bakalářská práce

Řízení pohonů v FPGA

Ondřej Novák

Květen 2016

Ing. Richard Šusta Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra řídící techniky

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Ondřej Novák**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Řízení pohonů v FPGA**

Pokyny pro vypracování:

1. Seznamte se s dosavadními pracemi řešenými na katedře řídicí techniky, které se zabývají implementací regulátorů synchronních motorů na FPGA a připojením zařízení na platformě FPGA k Profinetu.2. Navrhněte PI regulátor motoru a implementujte jej v FPGA. Integrujte regulátor, tj. jeho hardwarovou i softwarovou část, jako aplikační úlohu nad funkcionalitou Profinet IO Device, které jsou dílčím výsledkem současně probíhající diplomové práce.3. Otestujte vytvořené řešení s dostupnými komerčními produkty v integračním testu.

Seznam odborné literatury:

1. Softing SDAI Profinet stack documentation.2. PXMC library for motion control. Documentation.

Vedoucí: Ing. Richard Šusta, Ph.D.

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 31. 3. 2016

Poděkování

Chtěl bych poděkovat svému vedoucímu, panu Ing. Richardu Šustovi, Ph.D., za cenné rady a podporu při tvorbě této práce.

Prohlášení

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

Datum

Podpis

Abstrakt

Tato práce se zabývá implementací platformy pro zařízení typu IO-Device v síti PROFINET na FPGA. Úkolem zařízení je komutace a řízení jednoho nebo více BLDC motorů.

Řízení motoru je realizováno v hardware, komunikace v síti PROFINET v programu na Nios II procesoru. Pro komunikaci je využita implementace protokolu PROFINET od společnosti Softing Industrial Automation GmbH, která dodává knihovnu pro komunikaci (nejen) v síti PROFINET spolu s ukázkovou aplikací, ze které projekt vychází.

Klíčová slova

FPGA; BLDC; PROFINET; PID regulátor

Abstrakt

This thesis is focused on the implementation of a platform for the IO-Device in the PROFINET network on an FPGA. The function of the device is BLDC motor commutation and control.

The motor control is fully implemented in the hardware, the communication in PROFINET network is implemented in the software using Nios II processor. For the software part, the implementation by Softing Industrial Automation GmbH company is used. This company provides a library for the communication using PROFINET (and other protocols) along with the demo application that is used as the base of this project.

Keywords

FPGA; BLDC; PROFINET; PID regulator

Obsah

1 Úvod **1**

2 Teoretická část **2**
2.1 BLDC motory . . . 2
2.1.1 Konstrukce . . . 2
2.1.2 Elektronická komutace . . . 3
2.2 PID regulace rychlosti otáček . . . 5
2.2.1 Anti-windup . . . 6
2.3 PROFINET . . . 10
2.3.1 Kategorie PROFINET zařízení . . . 10
2.3.2 Přenos dat . . . 11
2.3.3 Popis zařízení . . . 11

3 Použité technologie a hardware **12**
3.1 Vývojová sada Terasic DE2-115 . . . 12
3.1.1 Tlačítka a přepínače . . . 12
3.1.2 LED . . . 12
3.1.3 Rozšiřující GPIO port . . . 12
3.2 Výkonová deska pro připojení BLDC motoru . . . 13
3.3 Nios II . . . 14
3.3.1 QSys . . . 15
3.4 VHDL knihovna pro fixed point výpočty . . . 15
3.5 Softing Protocol IP for PROFINET . . . 16
3.5.1 Hardwarový návrh . . . 16
3.5.2 Demo aplikace . . . 17
Board Support Package . . . 19
Software switche . . . 19
Přidání projektů do Nios II IDE . . . 19
3.5.3 Ukázkový program pro PLC . . . 19

4 Řešení projektu **22**
4.1 Řízení motoru . . . 22
4.1.1 PWM generátor . . . 22
4.1.2 Kvadraturní dekodér . . . 24
4.1.3 Implementace komutátoru . . . 24
4.1.4 Implementace PID regulátoru . . . 26
4.1.5 Výpočet rychlosti . . . 28
4.1.6 Generování hodinového signálu . . . 29
4.1.7 Propojení hardwarových bloků . . . 30
4.2 PROFINET aplikace . . . 32
4.2.1 Úprava funkčnosti Demo aplikace . . . 32
4.2.2 Zjednodušení kódu . . . 33
Soubory s funkcemi protokolů . . . 33
Demo aplikace . . . 33
Platforma . . . 33
4.2.3 Změny v Nios II subsystému . . . 34
Problémy se subsystémem . . . 34

4.3 PLC program 34
4.4 Struktura projektu 34

5 Závěr **36**

Přílohy

A DVD **37**

Literatura **38**

Zkratky

API	Application Programming Interface
BLDC	Brushless DC electric motor — bezkartáčový elektrický motor na stejnosměrný proud
CPU	Central Processing Unit
DAP	Device Access Point
DC	Direct current — stejnosměrný proud
DPRAM	Dual-Port RAM
EM	Elektromagnetic — elektromagnetický
EMF	Electromotive force — elektromotorická síla
GSD	Generic Station Description
GSDML	GSD Markup Language
GPIO	General Purpise IO
GND	Ground — zem
IDE	Integrated Development Environment
IO	Input/Output – Vstup/Výstup, či Vstupně/Výstupní
IP	Internet Protocol
IRC	Incrementary rotary counter — inkrementální rotační čítač
IRT	Isochronous real-time — Izochronní v reálném čase
LCD	Liquid Crystal Display — Display z tekutých krystalů
LE	Logic element — logický prvek
LED	Light-Emitting Diode — Světlo emitující dioda
LSB	Least significant bit — nejnižší bit
MM	Memory Mapped
MMU	Memory Management Unit
MPU	Memory Protection Unit
MSB	Most significant bit — nejvyšší bit
PID	Proportional-integral-derivate controller — proporcionálně integračně derivační regulátor
PLC	Programmable Logic Controller — programovatelný logický automat
PLL	Phase-Locked Loop — Fázový závěs
PWM	Pulse-width modulation — Pulzně šířková modulace
RAM	Random-Access Memory — Paměť s náhodným přístupem
RISC	Reduced Instruction Set Computing
RT	Real-time — V reálném čase
SDAI	Simple Device Application Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHISC	Very High Speed Integrated Circuit
XML	eXtensible Markup Language

1 Úvod

Elektrické motory jsou nedílnou součástí mnoha aplikací. V současné době je mnohdy výhodnější použít motory bez kartáčové komutace, vzhledem k tomu, že tyto motory v jistých ohledech vykazují lepší vlastnosti než motory s kartáčovou komutací. Jedním z bezkartáčových motorů, tedy motorů bez kartáčové komutace, je BLDC (Brushless DC) motor. Ten, na rozdíl od kartáčového motoru, potřebuje elektronický komutátor.

Při aplikaci je motory většinou třeba řídit, například pomocí PLC. Řídit můžeme různé veličiny, jako je rychlost otáčení, nebo úhel natočení. Někdy však není možné (nebo výhodné) připojit motory přímo k řídící jednotce, a proto byly vyvinuty metody distribuce povelů přes počítačovou síť. Jednou z možností je použít řídící jednotku, která posílá povely koncovým zařízením. Koncová zařízení provádí „rychlé řízení“ přímo připojeného motoru (či motorů), tedy přímo reaguje na chování jeho dynamiky, zatímco řídící jednotka provádí řízení celého výrobního procesu, který se může skládat z ovládání mnoha koncových zařízení.

Při řízení je mimo jiné důležitá doba odezvy. Běžné počítačové sítě, jako je například Ethernet, nevyhovují nárokům některých řídících procesů. Případné zpoždění komunikace totiž může mít vážné následky, a tak je v některých případech potřeba použít komunikaci v reálném čase. Taková komunikace má deterministické chování a navíc může mít garantovanou dobu odezvy.

Tato práce se bude zabývat implementací jednoduchého koncového zařízení pro řízení BLDC motoru v síti PROFINET na FPGA.

2 Teoretická část

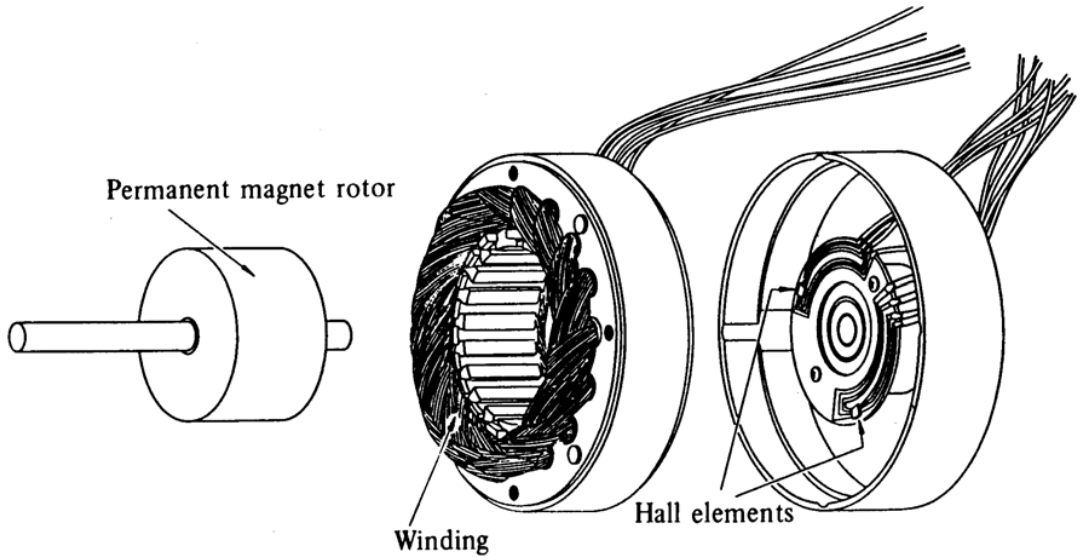
V této kapitole se budeme zabývat principy a mechanizmy, které je třeba znát pro správné pochopení fungování jednotlivých komponent projektu. V první části bude vysvětlena konstrukce a princip komutace BLDC motoru, dále budou uvedeny základní pojmy a principy PID regulátoru a na závěr kapitoly budou uvedeny základní pojmy a stručné vysvětlení fungování sítě PROFINET.

2.1 BLDC motory

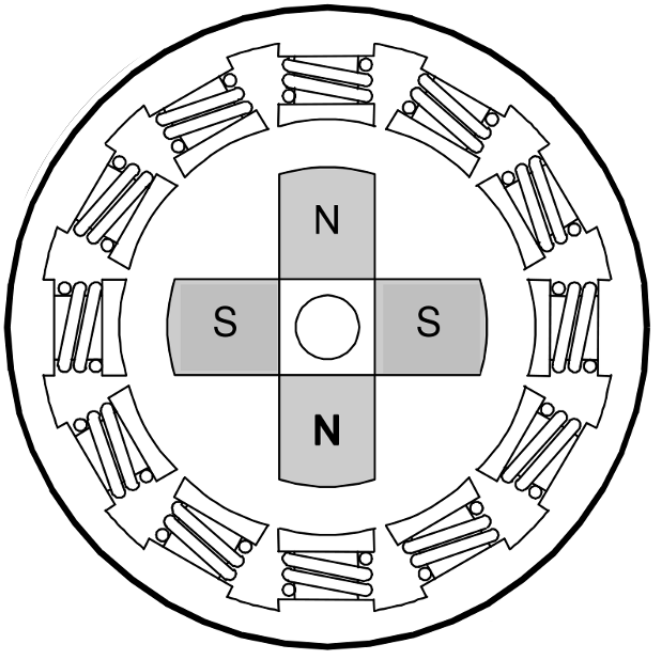
BLDC motor (z anglického Brushless DC motor) je bezkartáčový stejnosměrný motor s elektronickou komutací poháněný stejnosměrným proudem[1, 2, 3]. Hodí se jako efektivnější náhrada kartáčového DC motoru[2], protože na rozdíl od něj nepoužívá mechanickou komutaci a díky tomu má menší opotřebení, při rotaci nevzniká jiskření a nižší generování elektromagnetického (EM) rušení[4]. Od kartáčových DC motorů také přebírá vlastnosti, jako je nižší setrvačnost a díky tomu rychlá dynamická odezva, nebo nízké ohmické ztráty[2]. Jeho nevýhodou je komplexnější konstrukce, nutnost elektronické komutace a tedy složitější řízení a vyšší cena.

2.1.1 Konstrukce

BLDC motor se skládá z jednoho či více permanentních magnetů (rotor), několika vinutí (stator). Některé typy obsahují Hallovy sondy pro určení natočení rotoru vůči statoru, což umožňuje provádět jednoduchou elektronickou komutaci. Znázornění takového motoru je na obrázku 1.



Obrázek 1 Pohled na jednotlivé části BLDC motoru[5]



Obrázek 2 Uspořádání vinutí a permanentních magnetů třífázového čtyřpólového BLDC[1]

U motorů rozlišujeme pojmy mechanická a elektrická otáčka. Otáčka mechanická odpovídá jednomu fyzickému otočení rotoru vůči statoru o 360°. Elektrická otáčka odpovídá jednomu kompletnímu elektrickému (či signálovému) cyklu, tedy natočení rotoru vůči statoru nějakého z počátečního stavu (z hlediska vinutí) do stavu stejného[3].

V případě jednoho páru pólů připadá jedna otáčka elektrická na jednu otáčku mechanickou, pro vícepólové magnety, které se používají pro zjemnění rotace[6], to však neplatí. U motorů vícepólovými permanentními magnety, rotor se z nějakého stavu do stavu stejného dostane při jedné mechanické otáčce několikrát. Například u motoru se čtyřpólovým magnetem, který můžeme vidět na obrázku 2, se musí rotor natočit o 180°, aby se k vinutí, u kterého byl na počátku severní pól, přiblížil znovu severní pól. Obecně pro výpočet počtu elektrických otáček na jednu mechanickou otáčku platí vztah[4]

$$l = p \cdot n, \tag{1}$$

kde

kde

l je počet elektrických otáček,
n je počet mechanických otáček a
p je počet pólových párů.

