VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Ondřej Šoupal

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV VÝKONOVÉ ELEKTROTECHNIKY A ELEKTRONIKY
DEPARTMENT OF POWER ELECTRICAL AND ELECTRONIC ENGINEERING

PROGRAMOVÁNÍ MIKROKONTROLÉRŮ C2000 V PROGRAMU MATLAB/SIMULINK
MATLAB/SIMULINK MODEL BASED DESIGN USING C2000 MICROCONTROLLERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Ondřej Šoupal

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Ivo Pazdera, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Silnoproudá elektrotechnika a výkonová elektronika**

Ústav výkonové elektrotechniky a elektroniky

Student: Bc. Ondřej Šoupal ***ID:*** 186207

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Programování mikrokontrolérů c2000 v programu MATLAB/Simulink

POKYNY PRO VYPRACOVÁNÍ:

1. Popište princip programování mikrokontrolerů c2000 v programu MATLAB/Simulink.
2. Na laboratorním pracovišti realizujte, dle pokynů vedoucího, vybrané aplikace řízení elektrických pohonů pomocí platformy Texas Instrument Lunchpad.
3. Přehledně popište realizované aplikace a jejich výsledky.

DOPORUČENÁ LITERATURA:

[1] Klíma B., Střídavé pohony, 2014

[2] PATOČKA, M. Magnetické jevy a obvody ve výkonové elektronice, měřicí technice a silnoproudé elektrotechnice. odborné knihy. odborné knihy. Brno: VUTIUM, 2011. 564 s. ISBN: 978-80-214-4003- 6.

[3] Veltman A., Pulle D., Doncker R.W, Fundamentals of Electrical Drives, 2007

[4] Pyrhonen J., Hrabovcova V.,Semeken R., Electrical Machine Drives Control, Whiley, 2016

[5] Bacha S., Munteanu I., Bratcu A., Power Electronic Modeling and Control, Springer, 2014

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Ivo Pazdera, Ph.D.

doc. Ing. Ondřej Vítek, Ph.D.
předseda oborové rady

ABSTRAKT

Cílem této práce je prozkoumat možnosti metody automatického generování kódu, popsat koncept tvorby aplikace ve vybraném prostředí MATLAB/Simulink pro vývojový kit Texas Instruments LaunchPad a vytvořit pro něj aplikaci pro řízení DC a asynchronního motoru v tomto prostředí. Práce se zaměřuje na detailní popis aplikace, která spočívá v unipolárním/bipolárním řízení H-můstku výkonového měniče pro DC motor, měření výstupních proudů, otáček a jejich zobrazování v reálném čase s pomocí sériové linky. Dále se práce zabývá skalárním a vektorovým řízením asynchronního motoru. Všechny aplikace jsou vytvořeny v prostředí MATLAB/Simulink. Jsou k dispozici v přílohách práce spolu s naměřenými veličinami.

KLÍČOVÁ SLOVA

Automatické generování kódu, pulsní šířková modulace, unipolární/bipolární řízení, ADC převodník, enkodér, regulátor proudu, regulátor otáček, MATLAB, Simulink, blokové schéma, DC motor, asynchronní motor, skalární řízení, vektorové řízení, Parkova transformace, Clarkové transformace

ABSTRACT

The aim of this thesis is to explore possibilities of rapid control prototyping, describe the concept of creating the software application in MATLAB/Simulink environment with use for development kit Texas instruments LaunchPad and create an application for DC and induction motor control in this environment. This work describes the application for unipolar/bipolar control H-Bridge of power converter for DC motor, measurement of output currents, speed and its displaying in real time using serial control interface. This thesis also desribes scalar and vector control of induction motor. All software applications with measurements are created in MATLAB/Simulink and attached to the thesis.

KEYWORDS

Rapid control prototyping, pulse width modulation, unipolar/bipolar control, A/D converter, encoder, current regulator, speed regulator, MATLAB, Simulink, block diagram, DC motor, induction motor, scalar control, vector control, Park transformation, Clarke transformation

ŠOUPAL, Ondřej. *Možnosti řízení elektrických pohonů v aplikacích s automaticky generovaným kódem* . Brno, Rok, 135 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Ivo Pazdera, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Možnosti řízení elektrických pohonů v aplikacích s automaticky generovaným kódem “ 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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Ivu Pazderovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vznik a rozvoj elektrických pohonů sahá již do 19. století a v průběhu času zaznamenalo velký vývoj především jejich řízení. Od původního analogového až po digitální řízení realizované pomocí mikroprocesorů. Pro používání těchto mikroprocesorů je potřeba znalost složitých programovacích jazyků a v posledních letech je tento problém často řešen pomocí tzv. Rapid Control Prototyping (RCP) [1], kdy je pomocí softwaru přímo vygenerován kód v příslušném jazyce pro mikroprocesory (dále jen μP). Tento kód je následně inkrementován do μP a ten již ovládá danou aplikaci, v pohonech se jedná o řídicí obvody výkonové měniče pro daný elektromotor. Tato práce se bude zabývat metodou Model Based Design (dále jen MBD) [2], která je druhem RCP. Při MBD je zdrojový kód pro μP odvozen z blokového schématu vytvořeného ve vývojovém softwaru. Pro tuto práci bylo zvoleno prostředí MATLAB/Simulink.

První část této práce je zaměřena na mikrokontroléry využívané v elektrických pohonech, ve druhé části potom na vývojová prostředí, která umožňují RCP.

Blíže je zpracován koncept tvorby aplikace v prostředí MATLAB/Simulink pro mikroprocesory Texas Instruments, který nalezneme ve třetí kapitole.

Čtvrtá kapitola nás stručně seznámí s laboratorním standem s DC a asynchronním motorem, které budou řízeny pomoci vývojového kitu.

Pátá kapitola pojednává o tvorbě programu pro řízení a regulaci DC motoru. Jako příklad byl vytvořen program v Simulinku pro vývojový kit LaunchPad F28069M, který zajišťuje buď unipolární nebo bipolární řízení H-můstku výkonového měniče laboratorního standu, snímání proudu z čidla LEM, otáček z resolvéru na hřídeli motoru, proudovou a otáčkovou regulaci motoru. Po sériové lince je možné nastavit požadovanou hodnotu otáček a zobrazení veličin proudu a skutečných otáček v reálném čase.

V šesté kapitole nalezneme popis tvorby skalárního řízení v otevřené a uzavřené smyčce pro asynchronní motor.

Sedmá kapitola se zabývá zatěžováním asynchronního motoru pomocí stejnosměrného motoru.

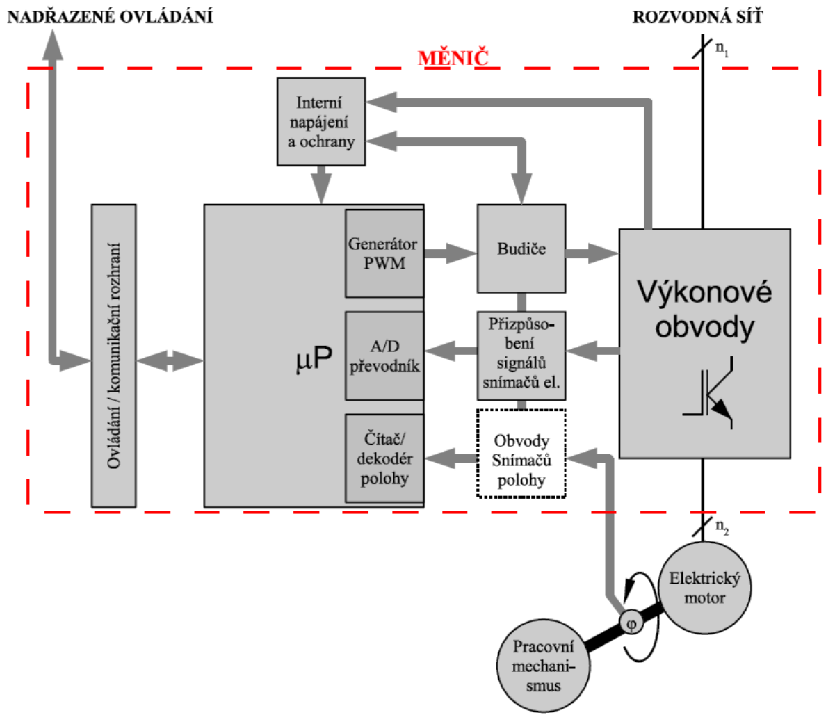
Osmá kapitola je zaměřena na vektorové řízení. Byla vytvořena jednodušší verze „kvazivektorového“ řízení bez estimace rotorového toku.

V přílohách jsou přiloženy schémata redukce DPS pro připojení kitu LaunchPad k laboratornímu standu, dále bloková schémata vytvořené aplikace v Simulinku a průběhy naměřených veličin.

1 Mikroprocesory využívané v elektrických pohonech

V elektrických pohonech nalézají využití procesory DSP - digitální signálový procesor. Tyto procesory slouží pro aplikace ovládané v reálném čase. Dále je rozdělujeme podle architektury do různých rodin, např.: AVR, PIC, DSP, ARM, STM32, MSP, TSM a další. Dělení podle využívané aritmetiky na celočíselné, s pevnou řádovou čárkou a s plovoucí řádovou čárkou.

Z hlediska výpočetního výkonu v reálném čase, jsou pro elektrické střídavé pohony dostačující mikrokontroléry 16-bitové s pevnou řádovou čárkou. Na trhu se ale poslední dobou objevují mikroprocesory 32-bitové s možností plovoucí řádové čárky [3]. Obrázek 1.1 znázorňuje blokové schéma pohonu s použitím mikroprocesoru.



Obr. 1.1: Hardwarová konfigurace měniče elektrického pohonu. Převzato z [3].

V následující části budou uvedeni někteří výrobci mikrokontrolérů a jejich konkrétní typy, které nalézají uplatnění v elektrických pohonech.

1.1 Texas Instruments

Od této firmy nalézá v pohonech uplatnění především série C2000 založená na stejnojmenné architektuře.

Mikrokontroléry C2000 od firmy TEXAS INSTRUMENTS jsou vysoce výkonné mikroprocesory pracující v reálném čase určené k řízení výkonové elektroniky a dále umožňují pokročilé zpracování digitálního signálu v průmyslových a automobilových aplikacích [4].

Jedná se o 32-bitové mikroprocesory, které nalézají uplatnění například v nabíjecích stanicích pro automobily (AC/DC), DC/DC měničích, v trakčních pohonech či v posilovačích řízeních.

Pro řízení pohonů se používají vývojové kity *LaunchPad* s těmito mikrokontroléry, ke kterým lze přímo pomocí konektoru připojit výkonový měnič *BOOSTXL-DRV*, datasheet [5]. Tyto měniče slouží k řízení 3-fázových bezkartáčových DC motorů - BLDC nebo PMSM. Výkonové prvky zde tvoří tranzistory MOSFET. Tyto měniče také umožňují měřit fázové napětí a také pomocí bočníku přímo měřit proud protékající fázemi. Některé vývojové kity, jako například *LaunchPadXL F28069M* podporuje snímání impulzů z enkodéru.

1.2 NXP

1.2.1 Architektura DSP56800

Jsou to mikrokontroléry o velikosti jádra (ALU) 16 bitů od firmy NXP. Poskytují poměrně dobrý výpočetní výkon za nízkou cenu a proto jsou velice rozšířené v moderních pohonech. DSP56800 dokáže operovat s celočíselnou i zlomkovou aritmetikou.

Jádro tohoto mikrokontroléru je složeno z funkčních jednotek pracujících paralelně ke zvýšení jeho výpočetní propustnosti [3].

Tento mikrokontrolér nalezneme například ve vývojovém kitu DSP56800 TDC kit od firmy NXP.

1.2.2 Architektura ARM

Pro řízení elektrických pohonů jsou na této architektuře jsou založeny také mikrokontroléry z rodiny Kinetis KV. Tyto mikrokontroléry jsou vhodné pro BLDC, PMSM a asynchronní motory.

1.3 Arduino

Také platforma Arduino nalézá široké uplatnění v elektrických pohonech, především v robotice. Můžeme pomocí nich řídit například DC motor, krokový motor, servomotor nebo BLDC motor. Arduino nám nabízí mnoho vývojových desek, například *Uno, Mega, Leonardo,* které využívají mikrokontroléry z rodiny ATmega od firmy Atmel. K vývojovým deskám Arduino připojujeme přímo pomocí konektorů další desky tzv. shieldy.

Pro řízení dvou DC motorů můžeme využít například *Arduino motor shield L298P 2A*, který je kompatibilní s deskami *UNO* a *Mega2560*.

Pro krokový motor můžeme použít *Arduino 30A motor shield VNH3ASP30*, který je kompatibilní s *UNO* a *Leonardo*.

2 Způsoby programování pomocí RCP

Jak již z názvu vyplývá, metoda RCP (Rapid control prototyping) je obecně založena na automatickém generování kódu z vývojového softwaru. Kód pro daný mikrokontrolér je odvozen z vývojového softwaru a je nám umožněno rychle a jednoduše programovat. Konkrétnější metoda, tedy generování kódu z vytvořeného blokového schématu modelu, se nazývá Model Based Design (MBD). Nyní si rozebereme konkrétní vývojové nástroje používané k RCP.

2.1 Prostředí MATLAB-Simulink

MATLAB (matrix laboratory) je mocný nástroj, který kombinuje možnost vytváření programů a výpočetní operace. Jak již z názvu vyplývá, výpočetní operace provádí pomocí matic. MATLAB obsahuje simulační software nazývaný Simulink. Ten slouží k vytváření modelů dynamických soustav a k jejich simulacím. K rapid control prototypingu v prostředí MATLAB-Simulink jsou využívány nástroje MATLAB Coder, Simulink Coder a Embedded Coder. Pro programování mikrokontroléru se uplatňují Simulink Coder a Embedded Coder.

MATLAB/Simulink k rychlému programování (RCP) využívá metodu založenou na vytvoření simulačního modelu, tedy MBD.

2.1.1 Simulink Coder

Simulink Coder nebo-li Real-Time Workshop je nástroj, který generuje kód v jazyce C a C++ z modelů v Simulinku [6]. Nástroj RTW umožňuje vytvořit z blokového schématu program, odeslat do cílového zařízení a následně externě řídit pomocí Simulinku. Máme-li v modelu blok Scope, můžeme sledovat v reálném čase změnu signálu při změně vstupních parametrů simulace.

Real-time workshop umožňuje následující real-time vývojové funkce [7]:

1. Rychlou a přímou cestu od návrhu k implementaci
2. Bezproblémovou integraci s Matlabem a Simulinkem
3. Jednoduché a snadno použitelné rozhraní
4. Otevřenou a rozšiřitelnou architekturu
5. Plně konfigurovatelný generátor kódu
6. Rychlou iteraci úpravou blokových schémat a automatické vytváření nově spustitelného souboru

2.1.2 Embedded Coder

Embedded Coder je nástroj, který rozšiřuje Simulink Coder a umožňuje rychlé vygenerování kódu v jazyce C nebo C+ pro nejpoužívanější embedded procesory. Také nabízí ovladače pro hardware z rodiny například Altium, ARM, NXP, STM nebo TI [8].

2.2 LabVIEW

LabVIEW, od firmy NATIONAL INSRUMENTS (NI), je software pro tvorbu modelů soustav pomocí blokových schémat. Využívá se pro aplikace, které vyžadují řízení v reálném čase. Také umožňuje automatické generování kódu z blokového schématu, jako v případě Simulinku.

Pro generování kódu v jazyce C se používá nástroj LabVIEW C Generator. Soubor vzniklý v prostředí LabVIEW se nazývá virtual instruments (VI). Výše zmíněný generátor z takového souboru vygeneruje kód v jazyce C pomocí funkce Build Specification. Po vytvoření projektu a C kódu s VI souboru můžeme vytvořit další soubor VI do tohoto projektu [9].

V prostředí LabView se pro RCP uplatňuje nástroj zvaný Quanser. Podporované HW platformy jsou od například NI myRIO nebo NI cRIO firmy NI. Dále prostředí LabView obsahuje embedded modul pro podporu mikrokontrolérů ARM.

2.3 Altair Embed

Kromě automatického generování kódu z blokového schématu, dokáže tento software snadno testovat připojený harware pomocí funkce HIL (Hardware-in-the-loop). Toto prostředí podporuje mikrokontroléry, například Texas Instruments (C2000), Atmel (ATmega) nebo Arduino.

2.4 Powersim

Je další software pro tvorbu blokových schémat. PSIM pro generování kódu C používá nástroj zvaný SimCoder module. Obecně automatické generování kódu zahrnuje následující postupy [10]:

1. Design a simulace systému pomocí PSIM s řízením v reálném čase
2. Převede kontrolní sekci systému na diskrétní oblast a zajišťuje její simulaci
3. Pokud není připojen hardware, umístí kontrolní sekci do subobvodu a vygeneruje kód

4. Pokud je připojen hardware, upraví systém podle hardwarových prvků a spustí simulaci pro ověření výsledků. Poté vygeneruje kód.

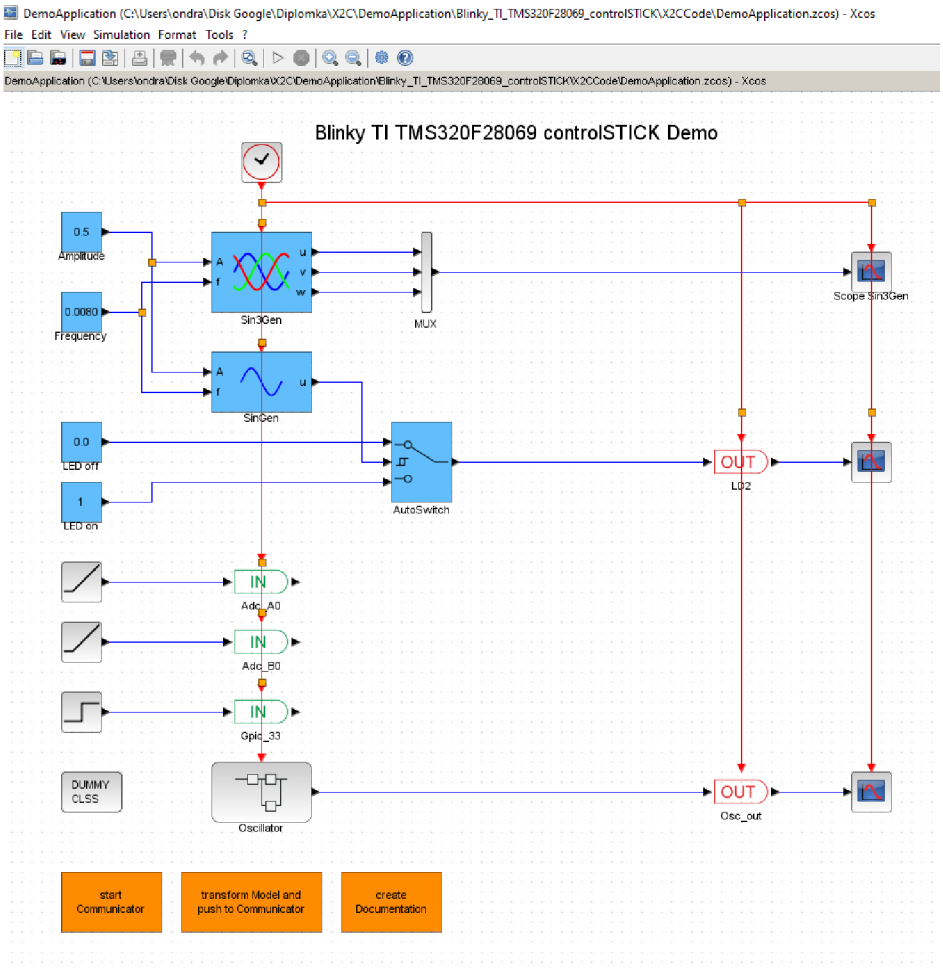
Powersim podporuje jako hardware zahrnující mikroprocesory typu TMSF od firmy TEXAS INSTRUMENTS.

2.5 X2C

Jedná se o software vyvinutý v centru pro mechatroniku v Linci (LCM). X2C zajišťuje následující vlastnosti [11]:

1. Podporuje Scilab/Xcos a Matlab/Simulink pro vytváření blokových schémat.
2. Na základě vytvořeného modelu je generován kód v C, který může být kompilován v rámci cílového integrovaného vývojového prostředí (IDE).
3. Generovaný kód je snadno čitelný.
4. Obsahuje virtuální osciloskop X2C Scope. Je to nástroj pro vizualizaci a vyladění dat online.
5. Nástroj X2C Communicator umožňuje vytvoření kódu v jazyce C a nahrání do cílového zařízení.

Na jednoduchém příkladu bude uveden způsob automatického programování pomocí metody MBD s využitím vývojové desky LaunchPadXL F28069M od firmy TEXAS INSTRUMENTS. Nejdříve je potřeba stáhnout X2C Free z oficiálních stránek *www.x2c.lcm.at*, soubor rozbalit a ve Scilabu otevřít složku. V přiloženém souboru pdf se nacházejí pokyny k instalaci prostředí X2C. Pro vývojovou desku LaunchPadF28069M je v adresáři X2C ve složce *DemoApplication* příklad *Blinky TI TMS320F28069controlSTICK*. Ve Scilabu je ve složce *X2CCODE* soubor *DemoAplicaton.zcos* který otevře v novém okně již hotové blokové schéma pro blikání LED. Pomocí oranžových bloků lze otevřít komunikátor, transformovat model a v komunikátoru se vytvoří kód, který již bude v jazyce C. V *CodeComposerStudio* je třeba importovat projekt. Nyní už jen zbývá projekt sestavit příkazem *Build all* a pomocí příkazu *Debug* nahrát do hardwaru. Červená LED na vývojové desce začne blikat. X2C lze používat i v prostředí MATLAB/Simulink, je ale nutno vlastnit verzi X2C Professional. Blokové schéma DEMO příkladu je uvedeno na obrázku 2.1



Obr. 2.1: DEMO aplikace blikání LED v prostředí SCILAB/X2C

2.6 instaSPIN

Pro vývojové desky od TEXAS INSTRUMENTS lze použít prostředí instaSPIN. InstaSPIN již nevyužívá metodu tvorby blokových schémat (MBD), zde nastavujeme parametry motoru do tzv. Graphical User Interface (GUI) a zde také provádíme samotné řízení motoru. Toto prostředí umožňuje řízení 3-fázových motorů, jak indukčních, tak stejnosměrných BLDC nebo PMSM. Dělíme ho dále na instaSPIN-FOC (field-oriented-control) a instaSPIN-MOTION. První zmíněné prostředí FOC umožňuje identifikovat motor připojený k měniči, nastavit a plně řídit 3-fázový motor, např. v aplikacích kompresorů nebo pump. Prostředí MOTION je vhodnější v aplikacích, které vyžadují přesně nastavení rychlosti a polohy, např. v robotice nebo CNC strojích. InstaSpin FOC a MOTION podporuje vývojové kity LaunchPad a měniče například DRV8301-69M-KIT nebo DRV8312-69M-KIT. Pro ostatní měniče například BOOSTXL-DRV8301 nebo BOOSTXL-DRV8305EVM není prostředí instaSPIN-FOC ani MOTION kompatibilní. Lze pro ně využít instaSPIN UNIVERSAL, který ale nedokáže automaticky rozpoznat parametry motoru a tak je musíme nastavit v souboru kódu jazyka C, který následně do instaSPIN UNIVERSAL nahrajeme. [12].

3 Způsob tvorby aplikace pro mikrokontroléry C2000 TI v Simulinku

Pro realizaci řízení pohonu je pro diplomovou práci zvolen mikrokontrolér *TMS320 F28069M* ze série C2000 na vývojové desce LaunchPadXL od firmy Texas Instruments a jako vývojové prostředí poslouží Simulink. Předpokládáme, že MATLAB-Simulink již máme nainstalovaný včetně Simulink coderu i Embedded coderu. Na oficiálních stránkách TI stáhneme a nainstalujeme software controlSuite pro podporu mikrokontrolérů TI. Následně stáhneme a nainstalujeme vývojové prostředí Code Composer studio (CCS), které Simulink potřebuje pro nahrání kódu do mikrokontroléru. Jako poslední krok v MATLABu pomocí Add-On manageru nainstalujeme Embedded Coder Support Package for TEXAS INSTRUMENTS C2000 Processors a nastavíme. V prvním kroku nastavení vybereme příslušnou vývojovou desku, v následujících krocích si MATLAB automaticky najde cestu k nainstalovanému controlSuite, Code Composer studiu a TI C2000 Code generation Tools kompiléru, který se nám nainstaloval spolu s CCS. Po správném nastavení můžeme již používat Simulink.

3.1 Knihovny a bloky Simulinku pro C2000 zařízení

Z klasické nabídky Simulinku můžeme využít ty bloky, které nevyžadují simulaci v reálném čase, ale fungují i v čase diskrétním. Můžeme využít většinu nejpoužívanějších bloků simulinku, jako například *Constant, Sum, Gain, Scope, Mux, Demux, Data Type Conversion, In, Out*. Pozor si ale musíme například dát na *Integrator*, který použít nemůžeme. Místo něj musíme zvolit blok *Discrete-Time Integrator*.

Jako nadstavbové knihovny pro produkty řady C2000 nás budou zajímat především knihovny *Embedded Coder* a *Emedded Coder Support Package for Texas Instruments C2000 Processors*.

První výše zmíněná obsahuje především bloky pro sériovou komunikaci SPI, pro komunikaci pomocí sběrnice CAN pro hostitelské zařízení, blok pro hardwarové přerušení a matematické funkce a transformace.

Ve druhé výše zmíněné knihovně nalezneme již bloky pro programování cílového zařízení, v našem případě nás bude zajímat knihovna pro *C2806x*. V této knihovně dopodrobna rozebereme nejdůležitější bloky:

- ***ADC*** - Tento blok slouží pro nastavení čtení analogového signálu přiváděného na *ADC* piny mikrokontroléru a převedení na digitální signál. Díky tomuto bloku můžeme například synchonizovat čtení vstupu se signálem PWM.

- ***Analog AIO Input*** - Slouží pro nastavení vstupu pro čtení analogového signálu na *AIO* pinech.
- ***Analog AIO Output*** - Nastaví *AIO* piny pro výstup analogového signálu.
- ***Digital Input*** - Nastaví vstupní digitální piny.
- ***Digital Output*** - Nastaví výstupní digitální piny. Pokud na vstup bloku přivedeme logickou hodnotu *True*, příslušný pin se přepne na hodnotu HIGH. Naopak logická *False* spojí pin s potenciálem země. Pokud v nastavení vybereme *Toggle mode*, příslušný pin přepne svoji aktuální úroveň vždy, když přivedeme logickou *True*. Naopak logická *False* nemá žádný vliv.
- ***ePWM*** - Tento blok umožňuje vytvořit na příslušných pinech pulsní šířkovou modulaci. Frekvenci i střídu můžeme buď nastavit na pevnou hodnotu a nebo ji můžeme přivádět přímo na vstup tohoto bloku. Blok nám umožňuje vybrat příslušný modul *PWM* a každý modul umožňuje nastavit dva výstupy: *ePWMA* a *ePWMB*. Část A je určena pro buzení horního tranzistoru ve větvi a část B pro buzení spodního tranzistoru. Výstupy A a B můžeme ale i navzájem prohodit. Nastavení ještě umožňuje zapnout a nastavit dead-time pro oba výstupy A i B a také povolení spouštěcích impulzů pro blok *ADC*
- ***eQEP*** - Pro podporu enkodéru je určen právě tento blok. Sleduje impulzy z enkodéru a díky nim dokáže určit informaci o pozici, směru a rychlosti mechanického otáčení.
- ***SCI Receive*** - Slouží pro obdržení dat z *SCIRXD* pinu a umožňuje sériovou komunikaci
- ***SCI Transmit*** - Data vstupující do tohoto bloku odesílá pomocí *SCITXD* pinu.

3.1.1 Hardware Interrupt

Tento blok nalezneme v knihovně *Emedded Coder Support Package for Texas Instruments C2000 Processors\Scheduling*. Umožňuje nám synchronizovat přerušení procesoru například s ADC předvodníkem nebo PWM moduly. Přerušení používáme tehdy, chceme-li přerušit činnost procesoru a vykonat příkaz obsažený v přerušení. Po vykonání obsluhy přerušení se opět obnoví předchozí činnost procesoru.

Do bloku zadáváme vektor hodnot, který reprezentuje CPU a PIE čísla dle tabulky viz [13]. Pomocí kanálu *ADCINTx* můžeme generovat přerušení pro ADC převodník.

Pro bloky tvořící tu část programu, kterou chceme přerušit, vytvoříme subsystém a do subsystému vložíme blok *Trigger*, který můžeme nechat defaultně nastavený. Nyní nám již půjde připojit výstup z *Hardware Interrupt* k vytvořenému subsystému.

3.1.2 ADC

Pomocí tohoto bloku můžeme číst ADC piny a nastavit ADC převodník. Čtení může probíhat buď v *Single sample mode* - čte jeden kanál nebo *Simultaneous sample mode* - čte dva kanály. Tyto kanály můžeme vybrat v záložce *Input channels*. Každému kanálu musíme přiřadit svoje vlastní *SOC trigger number*, které identifikuje spouštění *SOC - Start of conversion*.

Položka *SOC Aquisition number* určuje čas v hodinových cyklech, který ADC převodník potřebuje pro zachycení vstupního signálu. Můžeme nechat defaultně 7.

Důležitým je nastavit, jaká událost bude spouštět *SOC*. To určuje položka *SOCx trigger number*. Zde můžeme nastavit buď *Software, CPU Timers, XINT2* - což znamená spouštění pomocí externího pinu XINT2 SOC. Nás bude zajímat především *ePWM*, které umožňuje synchronizovat ADC převodník se spínáním PWM. Můžeme zde vybrat *ePWM* 1-8 pro modul A i B.

Jako *Sample time* můžeme nastavit buď -1 pro asynchronní vzorkování nebo zvolit čas mezi jednotlivými vzorky. Můžeme si i zvolit, jaký typ dat bude ADC generovat, například *Double, Single, int16, uint16....*

Přerušujeme-li ADC pomocí *Hardware interrupt*, zvolíme v položce *Interrupt selection* příslušný kanál *ADCINTx*.

3.1.3 ePWM

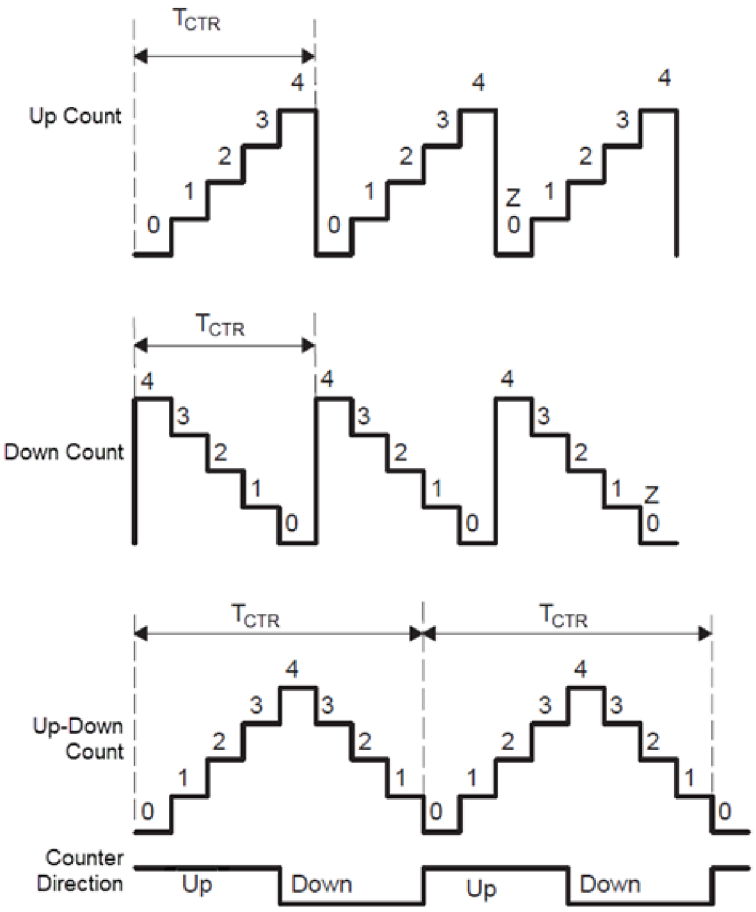
Tento blok slouží k nastavení a generování PWM. V následujících odstavcích bude vysvětleno nastavení v jednotlivých záložkách.

General

Tato záložka slouží pro všeobecné nastavení. Vybíráme zde, který modul bude daný blok využívat. Máme na výběr od *ePWM1-12*. Dále jednotku periody čítače *(Timer period)* buď sekundy nebo hodinové cykly. Periodu můžeme buď pevně nastavit v bloku a nebo modulovat pomocí vstupního signálu v Simulinku. V obou případech je potřeba nastavit počáteční periodu signálu pomocí políčka *Timer period units*. Abychom mohli dobu periody určit, je potřeba správně nastavit *Count mode*, jakým způsobem bude vytvářen nosný signál. Možnosti jsou 3 - *Up, Down* nebo *Up-Down* viz obrázek 3.1 [14].

Všechny možnosti výpočtu periody pro různé kombinace *Count mode* a *Timer period units* jsou v tabulce viz [14].

Dále zde můžeme nastavit například děličku časové základny nebo prohození ePWM modulu A a B volbou *Enable swap module A and B*, které uplatníme například při bipolárním řízení měniče.



Obr. 3.1: Způsob tvorby nosného PWM signálu

ePWMA, ePWMB

Vývojové kity od firmy TI jsou koncipovány vždy tak, že po připojení konektory k měniči ovládá výstup A vždy horní tranzistor ve větvi a výstup B spodní tranzistor. Tato záložka slouží pro nastavení ePWM výstupu A i B. Pro povolení toho výstupu musíme nejdříve zaškrtnout políčko *Enable ePWMxA* resp. *Enable ePWMxB*, kde x představuje číslo modulu, který jsme nastavili v záložce *General*. Samotná PWM se vytváří na základě porovnávání hodnot nosného trojúhelníkového signálu a hodnoty uložené v registru *CMPA* případně *CMPB*. V následujících 6 možnostech určujeme, jaký příkaz se provede v případě splnění podmínky:

- ***Action when counter=ZERO*** - Pokud hodnota čítače dosáhne 0.
- ***Action when counter=period (PRD)*** - Pokud hodnota čítače dosáhne periody (vrcholu nosného trojúhelníkového signálu).
- ***Action when counter=CMPA on up-count (CAU)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPA* při vzestupném počítání.

- ***Action when counter=CMPA on down-count (CAD)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPA* při sestupném počítání.
- ***Action when counter=CMPB on up-count (CBU)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPB* při vzestupném počítání.
- ***Action when counter=CMPB on down-count (CBD)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPB* při sestupném počítání.

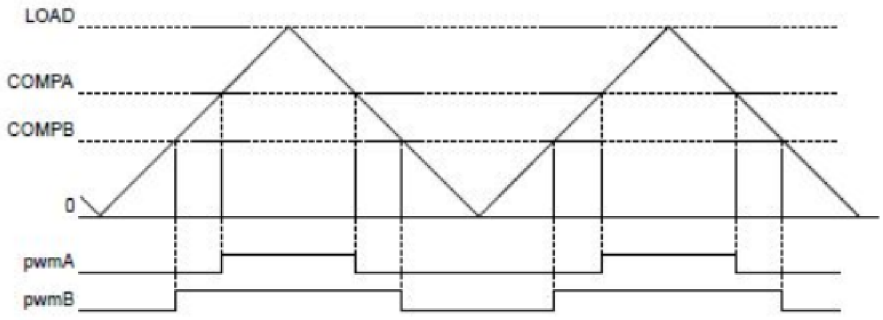
Pro každou možnost můžeme nastavit příkaz:

- ***Do nothing*** - Neprovede žádnou akci.
- ***Clear*** - Na výstup A vygeneruje 0.
- ***Set*** - Na výstup A vygeneruje 1.
- ***Toggle*** - Přepne aktuální logickou úroveň.

