obr. 4-3 je vidět výslednou rychlost motoru mírně oscilující kolem referenční hodnoty 0,3 (± 0,3). To může být způsobeno nastavením regulátorů. Funkce je však principiálně správná.



Obr. 4-3: Průběh úhlové rychlosti modelu motoru při rychlostním řízení

4.3.4 Poziční smyčka

Poslední test je využívající plné možnosti navrhované komponenty. Regulátoru je dodán požadovaný úhel, který se ten následně snaží nastavit na motoru. To je patrné na Obr. 4-4. Zde se však již příliš projevují nevhodné koeficienty regulátoru a výsledný úhel výrazně osciluje kolem požadovaného (±2,5 rad kolem požadovaných 3 rad). Funkce regulátoru jako taková je ale správná. Na průběhu úhlové rychlosti *omega* je vidět, že vždy při průchodu úhlu *theta* požadovanou hodnotou dojde ke změně znaménka gradientu rychlosti.



Obr. 4-4: Průběh orientace a úhlové rychlosti modelu motoru při pozičním řízení

4.4 Syntéza

Návrh byl na zkoušku pro zjištění využitých prostředků a maximálního orientačního kmitočtu syntetizován pro FPGA Spartan-3 XC3S200 od výrobce Xilinx nástrojem XST, který součástí softwarového prostředí ISE 14.7.

Při syntéze byla z důvodů aplikace ve vesmíru aktivována bezpečná implementace stavových automatů (přepínač safe_implementation). Pro zrychlení matematických operací pomocí dodatečných úrovní zpoždění byla zapnuta možnost register-rebalancing.

Nejnákladnější částí se podle reportů syntézy ukázalo být násobení. Proto velmi záleží, zda cílové zařízení má k dispozici specializované DSP bloky s hardwarovými násobičkami, či ne. V případě použití náhledových tabulek jejich spotřeba roste nelineárně se zvolenou šířkou operandů. To prodlužuje také kombinační cestu, a tedy snižuje dosažený kmitočet. Nicméně při vyšších šířkách je nutno spojovat více DSP bloků (v architektuře Spartan-3 jsou to násobičky 18x18 bitů [17]), čímž je kmitočet rovněž penalizován. Porovnání různých šířek operandů, pozice čárky a použití DSP bloků je uvedeno v Tabulka 1.

Tabulka 1: Využití prostředků Spartan-3 XC3200 pro různé konfigurace

	32/16 bitů, bez DSP	32/16 bitů, s DSP	18/8 bitů, bez DSP	18/8 bitů, s DSP
klopné obvody (z 3840)	393	414	241	359
4vstupové LUT (z 3840)	2113	1095	1042	676
BRAM (z 12)	1	1	1	1
MULT18x18 (z 12)	0	4	0	1
Přibližný maximální kmitočet [MHz]	41,954	64,371	54,377	107,393

5 ZÁVĚR

Tato diplomová práce se zabývala implementací vektorového řízení synchronních motorů s permanentními magnety (BLDC nebo PMSM) v jazyce VHDL do obvodů FPGA. Nejprve byly vysvětleny jejich rozdíly, co se od nich dá při provozu očekávat. Následně byl popsán jejich princip.

Dále bylo popsáno vektorové řízení a její využití základních principů těchto motorů. Kromě toho popsány nebytné matematické operace a transformace použité při tomto řízení.

V druhé kapitole byly tyto části spojeny do celkové regulační soustavy řídící požadovaný chod motoru. Regulace je provedena 3 zanořenými PI smyčkami, kdy první kontroluje proudy v motoru. Nad ní je postavena smyčka hlídající rychlost a ta je řízena poslední smyčkou regulující orientaci motoru. Při řízení je pak možno za běhu volit, kolik smyček je aktivních.

Ve třetí kapitole byla popsána samotná implementace celého kontroléru. Z důvodu vysokého rozdílu frekvence regulace a frekvence FPGA byla zvolena struktura s řídícím blokem realizujícím jednotlivé kroky algoritmu jako instrukce pro aritmeticko-logickou jednotku, přičemž všechna data, tedy vstupy, výstupy, konstanty apod. jsou uloženy v paměti RAM v FPGA, což zvyšuje univerzálnost regulátoru, kdy je možné i po implementaci do konkrétního obvodu je možné měnit parametry regulace. Z důvodu předpokládaného využití ve vesmírných aplikacích je ale další využívání blokové paměti na čipu omezeno. Instrukce řídící celý algoritmu jsou proto implementovány jako kombinační síť. K jejich definici je použita package, což usnadňuje budoucí změny v algoritmu.

V poslední části byla komponenta ověřena RTL simulací. Nejprve byly otestovány implementace goniometrických funkcí a transformací soustav. Poté byl design zapojen do zpětné vazby a otestovány byly postupně jednotlivé regulační PI smyčky. K tomuto byl využíván model motoru. Jelikož model využívá stejné transformace jako regulátor, bylo možné některé veličiny přímo porovnávat a tím určovat jejich korektnost. U zpětnovazebních simulací se ukázala vysoká citlivost na správné nastavení regulačních parametrů.