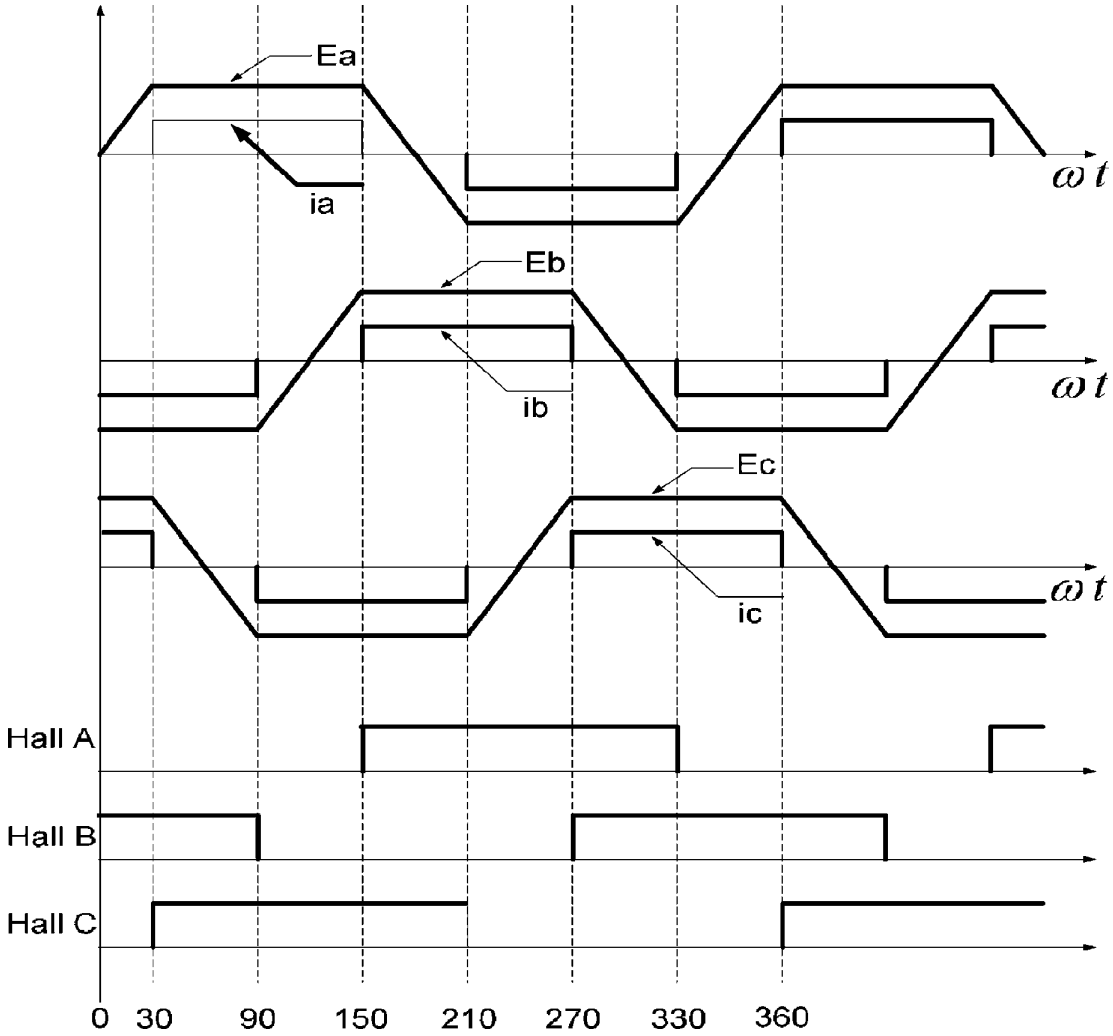
Na obrázku 3 je znázorněn průběh zpětné elektromotorické síly pro každou ze tří fází a stavu Hallových sond pro elektrickou otáčku (ve stupních).

2.1.2 Elektronická komutace

Komutace je proces přepínání směru proudu vedeného do cívek v motoru. U kartáčových DC motorů je realizována mechanickými kartáči, které jsou v kontaktu s vinutím cívek.

Díky elektrické komutaci je možno odstranit kartáče a proud v cívkách spínat řídícím systémem podle úhlu natočení rotoru vůči statoru. Nejjednodušší metodou pro určení úhlu je použití trojice Hallových sond od sebe vzdálených 120°, s jejichž pomocí je možno určit 6 různých poloh v každém elektrickém cyklu. Každá sonda se spíná a rozepíná podle polohy rotoru, pro tři sondy tedy máme $2^3 = 8$ možných kombinací, avšak pouze 6 z nich je dosažitelných. Vzhledem k rovnoměrnému rozmístění snímačů může být sepnut pouze jeden či dva snímače. Nemůže tedy nastat situace, kdy by byly všechny tři sondy sepnuty nebo rozepnuty[1, 3]. Požadavek na sepnutí fází a pořadí stavů můžeme vidět v tabulce 1.

Pokud je k dispozici informace o absolutní poloze, je možné provést komutaci pomocí přičítání 120°[1]. Motor se v tomto případě ovládá sinusovými průběhy napětí na všech třech fázích posunutých od sebe o 120°. Pro zjištění polohy se běžně používá inkrementální rotační čítač (IRC), tento způsob měření však neposkytuje absolutní hodnoty, a proto je třeba po spuštění provést počáteční zarovnání[7]. Inicializační část a nutnost pracovat s funkcí sinus však přidávají tomuto způsobu výpočetní i implementační složitost, ačkoliv při rotaci má tato metoda srovnatelné výsledky s komutací na základě polohy získané z Hallových sond[8]. Pro získání absolutní polohy lze také použít měření zpětné elektromotorické síly (EMF)[9], komutace prováděná na základě odhadu



Obrázek 3 Průběhy zpětné EMF a stavu Hallových sond pro jednu elektrickou otáčku[2]

Pořadí	Kód	Stav Hallových sond A B C	Sepnutí jednotlivých fází a b c
1	1	0 0 1	1 0 0
2	5	1 0 1	1 0 0
3	4	1 0 0	0 1 0
4	6	1 1 0	0 1 0
5	2	0 1 0	0 0 1
6	3	0 1 1	0 0 1

Tabulka 1 Tabulka s požadavkem na sepnutí fází pro rotaci proti směru hodinových ručiček

úhlu natočení podle EMF se nazývá bezsenzorová (anglicky sensorless). Díky tomu není třeba, obvykle drahý, magnetický či optický enkodér, nevýhodou je však vyšší složitost implementace.

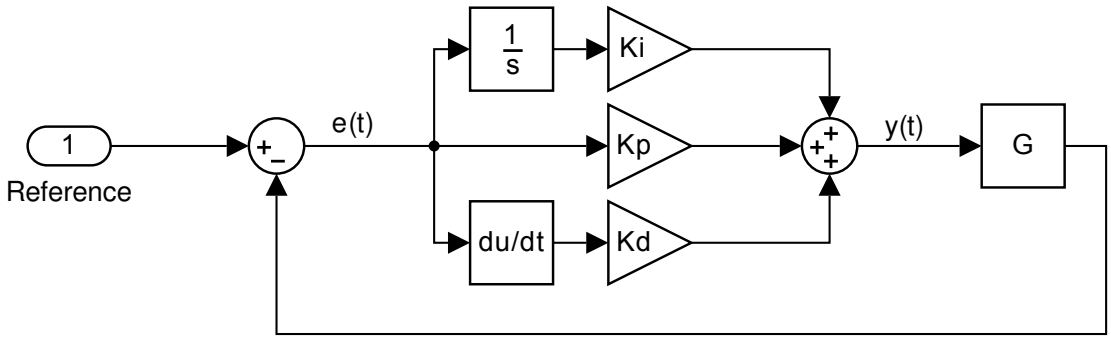
2.2 PID regulace rychlosti otáček

Pro regulaci rychlosti otáček je možno využít zpětnovazební regulátor. Jeho účelem je automaticky snižovat, a nejlépe zcela eliminovat, chybu mezi referenční a výstupní hodnotou regulovaného systému. Regulátor na základě chyby řídí vstupní hodnotu akčního členu (aktuátoru) tak, aby jeho výstupem byla požadovaná hodnota měřené veličiny.

Regulátory PID, nebo z něj odvozené, jsou jedny z nejpoužívanějších regulátorů pro řízení dynamických systémů[10]. Jejich výstupní hodnota je proporcionálně závislá na aktuální chybě, integrálu chyby a derivaci chyby. V případě regulace otáček díky integrační složce můžeme docílit (alespoň v ideálním případě) nulové ustálené regulační odchylky[11], derivační složka predikuje chování systému, čímž zvyšuje stabilitu a urychluje ustálení na konečné hodnotě. Blokové schéma PID regulátoru můžeme vidět na obrázku 4.

Analogový PID regulátor se implementuje přímo v elektronice, pomocí zesilovačů, integrátorů a derivátorů. Takové regulátory jsou spojité a citlivější, ale jejich parametry lze nastavovat pouze v hardware, tedy výměnou součástek, nebo pomocí proměnných součástek, jako je například potenciometr. Takový regulátor je popsán rovnicí[10]

$$y(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(t)dt + K_d \cdot \frac{de(t)}{dt}, \tag{2}$$



Obrázek 4 Blokové schéma PID regulátoru regulujícího systém s přenosem G

kde

y(t)	je akční zásah regulátoru,
e(t)	je rozdíl referenční a naměřené hodnoty (chyba),
K_p	je proporcionální konstanta,
K_i	je integrační konstanta a
K_d	je derivační konstanta.

Aplikací Laplaceovy transformace na rovnici 2 dostaneme

$$Y(s) = K_p \cdot E(s) + K_i \cdot \frac{E(s)}{s} + K_d \cdot sE(s), \tag{3}$$

kde

Y(s)	je akční zásah regulátoru,
E(s)	je chyba reference a naměřené hodnoty a
K_p, K_i, K_d	jsou stejné konstanty jako v rovnici 2.

Případně můžeme ještě vztah 3 vydělit obrazem chyby $\mathcal{L}\{e(t)\} = E(s)$, z čehož dostáváme přenos[10, 11]

$$G(s) = \frac{Y(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s. \tag{4}$$

Další možností je implementace digitálního regulátoru, jehož výhodou je například flexibilita. Na druhou stranu digitální regulátory mohou reagovat pomaleji, než analogové. Pro tyto účely lze použít například mikrokontroler, FPGA nebo PLC, protože ale výpočty na těchto zařízeních jsou diskrétní, je také potřeba upravit regulátor a to například aproximací diferenciální rovnice 2 rovnicí diferenční[10]

$$y(n) = K_p \cdot e(n) + K_i \cdot T \sum_{i=0}^{n-1} e(i) + \frac{K_d}{T}\left(e(k) - e(k-1)\right), \tag{5}$$

kde

T	je vzorkovací perioda a
e(n), y(n)	je chyba a akční zásah n-tého vzorku.

Vzorkovací perioda T by měla být menší, než je desetinásobek nejmenší časové konstanty systému[12].

V literatuře se můžeme setkat s pojmy jako jsou P, PI nebo PD regulátor. Jde o variace PID regulátoru s některými konstantami rovnými nule. Pokud jsou všechny tři z konstant K_p, K_i, K_d nenulové, mluvíme o PID regulátoru (obr. 5c). V případě nulové jedné z konstant K_i či K_d, jde o regulátor PI(obr. 5b), respektive PD. Pokud jsou nulové obě konstanty K_i i K_d, jde o P regulátor(obr. 5a), což je vlastně pouze zesilovač signálu.

2.2.1 Anti-windup

Reálné aktuátory většinou mají omezený rozsah vstupních hodnot. Může jít například o maximální napětí, minimální a maximální úhel a podobně. Jevu říkáme saturace vstupu

a kvůli své nelinearitě může zavádět problémy při regulaci lineárních systémů[13]. Pokud se chyba drží v hodnotách, pro které regulátor generuje akční zásah

$$U_{min} \leq u(t) \leq U_{max}, \tag{6}$$

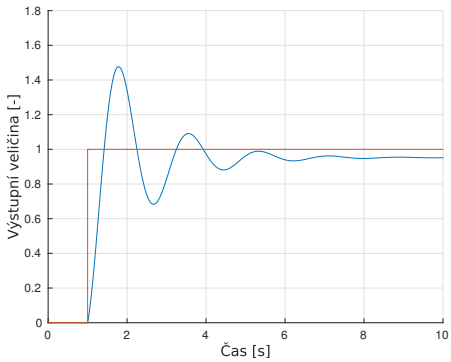
kde

U_{min}, U_{max} je minimální, respektive maximální hodnota vstupu aktuátoru a
u(t) je výstup regulátoru (tedy vstupní hodnota aktuátoru) v čase t,

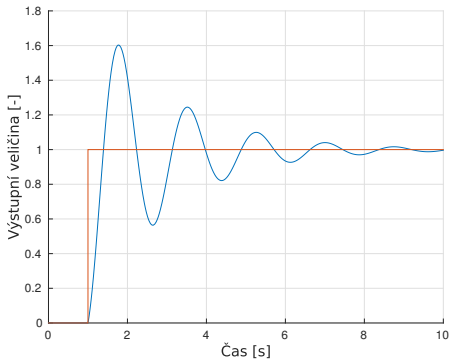
systém nevykazuje nežádoucí chování. Problém však nastane v případě velké změny reference, nebo působením velké poruchy, na níž regulátor reaguje nastavením velkého akčního zásahu u(t), pro který platí

$$u(t) \notin < U_{min}, U_{max} > . \tag{7}$$

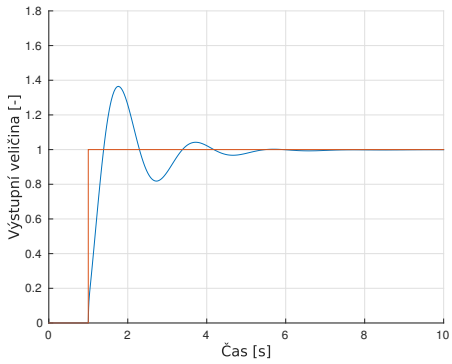
Mějme regulátor C bez anti-windup mechanizmu obsahující integrátor a dále mějme řízený systém G s minimálním a maximálním vstupem U_{min} a U_{max}. V případě dostatečně velké chyby nastane saturace vstupu, avšak výstupní hodnota aktuátoru stále nemusí být dostatečná. Integrátor v regulátoru však stále integruje a tím zvyšuje akční zásah, ačkoliv se tato změna na řízeném systému neprojeví, protože jeho vstup je saturován a zvýšením akčního zásahu se tedy chyba nesníží. Po snížení chyby na úroveň, kde aktuátor nesaturuje, je v integrátoru uložena vysoká hodnota a regulátoru trvá



a) $K_p = 10, K_i = 0, K_d = 0$, P regulátor



b) $K_p = 10, K_i = 5, K_d = 0$, PI regulátor



c) $K_p = 10, K_i = 5, K_d = 1$, PID regulátor

Obrázek 5 Odezva systému na jednotkový skok pro různé PID parametry regulátoru

dlouhou dobu, než se jeho výstup dostane na ustálenou hodnotu. Tomuto chování se říká windup efekt.