Tyto možnosti jsou shodné pro výstup A i B. Dále zde ještě můžeme nastavit, kdy budeme nahrávat novou střídu pomocí *Compare value reload condition*. Tu můžeme nahrávat buď v okamžiku, kdy se hodnota čítače rovná 0, periodě a nebo při obou rovnostech. Pro výstup B se nám ještě naskýtá možnost invertovat B oproti A. Ovšem v tomto případě nefunguje ruční nastavení doby odskoku *deadtime*.

Jako příklad se můžeme podívat na obrázek A.3 [15]. V tomto případě využíváme módu *Up-Down* a PWM pro výstup A se generuje od okamžiku rovnosti *CAU* po okamžik *CAD*. To stejné platí pro výstup B, od okamžiku *CBU* pro *CBD* a nová střída se nahrává v periodě nosného signálu.

Pozn.: Tento příklad je pouze ilustrační, v případě buzení tranzistorů výstupy A a B umístěných nad sebou by došlo ke zkratu ve větvi. V praxi požadujeme mezi A a B ochranou dobu *deadtime*.



Obr. 3.2: Způsob tvorby PWM signálu

Counter Compare

V této záložce nastavujeme hodnoty registrů *CMPA* a *CMPB*, tzn. nepřímo střídu. Pro oba registry můžeme nastavit buď *Seconds* nebo *Clock cycles*. Stejně jako v pří-

padě periody si můžeme vybrat mezi uložením hodnoty pevným nastavením v bloku *(Specify via dialog)* nebo vstupním portem *(Input port)* pomocí signálu v Simulinku. Hodnota uložena v registru *CMPA* nebo *CMPB* by neměla přesáhnout hodnotu uloženou v *Timer period*, protože jinak by porovnávaný signál přesáhl maximální hodnotu čítače a nikdy by nedošlo k jejich rovnosti.

Deadband unit

Ve většině případů chceme mezi výstupem A a B umístit ochrannou dobu, kdy nebude generována ani jedna PWM, čili dobu *deadtime*. Nejdříve zatrhneme obě políčka *Use deadband for ePWMxA* i *ePWMxB*. Aby ochranná doba správně fungovala, nesmíme mít zvolenou možnost invertování B oproti A. Signál B můžeme invertovat pomocí nastavení podmínek v záložce *ePWMB* nebo pomocí položky *Deadband polarity*, kde vybereme *Active high complementary*. I ochrannou dobu můžeme buď pevně nastavit v bloku a nebo pomocí vstupního portu pomocí signálu v Simulinku.

Dále nastavíme, pro který výstup PWM bude aplikovaná *deadtime* při vzestupné i sestupné hraně signálu. Ochranný čas nastavujeme pomocí hodnoty zapsané v registru *RED - Raising edge* pro vzestupnou hranu a *FED - Falling edge* pro sestupnou hranu. Pro výsledný čas odskoku v sekundách platí vztah:

$$t_{Deadtime} = \frac{1}{f_{proc}} \cdot RED \tag{3.1}$$

kde f_{proc} je pracovní frekvence procesoru a RED je hodnota uložená v tomto registru.

Stejný vztah platí i pro čas *Falling edge - FED*.

Event Trigger

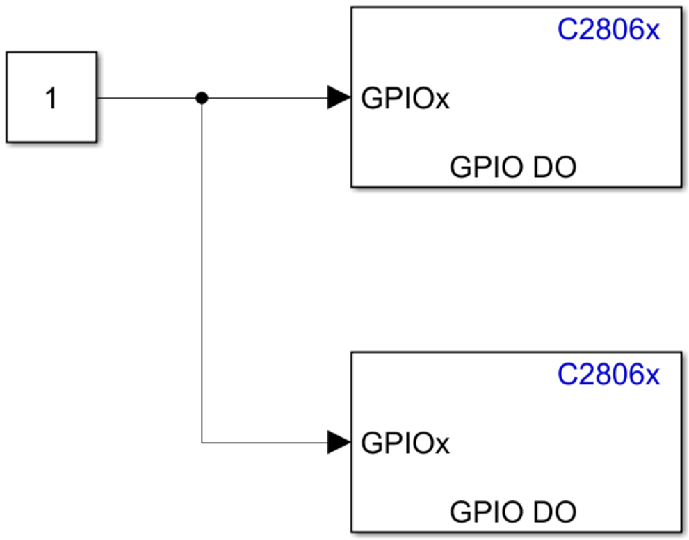
Umožňuje nastavit spouštění *SOC* pro A i B. Pro každý modul lze nastavit událost, při které se spustí *Start of conversion*:

- ***Counter equals to zero (CTR=ZERO)*** - Pokud je hodnota čítače rovno 0.
- ***Counter equals to period (CTR=PRD)*** - Pokud hodnota čítače dosáhne periody (vrcholu nosného trojúhelníkového signálu).
- ***Counter equals to zero or period (CTR=ZERO or CTR=PRD)*** - Pokud se hodnota čítače rovná 0 nebo periodě.
- ***Counter is incrementing and equals to the compare A register (CTRU=CMPA)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPA* při vzestupném počítání.

- ***Counter is decrementing and equals to the compare A register (CTRU=CMPA)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPA* při sestupném počítání.
- ***Counter is incrementing and equals to the compare B register (CTRU=CMPB)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPB* při vzestupném počítání.
- ***Counter is decrementing and equals to the compare B register (CTRU=CMPB)*** - Pokud se hodnota čítače rovná hodnotě v registru *CMPB* při sestupném počítání.

3.2 DEMO Program

V Simulinku můžeme přímo jako nový model otevřít *Code generation system* v záložce *Embedded Coder*. Nyní zbývá už jen pomocí *Configure Parameters* nastavit v záložce *Hardware Implementation* typ používaného hardwaru, v našem případě *TI Piccolo F28069M LaunchPad*. Nyní je již vše nastaveno a můžeme si správnost ověřit jednoduchým modelem, který nahrajeme do mikrokontroléru.



Obr. 3.3: Model blikání LED

Blokové schéma vytvoříme dle obrázku 3.3, kde blok *Constant* nastavíme na hodnotu 1, *Sample time* na hodnotu 0,5 a v *Signal Attributes* nastavíme jako *Output data* type *uint16*. Další 2 bloky jsou *Digital Output* z knihovny *Embedded Support Package for Texas Instruments C2000*, kde dále vybereme knihovnu pro náš hardware *C2806x*. V této knihovně jsou bloky kompatibilní s naším mikrokontrolérem.

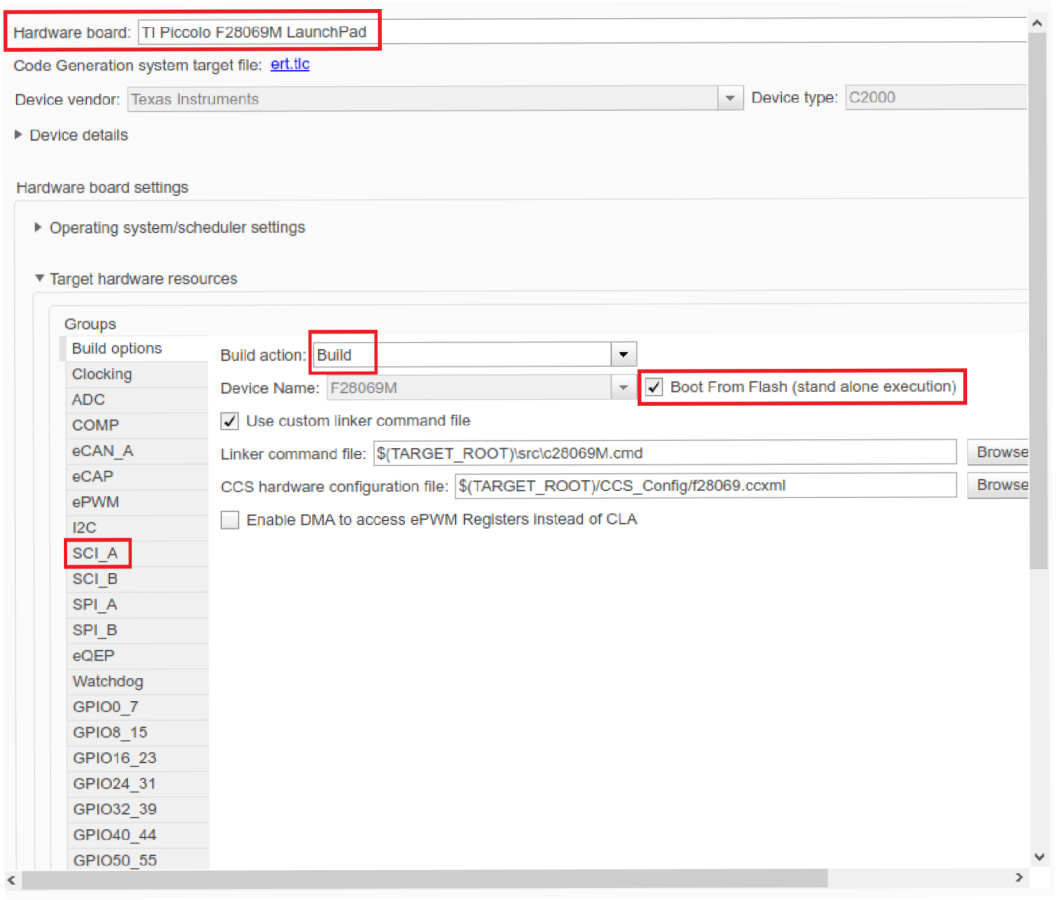
V obou blocích ještě vybereme příslušný pin, na kterém je připojena dioda. Červená LED D9 je spojena s pinem *GPIO34* a modrá LED D10 je spojena s pinem *GPIO39*. V obou blocích také zatrhneme ještě možnost *Toggle GPIO34* resp. *GPIO39*. Nyní ještě můžeme zkontrolovat, zda máme nastavený solver *Fixed-Step disrete*, protože mikrokontrolér umí pracovat jen v diskrétním čase. Nyní soubor uložíme. Připojíme hardware k pc pomocí sériového USB portu a pomocí tlačítka *Build* nahrajeme program. Pokud jsme vše učinili správně, LED začnou blikat s frekvencí 1 Hz.

3.3 External Mode

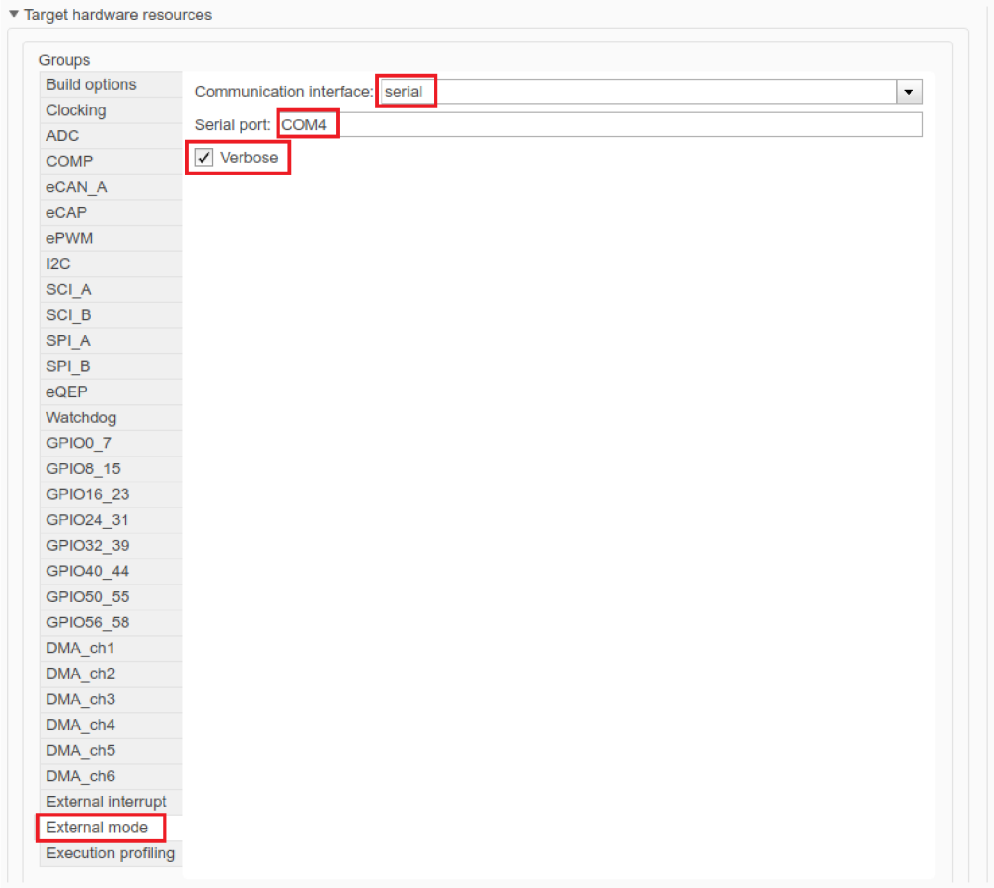
Prostředí Simulink taktéž umožňuje komunikaci s hardwarem v reálném čase. Můžeme tedy v Simulinku měnit parametry modelu během simulace a sledovat odezvy systému. Jako příklad nyní uvedeme jednoduchý model. Nejprve opět založíme nový soubor Embedded Coder a nastavíme příslušné parametry. V záložce *Hardware implementation* jako hardware board opět nastavíme *TI Piccolo F28069M LaunchPad* a dále budeme postupovat dle [16] viz obrázek 3.4.

V záložce SCI_A nastavíme parametr Desired baut rate in bits/sec: 115200. Jako poslední zbývá nastavit v záložce External Mode dle obrázku 3.5.

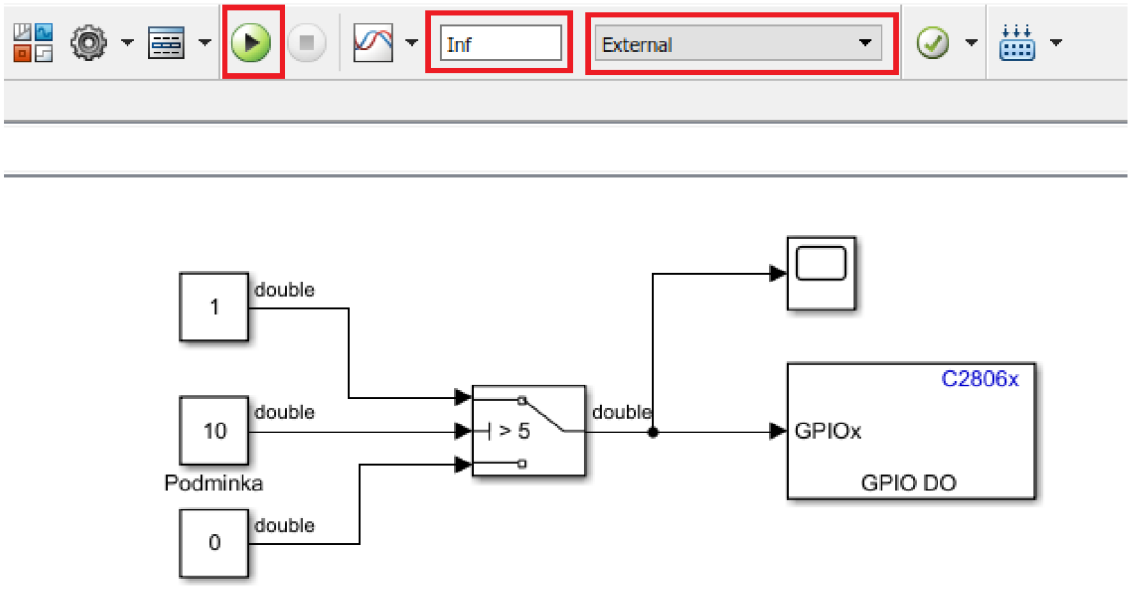
V našem případě byla deska připojena k počítači přes sériový port COM4. Tento sériový port se dá zjistit přes *Tento počítač - Vlastnosti - Správce zařízení - Porty(COM a LPT)*. Teď již můžeme přistoupit k samotnému modelu. Ten vytvoříme dle obrázku 3.6. U bloků *Constant* necháme defaultní nastavení, pouze nastavíme hodnotu *Constant value*. Podmínku bloku *Switch* nastavíme např. na hodnotu 5. Výstupní signál můžeme sledovat na *Scope* a připojíme ho k *Digital Output* a jako pin jsme vybrali GPIO19, který měříme voltmetrem případně osciloskopem. Nezapomeneme nastavit čas simulace na inf a external mode, viz obrázek 3.6. Simulaci spustíme tlačítkem *Run* a během simulace můžeme měnit parametry. Nastavíme-li *Constant value* bloku *Podminka* větší než 5, je na digitální pin přivedena hodnota *True*, pin se přepne na HIGH, tudíž na výstupu naměříme 3 V. Naopak, je li menší než 5, pin se uzemní a na výstupu naměříme 0 V. Parametry můžeme měnit i s pomocí *Workspace* MATLABu, pokud použijeme proměnnou a zapíšeme ji do příslušného bloku. Pokud se při prvotním spuštění simulace objeví error, je potřeba v nastavení v záložce *Code generation - Interface* odškrtnout položku *Remove error status field in real-time data model structure*.



Obr. 3.4: Nastavení parametrů



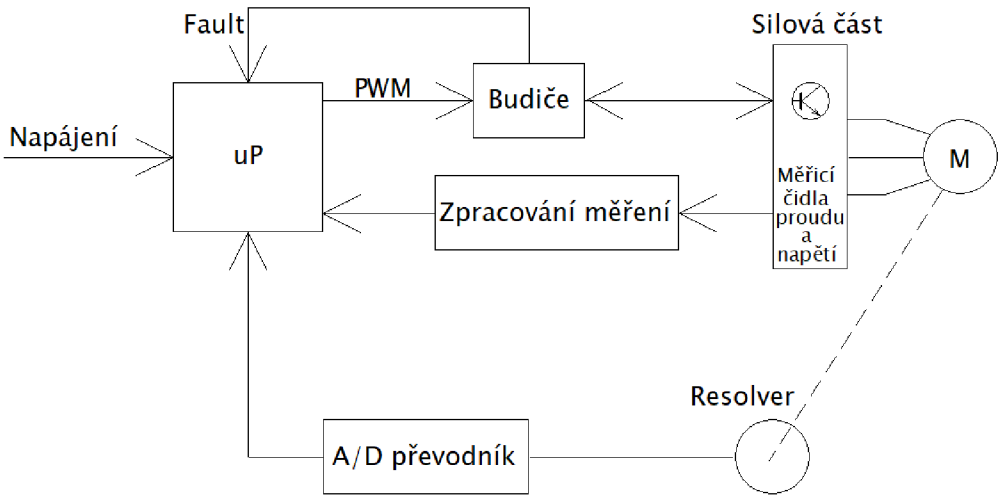
Obr. 3.5: Nastavení parametrů external



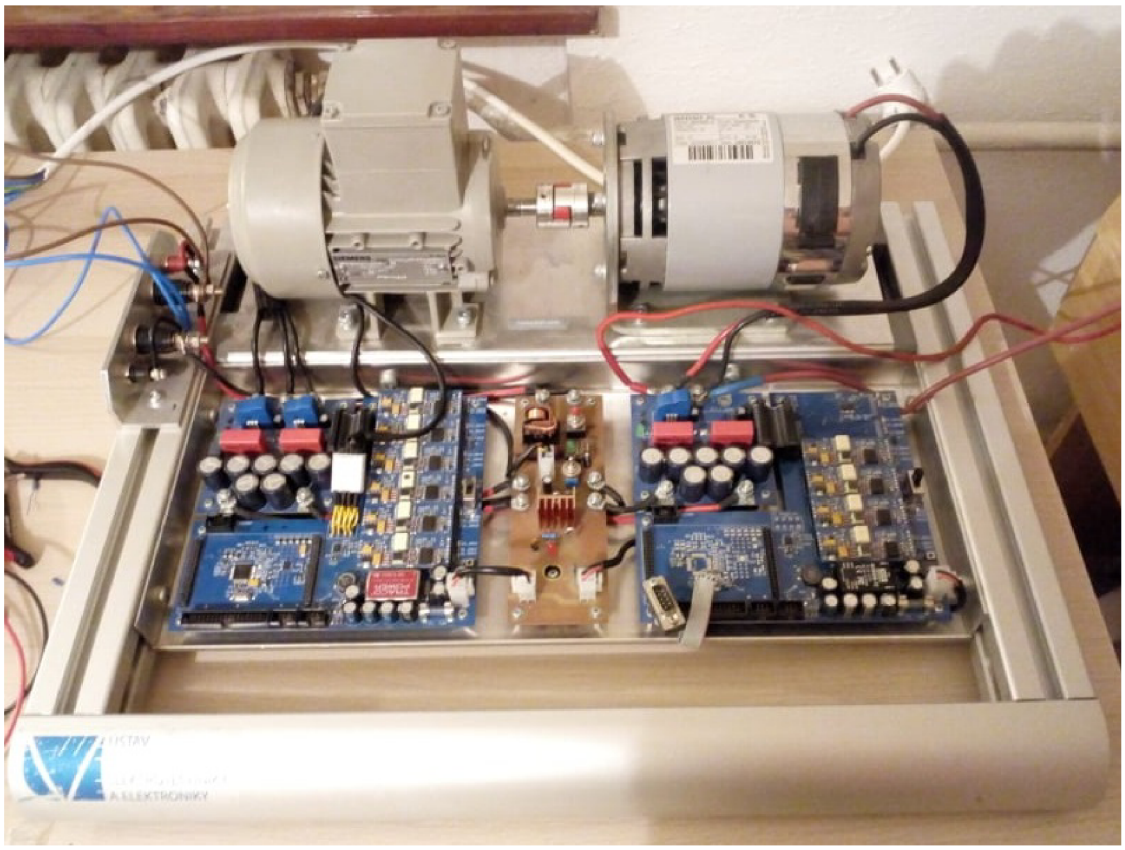
Obr. 3.6: Nastavení parametrů external

4 Seznámení s laboratorním přípravkem

Samotný laboratorní přípravek s motory obsahuje dva laboratorní standy, každý pro jeden motor viz obrázek 4.2. Laboratorní stand, který vznikl na Ústavu výkonové elektrotechniky a elektroniky, slouží k řízení asynchronního motoru a stejnosměrného motoru, které jsou mechanicky spojeny hřídelí. Tento stand je napájen ze zdroje 24 V DC a obsahuje řídicí, měřicí i silové obvody výkonového měniče. Ten je tvořen šestitranzistorovým můstkem. Laboratorní stand neumožňuje rekuperaci do sítě, napájecí jednotka obou standů obsahuje brzdný odpor. Blokové schéma laboratorního standu nalezneme na obrázku 4.1.



Obr. 4.1: Blokové schéma standu



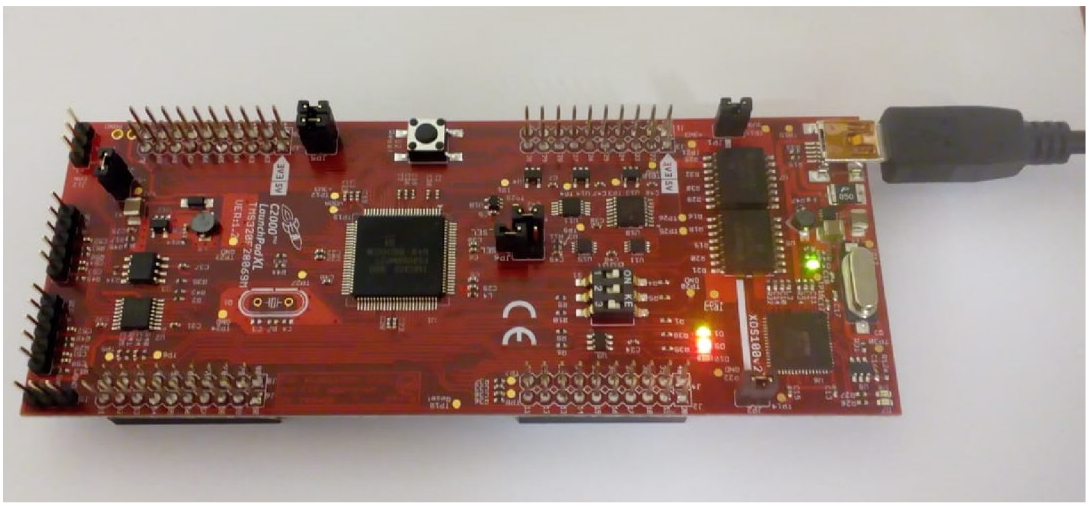
Obr. 4.2: Laboratorní stand s motory

4.1 Vývojový kit LaunchPad F28069M

Jak již bylo zmíněno v kapitole 3, pro realizaci aplikace na laboratorním přípravku byl zvolen vývojový kit LaunchPad F28069M od firmy Texas Instruments. Srdcem tohoto kitu je 32-bitový mikroprocesor TMS320F28069M s frekvencí 90 Mhz. Obsahuje 12-bitový ADC převodník s rozsahem 0-4096, kterému odpovídá napětí $0 - 3,3V$. Vývojový kit dokáže dohromady číst až 14 kanálů. Disponuje 8 moduly ePWM, každý modul potom 2 kanály PWM.

Pro snímání otáček je vývojový kit vybaven dvěma moduly eQEP pro dvoukanálový enkodér s indexovacím signálem. Vývojový kit umožňuje přímé spojení s hostitelským zařízením pomocí rozhraní USB. Po sériová komunikace může být také vyvedena na piny vývojového kitu. Vývojový kit není bohužel vybaven zdrojem 3,3 V, napájení musí být provedeno buď z USB a nebo z externího zdroje.

Vývojový kit LaunchPad nalezneme na obrázku 4.3.



Obr. 4.3: Vývojový kit Launchpad

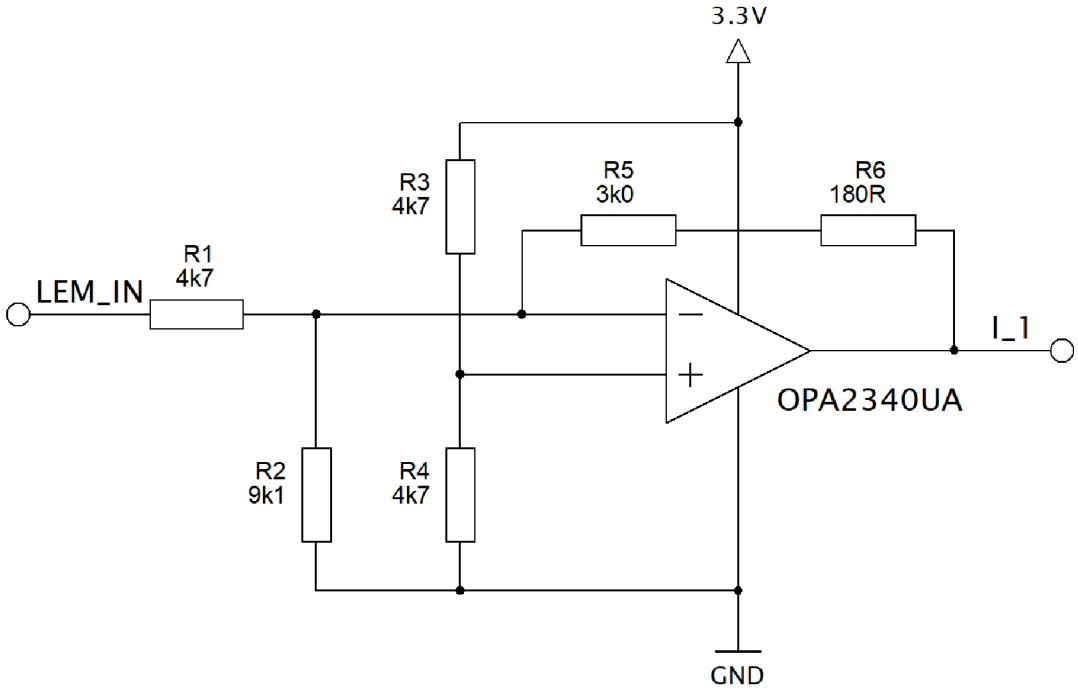
4.2 Redukce pro LaunchPad

Z důvodu nekompatibility konektorů laboratorního standu a vývojového kitu LaunchPad bylo nutno navrhnout redukci. Původně byla pro stand na ÚVEE navržena redukce pro připojení vývojového kitu *Nucleo STM 32*. Z tohoto předchozího návrhu převezmeme jednotlivé části do nové redukce pro LaunchPad. Bude se jednat především o zdroj 3,3 V, dále RC filtry pro měřené signály, signalizaci sepnutí PWM a také USB komunikaci. Schéma a návrh DPS nalezneme v přílohách A. Při návrhu se vycházelo především ze rozmístění pinů na konektoru standu, viz obrázek A.5

a vznikla tabulka ekvivalentních pinů pro LaunchPad, viz obrázek A.5. Po osazení DPS a oživení redukce bylo zjištěno, že zdroj 3,3 V poskytuje na výstupu pouze 3,15 V a proto na výstupu OZ čidla LEM v klidovém stavu nebude 1,65 V ale pouze 1,5 V. To musíme zohlednit pro správné vyhodnocení měřeného proudu.

4.3 Měření proudu

Pro měření proudu je laboratorní stand vybaven proudovými čidly LEM. Signál z čidla LEM_IN je nadále zpracováván a veden do invertujícího OZ pracujícího jako diferenční zesilovač, který upravuje rozsah měřeného proudového signálu při napájení 3,3 V od 1,65 V do 3 V podle potřeb vývojového kitu *LaunchPad*. Signál I_1 pak vede přímo do analogového vstupu vývojového kitu. Schéma takového zapojení nalezneme na obrázku 4.4. Stejným způsobem je měřen proud I_2.



Obr. 4.4: Invertující zesilovač pro čidlo LEM

4.4 Měření otáček

Pro měření otáček je laboratorní stand vybaven resolvérem na hřídeli s elektromotory.

Resolvér je elektromagnetické čidlo založené na principu transformátoru. Na rotoru je umístěno primární vinutí napájené sinusovým napětím a na statoru dvě sekundární vinutí, které tvoří cívky navzájem otočené o 90 stupňů. Při rotačním pohybu primárního vinutí se v sekundárním vinutí indukují dva napěťové signály fázově posunuté navzájem o 90 stupňů, a to sinus a cosinus. Tyto napěťové signály z resolveru dále vstupují do mikroprocesoru tzv. *Resolver-To-Digital-Converter*, který převede analogový signál na digitální a určí polohu a rychlost otáčení [17].

V našem případě jako A/D převodník slouží 12-bitový mikrokontrolér AD2S1200. Tento převodník vytváří virtuální enkodér se dvěma kanály A,B a s indexovacím kanálem. Počet pulsů za jednu otáčku je 1024 a maximální možné otáčky 1000 rps.

5 Realizace programu řízení DC motoru

Tato kapitola se bude zabývat postupem tvorby programu pro řízení DC motoru, snímání proudu a otáček a zpětnovazebnou regulaci. DC motor je připojen k H-můstku výkonového měniče laboratorního standu. Nejdříve si popíšeme způsob tvorby PWM pro unipolární a bipolární řízení můstku.

5.1 Tvorba PWM

PWM do programu zahrneme pomocí dvou bloků *ePWM*. Každý blok řídí jednu větev. Frekvenci pro PWM volíme 20 kHz. Jako jednotku periody zvolíme hodinové cykly *Clock cyckles* a počítací mód *Count mode* zvolíme *Up-down*. Potom pro periodu PWM platí [14]:

$$T_{CTR} = 2 \cdot T_{BPRD} \cdot T_{BCLK} \tag{5.1}$$

kde T_{BPRD} je hodnota, kterou vkládáme do parametru *Timer period* a T_{BCLK} je perioda jednoho cyklu mikroprocesoru. Protože máme mikroprocesor o frekvenci 90 MHz, bude potom:

$$T_{BCLK} = \frac{1}{90 \cdot 10^6} = 11,11 \text{ ns} \tag{5.2}$$

Pro frekvenci PWM 20 kHz bude T_{CTR} rovno:

$$T_{CTR} = \frac{1}{20 \cdot 10^3} = 50 \ \mu\text{s} \tag{5.3}$$

Potom bude T_{BPRD} rovno:

$$T_{BPRD} = \frac{T_{CTR}}{2 \cdot T_{BCLK}} = \frac{50 \cdot 10^{-6}}{2 \cdot 11,11 \cdot 10^{-9}} = 2250 \tag{5.4}$$

Toto je tedy hodnota, kterou vložíme v záložce *General* do políčka Timer period v obou blocích, abychom dostali požadovanou frekvenci 20 kHz a zároveň je to maximální hodnota, které čítač dosáhne a jedná se o jeho periodu.

5.1.1 Unipolární řízení

Nyní vysvětlíme, jak nastavit blok *ePWM* pro vytváření unipolárního řízení. Budeme navíc požadovat zařadit mezi horní a dolní tranzistor ochrannou dobu spínání 0,5 μs. Nastavení bloků pro modul *ePWM1* a *ePWM2* bude naprosto totožné.

V záložce *ePWMA* provedeme nastavení pro následující události. Pokud bude hodnota čítače nosného signálu rovné 0, nastavíme *Set*, což zajistí logickou 1 na výstupu A. Při dosažení periody nastavíme naopak příkaz *Clear*, který PWM přepne

na logickou 0, čili uzemní. Registr *CAU* nastavíme na *Clear* a registr *CAD* na *Set*. Tímto nastavením při vzrůstající hodnotě v registru *CMPA1* zvyšujeme střídu výstupu *ePWM1A*. Možnosti pro registr *CMPB* nevyplňujeme, necháme bez činnosti *(Do nothing)*. Nové nahrávání registru *CMPA1* můžeme nastavit pokud hodnota čítače bude 0. Stejným způsobem nastavíme záložku *ePWMB* s tím rozdílem, že místo registru *CAU* a *CAD* nastavujeme *CBU-Clear* a *CBD-Set* a pro *CMPA* necháme bez akce.

V další záložce *Counter Compare* budeme nastavovat hodnoty porovnávaných signálů. Abychom při řízení můstku dosáhli kladných i záporných napětí, budeme při vytváření střídy využívat střední hodnotu čítače, tedy 1125. Pro jednu větev tranzistorů (blok *ePWM1*) budeme k této hodnotě v registru *CMPA1* přičítat hodnoty až do 2250 a zároveň pro druhou větev (blok *ePWM2*) budeme hodnotu v registru *CMPA2* odečítat do 0. Tak bude dosaženo kladné polarity. Naopak záporné polarity výstupního napětí dosáhneme tak, že hodnotu v *CMPA1* odečítáme od 1125 do 0 a zároveň *CMPA2* zvyšujeme až k maximální hodnotě 2250 viz obrázek 5.5.

Registr *CMPA* budeme tedy modulovat pomocí vstupního portu a jako počáteční hodnotu nastavíme střed čítače, čili 1125. Registr *CMPB* nebudeme nijak ovládat, pouze nastavíme počáteční hodnotu opět 1125.

