ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA ELEKTROENERGETIKY A EKOLOGIE

DIPLOMOVÁ PRÁCE

Prototypování aplikací v prostředí Matlab-Simulink

Jitka Píchová **2015**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jitka PÍCHOVÁ**
Osobní číslo:	**E12N0021K**
Studijní program:	**N2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Název tématu:	**Prototypování aplikací v prostředí Matlab-Simulink**
Zadávající katedra:	**Katedra elektroenergetiky a ekologie**

Zásady pro vypracování:

Práce se zabývá prototypováním aplikací reálného času pro embedded systémy v prostředí Matlab-Simulink.

1. Popište základní metodiku prototypování aplikací v prostředí Matlab-Simulink.
2. Navrhněte algoritmus řízení stejnosměrného motoru s permanentními magnety.
3. Vytvořte simulační model algoritmu řízení a proveďte model in the loop verifikaci.
4. Pomocí prostředků pro automatické generování kódu vygenerujte řídící software a otestujte technikami software in the loop a processor in the loop.
5. Závěrem zhodnoťte efektivitu výsledného kódu a popište metodiku vývoje aplikace.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Firemní literatura firmy Mathworks**

Vedoucí diplomové práce: **Ing. Jakub Talla, Ph.D.**
Regionální inovační centrum elektrotechniky

Datum zadání diplomové práce: **15. října 2014**

Termín odevzdání diplomové práce: **11. května 2015**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Ing. Karel Noháč, Ph.D.
vedoucí katedry

V Plzni dne 15. října 2014

Abstrakt

Předkládaná diplomová práce je zaměřena na problematiku prototypování aplikací reálného času pro embedded systémy v prostředí MATLAB – Simulink. První část této práce je věnována historii MATLABu a současnému stavu oboru. Dále se práce zabývá popisem metodiky vývoje SW a stručným popisem metod X In the Loop pro řízení stejnosměrného motoru s permanentními magnety. Praktická část je věnovaná návrhu modelu pro řízení tohoto motoru a implementaci metod MIL, SIL, PIL a HIL. Pro metodu PIL a HIL je využit vývojový kit eZdsp™ F2812 s mikrokontrolérem z rodiny C2000 - TMS320F2812 od firmy Texas Instruments.

Klíčová slova

Stejnosměrný motor s permanentními magnety, V-model, verifikace a validace, regulace a řízení, Model In the Loop, Software In the Loop, Procesor In the Loop, Hardware In the Loop.

Abstract

The master thesis is focused on rapid prototyping of real-time applications for embedded systems in MATLAB-Simulink. The first part of the thesis deals with history of MATLAB and presents the actual state of this field. There is a description of SW development methods and a brief description of X In the Loop methods used for control of DC motor with permanent magnets. The practical part is aimed to describe model based design for control of this motor and to implement MIL, SIL, PIL and HIL methods. PIL and HIL methods are implemented on development kit eZdsp™ F2812 with microcontroller C2000 - TMS320F2812 by Texas Instruments.

Keywords

DC motor with permanent magnets, V-model, verification and validation, regulation and controlling, Model In the Loop, Software In the Loop, Procesor In the Loop, Hardware In the Loop.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, který je použitý při řešení této diplomové práce, je legální.

.....

podpis

V Plzni dne 11.5.2015 Jitka Píchová

Poděkování

Tímto bych ráda poděkovala vedoucímu diplomové práce Ing. Jakubovi Tallovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

Obsah

Seznam symbolů a zkratek

dB	decibel
e [-]	regulační odchylka
$F_{(p)}$	přenosová funkce
$F_{IM(p)}$	přenos proudu v kotvě na moment stroje
$F_{UI(p)}$	přenos svorkového napětí kotvy na proud v kotvě
$F_{M\omega(p)}$	přenos akceleračního momentu na otáčky
f_{PWM} [Hz]	frekvence Pulse Width Modulation [Herz]
I [A]	elektrický proud [Ampér]
I_a [A]	proud kotvy
I_n [A]	jmenovitý proud
J [kgm^2]	moment setrvačnosti rotoru [kilogram krát metr^2]
K_i [-]	integrační konstanta (zesílení)
K_{REG} [-]	regulační konstanta (zesílení)
k [-]	konstanta stroje
L_a [H]	indukčnost kotvy [Henry]
M [Nm]	moment stroje na hřídeli [Newtonmetr]
M_{akcel} [Nm]	akcelerační moment
Mz [Nm]	zátěžový moment
p [-]	Laplaceův operátor
R_a [Ω]	odpor kotvy
$R_{(p)}$	vstupní veličina
$Y_{(p)}$	výstupní veličina
T_D [s]	dopravní zpoždění
T_{PWM} [s]	perioda Pulse Width Modulation
U [V]	elektrické napětí [Volt]
U_i [V]	indukované napětí v kotvě
U_n [V]	jmenovité napětí
$U_{\check{R}}$ [V]	řídicí napětí
X_{FX}	zápis numerických hodnot ve formátu *fixed point*
X_{FP}	zápis numerických hodnot ve formátu *floating point*

Φ [Wb]	magnetický indukční tok [Weber]
φ_b [°]	fázová bezpečnost
ω [rads^{-1}]	úhlová rychlost stroje [radián krát sekunda^{-1}]
ω_n [rpm]	jmenovité otáčky [Revolutions Per Minutes]
τ_{REG} [-]	časová konstanta regulátoru

A/D	Analog / Digital
ADC	Analog Digital Converter
AIN	Analog Input
ALU	Arithmetic-Logic Unit
apod.	a podobně
atd.	a tak dále
ASIC	Application Specific Integrated Circuit
BIO	Binary Input / Output
CAD	Computer Aided Design
CAN	Controller Area Network
cca	circa
D/A	Digital / Analog
DLL	Dynamic Link Library
dSPACE	digital Signal Processing And Control Engineering, německá softwarová firma
EDA	Electronic Design Automation
FP	Floating Point
FPGA	Field Programmable Gate Array
FX	FiXed point
FXU	FiXed point Unit
HDL	Hardware Description Language
HIL	Hardware In the Loop
HW	Hardware
I/O	Input / Output
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Enginners

IGBT	Insulated Gate Bipolar Transistor
ISO	International Organization for Standartization
JTAG	Joint Test Action Group
LabVIEW	Laboratory Virtual Instrumentation Engineering Workbench
MATLAB	MATrix LABoratory
METES	Methodology Evaluation Systém
MEX	MATLAB EXecutables
MIL	Model In the Loop
např.	například
NI	National Instruments
PCI	Peripheral Component Interconnect
PI	Proporcionálně Integrační
PIL	Processor In the Loop
PM	Permanentní Magnety
PSpice	Personal SPICE
PWM	Pulse Width Modulation
QEP	Quadrature-Encoder Pulse
RAD	Rapid Application Development
RAM	Random Access Memory
resp.	respektive
rpm	Revolutions Per Minutes
RT	Real Time
SCI	Serial Communication Interface
SIL	Software In the Loop
SPI	Serial Peripherals Interface
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Synchronous Random Access Memory
ss	stejnosměrný
SW	Software
TÜV Süd	Technischer Überwachungsverein Süd, technické kontrolní sdružení na jihu Německa

tj.	to je
tzn.	to znamená
tzv.	tak zvaný
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
USB	Universal Serial Bus
WYSIWYG	What You See Is What You Get
Z	množina celých čísel (německy *čísla – die Zahlen*)

1 Úvod

1.1 Pohled do historie a mezioborové souvislosti

Počínaje vynálezem integrovaného obvodu[1] a jeho významným rozšířením v průběhu šedesátých a sedmdesátých let minulého století značně narostla komplexnost a složitost elektrických, resp. elektronických systému, a tedy i metodika jejich návrhu si vyžádala změnu přístupu a vývoj podpůrných prostředků. Jako přiklad takovýchto nástrojů nechť poslouží program SPICE, který byl vyvinut v akademickém prostředí v sedmdesátých létech minulého století a jenž sloužil a v různých podobách slouží dodnes pro modelování, simulaci a analýzu původně analogových obvodů, dnes také digitální logiky (např. známá implementace PSpice, jež je aktuálně součástí programového balíčku firmy Cadence Design Systems). Tento program a řada dalších patří do kategorie označované jako EDA (Electronic Design Automation), tedy nástroje pro automatizaci návrhu elektronických systémů spadající do skupiny CAD (Computer Aided Design) nástrojů. V současné době tyto programy pokrývají kompletní životní cyklus vývoje elektronické komponenty či systému od návrhu, přes simulaci, analýzu a ověření návrhu až po přípravu podkladů pro výrobní proces.

Paralelu tohoto vývoje samozřejmě nacházíme i v oblasti softwarových aplikací pro elektronické systémy řídící elektrické, mechatronické a další soustavy. Napříč historií a cílovým použitím existuje mnoho přístupů a metod [21], [22] programování řídicích, popř. vestavěných (anglicky *embedded*) systémů. Jedno ze základních rozdělení definuje tři paradigmata programovacích jazyků a to:

- strojový kód, resp. assembler
- procedurální jazyky
- objektově orientované programování

Při vývoji aplikací mimo svět vestavěných řídicích jednotek má vývojář v mnoha případech dostatek výpočetního výkonu, datové a programové paměti, dále zde většinou nejsou na aplikace kladeny požadavky na zpracování v reálném čase, tedy ve striktních časových intervalech. Tyto faktory krom jiných umožňují nasazení aplikací, které využívají mnoho

[1] Jack Kilby, jehož iniciály jsou užity v označení jednoho typu klopných obvodů, jako zaměstnanec firmy Texas Instruments představil první funkční vzorek integrovaného obvodu již v roce 1958, za což byl oceněn v roce 2000 Nobelovou cenou.

specializovaných knihoven funkcí s vlastním rozhraním. Tím však narůstá komplexnost softwaru, zvyšuje se počet volání funkcí knihoven a s tím související zvyšující se paměťové nároky a mnoho dalších.

V posledních minimálně dvaceti letech se při programování řídicích aplikací a operačních systému reálného času běžících na pestré škále procesorů od 8-bitových po 32-bitové (mnohdy již 64-bitové) platformy ve většině případů nasazení využívá kombinace výše zmíněných paradigmat. Mezi nejpoužívanější programovací jazyky v této oblasti použití [3] tedy patří jazyky C a C++. Tyto systémové jazyky umožňují vývojáři nízkoúrovňový přístup přímo k hardwaru cílové platformy, který je naprosto nezbytný např. při vývoji operačního systému. Na druhé straně vývojář aplikační vrstvy se soustředí především na efektivní implementaci řídicích algoritmů. Pochopitelně i tento vývojář musí mít povědomí o dostupnosti a možnostech systémových prostředků konkrétní hardwarové platformy, je však žádoucí, aby si mohl co nejjednodušší cestou požádat o tyto prostředky a snadno provést jejich konfiguraci bez nutnosti detailní znalosti konkrétní platformy, např. na úrovni registrů jednotlivých periférií atd.

Vývojář řídící aplikace se zaměřuje zejména na algoritmy aplikační vrstvy, tedy na měření vstupních hodnot, jejich zpracování a následně na výpočet a nastavení adekvátních výstupních hodnot akčních veličin. Je časté, že vývojářem aplikační vrstvy je odborník v oboru konkrétní řízené soustavy či technologického procesu, jehož cílem je rychlá a efektivní implementace řídicí aplikace bez nutnosti nabývání detailních znalostí nižších systémových vrstev softwaru, konfiguraci hardwaru apod.

V devadesátých letech minulého století na razantní pokrok v oblasti operačních systému s grafickým uživatelským rozhraním a jejich rozšíření mezi uživateli adekvátně zareagovali i výrobci vývojových prostředků pro aplikace. Jedním z nejvýznamnějších hráčů tohoto trhu byla firma Borland, která v osmdesátých letech nabízela dnes již legendární vývojová prostředí Turbo Pascal a Turbo C. Její odpovědí na trend devadesátých let byla a nyní jako produkty firmy Embarcadero Technologies v různých formách stále jsou integrovaná vývojová prostředí (IDE) Delphi pro jazyk Object Pascal a C++ Builder pro jazyky C/C++, která umožňují rychlý vývoj (RAD) prostřednictvím vizuálního návrh (drag & drop funkce jako součást WYSIWYG editoru) grafického uživatelského rozhraní. Algoritmy dané aplikace jsou samozřejmě nadále vytvářeny vývojářem zápisem v konkrétním programovacím jazyce. Nicméně grafická forma aplikace, vlastnosti grafických komponent či akce na určité události nad těmito objekty mohou být poměrně jednoduše definovány v rámci grafického IDE, které poté vygeneruje odpovídající kód konkrétního jazyka.

Od přelomu padesátých a šedesátých let minulého století až po současnost se pro vědecké a inženýrské numerické výpočty používá imperativní programovací jazyk Fortran. Po tuto dobu došlo k rozšíření jeho použití na různé oblasti přírodních a technických věd, nabízí tedy uživateli řadu matematických a dalších komerčně nebo volně dostupných knihoven. Jedná se o jazyk kompilovaný, což samozřejmě zaručuje vyšší rychlost zpracování programu, ale také jistou závislost na architektuře platformy, na které je výsledný kód vykonáván. Fortran dnes nabízí elementární datové typy, standardní konstrukce pro řízení běhu programu a příkazy jako řada dalších procedurálních jazyků (např. C). Zápis výrazů v jazyku Fortran je však stále vzdálený matematickému zápisu formulí a vyžaduje od uživatele programátorský přístup, což zvláště při řešení relativně jednoduchých dílčích úloh např. na počátku vývojového cyklu může být omezujícím faktorem.

1.2 MATLAB

V sedmdesátých létech minulého století Cleve Moler z University of New Mexico navrhl a vytvořil knihovnu LINPACK určenou pro numerické výpočty úloh lineární algebry. Tato knihovna mohla být využívána bez znalosti programování v jazyku Fortran a stále se základem pro nové prostředí a jazyk určený pro numerické výpočty – MATLAB. Jazyk MATLAB je interpretovaný, což může být na úkor rychlosti zpracování programu, nicméně není třeba zdlouhavé kompilace při každé změně programu. Dále se jedná o jazyk slabě typovaný s dynamickou typovou kontrolou. Všechny proměnné MATLABu jsou definovány jako pole, resp. n-rozměrné vektory, dále umožňuje volání funkcí vytvořených v jazycích Fortran a C. MATLAB disponuje velkým množstvím balíčku specializovaných funkcí, tzv. toolboxes pro různé oblasti jako např. řídicí systémy, zpracování signálů, statistiku a optimalizace. Výsledky výpočtu lze zobrazit v dvourozměrných či třírozměrných grafech, generovat a upravovat soubory obrazových dat. Verze 1.0 prostředí MATLAB byla vydána v roce 1984 pod hlavičkou firmy MathWorks. Přibližně v této době vznikly další výpočetní nástroje založené na různých konceptech a určené pro široké spektrum vědeckých a technických výpočtu jako např. Mathematica od firmy Wolfram Research a Maple firmy Maplesoft.

1.3 Simulink

Firma MathWorks záhy připravila grafické prostředí založené na blokových diagramech – Simulink, původně s označením Simulab. Prostřednictvím prostředí Simulink lze provádět analýzu a simulace dynamických systémů, automatické generování kódu a testování nejenom vestavěných elektronických systémů. V Simulinku lze konstruovat systémy a soustavy z existujících knihovních bloků, které realizují funkce a subsystémy širokého spektra technických oborů a vědních oborů, nebo je možné vytvářet vlastní bloky či funkce v rámci tzv. *s-funkce*. Ten může být napsán v jazyce MATLAB, C nebo Fortran. Simulink umožňuje vytváření nejen modelů řídicích systémů, ale i řízených. Implementaci takových systémů lze realizovat např. v mikrokontroléru, kdy je kód jazyka C vygenerován nástrojem, tzv. *toolboxem Embedded Coder*, nebo v FPGA, pro které je syntetizovatelný VHDL nebo Verilog kód vygenerován *toolboxem HDL Coder*.

1.4 LabVIEW

Víceméně paralelně s výše zmíněnými nástroji firma National Instruments (NI) začala vyvíjet LabVIEW, které je oproti těmto výpočetním analytickým nástrojům původně určeno pro ovládání přístrojů (zejména firmy NI), sběr a zpracování naměřených dat.

Tento nástroj oproti předchozím nabízí odlišnou koncepcí v komunikaci s uživatelem. Umožňuje vizuální návrh systému prostřednictvím grafických funkčních bloků, tzv. *virtual instruments*, vzájemně spojených, čímž je definován tok dat v systému. Popis systému je proveden v grafickém jazyce G. Program vytvořený v jazyku G se označuje právě jako *virtual instrument*, který může sestávat z podprogramů, tedy dílčích *virtual instruments*. Zpracování takového programu nemusí probíhat lineárně (sekvenčně), jednotlivé větve bloků v rámci systému mohou být provedeny paralelně.

1.5 Cíle práce

Text této práce se soustředí právě na využití prostředí MATLAB - Simulink pro účely rychlého prototypování SW aplikace, které jsou spustitelné na mikrokotrolérech vestavěných elektronických systémů.

Práce popisuje principy a realizaci metod používaných v různých fázích vývojového projektu, a to technik Model In the Loop (MIL), Software In the Loop (SIL), Processor In the Loop (PIL) a Hardware In the Loop (HIL). Prostřednictvím prostředí MATLAB - Simulink budou realizovány a otestovány metody MIL, SIL a PIL. V praktické části bude uvedenými metodami simulováno a implementováno řízení stejnosměrného motoru s permanentními

magnety. Pro metodu PIL bude využit vývojový kit eZdsp™ F2812 [14] firmy Spectrum Digital, který je osazen mikrokontrolérem z rodiny C2000 firmy Texas Instruments – TMS320F2812 [13].

V poslední době se v souvislosti např. s autonomními vozidly nebo se simulacemi pro vývoj elektronických asistenčních systémů ve vozidle hojně vyskytuje pojem Vehicle In the Loop [7]. Popis a nasazení zmíněné metodiky není součástí této práce.

1.5.1 Motivace

V automobilovém průmyslu v posledních dvaceti letech výrazně narostl podíl elektrických zařízení, elektronických jednotek a kabeláže na celkové ceně, hmotnosti a funkcionalitě, tudíž také na bezpečnosti a poruchovosti funkčních komponent vozidla. Tento růst samozřejmě strmě pokračuje.

Mnoho realizovaných funkcí je stále implementováno v jednoúčelových obvodech nebo v složitějších zákaznických obvodech ASIC (Application Specific Integrated Circuit). Mezi takové funkce patří např. řízení koncových výkonových prvků nebo bezpečnostní arbitery (např. externí hlídací obvod živosti, tzv. *watchdog*). Avšak s ohledem na výrobní náklady a cenu výrobku či modifikovatelnost a rozšíření funkcionality výrobku se stále výrazněji uplatnují programovatelné součástky, tedy mikroprocesory, resp. mikrokontroléry a programovatelná logická pole.

Se zvyšujícími se požadavky na funkce realizované softwarem programovatelných součástek úměrně roste komplexita těchto programů, nároky na rychlý proces vývoje, snadnou udržovatelnost kódu, jednoznačnou testovatelnost a reprodukovatelnost testů pro splnění požadavků norem a standardů, a tudíž pro minimalizaci bezpečnostních rizik a odhalení systémových chyb v co možná nejranější fázi vývojového cyklu, čímž dochází k výraznému snížení souvisejících finančních nákladů. Zejména tyto parametry nasměrovaly vývojáře a výrobce nejen v automobilovém průmyslu k modelování systémů, jejich prototypovaní v grafických prostředích s jednoznačnou vazbou na systémové a softwarové požadavky, dále k metodám generování zdrojových kódů pro cílovou platformu.

1.5.2 Současný stav oboru

V předchozích kapitolách byly zmíněny některé z programových nástrojů a balíků, které lze v této době použít při analýze, simulaci, návrhu a testování elektronických řídicích systémů. Mezi další zejména v automobilovém průmyslu hojně používané patří nástroje firmy dSPACE, které nabízí zejména nástroje pro HIL testování a nástroje pro automatické

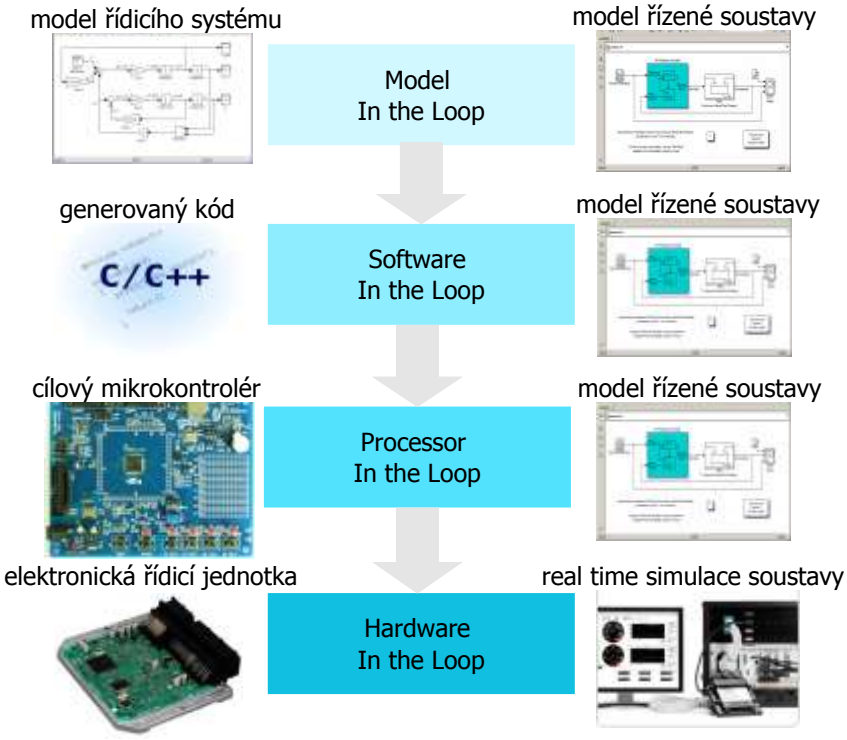
generovaní kódů (*TargetLink*). Právě *TargetLink* je v mnoha případech součástí *tool chain* (použité SW nástroje) společně s návrhovými systémy firmy MathWorks, a to se Simulinkem a StateFlow. *TargetLink* je v podstatě funkčním ekvivalentem nástroje firmy MathWorks. Mimochodem oba nástroje obdržely certifikáty pro vývoj funkcí a systémů se vztahem k bezpečnosti (pro průmyslové – IEC 51608 - či automobilové – ISO 26262 - standardy) od německého technického sdružení TÜV Süd, které je celosvětově uznávanou certifikační autoritou.

2 Teorie simulačních metod typu X In the Loop

2.1 Obecný úvod

Modelování a simulace systému získávají stále významnější roli během celého cyklu SW vývoje. Modelování umožňuje grafickou formou popsat návrh SW, čímž navíc, pokud je model dostatečně popsán, vytváří dokumentaci k návrhu SW. Simulace pak poskytuje možnosti pro testování a optimalizaci takového návrhu před jeho implementací. Důležitá je také možnost modelování řízené soustavy, kdy lze během testování nasimulovat mezní stavy, které jsou na reálné soustavě dosažitelné pouze za speciálních (často nákladných) podmínek nebo jsou dokonce nerealizovatelné (při dodržení nezbytných bezpečnostních předpisů). Vývojář může využít během tvorby řídicího SW metody, které stručně naznačuje Obr. 2.1. Levá část Obr. 2.1 se týká navrhovaného řídicího systému, v pravé části je pak naznačen možný způsob simulace řízené soustavy. Další kapitoly se budou zabývat popisem právě těchto metod a uvedením příkladu užití.

Jak naznačuje Obr. 2.1, vývojář postupuje od modelu řídicího systému a řízené soustavy v metodě MIL přes metody SIL (kód zatím není spouštěn na cílové platformě) a PIL (kód již běží na cílové platformě) pro ověření vygenerovaného kódu v jazyce C až po testování na reálném hardwaru v reálném čase v rámci metody HIL.



Obr. 2.1: Metody typu X In the Loop

2.2 Základní pojmy

2.2.1 Aritmetika v pevné a plovoucí řádové čárce

Na začátku této kapitoly bude určeno názvosloví vycházející z anglického jazyka, neboť se jedná o všeobecně používané výrazy. Navazující kapitoly budou také preferovat anglické označení *fixed point* (FX) pro formát pevné řádové čárky a *floating point* (FP) pro formát plovoucí řádové čárky [17].