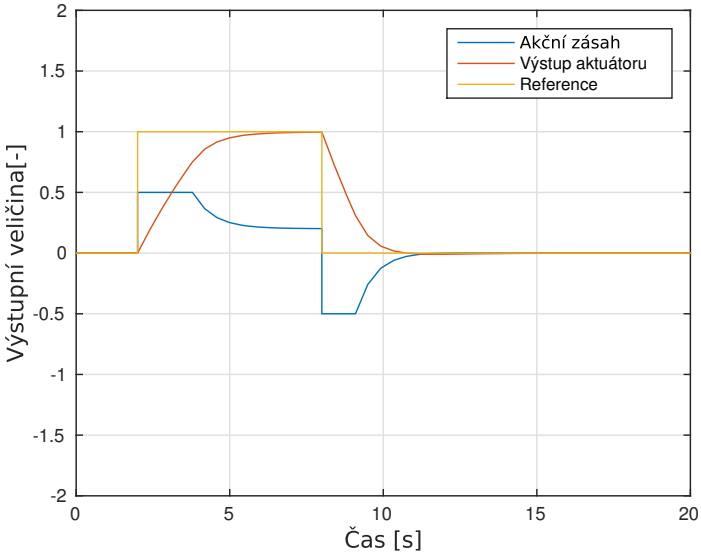
Jednou z možností snížení vlivu windup efektu je takzvaná tracking-back anti-windup strategie[14]. V tomto případě je integrace i výpočet akčního zásahu prováděn jako u běžného PID, avšak pokud akční zásah překročí povolenou hranici U_{bound} (obvykle tyto hranice bývají dvě - pro minimální a maximální vstupní hodnotu), je jako výstup regulátoru, tedy akční zásah, použita hodnota této hranice a z integrátoru je zpětně odečten rozdíl

$$K_i \cdot [u(t) - U_{bound}] \,, \tag{8}$$

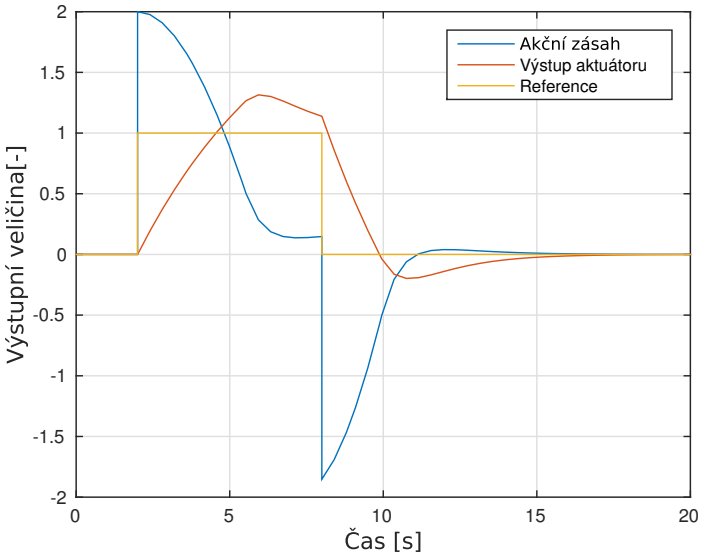
kde

K_i je integrační konstanta regulátoru.

Na obrázku 6 můžeme vidět srovnání odezvy systému i regulátoru na skok s použitím (obr. 6a) a bez použití (obr. 6b) anti-windup mechanizmu. Jak je vidět, anti-windup v tomto případě snižuje dobu ustálení i překmit.



a) S použitím back-tracking anti-windup mechanizmu



b) Bez použití anti-windup mechanizmu

Obrázek 6 Odezva systému regulovaného PID regulátorem na impulz dlouhý $t_s = 6\,s$

2.3 PROFINET

PROFINET je jeden z významných standardů v oblasti průmyslového Ethernetu (Industrial Ethernet), založený na standardech Ethernet (IEEE 802.3) a Fast Ethernet (IEEE 802.3u). Umožňuje RT (Real-time), IRT (Isochronous Real-Time) a běžnou TCP/IP či UDP/IP komunikaci bez jakýchkoliv omezení (tato komunikace však není v reálném čase)[15].

2.3.1 Kategorie PROFINET zařízení

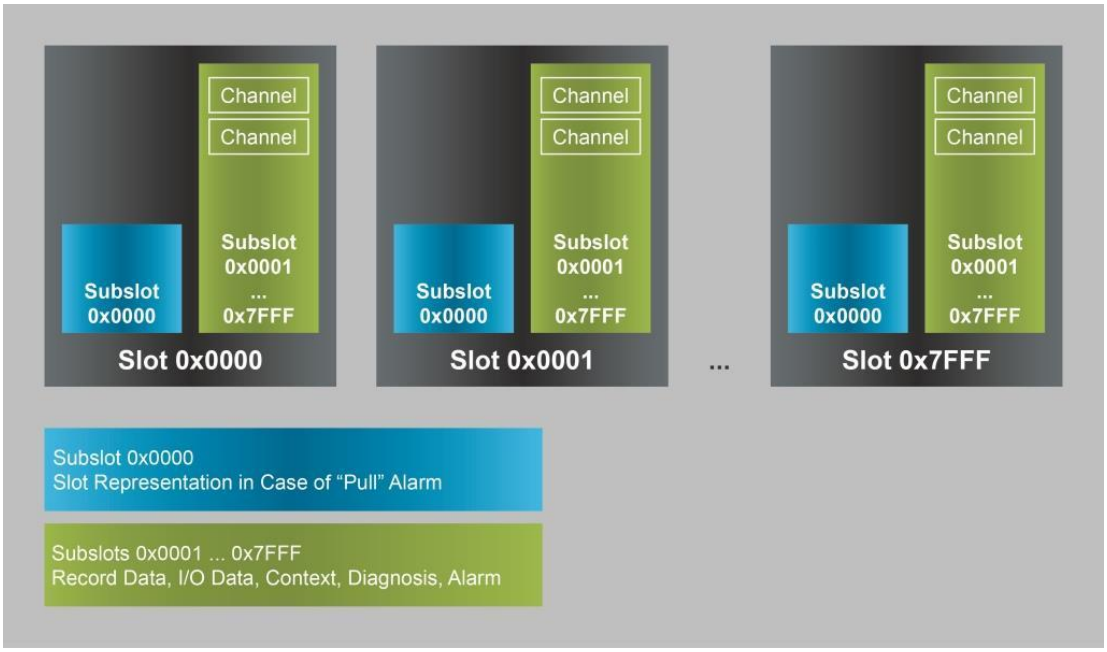
Zařízení v rámci sítě PROFINET jsou dělena do třech kategorií, každé z nich vykonávající jinou funkci v procesu[16]:

IO-Controller Zařízení typu master řídící automatizační proces pomocí cyklické i acyklické komunikace s IO-Device,

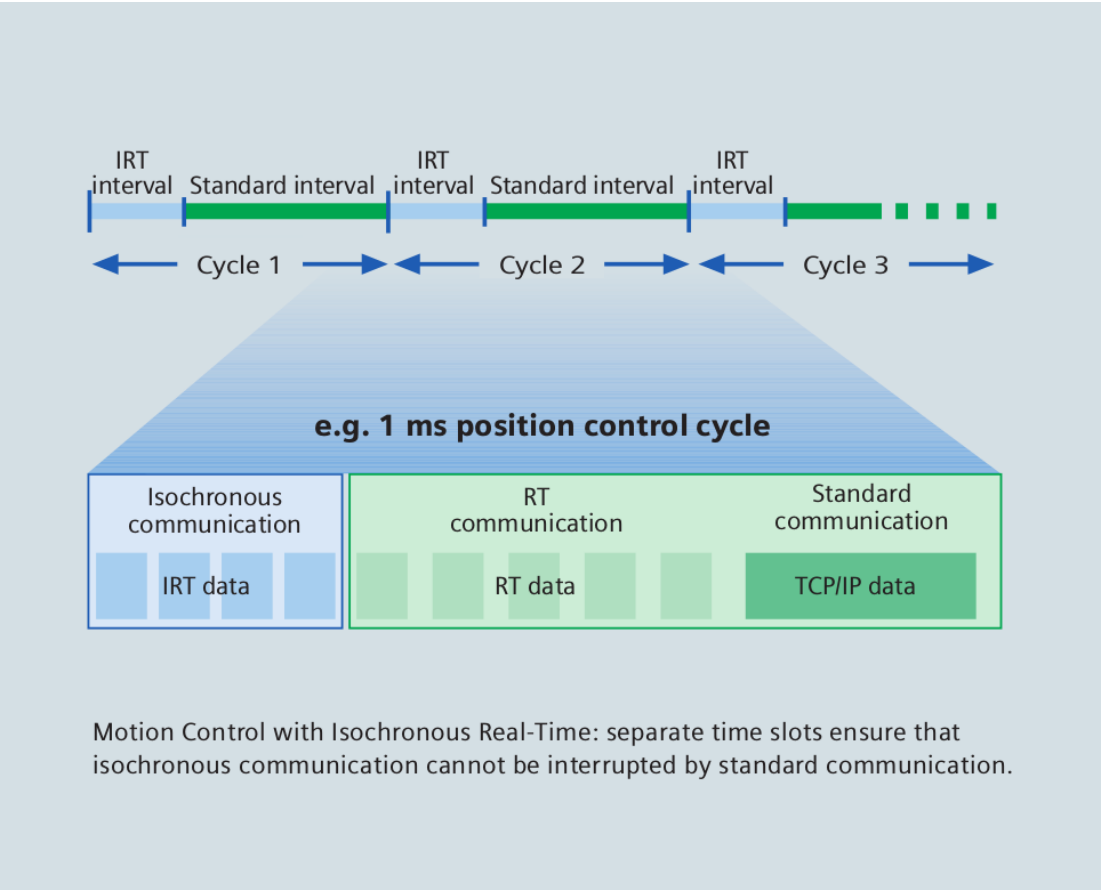
IO-Device Zařízení typu slave fungující jako senzory a aktuátory distribuované přes síť a

IO-Supervisor Zařízení pro obsluhu, monitorování či diagnostiku procesu.

Zařízení typu IO-Device jsou modulární (obr. 7), což uživateli (operátorovi) umožňuje zapojovat do slotů moduly umožňující zařízení vykonávat určité funkce. Pokud zařízení obsahuje pouze virtuální sloty a virtuální moduly, které nemohou být zapojeny uživatelem, je označováno jako *compact device*. Moduly zajišťující připojení vstupů a výstupů nazýváme DAP (Device Access Point) a v zařízení jich může být několik. Sloty a moduly jsou dále děleny do subslotů a submodulů. Speciální význam má submodul 0, zprostředkovávající pouze alarmy. Zbylé submoduly mohou obsahovat IO, diagnostické informace nebo záznamy[17].



Obrázek 7 Příklad PROFINET IO-Device s jednotlivými moduly a submoduly[17]



Obrázek 8 Rozdělení cyklů na intervaly[15]

2.3.2 Přenos dat

Přenos dat je řízen cyklem, jehož parametry jsou stanoveny při začátku komunikace. Důležitým parametrem cyklu přenosu je jeho perioda, což je vlastnost zařízení IO-Controller určující jeho výkon. Každý IO-Controller musí podporovat periodu 1 ms a libovolně může umožňovat komunikaci i kratších či delších cyklech[17].

Každý jeden cyklus je dělen na dva intervaly (obr. 8). Prvním intervalem je *IRT interval*, po který, jak vypovídá jeho název, jsou přenášena IRT data. Druhý interval je nazván *Standard interval*, v jehož průběhu jsou přenášena RT data a dále TCP/IP či UDP/IP časově nekritická data[15].

2.3.3 Popis zařízení

Pro konfiguraci a popis zařízení se používá GSD (Generic Station Description) soubor. Jde o XML (eXtensible Markup Language) soubor s GSDML (GSD Markup Language) syntaxí popisující implementaci IO-Device. Soubor obsahuje například strukturu cyklických IO dat, informace o zařízení (funkce, identita, ...), seznam modulů a DAP[16].

3 Použité technologie a hardware

V následující kapitole bude popsán použitý hardware a softwarové komponenty třetích stran. Nejdříve se podíváme na použitou vývojovou sadu pro FPGA a výkonovou desku, která byla použita jako interface mezi motorem a FPGA. V další části bude popsán soft-core procesor NIOS II, použité VHDL moduly a implementace PROFINET od společnosti Softing Industrial Automation GmbH.

3.1 Vývojová sada Terasic DE2-115

Pro tento projekt byl použit kit Terasic DE2-115 z důvodu kompatibility s použitou implementací PROFINET od společnosti Softing Industrial Automation GmbH, kterou se dále zabývá sekce 3.5. Jde o vývojovou desku pro FPGA Altera z rodiny Cyclone IV. Konkrétně se jedná o čip je Altera Cyclone IV EP4CE11529C7, který disponuje 114000 logickými prvky (LEs), 432 M9K paměťovými bloky, 266 násobiči a 4 PLL[18].

Deska kromě samotného čipu a nástrojů pro programování obsahuje řadu periferií. Od jednoduchých, jako jsou například tlačítka a LED, až po složitější, jako je USB rozhraní či LCD. Bylo by nad rámec této práce popisovat všechny periferie, zaměříme proto pozornost pouze na některé z těch, které jsou použity v tomto projektu. Kromě níže uvedených byl použit ještě LCD a Gigabit Ethernet port, avšak obsluha těchto dvou periferií je řešena ve výchozí PROFINET aplikaci a proto nemá smysl tyto periferie popisovat. Všechny informace o periferiích byly čerpány z [19].

3.1.1 Tlačítka a přepínače

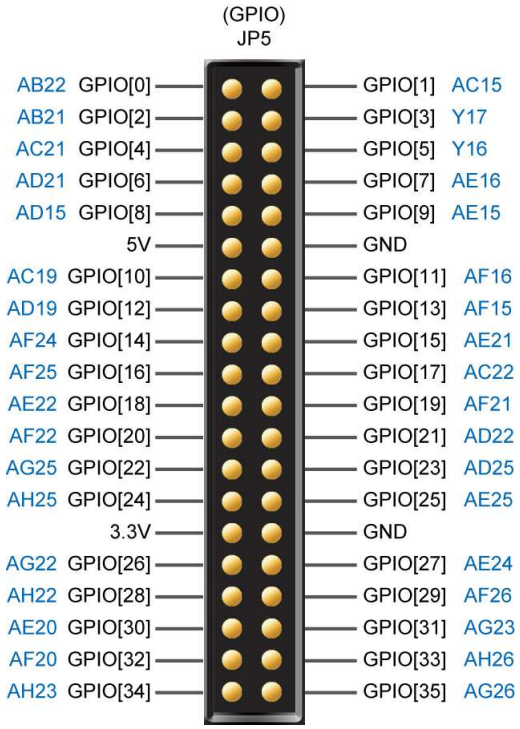
Mnohé vývojové desky obsahují několik tlačítek či přepínačů. V případě DE2-115 jde o 4 tlačítka s ošetřenými zákmity (debounce). Výstup tlačítek je invertovaný a tedy při stisknutí mají na výstupu logickou nulu. Dále deska obsahuje 17 přepínačů. Ty nemají ošetřené zákmity a v dolní poloze jsou ve stavu log. 0.

3.1.2 LED

Pro indikaci stavů je na desce přítomno 27 ovladatelných LED. Celkem 18 diod je červené barvy a zbylých 9 diod je barvy zelené. Po přivedení log. 1 na příslušný pin začne dioda svítit.

3.1.3 Rozšiřující GPIO port

Pro komunikaci desky s externím hardwarem je možno využít GPIO port (General Purpose IO). Konektor, který můžeme vidět na obrázku 9, se skládá celkem ze 40 pinů, dva z pinů jsou připojeny na zdroje napětí 5 V, respektive 3.3 V a další dva piny na zem (GND). Maximální odebíraný výkon je přibližně 5 W. Zbylých 36 pinů je přímo připojených na piny FPGA čipu. Vstupy jsou navíc chráněny proti podpětí a přepětí pomocí diod a rezistorů.



Obrázek 9 Rozložení GPIO vývodů na desce a seznam pinů FPGA čipu, na něž jsou připojené[19]

3.2 Výkonová deska pro připojení BLDC motoru

Použitá výkonová deska 3P-MOTOR-DRIVER-1 dodaná společností PiKRON slouží k připojení BLDC motoru k vývojové sadě Terasic DE2-115 popsané v kapitole 3.1. Deska zajišťuje galvanické oddělení FPGA od výkonové části, příjem diferenciálního signálu z IRC a ovládání motoru pomocí tří půlmůstků. Každý z půlmůstků je tvořen dvojicí výkonových tranzistorů a řídícím obvodem LT1158[1].