Nyní v záložce *Deadband unit* povolíme ochrannou dobu pro A i B. Jako *Deadband polarity* zvolíme *Active high complementary (AHC)*, které invertuje signál B oproti A. Registr *FED* i *RED* nastavíme pro modul *ePWMxA* a hodnotu registru pro ochrannou dobu spočítáme dle vzorce 3.1, kde vyjádříme hodnotu RED:

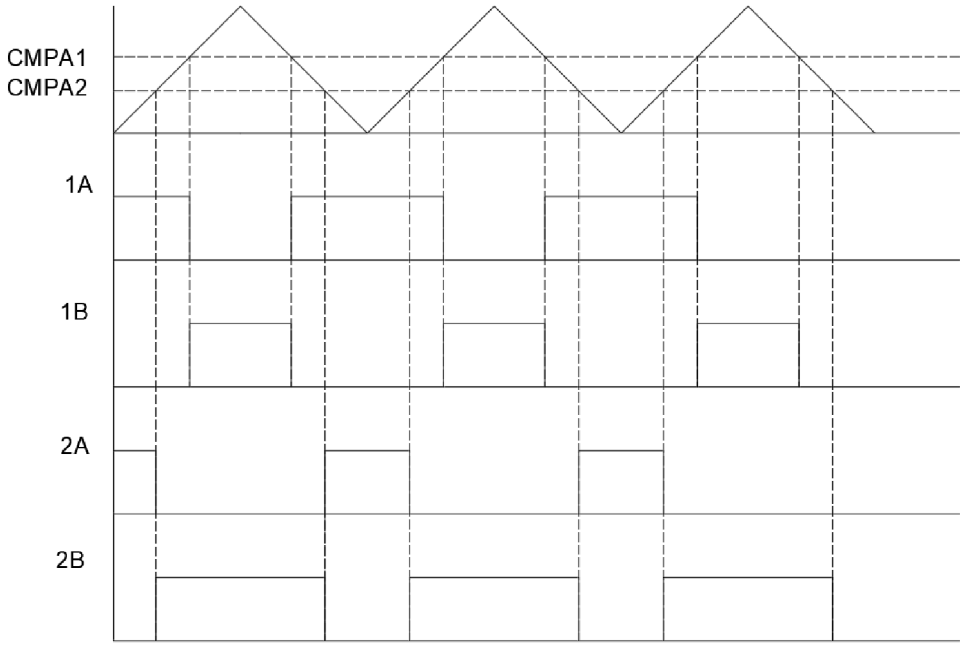
$$RED = t_{Deadtime} \cdot f_{proc} = 0,5 \cdot 10^6 \cdot 90 \cdot 10^6 = 45 \tag{5.5}$$

Hodnotu 45 tedy zapíšeme do registru *RED* i *FED*.

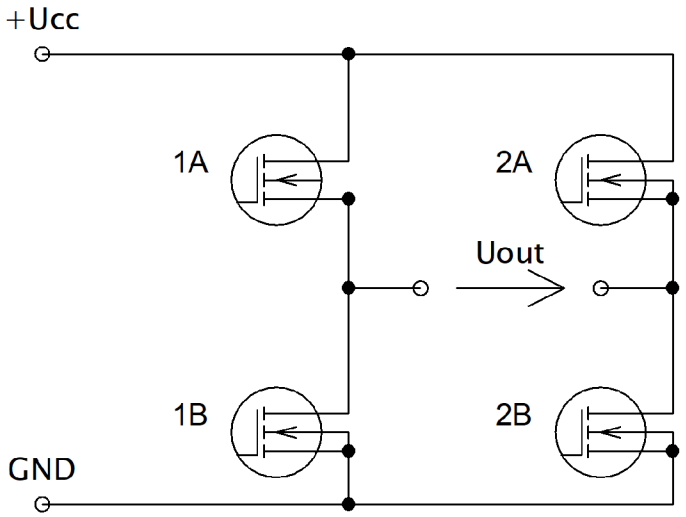
Na obrázku 5.1 vidíme, jakým způsobem se tvoří jednotlivé signály pro spínání tranzistorů.

A schéma označení tranzistorů v silové části na obrázku 5.2.

Jako poslední záložku si otevřeme *Event Trigger*, kde povolíme synchronizaci s ADC převodníkem. Toto provedeme pouze v bloku *ePWM1*, protože ADC převodník chceme synchronizovat s horním tranzistorem v první větvi. Takže zde zatrhneme políčko *Enable ADC start of conversion for module A*. Spouštění převodníku budeme provádět při každé události, takže ve druhém políčku zaškrtneme *First Event* a jako událost vybereme *Counter equals to zero (CTR=ZERO)*, tzn. že spouštění pro ADC převodník bude generováno vždy, pokud hodnota čítače nosného signálu bude rovno 0, čili horní tranzistor v první větvi sepnutý, viz obrázek 5.1 nebo 5.3.



Obr. 5.1: Způsob tvorby PWM signálu pro unipolární řízení

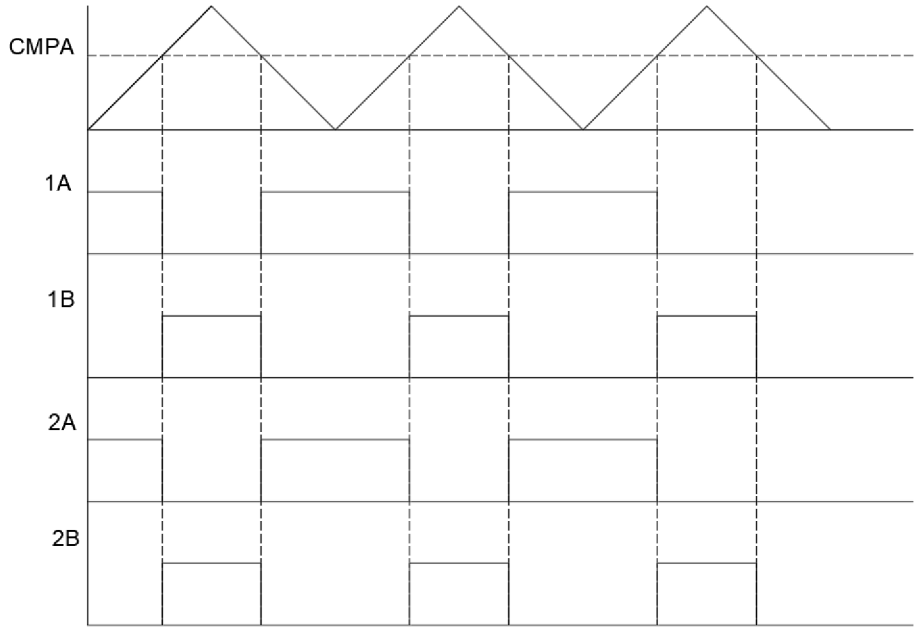


Obr. 5.2: Schéma označení tranzistorů H-můstku - unipolární řízení

5.1.2 Bipolární řízení

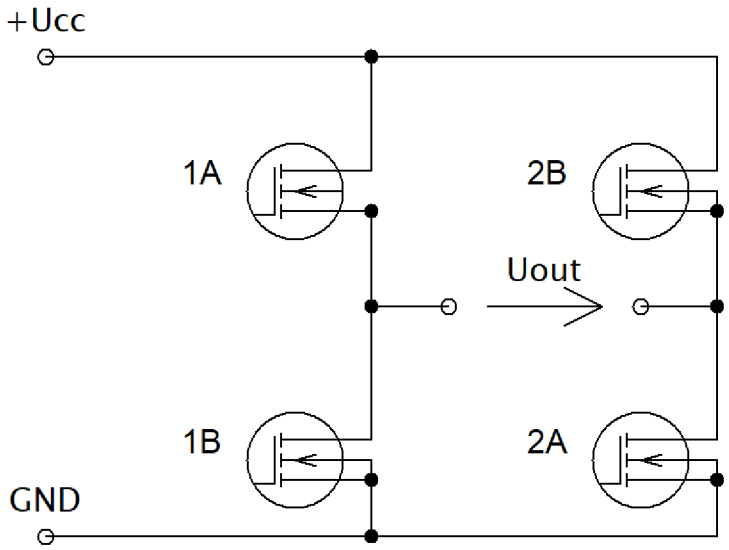
Pro bipolární řízení použijeme stejné nastavení jako pro unipolární, jen s rozdílem, že pro oba bloky používáme jednu hodnotu porovnávacího signálu *CMPA* a nesmíme zapomenout v jednom bloku *ePWM* zaškrtnout možnost *Enable swap module A and B*, abychom dosáhli totožného spínání tranzistorů v úhlopříčkách.

Výsledné buzení tranzistorů potom vypadá takto:



Obr. 5.3: Způsob tvorby PWM signálu pro bipolární řízení

A schéma tranzistorů:



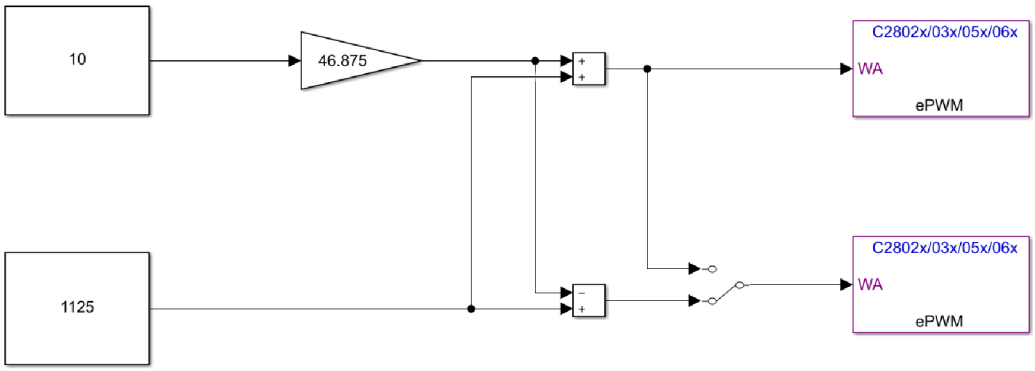
Obr. 5.4: Schéma označení tranzistorů H-můstku - bipolární řízení

5.1.3 Přepínání unipolárního a bipolárního řízení

Blokové schéma přepínání mezi jednotlivými typy řízení je zobrazeno na obrázku 5.5. Laboratorní stand napájíme napětím 24 V, čili při maximální střídě požadujeme toto napětí. Abychom mohli tyto hodnoty napětí přímo zapisovat do Simulinku, následuje za blokem *Constant* násobení 46,875, protože $24 \cdot 46,875 = 1125$ a to po přičtení střední hodnoty 1125 je rovno maximální hodnotě dosažitelné v čítači 2250. Pro bipolární řízení jsou pro oba bloky zapisovány stejné hodnoty do registru *CMPA*. Při unipolárním řízení přepneme manual switch a pomocí odčítání získáme dvě různé hodnoty *CMPA* viz kapitola 5.1.1. Nesmíme ještě zapomenout při požadavku unipolárního řízení odškrtnout v jednom bloku možnost *Enable swap A and B*.

5.2 Snímání proudu pomocí ADC převodníku

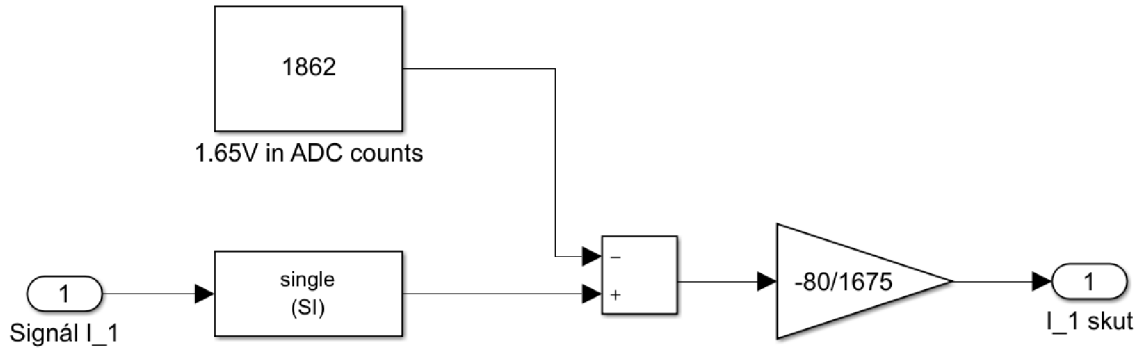
Spouštění ADC převodníku budeme synchronizovat s modulem *ePWM1A* tedy horním tranzistorem v první větvi. Měřit budeme vstup *ADCINB6* tzn. proud I_2 dle laboratorního standu viz obrázek A.6. Čidlo LEM, ze kterého snímáme proud, je připojeno k přívodnímu vodiči k motoru z uzlu první větve tranzistorů. Dále v bloku nastavíme číslo kanálu konverze *SOC0*, jako zdroj spouštění ADC převodníku zvolíme *EPWM1ADSOCA*, jak jsme zmínili výše. ADC převodník bude generovat přerušení,



Obr. 5.5: Blokové schéma přepínání řízení

proto musíme zaškrtnout políčko *Post interrupt at EOC trigger*, to znamená, že po skončení jednoho cyklu snímání ADC převodník vygeneruje přerušení. Pro přerušení můžeme vybrat hned první s názvem *ADCINT1*. Datový typ zvolíme *uint16*. Jelikož blok *ADC* umístíme do subsystému, který bude přerušován právě od ADC převodníku, musíme jako vzorkovací čas zvolit -1, tzn. že blok bude snímat proud asynchronně s přerušením. Toto nastavení funguje pro unipolární i bipolární řízení.

Jelikož je ADC převodník 12-bitový, výstupní signál nabývá hodnot od 0 do 4096, kde hodnota 2048 odpovídá 0 A. Dokážeme tedy měřit jak kladný tak i záporný proud. Musíme ale za výstup tohoto bloku zařadit přepočet na skutečnou hodnotu proudu v ampérech. Přepočet zachycuje obrázek 5.6.



Obr. 5.6: ADC přepočet na skutečnou hodnotu proudu

Čidlo LEM meří rozsah 0-80 A, kde 0 A odpovídá 1,65 V a 80 A odpovídá 3 V na výstupu OZ čidla LEM. Plný rozsah 3,3 V odpovídá hodnotě 4096 převodníku,

takže 3 V bude odpovídat hodnotě 3723 a 1,65 V odpovídá 2048. Jak bylo již napsáno v kapitole 4.2, 0 A odpovídá 1,5 V. Nejdříve posuneme referenční úroveň tak, aby 0 A, čili 1,5 V na výstupu OZ LEM odpovídalo hodnotě 0 měření. Pokud 1,65 V odpovídá hodnotě 2048, potom 1,5 V odpovídá hodnotě 1862. Musíme tedy od výstupu převodníku odečíst hodnou 1862. 80 A čidla LEM a 3 V na výstupu OZ odpovídá hodnotě 3723 - 2048 = 1675. Jelikož je OZ invertující, musíme ještě signál vynásobit -1. Takže signál dále násobíme číslem -80/1675. Tento převod na skutečnou hodnotu v ampérech nalezneme na obrázku 5.6.

5.2.1 Hardwarové přerušení

Hardwarové přerušení budeme synchronizovat s ADC převodníkem. Do části programu, který chceme přerušovat, vložíme blok *Trigger*. Přerušovat budeme subsystém s ADC převodníkem, tvorbou PWM, regulátorem proudu a otáček viz. obrázek B.2. V subsystému přerušení nyní vznikl nový vstupní port, ke kterému připojíme blok *Hardware interrupt*. Ten nastavíme dle tabulky [13] na ADCINT1. Číslo CPU bude 1 a číslo PIE také 1. Ostatní nastavení necháme výchozí.

5.3 Regulátor proudu

Podle metody optimálního modulu (OM) navrhneme regulátor proudu pro stejnosměrný motor, který jako zátěž představuje indukčnost a odpor. Hodnoty potřebné pro výpočet regulátoru nalezneme v tabulce 5.1:

L	R	f_{pwm}	K_m	K_i
[μH]	[Ω]	[Hz]	[-]	[-]
250	0.11	20000	24/1125	1

Tab. 5.1: Parametry proudové smyčky

Odpor a indukčnost dle [19]. Jako frekvenci PWM jsme zvolili 20 kHz. Konstantu měniče K_m jsme určili tak, že maximální napájecí napětí $U_{cc} = 24$ V odpovídá při nastavování střídy hodnotě 1125, celý rozsah $\pm$ 24 V odpovídá hodnotě od 0 do 2250. Snímaný proud máme již přepočítaný na skutečný, proto $K_i = 1$.

Pro přenos F_S otevřené proudové smyčky platí vztah:

$$F_S = \frac{K_m}{1 + p \cdot \tau_m} \cdot \frac{1/R}{1 + p \cdot T_a} \cdot K_i \tag{5.6}$$

kde T_a je časová konstanta motoru a τ_m časová konstanta měniče.

Jako časovou konstantu měniče τ_m budeme uvažovat dobu $T_{PWM}/2$ plus ještě dobu, za kterou nahrajeme střídu, tu budeme nahrávat, pokud hodnota čítače dosáhne 0. Potom τ_m:

$$\tau_m = \frac{T_{PWM}}{2} + \frac{T_{PWM}}{2} = T_{PWM} = \frac{1}{f_{PWM}} = \frac{1}{20000} = 50\ \mu\text{s} \tag{5.7}$$

Časová konstanta motoru T_a:

$$T_a = \frac{L}{R} = \frac{250 \cdot 10^{-6}}{0.11} = 2,3\ \text{ms} \tag{5.8}$$

Pro optimální modul F_o platí vztah:

$$F_o = \frac{1}{2 \cdot \tau_\sigma \cdot p \cdot (\tau_\sigma \cdot p + 1)} \tag{5.9}$$

kde τ_σ je časová konstanta OM

A pro výpočet regulátoru F_R platí:

$$F_R = \frac{1}{F_s} \cdot F_o \tag{5.10}$$

Po dosazení vztahů 5.6 a 5.9 do vztahu 5.13 bude pro výpočet regulátoru platit:

$$F_R = \frac{1 + p \cdot \tau_m}{K_m} \cdot \frac{1 + p \cdot T_a}{1/R} \cdot \frac{1}{K_i} \cdot \frac{1}{2 \cdot \tau_\sigma \cdot p \cdot (\tau_\sigma \cdot p + 1)} \tag{5.11}$$

Hodnotu τ_σ volíme tak, aby se nám vykrátila nejmenší časová konstanta v přenosu F_S, tedy 50 μs.

Po vykrácení bude vztah 5.11 vypadat následovně:

$$F_R = \frac{1 + p \cdot T_a}{K_m \cdot 1/R} \cdot \frac{1}{K_i} \cdot \frac{1}{2 \cdot \tau_\sigma \cdot p} \tag{5.12}$$

Po číselném dosazení bude vztah 5.12 vypadat následovně:

$$F_R = \frac{1 + 0,0023 \cdot p}{24/1125 \cdot 1/0.11 \cdot 1 \cdot 2 \cdot 50 \cdot 10^{-6} \cdot p} = \frac{1 + p \cdot 0,0023}{p \cdot 1,94 \cdot 10^{-5}} \tag{5.13}$$

Rozdělíme nyní tento zlomek na proporciální zesílení $P = 118$

A integrační zesílení I: 51546

Integrační zesílení v případě diskrétního PI regulátoru je děleno vzorkovací frekvencí, což plyne z principu diskrétního PI regulátoru. Pro vložení do bloku v Simulinku, musíme tuto hodnotu ještě vydělit samotnou frekvencí přerušení, která je shodná s frekvencí PWM.

$$I = \frac{51546}{20000} = 2,6 \tag{5.14}$$

Do modelu přidáme blok PID a vepíšeme do něj hodnoty PI regulátoru. V hlavní záložce bloku zaškrtneme *Discrete-time*. Ještě určíme meze saturace jako ± 1125. Integrátor omezíme pomocí *Anti-windup*, kde zvolíme *Clamping*. To zastaví integraci, pokud výstup překročí nastavené limity.

Na vstup PI regulátoru přivedeme rozdíl žádané a skutečné hodnoty. V nastavení *Configuration parameters/Code generation/Interface* musíme zaškrtnout možnost *Absolute time*, jinak regulátor nebude fungovat a Simulink bude generovat chybu.

5.4 Snímání otáček

Jak již bylo napsáno výše, laboratorní stand je vybaven resolvérem pro určení polohy a otáček. Analogový signál z resolvéru je převáděn na digitální a získáváme virtuální enkodér se dvěma kanály A a B, kde signál B je fázově posunutý o 90 stupňů vzhledem k signálu A. V tomto případě se jedná o kvadratický enkodér. Dále ještě dostáváme z enkodéru indexovací signál, který nám za každou jednu otáčku posílá jeden puls. V záložce *Hardware Settings/Target hardware resources/eQEP* nastavíme vstupy pro enkodér. Dle A.6 je signál z enkodéru přiveden na piny J2.12 a J2.13. Pro *EQEP1A* zde přiřadíme pin *GPIO50* a pro *EQEP1B* pin *GPIO51*.

V Simulinku budeme rychlost vypočítávat pomocí bloku *eQEP*, který vložíme do schématu. Jelikož jsme se rozhodli, že blok *eQEP* budeme spouštět externím signálem s frekvencí 0,005 s, musíme proto vytvořit z bloku subsystém a do vzniklého subsystému přidat blok *Function*, kde jako *Trigger type* vybereme náběžnou hranu, tedy *Rising*. Vrátíme-li se ze subsystému do hlavního schématu, přibyde nyní do subsystému vstup, na který přivedeme signál z bloku *Discrete pulse generator*, kde vyplníme periodu 0,005 s. Střídu a amplitudu můžeme nastavit na hodnotu 1.

Nyní již v nastavení bloku *eQEP* vybereme příslušný kanál, ke kterému máme na vývojovém kitu připojený signál z enkodéru. V našem případě zvolíme *eQEP1*. Dále nastavíme *Quadrature-count*, protože používáme kvadratický enkodér. Jako kladný směr otáčení zvolíme směr hodinových ručiček, tedy *Positive Rotation: Clokwise*. Dále v záložce *General* ještě nastavíme *Sample time*, jelikož je blok spouštěn externě, musíme zde zvolit -1.

Aktuálně máme dvě možnosti výpočtu rychlosti. Jako první možnost enkodér počítá počet impulzů během přesně stanovené doby. Tato varianta je vhodná pro větší otáčky. Pro nízké otáčky se používá druhá možnost, kdy enkodér měří dobu mezi dvěma po sobě jdoucími pulzy. Jako hranice mezi vyššími a nižšími otáčkami se nejčastěji udává 1000 otáček za minutu. V kapitole 5.4.1 si vysvětlíme první možnost výpočtu pro vysoké rychlosti.

5.4.1 Vysoké otáčky

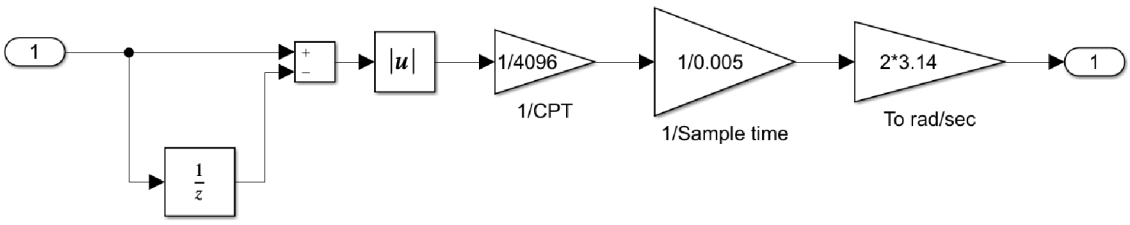
V záložce *Position Counter* zaškrtneme políčko *Output position counter* abychom dostali signál počtu pulzů uložený v registru *QPOSCNT*. Jako *Maximum position counter value* je u 32-bitového procesoru $2^{32} = 4294967295$. Tuto hodnotu omezíme na $2^{25} = 33554432$. Jelikož motor se může roztočit na oba směry, zaškrtneme políčko *Enable software initilization* a nastavíme počáteční hodnotu na polovinu z maximálního rozsahu, tedy 16777216. Při kladném směru otáčení bude čítač počítat od této hodnoty do maximální, v případě záporných otáček od této hodnoty do 0. Po dosažení 0 je hodnota čítače resetována na maximální hodnotu, naopak při dosažení maximální hodnoty je resetována na 0.

Nyní jsme hotovi s nastavením bloku *eQEP* a zbývá vytvořit blokové schéma pro výpočet rychlosti. Toto blokové schéma nalezneme na obrázku 5.7 Při tvorbě schématu budeme vycházet z rovnice 5.15 dle [18]:

$$v(k) = \frac{\Delta X}{T} \tag{5.15}$$

kde $v(k)$ je rychlost v okamžitém čase k, ΔX je přírůstek polohy v jednotkovém čase a T je pevný časový interval.

Blok *Unit Delay* funguje tak, že vzorek, který vstupuje do tohoto bloku pozdrží a v sumátoru se odečte od vzorku následujícího. Dostáváme tedy rozdíl polohy ΔX. Dále zařadíme blok *Abs*, který zajistí kladnou hodnotu i v případě záporných otáček. Nyní signál podělíme *Sample time* a počtem pulzů za jednu otáčku. Virtuální enkodér standu má 1024 pulsů za otáčku. Toto číslo ještě vynásobíme 4, protože máme zvoleno *Quadrature-count*, počet pulsů bude tedy 4096. Nyní dostáváme počet otáček za 1 sekundu na hřídeli motoru. Následně ještě vynásobíme 2π a získáme otáčky v radiánech za sekundu.



Obr. 5.7: Výpočet rychlosti vyšších otáček

5.4.2 Nízké otáčky

Pro výpočet nízkých otáček v záložce *Speed Calculation* zatrhneme políčko *Speed Calculation* a *Output capture period timer* abychom získali data z registru *QCPRD*. Další nastavení můžeme ponechat implicitní, tzn. obě děličky na hodnotě 128 a *Capture timer position* na *On position counter read*. Při tomto způsobu výpočtu rychlosti vycházíme z rovnice 5.18 dle [18]:

$$v(k) = \frac{X}{\Delta T} \tag{5.16}$$

kde X je pevně definovaná jednotková vzdálenost a ΔT je čas uplynulý za jednu jednotkovou vzdálenost X.

Rovnici 5.18 si můžeme dovolit přepsat do tvaru:

$$v(k) = \frac{1}{\Delta T} \tag{5.17}$$

Protože vzdálenost uvažujeme jednotkovou, jako jednu událost mezi dvěma pulzy.

Pro zjištění délky periody ΔT platí:

$$\Delta T = QCPRD \cdot QCTMRper \cdot UPEVNTperRev \tag{5.18}$$

kde QCTMRper je perioda časovače modulu eQEP a $UPEVNTperRev$ je počet pulzů za jednu otáčku.

Počet pulzů za jednu otáčku zjistíme tak, že vydělíme počet pulzů za jednu otáčku hodnotou, kterou jsme nastavili v děličce *Unit position event prescaler*. V našem případě pak:

$$UPEVNTperRev = \frac{4096}{128} = 32 \tag{5.19}$$

Jako další potřebujeme zjistit hodnotu QCTMRper. Registr QCTMR funguje tak, že s každým příchozím pulsem z enkodéru je resetován. Potom platí:

$$QCTMRper = SYSCLKOUT \cdot QCTMRprescaler \tag{5.20}$$

kde SYSCLKOUT je perioda frekvence mikroprocesoru a QCTMRprescaler je nastavená dělička *eQEP capture timer prescaler*.

Hodnotu SYSCLKOUT spočítáme jako:

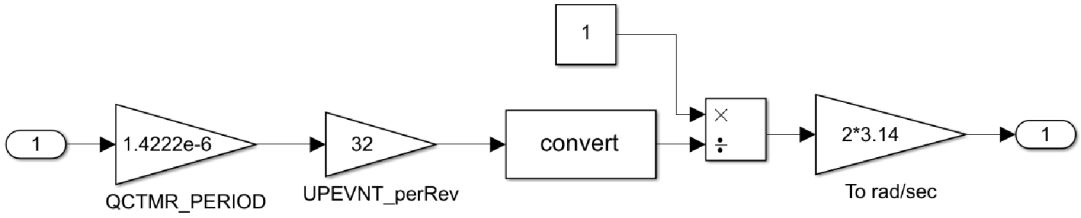
$$SYSCLKOUT = \frac{1}{SYSCLKfreq} = \frac{1}{90 \cdot 10^6} = 1.111 \cdot 10^{-8}\ s \tag{5.21}$$

kde SYSCLKfreq je frekvence mikroprocesoru.

Hodnota QCTMRper bude potom:

$$QCTMRper = 1.111 \cdot 10^{-8} \cdot 128 = 1,422 \cdot 10^{-6}\ s \tag{5.22}$$

Máme již vypočítané hodnoty pro blokové schéma výpočtu nízké rychlosti, které nalezneme na obrázku 5.8. Opět musíme ještě do schématu zavést násobení 2π pro získání radiánů za sekundu.



Obr. 5.8: Výpočet rychlosti nižších otáček

5.5 Regulátor otáček

Regulátor otáček budeme navrhovat metodou symetrického optima (SO). Nejdříve musíme zjistit všechny důležité parametry. Některé parametry převezmeme z tabulky 5.1.

Pro stejnosměrný motor je moment setrvačnosti $J = 0,019\ kg \cdot m^2$ dle [19] a pro asynchronní motor je $J = 0,0004\ kg \cdot m^2$ dle [20]. Jelikož jsou oba motory umístěny na společné hřídeli, dohromady mají moment setrvačnosti $J = 0,0194\ kg \cdot m^2$.

Hodnotu $C\phi$ vypočítáme ze jmenovitého napětí, proudu a jmenovitých otáček motoru. Ty jsou podle štítku $U_n = 24\ V$, $I_n = 20\ A$ a $n = 2800\ ot./min.$. Výpočet provedeme jako:

$$C\phi = \frac{U_n - R \cdot I_n}{2 \cdot \pi \cdot n/60} = \frac{24 - 0.11 \cdot 20}{2 \cdot \pi \cdot 2800/60} = 0,0743\ V/s \tag{5.23}$$

Přepočet otáček jsme opět provedli na jednotku rad/s, čili konstanta K_{td} bude rovna 1. Čidlo otáček, díky externímu spouštění nastaveného v kapitole 5.4, bude posílat vzorky každých 0,005 s. Toto číslo použijeme jako časovou konstantu čidla otáček.

Shrnutí parametru otáčkové smyčky nalezneme v tabulce 5.2.

L	R	J	$C\phi$	f_{pwm}	K_m	K_i	K_{td}	τ_f
[μH]	[Ω]	[$kg \cdot m^2$]	[V/s]	[Hz]	[-]	[-]	[-]	[s]
250	0.11	0,0194	0,0743	20000	24/1125	1	1	0,005

Tab. 5.2: Parametry otáčkové smyčky

Pro přenos F_S otáčkové smyčky platí vztah:

$$F_S = \frac{K_S}{(1 + p \cdot 2\tau_m)(1 + p \cdot \tau_f) \cdot p} \tag{5.24}$$

kde τ_f je časová konstanta čidla otáček, τ_m je časová konstanta měniče a K_S je konstanta soustavy dle vztahu:

$$K_S = \frac{1}{K_i} \cdot \frac{C\phi}{J} \cdot K_{td} = \frac{1}{1} \cdot \frac{0,0743}{0,0194} \cdot 1 = 3,83 \tag{5.25}$$

Obě časové konstanty ve vzorci 5.24 sečteme a dostaneme součtovou konstantu τ_σ:

$$\tau_\sigma = 2 \cdot \tau_m + \tau_f = 2 \cdot 50 \cdot 10^{-6} + 0,005 = 0,0051\ s \tag{5.26}$$

Tímto zjednodušíme přenos soustavy F_S na tvar:

$$F_S = \frac{K_S}{(1 + p \cdot \tau_\sigma) \cdot p} \tag{5.27}$$

Pro přenos otevřené smyčky $F_{O\omega}$ dle metody SO platí:

$$F_{O\omega} = \frac{1 + 4 \cdot p \cdot \tau_\sigma}{8 \cdot \tau_\sigma^2 \cdot p^2 \cdot (1 + p \cdot \tau_\sigma)} \tag{5.28}$$

Pro výpočet regulátoru opět platí stejný vztah 5.13, po dosazení pro regulátor $F_{r\omega}$ platí:

$$F_{r\omega} = \frac{1 + 4 \cdot p \cdot \tau_\sigma}{8 \cdot \tau_\sigma^2 \cdot p^2 \cdot (1 + p \cdot \tau_\sigma)} \cdot \frac{(1 + p \cdot \tau_\sigma) \cdot p}{K_S} = \frac{1 + 4 \cdot p \cdot \tau_\sigma}{8 \cdot \tau_\sigma^2 \cdot p \cdot K_S} \tag{5.29}$$

A po číselném dosazení:

$$F_{r\omega} = \frac{1 + 4 \cdot p \cdot 0,0051}{8 \cdot 0,0051^2 \cdot p \cdot 3,83} = \frac{1 + p \cdot 0,0204}{p \cdot 796,95 \cdot 10^{-6}} \tag{5.30}$$

Rozdělíme-li nyní tento zlomek na proporciální a integrační část, vyjde nám $P = 25,6$

A integrační část $I = 1254,8$

Stejně jako v předchozím případě budeme regulátor počítat v subsystému přerušení, toto číslo opět vydělíme frekvencí 20 kHz, potom $I = 0,063$.

Nyní do subsystému přerušení před proudovou smyčku vložíme opět blok *PID* a vyplníme proporciální a integrační složku. Jako meze saturace nyní vložíme maximálnímu hodnotu proudu motoru, tedy $I_{max} = 20\ A$. Nesmíme zapomenout opět zvolit metodu *Clamping*.

Rozdíl žádaných a měřených otáček budeme již vypočítávat mimo přerušení, viz obrázek B.1. Rozdíl otáček musíme ještě před vstup do subsystému přivést do bloku *Rate transition*, protože v subsystému je použito asynchronní vzorkování. Pokud nám bude Simulink nadále generovat chybu, v bloku *Rate transition* odškrtneme políčko *Ensure deterministic data transfer*. Žádané otáčky můžeme buď nastavit konstantě a nebo přijímat po sériové lince. Tomuto tématu se budeme věnovat v následující kapitole 5.6.

5.6 Komunikace po sériové lince

Po sériové lince budeme posílat informace o otáčkách a proudu do počítače a zároveň z počítače nastavovat žádanou hodnotu otáček. Sériovou komunikaci uskutečníme pomocí modulu SCI A. Nejdříve je potřeba v *Hardware configuration* nastavit požadovanou rychlost přenosu *Baud rate*. Dle [21] je pro *LaunchPad F28069M* potřeba nastavit tuto rychlost na hodnotu $5,625 \cdot 10^6$.

5.6.1 Odesílání dat

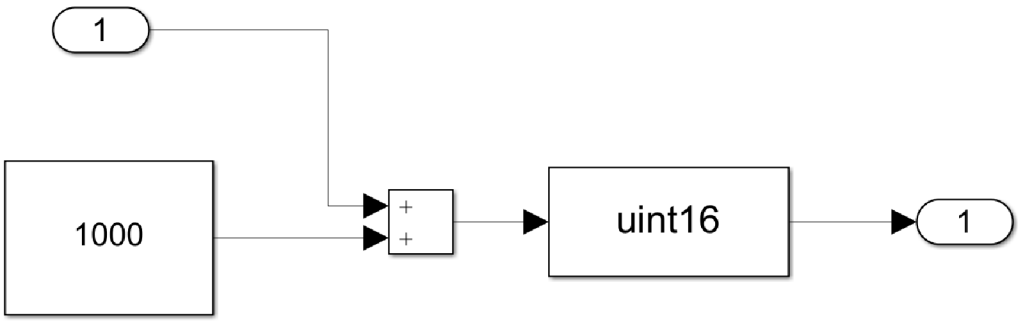
Odesílat budeme zároveň měřený proud a otáčky. Oba signály budeme posílat ve formátu kladných celých čísel *uint16*, proto musíme tuto konverzi ještě zařadit za blok *eQEP* a také blok *Rate transition*, protože budeme posílat data v subsystému přerušení, viz. obrázek B.1. Signály proudu i otáček sloučíme do jednoho pomocí bloku *Mux* a zavedeme do subsystému *Serial send*, který umístíme do subsystému přerušení, viz obrázek B.2.