Numerické hodnoty uložené ve formátu *fixed point* jsou podmnožinou racionálních čísel, kterou popisuje vztah (2.1):

$$x_{FX} = \frac{a}{b} \tag{2.1}$$

kde

a náleží do množiny celých čísel Z

b náleží do množiny celých čísel Z a zároveň $\neq 0$

Dále pro tuto podmnožinu platí omezující podmínka:

$$b = 2^k \tag{2.2}$$

kde

k náleží do množiny kladných celých čísel Z^+

Jak naznačují vztahy (2.1) a (2.2), převod na formát *fixed point* lze implementovat jako rychlou operaci bitového posunu čísla a doprava o k bitů namísto výpočetně náročné operace dělení. Další v podstatě implicitní podmínkou odkazující na název formátu *fixed point* je zachování stejného počtu binárních cifer v každém reprezentovaném čísle, jinými slovy, všechna čísla mají řádovou binární tečku pevně na stejném místě. Koeficient b bývá v literatuře také označován jako *scaling factor*. Pro označení datového formátu *fixed point* se používá prefix (předpona) *Q* následovaný číslem *m*, které definuje počet bitů celočíselné části (před desetinnou čárkou), a dále číslem *n,* které definuje počet bitů desetinné části (za desetinnou čárkou). Notace *Q0.15* [18] pro 16-bitové procesory deklaruje znaménkový typ bez celočíselné části s 15 bity pro desetinnou část. Pro formát *fixed point* není příliš rozšířena přímá podpora v programovacích jazycích, na druhou stranu výrobci nejen signálových procesorů často implementují na křemíkový čip (do procesorového jádra) vedle aritmeticko-logické jednotky (ALU) také HW jednotku (FXU) pro výpočty ve formátu *fixed point* nebo dodávají matematické knihovny [18] optimalizované pro daný typ procesoru.

Numerické hodnoty z podmnožiny reálných čísel ve formátu *floating point* jsou uložené ve dvou částech, a to v mantise a exponentu:

$$x_{FP} = mb^e \tag{2.3}$$

kde

m mantisa (množina celých čísel *Z*)

b báze, také nazývaná *radix*

e exponent (množina celých čísel *Z*)

Formát (počet bitů rezervovaných pro uložení mantisy a exponentu) uložení hodnot a pravidla implementace operací s hodnotami ve formátu *floating point* jsou specifikována normou IEEE 754 [19]. Podpora formátu *floating point* ze strany programovacích jazyků je oproti podpoře formátu *fixed point* výrazně širší a především standardizovaná díky zmíněné normě IEEE 754. Jazyk C především od verze C99 (mezinárodní standard ISO/IEC 9899:1999) požadavky této normy (konkrétně verze IEEE 754-1985 [20]) podporuje. V jazyce C jsou tak k dispozici datové typy pro práci s reálnými čísly:

Tab. 2.1: Datové typy v souladu s IEEE 754-1985 [20]

Datový typ	**Úroveň přesnosti**	**Rozsah**	**Přesnost**
Float	single precision	od $\pm 1.18\times10^{-38}$ do $\pm 3.4\times10^{38}$	cca 7 desetinných míst
Double	double precision	od $\pm 2.23\times10^{-308}$ do $\pm 1.80\times10^{308}$	cca 15 desetinných míst

2.2.2 Výpočet modelu v Simulinku – solvery

Výpočet stavů systému, který je popsán modelem v Simulinku, v sousledných časových krocích v rámci definovaného časového intervalu se nazývá řešení (*solving*) modelu. Pro různé druhy dynamických systému jsou vhodné odlišné typy komponent Simulinku, tzv. *solvery*. *Solver* určuje časové kvantum simulačního kroku a definuje, která z numerických metod bude použita pro řešení diferenciálních rovnic, které popisují model.

Základní dělení *solverů* [27]:

- s pevným nebo proměnným krokem
- diskrétní nebo spojitý
- explicitní nebo implicitní
- s jedním krokem nebo vícenásobným krokem

Simulace v této práci budou počítány spojitým explicitním *solverem* s pevným krokem – implementovaný funkcí *ode1*, tedy numerickou integrační metodou prvního řádu *Euler*. Díky

zvolenému *solveru* s pevným krokem je zaručeno, že nedojde k nastavení příliš dlouhého kroku, k čemuž může docházet u *solverů* s proměnným krokem. V takovém případě může dojít k časovému ovlivnění (např. zpoždění) při simulaci fyzikálních modelů – jako např. model polovodičového měniče.

Pokud je uživatelem pro model vybrán *solver* s pevným krokem bez dalšího upřesnění, pak Simulink standardně vybere metodu *Bogacki-Shampine* [28]. Pro *solvery* s pevným krokem obecně platí – vyšší řád přináší vyšší přesnost, avšak na úkor vyšší výpočetní náročnosti. Vyšší přesnosti lze také dosáhnout nastavením malého kroku, kdy opět naroste výpočetní náročnost.

2.2.3 Algebraické smyčky

Algebraická smyčka (řešení algebraické rovnice) v modelu existuje, pokud výstup bloku X je přiveden zpět na vstup tohoto bloku X a zároveň v řetězci bloků, který spojuje výstup bloku X s jeho vstupem, není alespoň jeden integrační blok, resp. obecně alespoň jeden blok, který udržuje stavovou proměnnou (kromě integrátoru např. blok zpoždění). Jinými slovy, výstup bloku X závisí (pouze) na hodnotě vstupu a tedy hodnota vstupu přímo ovládá výstupní hodnotu. Mezi takové bloky (*blocks with direct feed-through inputs*) patří:

- *Math Function*
- *State-Space*, pokud je maticový koeficient D je nenulový
- *Transfer Fcn*, pokud jsou čitatel a jmenovatel přenosové funkce stejného řádu
- *Zero-Pole*, pokud má blok stejný počet nul jako pólů

2.2.4 Metodika vývoje SW

Metodika vývoje SW určuje strukturu (praktiky, techniky a nástroje), způsob plánování a řízení (role, zodpovědnosti) procesu vývoje SW. Příslušná metodika definuje fáze projektu, vstupy a výstupy jednotlivých fází, aktivity vykonané v těchto fázích a metody kontroly kvality v konkrétních fázích.

Pro každý obor vývoje SW jsou typicky aplikovány odlišné metodiky. Typ zvolené metodiky závisí na druhu finálního produktu (webové stránky versus sériově vyráběný automobil), na časových možnostech a omezení pro vydání a na způsobu následného udržování produktu, na požadavcích, které vycházejí z použitých norem a standardů, a na mnoho dalších faktorech. Pro výběr vhodné metodiky lze např. použít původní systém hodnocení a výběru metodik METES [8].

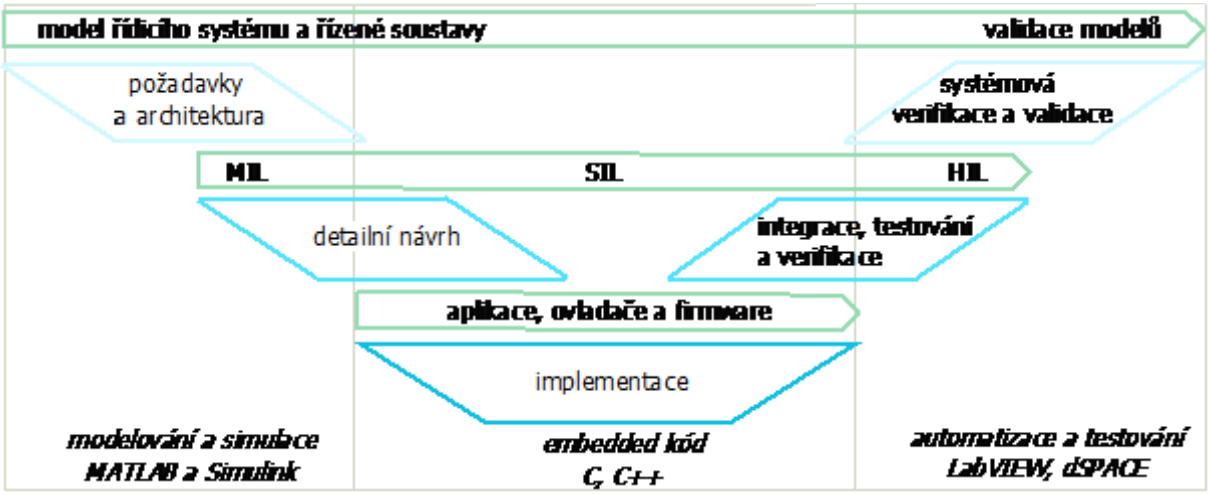
Obor softwarového inženýrství studuje a standardizuje několik metodik, kterými se řídí a definuje životní cyklus SW vývoje. Několik z nich zmiňuje následující výčet [8], [9]:

- vodopádový model a jeho modifikace, resp. rozšíření nazývaná V-model
- spirálový model
- inkrementální model
- agilní vývoj, který vychází z dvou výše uvedených modelů a na jehož principy navazují metodologie jako *Scrum* nebo *Exterme Programming*

2.2.5 V-model

Zejména v automobilovém, ale i drážním průmyslu se intenzivně využívá principů V-modelu, který horizontálně spojuje fáze, ve kterých je systém specifikován a implementován, s fázemi, kde probíhá integrace a testování systému, jak naznačuje Obr. 2.2.

Na základě požadavků na řídicí systém a znalosti parametrů řízené soustavy jsou vytvořeny a otestovány modely metodou MIL. V dalších fázích je generován kód v jazyce C, který je v případě metody SIL testován na PC, kdežto v rámci metody PIL je C kód přeložen do binárního kódu cílového procesoru. Finální verifikace a validace probíhají v rámci metody HIL, kdy je úspěšně otestovaný kód vydán do produkce.



Obr. 2.2: Životní cyklus dle V-modelu

Naznačené (Obr. 2.2) schéma V-modelu je pouze jedna z možných obecných variant. Počet a náplň konkrétních fází se může lišit dle normativních požadavků použitých standardů, dle interních firemních směrnic a norem kvality vývoje SW.

2.2.6 Verifikace a validace

V procesu vývoje software, tedy i v rámci různých metodik jsou používány pojmy verifikace a validace. Každý z těchto pojmů lze nahradit výrazem ověření, což ale zavádí

nejednoznačnost, a proto se v praxi ustálilo používání počeštěných termínů verifikace a validace.

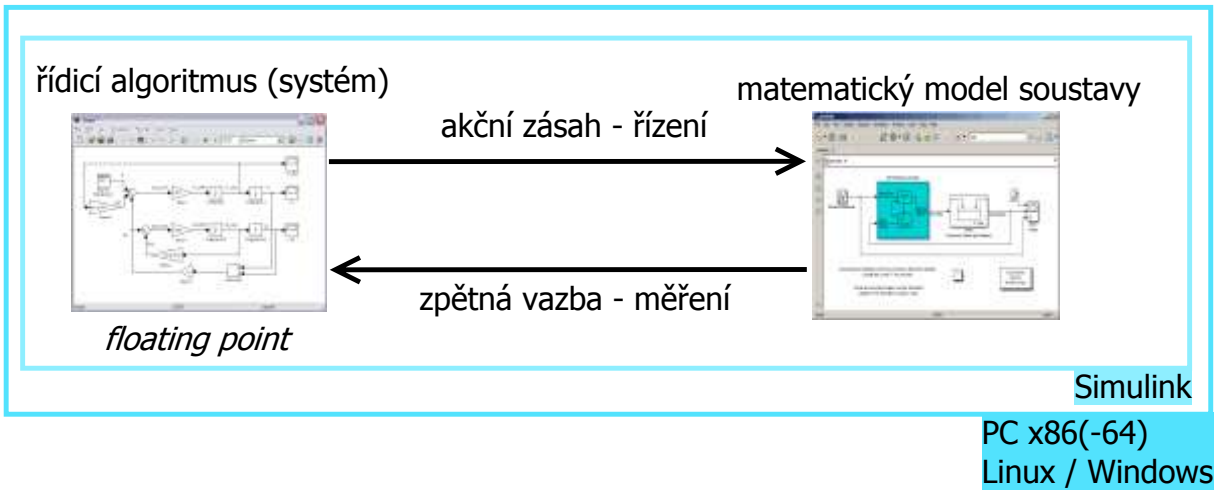
Pro správné pochopení a užívání těchto pojmů je však nezbytné upřesnit metody a cíle jednotlivých fází vývoje SW, což popisuje Tab. 2.2. Z podobného důvodu jednoznačnosti, který byl popsán v předchozím odstavci, jsou i v položkách Tab. 2.2 použity anglické výrazy užívané během procesu vývoje SW.

Tab. 2.2: Verifikace versus validace [6]

Kritérium	**Verifikace**	**Validace**
Definice	Proces, během kterého hodnotíme dílčí produkt (nikoliv finální software, může se jednat o zprávu, dokumentaci, apod.) v dílčí vývojové fázi. Rozhodujeme, zda li jsme v souladu se stanovenými požadavky pro dílčí vývojovou fázi, zda li jsme dosáhli cíle dílčí vývojové fáze správným způsobem.	Proces hodnotící software během a na konci vývojového procesu. Rozhodujeme, zda li uspokojujeme a splňujeme požadavky na výsledný produkt (software).
Cíl	Zajistit, že produkt *je vytvářen* podle specifikace požadavků a specifikace návrhu (designu). Zajistit, že dílčí produkt splňuje stanovené požadavky.	Zajistit, že *produkt skutečně naplňuje* potřeby a požadavky cílového uživatele. Zajistit, že požadavky byly správně specifikovány pro naplnění potřeb uživatele, tedy zamýšlené funkce software v cílovém prostředí nasazení.
Otázka	Vyvíjíme produkt *správně*?	Vyvíjíme *správný* produkt?
Nástroje a položky hodnocení	Plány, specifikace požadavků, specifikace návrhu (designu), zdrojový kód, testovací případ	Vlastní produkt, tedy software
Aktivity	Reviews (recenze kódu, dokumentu) Walkthroughs (párová kontrola kódu, dokumentu nezávislé osoby s původním autorem) Inspections (kontrola proti vstupnímu dokumentu s použitím checklistů)	Testování

2.3 Model In the Loop

Na počátku vývojového cyklu jsou dle vstupních požadavků navrženy za použití metod X In the Loop modely řídicího systému a řízené soustavy v prostředí MATLAB-Simulink. Výpočty obou modelů jsou prováděny s hodnotami ve formátu *floating point*. V této fázi je kladen důraz na včasnou verifikaci vstupních požadavků a navržených algoritmů pro finální řešení na cílové platformě. Jedná se tedy zejména o ověření parametrů regulátoru, resp. řídicího systému a na druhé straně o dostatečně komplexní a přesnou simulaci řízené soustavy. V této fázi návrhu není vývojáři systému poskytnuta informace o výpočetních nárocích algoritmů na konkrétní HW platformě. Tyto informace vyplývají až z následujících metod. Jak ukazuje Obr. 2.3, vytvoření a testování modelů probíhá na platformě PC, např. prostřednictvím MATLAB – Simulink.



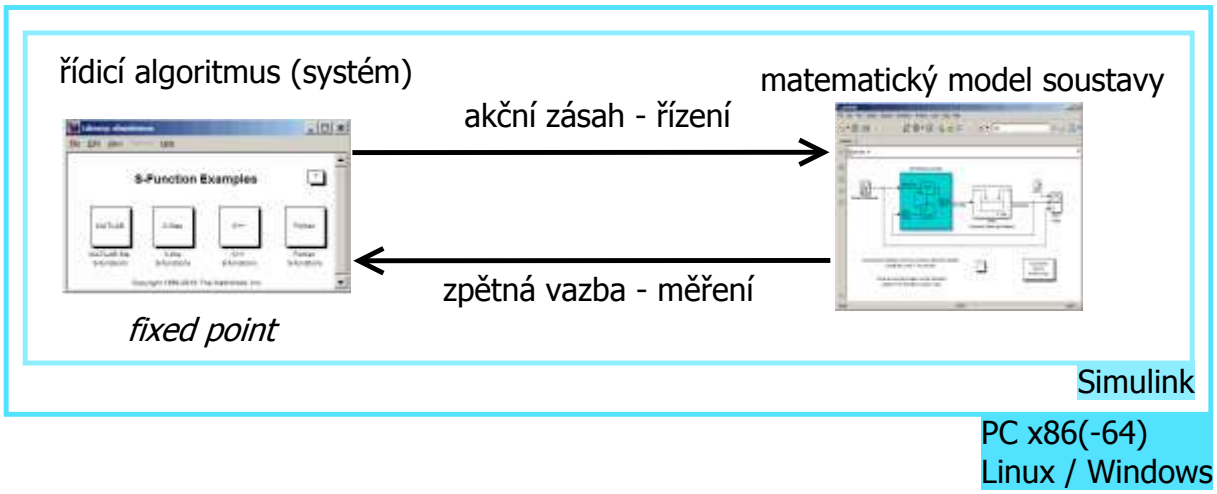
Obr. 2.3: Model In the Loop

2.4 Software In the Loop

V této fázi se testuje, zda li rozsah a přesnost navrženého formátu *fixed point* pro zvolenou cílovou platformu je dostačující a odpovídá nastavení regulátoru, resp. řídicího systému v předchozí fázi. Během SILu se provádí ověření správnosti a funkčnosti zdrojového kódu jednotlivých SW komponent. Jinými slovy, v této fázi může být odhaleno mnoho funkčních chyb bez ohledu na fakt, že přeložený kód není identický (např. odlišná bitová šířka) s kódem určeným pro cílovou platformu.

Řídicí systém je reprezentován tzv. *s-funkcí* (systémovou funkcí) [24]. *S-funkce*, která je naznačena v Obr. 2.4, může být napsána jazyky MATLAB (*m-soubory*), C, C++ nebo Fortran. Je li *s-funkce* vytvořena v C, C++ nebo Fortranu, pak je přeložena do tzv. *MEX souboru*, který je pak nahrán ve formátu DLL knihovny a spuštěn MATLAB interpreterem.

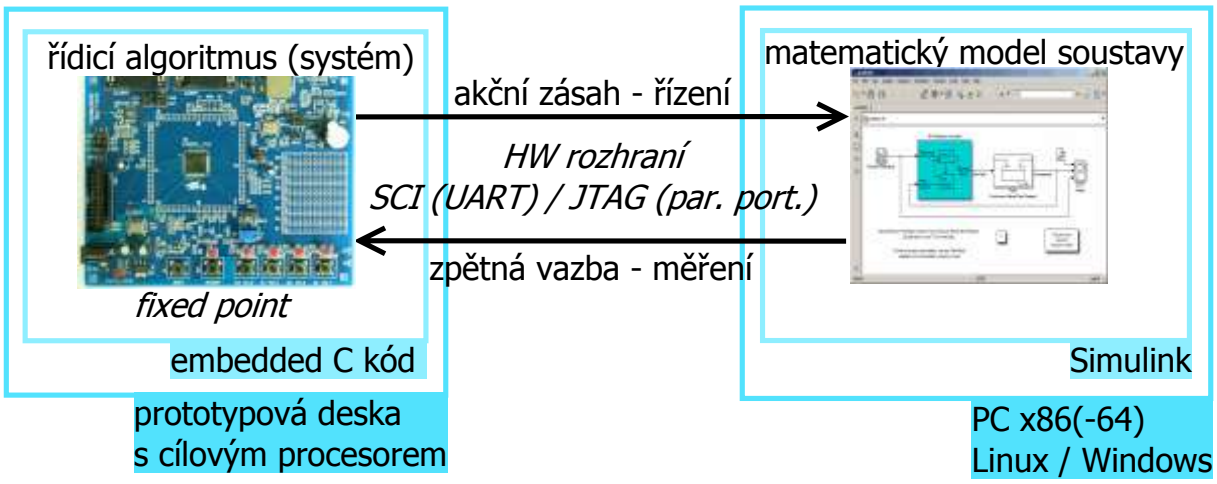
Pomocí *s-funkce* lze rozšiřovat možnosti Simulinku – je možné vytvářet uživatelské bloky pro obecné použití, bloky pro specifické HW ovladače, lze takto přidávat existující kód v jiných jazycích (které např. poskytuje odlišné datové typy).



Obr. 2.4: Software In the Loop

2.5 Processor In the Loop

V rámci metody PIL již běží přeložený C kód na procesoru cílové platformy. V této fázi lze orientačně posoudit požadavky na HW zdroje jako např. spotřeba paměti, výpočetní náročnost. Tyto požadavky na HW však nejsou přesné a definitivní, neboť stále se nejedná o finální SW aplikaci. Dílčí úlohy lze simulovat v reálném čase. Přesto některé *real time* scénáře a události nejsou stále pokryty, neboť ve fázi PIL není testována *real time* dynamika celé SW aplikace (přepínání úloh, latence přerušení) a periférie jako ADC či PWM nejsou používány, neboť simulace podvrhuje (počítá) hodnoty signálů těchto periférií a předává je řídicímu algoritmu. Ladicí komunikace mezi PC a cílovou platformou je realizována buď přes SCI protokol (např. sériovým kanálem přes USB) nebo JTAG protokol (např. paralelním portem), jak zobrazuje Obr. 2.5.



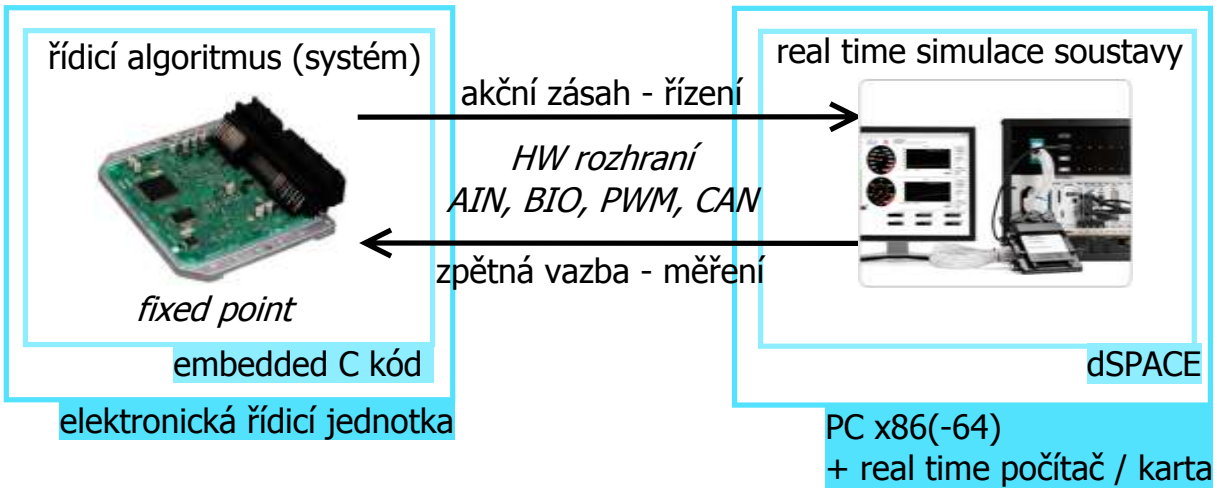
Obr. 2.5: Processor In the Loop

2.6 Hardware In the Loop

Při použití této metody již běží finální aplikace na reálném HW řídicího systému tzv. elektronické řídicí jednotce. Také simulace řízené soustavy běží v reálném čase - např. na PCI Express zásuvné kartě s *real time* systémem postaveným na FPGA apod. nebo na prototypové desce s dostatkem periférií a rozhraní.

Následující Obr. 2.6 ukazuje v levé části příklad elektronické řídicí jednotky pracující v reálném čase. V pravé části Obr. 2.6 je naznačen možný způsob realizace *real time* simulace řízené soustavy HIL systémem firmy dSpace.

Z uvedeného však vychází, že pořizovací náklady na potřebný HW a SW a náklady na samotný vývoj jsou vyšší oproti předchozím fázím. Např. složitost simulace řízené soustavy může být značně vysoká, simulace může vyžadovat až tisíce signálů.



Obr. 2.6: Hardware In the Loop

2.7 X In the Loop testování pro verifikaci a validaci

Následující Tab. 2.3 poskytuje odlišný pohled [23] na rozdělení X In the Loop metod oproti předchozím kapitolám a stručně popisuje vlastnosti a rozdíly jednotlivých metod, resp. odlišnosti od reálného běhu SW na cílové platformě.