Komunikace s deskou je realizována přes GPIO piny vývojové sady, konkrétně se jedná o tři vstupní PWM kanály, ke každému kanálu navíc ještě náleží vstup ENABLE. Výkonová deska má dále šest výstupů: stav tří Hallových sond, dva kanály z inkrementálního čítače (IRC) a jeden INDEX výstup signalizující jednu plnou mechanickou otáčku. Na obrázku 9 můžeme vidět GPIO piny a k nim příslušící piny na FPGA čipu. Pro komunikaci s výkonovou jednotkou jsou použity piny uvedené v tabulce 2. Navíc jsou ještě použité piny GND, 5 V a 3.3 V pro napájení integrovaných obvodů.

Funkce	Označení pinu FPGA čipu	Číslo GPIO pinu
PWM kanál 1	AB21	2
PWM kanál 2	AB21	6
PWM kanál 3	AB21	10
PWM ENABLE pro kanál 1	AB22	0
PWM ENABLE pro kanál 2	AC21	4
PWM ENABLE pro kanál 3	AD15	8
Hallova sonda 1	AH23	30
Hallova sonda 2	AF20	32
Hallova sonda 3	AH23	34
IRC kanál A	AG22	24
IRC kanál B	AE20	26
IRC index	AF20	28

Tabulka 2 Funkční význam použitých GPIO pinů

3.3 Nios II

Nios II je soft-core procesorové jádro používající RISC (Reduced Instruction Set Computing) architekturu, které je duševním vlastnictvím společnosti Altera. Ta distribuuje procesor spolu se sadou nástrojů pro vývoj uživatelům svých FPGA. Nios II disponuje například[20]:

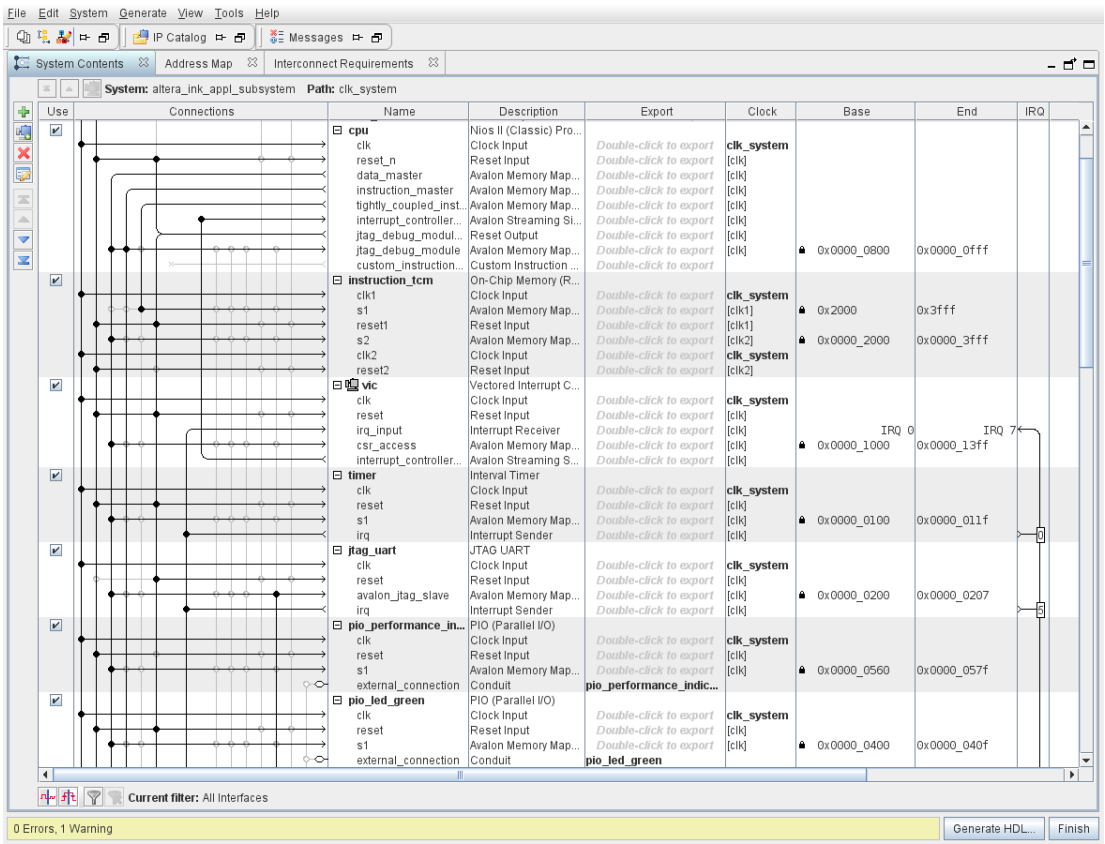
- 32-bitovou instrukční sadou, registry a adresací,
- 32 registry pro obecné použití (general purpose),
- 32 zdroji přerušení,
- rozhraním pro externí řadič přerušení pro připojení dalších zdrojů,
- násobením či dělením dvou 32-bitových čísel v jedné instrukci,
- instrukcemi pro násobení 64 a 128-bitových čísel,
- volitelnou možností podpory výpočtů s plovoucí desetinnou čárkou (single-precision floating-point),
- volitelnou podporou MMU (memory management unit) a MPU (memory protection unit) a
- modulem pro debugování software umožňující spustit, zastavit a krokovat program.

Poznámka 1

Soft-core procesorové jádro je možno plně implementovat pomocí logické syntézy a může tedy být implementováno například na FPGA. Mezi soft-core procesorová jádra patří Altera Nios/Nios II, nebo Xilinx MicroBlaze.

Protože je jádro soft-core, může programátor specifikovat vlastnosti procesoru a parametrizovat jej tak vhodně pro konkrétní projekt. Například je možné vytvořit komunikační rozhraní mezi procesorem a dalšími funkčními bloky umístěnými na FPGA, nebo přidat či odebrat podporu výpočtů s desetinnou čárkou[21].

Altera v rámci svého vývojového prostředí dodává programátorům Nios II Software Build Tools, což je vývojové rozhraní pro Nios II vycházející z prostředí Eclipse, využívající GNU C/C++ toolchain.



Obrázek 10 Ukázka uživatelského rozhraní programu Qsys se zapojeným Nios II procesorem a periferiemi

3.3.1 QSys

QSys je součástí vývojového rozhraní pro FPGA Altera nazvaného Altera Quartus II. Umožňuje uživateli propojování IP core[1] v grafickém rozhraní (obr. 10) a následné vygenerování hardware návrhu. Ve spojení s Nios II procesorem ho lze mimo jiné použít pro mapování adresního prostoru na IO piny procesoru. Tím je možné vytvořit rozhraní mezi software běžícím na procesoru a hardware návrhem na FPGA.

3.4 VHDL knihovna pro fixed point výpočty

Protože výchozí projekt je popsán v jazyce VHDL verze 1993, byl pro popis ostatních částí projektu zvolen stejný jazyk ve stejné verzi. Kromě verze 1993 existuje několik dalších verzí. Původní VHDL z roku 1987 a aktualizované standardy z roků 1993, 2000, 2002 a 2008. Verze 1993 však nepodporuje operace s čísly s desetinnou čárkou, tuto funkcionalitu přinesla až verze 2008[22]. Pro regulaci PID regulátorem je však práce s desetinnými čísly potřeba. Výchozí projekt by nebylo vhodné migrovat na novější verzi jazyka, kvůli riziku nekompatibility, a proto byl zvolen jiný přístup.

David W. Bishop, jeden z autorů standardu VHDL 2008, vytvořil balíky pro operace s pohyblivou (floating-point) i nepohyblivou (fixed-point) desetinnou čárkou pro VHDL 1993 a zveřejnil[23] je „free of copyright restrictions, and may be used for whatever purpose is needed“, tedy bez omezení copyrightem a použitelné k jakémukoliv účelu.

[1] znovupoužitelný logický obvod, který je duševním vlastnictvím firmy

Pro účely tohoto projektu byl použit pouze balík pro operace s nepohyblivou desetinnou čárkou. Čísla s nepohyblivou desetinnou čárkou nedosahují takové přesnosti, jako čísla s pohyblivou desetinnou čárkou, ale na druhou stranu vyžadují jednodušší a menší hardware design.

3.5 Softing Protocol IP for PROFINET

Softing Protocol IP for PROFINET je implementace standardu PROFINET RT od společnosti Softing Industrial Automation GmbH (dále jen Softing). Projekt navíc, v použité verzi 2015-06-12, podporuje funkcionalitu PROFINET IRT. Pokud není řečeno jinak, údaje uvedené v této sekci byly čerpány z dokumentace [17] a ze zdrojových kódů[24] stacku. Dále jsou také všechny zmiňované soubory uloženy na přiloženém CD.

Pro vývoj a testování dodává Softing balík obsahující potřebné knihovny a ukázkovou aplikaci. V tomto balíku je několik projektů kompatibilních s různým hardware. Konkrétně jde o FPGA vývojové desky:

- Altera Industrial Networking Kit (v projektu označováno jako INK),
- Altera CVE Development Kit (CVE) a
- Altera Cyclone V SoC Development Kit (CVSoC).

A dále PLC:

- Siemens CPU 315,
- Siemens CPU 1516 a
- Phoneix RFC 470.

Funkce ukázkové aplikace je ve všech případech téměř stejná. FPGA slouží jako PROFINET IO-Device a zároveň jako síťový switch (přepínač). Tlačítky na FPGA vývojové desce je možno nastavit rychlost blikání LED (INK, CVE) či rychlost změny délky indikační lišty na LCD (CVSoC). Požadavky na změnu rychlosti však nejsou obsluhovány přímo na FPGA, ale vyhodnocuje je PLC propojené s FPGA přes síť (obr. 11).

V případě této práce se budeme zabývat pouze projektem pro INK. Projekt je dělen do několika částí:

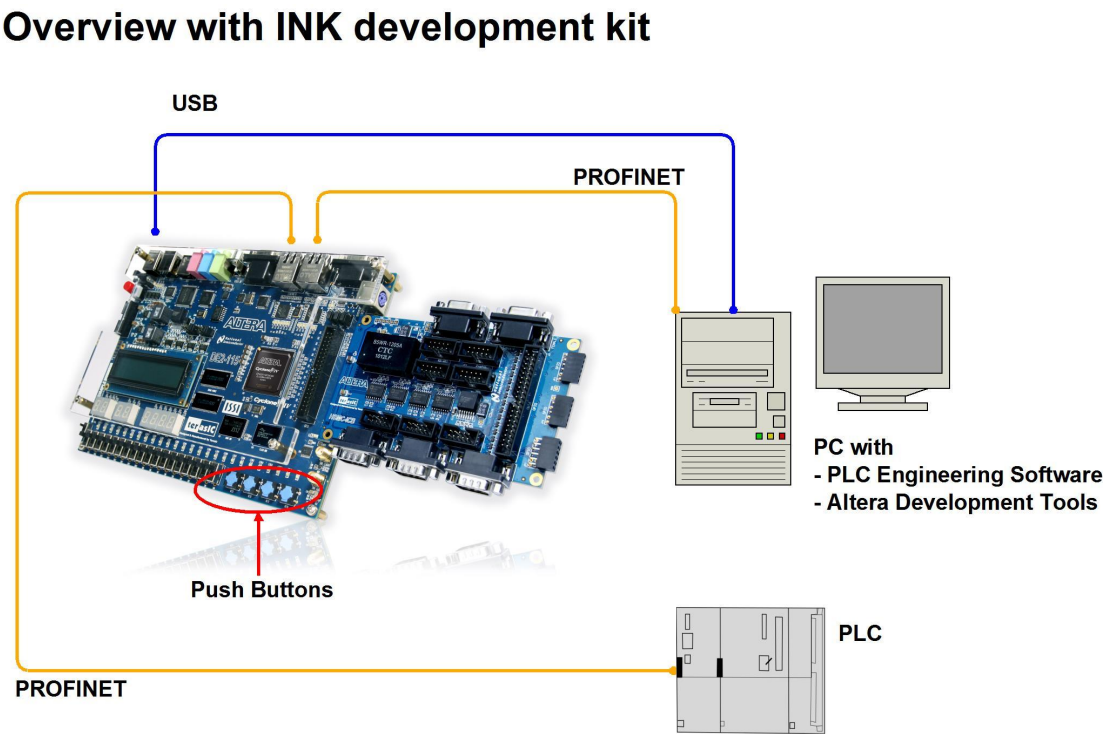
Hardwarová část Tato část obsahuje hardwarový návrh FPGA obvodu. Je dále dělená na dva subsystémy - aplikační a switch, tato část je detailněji popsána v kapitole 3.5.1.

Softwarová část V obou subsystémech (aplikačním i switch) jsou použity CPU Nios II. Softwarová část projektu obsahuje software pro obě tato procesorová jádra. Podrobnosti jsou uvedeny níže v kapitole 3.5.2.

PLC Controller Ukázková aplikace slouží jako PROFINET IO-Device. Pro ovládání tohoto zařízení je tedy třeba IO-Controller, jehož funkci plní PLC. Projekt s konfigurací a jednoduchým programem pro PLC je také dodán v rámci projektu a je popsán v kapitole 3.5.3.

3.5.1 Hardwarový návrh

Hardwarový návrh je projekt pro Altera Quartus II ve verzi 14.0 či 14.1 umístěný v adresáři *profinet / hardware / fpga / profinet / altera_ink_switch* . Základem projektu jsou QSys soubory popisující propojení potřebných subsystémů a IP core. Z těchto QSys souborů je možno vygenerovat hardwarový návrh ve jazyku VHDL či Verilog.



Obrázek 11 Znázornění požadovaného propojení jednotlivých zařízení v rámci INK projektu[17]

Konkrétně se jedná o soubory *altera_ink_appl_subsystem.qsys* a *softing_profinet_device_subsystem.qsys* , které popisují submoduly v rámci aplikačního subsystému, respektive subsystému switche. Soubor *qsys_profinet_system.qsys* popisuje propojení obou subsystémů s periferiemi a mezi sebou.

Hardwarový návrh se skládá ze dvou subsystémů. První částí je aplikační subsystém, což je samotné IO-Device. Jde o procesorové jádro Nios II připojené na periferie, jako je LCD a LED. Dále je Nios II připojen pomocí několika MM-bridge (Memory Mapped bridge) na paměti flash a RAM. Další MM-bridge jej propojuje s druhým subsystémem.

Druhou částí je subsystém síťového switche, což je další procesor Nios II s flash a RAM pamětí zpřístupněnou přes MM-bridge. Významnými komponentami tohoto subsystému jsou také IP core RTE-Switch, což je blok zprostředkovávající samotnou komunikaci, a dále dual port RAM, tj. RAM se dvěma porty, pomocí nichž komunikuje switch s aplikačním subsystémem. Blokový diagram propojení komponent a subsystémů je možno vidět na obrázku 12.