Subsystém *Serial send* nalezneme v examplu *Permanent Magnet Synchronous Motor Field-Oriented Control* od společnosti MathWorks, který spustíme v Matlabu příkazem *c28069pmsmfocdual_ert*. V našem případě se jedná o verzi Matlab R2019b. Čerpáme ze zdroje [21]. Zmíněný subsystém nalezneme v subsystému *FOC Alogrithm Motor 1*. Subsystém *Serial send* funguje tak, že čítač počítá od 0 do 599 a za tu dobu se plní buffer a po dosažení maximální hodnoty se tento buffer posílá po sériové lince a data jsou nadále zpracovávána a zobrazována v hostitelském programu. Najednou je vždy posíláno 600 vzorků. Samotné posílání je realizováno pomocí bloku *SCI Transmit*. V tomto bloku je zvolen SCI module A. V následujících dvou políčkách *Aditional package header* a *Aditional package terminator* vymažeme písmena a necháme pouze uvozovky. Tyto dvě písmena znamenají informaci o startu (S) a konci (E) posílání dat, tuto informaci nepotřebujeme.

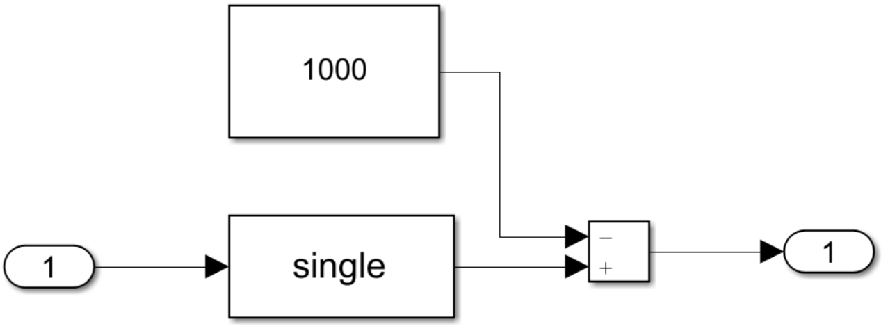
Jelikož data po sériové lince posíláme a přijímáme ve formátu nezáporných čísel *uint16*, nemohli bychom tímto způsobem posílat ani přijímat záporná čísla pro měření a nastavení záporných otáček. Z tohoto důvodu musíme odesílaná data posunout o dostatečně vysokou hodnotu. Otáčky se budou pohybovat nanejvýš v mezích od -314 do 314 rad/sec, posuneme tedy tento rozsah o hodnotu 1000. Stejnou hodnotu musíme při zobrazování měřených otáček odečíst. Pro tyto účely si vytvoříme 2 subsystémy, *To SCI* a *From SCI* viz obrázky 5.9 a 5.10.

5.6.2 Přijímání dat

Po sériové lince posíláme žádané otáčky do regulátoru otáček, viz obrázek B.1. Přijímat data z hostitelského programu budeme pomocí bloku *SCI Receive*. Tento blok



Obr. 5.9: Subsystém To SCI



Obr. 5.10: Subsystém From SCI

nastavíme následovně:

Jako modul SCI vybereme A. V následujících dvou políčkách *Aditional package header* a *Aditional package terminator* opět vymažeme písmena a necháme pouze uvozovky. Jako data type vybereme *uint16*, jak jsme zmínili výše. Délku dat *Data length* necháme 1, pokud bychom zadali vyšší hodnotu, jednalo by se o vektor dat. My budeme posílat pouze jednotlivé hodnoty. Počáteční hodnotu *Initial value* můžeme zvolit libovolně. V našem případě jsme vyplnili hodnotu 50, tzn. že motor se při nahrání programu do mikrokontroléru díky regulátoru otáček roztočí na otáčky 50 rad/sec. Aby byly vzorky přijímány s dostatečnou hustotou, zvolíme *Sample time* 5 x kratší než je vzorkovací čas měření otáček, tedy $0,001$ s. Tím jsme hotovi s nastavením bloku. Za blok ještě umístíme převod na datový typ *Single*, protože měřené otáčky vstupující do sumátoru regulátoru využívají tento datový typ viz B.3.

Z důvodu uvedeného v kapitole 5.6 zařadíme za blok ještě subsystém *From SCI*. V tomto případě získáme nulové otáčky, pokud do *Initial value* zapíšeme hodnotu 1000.

5.6.3 Hostitelský program

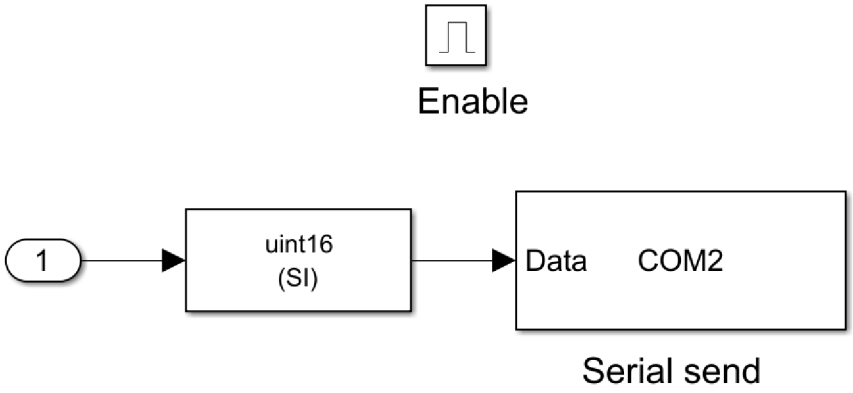
Pro hostitelský program si založíme nový model v Simulinku. Do něj jako první blok umístíme *Serial configuration*, nyní již z knihovny *Instrument Control Toolbox*. Při nastavování tohoto bloku již musíme mít připojen LaunchPad k hostitelskému zařízení pomocí sériové linky. V případě generování chyby, restartujeme Simulink. V první kolonce *Communication port* zvolíme port COM, ke kterému je připojen LaunchPad, v našem případě je to port *COM2*. Jako rychlost *Baud rate* zvolíme opět hodnotu $5,625 \cdot 10^6$. Ostatní nastavení můžeme ponechat.

Nyní do modelu přidáme blok *Serial receive* ze stejné knihovny, kde vybereme opět port *COM2*. Jako *Data size* napíšeme [2 600], protože posíláme dva signály a vždy 600 vzorků dat najednou. Jako datový typ vybereme opět *uint16*. Pokud ponecháme vzorkovací čas *Sample time* na hodnotě 0,01, hostitelský program přijme každých 0,01 s 600 vzorků dat. Při frekvenci přerušení 20 kHz je těchto 600 vzorků odesláno během doby 600/20000 = 0,03 s, což je 3x déle než *Sample time* 0,01 s. Proto je potřeba při tomto nastavení čas změřený v Simulinku násobit 3 pro zjištění skutečného času. Pokud nastavíme *Sample time* na 0,03 s, bude při frekvenci přerušení 20 kHz čas změřený v Simulinku totožný se skutečným časem.

Dle [21] je potřeba za tento blok zařadit blok *Transpose*, který vektor dat transponuje a dále blok *Unbuffer*, který vzniklou matici rozdělí na sekvenci skalárních výstupů. Dále zařadíme blok *Demux* a nyní již dostáváme samostatné signály. Signál o proudu odesíláme nepřepočítaný, proto musíme do hostitelského programu ještě zařadit subsystém *Speed Scale*, viz obrázek 5.6 a samozřejmě subsystém *From SCI* viz obrázek B.4. Nakonec zařadíme blok *Scope* se dvěma vstupy, kde již můžeme zobrazovat naměřená data.

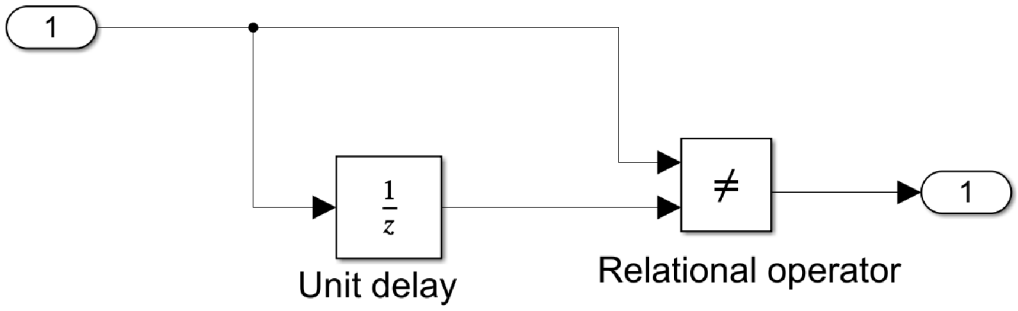
Pro odesílání dat do LaunchPad vložíme do modelu blok *Serial send* opět z knihovny *Instrument Control Toolbox*. Před něj ještě zařadíme převod na *uint16* a z těchto bloků vytvoříme subsystém. Tento subsystém budeme spouštět tehdy, pokud bude detekována změna hodnoty, data se nebudou neustále posílat. Do vzniklého subsystému musíme ještě přidat blok *Enable*, který na vstupu vytvoří spouštěcí port viz obrázek 5.11. Do vstupu *In1* pro data přivedeme subsystém *To SCI* a ještě před něj blok *Constant*, kde nastavíme *Sample time* na 0,001 s jako v případě *SCI Receive* v kapitole 5.6.2. Do tohoto bloku budeme zapisovat žádanou rychlost v radiánech za sekundu.

Do programu ještě zahrneme subsystém pro detekování změny. Ten vytvoříme ze dvou bloků *Unit Delay* a *Relational operator*, kde zvolíme podmínku nerovná se. Blok *Unit delay* bude tedy zpožďovat vzorek a porovnávat s následujícím v operátoru nerovná se. Pokud se obě hodnoty nebudou rovnat, blok *Relational operator* generuje logickou 1 a subsystém *Serial send* bude spuštěn a data budou poslána viz obrázek



Obr. 5.11: Subsystém Serial send

5.12.



Obr. 5.12: Detekce změny žádané hodnoty

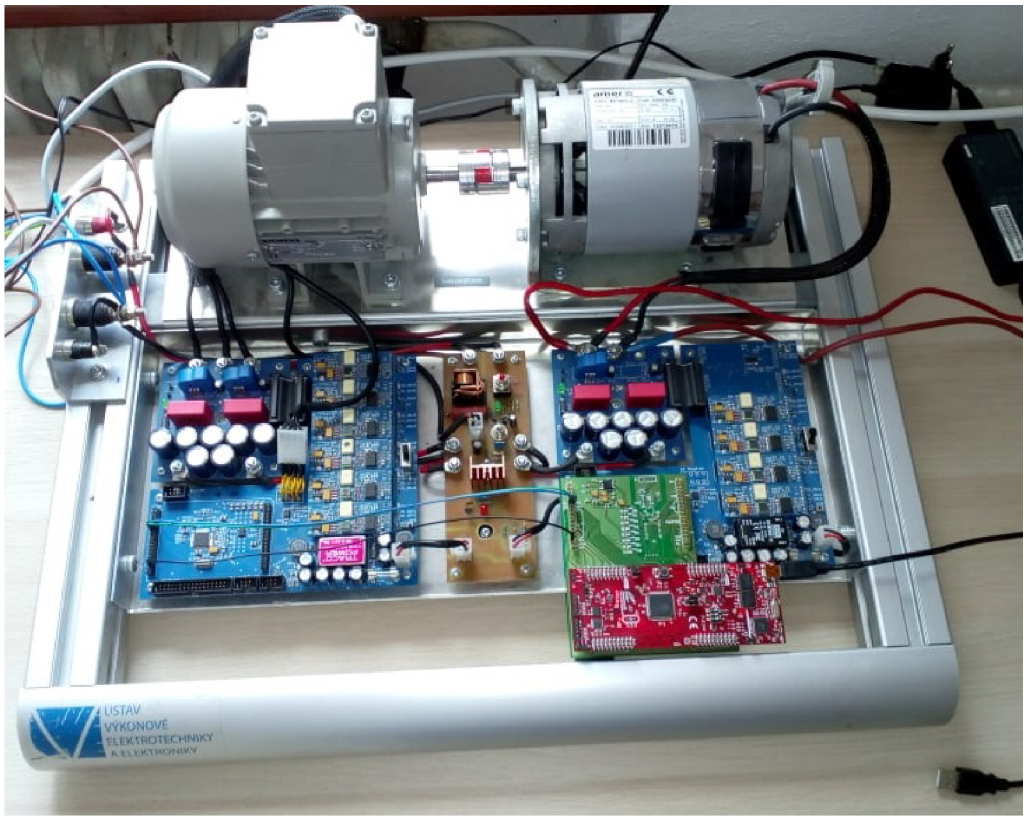
V samotném *Scope* je ještě pro porovnání přiveden signál o skutečných otáčkách a žádaných otáčkách, může se zde tedy pozorovat například regulační odchylka v reálném čase. Celkové schéma programu si můžeme prohlédnout v přílohách na obrázku B.4.

Hostitelský program spouštíme jako simulaci v Simulinku tlačítkem *Run*. Simulační čas nastavíme na nekonečno *inf*.

5.7 Hardwarové připojení LaunchPad k laboratornímu standu

Vývojový kit LaunchPad připojíme na pravý stand pro DC motor přes navrženou redukci. Deska standu pro DC motor nemá na svých pinech přiveden signál z en-

kodéru z resolvéru. Tento signál musíme přivést z levé části standu pomocí vodičů. Zapojení ilustruje obrázek 5.13. Zelená DPS na obrázku představuje zmíněnou redukci. LaunchPad připojíme pomocí USB kabelu do hostitelského zařízení, v našem případě do notebooku.



Obr. 5.13: Připojení LaunchPad ke standu pro DC motor

Program do LaunchPadu nahrajeme a spustíme pomocí tlačítka *Build, Deploy & Start* v záložce *Hardware* v Simulinku.

5.8 Výsledky a měření

Nejdříve jsme ověřili funkčnost bipolárního a unipolárního řízení. Dále byly testovány regulátory. S použitím vypočítaných hodnot regulátoru proudu byl proud lehce zvlněn, asi o 1 A, viz obrázek G.1. Muselo být zvětšeno proporciální zesílení z původní hodnoty 117 na hodnotu 250. Nyní bylo zvlnění sníženo na 0,5 A, viz obrázek G.2. Zvlnění se nepodařilo úplně eliminovat, při dalším zvyšování proporciální složky se zvlnění proudu opět zvětšovalo.

Při testování regulátoru otáček se ukázalo, že způsobuje kmitání motoru. Bylo třeba vyladit nové hodnoty regulátoru. Experimentálně jsme zjistili novou hodnotu

proporciální složky PI regulátoru otáček, která se změnila z původní hodnoty 25,6 na hodnotu 0,2. Pro rychlejší regulaci bylo integrační zesílení zvětšeno z hodnoty 0,062 na hodnotu 0,2. Při těchto hodnotách fungovala regulace správným způsobem. Snímání otáček je realizováno pouze pro vysoké rychlosti. Se snímáním pro nízké rychlosti se bohužel nepodařilo sladit regulátor otáček, navíc se objevil problém s nulovými otáčkami. I se snímáním pro vysoké otáčky ale fungoval regulátor otáček spolehlivě.

Správnost měřeného proudu v Simulinku byla ověřena pomocí naměřeného průběhu proudu z osciloskopu. Průběh proudu ze Simulinku, na obrázku G.2, téměř odpovídá průběhu proudu měřeného osciloskopem pomocí proudové sondy, obrázek G.3. Průběh ze Simulinku je lehce deformován náhlými špičkami, které mohou být způsobeny rušivým signálem. V Simulinku navíc nezachytíme vysokofrekvenční zvlnění proudu na indukčnosti motoru způsobené kmitočtem PWM.

Při ověření funkčnosti regulátorů byla nastavovány otáčky postupně od 0 do 1500 ot/min a zpět k 0 do záporných otáček až k -1500 ot/min a následné zastavení. Toto ověřování společně s naměřeným proudem zachycuje obrázek G.4. Žluté, skutečné otáčky byly vyregulovány až do žádané hodnoty, kterou reprezentuje modrý průběh.

Z detailu regulace na obrázku G.5 je patrné dopravní zpoždění regulace. Toto zpoždění může být způsobeno opožděnou komunikací po sériové lince a nebo pomalým vzorkováním snímání otáček, to je nastaveno na 5 ms. Dopravní zpoždění je ale nepatrné a požadavek na dynamiku DC motoru není kritický, takže zpoždění nemá negativní účinky na dynamiku regulace.

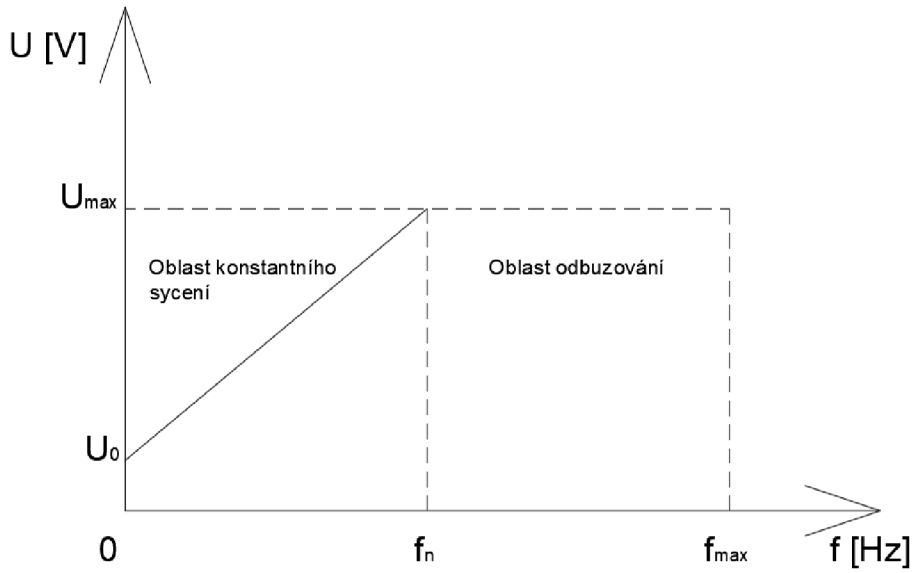
Pro omezení proudových nárazů při rozběhu motoru, byl DC motor spouštěn po rampě až do 1500 ot./min. viz obrázek G.6.

Výsledkem tohoto programu je tedy nastavování žádaných otáček v hostitelském zařízení po sériové lince pomocí hostitelského programu, regulace žádaných otáček a následné zobrazení skutečného proudu a otáček pomocí hostitelského programu.

6 Skalární řízení Asynchronního motoru

6.1 Skalární řízení v otevřené smyčce

Skalární řízení asynchronního motoru spočívá v nastavování hodnoty amplitudy U a frekvence f napájecího napětí motoru. Motor chceme provozovat v oblasti konstantního sycení, kdy poměr U/f, případně $U/\omega = \text{konst}$. Pokud napětí již nelze zvětšovat, čili je dosaženo maximálního napájecího napětí, můžeme dále motor odbuzovat, viz obrázek 6.1. Řízení realizujeme v otevřené smyčce, tedy bez zpětnovazebních regulátorů.



Obr. 6.1: Řízení U/f

Nejprve je potřeba zjistit jmenovité napětí a frekvenci našeho motoru, to zjistíme z jeho štítku, viz obrázek 6.2. Jmenovitá frekvence statorového napětí je 50 Hz a jelikož jmenovité otáčky jsou 1350 ot./min. bude motor čtyřpólový, tedy $2p = 4$. Motor máme zapojený do hvězdy (Y), potom jmenovitá efektivní sdružená hodnota napětí mezi dvěma fázema je dle štítku 23,4 V. Maximální hodnota bude potom $23,4 \cdot \sqrt{2} = 33\ V$. Hodnota napětí v meziobvodu je 24 V. Toto je zároveň amplituda sdruženého napětí. Můžeme tedy využít celý rozsah měniče 0-24 V.

Jako napětí U_{max} dle obrázku 6.1 zvolíme amplitudu fázového napětí, tedy $24/\sqrt{3}$ =13,9 V. Při skalárním řízení se realizuje mírný odskok napětí od 0, viz obrázek 6.1, aby se motor roztočil již z nulových otáček. Jako napětí při nulových otáčkách U_0 zvolíme 2 V.

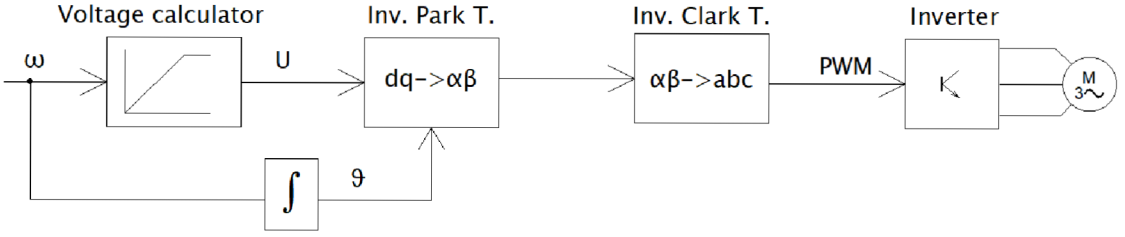
Nakonec přepočítáme jmenovitou frekvenci na úhlovou rychlost v radiánech:

$$\omega = 2 \cdot \pi \cdot f = 2 \cdot \pi \cdot 50 = 314 \; rad/sec \tag{6.1}$$

Obr. 6.2: Štítek asynchronního motoru umístěného na laboratorním standu

6.1.1 Skalární řízení

Nejprve realizujeme subsystém pro samotné skalární řízení. Při tvorbě blokového schématu v Simulinku, budeme vycházet z blokového schématu skalárního řízení, dle obrázku 6.3.



Obr. 6.3: Blokové schéma skalárního řízení v otevřené smyčce

Skalární řízení bude s orientací na statorový tok. Při tvorbě řízení budeme vycházet z rovnic pro napětí u_{sd} a u_{sq} v dq osách dle [22]:

$$u_{sd} = R_s \cdot i_{sd} \tag{6.2}$$

$$u_{sq} = R_s \cdot i_{sq} + \omega_s \cdot \psi_s \tag{6.3}$$

kde R_s je odpor fáze statorového vinutí, i_{sd} je statorový proud v ose d, i_{sq} je statorový proud v ose q, ω_s jsou synchronní otáčky a ψ_s je spřažený magnetický tok statoru.

V řízení s otevřenou smyčkou zanedbáme ohmické odpory statorového vinutí, potom se rovnice zjednoduší na tvary:

$$u_{sd} = 0 \tag{6.4}$$

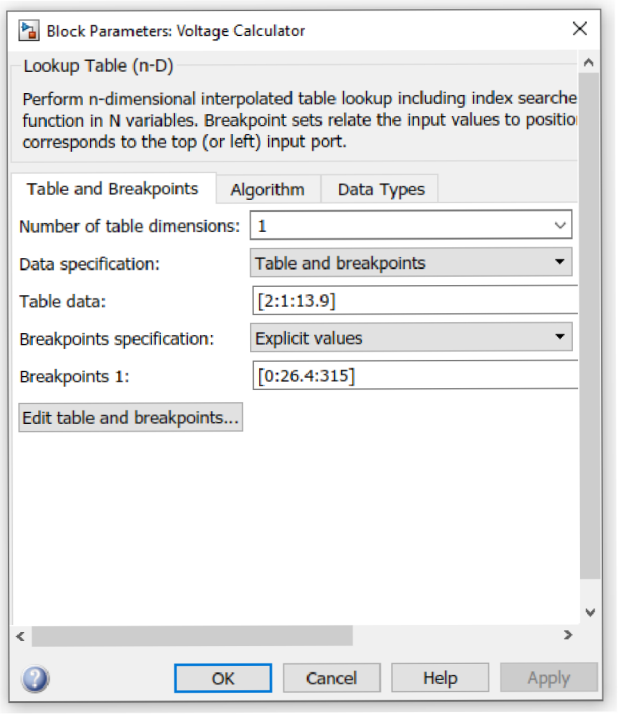
$$u_{sq} = \omega_s \cdot \psi_s \tag{6.5}$$

6.1.2 Voltage Calculator

Nejprve do schématu přidáme blok *1-D Lookup Table*, který nám bude představovat *Voltage Calculator* viz. obrázek 6.3. V tomto bloku si definujeme funkci, která každé hodnotě ω přiřadí hodnotu napětí U. Tuto funkci definujeme pouze v lineární části, tedy v oblasti konstantního buzení motoru. Jako první nastavíme jednu dimenzi, nemáme víc vstupních portů. Data budeme specifikovat pomocí *Table and Breakpoints*. *Table* udává funkční obor, tedy hodnoty napětí a *Breakpoints* udává obor hodnot, tedy úhlovou rychlost. Jako první hodnoty definujeme do hranatých závorek obou polí počátek jako 2 V a 0 rad/s. Všechny čísla musíme oddělit dvojtečkou. Jako další hodnoty zapíšeme 1 V a do spodního pole hodnotu otáček, které odpovídá právě 1 V. Jelikož celkový rozsah napětí je 2-13,9 V a rozsah otáček 0 - 314 rad/s výpočet provedeme následovně: $(314 - 0)/(13,9 - 2) = 26,4$. Jako druhé hodnoty tedy zapíšeme do pole *Table* 1 a do pole *Breakpoint* číslo $26,4$. Jako poslední třetí čísla budou následovat maximální rozsahy, tedy hodnota 13,9 a 315. Hodnotu 315 jsme museli zaokrouhlit nahoru, Simulink generoval chybu neplatného rozsahu. Zkontrolovat nastavení můžeme podle obrázku 6.4. Další záložky můžeme nechat explicitně nastavené.

Výstupem z Voltage Calculatoru je napětí u_{sq}. Vstup, který nám představuje synchronní otáčky, násobíme s poměrem U/ω, který představuje konstantní tok ψ_s.

Opět budeme požadovat nastavování kladných i záporných otáček. Proto musíme ještě před blok *1-D Lookup Table* vložit blok *ABS*, který zajistí absolutní hodnotu signálu, protože *1-D Lookup Table* je definován pouze pro kladná čísla. Při požadavku záporných otáček je třeba tuto zápornou hodnotu přivést pouze do vstupu integrátoru a vzniklý záporný úhel při transformaci zajistí opačný sled fází. Tím bude motor poháněn do záporných otáček.



Obr. 6.4: Nastavení Look-Up table - Voltage Calculator

6.1.3 Inverzní Parkova transformace

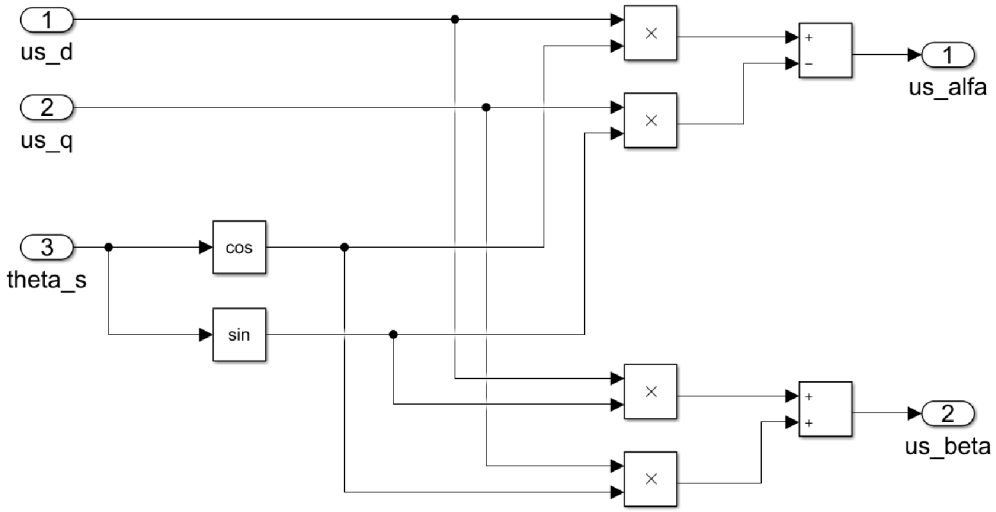
Inverzní Parkovu transformaci vytvoříme podle rovnic:

$$u_{s\alpha} = cos(\vartheta) \cdot u_{sd} - sin(\vartheta) \cdot u_{sq} \tag{6.6}$$

$$u_{s\beta} = sin(\vartheta) \cdot u_{sd} + cos(\vartheta) \cdot u_{sq} \tag{6.7}$$

kde $u_{s\alpha}$ a $u_{s\beta}$ jsou statorové hodnoty napětí v $\alpha\beta$ souřadnicích a ϑ je úhel mezi $\alpha\beta$ a dq souřadnicemi. $\alpha\beta$ souřadnice jsou spjaty se statorem a dq souřadnice s točivým magnetickým polem statoru.

Matematický model Parkovy transformace podle výše uvedených rovnic nalezneme na obrázku 6.5. Jako napětí v ose q u_{sq} přivedeme nově vytvořené napětí z bloku *Lookup Table* a jako napětí v ose d u_{sd} přivedeme 0 z bloku *Constant*. Ještě musíme do subsystému Parkovy transformace přivést úhel ϑ. Ten získáme tak, pokud zintegrujeme signál o synchronních otáčkách ω_s. To provedeme pomocí bloku diskrétní integrace *Discrete-Time Integrator*.



Obr. 6.5: Inverzní Parkova transformace

6.1.4 Inverzní Clarkové transformace

Blok SVM v blokovém schématu na obrázku 6.3 znamená Space Vector Modulation, čili Inverzní Clarkové transformaci. Tato transformace již vytvoří 3-fázový systém pro PWM. Transformaci můžeme opět vytvořit pomocí rovnic a nebo v Simulinku v knihovně *Simscape/Electrical/Control/Mathematical Transforms* nalezneme přímo blok *Inverse Clarke transform*. Tato knihovna také obsahuje blok pro

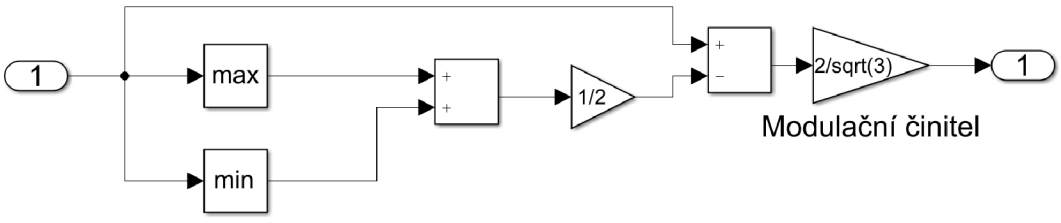
Parkovu transformaci. Rozhodneme se pro druhou možnost a do modelu nyní vložíme blok z knihovny. Kromě napětí $u_{s\alpha}$ a $u_{s\beta}$ potřebujeme do této transformace přivést 0, opět pomocí bloku *Constant*.

Nyní nám již vzniknou 3 sinusoidy vzájemné posunuté o 120 stupňů s amplitudou příslušného napětí k otáčkám. Amplitudy sinusoid se pohybují v rozsahu od -13,9 do 13,9 V.

6.1.5 Tvorba PWM

Jako v případě řízení DC motoru, požadujeme stejnou hodnotu frekvence 20 kHz, tu budeme nyní ale realizovat pomocí děličky kmitočtu, ukázalo se to být lepším řešením. Nastavíme proto frekvenci 40 kHz a děličku 2. Pro 40 kHz bude počítadlo vysokofrekvenčního signálu dosahovat hodnot od 0 do 1125. Vzniklé signály se pohybují kolem 0 od 13,9 do -13,9. Pro modulační signál požadujeme, aby střed byl v hodnotě 1125/2 = 562,5. Dle [23] je třeba u 3-fázového střídače zvětšit nízkofrekvenční modulační signál vzhledem k vysokofrekvenčnímu o hodnotu $2/\sqrt{(3)}$. Po tomto zvětšení by signál při maximálním napětí přesahoval vysokofrekvenční nosný signál a přečnívající část by byla saturací omezena. To by způsobilo deformaci statorového napětí při přiblížení k jmenovitým otáčkám. Aby k tomuto jevu nedošlo, můžeme dle [23] doplnit modulační signál signálem o trojnásobném kmitočtu.

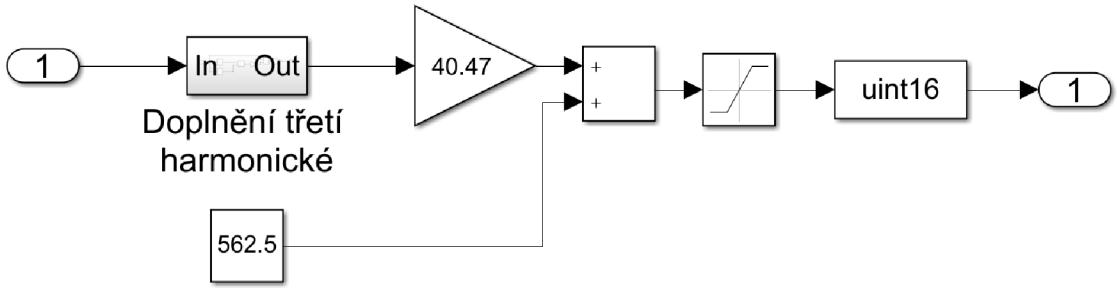
Dle [24] to můžeme realizovat pomocí trojúhelníkového signálu s amplitudou o velikosti 1/4 amplitudy sinusového signálu. Trojúhelníkový signál získáme, pokud ze vzniklého 3-fázového sinusového signálu vyfiltrujeme pouze minimální a maximální špičky pomocí bloku *MinMax*. Tyto pulzy budou mít poloviční amplitudu oproti původnímu signálu. Pokud je vzájemně sečteme, dostaneme již trojúhelníkový signál s trojnásobnou frekvencí, který bude mít poloviční amplitudu oproti původnímu signálu. Musíme jej proto ještě vydělit 2 a tento signál odečíst od 3-fázového sinusového signálu. Nakonec ještě vynásobíme modulačním činitelem $2/\sqrt{(3)}$. Tuto situaci nám zachycuje obrázek 6.6.



Obr. 6.6: Doplnění třetí harmonické

Nyní je potřeba ještě škálovat signál pro potřeby vysokofrekvenčního nosného signálu. Sinusový modulační signál se bude pohybovat symetricky kolem hodnoty 562,5 v rozmezí 0 až 1125. Vzniklý modulační signál tedy musíme vynásobit číslem 562,5/13,9 = 40,47. Nyní přičteme hodnotu 562,5. Tento signál ještě omezíme saturační ochranou podle hodnot vf nosného signálu, tedy od 0 do 1125. Škálování pro PWM nalezneme na obrázku 6.7.

Signály již přivedeme přímo do bloků *ePWM*, které nastavíme stejně, jako v případě unipolárního řízení DC motoru v kapitole 5.1.1. Jediný rozdíl bude v tom, že nastavíme *Timer Period* na 1125 a *Time base clock prescaler divider* zvolíme 2. V případě asynchronního motoru budeme využívat 3 moduly *ePWM*. Výsledné schéma skalárního řízení v otevřené smyčce nalezneme na obrázku C.3. Vytvořený PWM signál pro maximální hodnotu napětí s již oříznutými špičkami nalezneme v přílohách na obrázku H.1.



Obr. 6.7: Škálování pro PWM

6.1.6 Snímání proudu

Snímání proudu realizujeme stejným způsobem jako v případě DC motoru. Rozdíl bude pouze v tom, že nyní budeme snímat proudy dvou fází. Přidáme tedy do schématu další blok, který bude měřit vstup *ADCINA1*, viz obrázek A.6, protože signál proudu I_1 je přiveden na pin IND5. První blok měří opět vstup *ADCINB6*, tedy proud I_2. Nesmíme zapomenout nastavit v každém bloku rozdílné číslo *SOC trigger number*. Oba bloky budou synchronizovány s *ePWM1A*. Pouze jeden z nich nastavíme na spouštění přerušení pomocí *ADCINT1*.

Do tohoto přerušení umístíme oba bloky *ADC* společně se subsystémem *Serial Send* a subsystémem skalárního řízení, viz obrázek C.2. Blok *Hardware Interrupt* nastavíme stejně jako v případě DC motoru na přerušení *ADCINT1*

6.1.7 Snímání otáček

Subsystém pro snímání otáček můžeme převzít z programu DC motor. Opět budeme měření spouštět externím signálem 5 ms.