Vedle PIL testování na prototypové desce s procesorem cílové platformy ukazuje Tab. 2.3 také možnost PIL testování bez použití hardware. Pro některé procesory je dostupný simulátor, který spouští binární kód cílové platformy. Samotný simulátor však běží na PC platformě a v podstatě provádí vizualizaci HW prostředků cílové platformy, které poskytuje testovanému binárnímu kódu.

Jak je uvedeno v Tab. 2.3, pro PIL testování je kód vykonáván v *real time* pouze částečně, po krocích. Takový kód je vykonáván v reálném čase v rámci jedné iterace (jednoho kroku) simulace, tedy např. jeden průběh výpočetní smyčky proběhne v reálném čase, ale další průběh je spuštěn až při definovaných podmínkách v rámci simulace.

Ve fázi HIL testování již kód běží na cílové platformě. Takové systémy pracují v *hard real time* režimu, tedy mají striktně definované časové intervaly pro vykonávání jednotlivých úloh. Porušení těchto pravidel není tolerováno a musí být ošetřeno jako porucha.

Tab. 2.3: X In the Loop testování pro verifikaci a validaci [23]

	SIL testování	**PIL testování na embedded HW**	**PIL testování na simulátoru instrukční sady**	**HIL testování**
Účel	Verifikovat zdrojový kód komponent	Verifikovat objektový (přeložený) kód komponent	Verifikovat objektový (přeložený) kód komponent	Verifikovat systémovou funkcionalitu
Věrnost a přesnost	Dvě možnosti: * Stejný zdrojový kód jako pro cílovou platformu, ale mohou se lišit numericky (bitová šířka slova) * Změna zdrojového kódu pro emulaci šířky slova cílové platformy, ale stejná přesnost ve *fixed point* výpočtech	Stejný objektový (přeložený) kód Stejná přesnost ve *fixed point* výpočtech Stejný počet cyklů, neboť kód běží na HW	Stejný objektový (přeložený) kód Stejná přesnost ve *fixed point* výpočtech Počet cyklů může být odlišný, neboť kód běží v rámci simulátoru na hostiteli (např. PC)	Stejný objektový (přeložený) kód Stejná přesnost ve *fixed point* výpočtech Stejný počet cyklů, neboť kód běží na HW Použití reálných a emulovaných vstupních / výstupních kanálů
Snadnost použití a cena (náklady)	Snadná práce na PC Simulink Žádný HW (cílové platformy).	Spouštěn na stole nebo testovací stanici Prototypová deska s cílovým procesorem.	Snadná práce na PC Simulink a simulátor Žádný HW (cílové platformy).	Spouštěn na stole nebo testovací stanici Elektronická řídicí jednotka
Real time (RT) možnosti	Není RT	Není RT (mezi kroky/ vzorky)	Není RT (mezi kroky / vzorky)	Hard real time

3 Základní principy řízení v otevřené a uzavřené smyčce

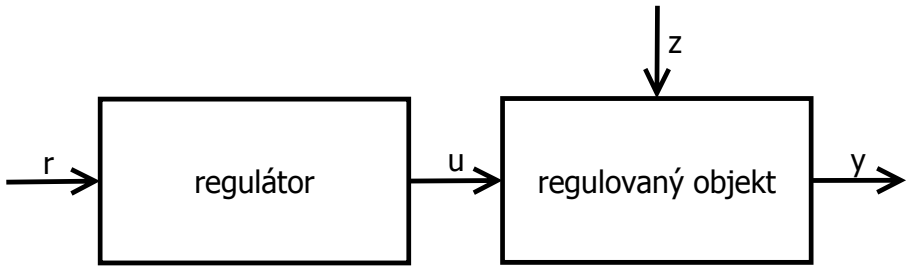
Pro návrh řízení je nejprve nutné definovat některé pojmy jako regulovaná soustava a řízení v otevřené a uzavřené smyčce.

Regulovaná soustava (dynamická soustava), je taková soustava, ve které se pomocí regulátoru ovlivňuje regulovaný objekt, tím je myšlen jeho stav a chování. Regulovaným objektem je v této práci polovodičový měnič a stejnosměrný motor s PM.

„Teorie řízení je soubor poznatků z aplikované matematiky, umožňující analýzu a syntézu tj. návrh takových vhodných regulátorů, aby zpětnovazební soustava byla stabilní a měla požadované dynamické chování, např. velikost překmitu, koeficient tlumení, ustálenou chybu apod.“ [16]

3.1 Řízení v otevřené smyčce

Tento druh řízení se využívá především pro nenáročnou regulaci motorů. Pomocí principu otevřené smyčky nelze zjistit informace o stavu motoru, např. zda se motor otáčí požadovanou rychlostí či není zablokován. Jednoduché schéma řízení v otevřené smyčce je uvedeno na Obr. 3.1.



Obr. 3.1: Řízení v otevřené smyčce

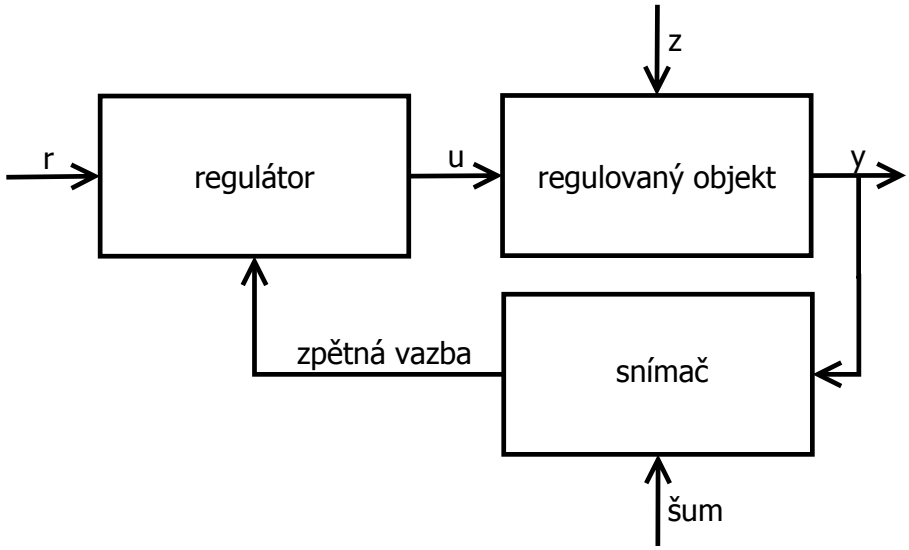
kde

- r referenční vstup (žádaná hodnota výstupní veličiny)
- u řídicí veličina (akční veličina)
- y řízená veličina (výstupní veličina)
- z porucha

Pomocí regulátoru se nastavuje řídicí veličina u. Nelze však získat informaci o skutečné hodnotě řízené veličiny y (chybí zde zpětná vazba). Tato veličina y je ovlivněna také poruchovou veličinou (např. změna síťového napětí či zatížení motoru).

3.2 Řízení v uzavřené smyčce (zpětnovazební řízení)

U této metody řízení lze regulaci nejen monitorovat, ale také řídit (rozběh a doběh motoru, rychlost otáček atd.) a kompenzovat parametry soustavy a vliv poruchové veličiny. Princip řízení v uzavřené smyčce je znázorněn na Obr. 3.2.



Obr. 3.2: Řízení v uzavřené smyčce

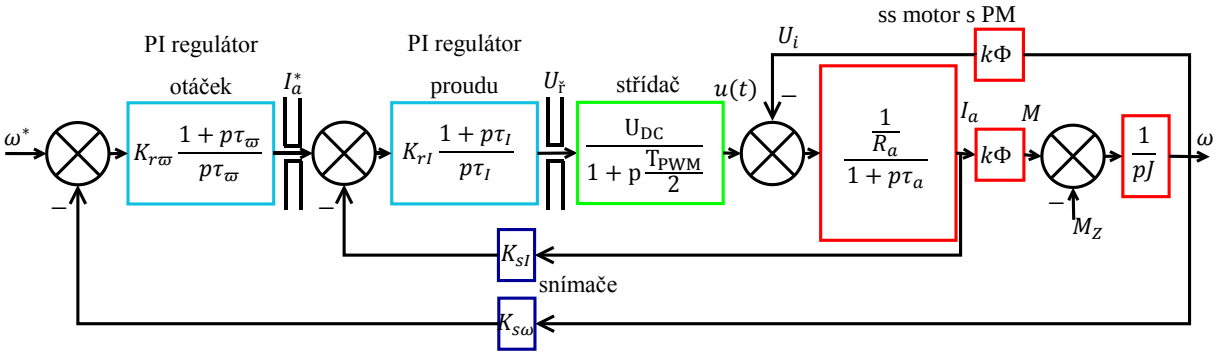
Řídicí veličina *u* se nastavuje pomocí regulátoru po zjištění informace o hodnotě žádané veličiny *r* a také informace o hodnotě regulované veličiny *y*.

Tato hodnota se získává měřením, tj. zařazením snímače do zpětné vazby a jeho přivedením na druhý vstup regulátoru. Snímačem mohou být různé druhy senzorů a čidel [16].

Bloky regulované soustavy řešené v této práci jsou realizovány následujícím způsobem:

- regulátor
    - procesor (řídicí jednotka) se vstupy (měření analogových veličin) a PWM výstupy připojenými na spínací tranzistorové můstky (polovodičový měnič)
- regulovaný objekt
    - polovodičový měnič a stejnosměrný motor s permanentními magnety
- zpětná vazba
    - měření analogových veličin

4 Popis elektromechanického pohonu - řídicího systému a řízené soustavy



Obr. 4.1: Náhradní blokové schéma pohonu

Popis jednotlivých bloků regulačního řetězce (viz Obr. 4.1) a postup jejich návrhu je předmětem následujících kapitol.

4.1 Regulátor

V této kapitole je popsán teoretický přenos PI regulátoru. Vyčíslení parametrů přenosu PI regulátoru je pak popsáno v kapitolách 5.1 a 5.1.3. Dále je zmíněn důležitý jev dopravního zpoždění, který je nutné zahrnout do návrhu PI regulátoru a jeho implementaci v číslicovém systému. Na závěr kapitoly je stručně popsán vývojový kit s mikrokontrolérem TMS320F2812, který je v této práci zvolen pro implementaci PI regulátoru.

4.1.1 Přenos PI regulátoru

Pro návrh regulátoru proudu a stejně tak i regulátoru otáček je použit PI regulátor, který je tvořen paralelním spojením proporcionálního a integračního členu regulátoru. Jedná se o nejrozšířenější spojitý regulátor používaný v technice pohonů, s jehož pomocí je při regulaci v uzavřené smyčce odstraněna regulační odchylka, která by nepříznivě ovlivnila výsledky regulace při použití pouze P (proporcionálního) regulátoru.

Vztah pro řídicí napětí:

$$U_{Ř_{(t)}} = K_p e_{(t)} + K_i \int e\,dt \tag{4.1}$$

kde

$U_{Ř}$	řídicí napětí [V]
K_p	proporcionální konstanta (zesílení) [-]
e	regulační odchylka [-]
K_i	integrační konstanta [-]

Pro nulové počáteční podmínky pak lze dle operátorového slovníku Laplaceovy transformace rovnici (4.1) přepsat do tvaru:

$$U_{\check{\mathrm{R}}} = K_p \left(1 + \frac{K_i}{K_p}\frac{1}{p}\right) e_{(p)} \tag{4.2}$$

Vztah pro přenos proudového regulátoru vyjádříme jako:

$$F_{(p)} = \frac{U_{\check{\mathrm{R}}(p)}}{e_{(p)}} = ? \tag{4.3}$$

Po dosazení z rovnice (4.2) je přenos regulátoru:

$$F_{(p)} = \frac{U_{\check{\mathrm{R}}(p)}}{e_{(p)}} = K_p \frac{1 + \frac{K_i}{K_p}\frac{1}{p}}{1} \tag{4.4}$$

Po vynásobení Laplaceovým operátorem p a rozšíření:

$$F_{(p)} = K_p \frac{p + \frac{K_i}{K_p}\frac{K_p}{K_i}}{p\frac{K_p}{K_i}} = K_p \frac{1 + \frac{K_p}{K_i}p}{\frac{K_p}{K_i}p} \tag{4.5}$$

Poměr regulační a proudové konstanty regulátoru se rovná časové konstantě regulátoru:

$$\frac{K_p}{K_i} = \tau_{REG} \tag{4.6}$$

Konečný stav pro přenos regulátoru je po dosazení vztahu (4.6) do (4.5) následující:

$$F_{(p)} = K_p \frac{1 + p\tau_{REG}}{p\tau_{REG}} \tag{4.7}$$

4.1.2 Dopravní zpoždění

Je třeba uvažovat dopravní zpoždění, které plyne z principů a mechanismů číslicového řízení jako je synchronismus, vzorkování, latence přerušení (doba mezi výskytem události a následnou obsluhou události) apod. Dopravní zpoždění vzniká jak při generování PWM, které právě spouští A/D převod, tak při následném měření A/D převodníkem.

Vychází ze spínací frekvence, resp. periody spínání T_{PWM}:

$$T_D = \frac{T_{PWM}}{2} \tag{4.8}$$

4.1.3 Popis vývojového kitu eZdsp™ F2812

Číslicový regulátor je v této práci implementován na mikrokontroléru TMS320F2812. Pro vývoj aplikací s tímto mikrokontrolérem je vhodné použít vývojový kit eZdsp™ F2812.

Parametry vývojové desky [14] eZdsp™ F2812 a jejich komponent:

- digitální signálový procesor TMS320F2812 [13] pracující na frekvenci až 150 MHz ve formátu *fixed point*
- pamětové sběrnice – Harvardská architektura[2]
- interní RAM o velikosti 18K (tzv. slov / *words*) × 16 bitů - těchto 36 KBytů je rozděleno do několika bloků / oblastí)
- externí SRAM o velikosti 64K × 16 bitů
- interní Flash (viz písmeno „F“ v typovém označení) o velikosti 128K × 16 bitů
- IEEE 1149.1 JTAG řadič s vyvedením na JTAG konektor nebo na paralelní port
- Expanzní konektory (analogové a binární signály)

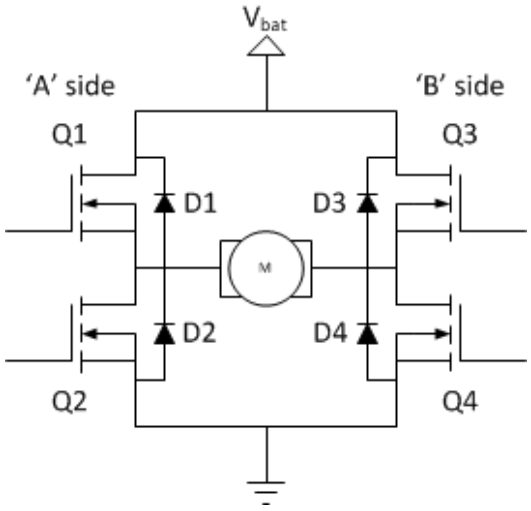
Jak je výše, vývojová deska eZdsp™ F2812 je osazena expanzními konektory a to konkrétně:

- konektory P4 / P8 / P7, na které jsou přivedeny I/O signály jako např.:
    - 12 signálů PWM
    - signály modulů tzv. *Event Manager modules* (*Capture Unit, Quadrature-Encoder Pulse Circuit*)
    - 2 kanály SCI / UART
    - 1 kanál SPI sběrnice
    - 1 kanál CAN sběrnice
- konektory P5 / P9 pro analogové signály:
    - 16 kanálů (2 × 8) 12-bitového ADC

[2] V případě TMS320F2812 se jedná o sběrnici pro programovou paměť o 22 adresních a 32 datových signálech a sběrnici pro datovou paměť sestávající z 32 adresních a 32 datových pro oba směry (čtení a zápis).

4.2 Měnič

Použitý měnič má topologii H-můstku, který je pro názornost zobrazen na Obr. 4.2, kde je však realizován unipolárními tranzistory. V měniči, který je použit v této práci, jsou spínacími prvky IGBT.



Obr. 4.2: Zapojení H můstku

4.2.1 Jednofázový střídač s IGBT

Pro řízenou soustavu řešenou v této práci je uvažována spínací frekvence $f_{PWM} = 10\ kHz$ resp. periody spínání $T_{PWM} = 100\ \mu s$.

Vstupem měniče je PWM signál (viz dopravní zpoždění (4.8)), na výstupu měniče je generováno $U_{DC} = 42$ V, pak přenosová funkce:

$$F_{(p)} = \frac{U_{DC}}{1 + p\frac{T_{PWM}}{2}} \tag{4.9}$$

4.2.2 Pulzně šířková modulace

Pulzně šířková modulace z anglického PWM *(Pulse Width Modulation)* se řadí mezi diskrétní modulace. Slouží pro přenos (kódování) analogové veličiny prostřednictvím dvouhodnotového signálu s konstantní periodou. Dvouhodnotovým signálem může být například napětí nebo proud. Signál je přenášen pomocí střídy, což je poměr délky impulsu k délce mezery jedné periody. Střídu lze vyjádřit také procentuálně. PWM je používána pro spínání napájecích zdrojů a také výkonové elektrotechnice pro řízení otáček stejnosměrných strojů.

4.2.3 Mrtvé časy

Reálné (nikoliv ideální) elektronické spínací prvky (např. IGBT nebo unipolární tranzistory) nemohou vypnout protékající proud v nekonečně krátkém čase [26]. Tento fenomén je nezbytné uvažovat při navrhování a řízení měničů. Je třeba zajistit, aby ve větvi byla sepnuta pouze jedna součástka. V opačném případě hrozí vytvoření větvového zkratového proudu, a tím poškození spínacích prvků. Vypínací časy v současnosti vyráběných součástek se nacházejí v intervalu jednotek ns až desítek μs v závislosti na velikosti spínaných výkonů, spínací frekvenci apod.

V této práci je počítáno s mrtvým časem 2 μs.

4.3 Stejnosměrný motor s permanentními magnety

4.3.1 Stručný teoretický popis stejnosměrného motoru s permanentními magnety

Elektrické stroje obecně slouží k přeměně elektrické energie na mechanickou energii, tedy mechanickou práci - nejčastěji s využitím silových účinků magnetického pole (motory) - nebo k opačné přeměně (v tomto případě stroje označujeme jako generátory, alternátory, dynama).

Princip funkce stejnosměrného motoru byl objeven již kolem roku 1870, proto se tyto motory řadí mezi nejstarší elektrické stroje. Vzhledem k jejich dlouhodobému vývoji patří mezi detailně propracované stroje. K jejich charakteristickým atributům patří dobré dynamické a regulační vlastnosti a dobrá přizpůsobivost různým druhů zátěže. Hlavní nevýhodou těchto strojů může být především vyšší pořizovací cena, ke které se musí ještě uvažovat cena napájecího systému. V současnosti se používají především jako servo motory, tzn. jako motory pro pohony, dále také v robotice [15].

Stejnosměrný stroj je založen na principu smyčky rotující v magnetickém poli. Konstrukce stroje sestává ze statoru (pevná část), který je obvykle tvořen magnetickým obvodem, zhotoveným z plného materiálu, a rotoru (pohybující se část), jinak nazývaného kotva, která je zhotovena s elektromagnetu se dvěma póly. Magnetický tok obvodu je u tohoto druhu motoru buzen permanentními magnety (dále jen PM), přičemž obvod je tvořen z transformátorových plechů. U stroje s PM je klasický komutátor (mechanický rotační usměrňovač) nahrazen komutací elektronickou, tzn. komutací pomocí polovodičových spínacích prvků. Rotační přepínač, zvaný komutátor, slouží jako přepínač polarity proudu a polarity magnetického pole, které prochází kotvou dvakrát během každé otáčky. Tím je zajištěn stejný směr síly působící na póly rotoru – pokud je přepnuta polarita (mrtvý úhel

motoru), běh motoru je udržen stále ve správném směru.

V případě stejnosměrných strojů s PM, může být použito tzv. obrácené uspořádání – budící magnety jsou umístěné na rotoru, zatímco kotva se nachází na statoru [15].

Dle druhu buzení rozdělujeme stejnosměrné motory do čtyř skupin:

- s cizím buzením – budící vinutí hlavních pólů napájeno z nezávislého stejnosměrného zdroje, popř. je stroj osazen permanentními magnety
- s derivačním buzením – budící vinutí hlavních pólů je zapojeno paralelně ke kotvě
- sériové – budící vinutí hlavních pólů je zapojeno do série s kotvou
- kompaundní – na hlavních pólech jsou budicí vinutí derivační a sériové

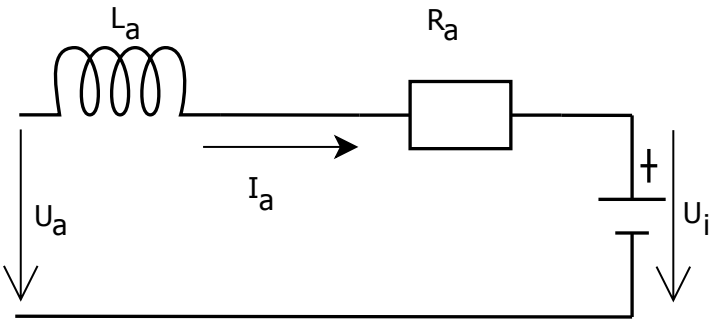
Motory s cizím buzením dále dělíme:

- motory s permanentními magnety
- motory s vinutým statorem

Rychlost, resp. otáčky stejnosměrného motoru s cizím buzením lze řídit následujícími metodami:

- změnou celkového odporu v obvodu kotvy (zapojení přídavného R_S)
- změnou přiváděného svorkového napětí U_a na kotvu motoru
- změnou magnetického toku Φ
- kombinací řízení změnou napětí U_a kotvy i změnou magnetického toku Φ

4.3.2 Elektrická a mechanická část stejnosměrného motoru s permanentními magnety



Obr. 4.3: Náhradní schéma kotvy stejnosměrného motoru

Vztah pro svorkové napětí U_a dle náhradního schématu je dán diferenciální rovnicí prvního řádu jako:

$$U_a = L_a \frac{dI_a}{dt} + R_a I_a + U_i \; [V] \tag{4.10}$$

kde

L_a indukčnost kotvy [H]

R_a odpor kotvy [Ω]

I_a proud kotvy [A]

U_i indukované napětí v kotvě [V]

Dále je nutné zavést vztah pro indukované napětí stroje U_i:

$$U_i = k\Phi\omega \ [V] \tag{4.11}$$

kde

U_i indukované napětí v kotvě [V]

k konstanta stroje (závisí na konstrukci, počtu závitů) [-]

Φ magnetický indukční tok [Wb]

ω úhlová rychlost stroje [$rads^{-1}$]

Po dosazení vztahu (4.11) do rovnice (4.10) získáme:

$$U_a = L_a \frac{dI_a}{dt} + R_a I_a + k\Phi\omega \ [V] \tag{4.12}$$

Vyjádření momentu stroje na hřídeli M:

$$M = k\Phi I_a \ [Nm] \tag{4.13}$$

kde

k konstanta stroje (závisí na konstrukci, počtu závitů) [-]

Φ magnetický indukční tok [Wb]

I_a proud kotvy [A]

Akcelerační moment M_{akcel}:

$$M_{akcel} = M - M_z \ [Nm] \tag{4.14}$$

kde

M moment stroje (moment na hřídeli) [Nm]

M_z zátěžový moment [Nm]

Pohybová rovnice s dosazení vztahů (4.14) a (4.13):

$$J\frac{d\omega}{dt} = M_{akcel} = M - M_z = k\Phi I_a - M_z \tag{4.15}$$

kde

J moment setrvačnosti rotoru [kgm^2]

ω úhlová rychlost stroje [rads^{-1}]

M_{akcel} akcelerační moment [Nm]

k konstanta stroje (závisí na konstrukci, počtu závitů) [-]

Φ magnetický indukční tok [Wb]

I_a proud procházející kotvou [A]

M_z zátěžový moment [Nm]