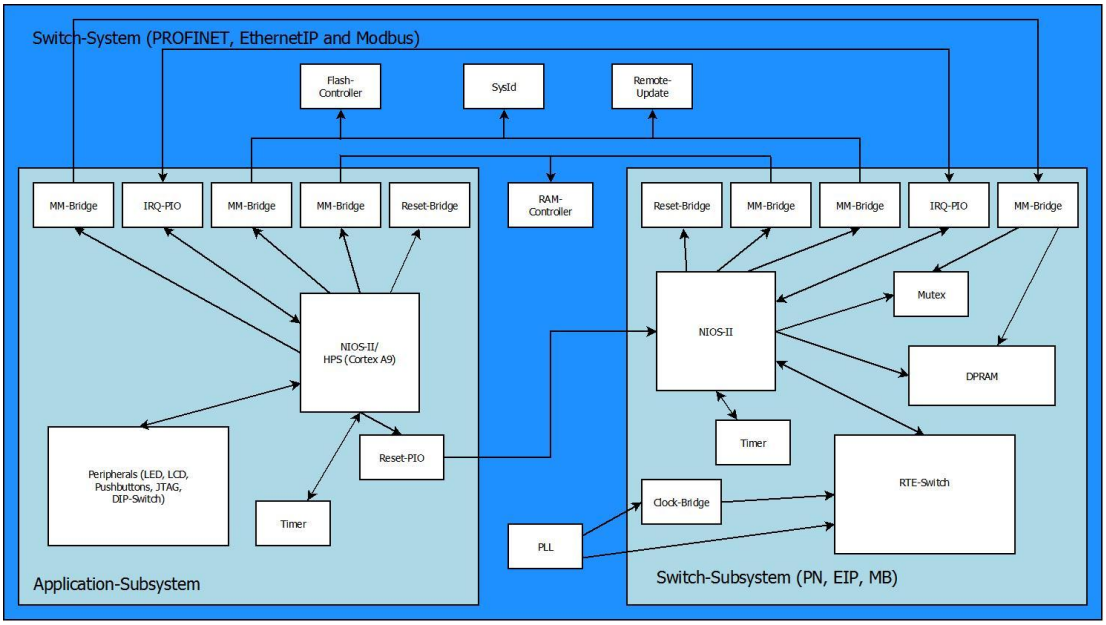
Dalším významným adresářem hardwarové části projektu je adresář *profinet / hardware / ip_core* obsahující v projektu používané IP core. Kromě samotných IP core také obsahuje licenční soubory pro jejich používání.

3.5.2 Demo aplikace

Další částí projektu je ukázková aplikace nazvaná Demo. Aplikace je napsána v jazyce C a jejím účelem je komunikace pomocí PROFINET stacku a obsluha periferií.

Projekt aplikace Demo pro vývojové rozhraní Nios II Software Build Tools for Eclipse (dále jen Nios II IDE) se nachází v podadresáři *appl_altera_hal / application.* Soubory projektu jsou však i v dalších podadresářích, například *appl_common.*

Mezi nejvýznamnější soubory obsahující kód programu patří soubor *appl_common / demo.c* a k němu přidružený hlavičkový soubor *appl_common / demo.h.* Ten



Obrázek 12 Schema propojení bloků a subsystémů stacku[17]

Poznámka 2

Zdrojové kódy aplikace se nacházejí v adresáři *profinet / software / application / altera_ink_switch,* vůči kterému budou uváděny všechny následující cesty k adresářům a souborům.

obsahuje samotnou Demo aplikaci, jejíž hlavní funkce `demo_task_main_function` inicializuje periferie, potřebné struktury knihovny SDAI, zapojí PROFINET moduly a poté spustí hlavní smyčku programu, která zpracovává data přijatá přes síť a obsluhuje periferie.

Dále soubor obsahuje několik funkcí, jejichž název končí slovem `_callback,` které jsou volány při různých událostech v síti PROFINET. Například se jedná o událost vyvolanou při přijetí dat (funkce `demo_update_output_data).`

Poznámka 3

SDAI je společné API (Application Programming Interface) pro různé komunikační protokoly. Toto API ulehčuje programátorovi práci s implementací síťové komunikace a umožňuje mu soustředit se na implementaci samotného programu. SDAI podporuje například protokoly

- PROFINET IO,
- EtherNet/IP,
- EtherCAT,
- Modbus TCP či
- PROFIBUS DP[25].

Dalším významným souborem Demo aplikace je *appl_altera_hal / application / demo_platform.c* a k němu přidružený hlavičkový soubor nacházející se ve stejném adresáři. Tento soubor obsahuje funkce a preprocesorová makra pro obsluhu periferií. Na-

chází se zde například funkce `platform_lcd_clear,` která smaže znaky zobrazené na LCD, nebo funkce `platform_get_buttons` pro čtení stavu tlačítek na desce. V přidruženém hlavičkovém souboru můžeme najít například makra `DATA_LED_SET` a `STATUS_LED_SET` pro nastavení LED indikátorů dat, respektive stavu, nebo konstanty `_TRACE_ENABLE` a `_DEBUG_ENABLE` nastavující úroveň logování na konzoli.

V souboru *appl_altera_hal / application / demo_platform.c* se také nachází funkce `main` spouštěná po zapnutí programu, která spouští funkci `platform_demo_task` ze souboru *appl_common / demo.c.*

Navíc ještě k projektu náleží důležitý soubor *appl_common / profinet.c* . Tento soubor obsahuje pomocné funkce pro komunikaci přes PROFINET. Nachází se zde například funkce `profinet_get_unit_data,` která vrátí seznam modulů, které se mají připojit do příslušných slotů, ty se následně v samotné Demo aplikaci připojí. Další funkcí v tomto souboru je funkce `profinet_set_output_data` pro zpracování přijatých dat. V případě Demo aplikace jde o zapsání stavu LED na výstup procesoru.

Poznámka 4

V souboru *appl_altera_hal / application / demo_platform.c* je přítomen pomocný kód pro práci s PROFINET, který však není použit. Pro zakompilování kódu do výsledného programu by musela existovat konstanta `PROFINET_PLUGIN_ENABLED.` V Makefile *appl_altera_hal / application / Makefile* můžeme vidět, že řádek 176 s definicí je zakomentován.

Board Support Package

Kromě samotného projektu Demo aplikace je ke spuštění potřeba BSP (Board Support Package). Nachází se v adresáři *appl_altera_hal / bsp_ink_pn* a obsahuje knihovny pro práci s periferiemi. Jako příklad lze uvést LCD, flash paměť nebo JTAG řadič.

Software switche

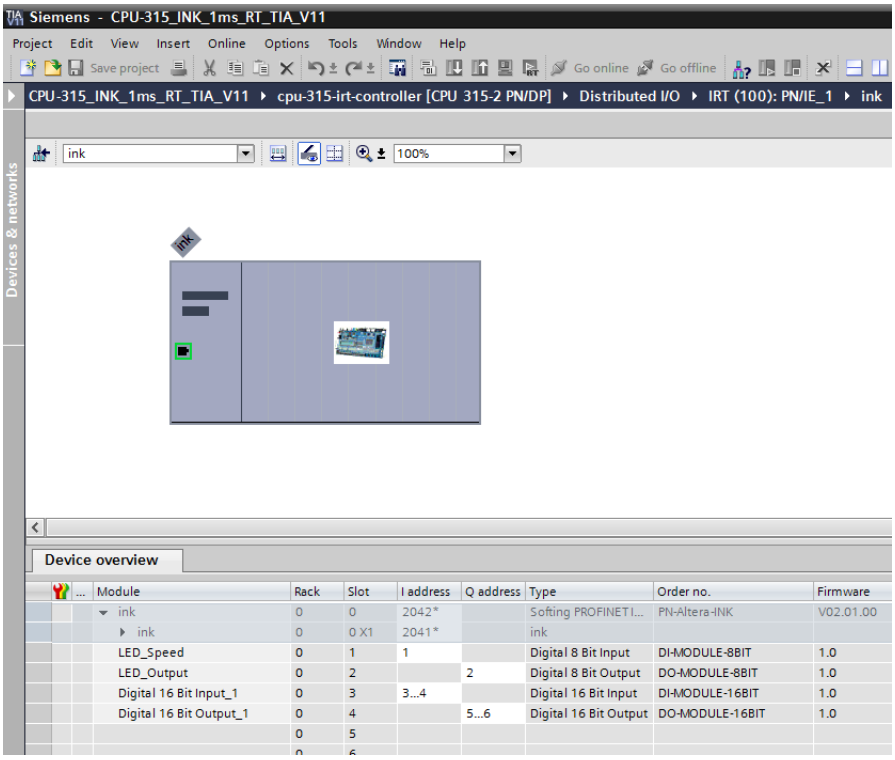
Kromě samotné Demo aplikace je v adresáři *switch_subsystem* umístěn projekt se software pro switch. Do funkcionality switche však nebylo třeba v rámci této práce zasahovat, a proto zde tento projekt nebude popsán.

Přidání projektů do Nios II IDE

Pro správné vytvoření projektů v Nios II IDE je třeba konzultovat dokumentaci[17]. Chybí v ní však zmínka o projektu s BSP (kap. 3.5.2). Projekt s BSP je třeba přidat stejným způsobem, jaký je v dokumentaci uveden pro Demo aplikaci. Všechny tři projekty (Demo aplikace, BSP a software switche) jsou otevřeny ve workspace pro Nios II IDE v adresáři *nios-workspace,* který je umístěn v kořenovém adresáři.

3.5.3 Ukázkový program pro PLC

Jako IO-Controller je ve výchozím projektu používáno PLC s programem, který byl v rámci balíku také dodán. Program můžeme najít v adresáři *profinet / software / plc / profinet / altera_ink_switch* a to hned v několika verzích. Za prvé jde o verze pro PLC od různých výrobců, konkrétně jde o zařízení Siemens a Phoenix. Za druhé pak pro různé modely PLC od těchto výrobců.



Obrázek 13 Mapování vstupů a výstupů na jednotlivé moduly

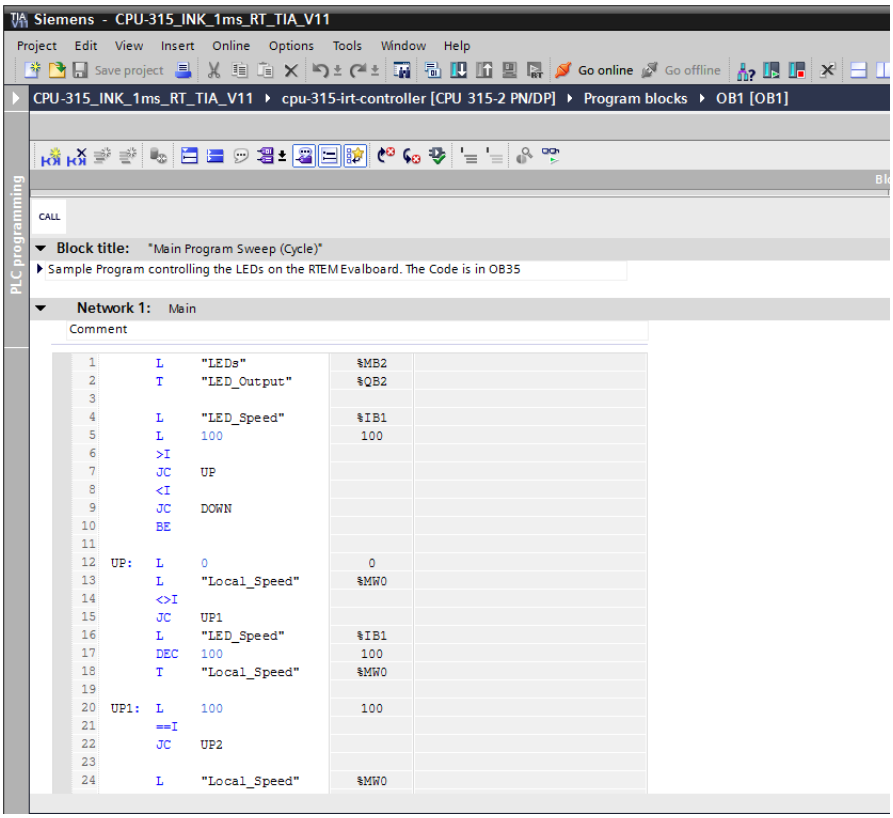
V této práci bylo použito PLC Siemens CPU 315, pro které se projekt nachází v souboru *CPU-315_INK_1ms_RT_TIA_V11.zip* , který se nachází v adresáři *profinet / software / plc / profinet / altera_ink_switch / Siemens / TIA-Portal /.* Vůči tomuto adresáři budou uváděny všechny následující cesty, nebude-li uvedeno jinak.

Jedná se projekt pro vývojové prostředí TIA Portal verze 11 a pro jeho instalaci je vyžadováno několik kroků:

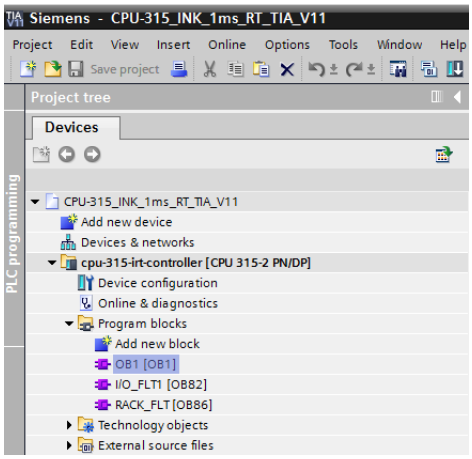
- rozbalit projekt v zip archivu
- otevřít rozbalený projekt v prostředí TIA Portal
- kliknout na položku „Install general station description file (GSD)“ v menu „Options“
- v nově otevřeném dialogovém okně jako „Source path“ zvolit GSD soubor *AdditionalFiles / GSD / GSDML-V2.31-Softing-Altera-INK-20150309.xml* v adresáři projektu
- kliknout na tlačítko „Install“

Jakmile je v rozhraní TIA Portal nainstalován GSD soubor, můžeme v nabídce „Devices & networks“ vidět propojený IO-Controller (PLC) s IO-Device (FPGA). Dvojklikem na IO-Device otevřeme jeho nastavení, v záložce „Device overview“ potom můžeme vidět mapování vstupů a výstupů na moduly. Ukázka mapování modulů je na obrázku 13. Můžeme zde vidět, že vstup „LED_Speed“ je namapován na modulu 1 a výstup „LED_Output“ na modulu 2.

Samotný kód programu (obr. 14) je přístupný v sekci „cpu-315-irc-controller“, pod nabídkou „Program blocks“, jako položka „OB1“ (obr. 15). Program je možné do PLC nahrát pomocí položky „Download to device“ v nabídce „Online“.



Obrázek 14 Rozhraní TIA Portal s otevřeným kódem hlavní smyčky programu



Obrázek 15 Umístění hlavní smyčky programu v nabídce rozhraní TIA Portal

4 Řešení projektu

V závěrečné kapitole bude uveden postup, kterým jsem řešil jednotlivé body bakalářské práce. V první části se zaměříme na popis hardware, tedy například na to, jak jsem řešil komutaci BLDC motoru nebo řízení otáček pomocí PID regulátoru. V druhé části si můžeme přečíst o změnách v softwarové části. Projekt je založen na Demo aplikaci dodané společností Softing Industrial Automation GmbH (dále jen Softing) v rámci balíku, který byl popsán v kapitole 3.5. Demo aplikaci jsem do jisté míry zjednodušil a upravil. V poslední části se zaměříme na jednoduchý program pro PLC, které slouží jako PROFINET IO-Controller.

