VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

IMPLEMENTACE VYBRANÝCH ŘÍDICÍCH ALGORITMŮ PRO FPGA

IMPLEMENTATION OF SELECTED CONTROL ALGORITHMS FOR FPGA

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE — Bc. MARTIN BRADÁČ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ROBERT GREPL, Ph.D.
SUPERVISOR

BRNO 2011

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav mechaniky těles, mechatroniky a biomechaniky
Akademický rok: 2010/11

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Martin Bradáč

který/která studuje v **magisterském studijním programu**

obor: **Mechatronika (3906T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Implementace vybraných řidicích algoritmů pro FPGA

v anglickém jazyce:

Implementation of selected control algorithms for FPGA

Stručná charakteristika problematiky úkolu:

Programovatelná hradlová pole (FPGA) jsou alternativou ke standardnímu mikroprocesorovému řízení. Jejich hlavní výhodou je vysoká rychlost výpočtu. Prostředí NI LabView a hardware Compact RIO umožňují relativně snadné grafické programování FPGA.

Práce se bude zabývat implementací vybraných řídicích algoritmů na real-time procesoru a na FPGA a jejich porovnáním z hlediska funkčních možností (omezení FPGA), rychlosti výpočtu a náročnosti programování. Práce bude omezena na konkrétní hardware NI cRIO 9073 s vybranými I/O moduly.

Cíle diplomové práce:

1) Studium vlastností HW a SW prostředí (NI LabView, FPGA, cRIO) a testování jejich možností. Výstupem z této části práce bude soubor podrobně komentovaných testovacích VI.
2) Studium možností importu modelu v Simulinku do prostředí NI LabVIEW pro RT procesor i FPGA na cRIO. Výstupem bude opět soubor podrobně komentovaných testovacích VI.
3) Implementace několika vybraných typů řídicích algoritmů pro konkrétní laboratorní reálnou soustavu a to pro: a) RT procesor v LabVIEW; b) FPGA v LabVIEW; c) RT procesor v Simulinku; d) FPGA v Simulinku. Výstupem z této části práce bude porovnání uvedených čtyř přístupů k programování z hledisky kvality řízení, rychlosti výpočtu na cRIO a náročnosti implementace.

Seznam odborné literatury:

* Materiály ke školení NI Real-Time systems, NI FPGA.
* http://zone.ni.com/devzone/cda/tut/p/id/3555
* http://www.ni.com/fpga/

Vedoucí diplomové práce: Ing. Robert Grepl, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2010/11.

V Brně, dne 24.11.2010

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ DĚKANÁT

prof. Ing. Jindřich Petruška, CSc.
Ředitel ústavu

prof. RNDr. Miroslav Doupovec, CSc.
Děkan

Abstrakt

Real-time počítač cRIO-9073 firmy National Instruments spolu se softwarem LabVIEW je všestranný nástroj využitelný v mnoha oborech, jak při výzkumu, tak při ryze praktických úlohách.

Práce se zabývá studiem možností tohoto systému. Je cílena především na testování výkonnosti celého zařízení, možných omezení a ukázání jeho reálného využití při řízení procesů. Celý systém byl podroben testovaní s pomocí navržených aplikací pokrývajících široké spektrum využitelnosti. Systém cRIO umožňuje také využití dalších softwarových nástrojů při jeho programování. Tato práce zkoumá možnosti využití programu Matlab/Simulink od firmy Mathworks a také generování VHDL kódu s pomocí produktu firmy Xilinx. Cílem je porovnat náročnost programů na zdroje hradlového pole (FPGA) v cRIO a také množství výkonu real-time procesoru potřebného pro provoz daných aplikací.

Abstract

A real-time computer „cRIO-9073“ supplied by National Instruments, together with the software tool LabVIEW is a versatile system used in many fields, for both research and practical tasks.