6.1.8 Sériová komunikace

Jak bylo zmíněno v kapitole 6.1.6, do přerušení přidáme subsystém *Serial Send*. Do hlavního programu přidáme *SCI Receive*, který opět nastavíme stejně jako v případě DC motoru. Jako počáteční hodnotu můžeme nastavit 5000, která po škálování zajistí 0. Budeme zde mít totiž jinou posunovací úroveň v případě subsystému *To SCI* a *From SCI*. Hodnotu otáček posíláme v rozsahu -1500 až 1500 a jako offset tedy zvolíme právě hodnotu 5000.

Pro odesílání otáček musíme před subsystém přerušení opět zařadit blok *Rate Transition*. Veškeré škálování měřeného signálu proudu a otáček provádíme v hositelském programu. Při přijímání otáček z hostitelského programu nesmíme ve škálování *Omega synchronní* zapomenout na násobení počtem pólpárů, v našem případě $pp = 2$. Schéma hlavního programu nalezneme na obrázku C.1.

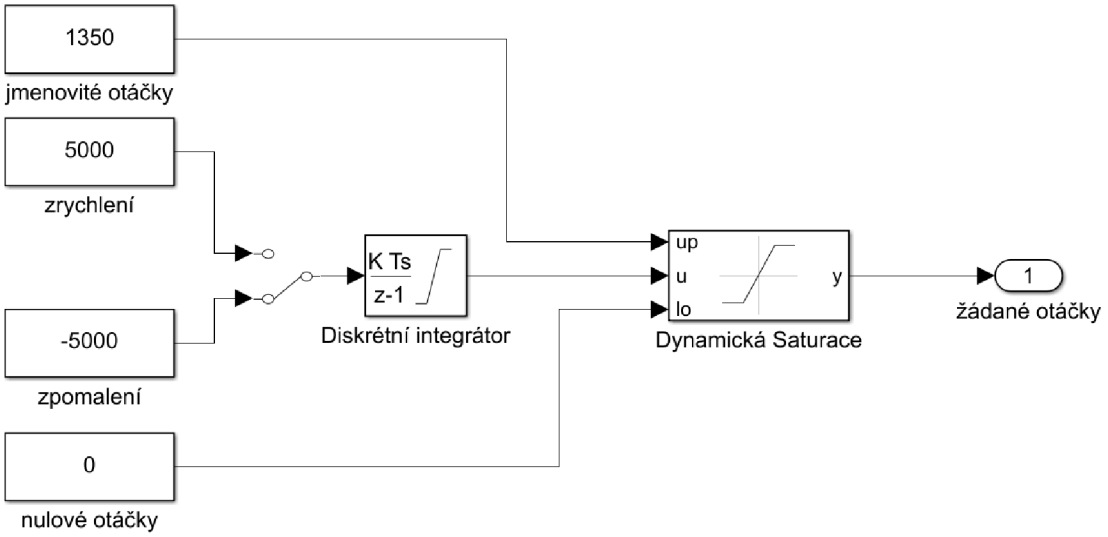
6.1.9 Hostitelský program

Při tvorbě hostitelského programu můžeme schéma převzít z realizace hostitelského programu pro DC motor. Rozdíl bude pouze v tom, že budeme přijímat 3 signály, 2 proudy a 1 signál otáček. Také nesmíme zapomenout v subsystémech *To SCI* a *From SCI* změnit hodnotu z 1000 na 5000. Schéma toho programu je v přílohách na obrázku C.4.

Pro plynulý rozběh motoru a omezení rozběhových proudů vybavíme hostitelský program rampou pro plynulý rozběh. Ta je tvořena dynamickou saturací. Omezení určuje 0 a jmenovité otáčky. Samotná rampa vzniká integrací konstanty v diskrétním integrátoru. Strmost nárůstu můžeme změnit zvýšením nebo snížením konstanty. Pro snížení rampy se přepne přepínačem na zápornou strmost. Schéma rampy je na obrázku 6.8.

6.1.10 Hardwarové připojení

Hardwarové připojení realizujeme jednoduše pomocí vytvořené redukce. Redukci nyní připojíme k levému standu, který řídí asynchronní motor. Nyní již nepotřebujeme žádné další vodiče, enkodér je v tomto případě přiveden na piny konektoru.



Obr. 6.8: Rampa plynulého rozběhu

6.1.11 Výsledky a měření

Ověřili jsme správnost kladných i záporných otáček, které jsme nastavovali v hostitelském programu. Při nastavení synchronních otáček 1500 ot/min byly skutečné otáčky 1450 ot/min. Skluzové otáčky tedy činí 50 ot/min a jsou způsobeny mírným mechanickým zatížením DC motoru a třecími ztrátami. Průběhy měření jsou v přílohách na obrázku H.2 a H.3.

Na obrázku H.4 je zobrazeno spouštění na jmenovité otáčky pomocí rampy. Rozběhový proud zde dosahuje maximálně 8 A. Při rozběhu bez plynulého náběhu otáček rozběhový proud přesáhl hodnotu 20 A, obrázek H.5.

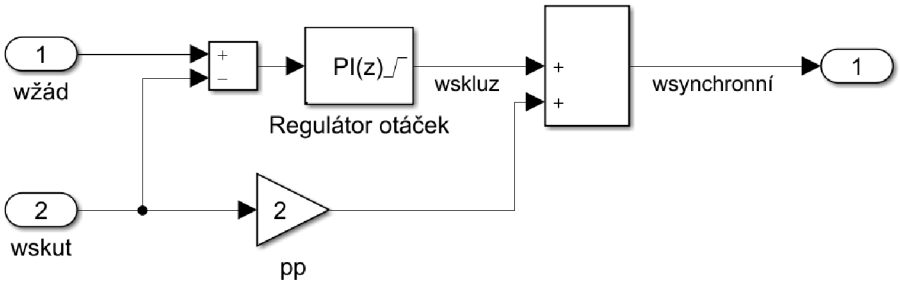
6.2 Skalární řízení v uzavřené smyčce s čidlem otáček

Skalární řízení v uzavřené smyčce spočívá v tom, že přivedeme zpětnou vazbu z motoru do řízení. Skalární řízení v uzavřené smyčce realizujeme pomocí regulátoru otáček.

6.2.1 Regulátor otáček

Regulátor otáček umístíme před subsystém *Skalární řízení* do subsystému přerušení. Na vstup regulátoru přivedeme rozdíl žádaných a skutečných otáček. Samotný regulátor je realizován jako PI. Výstupem budou skluzové otáčky. K těmto otáčkám ještě musíme připočíst skutečné otáčky vynásobené počtem pólpáru. Tím dostáváme

synchronní otáčky, které již přivedeme na vstup subsystému *Skalární řízení*. Schéma regulátoru otáček nalezneme na obrázku 6.9.



Obr. 6.9: Regulátor otáček ASM

Ještě provedeme drobnou úpravu. že přesuneme subsystém *To SCI* až do subsystému přerušení před samotné odesílání dat. Subsystém přerušení nalezneme na obrázku D.2, hlavní program potom v příloze na obrázku D.1.

6.2.2 Výsledky a měření

Při měření byl motor postupně roztočen na jmenovité otáčky a poté byla ověřena možnost nastavení záporných otáček také do jmenovité velikosti. Experimentálně byly vyladěny hodnoty PI regulátoru jako P = 1 a I = 0,05.

Při použití regulátoru otáček byly skutečné otáčky vyregulovány až na požadovanou rychlost, viz obrázek H.6. Detail regulace je potom na obrázku H.7.

7 Zatěžování Asynchronního motoru DC motorem

Dosud jsme motory, při ověřování programů a měření, nijak nezatěžovali, hřídel byla zatížena pouze protějším motorem, který byl odpojen od napájení. V této kapitole se budeme zabývat zatěžováním asynchronního motoru pomocí stejnosměrného motoru, který poslouží jako dynamometr. V hostitelském programu budeme nastavovat zátěžný proud DC motoru a sledovat průběhy otáček a proudu. Zatěžování provedeme pro skalární řízení s otevřenou smyčkou i s uzavřenou smyčkou.

7.1 Skalární řízení v uzavřené smyčce

Pro vytvoření modelu použijeme již vytvořený model pro skalární řízení v uzavřené smyčce. Do něj přidáme ještě subsystém přerušení z vytvořeného modelu řízení DC motoru. Tento model musíme ještě upravit. Odebereme odesílání po sériové lince, to musíme posílat společně s veličinami asynchronního motoru. Bloky pro tvorbu PWM jsme přenastavili na moduly *ePWM4* a *ePWM6*, piny příslušné k těmto modulům byly dosud neobsazeny. Dále musíme přenastavit ADC převodník. V bloku *ADC* vybereme volný ADCIN LaunchPadu. Vybrali jsme ADCINB0. SOC čísla 0 a 1 jsou již zabraná, musíme vybrat SOC2. Převodník budeme spouštět s EPWM4A a převodník bude generovat přerušení na kanálu *ADCINT2*.

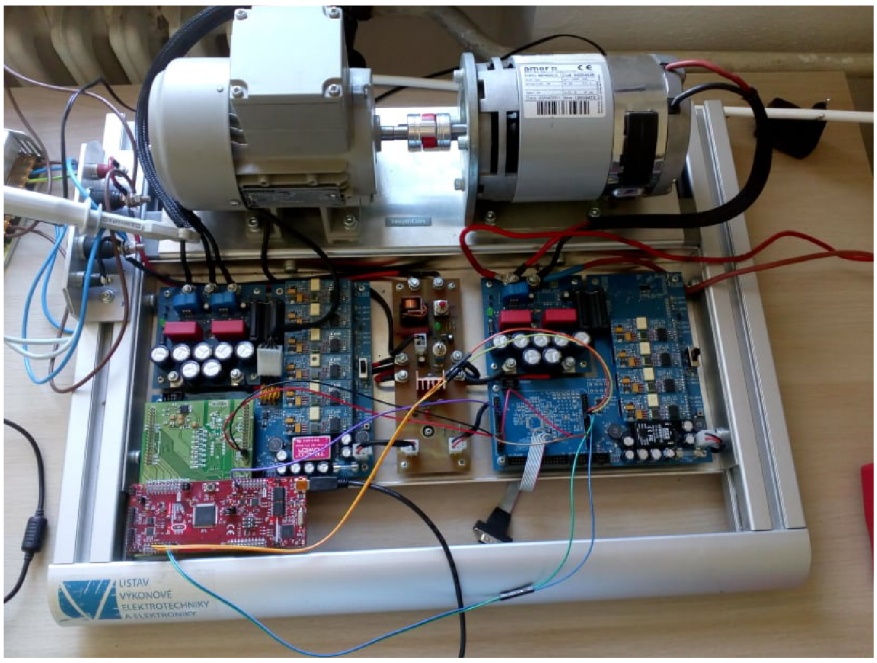
Ze subsystému dále odstraníme regulátor otáček a necháme pouze regulátor proudu, budeme nastavovat moment. Žádaný proud, respektive moment, přivedeme ze sériové linky z bloku *SCI Receive*, budeme ho nastavovat v hostitelském programu.

Nakonec musíme pro nově vzniklý subsystém přivést přerušení na kanále *ADCINT2*. V bloku *Hardware interrupt* dle tabulky [13] bude nastavení vektoru CPU [1 1] a vektoru PIE [1 2]. Aby se obě přerušení stíhaly, nastavíme v záložkách *Event Trigger* modulů *ePWM1 a ePWM4* vznik přerušení při každé druhé události *Second event*. V bloku *Hardware Interrupt* ještě nastavíme těmto přerušením nejvyšší priority, tedy vektor hodnot *Simulink task priorities* bude [1 2], protože nižší čísla znamenají vyšší prioritu.

Celkové blokové schéma v Simulinku je na obrázku E.1.

7.1.1 Hardwarové připojení

Ke standu pro asynchronní motor je LaunchPad připojen přes redukce, pro DC motor potom přes vodiče připojená PWM, napájení standu a kanál pro čtení proudu *ADCINB0*. Tuto situaci zachycuje obrázek 7.1.



Obr. 7.1: Hardwarové připojení pro zatěžování asynchronního motoru

7.1.2 Výsledky a měření

Motor byl roztočen na jmenovité otáčky 1350 ot/min a postupně zatěžován DC motorem. Proud DC motoru byl nastavován od 0 do 14 po 1 A. Zatěžovací moment DC motoru zjistíme jednoduše ze vztahu 7.1.

$$M = c\phi \cdot I \tag{7.1}$$

Konstanta $c\phi$ DC motoru je 0,0743. Při zatížení proudem 8 A, což odpovídá momentu přibližně 0,6 Nm DC motoru, vyreguloval asynchronní motor ještě otáčky na 1300 ot/min a při dalším zatěžování otáčky postupně klesaly. Při zatížení proudem 14 A se motor postupně zastavil, viz obrázek H.9. Proud 14 A odpovídá momentu zvratu 1 Nm. DC motor byl připojen na vodičích a měřený proud byl zarušený. Obrázek proto zachycuje pouze žádaný proud DC motorem.

7.2 Skalární řízení v otevřené smyčce

Postupujeme stejně jako v kapitole 7.1. Pouze zde vynecháme regulátor otáček. Při zkoušce zatěžování jsme nastavili synchronní otáčky 1500 ot/min. Skutečné otáčky byly stejně jako v případě skalárního řízení bez regulátoru otáček bez zatížení 1450 ot/min. Při postupném zvyšování zátěžného proudu otáčky klesaly, jmenovitých

otáček, tedy 1350 ot/min bylo dosaženo při zátěžném proudu 8 A, to odpovídá momentu 0,6 Nm. Proud byl nastavován opět po 1 A od 0 do 16 A. Motor se postupně zastavil při zátěžném proudu 16 A. Měření je v přílohách na obrázku H.8.

8 Vektorové řízení Asynchronního motoru

Při vektorovém řízení řídíme nejen amplitudy prostorových vektorů, ale také jejich vzájemný úhel. Podle určení vektoru magnetického toku rozlišujeme dva typy řízení [26]:

- **Přímé řízení:** Prostorový vektor magnetického toku se rekonstruuje z naměřených statorových napětí a proudů. S použití rekonstrukce skluzu není třeba čidlo otáček.
- **Nepřímé řízení:** Prostorový vektor magnetického toku je určen z naměřených skutečných otáček a rekonstrukce skluzu.

V této práci se budeme zabývat jednodušším nepřímým řízením, protože máme k dispozici čidlo otáček. Nejčastěji se v praxi setkáme v řízením orientovaným na rotorový tok. Skluz se vypočítává z proudu v ose q i_{sq} a rotorový tok z proudu v ose d i_{sd}. Osy d a q jsou navzájem kolmé a jsou spjaty s točivým magnetickým polem statoru. Vzhledem ke statoru tedy rotují synchronní rychlostí ω_s. Proudy i_{sdq} získáme měřením dvou statorových fázových proudů a následnou transformací pomocí Parkovy a Clarkové transformace.

8.1 Clarkové transformace

Než přivedeme měřené fázové proudy statoru i_a a i_b do transformace, musíme je škálovat. To provedeme opět stejným způsobem jako v kapitole 6.1.6. Jelikož měříme pouze 2 ze 3 fázových proudů, Clarkové transformaci provedeme dle vztahů [25]:

$$i_{s\alpha} = i_a \tag{8.1}$$

$$i_{s\beta} = \frac{1}{\sqrt{3}} \cdot i_a + \frac{2}{\sqrt{3}} \cdot i_b \tag{8.2}$$

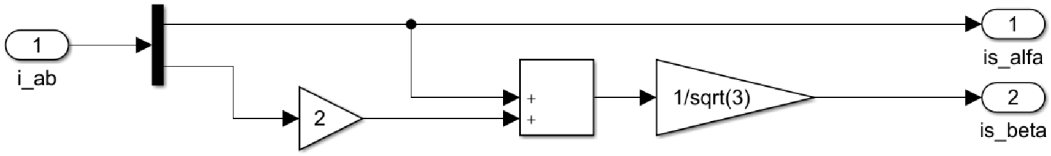
kde $i_{s\alpha}$ a $i_{s\beta}$ jsou statorové proudy v $\alpha\beta$ souřadnicích.

Clarkové transformace v Simulinku je zachycena obrázku 8.1

8.2 Parkova Transformace

Parkova transformace již převede proudy $i_{s\alpha}$ a $i_{s\beta}$ do na proudy i_{sd} a i_{sq} v dq souřadnicích. Vytvoříme ji dle vztahů:

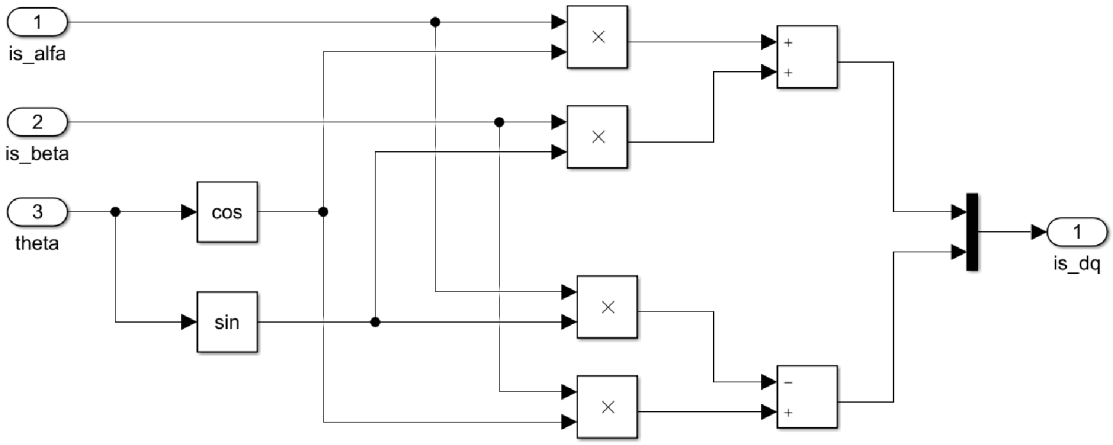
$$i_{sd} = cos(\vartheta) \cdot i_\alpha + sin(\vartheta) \cdot i_\beta \tag{8.3}$$



Obr. 8.1: Clarkové transformace

$$i_{sq} = -sin(\vartheta) \cdot i_\alpha + cos(\vartheta) \cdot i_\beta \tag{8.4}$$

kde ϑ je úhel natočení mezi $\alpha\beta$ a dq souřadnicemi.
V Simulinku je Parkova transformace znázorněna na obrázku 8.2.



Obr. 8.2: Parkova transformace

Schéma celé transformace proudů je v přílohách na obrázku F.3.

8.3 Identifikace parametrů Asynchronního motoru

K realizaci vektorového řízení v Simulinku budeme potřebovat vypočítat parametry náhradního modelu motoru. Ze štítku na obrázku 6.2 zjistíme jmenovitou napájecí frekvenci 50 Hz, jmenovité sdružené napětí 24 V a jmenovitý proud 9,45 A, jmenovitý moment 1,3 Nm, počet pólpáru 2 a skluzové otáčky 150 ot/min. Dále jsme změřili proud naprázdno 4,5 A a odpor fáze statoru 0,35 Ω. Následující výpočty vycházejí z modelu gama článku. Pro statorový tok Ψ_s platí:

$$\Psi_s = \frac{1}{\omega_s} \cdot \sqrt{{u_{s0}}^2 - (R_s \cdot i_{sd0})^2} \tag{8.5}$$

kde ω_s je synchronní rychlost, u_{s0} amplituda fázového napětí statoru, R_s odpor fáze statoru a proud i_{sd0} odpovídá amplitudě proudu naprázdno.

Po číselném dosazení:

$$\Psi_s = \frac{1}{2 \cdot \pi \cdot 50} \cdot \sqrt{(\frac{24}{\sqrt{3}} \cdot \sqrt{2})^2 - (0.35 \cdot 4,5 \cdot \sqrt{2})^2} = 0,062Wb \tag{8.6}$$

Indukčnost statorového vinutí:

$$L_s = \frac{\Psi_s}{i_{s0}} = \frac{0,062}{4,5 \cdot \sqrt{2}} = 10mH \tag{8.7}$$

Jmenovitý proud v ose q:

$$i_{sqn} = \frac{2}{3} \cdot M \cdot \frac{1}{pp \cdot \Psi_s} = \frac{2}{3} \cdot 1,3 \cdot \frac{1}{2 \cdot 0,062} = 7A \tag{8.8}$$

kde pp je počet pólpárů.

Rotorový odpor přepočítaný na stranu statoru:

$$R_R{}' = \frac{\omega_{skl} \cdot \Psi_s}{i_{sqn}} = \frac{2 \cdot \pi \cdot 150/60 \cdot 0,062}{7} = 0,139\Omega \tag{8.9}$$

kde ω_{skl} je skluzová rychlost.

Jmenovitý proud v ose d:

$$i_{sdn} = \sqrt{i_{sn}^2 - i_{sqn}^2} = \sqrt{(9,45 \cdot \sqrt{2})^2 - 7^2} = 11,4A \tag{8.10}$$

kde i_{sn} je amplituda jmenovitého fázového proudu.

Rozptylová indukčnost rotoru přepočítaná na stranu statoru:

$$L_{\sigma r}{}' = (i_{sdn} - i_{s0}) \cdot \frac{R_R{}'}{\omega_{skl} \cdot i_{sqn}} \tag{8.11}$$

Po dosazení:

$$L_{\sigma r}{}' = (11,4 - 4,5 \cdot \sqrt{2}) \cdot \frac{0,139}{2 \cdot \pi \cdot 150/60 \cdot 7} = 6,37mH \tag{8.12}$$

Činitel vazby k:

$$k = \sqrt{\frac{L_s}{L_s + L_{\sigma r}{}'}} = \sqrt{\frac{10}{10 + 6,37}} = 0,78 \tag{8.13}$$

Nyní vypočítáme parametry inverzního gama článku.

$$R_{Rinv}{}' = R_R{}' \cdot k^4 = 0,139 \cdot 0,78^4 = 0,05\Omega \tag{8.14}$$

Rozptylová indukčnost statoru:

$$L_{\sigma s} = L_s \cdot (1 - k^2) = 4mH \tag{8.15}$$

Indukčnost rotoru přepočítaná na stranu statoru:

$$L_R{}' = L_s \cdot k^2 = 10 \cdot 0,78^2 = 6mH. \tag{8.16}$$

A rotorový tok přepočítaný na stranu statoru:

$$\Psi_R{}' = \frac{U_{sa}}{\omega} = \frac{13,9}{2 \cdot \pi \cdot 50} = 0,044Wb \tag{8.17}$$

8.4 Kvazivektorové řízení s orientací na rotorový tok

„Kvazivektorové“ řízení je jednodušší forma vektorového řízení bez estimace rotorového toku. Tok se v našem případě nastaví na pevnou hodnotu. Při nepřímém vektorovém řízení s orientací na rotorový tok je skluzová úhlová rychlost ω_{skl} určována ze vztahu:

$$\omega_{skl} = \frac{i_{sq} \cdot {R_{Rinv}}'}{{\Psi_R}'} \tag{8.18}$$

kde ${\Psi_R}'$ je rotorový tok přepočten na stranu statoru.

Synchronní úhlovou rychlost určíme součtem skluzové a skutečné rychlosti:

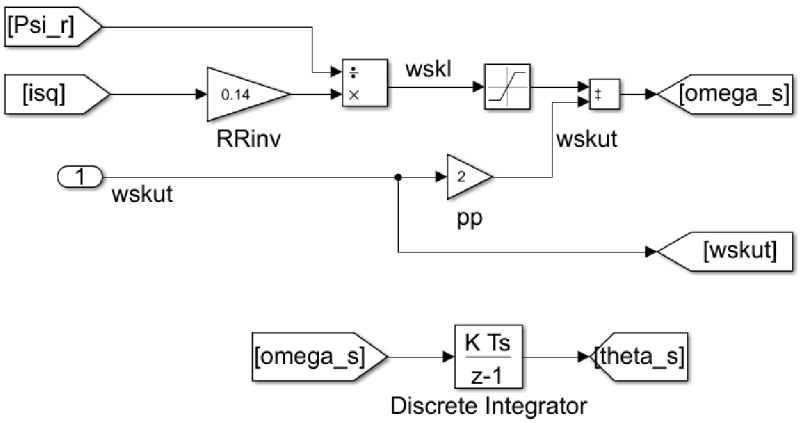
$$\omega_s = \omega_m \cdot pp + \omega_{skl} \tag{8.19}$$

kde ω_m je měřená úhlová rychlost a pp je počet pólpárů motoru.

Úhel ϑ mezi statorem a dq souřadnicemi je potom:

$$\vartheta = \int \omega_s dt \tag{8.20}$$

Určení synchronní rychlosti a úhlu v Simulinku je na obrázku 8.3.

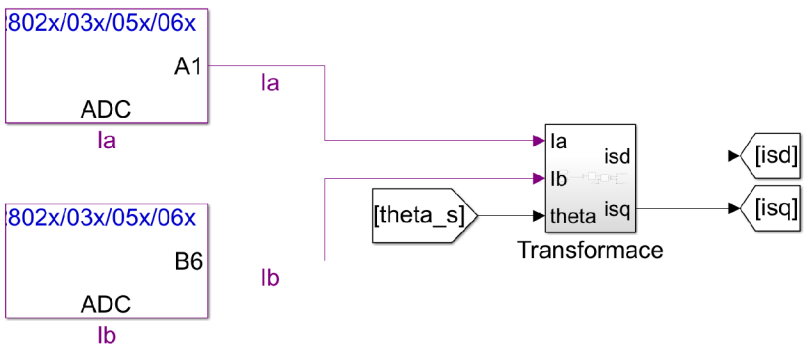


Obr. 8.3: Určení synchronní rychlosti a úhlu

Důležité je saturovat skluzovou rychlost, aby se nám motor nekontrolovatelně neroztočil do vysokých otáček. Meze saturace volíme ± 20 rad/s.

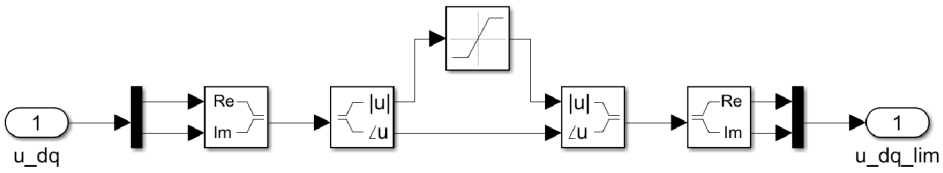
Dále vložíme do modelu ADC převodníky, které budou měřit proudy I_1 a I_2 a pomocí subsystému transformace F.3 převáděny do dq os viz obrázek 8.4.

Tvorba PWM, škálování PWM, inverzní Clarkové a Parkova transformace budou stejné jako v případě skalárního řízení, viz kapitola 6.1. Do Parkovy transformace přivedeme napětí u_{sd} a u_{sq}, které budou výstupem regulátorů proudů i_{sd} a i_{sq}. Omezení napětí u_{sd} a u_{sq} nebude stačit pouze nastavit v regulátorech, ale tyto dvě napětí



Obr. 8.4: Trasnformace proudů do dq os

převedeme na komplexní číslo a omezíme jeho modul. Jakou reálnou složku zvolíme napětí u_{sd} a imaginární složku napětí u_{sq}. Komplexní číslo vytvoří blok *Real-Imag to Complex* a jeho modul a úhel potom blok Complex to Magnitude-Angle. Modul potom přivedeme do bloku *Saturation* kde nastavíme meze jako ±15, to je asi 1,1 násobek fázového napětí 13,9 V. Modul a úhel opět převedeme na složky dq. Limitaci napětí najdeme na obrázku 8.5.



Obr. 8.5: Limitace napětí v osách dq

Regulátor proudu v ose d bude mít pevnou žádanou hodnotu proudu. Jako skutečná hodnota bude přiveden transformovaný proud i_{sd}. Regulátor proudu v ose q má jako žádanou hodnotu předřazený regulátor otáček a jako skutečná hodnota je přiveden transformovaný proud i_{sq}. Regulátor otáček porovnává žádanou hodnotu otáček posílanou po sériové lince a naměřenou hodnotu z čidla otáček.

Výše popsaný model je v subsystému přerušený na obrázku F.2. Mimo přerušení se potom nachází měření otáček, které je spouštěné s periodou 0,005 s, příjem ze sériové linky a hardwarové přerušení, které je opět synchronizováno s ADC převodníkem na kanále ADCINT1. Celkový model je na obrázku F.1.

8.4.1 Výsledky

Nejdříve byly vyladěny regulátory proudu. Po sériové lince byla odesílána data na vstupu a výstupu z regulátoru a podle nich byla nejdříve nastavena proporciální složka. K té byla dále nastavována integrační složka a proporciální byla úměrně snižována. Regulátory proudu byly vyladěny na hodnoty P: 0,5 a I: 100. Po vyladění regulátorů proudu byl stejným způsobem vyladěn regulátor otáček na hodnoty P: 0,5 a I: 1.

Proud v ose d a rotorový tok $\Psi_R{}'$ je pevně nastaven v programu. Po sériové lince se dají posílat žádané otáčky. Jako rotorový odpor $R_{Rinv}{}'$ byla v programu zvolena hodnota 0,14. Rotorový tok $\Psi_R{}'$ 0,044 Wb a proud v ose d 5 A.

Při rozběhu byl asynchronní motor spouštěn po rampě žádaných otáček viz obrázek H.10.

Závěr

Cílem této práce bylo prozkoumat metody rychlého programování mikrokontrolérů. Pro tuto práci byl vybrán mikrokontrolér z řady C2000 od firmy Texas Instruments na vývojovém kitu LaunchPad F28069M. Jako vývojové prostředí jsme zvolili MATLAB/Simulink a výsledný kód pro mikrokontrolér je odvozen z blokového schématu. Práce se zabývala návodem pro programování výše uvedeného mikrokontroléru v prostředí Simulink.

Dále je práce zaměřena již na praktické využití s laboratorním standem vyvinutým na Ústavu výkonové elektrotechniky a elektroniky FEKT VUT. Pro kompatibilní připojení LaunchPadu k tomuto standu byla navržena redukční DPS v programu EAGLE.

Jako první příklad aplikace v Simulinku popisuje tato práce řízení a regulaci DC motoru. Bylo vytvořeno bipolární i unipolární řízení H-můstku výkonového měniče. Frekvence PWM byla zvolena 20 kHz. Proud je měřen pomocí čidla LEM na laboratorním standu a snímán pomocí ADC převodníku, který je synchronizován s PWM. S ADC převodníkem je potom synchronizováno hardwarové přerušení. Otáčky jsou měřeny pomocí resolvéru, na standu potom měřený signál převáděn na enkodérové pulzy. Tyto pulzy snímá mikrokontrolér pomocí modulu QEP. Pro DC motor byl navržen regulátor proudu a otáček. Střídu do PWM lze nastavit buď jako pevnou hodnotu nebo po sériové lince zadat žádané otáčky. Pomocí sériové linky lze také v hostitelském zařízení zobrazovat průběh proudu a otáček.

Jako druhý příklad bylo realizováno skalární řízení asynchronního motoru v otevřené i uzavřené smyčce pomocí regulátoru proudu. Tvorba těchto programů vychází částečně z již realizované aplikace řízení DC motoru. Pro omezení proudových nárazů je v hostitelském programu vytvořena rozběhová rampa.

Jako třetí příklad bylo realizováno kvazivektorové řízení asynchronního motoru s orientací na rotorový tok.

Výhody prostředí Simulink spočívají v rychlém programování bez nutnosti znalosti programovacího jazyka. Také je možno využít externí mód pro vyladění například hodnot PI regulátoru.

Nevýhodou mohou být některé chybějící funkce u vytvořených bloků. Některé mohou být naopak složité a je potřeba důkladné prostudování těchto bloků v nápovědě od MathWorks. To je ostatně jediná dostupná literatura k tomuto tématu. Také je důležité pochopit princip a funkci jednotlivých periferií mikrokontroléru.

Výhodou použitého mikrokontroléru je možnost tvorby blokových schémat s využitím datového typu float, protože mikrokontrolér disponuje jednotkou fixed-point CPU. To umožňuje tvorbu jednodušších blokových schémat, kde není nutno provádět složité aritmetické operace. Ovšem použití datového typu float může ve složitějších

aplikacích způsobit zpomalení procesoru.

Přínosem této práce pro autora je základní pochopení principu a funkce mikrokontrolérů v pohonech, využití Simulinku pro praktické aplikace a důkladnější pochopení jednotlivých typů řízení motorů.

Na ústavu ÚVEE bude nadále vývojový kit LaunchPad spolu s laboratorním standem fungovat pro budoucí výuku studentů.

Literatura

[1] HÖLTTÄ, V., ERIKSSON, L., PALMROTH, L.: *Rapid control prototyping tutorial with application examples.* January, 2014. 7 s.

[2] BERGMANN, A.: *Benefits and Drawbacks of Model-based Design.* September, 2014. 6 s. KMUTNB International Journal of Applied Science and Technology. 7. 15-19. DOI: 10.14416/j.ijast.2014.04.004.

[3] KLÍMA, B.: *Mikroprocesorové řízení elektrických pohonů.* El. skriptum FEKT VUT Brno. 2014. 89 s.

[4] TEXAS INSTRUMENTS: *C2000 32-bit microcontrollers* [online]. [cit. 7. 10. 2019]. Dostupné z URL: <http://www.ti.com/microcontrollers/c2000-real-time-control-mcus/overview.html>.

[5] TEXAS INSTRUMENTS: *BOOSTXL-DRV8301 Hardware User's Guide.* SLVU974–October 2013. 13 s.

[6] MATH WORKS: *Generate C and C++ code from Simulink and Stateflow models* [online]. [cit. 5. 10. 2019]. Dostupné z URL: <https://www.mathworks.com/products/simulink-coder.html>.

[7] MATH WORKS: *Real-Time Workshop For Use with SIMULINK User's guide* Version 3, 1999. 386 s.

[8] MATH WORKS: *Embedded Coder* [online]. [cit. 18. 12. 2019]. Dostupné z URL: <https://www.mathworks.com/products/embedded-coder.html>.

[9] LabView: *Getting Started with the NI LabVIEW™ C Generator* July, 2010. 16 s.

[10] POWERSIM: *SimCoder User's guide* Version 12.0, 2019. 284 s.

[11] LINZ Center of Mechatronics GMBH: *X2C Rapid software prototyping and simulation tool* [cit. 9 10. 2019]. Dostupné z URL: <https://x2c.lcm.at/>

[12] TEXAS INSTRUMENTS: *TI InstaSPIN™ motor control solutions* [online]. [cit. 18. 11. 2019]. Dostupné z URL: <http://www.ti.com/microcontrollers/c2000-real-time-control-mcus/applications/instaspin.html#foc>.

[13] MATH WORKS: *C28x Hardware Interrupt* [online]. [cit. 18. 12. 2019]. Dostupné z URL:
<https://www.mathworks.com/help/supportpkg/texasinstrumentsc2000/ref/c28xhardwareinterrupt.html>.

[14] MATH WORKS: *C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x /F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x ePWM* [online]. [cit. 25. 11. 2019]. Dostupné z URL:
<https://www.mathworks.com/help/supportpkg/texasinstrumentsc2000/referencelist.html?type=block&s_cid=doc_ftr>.

[15] EmSys:TM4C123 PWM Programming *Pulse-Width Modulation — What is it?* [online]. [cit. 19. 12. 2019]. Dostupné z URL:
<http://shukra.cedt.iisc.ernet.in/edwiki/EmSys:TM4C123_PWM_Programming>.

[16] MATH WORKS: *Parameter Tuning and Signal Logging with Serial Communication* [online]. [cit. 28. 10. 2019]. Dostupné z URL:
<https://www.mathworks.com/help/supportpkg/texasinstrumentsc2000/examples.html?s_tid=CRUX_gn_example>.

[17] SZYMCZAK, J., O'MEARA, S., GEALON, J., S., NELSON DE LA RAMA, CH.: *Precision Resolverto-Digital Converter Measures Angular Position and Velocity.* Analog Dialogue 48-03, March, 2014. 6 s.

[18] TEXAS INSTRUMENTS: *TMS320x2806x Technical Reference Manual* November, 2019. 1126 s.