4.1 Řízení motoru

Řízení motoru jsem plně implementoval v hardware v jazyku VHDL. Jedna z výhod tohoto návrhu je, že každá z popsaných komponent je vytvořena jako znovupoužitelný logický blok a je tedy možné přidat do návrhu celou sestavu několikrát pro řízení více motorů. Tyto bloky navíc vykonávají práci paralelně, a proto nedochází k prodlužování reakční doby při přidávání nových prvků. Počet prvků je však omezen „velikostí“ čipu, konkrétně jde například o počet LE (logic element), nebo paměťových bloků.

Pro regulaci otáček motoru jsem použil PID regulátor, jehož teoretické základy jsou uvedeny v kapitole 2.2. Výstupem regulátoru je číslo určující střídu PWM signálu, který je generován generátorem PWM. Signál následně spíná jednotlivé fáze motoru. Pro měření otáček motoru je použit IRC. Signál z něj je dále zpracován kvadraturním enkodérem. Pro určení toho, kterou konkrétní fázi motoru sepnout, je použit jednoduchý komutátor používající pro určení polohy Hallovy sondy (kap. 2.1.2).

Poznámka 5

Všechny cesty k souborům a adresářům uvedené v této kapitole jsou relativní vůči adresáři *profinet / hardware / fpga / profinet / altera_ink_switch,* není-li uvedeno jinak.

4.1.1 PWM generátor

Účelem generátoru je generovat PWM signál o zadané frekvenci a střídě. Kdyby však byla jako vstup generátoru přímo frekvence a střída, pro výpočet některých mezihodnot by bylo třeba použít operace násobení a dělení, což by výrazně zvýšilo složitost hardwarového návrhu. Blok má proto určenou frekvenci staticky, přímo v kódu. Z frekvence vstupního hodinového signálu a z požadované frekvence PWM signálu je vypočteno číslo, určující maximální počet mezikroků v rámci jedné periody výstupního signálu

$$STEP_{max} = \frac{F_{sys}}{F_{req}}, \tag{9}$$

a) se střídou 63%

b) se střídou 98%

Obrázek 16 Generovaný PWM signál o frekvenci 50 kHz

kde

$STEP_{max}$ je maximální počet kroků za jednu periodu PWM signálu,
F_{sys} je frekvence vstupního hodinového signálu a
F_{req} je frekvence výstupního PWM signálu.

Vstupem PWM generátoru pak je číslo 0 až $STEP_{max}$, které nastavuje střídu dle vztahu

$$DC = \frac{STEP}{STEP_{max}} \cdot 100, \tag{10}$$

kde

DC je střída a
STEP je vstupní hodnota generátoru.

Pro výpočet frekvence při zadaném maximálním počet kroků rovnému 1000 a frekvenci vstupního hodinového signálu 50 MHz, dle patřičně upraveného vztahu 9, dostáváme

$$F_{req} = \frac{50 \cdot 10^6}{1000} = 50000\,Hz \tag{11}$$

a například pro vstup 300 dostáváme střídu

$$DC = \frac{300}{1000} = 30\%. \tag{12}$$

Interně generátor obsahuje čítač, který je při každé náběžné hraně vstupního hodinového signálu inkrementován o jedna. Ve výchozím stavu je výstupní signál (`pwm_out`) nastaven na log. 1. Jakmile hodnota čítače překročí vstupní hodnotu určující DC (vstupní port `duty_cycle`) je výstupní signál nastaven na log. 0. Hodnota je na výstupu, dokud čítač nepřekročí hodnotu maximálního počtu kroků. Po jejím překročení je na výstupu opět nastavena log. 1, čítač je resetován na hodnotu 0 a celý cyklus se opakuje.

Tento blok jsem převzal z příbuzného projektu popsaného v [21]. Oproti původní implementaci od T. Ryčla jsem přidal chování pro zápornou střídu, pro

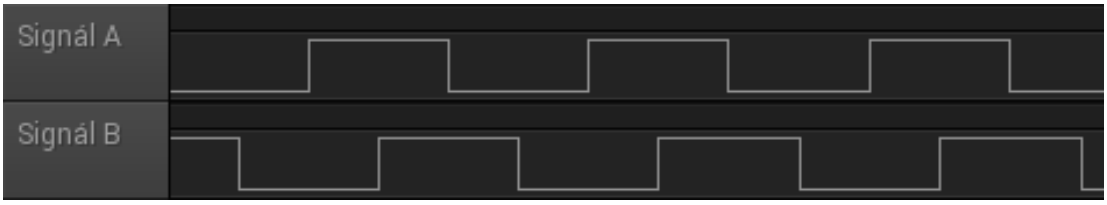
$$DC < 0, DC := -DC. \tag{13}$$

Dále jsem přesunul definice konstant určující parametry generátoru do generik, tím bylo usnadněno parametrizování projektu. Nakonec byla ještě přejmenována entita tak, aby odpovídala pojmenovávacímu vzoru ostatních komponent projektu.

Zdrojový kód komponenty je možno najít v souboru *utils / PWMGenerator.vhdl* .

Signál A	Signál B	Stav
0	0	1
0	1	2
1	0	3
1	1	4

Tabulka 3 Stavy a jejich signály při rotaci po směru hodinových ručiček



Obrázek 17 Průběhy výstupních signálů A a B inkrementálního rotačního enkodéru

4.1.2 Kvadraturní dekodér

Pro určení natočení motoru je použit inkrementální rotační enkodér. Při otáčení jsou generovány dva obdélníkové signály, A a B, vůči sobě posunuté o 90°. Ty jsou dekódovány a je z nich určena relativní pozice rotoru a směr rotace. Dva binární signály mohou dohromady dát 4 stavy, viz tabulka 3. Průběh signálů A a B je znázorněn na obrázku 17.

Vstupem dekodéru je hodinový signál, signály A, B a navíc signál `INDEX` indikující jednu plnou mechanickou otáčku. Vstupní signály A, B a `INDEX` mají ošetřeny zákmity a dále jsou synchronizovány se vstupním hodinovým signálem.

Při každé náběžné hraně hodinového signálu jsou porovnány aktuální stavy signálů A a B se stavy z předchozího cyklu. V případě, že se liší, je dle současných a minulých stavů určen směr rotace. Ten je nastaven na výstup a dále je výstupní čítač, v závislosti na směru, inkrementován, či dekrementován o jedna.

Tuto entitu jsem převzal, stejně jako PWM generátor, z [21]. V jejím kódu však byly provedeny změny. Entitu jsem přejmenoval tak, aby pojmenovávací vzor odpovídal ostatním komponentám v projektu. Dále jsem přidal výstup `cnt_event`, který má při změně stavu vstupních signálu hodnotu log. 1 po dobu jedné periody hodinového signálu.

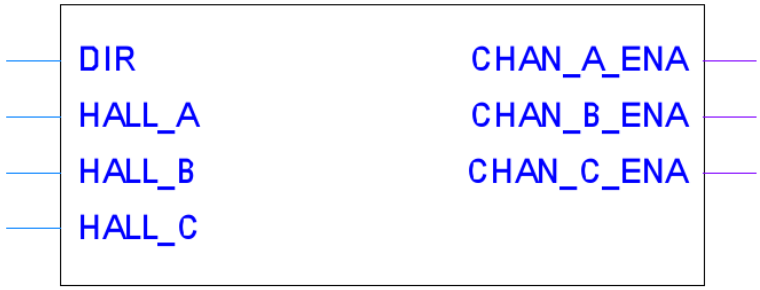
Zdrojový kód se nachází v adresáři *utils / QuadCount.vhdl* .

4.1.3 Implementace komutátoru

Protože cílem práce bylo primárně pochopení výchozího PROFINET projektu a úpravy v něm, pro komutaci jsem zvolil pouze jednoduchou metodu určování polohy pomocí Hallových sond. Na základě stavů jednotlivých sond je určena poloha rotoru vůči statoru a dle polohy je sepnuta příslušná fáze motoru (více v kapitole 2.1.2). Takovouto funkčnost je možno implementovat jako kombinační obvod.

Vstupem komutátoru jsou stavy tří Hallových sond a požadovaný směr rotace. Výstupem jsou signály spínající jednotlivé fáze motoru. Na obr. 18 můžeme vidět symbolické označení tohoto komutátoru. Název entity reprezentující komutátor je `CommutatorHALL` a její definice je uložena v souboru *regulator / CommutatorIRC.vhd.*

Pro implementaci komutátoru v jazyce VHDL byla použita konstrukce `with-select`, v bloku Ukázka kódu 1 můžeme vidět její konkrétní použití.



Obrázek 18 Symbol bloku CommutatorHALL

Ukázka kódu 1 Použití konstrukce with-select pro řízení komutace

```
with HALL select CHAN_ENA <=
        -- DIR = '0'
        "001" when "0001",
        "100" when "0010",
        "100" when "0011",
        "010" when "0100",
        "001" when "0101",
        "010" when "0110",
        -- DIR = '1'
        "010" when "1001",
        "001" when "1010",
        "010" when "1011",
        "100" when "1100",
        "100" when "1101",
        "001" when "1110",

        "000" when others;
```

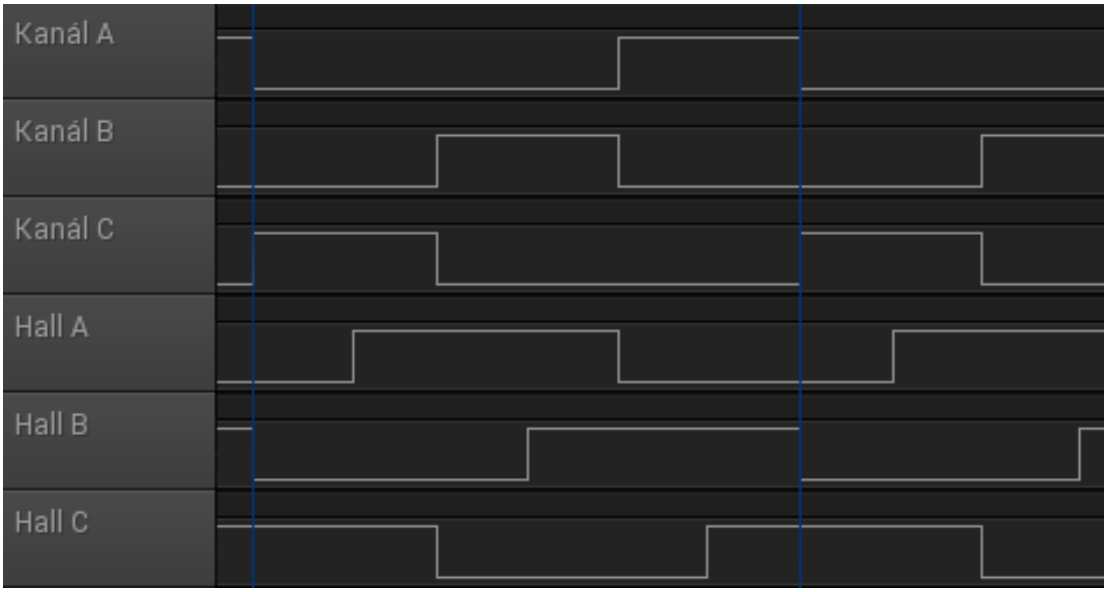
Vstupem, respektive výstupem konstrukce, může být pouze jeden signál, proto navíc entita obsahuje sloučení vstupní čtveřice signálů do vektoru a dále převod výstupního vektoru konstrukce do jednotlivých výstupních signálů entity. Vstupní vektor konstrukce `HALL` se skládá (od MSB k LSB) ze signálů

1. DIR,
2. HALL_A,
3. HALL_B a
4. HALL_C,

výstupní vektor konstrukce `CHAN_ENA` ze signálů

1. CHAN_A_ENA,
2. CHAN_B_ENA a
3. CHAN_C_ENA.

Na obrázku 19 můžeme vidět proces komutace zaznamenaný pomocí logické sondy. Všimněme si svislých modrých čar značící jednu periodu elektrické otáčky.



Obrázek 19 Změřené hodnoty vstupů (Kanál A,B,C) a výstupů (Hall A,B,C) komutátoru při rotaci motoru. Svislá modrá čára označuje začátek jedné elektrické otáčky.

4.1.4 Implementace PID regulátoru

Jedním z cílů projektu byla implementace PI regulátoru v jazyce VHDL. Po dohodě s vedoucím práce jsem však implementoval PID regulátor. Toto rozhodnutí nijak výrazně nezvýšilo složitost implementace, ani nezvětšilo hardwarový návrh a regulátor je díky tomu možné nasadit v dalších aplikacích. Pokud by bylo potřeba použít PI regulátor, stačí nastavit konstantu zesílení derivační složky K_d na hodnotu 0.

Regulátor disponuje tracking-back anti-windup mechanizmem a možností měnit konstanty zesílení (K_p, K_i, K_d) za běhu. Hodinový signál je generován externě a při použití komponenty `ClockDividerVariable` (kap. 4.1.6) je možné za běhu nastavovat i frekvenci řídící smyčky. Regulátor je reprezentován entitou s názvem `RegulatorPID` a je umístěn v souboru *regulator / RegulatorPID.vhd.*

Některé parametry regulátoru je možno nastavovat přes generika. Generika mají své výchozí hodnoty a navíc je jejich hodnotu možné nastavit z rodičovské entity. V projektu jsou nastavené a výchozí hodnoty shodné (uvedeny jsou v tabulce 4). V krátkém seznamu je seznam generik s jejich názvy a popisem:

PWM_DC_MAX_INTEGER Horní hranice vstupní hodnoty PWM generátoru (použito v anti-windup mechanizmu). Pokud u konstanty změníme znaménko, dostáváme spodní hranici výstupní hodnoty PWM generátoru.

DECIMAL_POINT_BITS Velikost desetinné části fixed-point čísel v bitech.

CALC_BITS Velikost některých vstupů a interních hodnot pro mezi výpočty v bitech.

VAL_BITS Velikost vstupů pro úpravu regulačních konstant a interních proměnných uchovávajících konstanty v bitech (včetně desetinné části).

Vstupem bloku je hodinový signál, referenční a naměřená hodnota a vstupy pro úpravu konstant. Výstupem je akční zásah regulátoru. Nejdříve se podívejme na vstupní

Název generika	Nastavená hodnota
PWM_DC_MAX_INTEGER	1000
DECIMAL_POINT_BITS	7
CALC_BITS	32
VAL_BITS	21

Tabulka 4 Nastavené hodnoty generik bloku RegulatorPID

porty. Vstupy `REFERENCE` a `STATE_MEAS` jsou čísla typu signed (tedy celá čísla se znaménkem) o velikosti `CALC_BITS` bitů. Vstupy jsou použity pro předání požadované referenční hodnoty, respektive naměřené hodnoty odpovídající aktuálnímu stavu. Hodinový signál určující frekvenci řídící smyčky musí být přiveden na vstupní port `REGULATOR_CLOCK` . Jednotky referenční a změřené hodnoty mohou být libovolné, ale je potřeba, aby byly u obou vstupů stejné. Více k použitým jednotkám v kap. 4.1.5.

Poznámka 6

Čísla typu „signed“ o velikosti n bitů mají rozsah -2^{n-1} až $2^{n-1} - 1$.