Pro návrh modelu v MATLAB-Simulinku převedeme soustavu diferenciálních rovnic (4.12) a (4.15) pomocí Laplaceovy transformace na rovnice algebraické. Pro nulové počáteční podmínky pak platí dle operátorového slovníku Laplaceovy transformace:

$$\frac{d}{dt} \rightarrow p \tag{4.16}$$

Po aplikaci Laplaceovy transformace lze rovnice (4.12), (4.13) a (4.15) přepsat do tvarů**:**

$$U_{a(p)} = L_a I_{a(p)} p + R_a I_{a(p)} + k\Phi\omega_{(p)} \tag{4.17}$$

$$J\omega_{(p)} p = M_{akcel(p)} = k\Phi I_{a(p)} - M_z \tag{4.18}$$

$$M_{(p)} = k\Phi I_{a(p)} \tag{4.19}$$

4.3.3 Přenosová funkce stejnosměrného motoru s permanentními magnety

Přenosové funkce (přenos) vyjadřují vztah [16] mezi vstupní a výstupní veličinou, jedná se tedy o podíl Laplaceových obrazů vstupu a výstupu při nulových počátečních podmínkách.

$$F_{(p)} = \frac{Y_{(p)}}{R_{(p)}} \tag{4.20}$$

kde

$Y_{(p)}$ výstupní veličina

$R_{(p)}$ vstupní veličina

Z rovnic (4.17), (4.18) a (4.19) jsou následně odvozeny přenosy pro návrh základního regulačního schématu:

- přenos svorkového napětí kotvy na proud v kotvě (4.23) získáme následným dosazením do rovnice (4.21):

$$F_{UI_{(p)}} = \frac{I_{a(p)}}{U_{a(p)}} = ? \tag{4.21}$$

Při zanedbání indukovaného napětí U_i lze rovnici (4.17) přepsat ve tvaru:

$$U_{a(p)} = L_a I_{a(p)} p + R_a I_{a(p)} \tag{4.22}$$

$$F_{UI_{(p)}} = \frac{I_{a(p)}}{U_{a(p)}} = \frac{1}{R_a + L_a p} = \frac{\frac{1}{R_a}}{1 + \frac{L_a}{R_a} p} \tag{4.23}$$

- přenos proudu v kotvě na moment stroje:

$$F_{IM_{(p)}} = \frac{M_{(p)}}{I_{a(p)}} = ? \tag{4.24}$$

Z rovnice (4.19) je odvozeno:

$$F_{IM_{(p)}} = \frac{M_{(p)}}{I_{a(p)}} = k\Phi \tag{4.25}$$

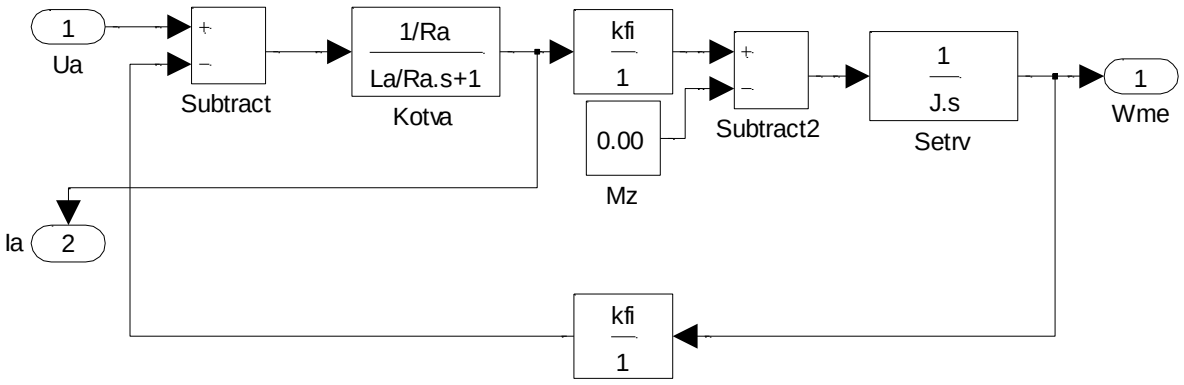
- přenos akceleračního momentu na otáčky

$$F_{M\omega_{(p)}} = \frac{\omega_{(p)}}{M_{akcel(p)}} = ? \tag{4.26}$$

Z rovnice (4.18) je vyjádřen jako:

$$F_{M\omega_{(p)}} = \frac{\omega_{(p)}}{M_{akcel(p)}} = \frac{1}{pJ} \tag{4.27}$$

4.3.4 Model stejnosměrného motoru s permanentními magnety



Obr. 4.4: Model stejnosměrného motoru s permanentními magnety

Na Obr. 4.4 vstupuje do bloku *Subtract2* nulová konstanta jako menšitel, neboť uvažujeme $\mathrm{M_z} = 0\ [Nm]$.

Mechanické otáčky *Wme* jsou přímo úměrné elektrickým otáčkám *Wel* s přepočtem dle počtu magnetických pólových dvojic (tzv. pólpárů) motoru. V tomto případě, kdy je použit motor s jedním pólpárem jsou mechanické otáčky rovny elektrickým.

4.3.5 Parametry použitého stejnosměrného motoru s permanentními magnety

Parametry pro model a simulaci řízené soustavy, tedy stejnosměrného motoru s permanentními magnety odpovídají motoru typu P2XR506 (viz Tab. 4.1), který je používán pro účely výuky na katedře elektromechaniky a výkonové elektroniky FEL ZČU.

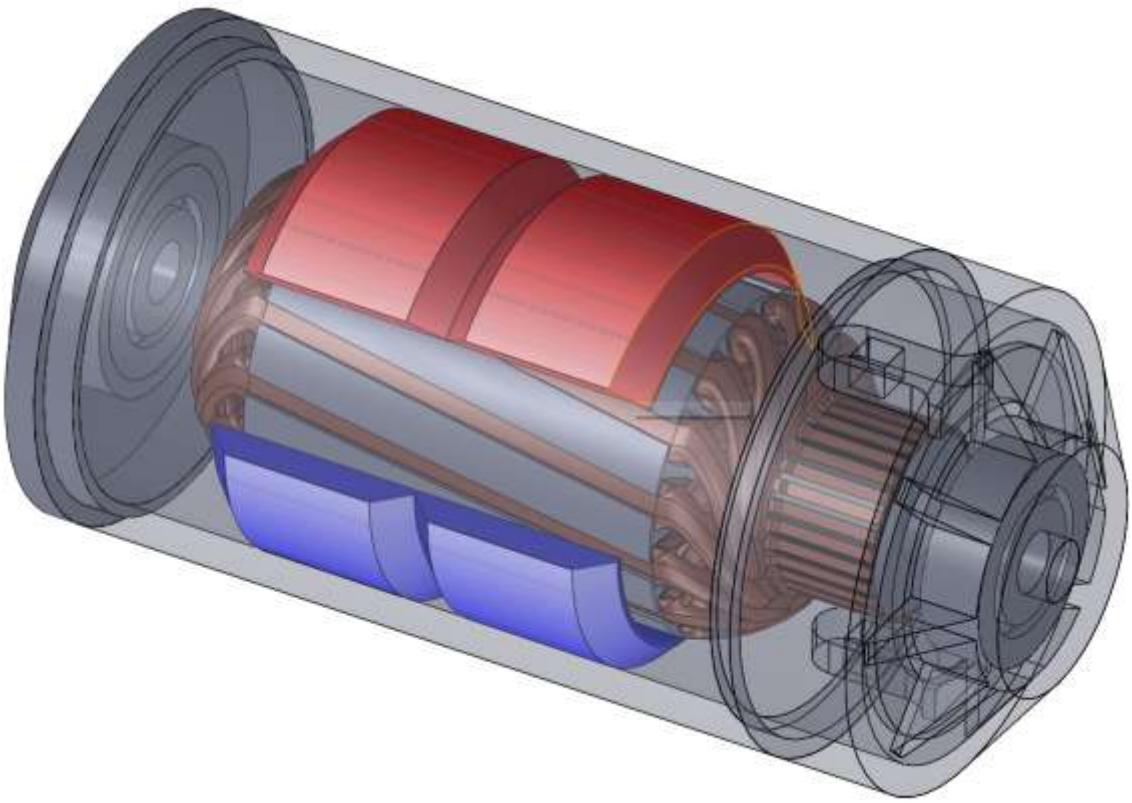
Tab. 4.1: Štítkové a katalogové hodnoty stejnosměrného motoru s permanentními magnety

Parametr	**Hodnota**
Jmenovité napětí U_n	42 [V]
Jmenovitý proud I_n	3,3 [A_{rms}]
Jmenovité otáčky ω_n	4000 [rpm]

Tab. 4.2: Změřené hodnoty stejnosměrného motoru s permanentními magnety

Parametr	**Hodnota**
Odpor kotvy R_a	4,9 [Ω]
Indukčnost kotvy L_a	2,9 * 10^{-3} [H]
Moment setrvačnosti J	3 * 10^{-4} [kgm^2]

Na obrázku Obr. 4.5 je znázorněn model stejnosměrného motoru (typ P2XR506 [29]**)** s permanentními magnety, který je navrhnut v programu Solid Works a v této práci je použit v experimentech v kapitole 8.3.



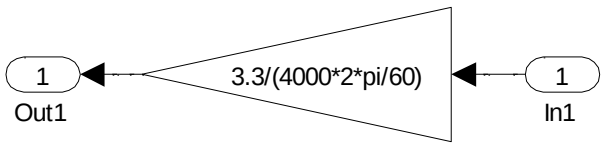
Obr. 4.5: Model stejnosměrného motoru s permanentními magnety - typ P2XR506 [29]

4.4 Snímače

V náhradním blokovém schématu pohonu (Obr. 4.1) jsou naznačeny dvě zpětné vazby pro měření skutečných hodnot otáček a proudu. Zpětná vazba je realizována prostřednictvím čidel otáček a proudu vytvářejících napěťový signál, který vstupuje do regulátoru, kde je signálově upraven, diskretizován a předán algoritmu pro regulaci. Bloky pro měření, úpravu a diskretizaci signálu popisují následující kapitoly.

4.4.1 Měření otáček - tachodynamo

Elektromechanická sestava, která je modelována jako řízená soustava a na které jsou provedena jistá měření (např. rychlostního profilu), sestává vedle stejnosměrného motoru s permanentními magnety z tachodynama K5A9-00, které je připojené přímo na rotor motoru pro měření rychlosti, resp. otáček rotoru. Převod tachodynama je dán výstupním napětím 20 [V] při otáčkám rotoru 1000 [rpm]. Štítek tachodynama definuje maximální otáčky 6000 [rpm], tedy při uvažované lineární převodní charakteristice odpovídající maximálnímu výstupnímu napětí 120 [V].

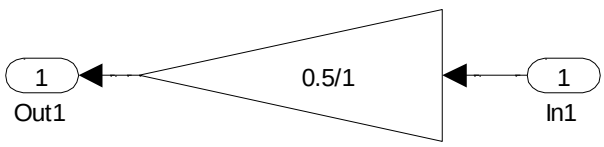


Obr. 4.6: Model tachodynama

Výstup tachodynama je přes vstupní napěťové děliče připojen na interní ADC kanál s rozsahem 0 – 3 [V]. Díky vstupní síti operačních zesilovačů je unipolární rozsah ADC kanálu převeden na bipolární ±5V. Pro jmenovité otáčky motoru 4000 [rpm] bylo změřeno napětí před A/D převodníkem 3,3 [V], což naznačuje Obr. 4.6, kde je zahrnut přepočet z [rpm] na [rads^{-1}].

4.4.2 Měření proudu – LEM

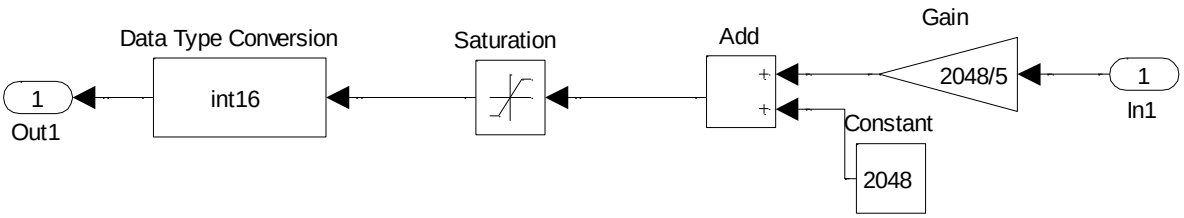
Měření proudu je realizováno čidly od známé firmy LEM. Velikost výstupního napětí je dána převodem 0,5 [V] / 1 [A]. Napěťový výstup je pak dále měřen vnitřním A/D převodníkem.



Obr. 4.7: Model čidlo proudu – LEM

4.4.3 Vstupní obvody a A/D převodníky

Jak zmiňují předchozí kapitoly, napěťové výstupy čidla otáček a čidla proudu jsou na vstupu procesorové desky upraveny vstupními děliči a operačními zesilovači. Vstupní bipolární signál ± 5[V] je tedy upraven na unipolární rozsah interních A/D převodníků 0 – 3 [V]. Na Obr. 4.8 je znázorněn blok saturace, který je nastaven na nejvyšší hodnotu 12-bitového A/D převodníku, tedy 4095. A/D převodníky tak provádějí diskretizaci vstupních hodnot – viz následující kapitola 5.3.5 *Vstupní obvody a A/D převodníky*.

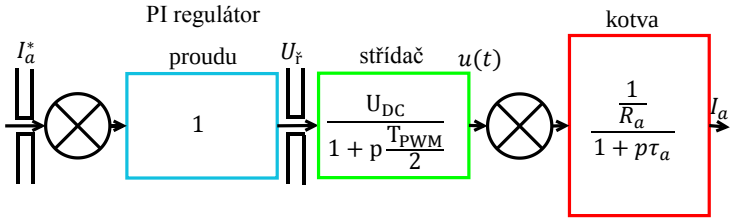


Obr. 4.8: Model vstupních obvodů a A/D převodníku

5 Návrh a implementace metody MIL v prostředí Simulink

5.1 Návrh regulátoru proudu

5.1.1 Otevřená smyčka



Obr. 5.1: Regulátor proudu – návrh konstanty $\boldsymbol{K_{rI}}$

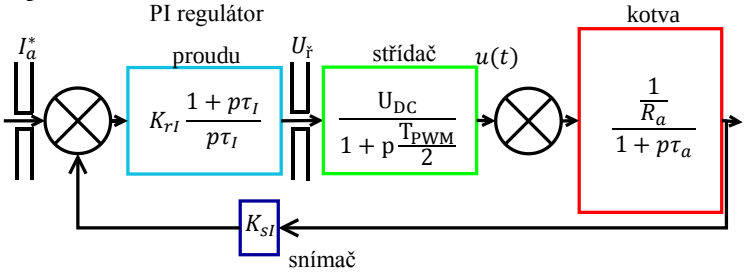
Pro výpočet přenosu otevřené smyčky je PI regulátor proudu nahrazen blokem s jednotkovým zesílením, resp. přenosem $F_{(p)} = 1$ (v tuto chvíli je tedy uvažován pouze jako P regulátor proudu). Vliv indukovaného napětí U_i je zanedbán, neboť změny proudu kotvy jsou mnohem rychlejší než změny otáček a zároveň integrační složka PI regulátoru odstraňuje vliv indukovaného napětí.

S těmito předpoklady a se znalostí přenosu otevřené smyčky, tedy měniče a kotvy motoru jsou sestrojeny logaritmické frekvenční charakteristiky pro amplitudu (resp. zesílení) a fázi.

Z charakteristik jsou určeny (frekvence) a dopočítány (potřebné zesílení) hodnoty, které odpovídají zvolené fázové bezpečnosti. Běžně se fázová bezpečnost volí mezi 50° a 70°. Pro tento návrh je určena fázová bezpečnost $\varphi_b = 70°$, což je úmyslně hodnota dostatečně velká. Je to dáno několika faktory, a to poměrně velkými úbytky na použitých IGBT prvcích, napájení malým napětím a možnými nepřesnostmi při měření parametrů.

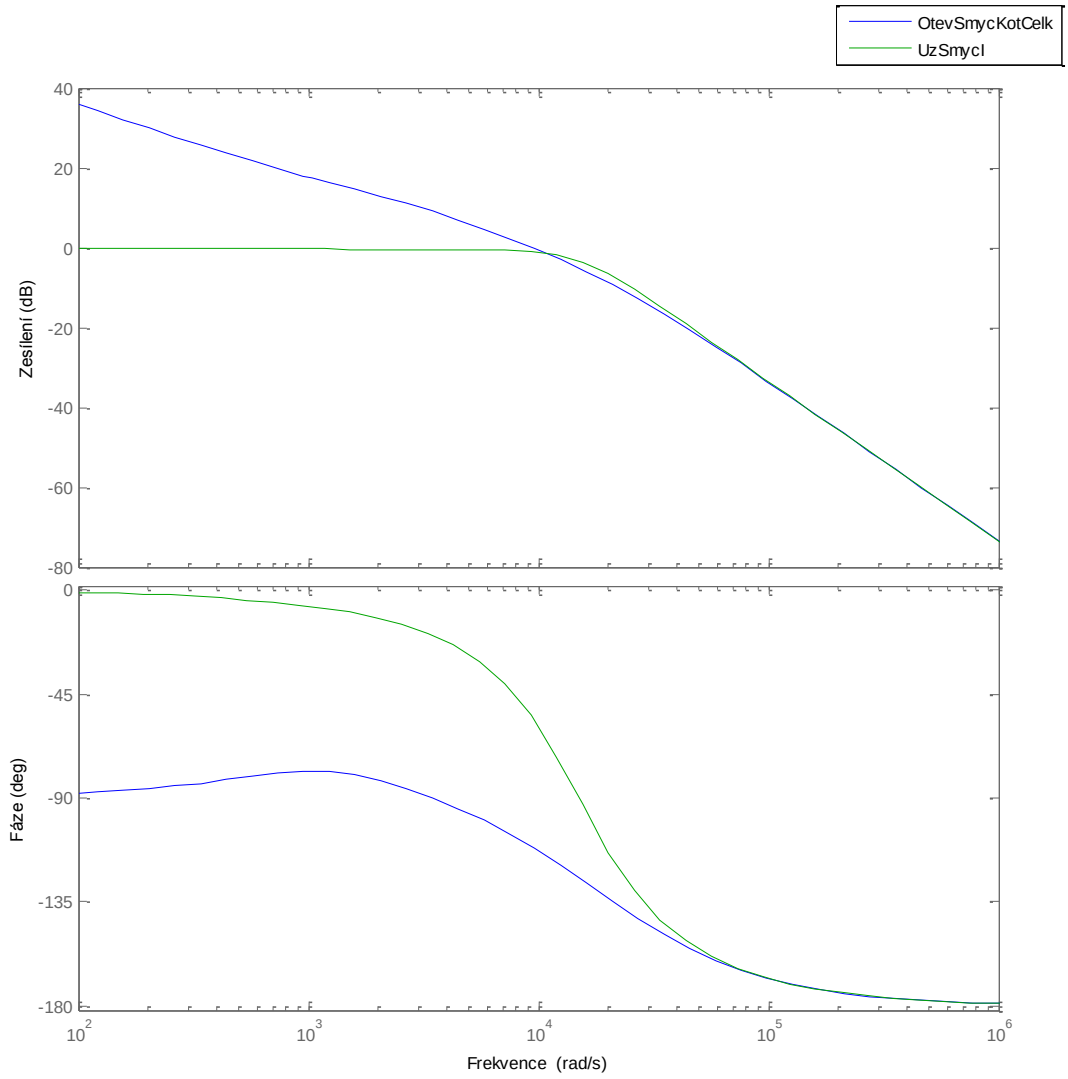
Časová konstanta τ_I proudového regulátoru je volena tak, aby odpovídající frekvence byla alespoň o dekádu nižší (tedy na ose *x* vlevo od) frekvence, která odpovídá průchodu nulou amplitudové logaritmické frekvenční charakteristiky pro přenos v otevřené smyčce, kdy přenos P regulátoru je K_{rI}.

5.1.2 Uzavřená smyčka



Obr. 5.2: Uzavřená proudová smyčka

5.1.3 Logaritmické frekvenční charakteristiky regulátoru proudu



Obr. 5.3: Logaritmické frekvenční charakteristiky regulátoru proudu

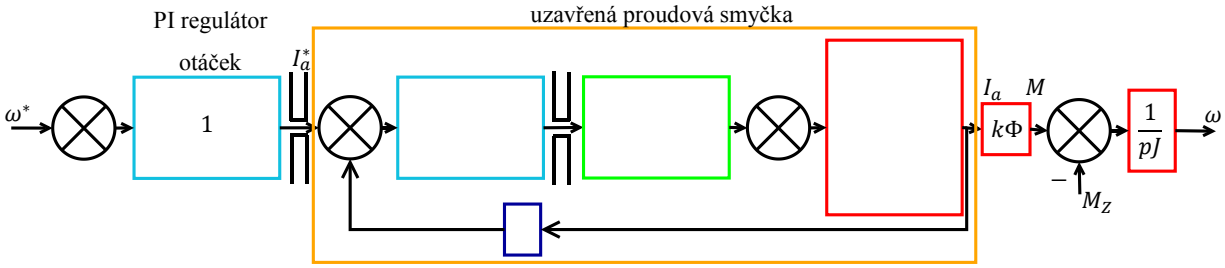
Legenda k Obr. 5.3:

- modré křivky: amplitudová a fázová charakteristika otevřené smyčky pro regulátor proudu
- zelená křivky: amplitudová a fázová charakteristika uzavřené smyčky pro regulátor proudu

5.2 Návrh regulátoru otáček

5.2.1 Otevřená smyčka

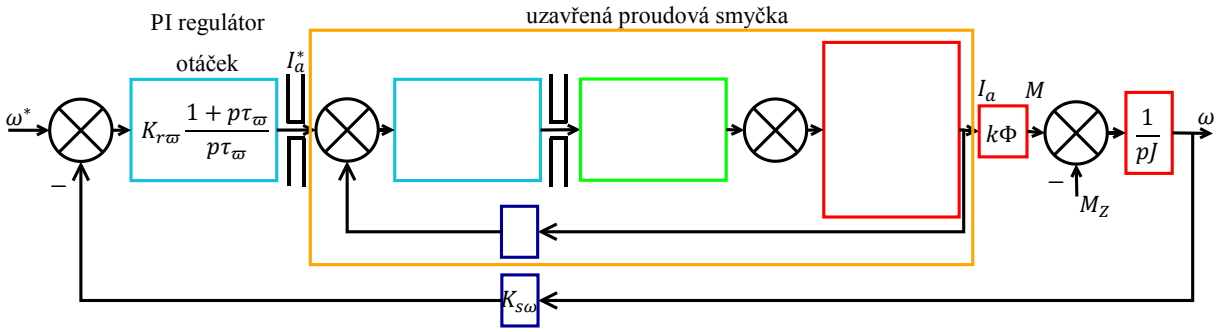
Postup je obdobný postupu návrhu regulátoru proudu.