[19] LIDMILA, P.: *Identifikace parametrů motorů pro laboratorní stand.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 45 s. Vedoucí bakalářské práce Ing. Dalibor Červinka, Ph.D..

[20] HUDÁK, O.: *Laboratorního soustrojí s asynchronním a stejnosměrným motorem.* [cit. 12. 4. 2020]. Dostupné z URL:
<https://www.vutbr.cz/studenti/zav-prace/detail/48728>. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 59 s. Vedoucí bakalářské práce Ing. Dalibor Červinka, Ph.D..

[21] MATH WORKS: *Permanent Magnet Synchronous Motor Field-Oriented Control* [online]. [cit. 15. 3. 2020]. Dostupné z URL:
<https://ch.mathworks.com/help/supportpkg/texasinstrumentsc2000/

examples/permanent-magnet-synchronous-motor-field-oriented-control.html>.

[22] VIDLÁK, M.: *Modely řízení asynchronního motoru* [online]. [cit. 12. 4. 2020]. Dostupné z URL: <https://www.vutbr.cz/studenti/zav-prace/detail/116826>. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav výkonové elektrotechniky a elektroniky. 78 s Vedoucí práce Ing. Ivo Pazdera, Ph.D..

[23] PATOČKA, M.: *Vybrané statě z elektroniky - Svazek II Pulsní měniče bez transformátoru*. El. skriptum FEKT VUT Brno. 2005. 109 s.

[24] PABLO, Santiago de, L., C., HERRERO a J., M., RUIZ: *A simpler and faster method for SVM implementation* [online]. 2007. Aalborg [cit. 2020-05-19]. ISBN 9789075815108. Dostupné z URL: <http://www.epyme.uva.es/Docs/Public/Conferences/Epe2007a.pdf>.

[25] TEXAS INSTRUMENTS: *Clarke & Park Transforms on the TMS320C2xx* [online]. [cit. 12. 4. 2020]. Dostupné z URL: <http://www.ti.com/lit/an/bpra048/bpra048.pdf>.

[26] SKALICKÝ, J.: *Elektrické regulované pohony*. El. skriptum FEKT VUT Brno. 2007. 123 s.

Seznam symbolů, veličin a zkratek

ADC	Analog-to-digital converter
AC	Alternating Current
BLDC	Brushless DC electric motor
CCS	Code Composer Studio
DC	Direct Current
DSP	Digital Signal Processor
EC	Electronic commutator
FED	Falling Edge
HIL	Hardware-in-the-loop
LED	Light-Emitting Diode
MBD	Model Based Design
PI	Proportional-Integral
PWM	Pulse Width Modulation
SOC	Start of Conversion
RED	Raising Edge
RTW	Real-Time Workshop
TI	Texas Instruments
USB	Universal Serial Bus
μ**P**	Mikroprocesor
$C\Phi$	Konstanta motoru [V/s]
$cos\Phi$	Účiník [-]
f	Frekvence [Hz]
F_O	Přenos optimálního modulu [-]
f_{proc}	Pracovní frekvence procesoru [Hz]
f_{pwm}	Frekvence pulsní šířkové modulace [Hz]
F_{Ri}	Přenos regulátoru proudu [-]
$F_{R\omega}$	Přenos regulátoru otáček [-]
F_S	Přenos otevřené smyčky [-]
I	Elektrický proud [A]
I_f	Fázový proud [A]
I_{max}	Maximální proud [A]
I_n	Jmenovitý proud [A]
i_{s0}	Statorový proud naprázdno [A]
$i_{s\alpha}$	Statorový proud v ose α [A]
$i_{s\beta}$	Statorový proud v ose β [A]
i_{sd}	Statorový proud v ose d [A]
i_{sqn}	Jmenovitý statorový proud v ose d [A]

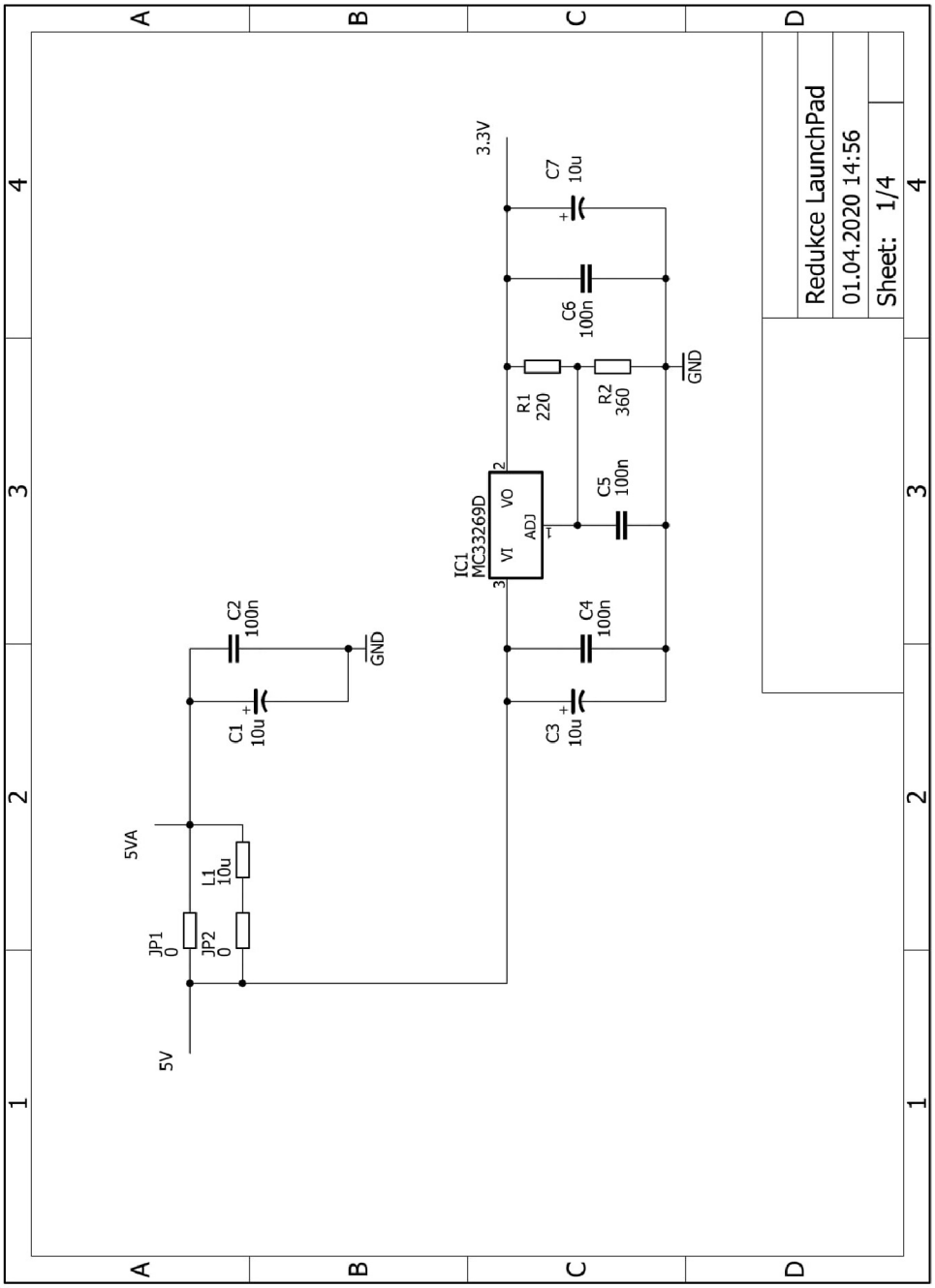
i_{sq}	Statorový proud v ose q [A]
i_{sqn}	Jmenovitý statorový proud v ose q [A]
J	Moment setrvačnosti $[kg \cdot m^2]$
k	Činitel vazby [-]
K_i	Zesílení čidla proudu [-]
K_m	Konstanta měniče [-]
K_S	Konstanta soustavy [-]
K_{td}	Zesílení čidla otáček [-]
L	Indukčnost [H]
L_s	Indukčnost statorového vinutí [H]
$L_{\sigma s}$	Rozptylová indukčnost statoru [H]
$L_{\sigma r'}$	Rozptylová indukčnost rotoru přepočítaná na stranu statoru [H]
$L_R{}'$	Indukčnost rotoru přepočítaná na stranu statoru [H]
M	Moment [Nm]
M_{mech}	Moment mechanický [Nm]
n	Otáčky [1/min]
η	Účinnost [-]
P_{el}	Elektrický výkon [W]
P_{mech}	Mechanický výkon [W]
pp	Počet pólpárů motoru [-]
R	Odpor [Ω]
R_s	Odpor statorového vinutí [Ω]
$R_R{}'$	Rotorový odpor přepočítaný na stranu statoru [Ω]
$R_{Rinv}{}'$	Rotorový odpor inverzního gama článku přepočítaný na stranu statoru [Ω]
T_A	Časová konstanta zátěže [s]
T_{BCLK}	Perioda jednoho cyklu mikroprocesoru [s]
T_{BPRD}	Hodnota registru periody nosného trojúhelníkového signálu [-]
T_{CTR}	Perioda nosného trojúhelníkového signálu [s]
$t_{deadtime}$	Doba deadtime [s]
T_{PWM}	Perioda pulsní šířkové modulace [s]
τ_f	Časová konstanta čidla otáček [s]
τ_m	Časová konstanta měniče [s]
τ_σ	Součtová časová konstanta [s]
ϑ	Úhel [-]
U_{cc}	Napájecí napětí měniče [V]
U_f	Fázové napětí [V]
U_n	Jmenovité napětí [V]
u_{s0}	Statorové napětí naprázdno [V]

$u_{s\alpha}$	Statorové napětí v ose α [V]
$u_{s\beta}$	Statorové napětí v ose β [V]
u_{sd}	Statorové napětí v ose d [V]
u_{sd}	Statorové napětí v ose q [V]
v	Rychlost [ms]
ω	Úhlová rychlost [rad/s]
ω_m	Mechanická úhlová rychlost [rad/s]
ω_s	Synchronní úhlová rychlost [rad/s]
ω_{skl}	Skluzová úhlová rychlost [rad/s]
ΔX	Přítůstek polohy [-]
Ψ_s	Statorový magnetický tok [Wb]
$\Psi_R{}'$	Rotorový tok přepočítaný na stranu statoru [Wb]

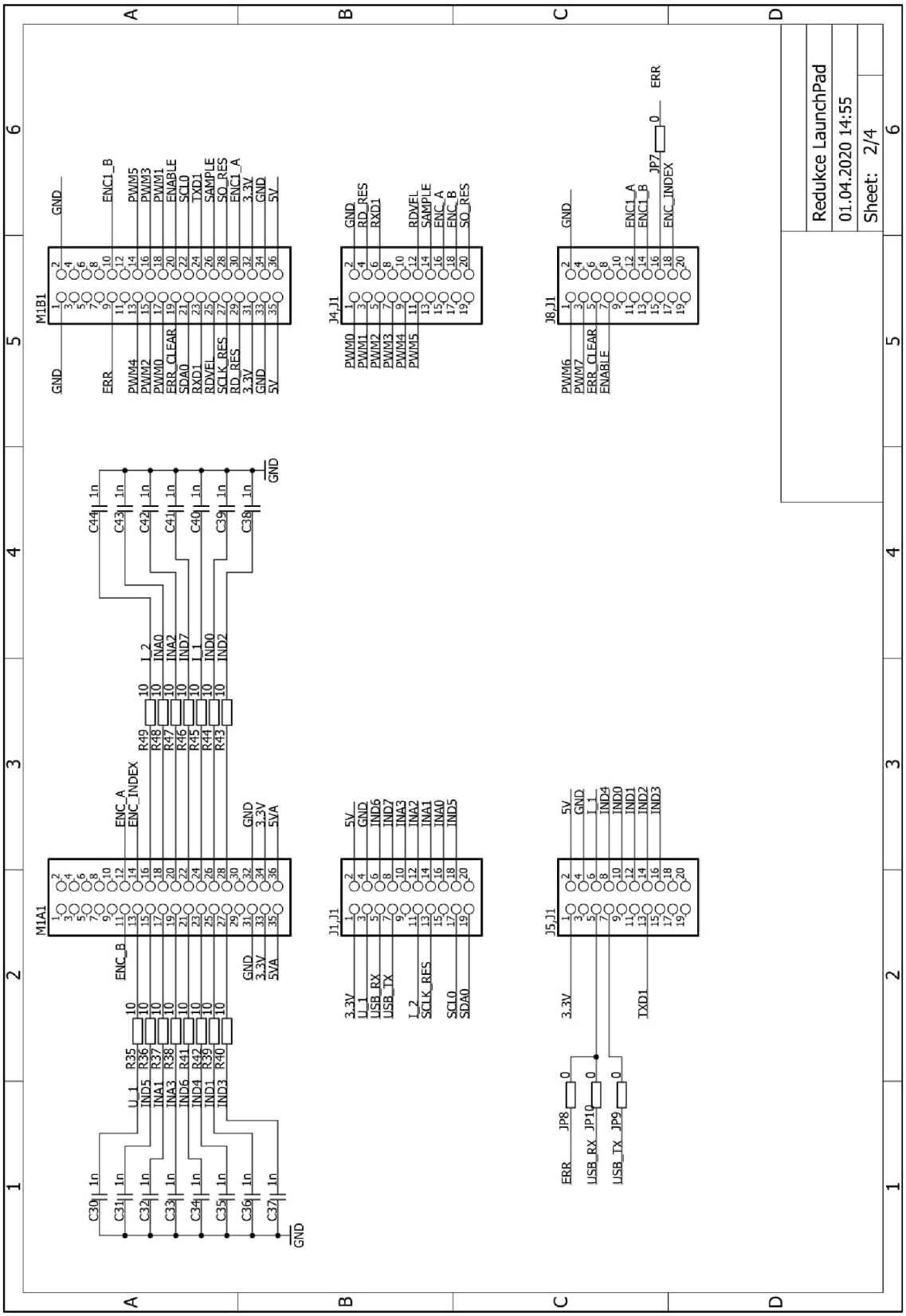
Seznam příloh

A Redukce pro LaunchPad F28069M

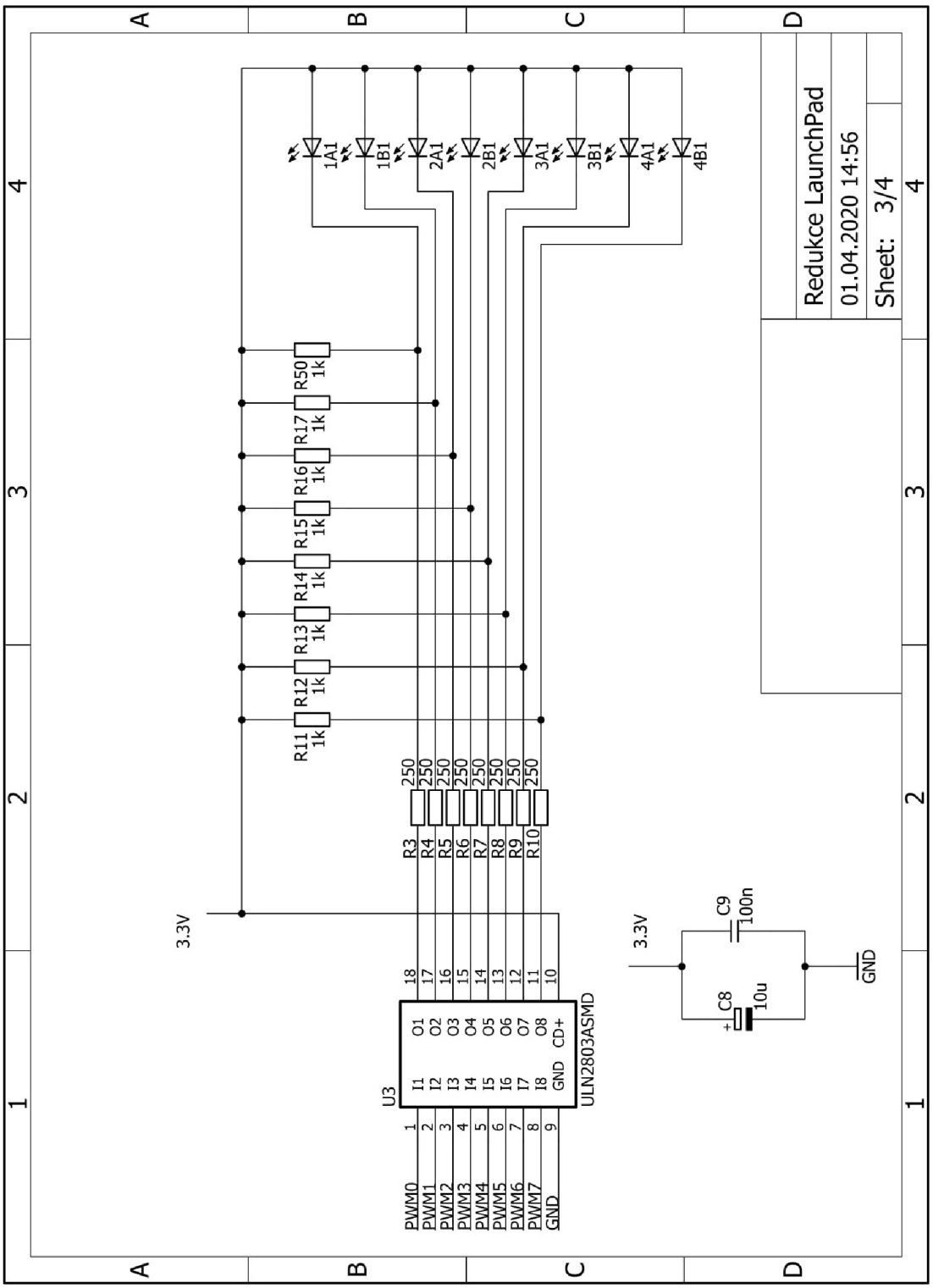
A.1 Schémata



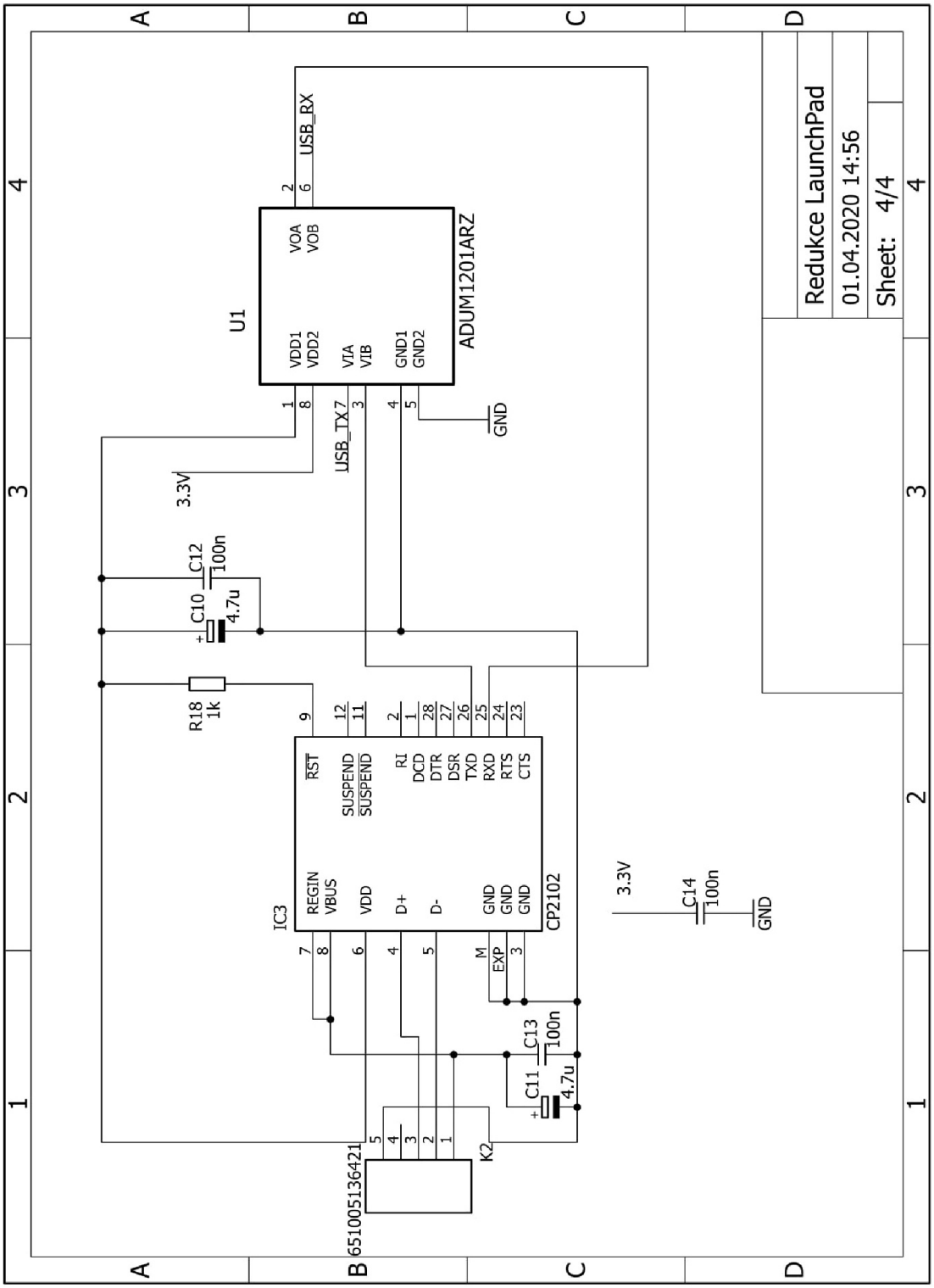
Obr. A.1: Zdroj 3,3 V



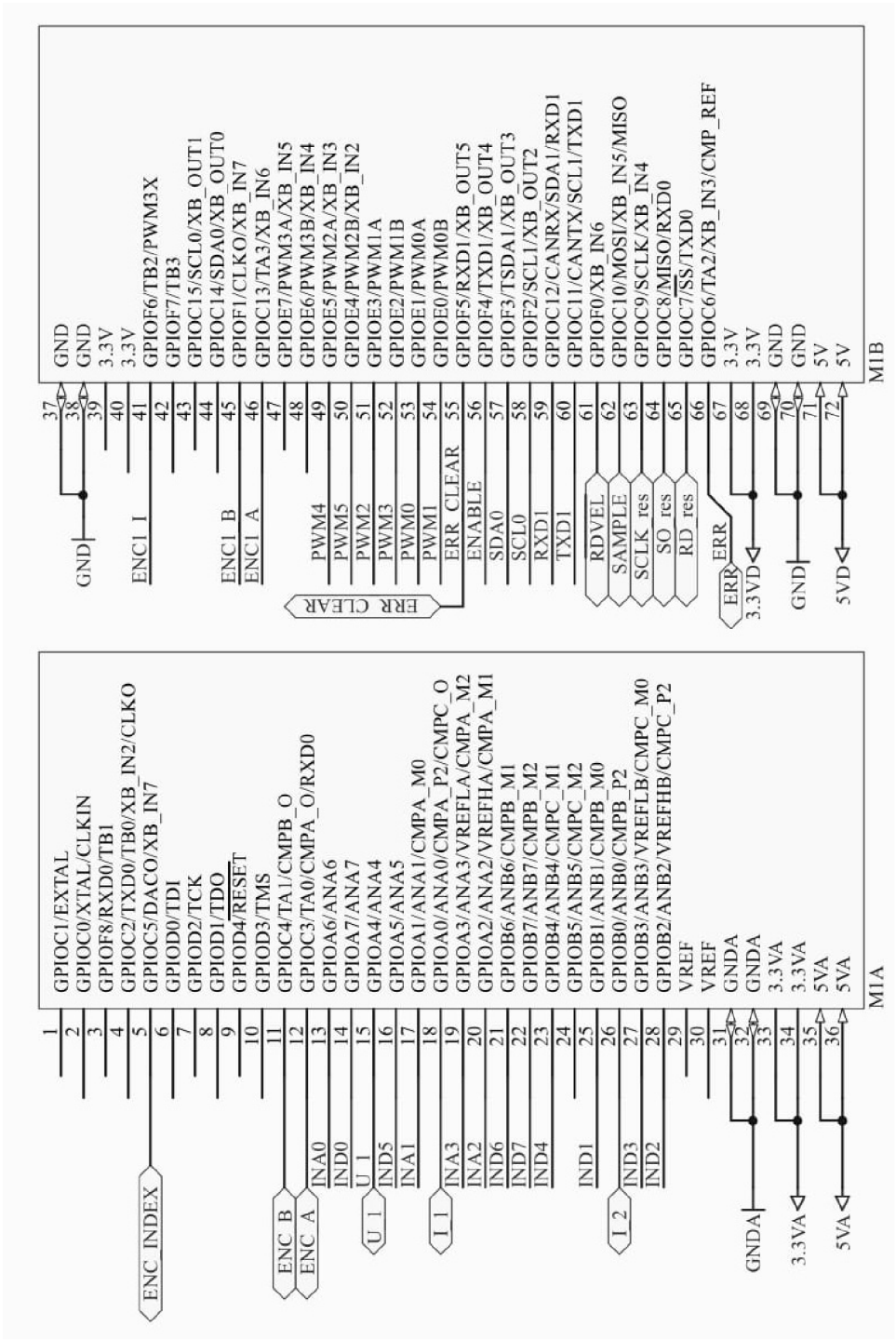
Obr. A.2: Zapojení konektorů mezi standem a LaunchPad



Obr. A.3: Detektor PWM



Obr. A.4: Rohraní USB



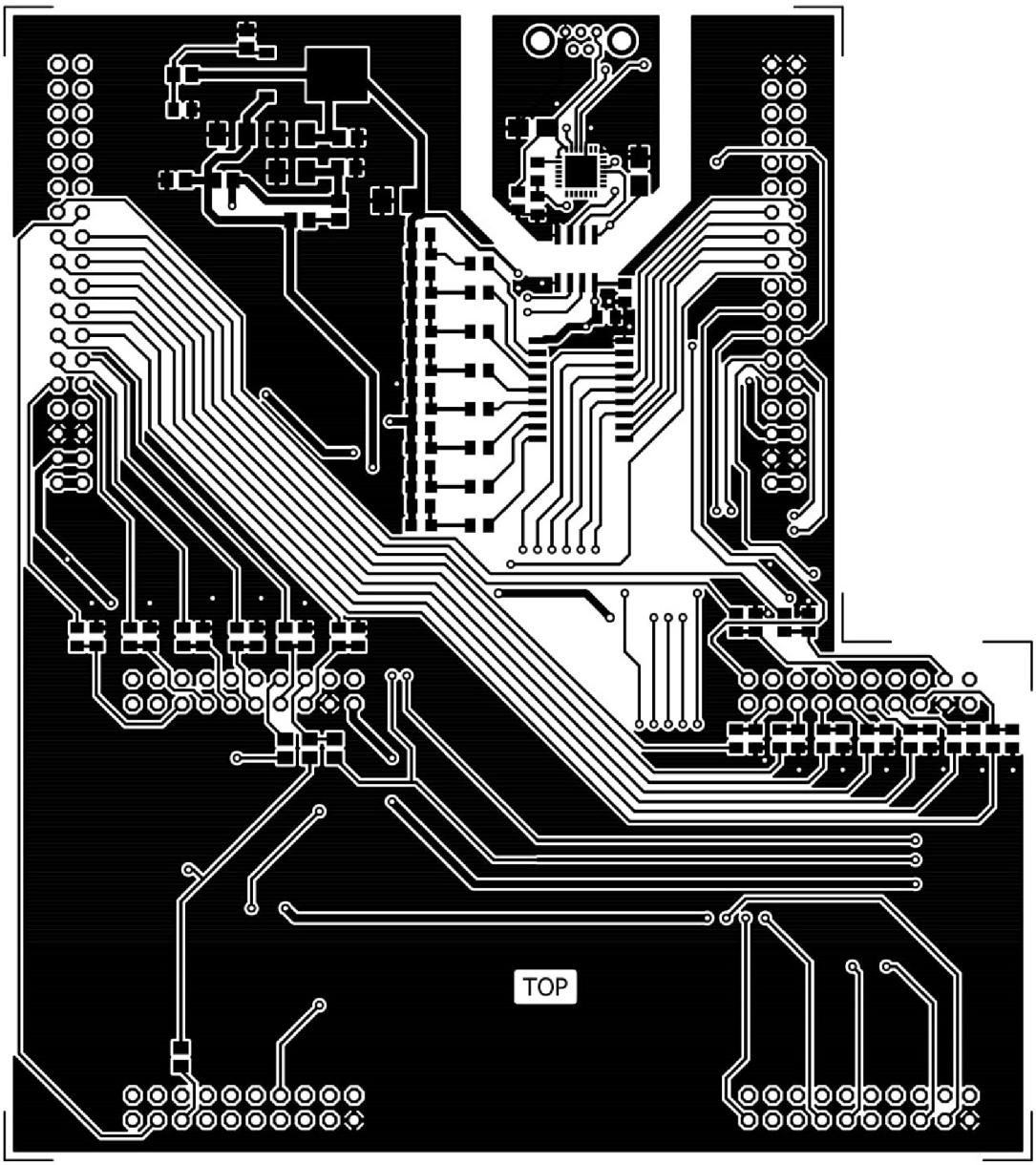
Obr. A.5: Konektory Standu

Stand		LaunchPad	
Pin	Signál	Pin	Signál
11	ENC_B	J2.12	EQEP1B
12	ENC_A	J2.13	EQEP1A
13	U_1	J1.2	ADCINA6
14	ENC_IND	J6.52	EQEP1I
15	IND5	J3.29	ADCINA1
16	I_2	J1.6	ADCINB6
17	INA1	J3.27	ADCINA0
18	INA0	J3.28	ADCINB0
19	INA3	J3.25	ADCINA2
20	INA2	J3.26	ADCINB2
21	IND6	J3.23	ADCINA7
22	IND7	J3.24	ADCINB1
23	IND4	J7.64	ADCINB4
24	I_1	J7.63	ADCINB7
25	IND1	J7.66	ADCINB5
26	IND0	J7.65	ADCINA5
27	IND3	J7.68	ADCINB3
28	IND2	J7.67	ADCINA3
31	GND	-	GND
32	GND	-	GND
33	3.3V	-	3.3V
34	3.3V	-	3.3V
35	5VA	-	-
36	5VA	-	-
37	GND	-	GND
38	GND	-	GND
45	ERR	J5.43, J6.53	FAULT
46	ENC1_B	J6.54	EQEP2B
49	PWM4	J4.39	EPWM3A
50	PWM5	J4.40	EPWM3B
51	PWM2	J4.35	EPWM2A
52	PWM3	J4.36	EPWM2B
53	PWM0	J4.37	EPWM1A
54	PWM1	J4.38	EPWM1B
55	ERR_CLEAR	J8.78	GPIO8
56	ENABLE	J8.77	GPIO9
57	SDA0	J1.10	SDAA
58	SCL0	J1.9	SCLA
59	RXD1	J2.18	SCITXDB
60	TXD1	J5.47	SCIRXDB
61	RDVEL	J2.15	GPIO10
62	SAMPLE	J2.14	GPIO11
63	SCLK_RES	J1.7	SPICLK
64	SO_RES	J2.11	SPISOMIA
65	RD_RES	J2.19	SPISTEA
66	ENC1_A	J6.55	EQEP2A
67	3.3V	-	3,3V
68	3.3V	-	3,3V

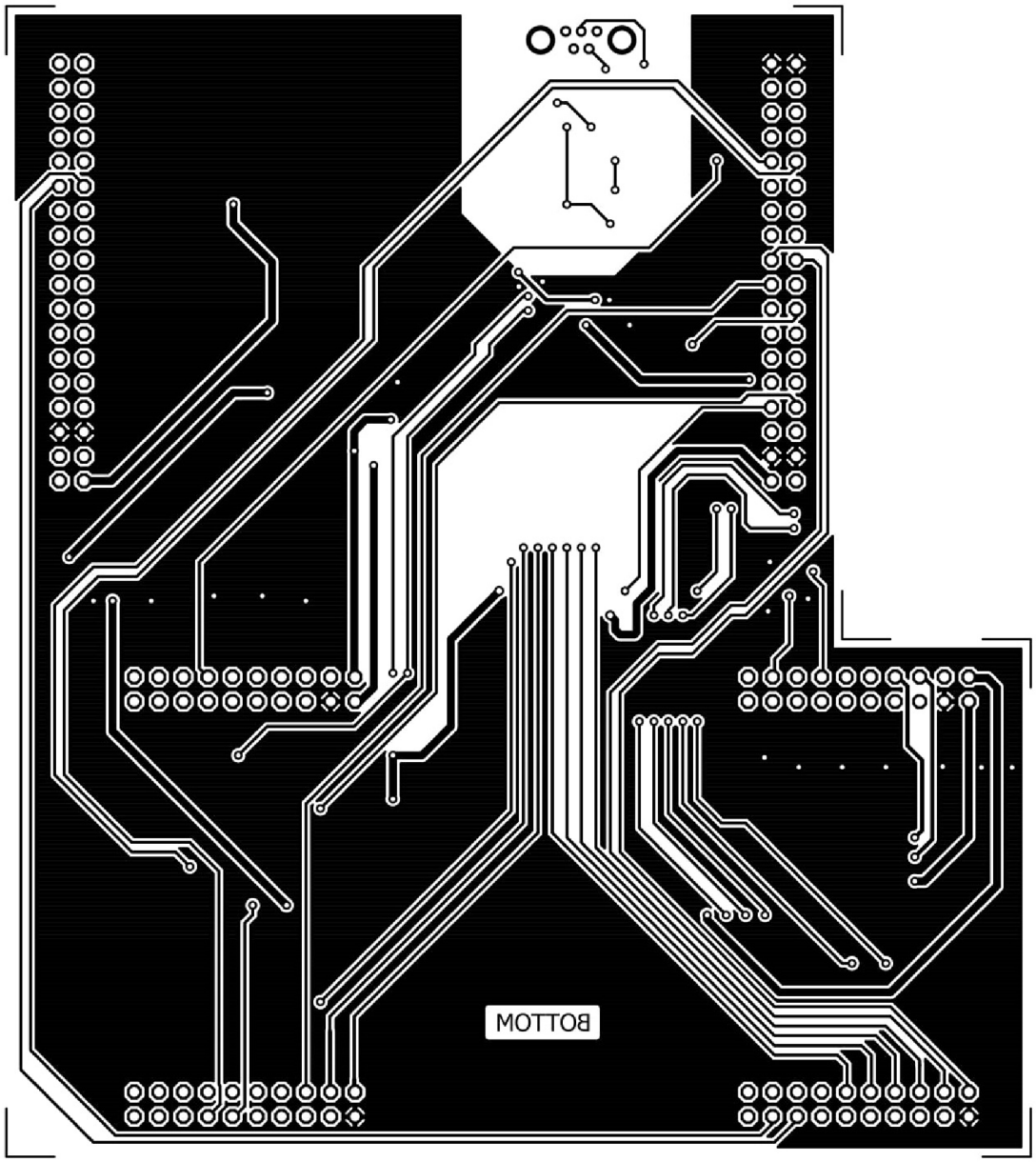
Stand		LaunchPad	
Pin	Signál	Pin	Signál
69	GND	-	GND
70	GND	-	GND
71	5V	-	5V
72	5V	-	5V
-	PWM6	J8.79	EPWM4A
-	PWM7	J8.80	EPWM4B
-	VOB	J1.3, J5.43	USB_RX
-	VIA	J1.4, J5.44	USB_TX