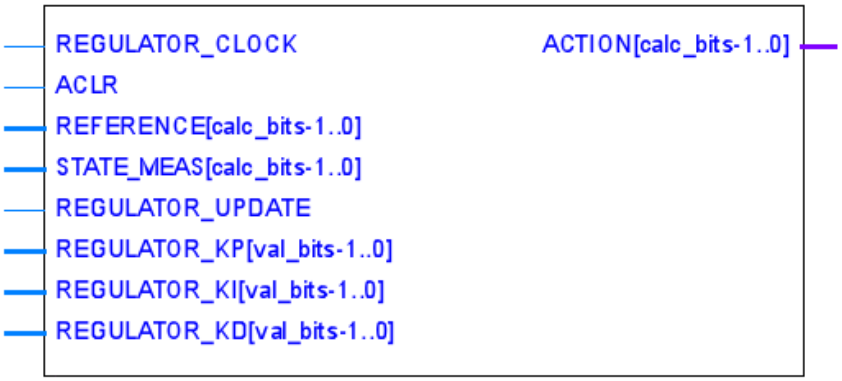
Porty `REGULATOR_KP,` `REGULATOR_KI` a `REGULATOR_KD` slouží k nastavování konstant regulátoru. Jsou to čísla typu „signed“ o velikosti `VAL_BITS` . Vstupní hodnoty musí být binárně posunuty o `DECIMAL_POINT_BITS` vlevo, neboť jsou v dále převedeny na datový typ „sfixed“ z balíku `fixed_pkg` (kap. 3.4), který reprezentuje „fixed-point“ desetinná čísla. Výsledná velikost je tedy `VAL_BITS - DECIMAL_POINT_BITS` a číslo je uloženo s přesností `DECIMAL_POINT_BITS` desetinných míst ve dvojkové soustavě. Po přivedení log. 1 na vstup `REGULATOR_UPDATE` se nové hodnoty asynchronně s řídící smyčkou uloží do dočasných proměnných a při další náběžné hraně hodinového signálu jsou řídící smyčkou použity.

Poznámka 7

Čísla typu „sfixed“ o velikosti n bitů s d bity pro desetinnou část čísla mají rozsah -2^{n-1} až $2^{n-1} - 1$ a reprezentují čísla s přesností na d desetinných míst ve dvojkové soustavě. Celková velikost proměnné (či signálu) je $n + d$ bitů. Zápis v jazyce VHDL je následovný:

```
sfixed(n downto -d)
```

Samotná regulace je prováděna v procesu `regulator` . Nejprve je vypočtena aktuální chyba, ta se poté vynásobí a přičte k současné hodnotě integrátoru (integrátor tedy obsahuje hodnoty již vynásobené konstantou K_i). Následně se vypočte akční zásah a je ověřeno, zda akční zásah nepřesahuje stanovené limity a případně se provede korekce integrátoru a výstupního signálu. Zjednodušený kód PID regulátoru můžeme vidět v bloku Ukázka kódu 2. V kódu projektu jsou navíc podmínky ošetřující různé stavy, konverze datových typů a změny velikostí proměnných.



Obrázek 20 Symbol bloku RegulatorPID

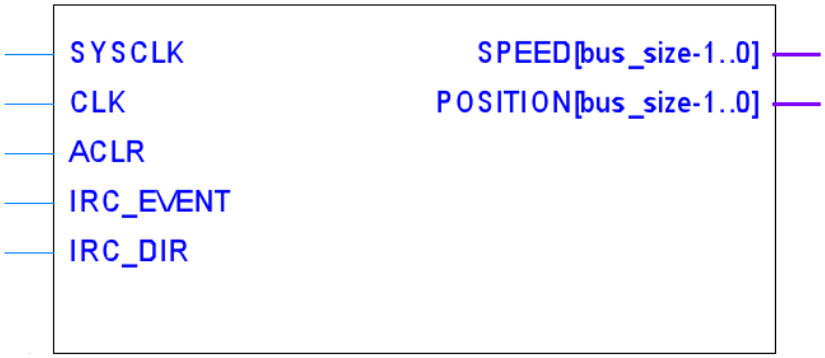
Ukázka kódu 2 Zjednodušená ukázka kódu PID regulátoru v jazyce VHDL

```
-- vypocet chyby
error := ACTUAL_STATE_in - REFERENCE_in;
-- pricteni soucasne chyby k hodnote integratoru
integral := integral + error * REGULATOR_KI_in;
-- vypocet akcniho zasahu
action_required := REGULATOR_KP_in * error + integral +
  (error - error_last) * REGULATOR_KD_in;

-- anti-windup mechanizmus
if action_required > PWM_DC_MAX then
  action_bound := PWM_DC_MAX;
elsif action_required < PWM_DC_MIN then
  action_bound := PWM_DC_MIN;
else
  action_bound := action_required;
end if;

integral := integral -
  (action_required - action_bound) * REGULATOR_KI_in;

-- ulozeni akcniho zasahu na vystup
ACTION <= to_integer(action_bound) / (2 ** DECIMAL_POINT_BITS);
```

4.1.5 Výpočet rychlosti

Výpočet aktuální rychlosti je prováděn v bloku `Differentiator`, (obr. 21), jehož zdrojový kód je v souboru *regulator / Differentiator.vhd*. Pro získání aktuální rychlosti jsou použity výstupy quadraturního dekodéru `cnt_event` (změna stavu čítače) a `cnt_way` (směr poslední změny čítače).

Při změně stavu čítače dekodéru je inkrementován (nebo dekrementován, podle směru rotace) vnitřní čítač. Při náběžné hraně vstupního signálu `CLK` je hodnota čítače zapsána na výstup `SPEED` a hodnota interního čítače je nastavena na nulu. Výstupní jednotkou rychlosti je tedy počet IRC událostí za jednu periodu hodinového signálu. V



Obrázek 21 Symbol bloku Differentiator

projektu je použit hodinový signál o frekvenci 1 kHz a rychlost je tedy udávána v počtu IRC událostí za 1 ms.

Blok také počítá relativní pozici od 0 do `IRC_MAX`, což je maximální počet IRC událostí za jednu mechanickou otáčku.

4.1.6 Generování hodinového signálu

Pro generování hodinového signálu jsou v projektu přiloženy dva bloky. Prvním z nich je `ClockDividerVariable`, druhým je `ClockDivider`. Použit je pouze první z nich.

Vstupem obou bloků je hodinový signál, jehož frekvence je následně snižována pomocí prescaleru. Vstupní hodinový signál s každou náběžnou hranou inkrementuje hodnotu vnitřního čítače. Pokud hodnota přesáhne určitou hranici, je čítač resetován na hodnotu 0 a výstupní signál je invertován (Uukázka kódu 3).

Ukázka kódu 3 Část kódu bloku ClockDivider

```
if rising_edge(CLKSYS) then
  if CNT = (PRESCALER-1) then
    CNT := 0;
    CLK_in <= not CLK_in;
  else
    CNT := CNT + 1;
  end if;
end if;
```

V případě `ClockDividerVariable` je hodnota prescaleru určována vstupem `PRESCALER_SIZE`. U druhého typu bloku je hodnota určována konstantou vypočtenou ze zadaných generik. Hodnota prescaleru je v tomto případě po kompilaci neměnná (lze ji nastavit pouze v kódu).

Výstupní frekvence závisí na vstupní frekvenci a velikosti prescaleru. Hledanou hodnotu prescaleru pro požadovanou výstupní frekvenci můžeme vypočítat dosazením do vztahu

$$P = \frac{F_i}{2F_o}, \tag{14}$$

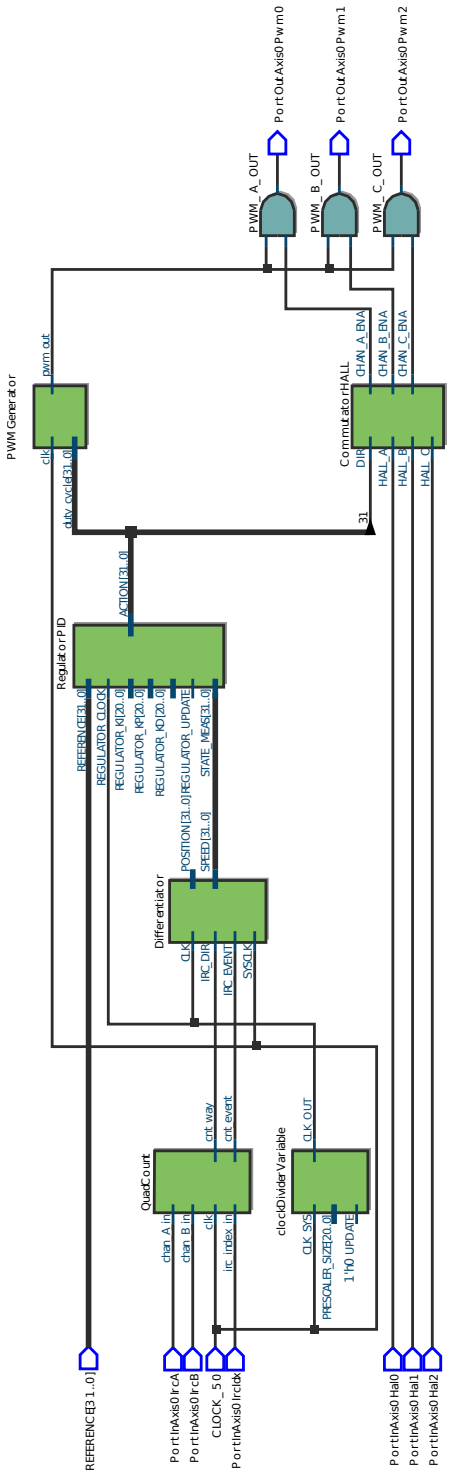
kde

P je hledaná hodnota velikosti prescaleru,
F_i je vstupní frekvence a
F_o je výstupní frekvence.

4.1.7 Propojení hardwarových bloků

Bloky popsané v předchozích podkapitolách jsou propojeny v top-level entitě `altera_ink_pn`. V této entitě je také umístěn subsystém `qsys_profinet_system` obsahující Nios II procesory pro aplikaci a switch popsané v kapitole 3.5.1. Schéma je možno vidět na obrázku 22.

Vstup `REFERENCE` je připojen k příslušnému výstupu procesoru, ostatní vstupy a výstupy jsou namapovány na piny FPGA čipu.



Obrázek 22 Propojení jednotlivých bloků hardwarového návrhu. Pro zlepšení přehlednosti schema neobsahuje některé vstupy a výstupy.

4.2 PROFINET aplikace

Pro komunikaci jsem upravil a do jisté míry zjednodušil Demo aplikaci od společnosti Softing pro vývojovou desku Terasic DE2-115, která je součastí Altera Industrial Networking Kitu. Softingu tuto sadu označuje jako „altera_ink.“

Poznámka 8

Všechny uvedené soubory a adresáře jsou uváděny vůči adresáři *profinet / software / application / altera_ink_switch,* není-li uvedeno jinak.

4.2.1 Úprava funkčnosti Demo aplikace

V první fázi bylo potřeba změnit funkčnost aplikace tak, aby sloužila k ovládání PID regulátoru ve VHDL pomocí IO-Controlleru popsaného v kap. 4.3.

Výchozí aplikace fungovala tak, že uživatel nastavil tlačítky na desce s FPGA rychlost spínání LED (stisk tlačítek inkrementoval či dekrementoval hodnotu proměnné v Demo aplikaci), tuto rychlost si četlo PLC a dle její hodnoty měnilo hodnotu výstupu Demo aplikace. Ta přijatou hodnotu zapsala na výstup připojený k LED.

Nejdříve bylo třeba upravit připojené moduly IO-Device a přepsat kód nastavování vstupu a výstupu tak, aby splňoval požadavky projektu. To je možné v souboru *appl_common / profinet.c,* konkrétně ve funkci `profinet_get_unit_data`. Tato funkce je volána Demo aplikací při inicializaci a vrací seznam modulů, které mají být připojeny. Připojených vstupních a výstupních modulů bylo osm a polovina z nich nebyla uvedena v GSD souboru, ani v dokumentaci. Všechny moduly jsem odstranil a nahradil jsem je dvěma moduly o velikosti 32 bitů. První z nich je vstupní, druhý výstupní. Část kódu je uvedena v bloku Ukázka kódu 4.

Ukázka kódu 4 Ukázka kódu funkce profinet_get_unit_data

```
/* plug the head module it must be in Slot 0, Subslot 1*/
profinet_add_unit (pUnitManagement, SDAI_UNIT_TYPE_HEAD, 0, 0, 0
    x00000200, 0x00000000, _SDAI_PN_SLOT_AND_SUBSLOT_TO_ID(0, 1),
    SDAI_PN_UNIT_FLAG_SUPPORT_MODULE_SUBSTITUTION,
    SDAI_PN_DEFAULT_PROFILE, SDAI_PERIPHERAL_OFFSET_DPRAM,
    SDAI_PERIPHERAL_OFFSET_DPRAM);

/* plug 32 Bit input unit - used as reference input (to plc)*/
profinet_add_unit (pUnitManagement, SDAI_UNIT_TYPE_GENERIC_INPUT, 4, 0,
    0x00000103, 0x00000000, _SDAI_PN_SLOT_AND_SUBSLOT_TO_ID(1, 1), 0u,
    SDAI_PN_DEFAULT_PROFILE, SDAI_PERIPHERAL_OFFSET_DPRAM,
    SDAI_PERIPHERAL_OFFSET_DPRAM);

/* plug 32 Bit output unit - used as reference output (to regulator)*/
profinet_add_unit (pUnitManagement, SDAI_UNIT_TYPE_GENERIC_OUTPUT, 0, 4,
     0x00000203, 0x00000000, _SDAI_PN_SLOT_AND_SUBSLOT_TO_ID(2, 1), 0u,
    SDAI_PN_DEFAULT_PROFILE, SDAI_PERIPHERAL_OFFSET_DPRAM,
    SDAI_PERIPHERAL_OFFSET_DPRAM);
```

Následně jsem musel změnit, opět v souboru *appl_common / profinet.c,* chování při čtení a zápisu dat. Nejprve jsem přejmenoval všechny proměnné a funkce tak, aby názvy popisovaly skutečnou funkčnost aplikace. Tedy například proměnnou `LEDSpeed` jsem přejmenoval na `Axis0Reference`. Poté bylo potřeba ve funkci

`profinet_set_output_data` poskládat přijatou výstupní pole hodnot o velikosti 4 krát 1 byte do proměnné typu integer a tu následně zapsat na výstup pomocí makra `AXIS0_SET_REFERENCE` definovaného v souboru *appl_altera_hal / application / demo_platform.c.*

Nakonec jsem změnil maximální hodnotu, na kterou je možno nastavit rychlost otáčení motoru. Inkrementace a dekrementace je prováděna ve stejném souboru, jako zpracování vstupů a výstupů, konkrétně ve funkci `profinet_button_event.`

Kromě změny funkčnosti bylo třeba nastavit v souboru *appl_common / application_config.S* IP adresu a název zařízení. Obě hodnoty je možné nastavit i přes rozhraní TIA Portal, ale po každém nahrání Demo aplikace do procesoru je potřeba tyto hodnoty opětovně nastavit.

Musím zmínit, že ačkoliv bez správného názvu zařízení komunikace nefunguje, o potřebě změny názvu není v dokumentaci [17] zmínka. Název zařízení nastavený v TIA Portal musí být shodný s názvem nastaveným v tomto souboru.