This thesis deals with the possibilities of the system. It is aimed primarily at performance testing, identifying potential limitations and showing its practical use. The whole system has undergone testing through the use of programs covering the wide spectrum of utilisation. cRIO also allows the use of other software tools in its programming. This thesis explores the options of using Matlab/Simulink from MathWorks and the generation of VHDL code using a product of Xilinx. The aim is to compare the demands on the resources consumed, by the Field-programmable gate array (FPGA) in cRIO as well as the CPU load running the Real-time process.

Poděkování

Na tomto místě bych rád poděkoval především Ing. Robertu Greplovi, PhD. za jeho vedení při tvorbě mé diplomové práce, za jeho připomínky a návrhy a čas věnovaný mé práci. Dále kolektivu studentů laboratoře MechLab za vytvořené příjemné pracovní prostřední a možnost diskutovat aktuální problémy.
Děkuji také mé rodině za psychickou podporu v průběhu studia.

Čestné prohlášení

Čestně prohlašuji, že jsem tuto práci vypracoval samostatně s použitím uvedené literatury a pod vedením vedoucího DP.

Martin Bradáč, Brno, 2011

Obsah

1 Úvod

Americká firma National Instruments nabízí mnoho produktů, které spadají do oblastí zájmů týkajících se oboru Mechatronika. Tato firma je známá především svými systémy pro sběr dat zejména v průmyslových aplikacích. Dodává k tomuto účelu vlastní hardware, například měřící karty do počítače, externí víceúčelové zařízení s volitelnými moduly i specializovaný software LabVIEW. Tato zařízení se stávají stále více mnohostrannější a nabízí se jejich využití jako řídící jednotky pro real-time aplikace.

Tato práce se bude zabývat studiem možností a vlastností právě jednoho takovéhoto víceúčelového zařízení vhodného pro real-time aplikace nazvaného CompactRIO. Svým vybavením se řadí mezi středně výkonné zařízení schopné zvládat i složité úlohy řízení. Programování probíhá v prostředí LabVIEW, jedná se tedy o klasického představitele produktu firmy NI.

Konkurencí k těmto systémům mohou být produkty firmu MathWorks s jejich známým softwarem Matlab/Simulink. Tento software je hojně používaný a proto se zde nabízí otázka, jaké jsou možnosti použití tohoto SW nástroje při práci s produkty NI (především tedy v real-time aplikacích). Testování těchto možností bude prováděno na zařízení cRIO dostupném v laboratoři.

Pro zhodnocení možností daného HW a SW bude navrženo několik programů pro ovládání reálných aplikací, na kterých budou ukázány vhodné postupy pro vývoj, zjištěny hranice zařízení a zhodnocena užitečnost a náročnost tohoto přístupu k řízení.

2 Formulace problémů a cílů práce

Cílem této práce je prozkoumat možnosti HW a SW od společnosti National Instruments (NI). Zjistit, jaké jsou omezení, přednosti, výhody či nevýhody a v neposlední řadě také zhodnotit výkon celé soustavy. Problematiku lze rozdělit do následujících třech oblastí:

- Studium vlastností HW a SW v prostředí (NI LabView, FPGA, cRIO) a testování jejich možností. Výstupem z této části práce bude soubor podrobně komentovaných testovacích VI.
- Studium možností importu modelu v Simulinku do prostředí NI LabVIEW pro RT procesor i FPGA na cRIO. Výstupem bude opět soubor podrobně komentovaných testovacích VI.
- Implementace několika vybraných typů řídících algoritmů pro konkrétní laboratorní reálnou soustavu a to pro:

  I: RT procesor v LabVIEW

  II: FPGA v LabVIEW

  III: RT procesor v Simulinku

  IV: FPGA v Simulinku

Výstupem z této části práce bude porovnání uvedených čtyř přístupů k programování z hlediska kvality řízení, rychlosti výpočtu na cRIO a náročnosti implementace.