Obr. A.6: Tabulka pinů

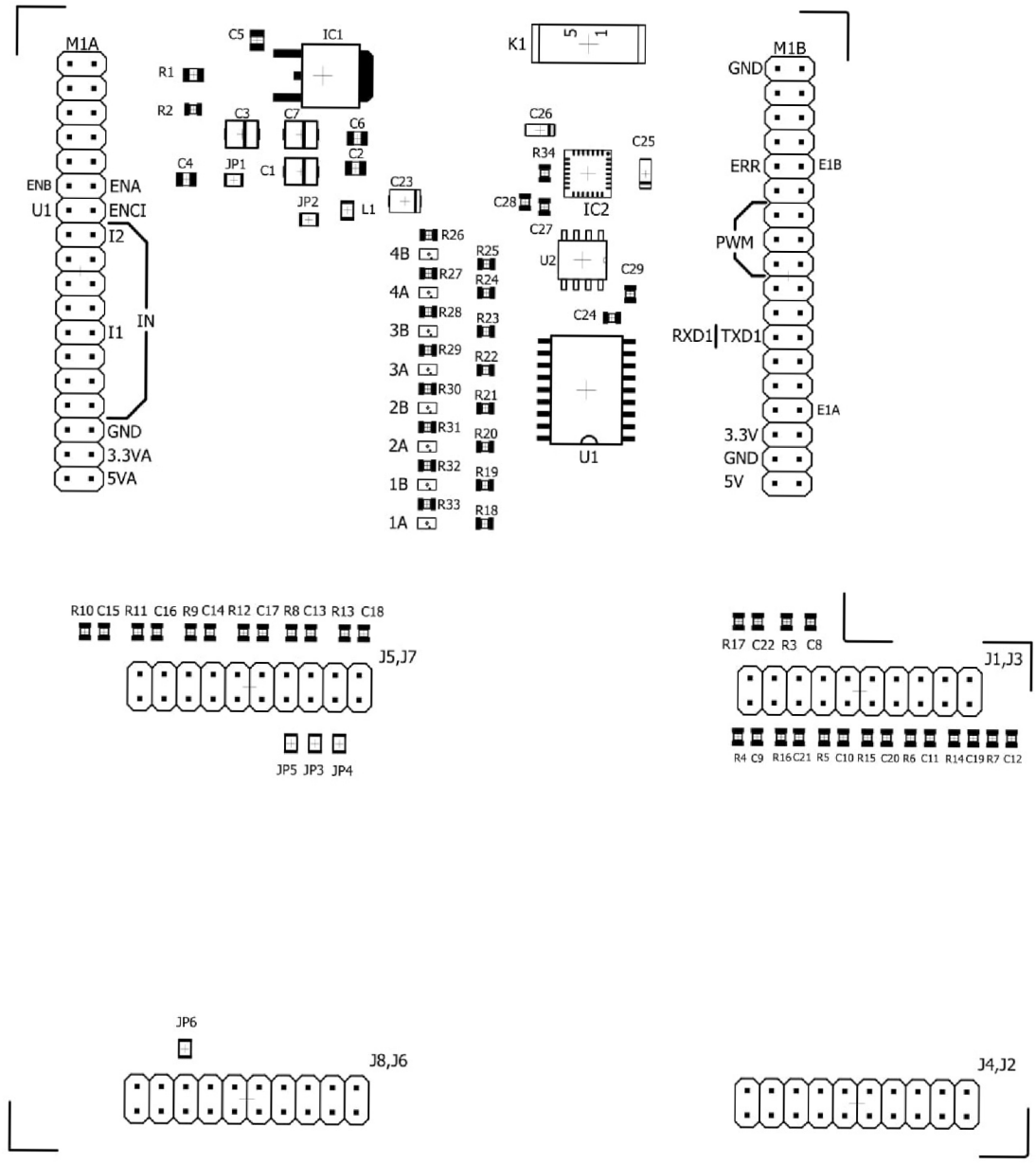
A.2 DPS



Obr. A.7: DPS ze strany TOP



Obr. A.8: DPS ze strany BOTTOM



Obr. A.9: Rozmístění součástek na straně TOP

Součást	Hodnota	Pouzdro
1A	LED	SML0805
1B	LED	SML0805
2A	LED	SML0805
2B	LED	SML0805
3A	LED	SML0805
3B	LED	SML0805
4A	LED	SML0805
4B	LED	SML0805
C1	10u	C3528-21/T
C2	100n	C0805
C3	10u	C3528-21/T
C4	100n	C0805
C5	100n	C0805
C6	100n	C0805
C7	10u	C3528-21/T
C8	1n	805
C9	1n	805
C10	1n	805
C11	1n	805
C12	1n	805
C13	1n	805
C14	1n	805
C15	1n	805
C16	1n	805
C17	1n	805
C18	1n	805
C19	1n	805
C20	1n	805
C21	1n	805
C22	1n	805
C23	10u	B/3528-21R
C24	100n	805
C25	4.7u	A/3216-18W
C26	4.7u	A/3216-18W
C27	100n	805
C28	100n	805
C29	100n	805
IC1	MC33269D	DPACK
IC2	CP2102	MLP28
J1,J3		2X10
J4,J2		2X10
J5,J7		2X10
J8,J6		2X10
JP1	0	M0805
JP2	0	M0805
JP3	0	M0805
JP4	0	M0805
JP5	0	M0805

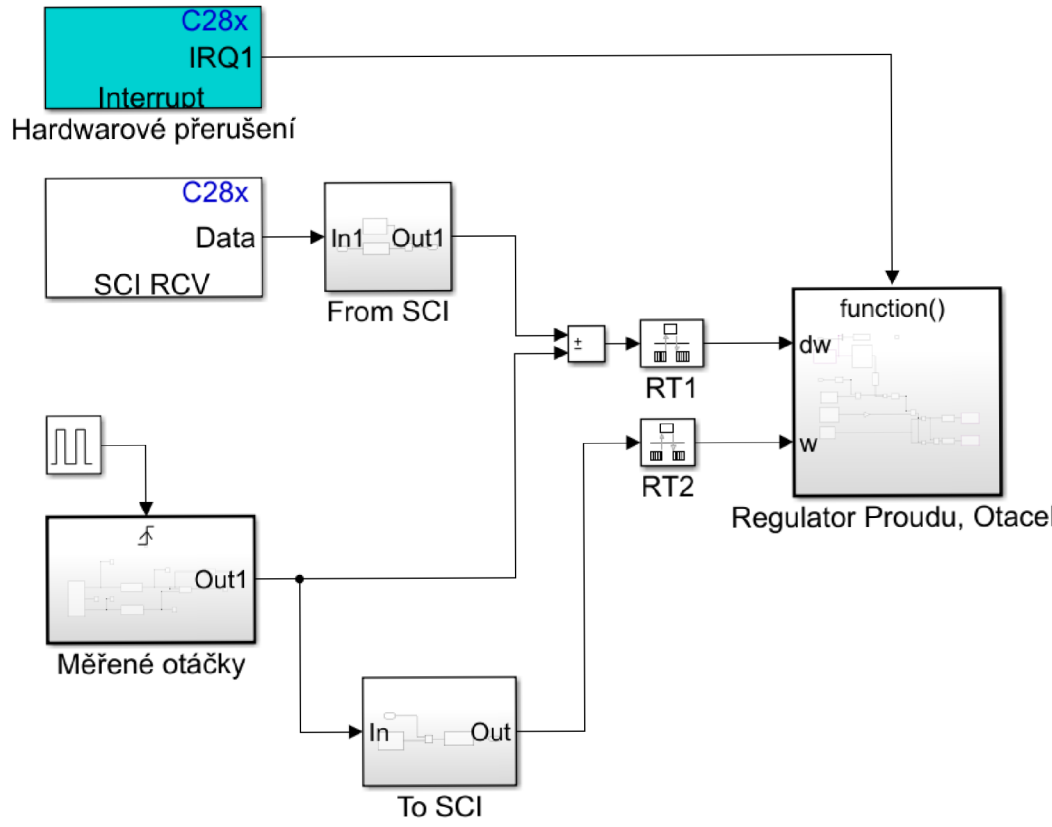
Součást	Hodnota	Pouzdro
JP6	0	M0805
K1	6510051364	21 651005136421
L1	10u	M0805
M1A		2X18
M1B		2X18
R1	220	R0805
R2	360	805
R3	10	805
R4	10	805
R5	10	805
R6	10	805
R7	10	805
R8	10	805
R9	10	805
R10	10	805
R11	10	805
R12	10	805
R13	10	805
R14	10	805
R15	10	805
R16	10	805
R17	10	805
R18	250	805
R19	250	805
R20	250	805
R21	250	805
R22	250	805
R23	250	805
R24	250	805
R25	250	805
R26	1k	805
R27	1k	805
R28	1k	805
R29	1k	805
R30	1k	805
R31	1k	805
R32	1k	805
R33	1k	805
R34	1k	805
U1	ULN2803ASM	D SOP18
U2	ADUM1201AR	Z SOIC127P600X

Obr. A.10: Seznam součástek

B Řízení DC motoru v Simulinku

B.1 Celkové schéma programu

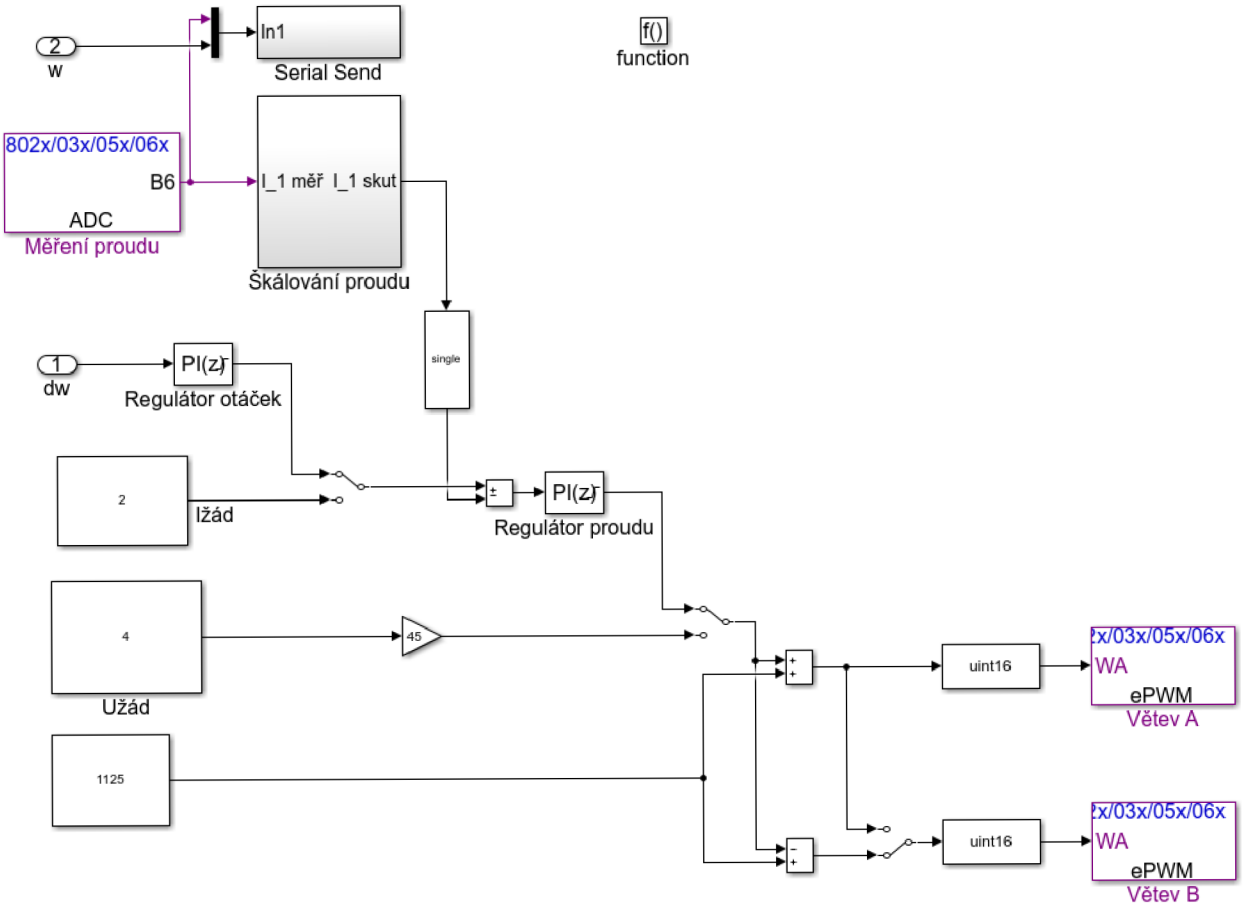
Výsledné schéma programu řízení DC motoru nalezneme na obrázku B.1:



Obr. B.1: Základní schéma programu řízení DC motoru

B.2 Schéma subsystému přerušení

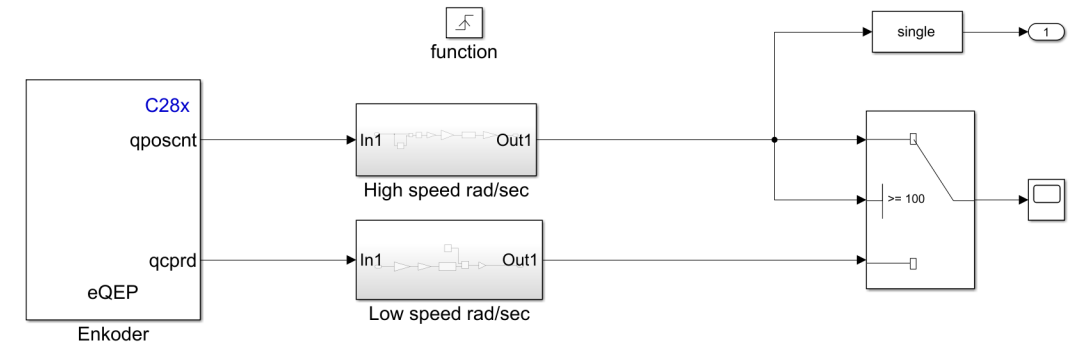
Na obrázku B.2 nalezneme schéma subsystému přerušení, které obsahuje snímání proudu, tvorbu PWM, regulátor proudu a regulátor otáček.



Obr. B.2: Schéma subsystému přerušení

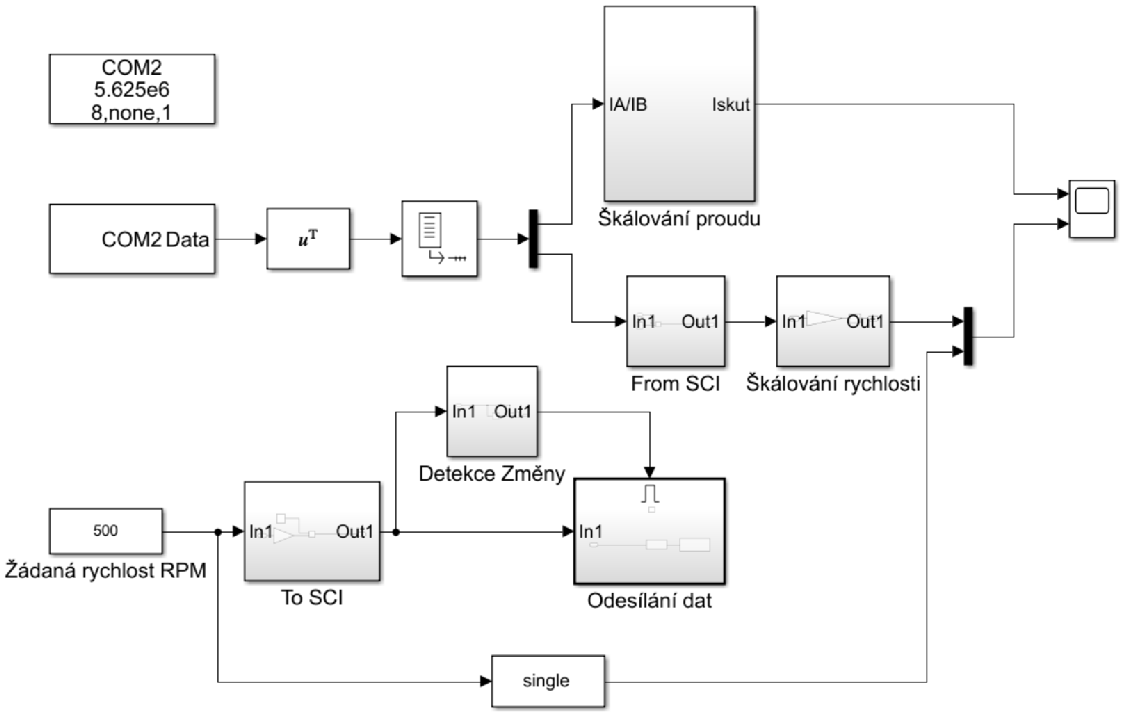
B.3 Blokové schéma snímání rychlosti

Přepínání rychlosti mezi High speed a Low speed je realizování přepínačem při dosažení 100 rad/s.



Obr. B.3: Schéma snímání rychlosti z enkodéru

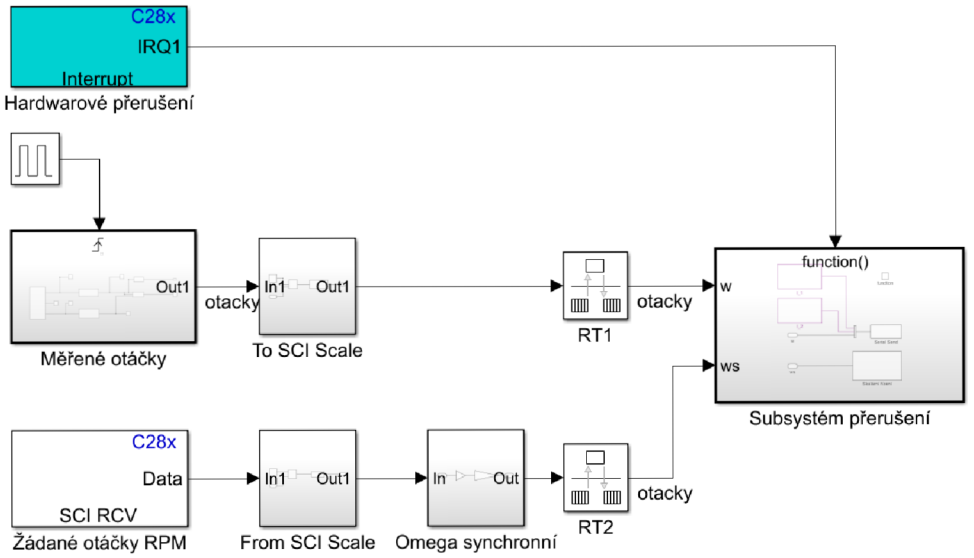
B.4 Hostitelský program pro sériovou linku



Obr. B.4: Hostitelský program

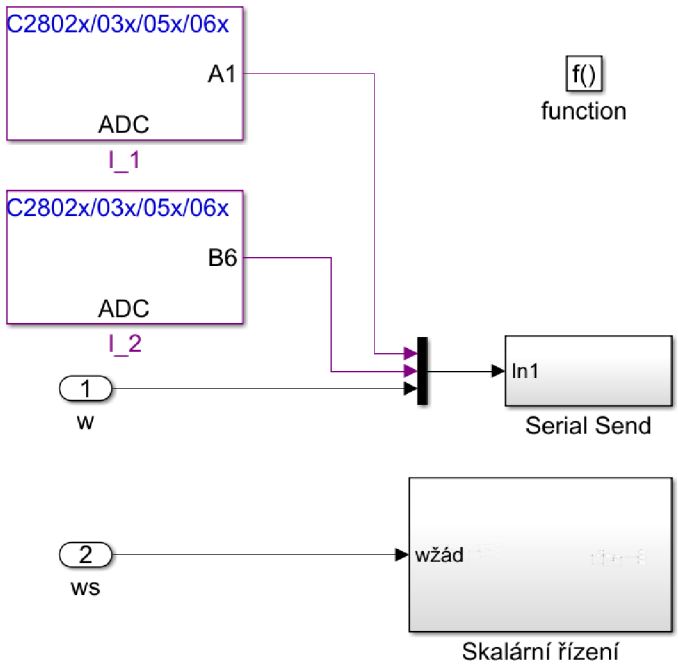
C Schémata skalárního řízení asynchronního motoru v otevřené smyčce

C.1 Hlavní program



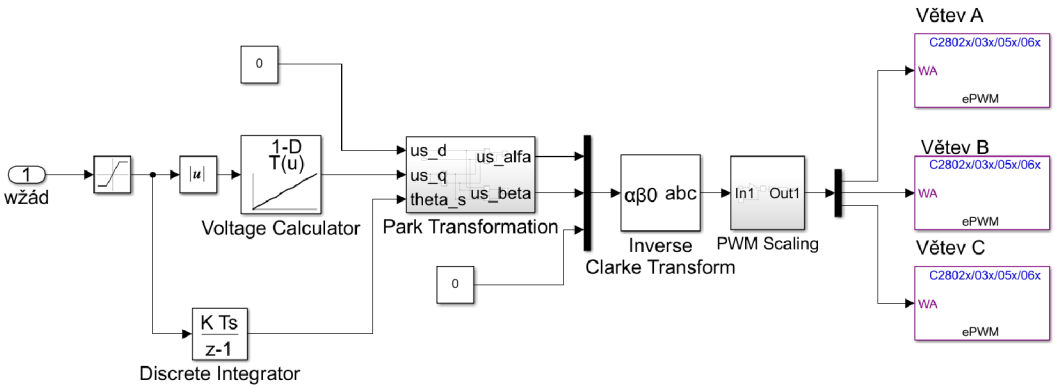
Obr. C.1: Schéma celkového programu

C.2 Subsystém přerušení



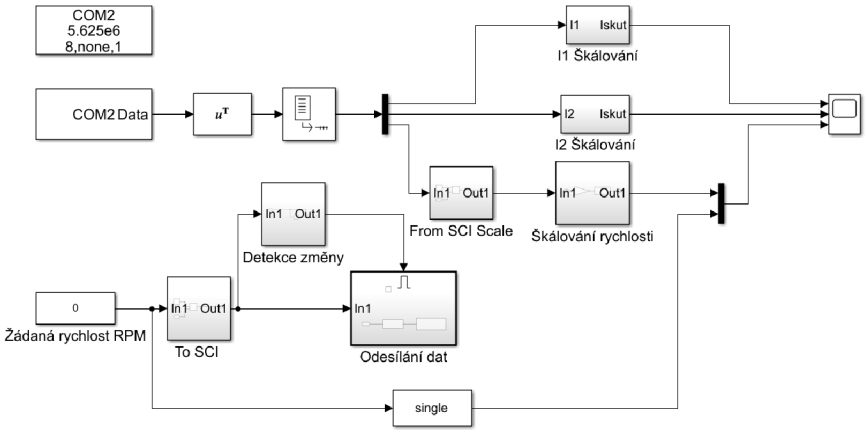
Obr. C.2: Měření proudu v přerušení

C.3 Skalární řízení



Obr. C.3: Schéma skalárního řízení v otevřené smyčce v Simulinku

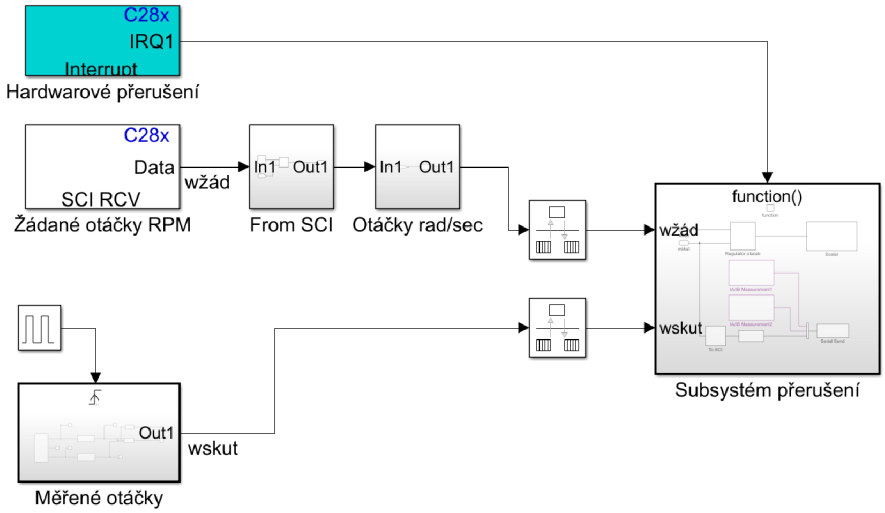
C.4 Hostitelský program



Obr. C.4: Schéma hostitelského programu pro ASM

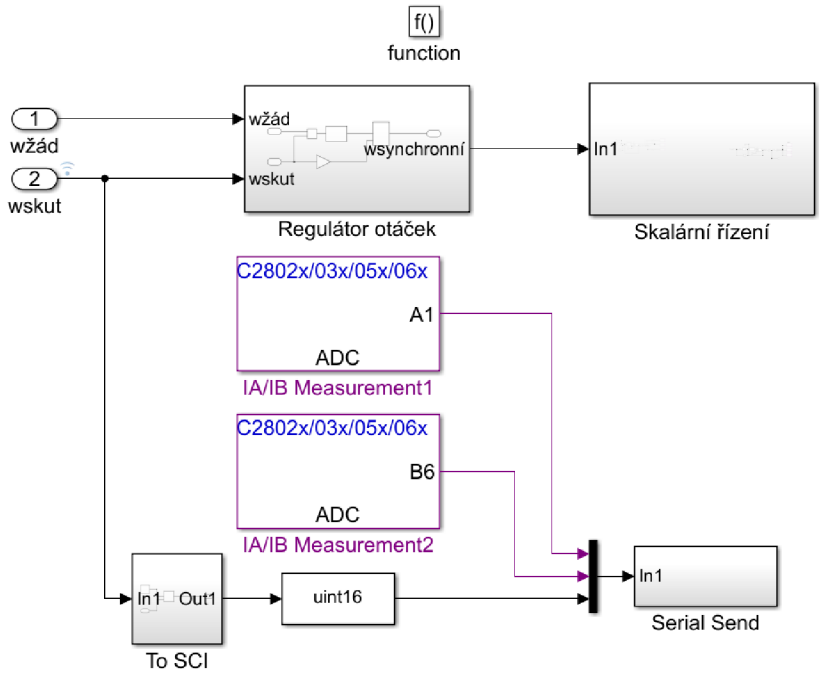
D Schémata skalárního řízení asynchronního motoru v uzavřené smyčce