4.2.2 Zjednodušení kódu

Demo aplikace slouží k ukázce komunikaci nejen v síti PROFINET, ale i v dalších sítích, jako jsou například PROFIBUS DP, nebo EtherCAT. Protože ale naše aplikace bude fungovat pouze v síti PROFINET, je možné podporu ostatních protokolů odstranit, čímž se výrazně sníží složitost aplikace.

Soubory s funkcemi protokolů

Nejprve jsem tedy odstranil soubory s pomocnými funkcemi nepoužitých protokolů, konkrétně se jednalo o soubory:

- *ethercat.c,*
- *ethernetip.c,*
- *modbus.c,*
- *powerlink.c* a
- *profibus.c*

v adresáři *appl_common.* Poté bylo ještě třeba odstranit požadavek na tyto soubory při kompilaci v souboru *Makefile.*

Demo aplikace

Dalším krokem bylo odstranění zmínek o ostatních protokolech ze souboru *appl_common / demo.c.* Kromě inicializace, na jejímž začátku jsou do struktury ukládány ukazatelé na funkce zvoleného protokolu, nikde nebyla přímo reference na žádnou z funkcí protokolů. Pro volání těchto funkcí bylo použito makro `CALL_PROTO_FUNC.` Všechny výskyty makra byly nahrazeny přímým odkazem na pomocnou funkci protokolu PROFINET.

Platforma

Posledním vyčištěným souborem byl soubor *appl_altera_hal / demo_platform.c.* Ten obsahoval nepoužité pomocné funkce s předponou `platform_profinet,` které byly navíc ohraničeny preprocesorovým makrem `#ifdef` vyhodnoceným jako `FALSE,` a tedy nebyly zakompilovány do výsledného programu. Tyto funkce jsem z kódu odstranil.

4.2.3 Změny v Nios II subsystému

Pro umožnění zápisu přijaté referenční hodnoty na vstup regulátoru bylo potřeba vytvořit příslušný výstupní port aplikačního procesoru Nios II.

V subsystému `altera_ink_appl_subsystem,` který se nachází v souboru *profinet / hardware / fpga / profinet / altera_ink_switch / altera_ink_appl_subsystem.qsys,* jsem kvůli tomu přidal výstupní port typu Parallel IO s názvem `pio_axis0_reference` o velikosti 32 bitů. Tento port jsem namapoval na adresní rozsah, připojil jsem ho na sběrnici připojenou k CPU a nakonec jsem ho vyexportoval, aby byl přístupný z rodičovského subsystému `qsys_profinet_system.`

Dále bylo třeba vyexportovat tento signál z rodičovského subsystému, pro který jsem nakonec vygeneroval zdrojové soubory. Rodičovský subsystém je umístěn v top-level entitě hardware návrhu a vyexportovaný výstup je napojen na referenční vstup PID regulátoru.

Problémy se subsystémem

Ve výchozím projektu nebylo možné vygenerovat žádný ze subsystémů. Pro nápravu bylo potřeba u komponenty `clk_system` v rodičovském subsystému označit hodnotu „Clock frequency is known“.

Dále byl ještě odhalen problém s laděním aplikace přes Nios II EDS. Ladění nebylo možné kvůli nepřítomnosti mostu mezi JTAG a Avalon sběrnicí v subsystému `altera_ink_appl_subsystem.` Přidáním komponenty typu `JTAG to Avalon Master Bridge` s názvem `jtag_mm_ink_switch` a připojení této komponenty ke komponentě `jtag_uart` byl problém úspěšně vyřešen.

Oba zmíněné problémy vyřešil vedoucí práce Ing. Richard Šusta, Ph.D., kterému za to děkuji.

4.3 PLC program

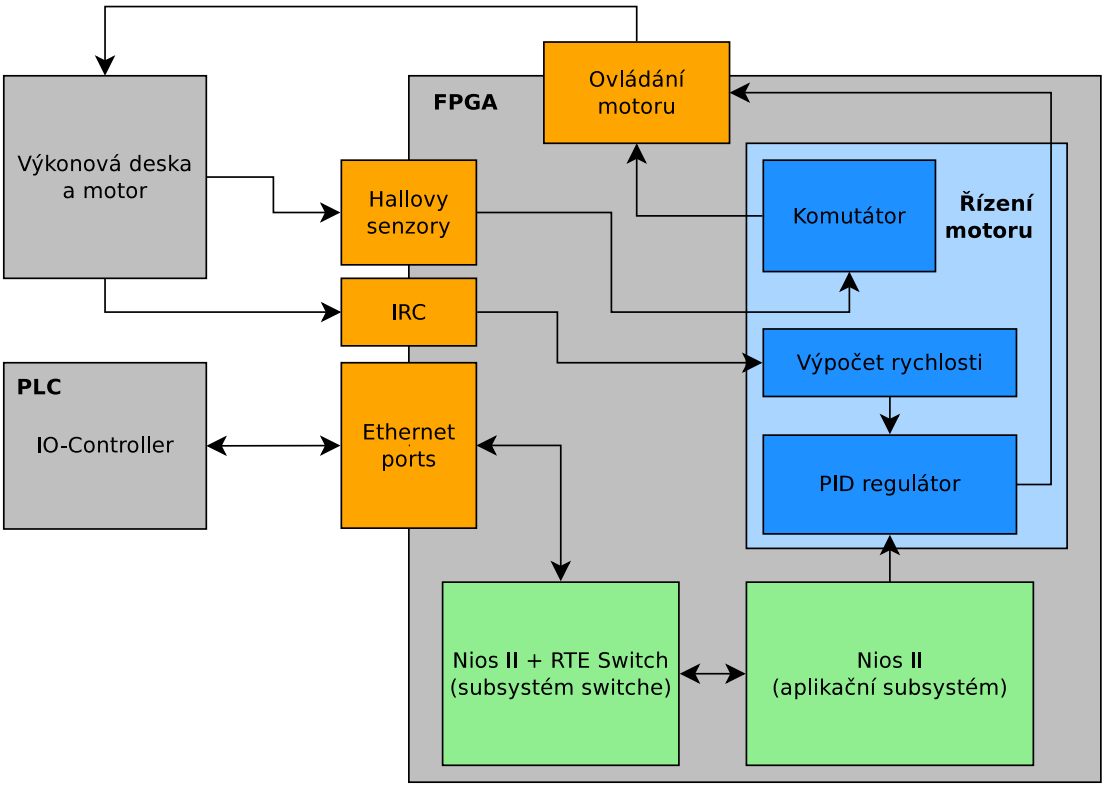
Program pro PLC vychází z projektu pro Siemens CPU 315 v rozhraní TIA Portal, který je popsaný v kap. 3.5.3. Z projektu jsem však použil pouze nastavení a namísto dodaného kódu programu jsem napsal vlastní kód, který načte hodnotu ze vzdáleného vstupu a poté ji zapíše na vzdálený výstup na FPGA.

Kromě samotného programu jsem upravil i přiložený GSD soubor a PLC tagy, které jsem namapoval na správné adresy modulů tak, jak jsou popsány v GSD souboru. Místo IO portů používaných Demo aplikací (8 bitový vstup a výstup pro LED a nepoužitý 16 bitový vstup a výstup) jsem přidal dva porty - vstupní a výstupní port, každý o velikosti 32 bitů.

4.4 Struktura projektu

Jednotlivé komponenty jsem na závěr propojil v jeden funkční celek pro jednoduché řízení motoru. Referenční hodnota nastavená tlačítky na FPGA je předána PLC, které hodnotu odešle zpět do FPGA. Demo aplikace na FPGA přijme referenční hodnotu a zapíše ji na vstup PID regulátoru. Následně je generován PWM signál, který je přiváděn na jednotlivé fáze motoru. Fázi motoru, která má být sepnuta, určuje komutátor dle polohy rotoru vůči statoru, která je určená signálem z Hallových sond. Pro zpětnovazební regulaci je použita rychlost počítaná na základě signálu z IRC.

Zjednodušené blokové schema můžeme vidět na obr. 23.



Obrázek 23 Blokové schema popisující propojení jednotlivých komponent. Zeleně označeny jsou subsystémy s Nios II procesory, modře hardwarové bloky pro řízení motoru a oranžově vstupní a výstupní porty FPGA.

5 Závěr

Cílem práce bylo vyvinout zařízení typu IO-Device pro řízení motoru pomocí FPGA a jeho ovládání přes síť PROFINET. Na základě ukázkové aplikace a knihovny SDAI od společnosti Softing Industrial Automation GmbH jsem realizoval aplikaci, která se vykonává na soft-core procesoru Nios II.

Původní ukázková aplikace podporuje vícero protokolů, kromě PROFINET například PROFIBUS DP a EtherCAT. Protože v projektu je potřeba pouze komunikace přes PROFINET, byla z aplikace odstraněna podpora přebytečných protokolů, čímž se výrazně zvýšila přehlednost kódu.

Kromě „čištění“ kódu aplikace jsem také změnil její funkčnost. Namísto regulace rychlosti přepínání LED je měněna rychlost motoru, což obnášelo, kromě změn chování při přijetí dat, úpravu programu pro IO-Controller, který je implementován na PLC. Přijatá data se zapisují na výstup procesoru, a dále zpracovávají v hardwarové části.

V hardwarové části jsem vytvořil PID regulátor, jednoduchý komutátor a další potřebné jednotky, jako například PWM generátor nebo jednotku pro výpočet rychlosti. Řízení motoru díky tomu nezatěžuje CPU, takže může komunikovat s IO-Controllerem téměř bez přerušení.

Projekt poslouží jako platforma pro implementaci složitějších zařízení. Složitější zařízení může mít například implementovanou podporu PROFIdrive, pokročilejší komutace s určováním polohy za pomoci inkrementálního čítače, či podporu více motorů.

Největší překážkou při řešení projektu byla neúplnost dokumentace [17] a absence programátorského manuálu. Dokumentace [17] je spíše uživatelskou příručkou popisující instalaci výchozí aplikace. Několik potřebných kroků navíc není vůbec zmíněno. Absence programátorského manuálu byla hlavním důvodem zjednodušení aplikace. Jde o relativně složitý projekt a orientace v něm mi zabrala značnou dobu. Odebráním nepoužitých protokolů a funkcí pro nepoužité hardwarové platformy jsem výrazné zvýšil přehlednost, což by mělo pomoci při používání této platformy.

Příloha A

DVD

Na přiloženém DVD jsou uloženy následující soubory:

- Text práce v souboru *BP_Novak_Ondrej.pdf,*
- Zdrojové kódy projektu pro FPGA, Nios II procesor a PLC v adresáři *profinet*
- Workspace pro Nios II EDS v adresáři *nios-workspace*

Zdrojové soubory jsou navíc přístupné přes GIT repozitář

https://gitlab.fel.cvut.cz/novako19/pid-profinet.git

Literatura

[1] M. Prudek. "Řízení bezkartáčových motorů s deskou Raspberry Pi a Linuxem". Bak. ČVUT, 2015.

[2] A. Sathyan; N. Milivojevic; Y. J. Lee; M. Krishnamurthy; A. Emadi. "An FPGA-Based Novel Digital PWM Control Scheme for BLDC Motor Drives". In: *IEEE Transactions on Industrial Electronics* 56 (2009), s. 3040–3049.

[3] J. Zhao; Y. Yu. *Brushless DC Motor Fundamentals Application Note*. Čvc 2011. URL: `https://www.monolithicpower.com/Portals/0/Documents/Products/Documents/appnotes/Brushless%20DC%20Motor%20Fundamentals.pdf`.

[4] M. M. Nikolic; N. U. Pjevalica; J. V. Kovacevic; I. P. Djukic. "FPGA based development platform for Implementation of Brushless DC Motor Control". In: *Telecommunications Forum (TELFOR), 2013 21st*. Lis. 2013.

[5] M. R. Hazari; E. Jahan ; M. E. Siraj ; M. T. I. Khan ; A. M. Saleque. "Design of a Brushless DC (BLDC) Motor Controller". In: *Electrical Engineering and Information & Communication Technology (ICEEICT), 2014 International Conference on*. Dub. 2014.

[6] A. Tashakori; M. Hassanudeen; M. Ektesabi. "FPGA Based Controller Drive of BLDC Motor Using Digital PWM Technique". In: *Power Electronics and Drive Systems (PEDS), 2015 IEEE 11th International Conference on*. Červ. 2015.

[7] *3-Phase BLDC Motor Control with Quadrature Encoder using 56F800/E*. Freescale Semiconductor. 2005. URL: `http://cache.nxp.com/files/dsp/doc/app_note/AN1961.pdf`.

[8] A. Simpkins; E. Todorov. "Position Estimation and Control of Compact BLDC Motors Based on Analog Linear Hall Effect Sensors". In: *2010 American Control Conference*. Červ. 2010. URL: `https://homes.cs.washington.edu/~todorov/papers/SimpkinsACC10.pdf`.

[9] S. B. Ozturk; H. A. Toliyat. "Direct Torque Control of Brushless DC Motor with Non-sinusoidal Back-EMF". In: *2007 IEEE International Electric Machines & Drives Conference*. 2007.

[10] P. Rusia; S. Bhongade. "Design and implementation of digital PID controller using FPGA for precision temperature control". In: *Power India International Conference (PIICON), 2014 6th IEEE*. 2014.

[11] G. F. Franklin; J. D. Powell; A. E. Naeini. *Feedback Control of Dynamics Systems*. 4. vyd. Prentice Hall, 2002.

[12] *AVR221: Discrete PID controller*. Atmel®. URL: `http://www.atmel.com/images/doc2558.pdf`.

[13] A. Packard. *ME 132: Feedback and Dynamic Systems, Saturation and Antiwindup Strategies*. UC Berkeley. 2005. URL: `https://jagger.berkeley.edu/~pack/me132/Section15.pdf`.

[14] J. W. Choi; S. C. Lee. "Antiwindup Strategy for PI-Type Speed Controller". In: *IEEE Transactions on Industrial Electronics* 56 (2009), s. 2039–2046.

[15] *PROFINET*. Siemens. URL: https://w3.siemens.com/mcms/water-industry/en/Documents/PROFINET.pdf.

[16] *PROFINET IO*. Real Time Automation. URL: http://www.rtaautomation.com/technologies/profinet-io/.

[17] *Softing Protocol IP for PROFINET*. Softing Industrial Automation GmbH.

[18] *Cyclone IV GX and E FPGA Features Resources*. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/pt/cyclone-iv-product-table.pdf.

[19] *DE2-115 User Manual*. Terasic.

[20] *Nios II Classic Processor Reference Guide*. Altera. URL: https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf.

[21] T. Ryčl. "Řízení pohonů na FPGA v síti Profinet". Dipl. ČVUT, 2015.

[22] J. Lewis. *VHDL-2008: Why It Matters*. SynthWorks VHDL Training. URL: https://s3.amazonaws.com/verificationhorizons.verificationacademy.com/volume-8_issue-3/articles/stream/vhdl-2008-why-it-matters_vh-v8-i3.pdf.

[23] David W. Bishop. *VHDL-2008 Support Library*. URL: http://www.eda.org/fphdl/.

[24] *Softing Protocol IP for PROFINET (zdrojový kód)*. Softing Industrial Automation GmbH.

[25] *Simple Device Application Interface (SDAI)*. SOFTING Industrial Automation GmbH.