Obr. 5.4: Regulátor otáček – návrh konstanty $\boldsymbol{K_{r\omega}}$

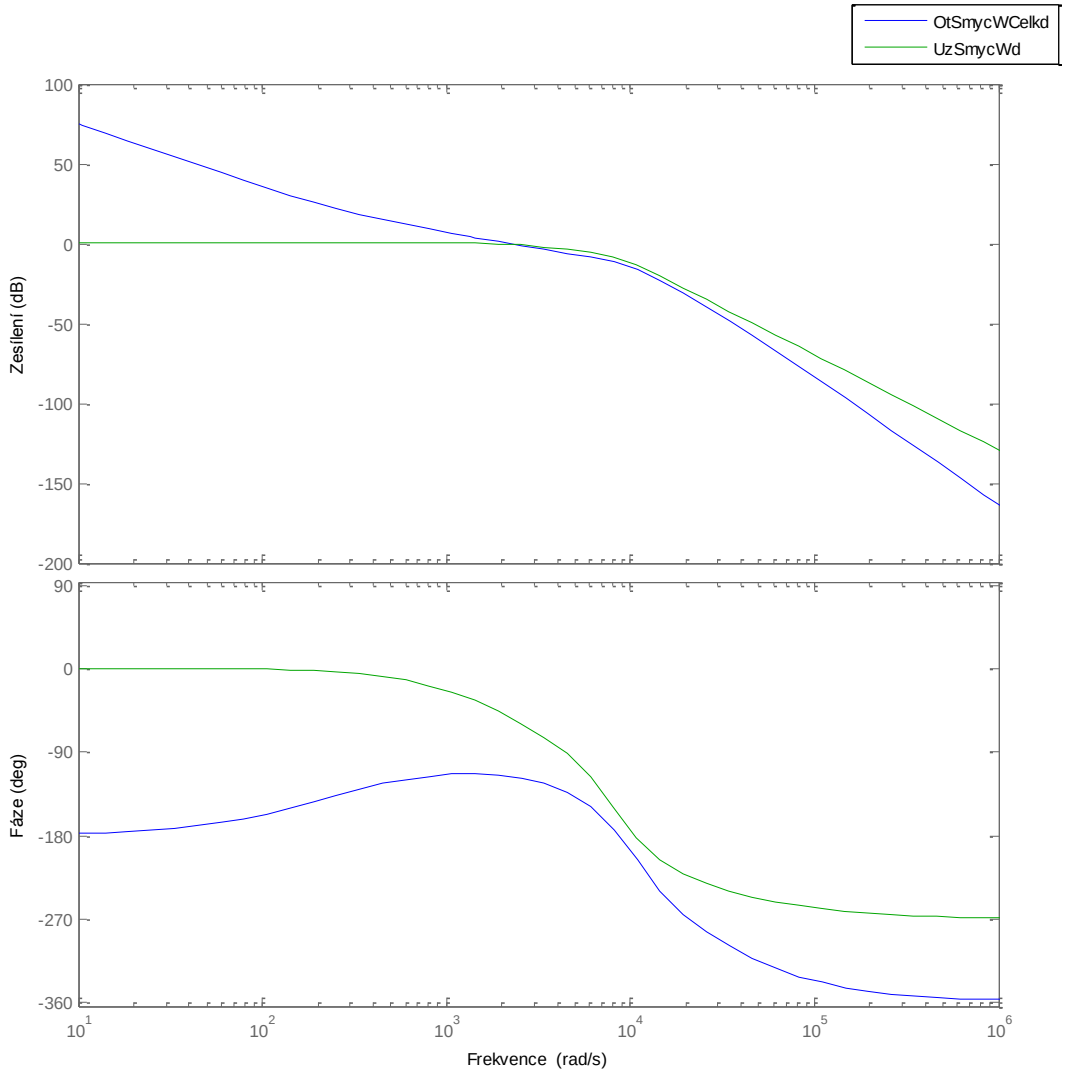
5.2.2 Uzavřená smyčka

Uzavřená smyčka regulátoru otáček je posledním krokem návrhu a příslušné blokové schéma Obr. 5.5 je identické s náhradním blokovým schématem pohonu Obr. 4.1.



Obr. 5.5: Uzavřená smyčka regulátoru otáček

5.2.3 Logaritmické frekvenční charakteristiky regulátoru otáček



Obr. 5.6: Logaritmické frekvenční charakteristiky regulátoru otáček

Legenda k Obr. 5.6:

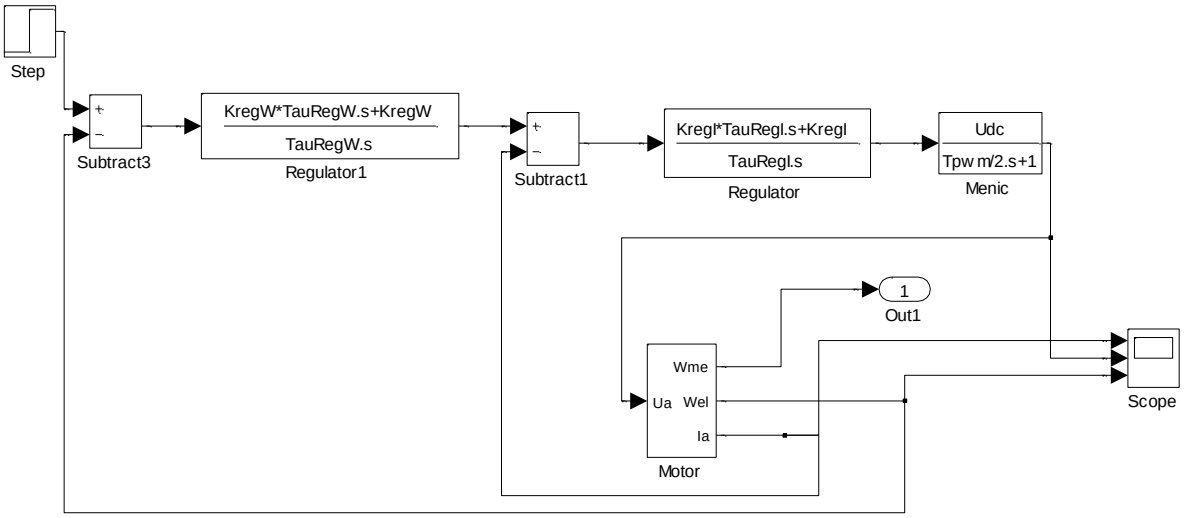
- modré křivky: amplitudová a fázová charakteristika otevřené smyčky pro regulátor otáček
- zelená křivky: amplitudová a fázová charakteristika uzavřené smyčky pro regulátor otáček

5.3 Postup metody MIL v prostředí Simulink

Následující kapitoly popisují jednotlivé kroky, které od obecného spojitého modelu směřují k diskrétnímu modelu, který počítá s reálnými parametry subsystémů regulačního řetězce.

5.3.1 Model 1

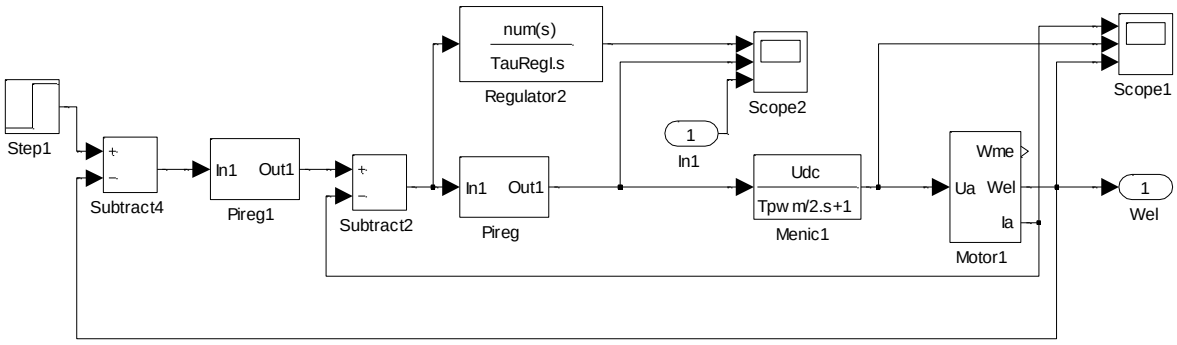
V prvním kroku je ověřena správnost navržených parametrů regulátoru, které jsou znázorněny v modelu na Obr. 5.7. V tuto chvíli není definováno žádné fyzikální omezení (např. napájecí napětí).



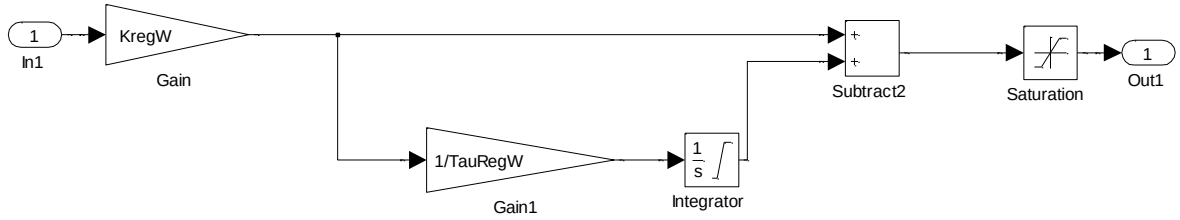
Obr. 5.7: Model 1 - spojitý model uzavřené smyčky bez fyzikálního omezení

5.3.2 Model 2

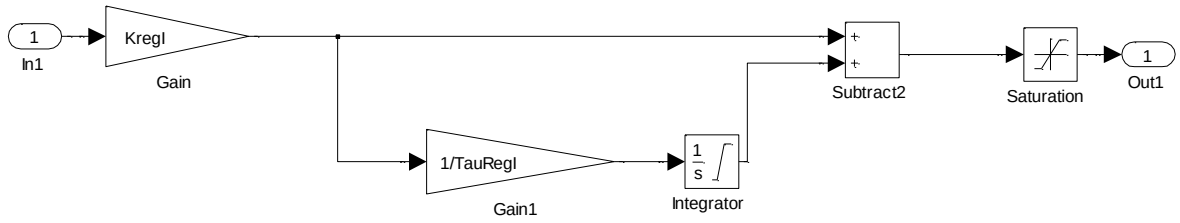
Tento model vychází z předchozího Modelu 1 – spojitého modelu uzavřené smyčky bez fyzikálního omezení. Rozdíl mezi modely je pouze v implementaci PI regulátorů otáček a proudu, jak je vidět na blokových schématech - Obr. 5.9 znázorňuje PI regulátor otáček a Obr. 5.10 potom regulátor proudu, kde jsou přidány integrátory. Model 2 (Obr. 5.8) je navržen již s omezeními realizovanými pomocí bloku saturace v jednotlivých regulátorech - omezení na jmenovitý proud $<+I_n, -I_n>$ u regulátoru otáček a <+1, -1> u regulátoru proudu.



Obr. 5.8: Model 2 - model s fyzikálním omezením



Obr. 5.9: PI regulátor otáček

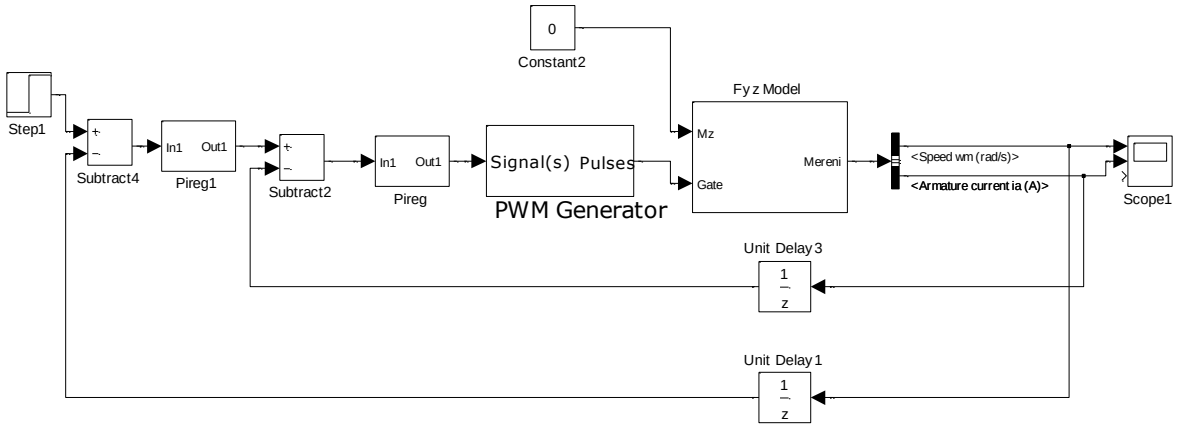


Obr. 5.10: PI regulátor proudu

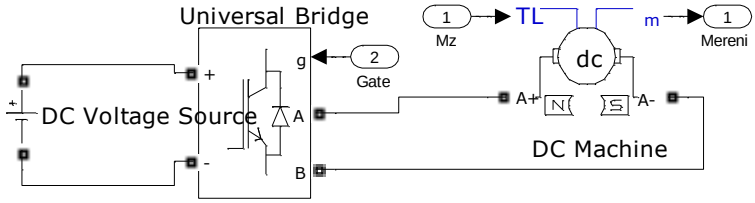
5.3.3 Model 3

V blokovém schématu pro Model 3 (Obr. 5.11) je zaveden blok pro fyzikální model, jehož schéma je znázorněno na Obr. 5.12. Blok fyzikálního modelu je tvořen stejnosměrným napěťovým zdrojem pro napájení, H-můstkem a ss motorem s permanentními magnety, ke kterému je přiveden zátěžový moment M_z=0.

Model 3 je realizován bez mrtvých časů a úbytku na měniči. Je zde také přidán blok pro simulaci PWM přivedený do *Gate* fyzikálního modelu.



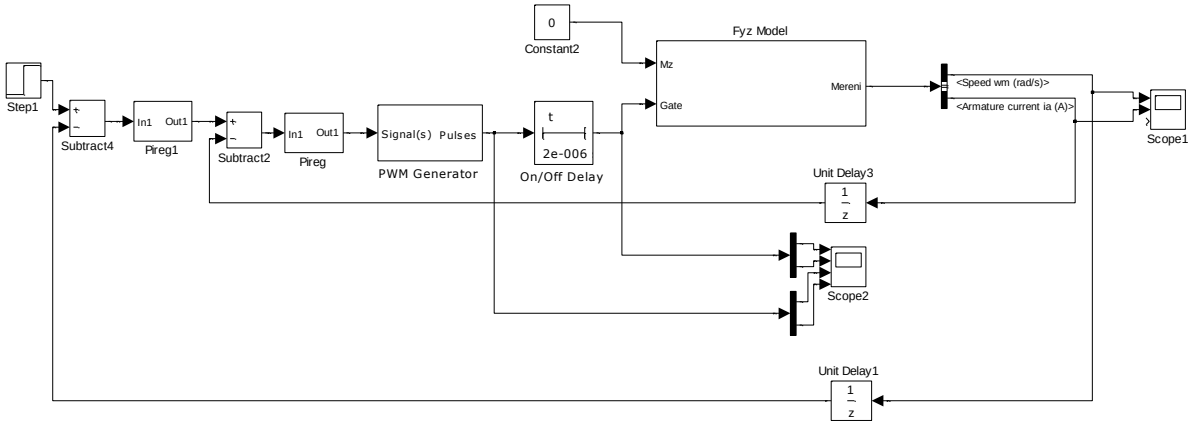
Obr. 5.11: Model 3 - fyzikální model bez mrtvých časů a úbytků na měniči



Obr. 5.12: Blok fyzikálního modelu

5.3.4 Model 4

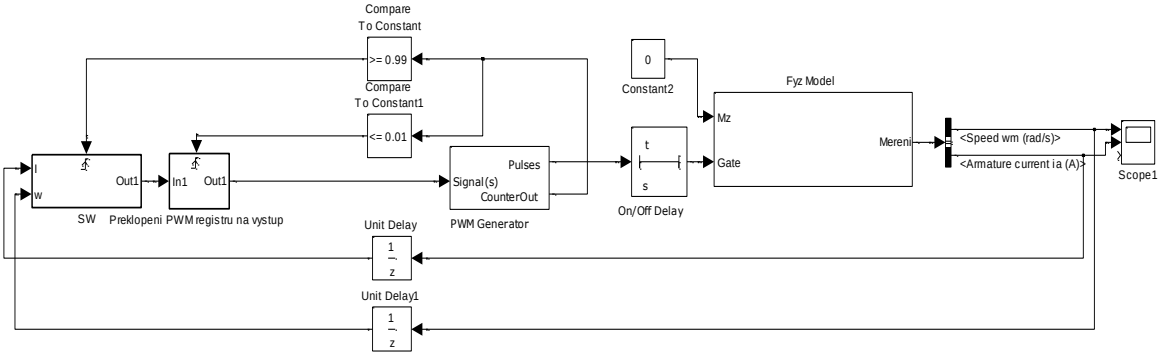
Model 4 (Obr. 5.13) je rozšířen o blok nastavující mrtvé časy (*On/Off Delay*) před blokem fyzikálního modelu. Nastavení těchto mrtvých časů zabraňuje současnému sepnutí dvou větví H-můstku a tedy vzniku větvového zkratového proudu – ten by totiž mohl vést k nenávratnému poškození spínacích součástek. V H-můstku vznikají napěťové úbytky na tranzistoru a diodě.



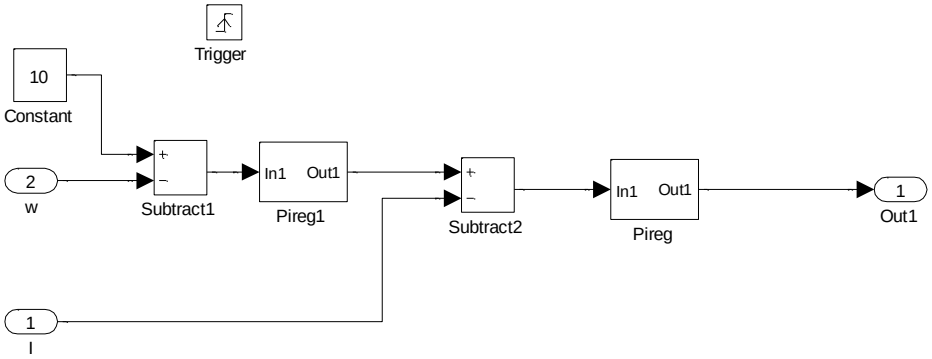
Obr. 5.13: Model 4 - fyzikální model s mrtvými časy a úbytkem na měniči

5.3.5 Model 5

Model 5 (Obr. 5.14) je nutné zdiskretizovat pro synchronizaci s PWM a A/D převodníkem – původní spojitá integrace v regulátorech proudu a otáček není možná v triggerovaném subsystému. Ve vytvořeném subsystému SW s *triggerem* je vidět, že sestává z diskretizovaných regulátorů proudu a otáček - Obr. 5.15, a v těchto blocích je původní integrátor změněn na diskrétní formu integrace (metoda Euler). *Trigger* je nastaven tak, že reaguje na náběžnou hranu. Další triggerovaný blok zajišťuje překlopení PWM registru na výstup.



Obr. 5.14: Model 5 - model s diskrétním vzorkováním



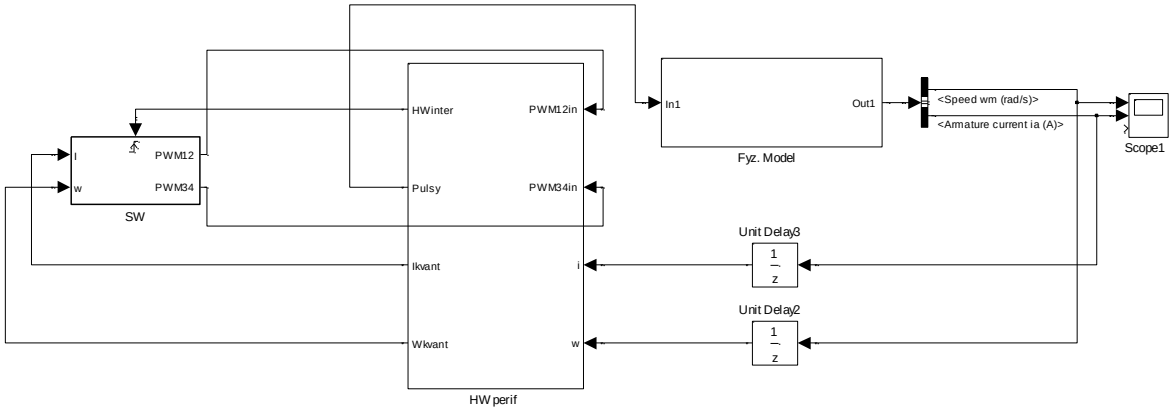
Obr. 5.15: Blok SW

5.3.6 Model 6

V tomto kroku jsou všechny bloky převedeny na aritmetiku v pevné řádové čárce - *fixed point*. Triggerovaný subsystém SW je schopen stimulovat vstupy pro PWM generátor (Obr. 5.17). Nově je přidán blok pro nastavení HW periférií (Obr. 5.18).

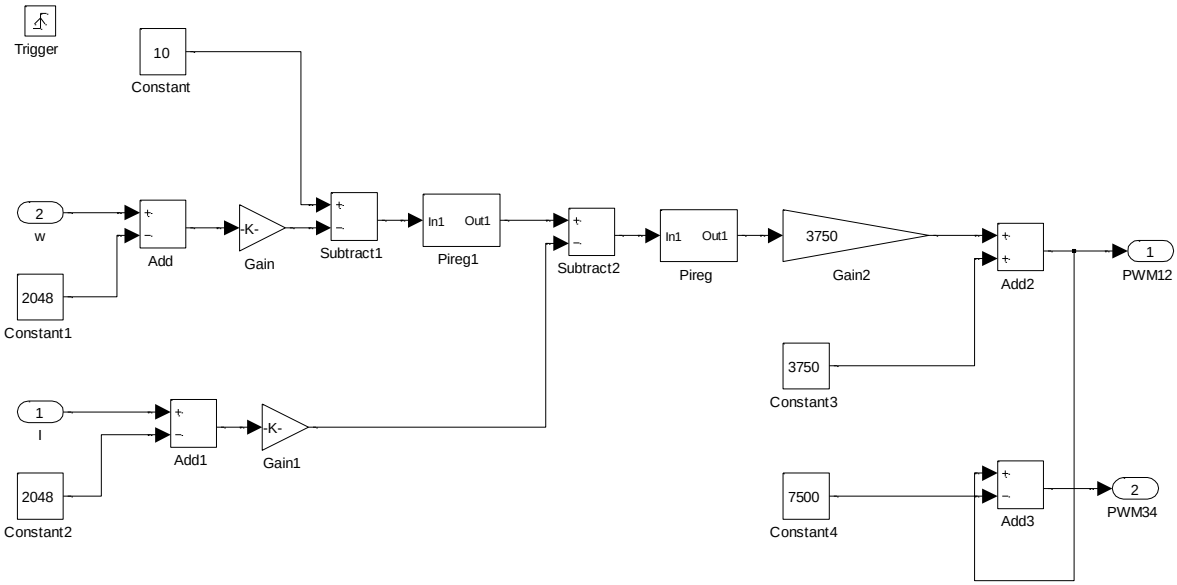
Je možné zjistit, jak velká je kvantizační chyba (plynoucí ze vzorkování v Modelu 5). Kvantizační chyba se např. projevila u použitého tachodynama s výstupním napětím (po převodu vstupními děliči) 3,3 [V] při jmenovitých 4000 [rpm]

PWM signál může nabývat hodnot 0 až 7500. Hodnoty vychází z rozlišení PWM bloku v mikrokontroléru TMS320F2812, který pracuje na poloviční frekvenci procesoru, tedy 150 / 2 MHz = 75 MHz, a z frekvence regulace 10 kHz.



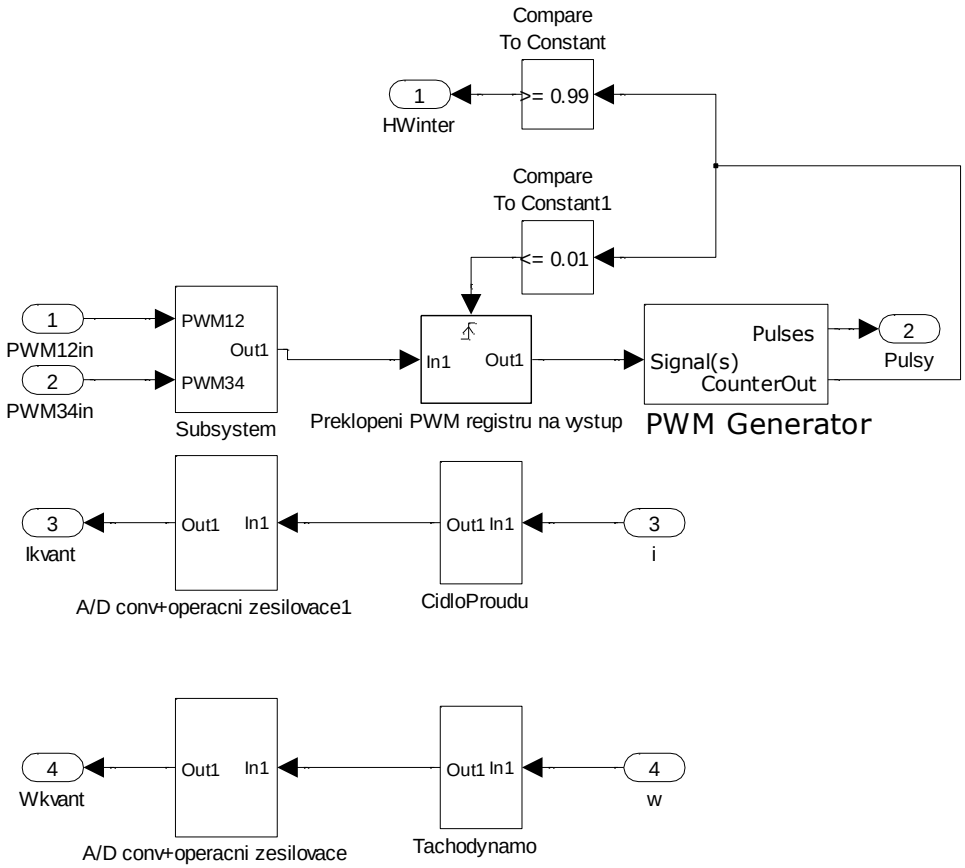
Obr. 5.16: Model 6 - model s HW periferiemi

Následující blokové schéma Obr. 5.17 popisuje regulační algoritmus, který je v identické formě použit v kódu pro finální krok Model 7 metody MIL a u metod SIL a PIL.