D.1 Hlavní program



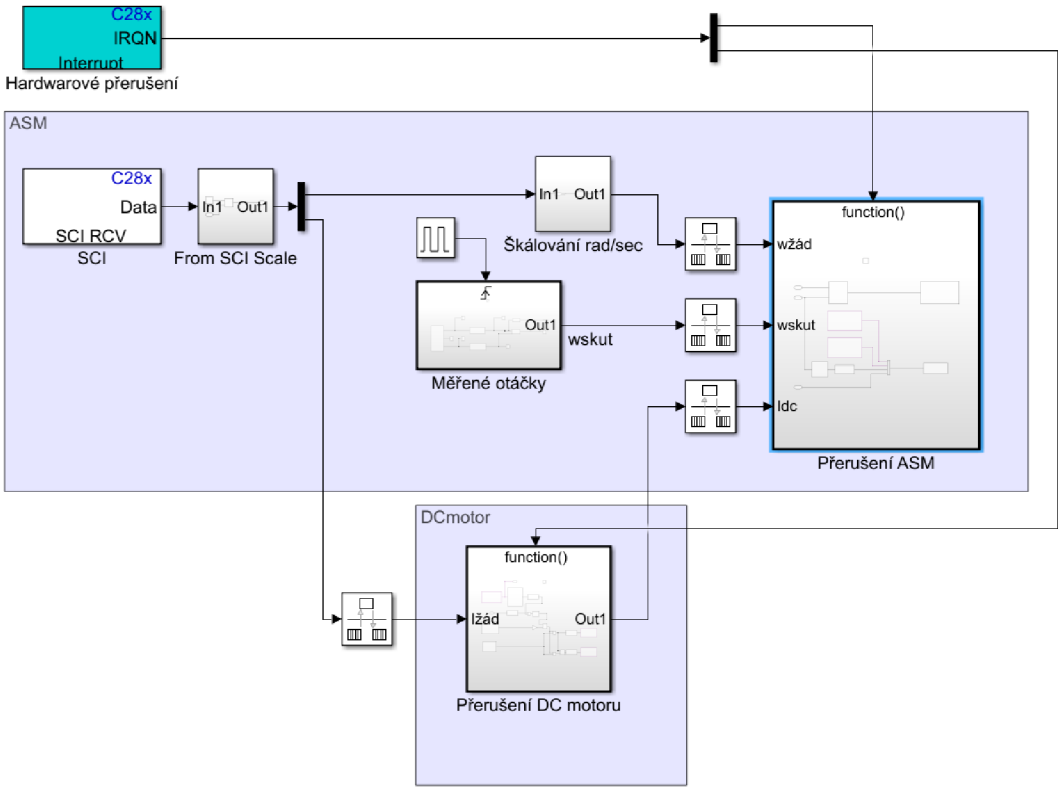
Obr. D.1: Schéma celkového programu

D.2 Subsystém přerušení



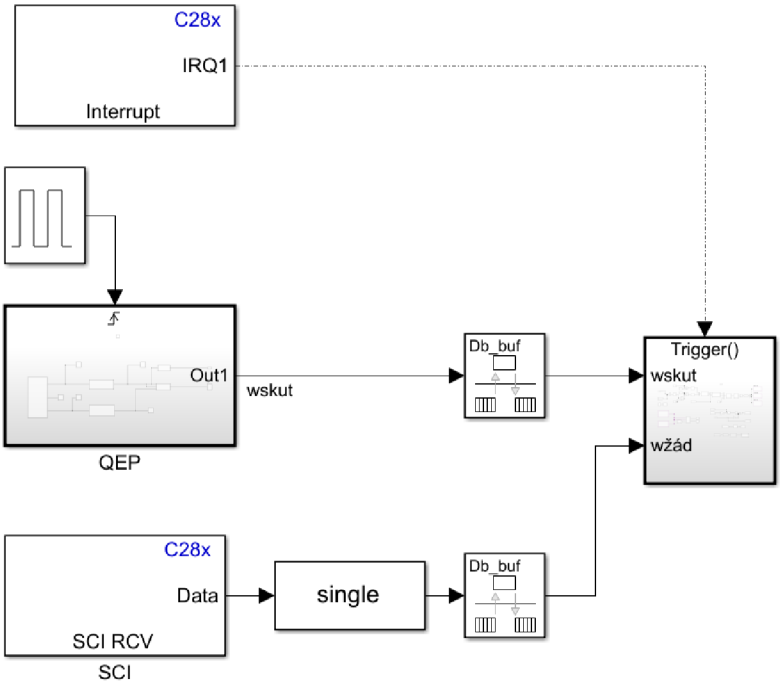
Obr. D.2: Subsystém přerušení

E Schémata zatěžování Asynchronního motoru DC motorem

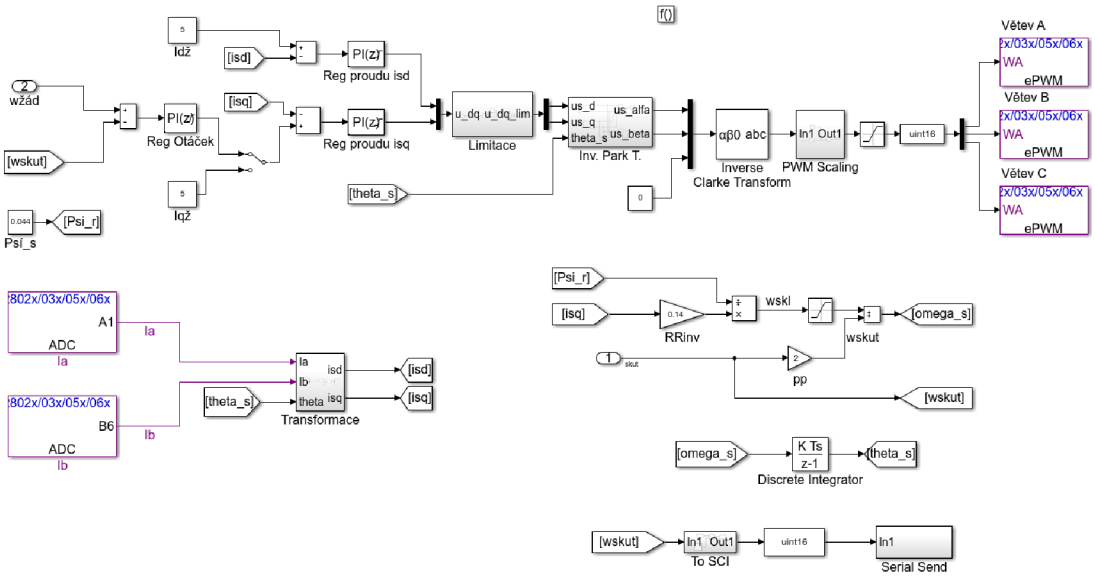


Obr. E.1: Hlavní program

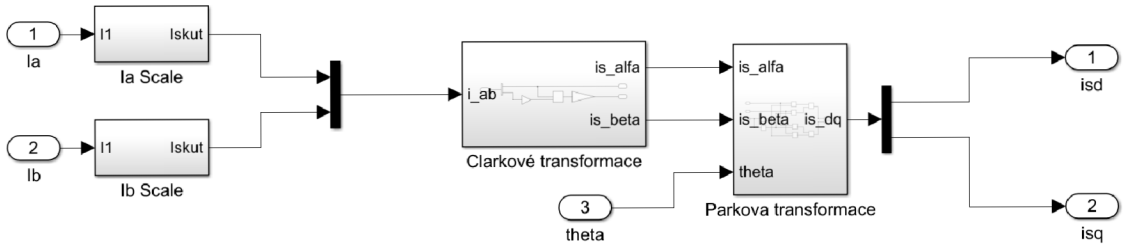
F Schémata vektorového řízení



Obr. F.1: Model vektorového řízení

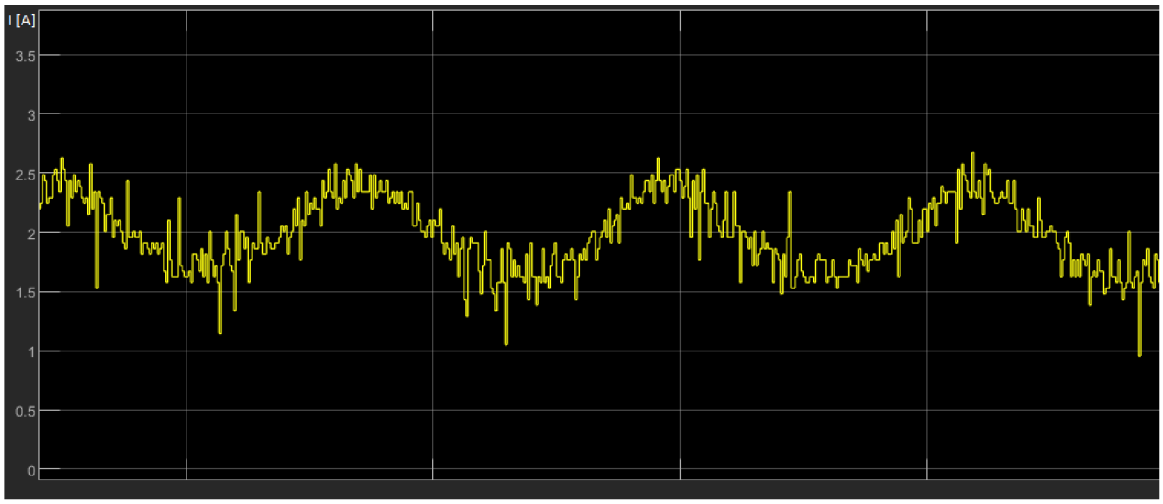


Obr. F.2: Subsystém přerušení vektorového řízení

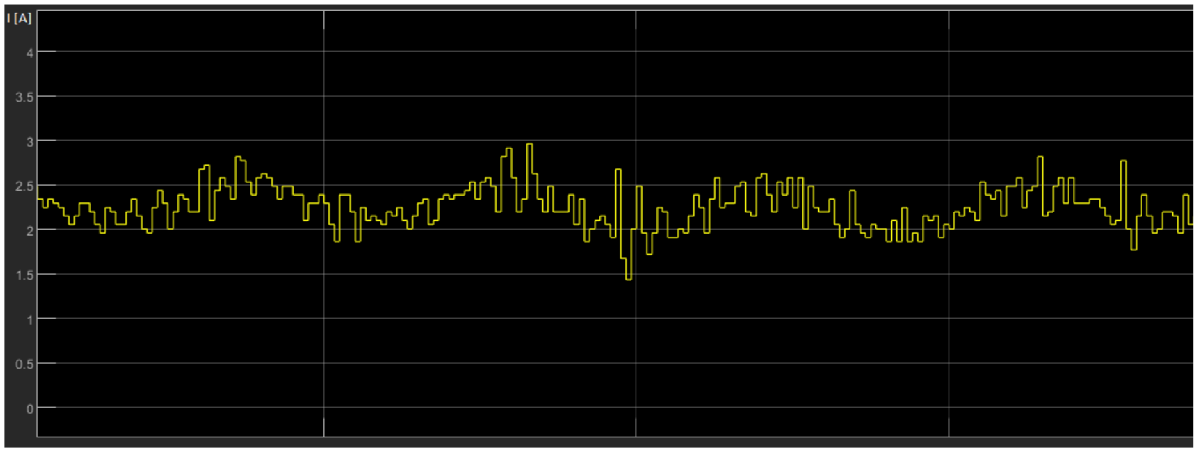


Obr. F.3: Transformace statorových proudů do dq os

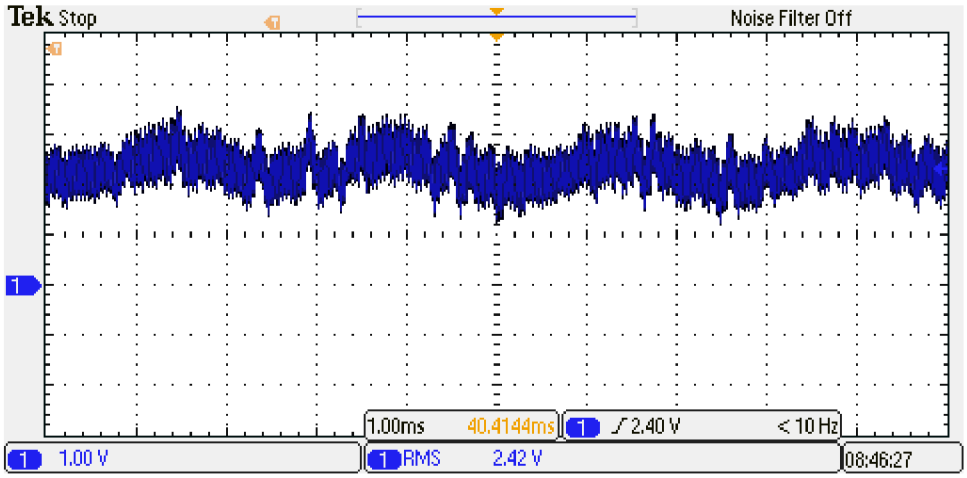
G Měření DC motoru



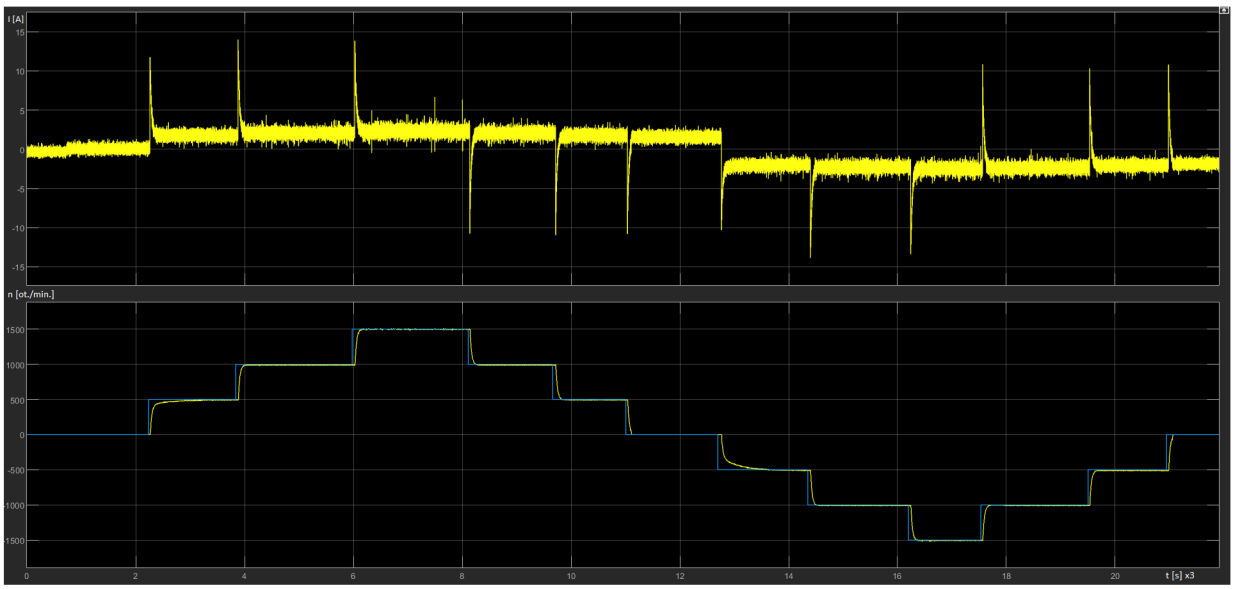
Obr. G.1: Naměřený proud DC motoru v Simulinku - regulátor proudu vypočítané hodnoty.



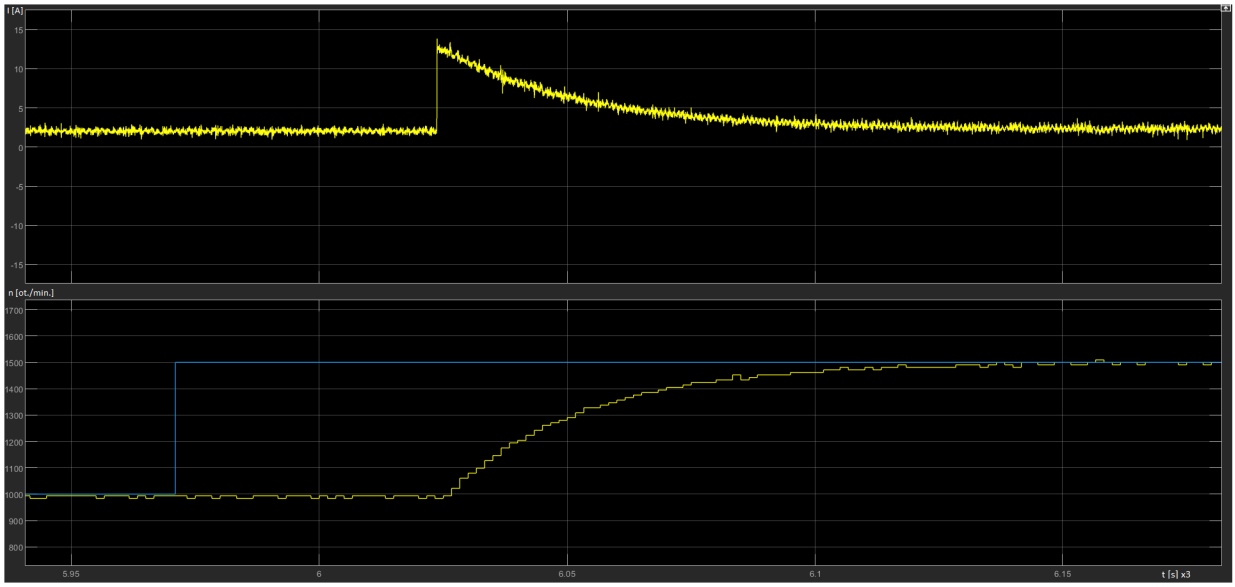
Obr. G.2: Naměřený proud DC motoru v Simulinku - regulátor proudu nové hodnoty.



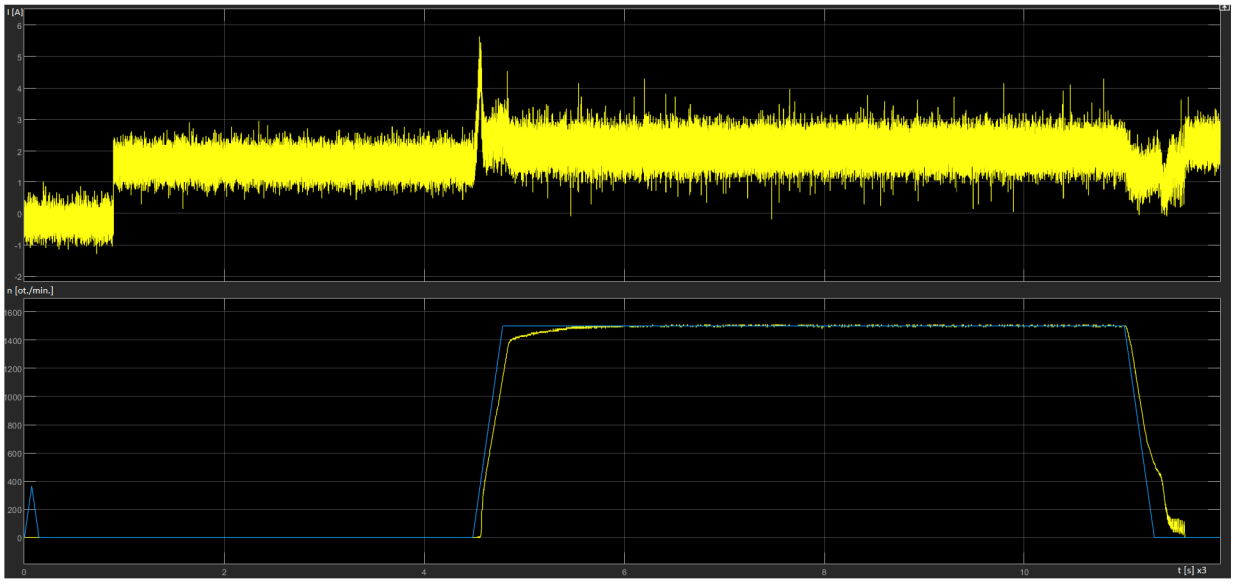
Obr. G.3: Naměřený proud DC motoru v osciloskopu



Obr. G.4: Regulace otáček DC motoru



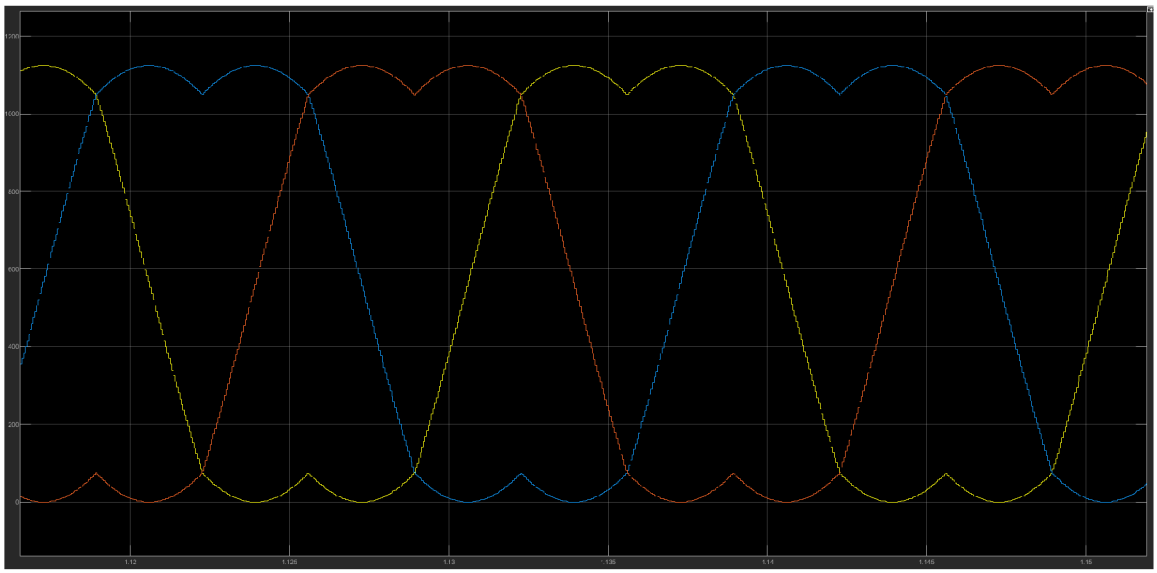
Obr. G.5: Detail regulace



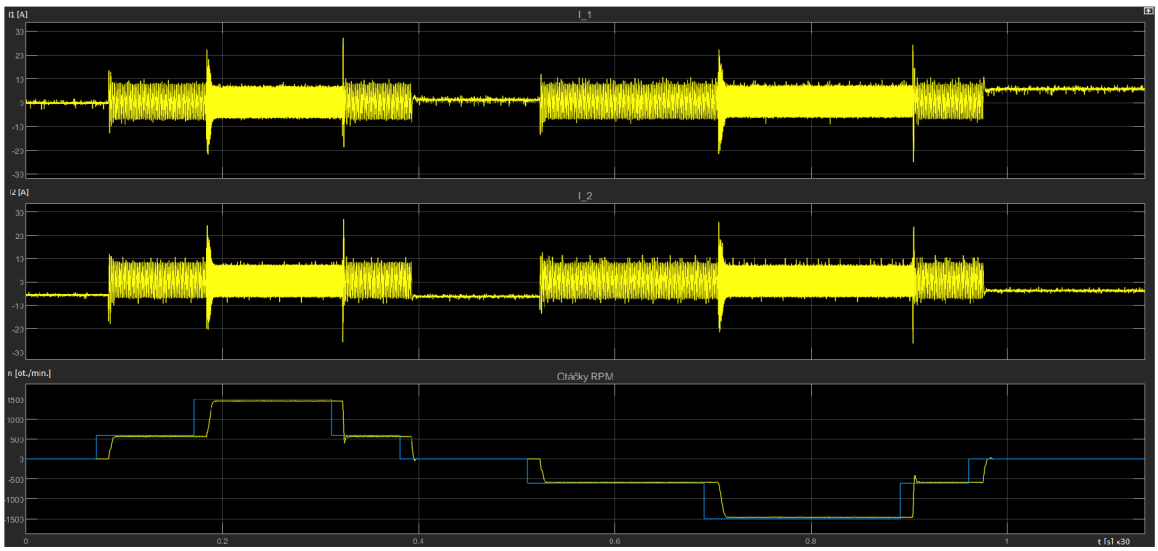
Obr. G.6: Rozběh motoru po rampě na otáčky 1500 ot./min.

H Měření Asynchronního motoru

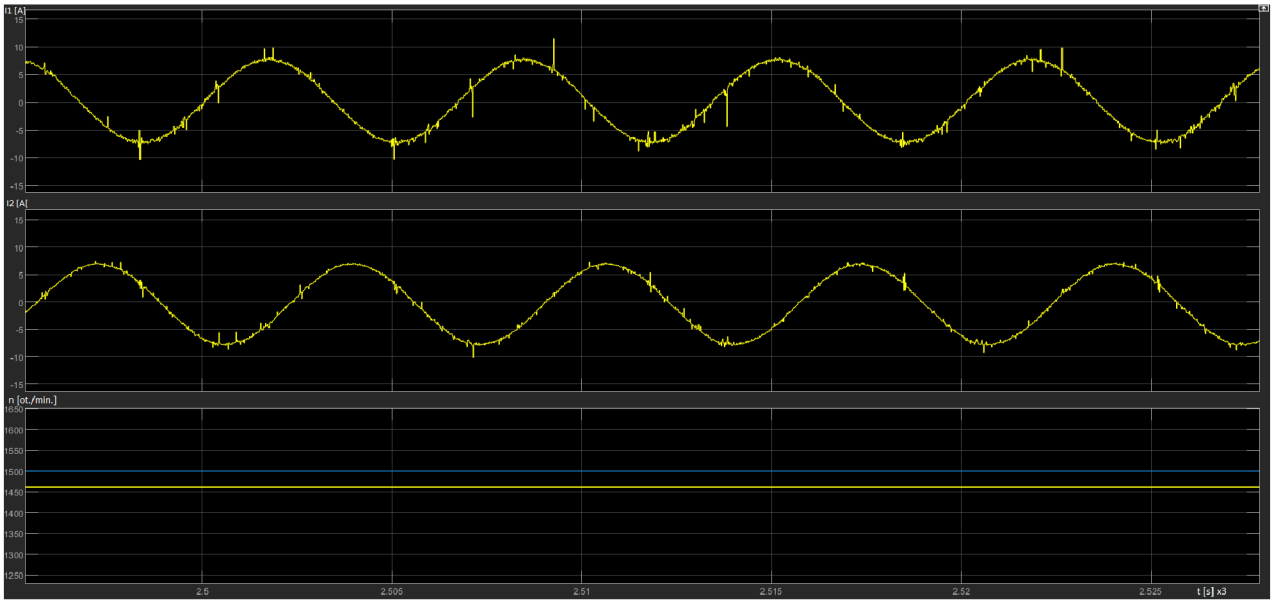
H.1 Skalární řízení v otevřené smyčce



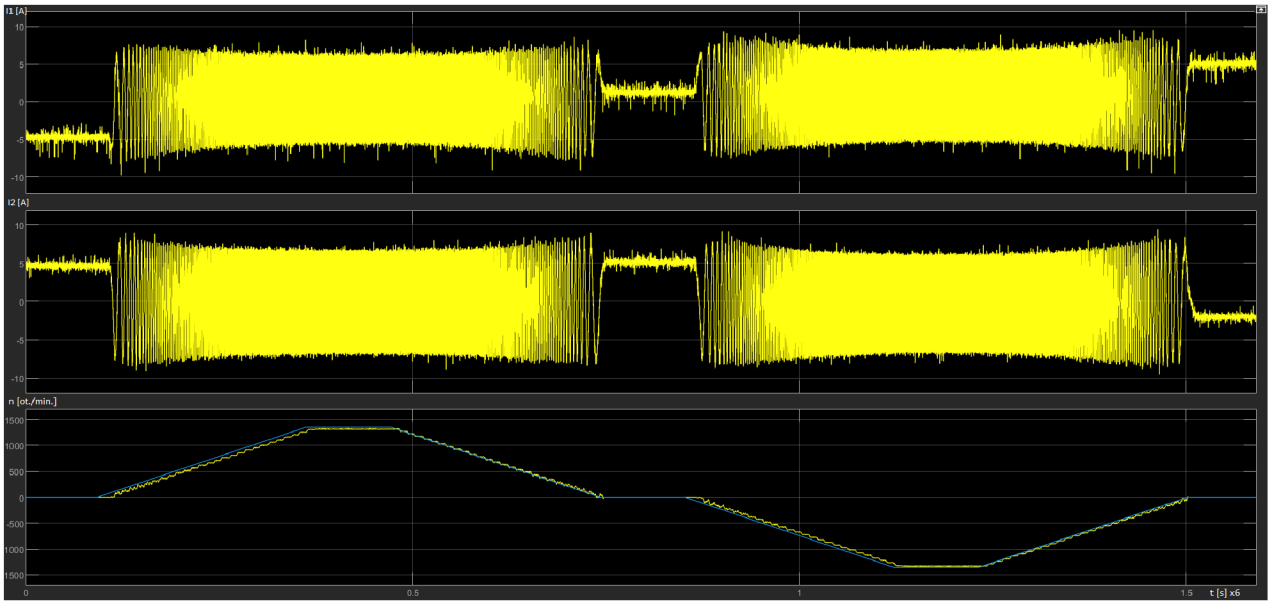
Obr. H.1: Tvorba PWM skalárního řízení



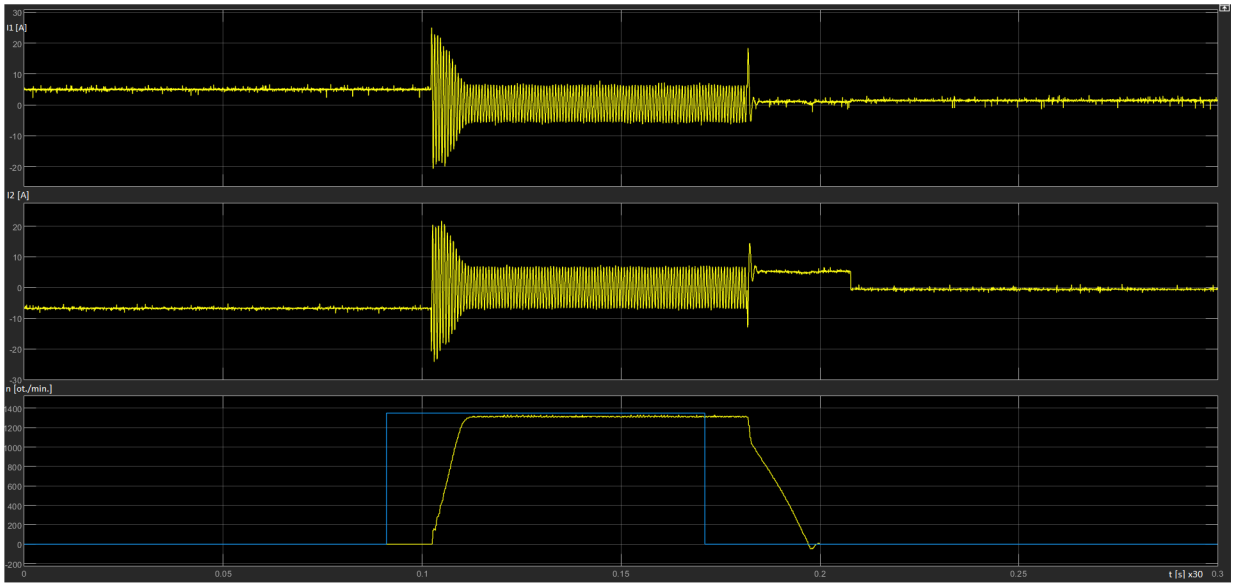
Obr. H.2: Řízení otáček v otevřené smyčce



Obr. H.3: Detail otáček a proudu

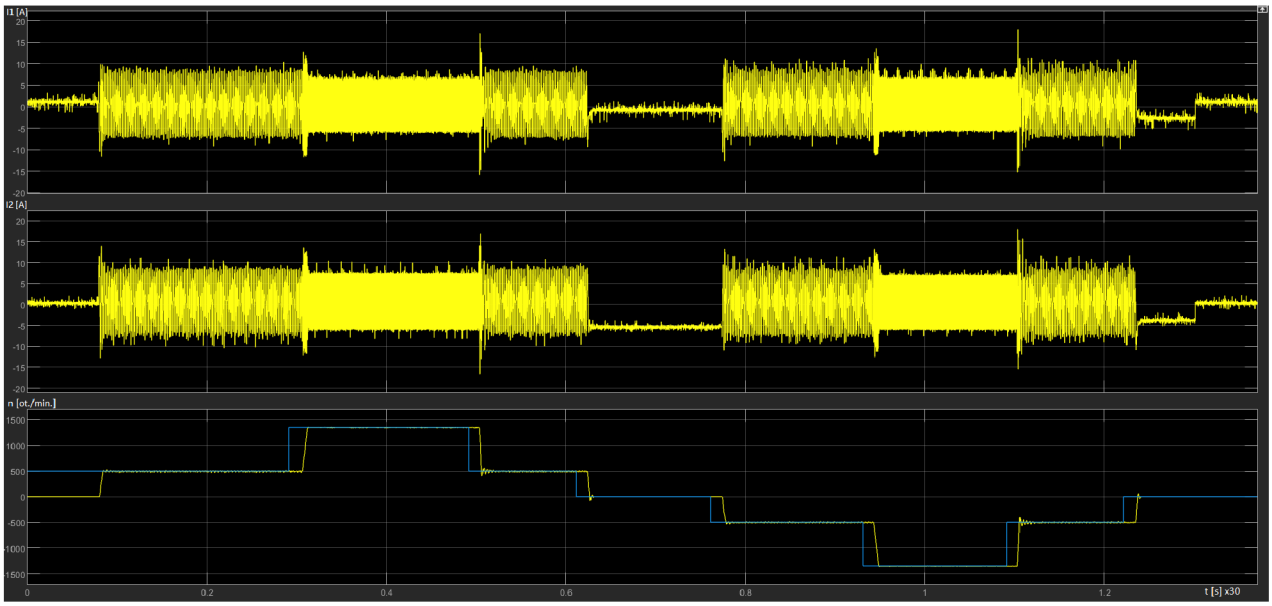


Obr. H.4: Plynulý rozběh pomocí rampy

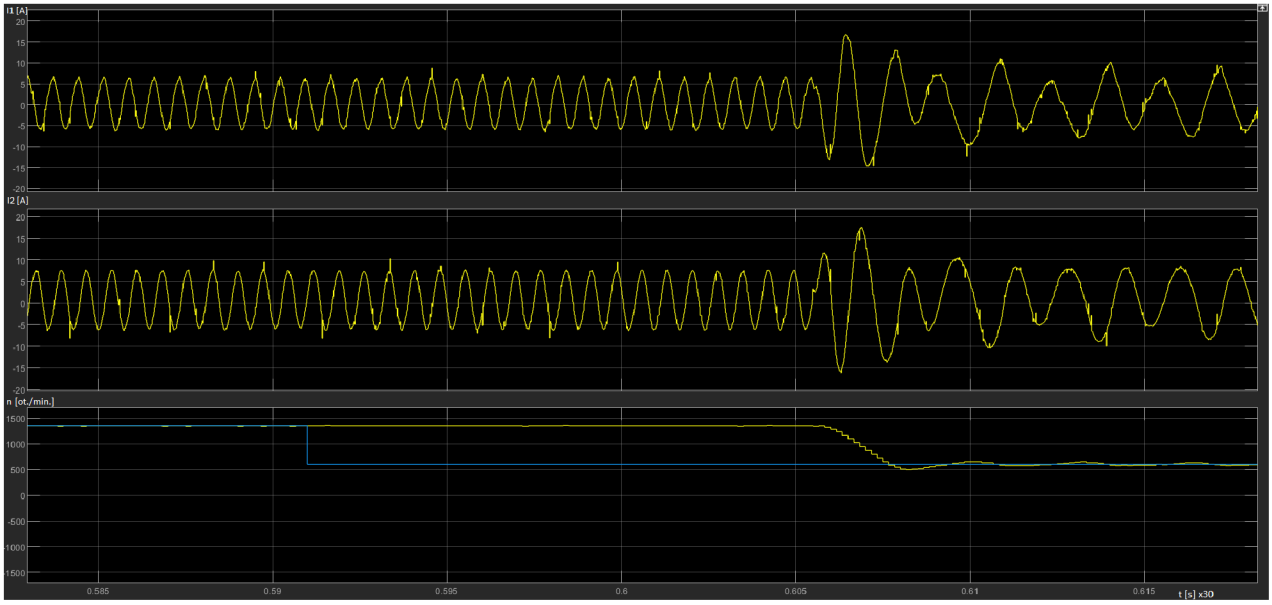


Obr. H.5: Rozběhový proud bez rampy

H.2 Skalární řízení v uzavřené smyčce s čidlem otáček

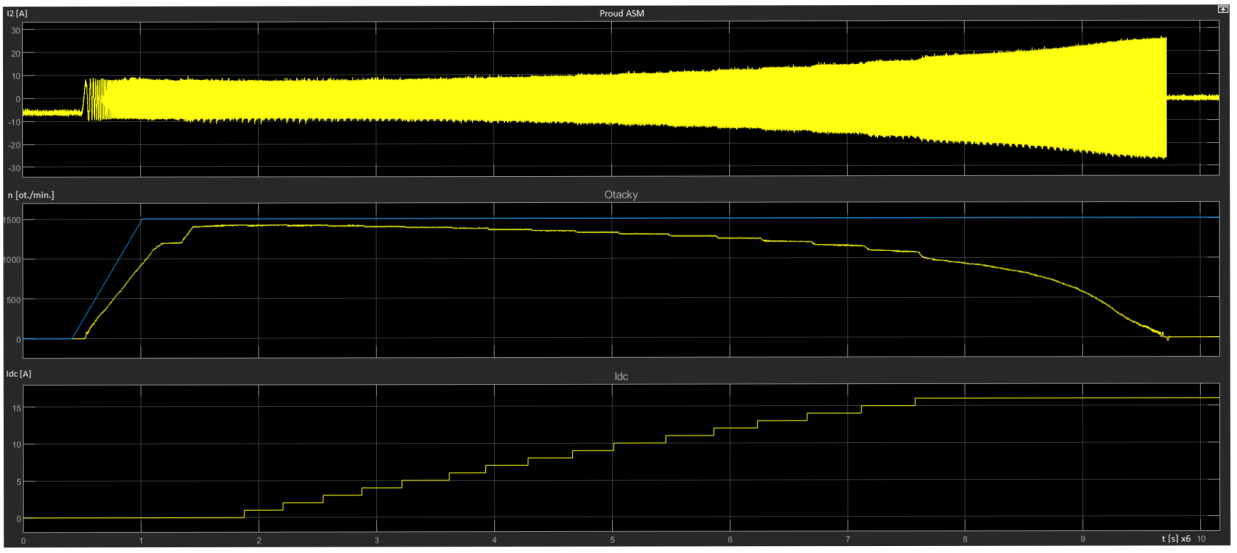


Obr. H.6: Regulace otáček v uzavřené smyčce



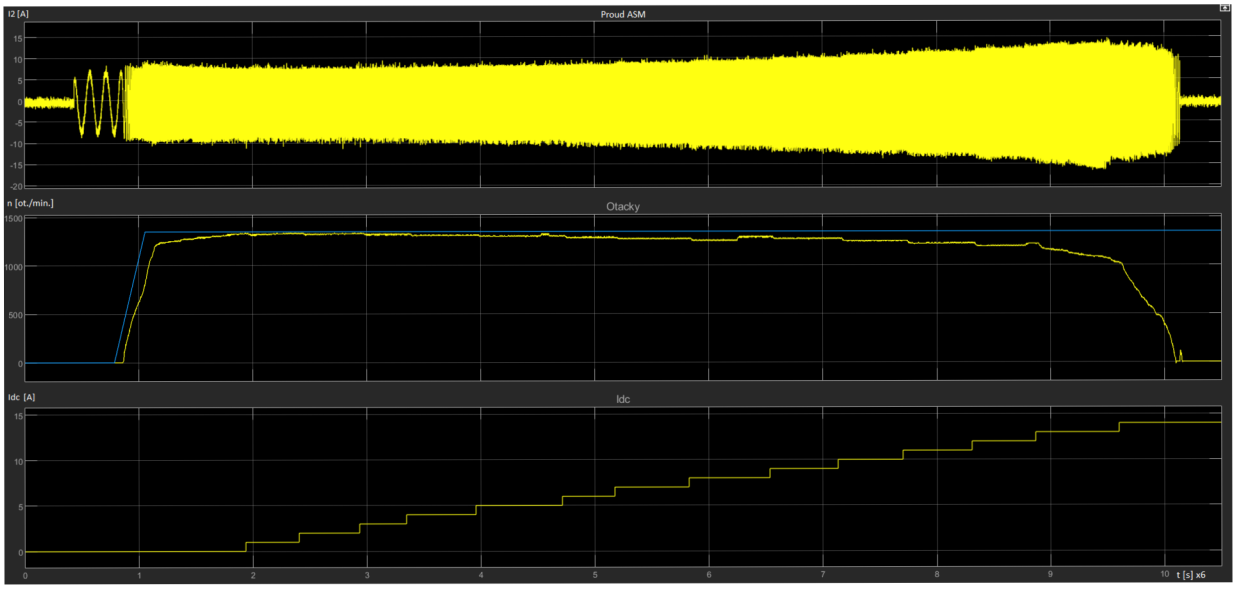
Obr. H.7: Detail regulace

H.3 Zatěžování s otevřenou smyčkou



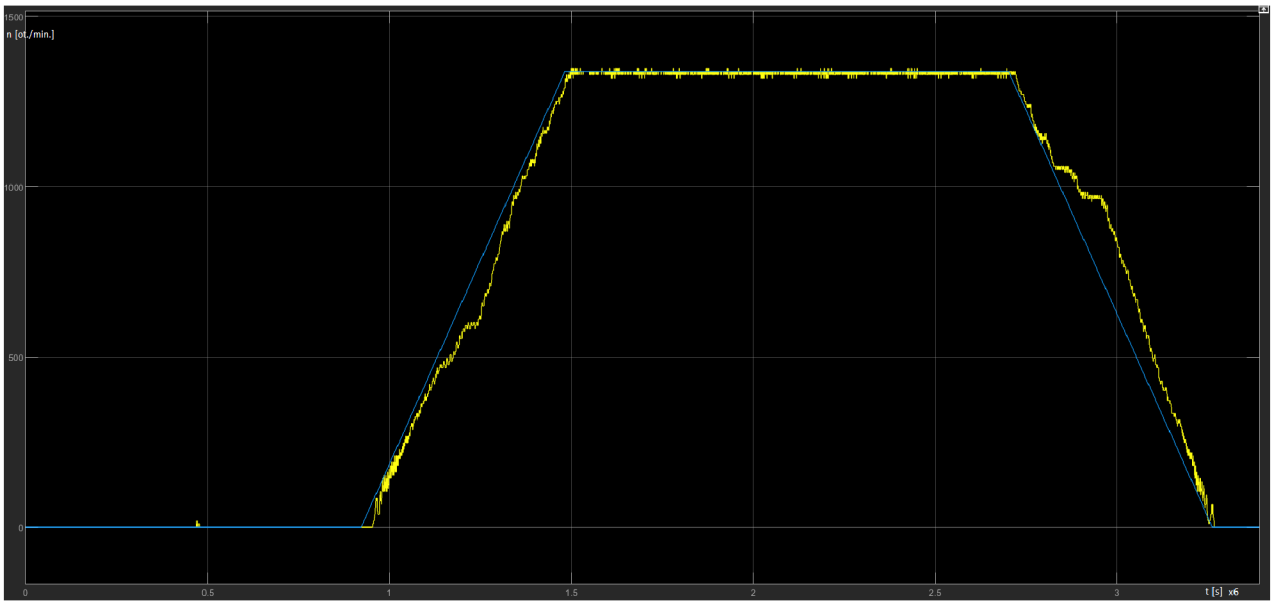
Obr. H.8: Zatěžování při řízení v otevřené smyčce

H.4 Zatěžování s uzavřenou smyčkou s čidlem otáček



Obr. H.9: Zatěžování při řízení v uzavřené smyčce s čidlem otáček

H.5 Vektorové řízení



Obr. H.10: Rozběh vektorového řízená na jmenovité otáčky po rampě

I Přílohy na CD

Eagle

- Redukce_LlaunchPad_gerberdata.zip
- Redukce_LaunchPad.sch
- Redukce_LaunchPad.brd

Simulink

- **ASM**
  - ASMotevrenasmycka
    - asm_otevrena_smycka.slx
    - serial_host.slx
  - ASMuzavrenasmycka
    - asm_regulator.slx
    - serial_host.slx
  - ASMvektor
    - asm_rot_tok_kvazi.slx
    - serial_host.slx
  - ASMzatezovaniotevrenasmycka
    - asm_zatezovani_otevrena_smycka.slx
    - serial_host.slx
  - ASMzatezovaniuzavrenasmycka
    - asm_zatezovani_uzavrena_smycka.slx
    - serial_host.slx
- **DC**
  - DC_motor.slx
  - DCmotorparam.m
  - serial_host.slx

Pozn.: Veškeré modely v Simulinku byly zpracovány pomocí studentské verze Matlab R2019b. Modely asm_otevrena_smycka.slx, asm_regulator.slx, asm_rot_tok_kvazi.slx, asm_zatezovani_otevrena_smycka.slx, asm_zatezovani_uzavrena_smycka.slx a DC_motor.slx obsahují subsystém *Serial Send* převzatý z example *Permanent Magnet Synchronous Motor Field-Oriented Control* [21] z verze Matlab R2019b.