Na konci byla provedena testovací syntéza, do které se projevila vysoká závislost využitých prostředků FPGA na šířce operandů, kdy nejvíce vždy zabírá násobička, která roste nelineárně s bitovou šířkou. Ta také obvykle určuje výsledný dosažitelný kmitočet. Pokud cílové FPGA obsahuje DSP bloky, lze je použít k výpočtu násobení, což ušetří

značnou část náhledových tabulek, nicméně při vyšších bitových šířkách to nemusí mít na dosažitelnou frekvenci vysoký vliv. Záležet bude na použité architektuře FPGA.

Literatura

[1] GIERAS, Jacek F. *Permanent magnet motor technology: design and applications*. 3rd ed. Boca Raton: CRC Press, c2010. ISBN 978-1-4200-6440-7.

[2] HUGHES, Austin a Bill DRURY. *Electric motors and drives: fundamentals, types and applications*. 4th ed. Amsterdam: Elsevier, 2013. ISBN 978-0-08-098332-5.

[3] PAVELKA, Jiří. *Elektrické pohony*. Praha: Nakladatelství ČVUT, 2007. ISBN 978-80-01-03588-7.

[4] ANDREESCU, Gheorghe-Daniel, Cristina-Elena COMAN, Ana MOLDOVAN a Ion BOLDEA. Stable V/f control system with unity power factor for PMSM drives. *2012 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*. IEEE, 2012, 2012, , 432-438. DOI: 10.1109/OPTIM.2012.6231937. ISBN 978-1-4673-1653-8. Dostupné také z: http://ieeexplore.ieee.org/document/6231937/

[5] LEPKA, Jaroslav a Petr STEKL. *3-Phase AC Induction Motor Vector Control Using a 56F80x, 56F8100 or 56F8300 Device* [online]. [cit. 2018-12-14]. Dostupné z: http://cache.freescale.com/files/product/doc/AN1930.pdf

[6] ŠTULRAJTER, Marek, Valéria HRABOVCOVÁ a Marek FRANKO. Permanent magnets synchronous motor control theory. *Journal of ELECTRICAL ENGINEERING,*. 2006, **2007**(58), 79-84.

[7] LEPKA, Jaroslav. *Moderní metody bezsnímačového řízení pohonů s PMSM motorem*. Brno, 2018. Dizertační práce. VUT FEKT.

[8] PROKOP, Libor a Pavel GRASBLUM. *3-Phase PM Synchronous Motor Vector Control Using a 56F80x, 56F8100, or 56F8300 Device* [online]. [cit. 2018-12-14]. Dostupné z: https://cache.freescale.com/files/product/doc/AN1931.pdf

[9] *Field Orientated Control of 3-Phase AC-Motors* [online]. [cit. 2018-12-14]. Dostupné z: http://www.ti.com/lit/an/bpra073/bpra073.pdf

[10] *Park, Inverse Park and Clarke, Inverse Clarke Transformations MSS Software Implementation* [online]. [cit. 2018-12-14]. Dostupné z: https://www.microsemi.com/document-portal/doc_view/132799-park-inverse-park-and-clarke-inverse-clarke-transformations-mss-software-implementation-user-guide

[11] SKALICKÝ, Jiří. *Teorie řízení 1*. Brno: Vysoké učení technické, 2002. Učební texty vysokých škol. ISBN 80-214-2112-6.

[12] ÅSTRÖM, Karl J. a Richard M. MURRAY. *Feedback systems: an introduction for scientists and engineers*. Princeton: Princeton University Press, c2008. ISBN 978-0-691-13576-2.

[13] Comparison and Evaluation of Anti-Windup PI Controllers. *Journal of Power Electronics*. 2011, **11**(1), 45-50. DOI: 10.6113/JPE.2011.11.1.045. ISSN 1598-2092.

[14] ANDRAKA, Ray. A survey of CORDIC algorithms for FPGA based computers. *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays - FPGA '98*. New York, New York, USA: ACM Press, 1998, 1998, , 191-200. DOI: 10.1145/275107.275139. ISBN 0897919785. Dostupné také z: http://portal.acm.org/citation.cfm?doid=275107.275139

[15] DE DINECHIN, Florent, Matei ISTOAN a Guillaume SERGENT. Fixed-point trigonometric functions on FPGAs. *ACM SIGARCH Computer Architecture News*. 2014, **41**(5), 83-88. DOI: 10.1145/2641361.2641375. ISSN 01635964. Dostupné také z: http://dl.acm.org/citation.cfm?doid=2641361.2641375

[16] *Using EDAC RAM for RadTolerant RTAX-S/SL and Axcelerator® FPGAs* [online]. [cit. 2019-05-21]. Dostupné z: https://www.microsemi.com/document-portal/doc_download/129919-ac319-using-edac-ram-for-radtolerant-rtax-s-sl-fpgas-and-axcelerator-fpgas-app-note

[17] *Spartan-3 FPGA FamilyData Sheet* [online]. [cit. 2019-05-21]. Dostupné z: https://www.xilinx.com/support/documentation/data_sheets/ds099.pdf