Obr. 5.17: Blok SW

Hardwarové periférie jsou znázorněny blokovým schématem Obr. 5.18. Parametry vstupních periférií pro měření jsou popsány v kapitole 4.4. Popisu parametrů PWM bloku je věnován začátek této kapitoly 5.3.6.

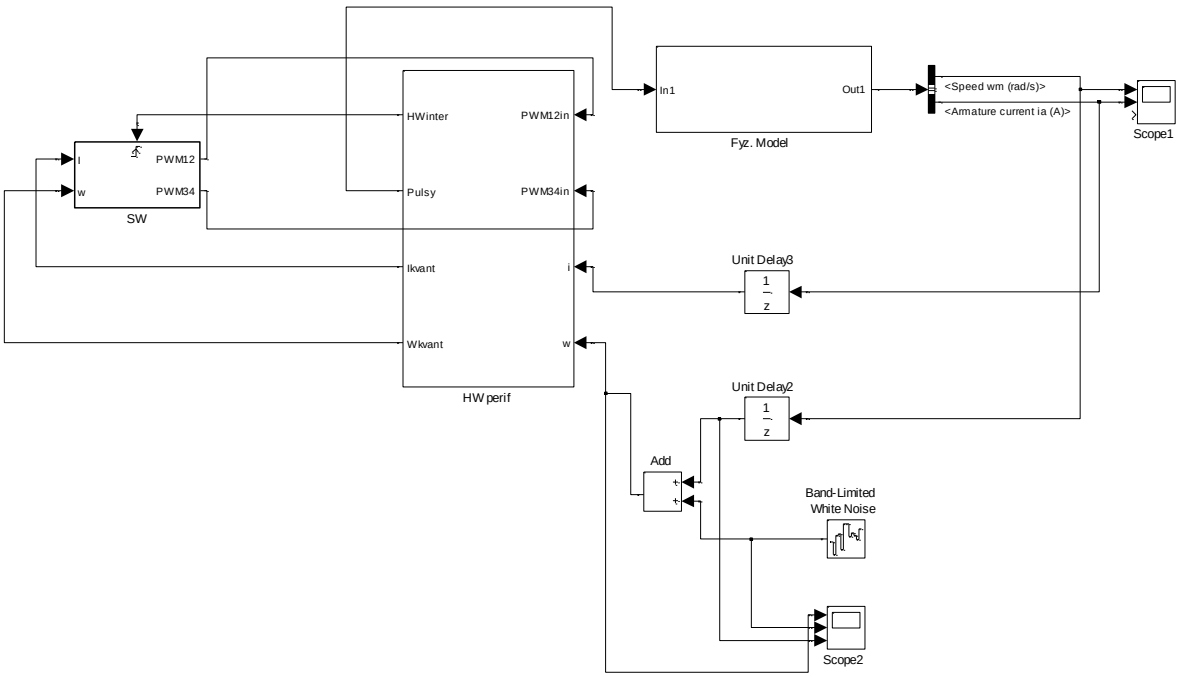


Obr. 5.18: HW periférie

5.3.7 Model 7

Do Modelu 7 je přidán šum, který byl získán měřením (např. tachodynama). Pro potlačení šumu je nutné přidat *Moving-Average filter*. Pro zachování fázové bezpečnosti byla snížena zesílení bloků v regulačním řetězci. Následným měřením bylo ověřeno, že vložení filtru nemá vliv (stabilita apod.) na regulační řetězec.

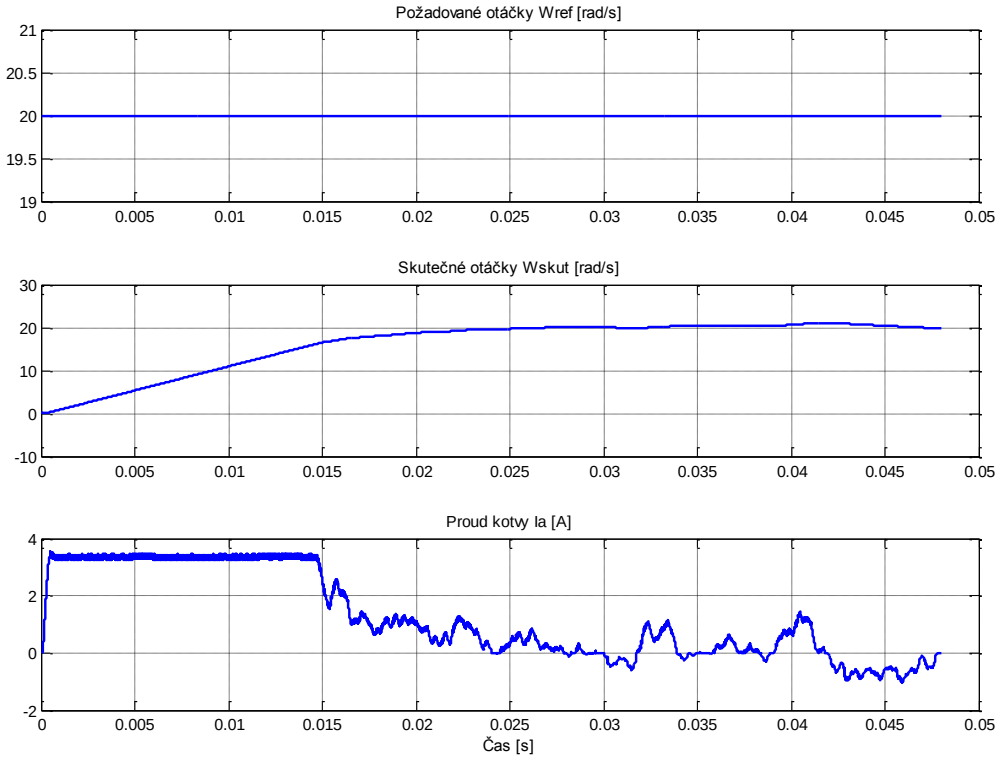
SS motor s PM má malou indukčnost L_a a velký odpor R_a, tedy malou časovou konstantu → proud by rychle narůstal, v PI regulátoru je dominantní integrační složka pro omezení dynamiky (rychlosti) změn proudu.



Obr. 5.19: Model 7 - model s *Moving-Average* filtrem

5.3.8 Odezva na jednotkový skok – regulace na konstantní otáčky

Pro ověření správnosti návrhu a implementace metody MIL je třeba pro výsledný návrh spustit simulaci. Pro tento účel byla do Modelu 7 přidána konstantní požadovaná rychlost $\omega_{\mathrm{ref}} = 20\,\mathrm{rad/s}$. Ve výsledku konečné simulace (Obr. 5.20) metody MIL je vidět zadaná konstantní rychlost a je zde změřena skutečná rychlost, která se po odeznění přechodového jevu ustálí kolem požadované hodnoty a proud kotvy. Tímto krokem bylo ověřeno, že postup návrhu u metody MIL byl správný.



Obr. 5.20: Regulace na požadované otáčky v MIL modelu 7

6 Návrh a implementace metody SIL a PIL v prostředí Simulink

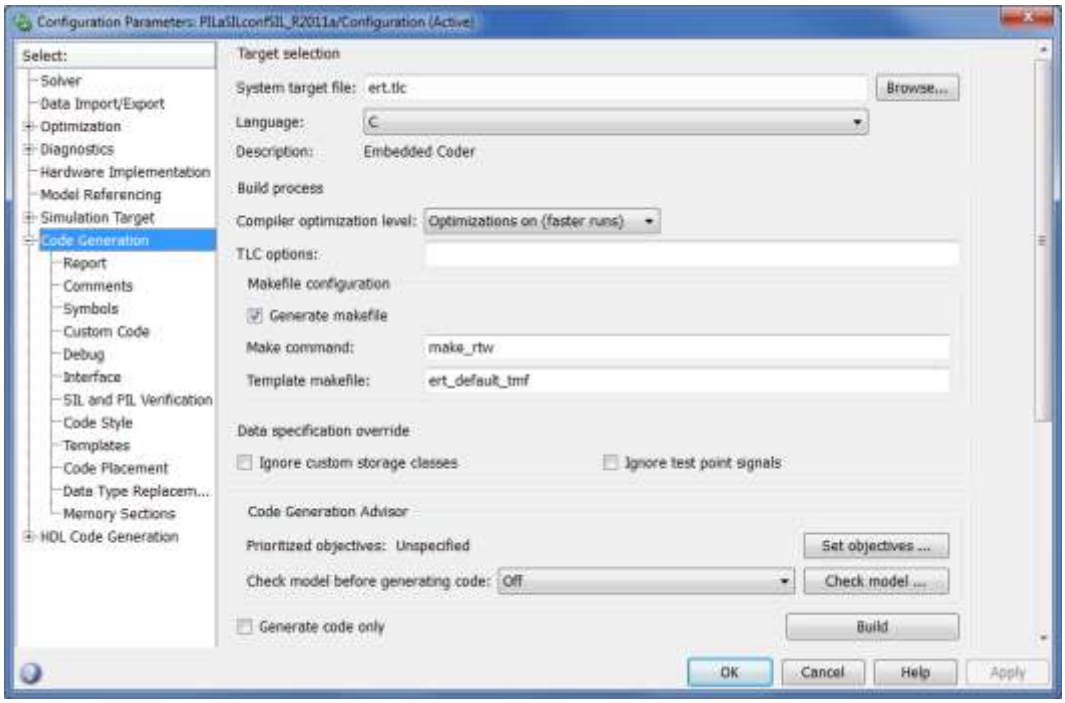
Regulační algoritmus, tzv. blok SW vytvořený v kapitole 5.3.6 (viz Obr. 5.17) je použit v návrhu obou metod SIL i PIL.

Následující kapitoly popisují specifická nastavení pro každou z metod a na závěr této kapitoly jsou ověřeny identické výsledky metod SIL a PIL.

6.1 Nastavení prostředí Simulink pro SIL

Pro návrh a implementaci metody je nejprve nutné vygenerovat SIL blok, který je tvořen S-funkcí. Po otevření Modelu 7 je výhodné pro zjednodušení a přehlednost zkopírovat blok SW do nového okna, kde je nezbytné upravit jeho vstupy a výstupy dle původního modelu (fyzikální model s přidaným *Moving-Average* filtrem) a poté je postup následující:

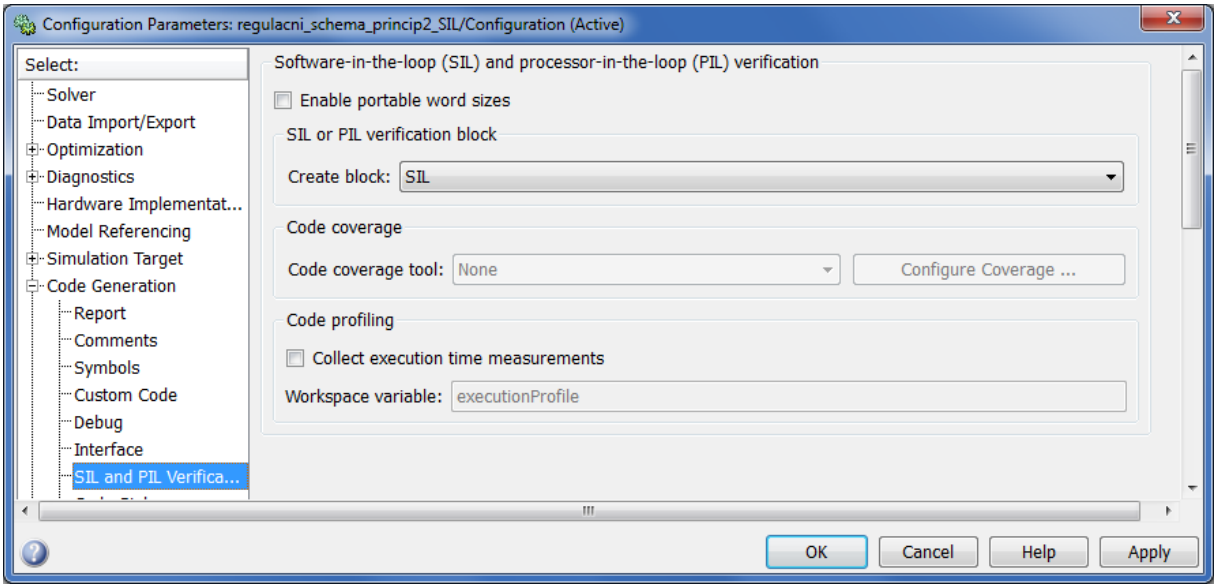
- po kliknutí pravým tlačítkem se otevře výběrové menu, kde se nachází volba *Configuration Parametres*
- po otevření dialogového okna je důležité se zaměřit na položku *Code Generation* a v tabulce *Target selection* je nutné nejprve zvolit správný *System target file*, což je v tomto případě ert.tlc - Obr. 6.1



Obr. 6.1: Výběr *System target file* – metoda SIL

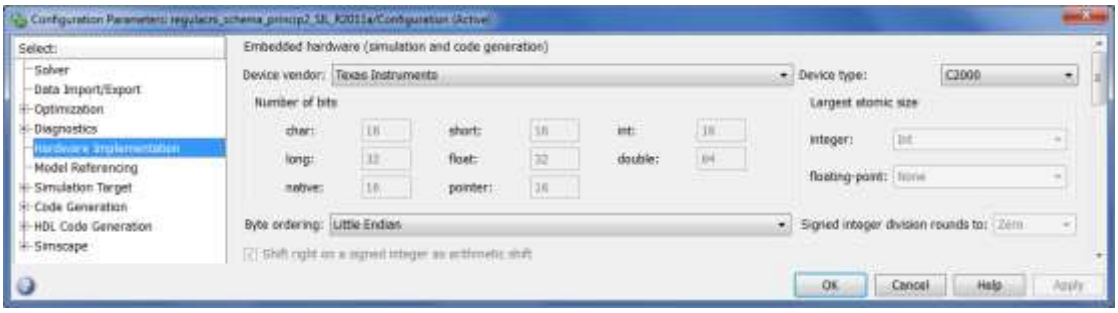
- poté se v *Code Generation* vytvoří několik nových položek a jejich možností nastavení včetně *SIL and PIL verification*

- v rozbalovacím menu této položky je vybrána možnost SIL - Obr. 6.2



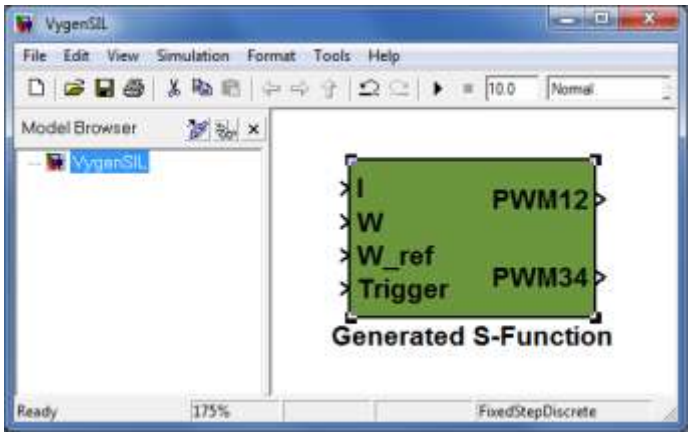
Obr. 6.2: Vytvoření bloku SIL

- dále je nutné v položce *Hardware Implementation -> Embedded hardware (simulation and code generation)* nastavit správně *Device vendor* (dodavatele procesoru) a to Texas Instruments a *Device type* (typ procesoru), což je C2000. *Byte ordering* (pořadí bajtů) je nastaveno jako *Little Endian* - Obr. 6.3. To znamená, že se na paměťové místo s nejnižší adresou uloží nejméně významný bajt (LSB – *Last Significant Byte*) a za něj se ukládají ostatní bajty až po nejvíce významný bajt (MSB - *Most Significant Byte*).



Obr. 6.3: Hardware implementace SIL

- v *Code Generation* následuje volba *Build* a tímto je vytvořen SIL blok, který vypadá následovně:

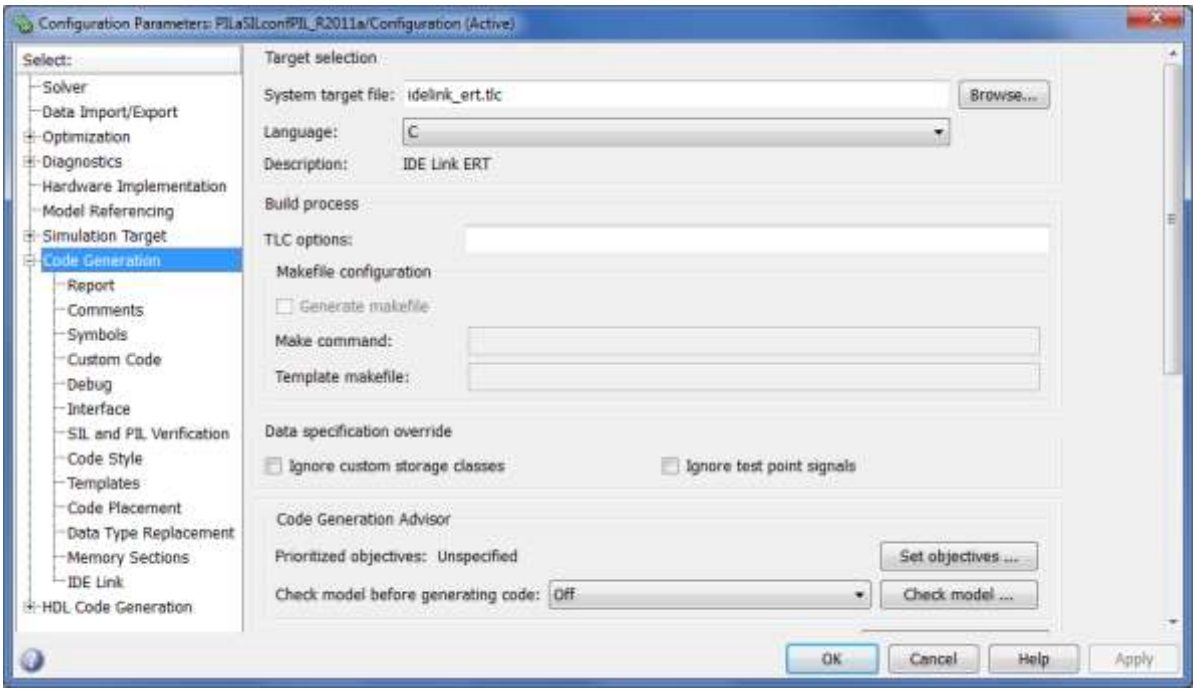


Obr. 6.4: Vygenerovaný SIL blok

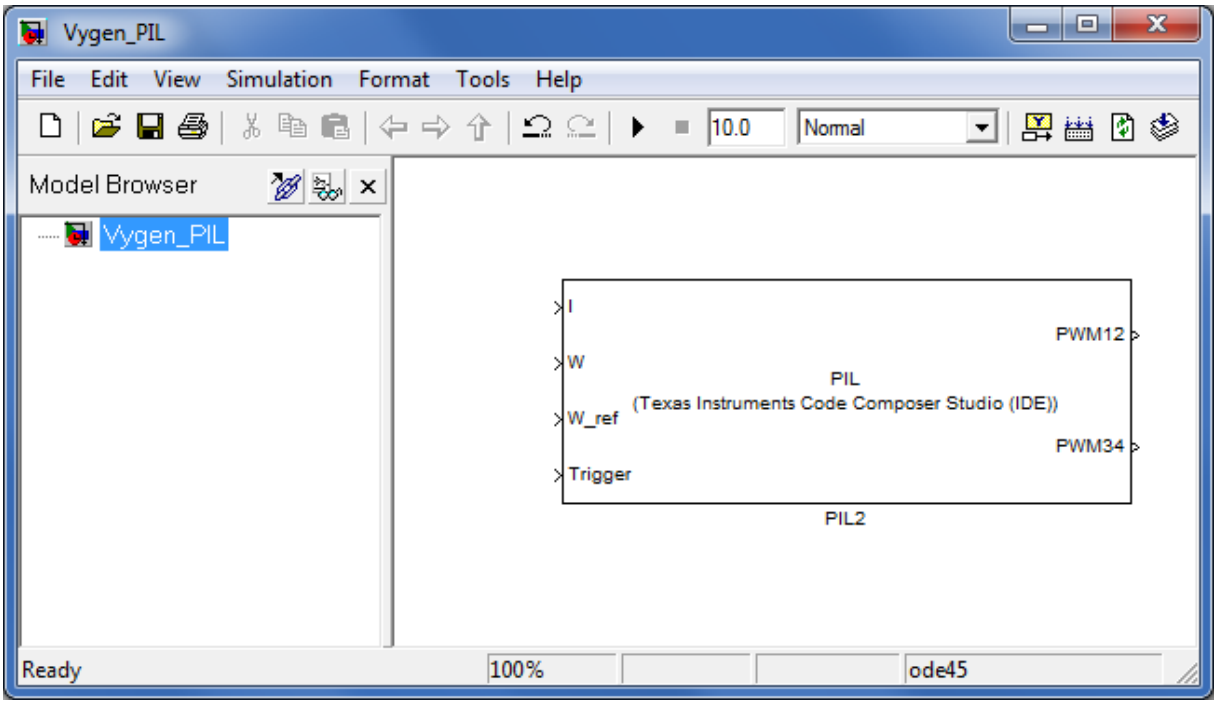
Tímto nově vygenerovaným SIL blokem s S-funkcí je následně nahrazen původní blok SW v Modelu 7, na Obr. 6.4 jsou názorně vidět vstupy a výstupy nového bloku, které se nijak neliší od původního bloku je tedy jednoduché původní blok nahradit. Po tomto kroku je návrh metody SIL dokončen.

6.2 Nastavení prostředí Simulink pro PIL

U metody PIL musí být také nejprve vygenerován PIL blok. Postup pro generování PIL bloku je stejný jako u SIL, liší se pouze ve výběru možnosti v *Code Generation-> SIL and PIL verification* a poté je vybrána možnost PIL. Zároveň musí být vybrána správná možnost *System target file* – tedy idelink_ert.tlc.



Obr. 6.5: Výběr *System target file* - metoda PIL



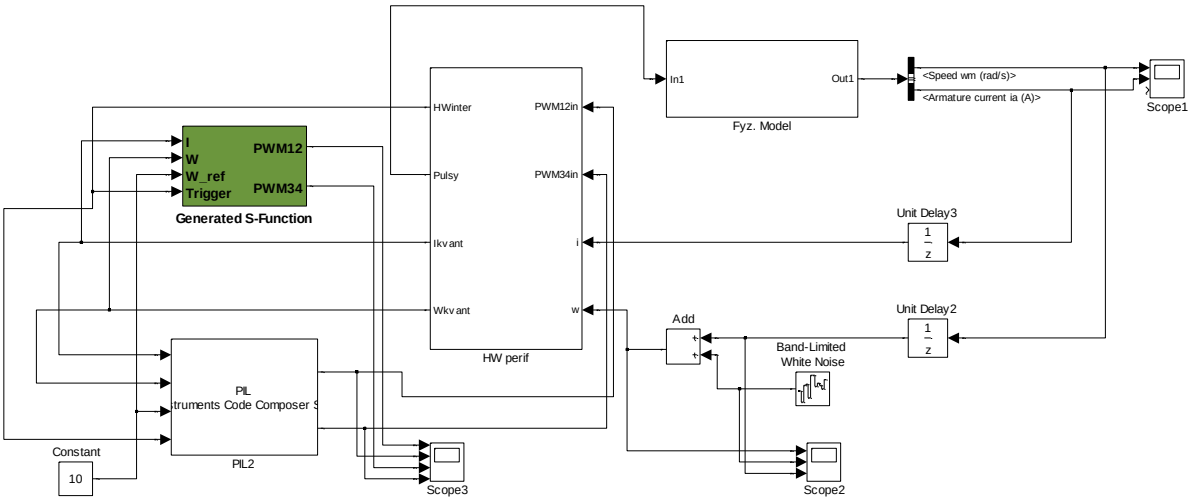
Obr. 6.6: Vygenerovaný PIL blok

6.2.1 Hardware cílové platformy pro spuštění kódu metody PIL

Prototypová deska eZdsp™ F2812 s cílovým procesoru TMS320F2812 je propojena se Simulink modelem (výpočty simulace) JTAG protokolem, v tomto případě přenášeném přes paralelní port. Uživatelská funkce v kódu běžícím na TMS320F2812 je spouštěna, pokud náběžná hrana taktu simulace (např. 1 MHz) je souběžná s náběžnou hranou taktu regulace 10 kHz. Jinými slovy, uživatelská funkce částečně běží v reálném čase, po krocích. Mezi kroky spuštění provádí simulace běžící na PC analýzu naměřených hodnot a výpočet vstupních hodnot pro procesor TMS320F2812.

6.3 Ověření modelu pomocí metod SIL a PIL

Po vygenerování SIL a PIL bloku a po následném přidání těchto bloků do původního modelu vypadá konečný model připravený pro obě metody takto Obr. 6.7 - je zřejmé, že zeleně podbarvený blok je SIL blok s vygenerovanou S-funkcí a pod ním se nachází PIL blok. Jak už bylo popsáno v předchozím textu, tyto bloky se vzájemně liší především nastavením *System target file.*



Obr. 6.7: Výsledná podoba modelu s bloky SIL a PIL

Po vytvoření bloků SIL a PIL bylo nutné znovu ověřit původní simulaci MIL Modelu 7. Výsledky byly změřeny dle očekávání stejné, jako v předchozím případě, je tedy zaručeno, že bloky SIL a PIL neměly vliv na výsledky simulace. Tímto je zároveň ověřena funkčnost kódu pro metodu SIL i PIL.

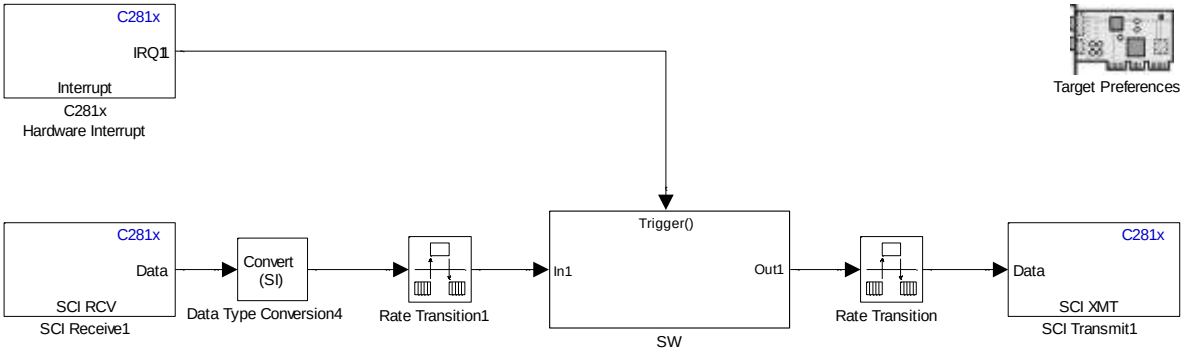
7 Návrh a implementace metody HIL v prostředí Simulink

Metoda HIL lze provozovat např. na platformě dSpace či LabVIEW, toto provedení však klade vysoké finanční a technické nároky na použitý HW, tak nezbytný SW. Další možností je simulaci HIL vyřešit vlastním vývojovým kitem – viz následující kapitola 7.1.

Výsledná simulace HIL nemusí dosáhnout stejných výsledků metody SIL a PIL, neboť pro metodu HIL je řízená soustava *real-time* simulací, nikoliv model běžící v Simulinku.

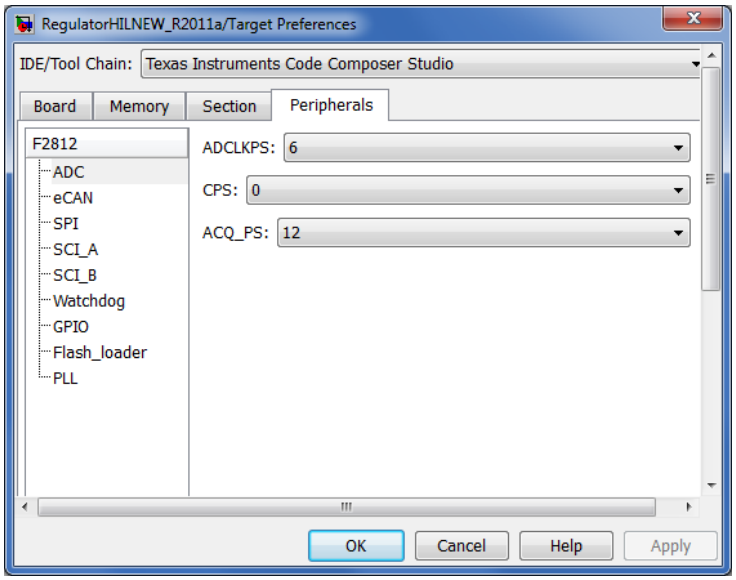
Nicméně pro implementaci metody HIL je použit téměř identický kód (viz blok SW v Obr. 7.1), který bude oficiálně vydán do produkce. Rozdíly SW pro HIL a finálního produkčního SW jsou v přepočtech vstupních veličin. Produkční SW čte a generuje signály fyzikálních veličin různých rozsahů. U metody HIL předpokládáme rozsah a velikost signálů mezi řídicí a řízenou soustavou, které jsou dány např. způsobem spojení – viz další kapitola 7.1 popisuje vzájemné signálové propojení přes D/A a A/D převodníky.

Na Obr. 7.1 je principiálně znázorněno propojení Simulinku (požadované otáčky) - PC prostřednictvím sériové linky SCI (rychlostí 115,2 [kBaud] každých 10 [ms]), dále na mikrokontrolér TMS320F2812 a zpět do Simulinku pro zobrazení a změření skutečných otáček, proudu a napětí regulátory.



Obr. 7.1: Propojení Simulinku a PC pře sériovou linku SCI

K tomuto modelu je nutno přidat kartu *Target Preferencies* , ve které jsou nastavovány vlastnosti a periférie desky s TMS320F2812, jak je vidět na Obr. 7.2. V záložce *Board ->Board Properties* je možné nastavit druh desky a *CPU clock*. V záložce *Peripherials* je pak možno nastavovat např. vlastnosti A/D převodníku.

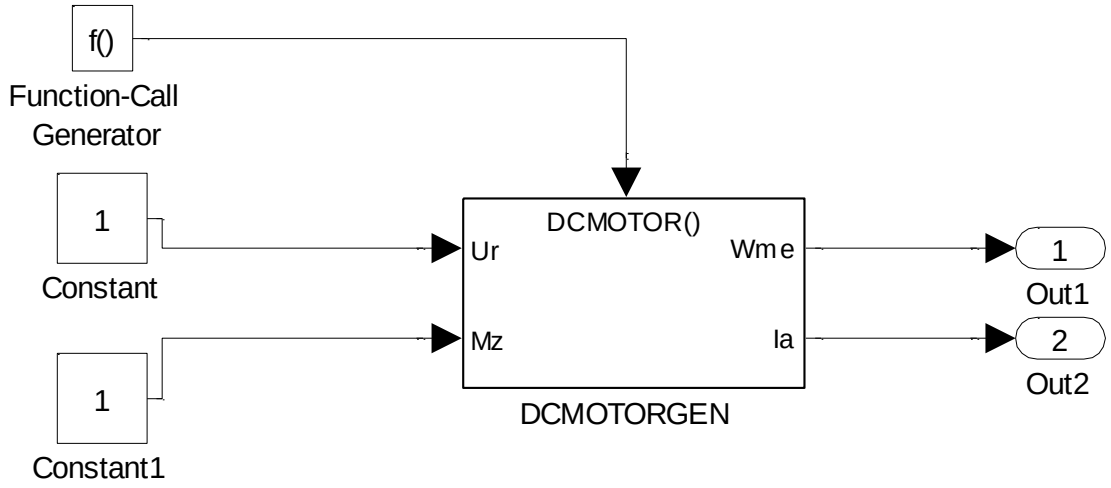


Obr. 7.2: Nastavení Target Preferencies

7.1 Hardwarová simulace řízené soustavy na vývojovém kitu

Program pro simulaci řízené soustavy na vývojovém kitu byl automaticky vygenerován nástrojem Embedded Coder a sestaven překladačem CodeComposer od firmy Texas Instruments. Hardwarová simulace běží na miktokontroléru TMS320F28335 v aritmetice *floating point.* Další charakteristiky tohoto *real time* modelu jsou:

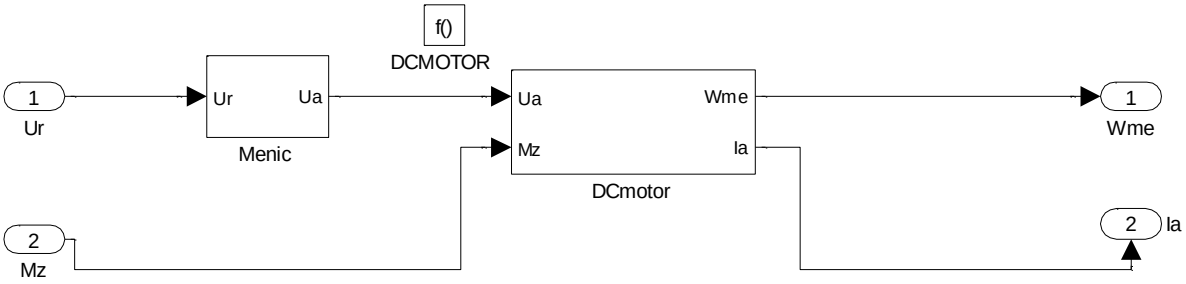
- řídicí systém představuje kit s mikrokontrolérem TMS320F2812, řízená soustava je realizována kitem s mikrokontrolérem TMS320F28335
- řídicí systém a řízená soustava mají vytvořeny vzájemné signálové propojení přes D/A a A/D převodníky
- model řízené soustavy v TMS320F28335 pracuje s krokem 25 [μs], tedy na frekvenci 40 [kHz]
- kód řídicího systému v TMS320F2812 - oproti metodě PIL je zde navíc pouze obsluha periférií jako např. A/D a D/A převodníků
- u metody HIL není řídicí signál modulován PWM z důvodu vysoké náročnosti zpracování v HW modelu řízené soustavy, do TMS320F28335 je přímo zavedeno nemodulované řídicí napětí
- 2 HW interrupty v TMS320F2812 – interrupt od ADC (dokončení převodu vzorku) a interrupt od timeru pro SCI komunikaci s PC (Simulinkem)



Obr. 7.3: Blokové schéma řízené soustavy

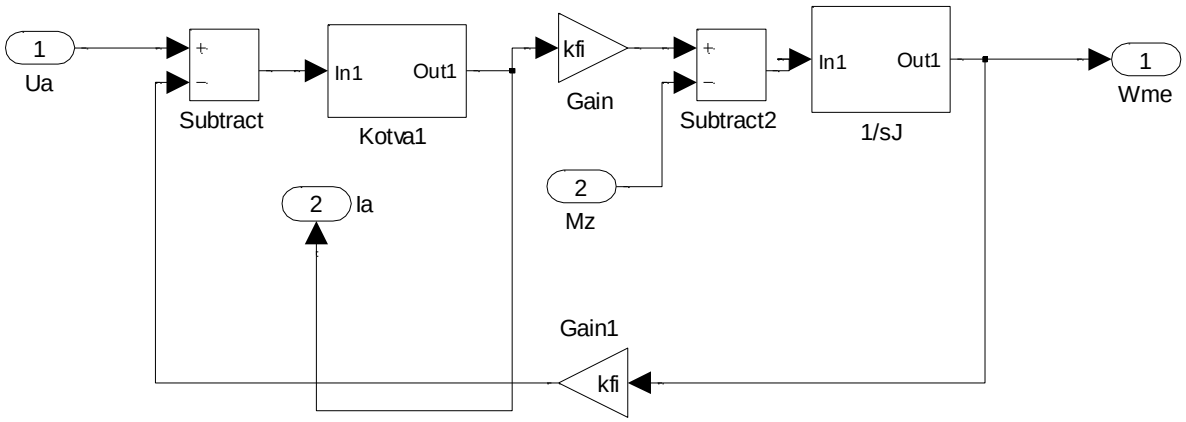
Obr. 7.3 znázorňuje princip simulace HIL, na vstup do ss motoru - kitu s TMS320F28335, přivádíme řídicí napětí $U_{\check{R}}$ a zátěžový moment M_z. Z motoru pak vystupují skutečné otáčky ω_{skut} a proud kotvou I_a.

Výhoda využití vývojového kitu pro řídicí systém a řízenou soustavu spočívá mimo jiné v tom, že můžeme simulovat takové podmínky, které by s reálným HW nebyly možné (např. simulace zátěžového momentu M_z). Na Obr. 7.4 je znázorněn blok řízené soustavy s TMS320F28335, který simuluje měnič a stejnosměrný motor s permanentními magnety.

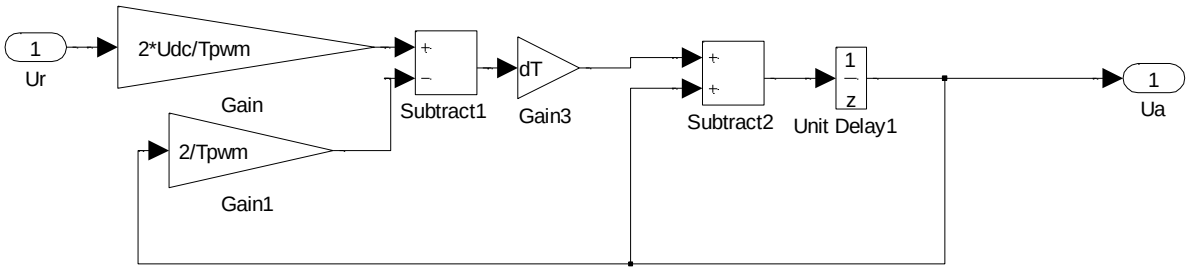


Obr. 7.4: Subsystémy v bloku DCMOTORGEN

Subsystémy bloku DCMOTORGEN lze diskrétně popsat blokovými schématy, jak naznačují následující obrázky Obr. 7.5a Obr. 7.6.



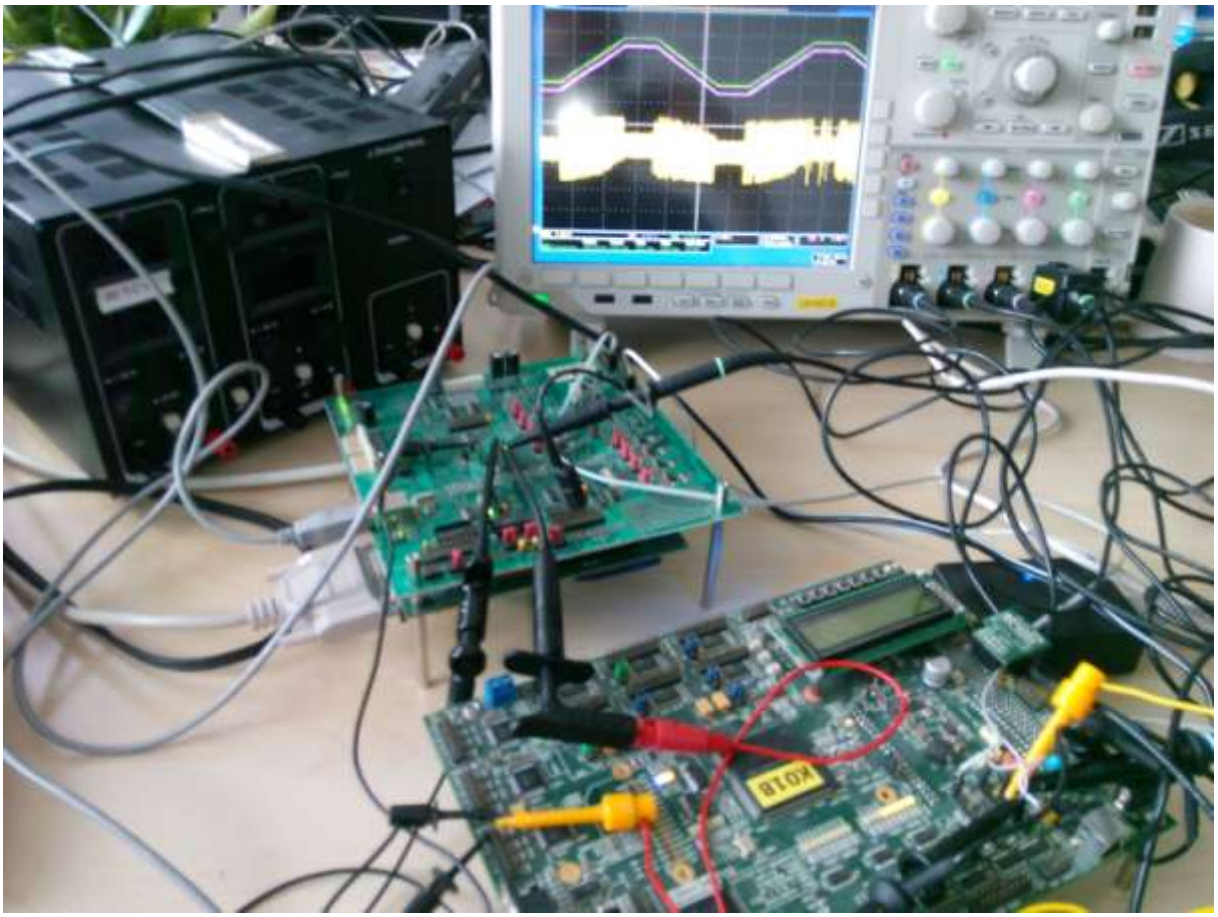
Obr. 7.5: Blok DC motoru



Obr. 7.6: Blok měniče

7.2 Laboratorní provedení metody HIL

V popředí fotografie Obr. 7.7 je vidět model řízené soustavy v TMS320F28335 a v pozadí pak model řídicí soustavy TMS320F2812, které jsou vzájemně propojeny přes D/A a A/D převodníky. Osciloskop zobrazuje průběhy otáček požadovaných ω_{REF}, otáček skutečných ω_{skut} a proud kotvou I_{a}. Řízená soustava je propojena přes sériovou linku přímo do PC (Simulinku).

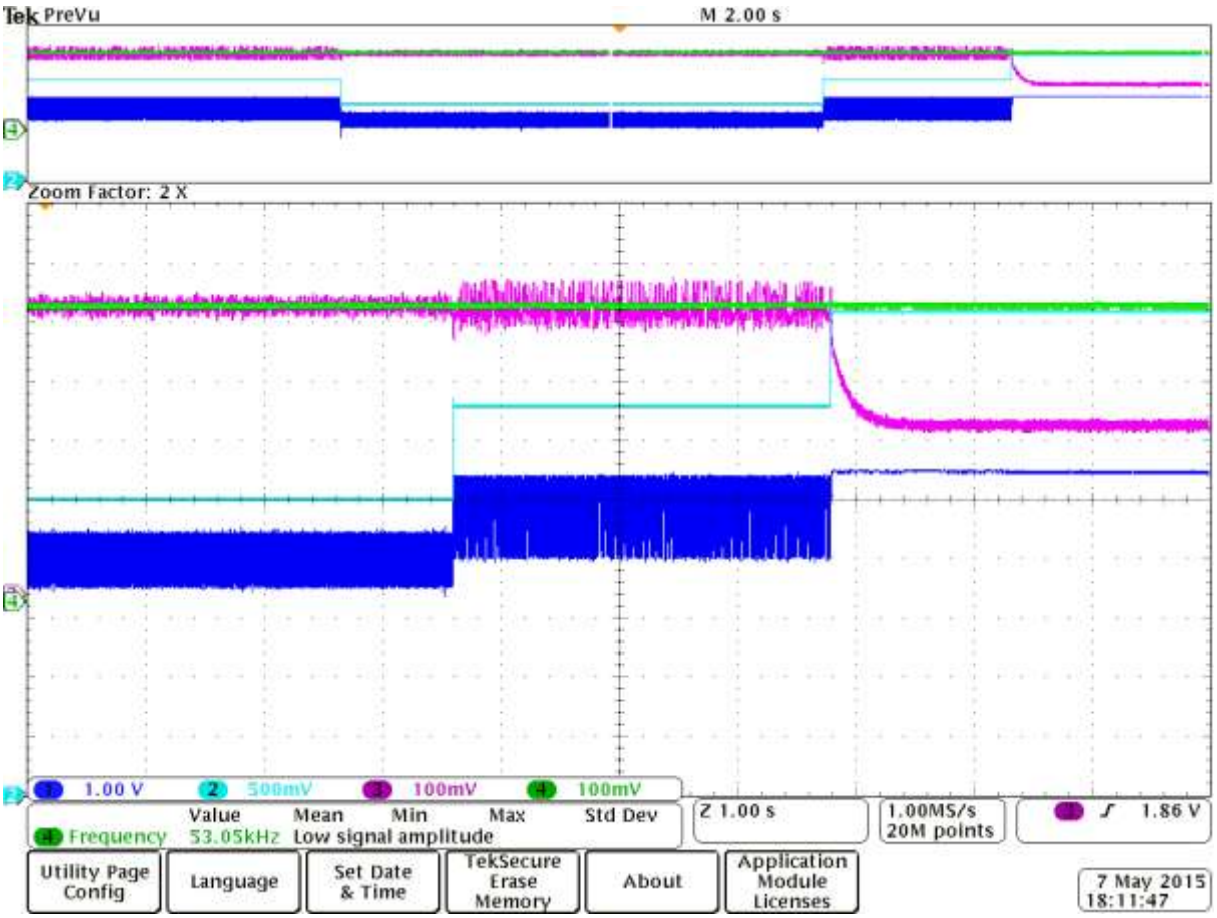


Obr. 7.7: Fotografie z praktického provedení metody HIL

7.3 Měření zátěžového momentu na HIL sestavě

Následující oscilogram Obr. 7.8 zobrazuje výsledku měření zátěžového momentu na HIL sestavě, která je popsána v předchozí kapitole 7.2.

Jsou zadány konstantní požadované otáčky ω_{REF} = 50 [rads^{-1}]. Měření startuje s nulovým zátěžovým momentem M_{z}, poté je zátěžový moment skokem zvýšen na M_{z}= 0,4 [Nm]. Konečná skoková změna je na hodnotu M_{z}= 0,8 [Nm]. Pro takový zadaný zátěžový moment je výkon motoru nedostačující a skutečné otáčky z požadované hodnoty 50 [rads^{-1}] poklesnou na 30 [rads^{-1}]. V této části charakteristiky, kdy je M_{z}= 0,8 [Nm], však dochází k nekorektnímu chování regulátoru. To je zřejmě dáno tím, že D/A převodník řízené soustavy v mikrokontroléru TMS320F28335, který poskytuje signál proudu I_{a}, dosahuje limitace a to právě +5 [V].



Obr. 7.8: Měření zátěžového momentu

Legenda k Obr. 7.8:

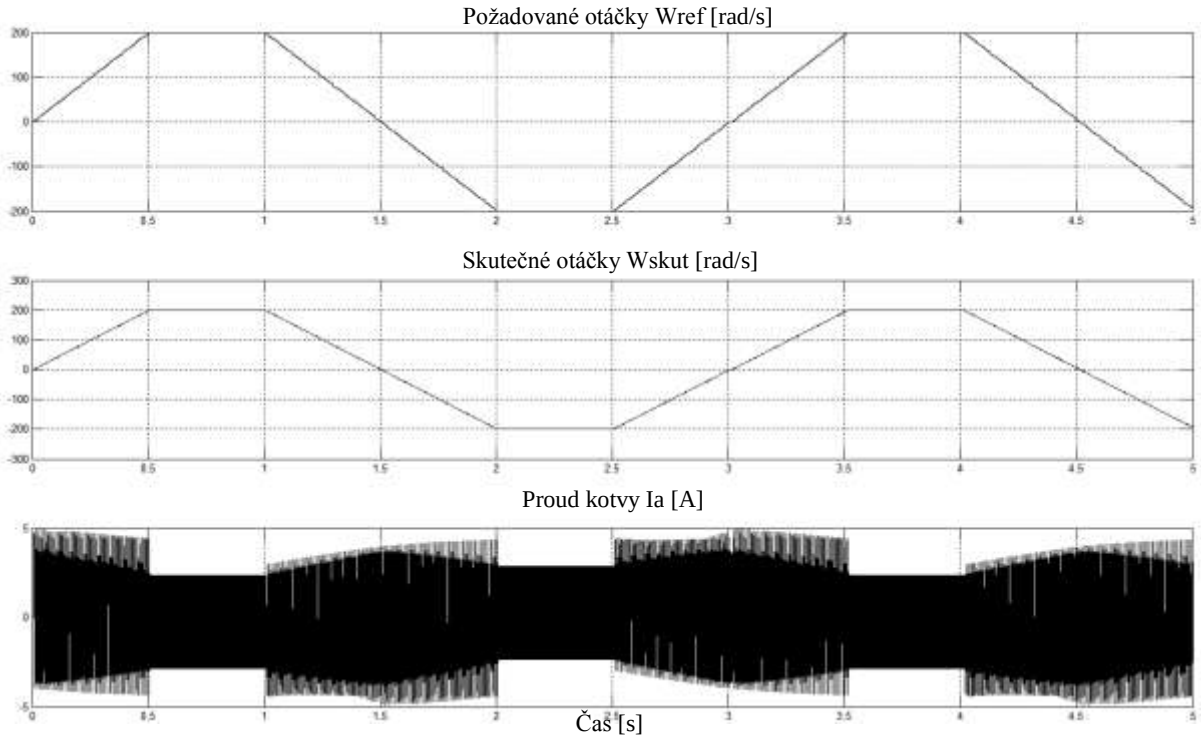
- časové měřítko: 1 [s] / dílek
- zelená: požadované otáčky ω_{REF}, měřítko 10 [rads^{-1}] / dílek
- fialová: skutečné otáčky ω_{skut}, měřítko 10 [rads^{-1}] / dílek
- modrá: proud kotvou I_{a}, měřítko 2,5 [A] / dílek
- tyrkysová: zátěžový moment M_{z}, měřítko 0,25 [Nm] / dílek

8 Experimentální ověření

Následující kapitoly srovnávají výsledky simulace a měření řídicích systémů, které byly navrženy různými metodami X In the Loop.

8.1 Měření lichoběžníkového průběhu otáček na MIL sestavě s konfigurací Model 7

Výsledky simulace pro zadaný lichoběžníkový průběh otáček jsou naznačeny na Obr. 8.1.



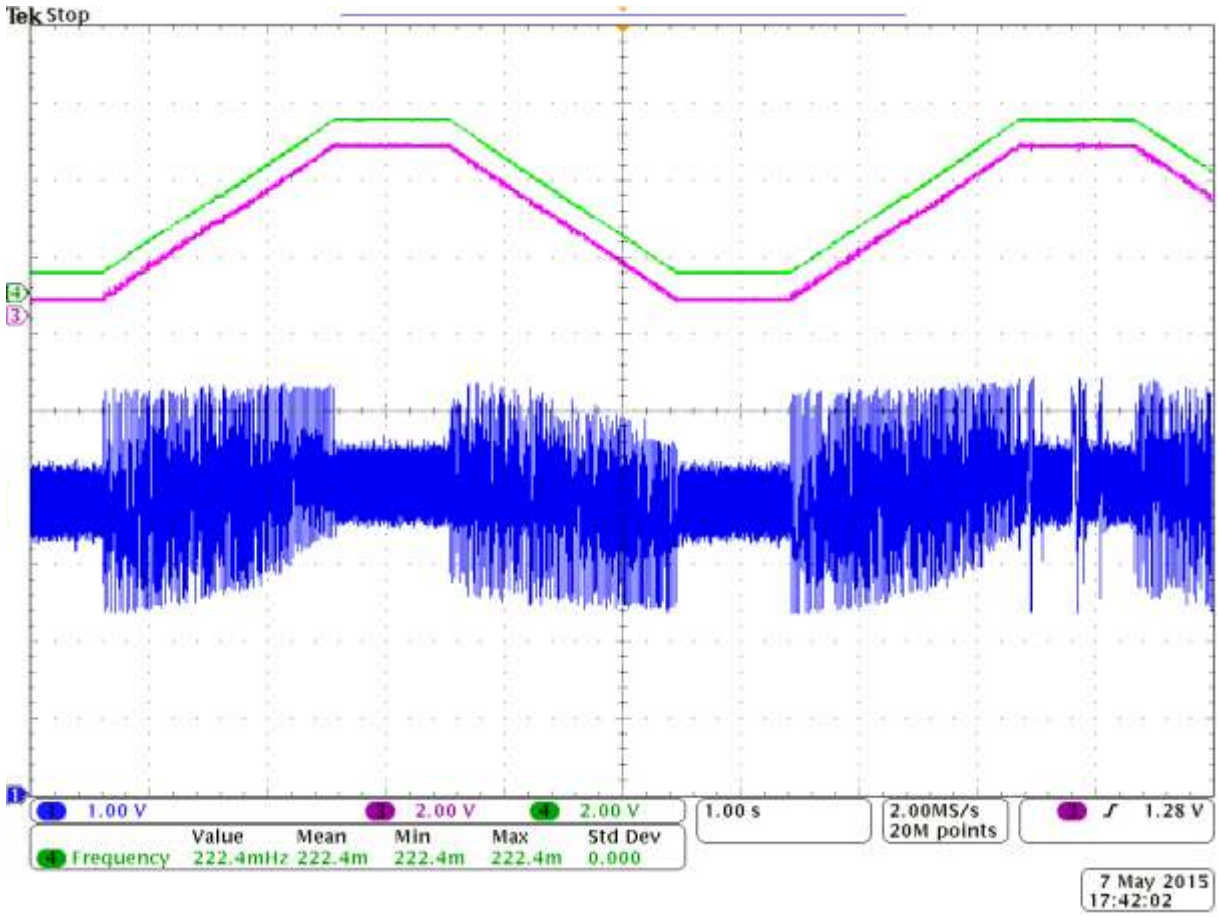
Obr. 8.1: Měření lichoběžníkového průběhu otáček na MIL modelu 7

Legenda k Obr. 8.1:

- časové měřítko: 0,5 [s] / dílek
- požadované otáčky ω_{REF}, měřítko 100 [rads^{-1}] / dílek
- skutečné otáčky ω_{skut}, měřítko 100 [rads^{-1}] / dílek
- proud kotvou I_{a}, měřítko 5 [A] / dílek

8.2 Měření lichoběžníkového průběhu otáček na HIL sestavě

Zaznamenaný oscilogram ukazuje reakci regulátoru na lichoběžníkovou změnu požadovaných otáček. Měření je provedeno na HIL sestavě, která je popsána v předchozí kapitole 7.2.



Obr. 8.2: Měření lichoběžníkového průběhu otáček na HIL sestavě

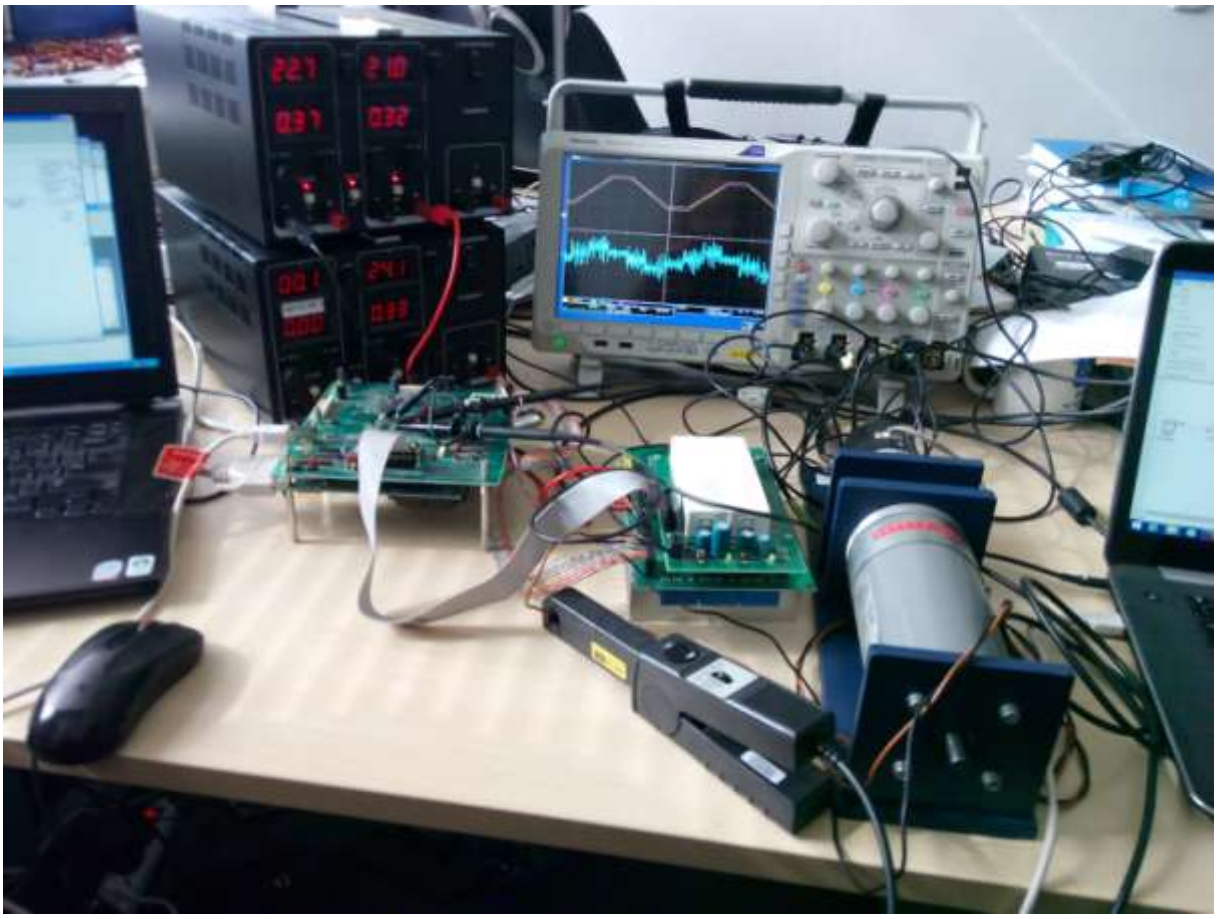
Legenda k Obr. 8.2:

- časové měřítko: 1 [s] / dílek
- zelená: požadované otáčky ω_{REF}, měřítko 200 [rads^{-1}] / dílek
- fialová: skutečné otáčky ω_{skut}, měřítko 200 [rads^{-1}] / dílek
- modrá: proud kotvou I_{a}, měřítko 2,5 [A] / dílek

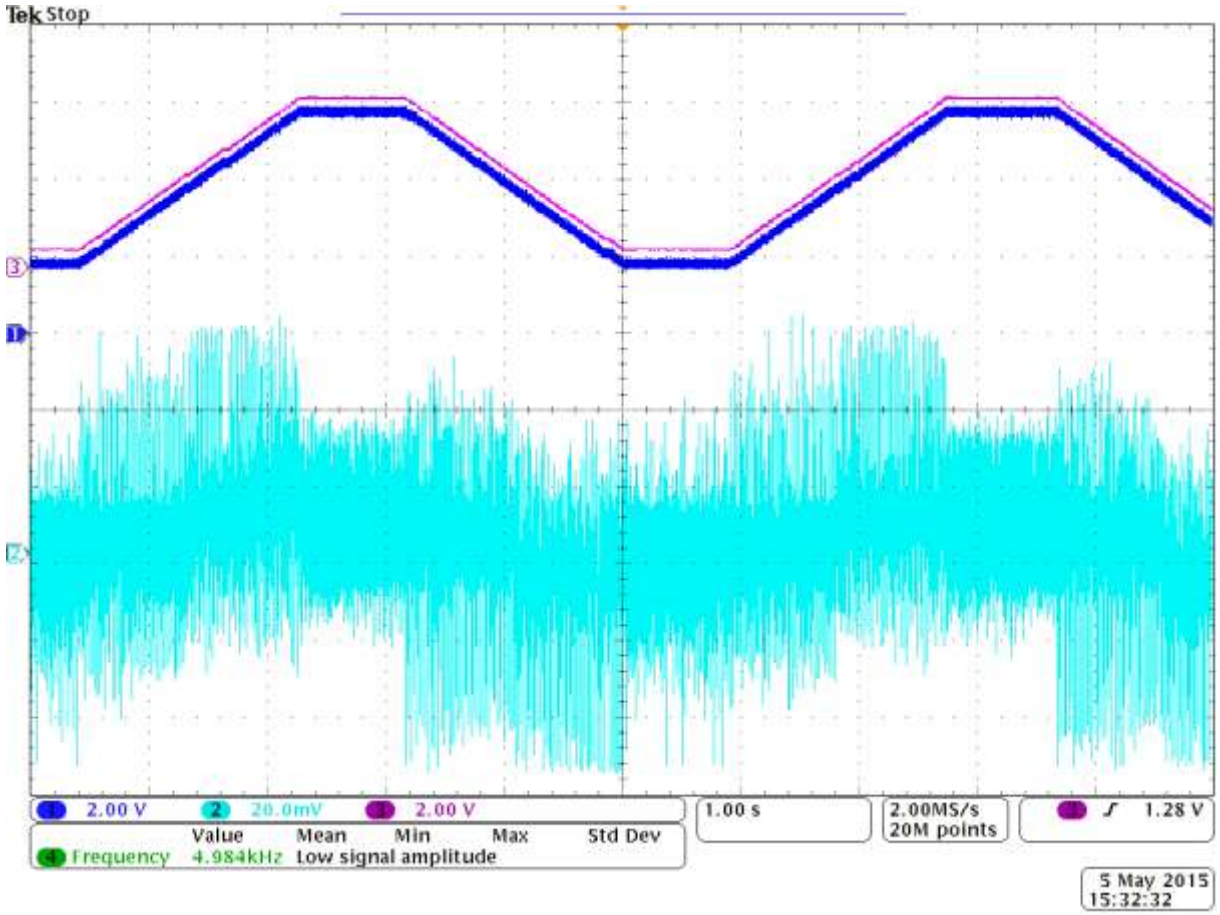
8.3 Měření lichoběžníkového průběhu otáček na reálné sestavě – standu

Od předchozího měření metody HIL jsou na následující fotografii Obr. 8.3 na první pohled patrné rozdíly – řízenou soustavu realizovanou v mikrokontroléru TMS320F28335 nahrazuje skutečný stejnosměrný motor s permanentními magnety napájený přes měnič diametrálním zdrojem napětí. Motor je spojený s řídicím systémem realizovaným v mikrokontroléru TMS320F2812. V pozadí je možné znovu vidět měření průběhů otáček požadovaných ω_{REF}, otáček skutečných ω_{skut} a proud kotvou I_{a} na osciloskopu, který je měřen proudovou sondou (na Obr. 8.3 vlevo od ss motoru). Proudová sonda má volitelný převod, měřeno bylo s nastavením 10 [mV] / 1 [A].

Mikrokontrolér TMS320F2812 obsahuje téměř identický SW jako při testování metody HIL. Jak je nastíněno v kapitole 7, liší se pouze v přepočtech měřených veličin.



Obr. 8.3: Fotografie z experimentálního měření na ss motoru



Obr. 8.4: Měření lichoběžníkového průběhu otáček na reálné sestavě - standu

Legenda k Obr. 8.4:

- časové měřítko: 1 [s] / dílek
- fialová: požadované otáčky ω_{REF}, měřítko 200 [rads^{-1}] / dílek
- modrá: skutečné otáčky ω_{skut}, měřítko 200 [rads^{-1}] / dílek
- tyrkysová: proud kotvou I_{a}, měřítko 2 [A] / dílek

9 Závěr

Cílem této diplomové práce bylo seznámit se teoreticky s metodami X In the Loop, získané znalosti aplikovat v návrhu modelů pro řízení stejnosměrného motoru s permanentními magnety v programu MATLAB – Simulink a správnost návrhu modelů ověřit praktickým měřením – simulací. Část zaměřenou na teorii a vysvětlení základních principů použitých metod je možné najít v kapitolách 1 až 4. Od kapitoly 5 je diplomová práce zaměřena na praktický návrh modelů v Simulinku a jejich simulací.

Kapitola 5 obsahuje kromě návrhu regulátoru proudu a otáček také sedm na sebe logicky navazujících modelů MIL, které byly sekvenčně vytvořeny dle popsaných postupů. Každý krok byl ověřen simulací a porovnáním s výsledky kroku předchozího. Výsledná simulace MIL byla provedena na Modelu 7 – na fyzikálním modelu s přidaným šumem a *Moving – Average* filtrem. Simulací bylo zjištěno, že aplikovaný postup návrhu MIL byl správný a výsledky tohoto měření jsou popsány na Obr. 5.20.

Na MIL navazují další kapitoly pro návrh SIL a PIL. Změřená výstupní simulace těchto metod je dle očekávání stejná jako u metody MIL. Cílem bylo ověřit správnost vygenerovaného kódu jednotlivých metod. Ta se prokázala právě tím, že nové bloky pro SIL a PIL neměly vliv na výslednou simulaci a obě metody vykazovaly identické výsledky.

Pro ucelenou informaci o metodách X In the Loop se práce zaměřuje také na metodu HIL (*Hardware In the Loop*).

Cílová platforma s mikrokontrolérem TMS320F2812 pracuje dle očekávání a jsou měřeny identické výstupy jak pro metodu PIL, tak HIL. Ostatně pro generování kódu pro simulaci řízené soustavy (měniče a motoru) na mikrokontroléru TMS320F28335 byly také využity bloky navržené v kapitolách, které předcházejí kapitole 7.1. Hardwarově závislé nastavení těchto bloků bylo samozřejmě upraveno pro mikrokontrolér TMS320F28335 s podporou *floating point* aritmetiky.

V experimentální části bylo provedeno měření lichoběžníkového průběhu otáček na několika konfiguracích, resp. sestavách. Tímto měřením bylo potvrzeno, že stejné výsledky a chování jsou pozorovány u řídicího systému, který byl navržen metodou MIL (finálním krokem, tzv. Modelem 7), u systému z metody HIL, stejně tak u řídicího systému, který ovládá reálnou řízenou soustavu měniče a motoru.

Seznam literatury a informačních zdrojů

[1] JELÍNEK Pavel: *Simulace Processor In the Loop a Hardware In the Loop*, [online] [cit. 2015-03-17] <http://automa.cz/index.php?id_document=34311>

[2] Encyklopedie Wikipedia: *Hardware.in-the-loop simulation*, [online] [cit. 2015-03-17] <http://en.wikipedia.org/wiki/Hardware-in-the-loop_simulation>

[3] VDC Research: *2011 Embedded Engineer Survey Results – Programming languages used to develop software*, [online] [cit. 2015-03-17] <http://blog.vdcresearch.com/embedded_sw/2011/06/2011-embedded-engineer-survey-results-programming-languages-used-to-develop-software.html>

[4] TAŠNER Tadej, LOVREC Darko, TAŠNER Frančišek, EDLER Joerg: *Comparison of LabVIEW and MATLAB for scientific research,* Annals of Faculty Engineering Hunedoara, ISNSN 1584-2673, 289-394 [online] [cit. 2015-03-17] <http://annals.fih.upt.ro/pdf-full/2012/ANNALS-2012-3-68.pdf>

[5] NILSSON Håkan: *Rapid Prototyping with Matlab/Simulink – A case study*, [online] [cit. 2015-03-17] <http://www.control.lth.se/documents/2002/5692.pdf>

[6] Software Testing Fundamentals: *Verification vs Validation*, [online] [cit. 2015-03-20] <http://softwaretestingfundamentals.com/verification-vs-validation/>

[7] BOCK Thomas, FÄRBER Georg, MAURER Markus: *Vehicle in the Loop (VIL) – A new simulator set-up for testing Advanced Driving Assistence Systems* [online] [cit. 2015-03-20] <https://www.nads-sc.uiowa.edu/dscna/2007/papers/Section%202B%20-%20Simulator%20Characteristics%20-%20Applications%20I/Bock.pdf>

[8] BUCHALCEVOVÁ Alena: *Metodiky budování informačních systémů*. 1. vyd Praha: Vysoká škola ekonomická v Praze, Oeconomia, 2009, 206 s, ISBN 978-80-245-1540-3.

[9] Encyklopedie Wikipedia: *Software development process*, [online] [cit. 2015-03-20] <http://en.wikipedia.org/wiki/Software_development_methodology>

[10] ZANDER Justyna, SCHIEFIERDECKER Ina, MOSTERMAN Pieter J.: *Model-Based Testing for Embedded Systems*. Taylor & Francis Group, LCC, 2012, 668 s, ISBN 978-1-4398-1847-3.

[11] BROEKMAN Bart, NOTENBOOM Edwin: *Testing Embedded Software.* 1. vyd. Velká Británie: Addison-Wesley, 2003, 368 s, ISBN 0-321-15986-1.

[12] KIRNER Raimund: *Testen Von Embedded Systems*, [online] [cit. 2015-03-20] <https://ti.tuwien.ac.at/cps/teaching/courses/testing_emb_sys/Documents/old-lecture-slides/tes6_hil_testing.pdf/at_download/file>

[13] Firemní web Texas Instruments: *TMS320F2812 Data Manual*, revize SPRS174T, květen 2012, [online] [cit. 2015-03-20] <http://www.ti.com/lit/ds/symlink/tms320f2812.pdf>

[14] Firemní web Spectrum Digital: *eZdsp™ F2812 Technical Reference*, revize 506265-0001 F, září 2003, [online] [cit. 2015-04-01] <http://c2000.spectrumdigital.com/ezf2812/docs/ezf2812_techref.pdf>

[15] BARTOŠ Václav: *Elektrické stroje*, 1. vyd. Plzeň: Západočeská univerzita 2006, 139 s., ISBN 978-80-7043-444-4

[16] SKALICKÝ Jiří: *Teorie řízení,* 2. vyd. Brno: Vysoké učení technické v Brně 2002, 98 s., ISBN 978-80-2142-112-7

[17] TIŠŇOVSKÝ Pavel: *Fixed point arithmetic*, [online] [cit. 2015-04-01] <http://www.root.cz/clanky/fixed-point-arithmetic/>

[18] Firemní web Texas Instruments: *C28x IQ – Math Library*, [online] [cit. 2015-04-01] <http://processors.wiki.ti.com/images/8/8c/IQMath_fixed_vs_floating.pdf Ccc>

[19] Encyklopedie Wikipedia: *IEEE floating point*, [online] [cit. 2015-04-01] <http://en.wikipedia.org/wiki/IEEE_floating_point>

[20] Encyklopedie Wikipedia: *IEEE 754-1985*, [online] [cit. 2015-04-01] <http://en.wikipedia.org/wiki/IEEE_754-1985>

[21] Magazín Embedded.com: *Milestones in embedded systems design*, [online] [cit. 2015-04-01] <http://www.embedded.com/design/prototyping-and-development/4007514/Milestones-in-embedded-systems-design>

[22] Encyklopedie Wikipedia: *Embedded systém*, [online] [cit. 2015-04-01] <http://en.wikipedia.org/wiki/Embedded_system>

[23] MathWorks MATLAB R2011a - Simulink Coder Help: *V-Model for System Development :: Product Overview (Simulink® Coder™)*

[24] MathWorks Documentation: *What is an s-function?*, [online] [cit. 2015-04-01] <http://www.mathworks.com/help/simulink/sfg/what-is-an-s-function.html>

[25] MathWorks MATLAB R2011a - Simulink Help: *Simulating Dynamic Systems :: How Simulink Works (Simulink®)*

[26] MATYS Vojtěch: *Potlačení mrtvých časů v napětovém střídači*, Plzeň, Západočeská univerzita 2012, diplomová práce, Fakulta elektrotechnická ZČU, Katedra elektromechaniky a výkonové elektroniky.

[27] MathWorks MATLAB R2011a - Simulink Help: *Choosing a Solver: Running Simulations (Simulink®)*

[28] MathWorks Documentation: *Choose a Solver*, [online] [cit. 2015-04-01] <http://www.mathworks.com/help/simulink/ug/choosing-a-solver.html#f11-56725>

[29] PÍCHOVÁ Jitka: *Model stejnosměrného motoru s permanentními magnety typ P2XR506,* semestrální práce Programování v silnoproudé elektrotechnice, Fakulta elektrotechnická ZČU, Katedra elektromechaniky a výkonové elektroniky