2.1 Volba hardwaru a softwaru

Pro potřeby laboratoře MechLab byl zakoupen real-time počítač s označením „cRIO-9073“. Jeho vlastnosti by se daly shrnout do následujícího přehledu:

- průmyslový real-time procesor běžící na taktu 266 MHz
- hradlové pole (FPGA) s 2 milióny systémových hradel
- FPGA čip řady Spartan 3 firmy Xilinx
- 64 MB operační paměti
- 128 MB paměti pro ukládání dat
- šasi až pro 8 modulů (napěťové, proudové I/O moduly, moduly pro připojení akcelerometrů, mikrofonů, teploměrů, relé, čítače pulzů, moduly pro komunikaci pomocí různých sběrnic a jiné)

- komunikace s Hostitelským PC pomocí ethernetového portu 10/100 Mb/s
- napájení stejnosměrným napětím v rozmezí od 19 do 30 V
- odolnost teplot v rozsahu od -20 do 50 °C

Pozn.: V laboratoři MechLab jsou v současné době k dispozici modul NI 9215 pro čtení čtyř analogových hodnot a modul NI 9401 pro čtení nebo zápis osmi digitálních hodnot.

Obr. 2.1: Systém cRIA s osmi moduly [2]

Firma NI nabízí další verze s různorodými obměnami a výkonem. Je možno pořídit zařízení obsahující real-time procesor o taktu 800 MHz, 512 MB operační paměti a 4 GB fyzické paměti. Jiná zařízení jsou například odolnější vůči teplotám (od -40 do 70 °C).

Jinou variantou jsou robustní průmyslové počítače označované PXI. Více o těchto systémech lze naleznout například na stránkách `http://www.ni.com/pxi/`. [1]

Obr. 2.2: Průmyslový počítač PXI [2]

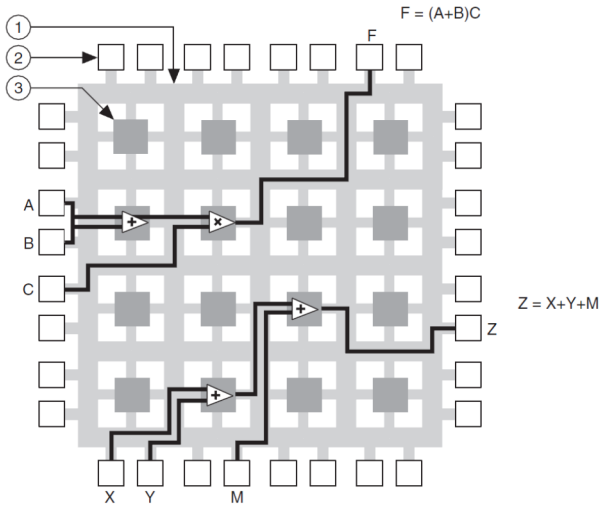
3 Rešerše

V této kapitole bude stručně nastíněna teorie týkající se hradlových polí, smyček v LabVIEW, počítání s čísly s pevnou desetinnou čárkou (neboli FXP aritmetika), realizace real-time aplikace a také možnosti přenosu simulinkovského modelu do produktů NI. Tyto dílčí záležitosti jsou důležité pro snadnější pochopení a proniknutí do problematiky zabývající se vývojem dané aplikace.

3.1 Představení FPGA technologie

3.1.1 FPGA technologie

Programovatelná hradlová pole (z anglického názvu Field Programmable Gate Arrays) jsou silikonové čipy s nespojenými hradly. Funkcionalita může být definována s pomocí nakonfigurování hradel. Samotné programování se skládá z procesu psaní kódu (používají se především dva programovací jazyky HDL a VDHL), poté je kód zkompilován do souboru obsahujícího informaci o tom, jak by měli být hradla spolu navzájem propojena, aby se vytvořil žádaný proces.



Obr. 3.1: Schéma principu FPGA [4]

V dnešní době čipy obsahují i další komponenty jako mohou být například násobičky, tabulky hodnot nebo paměti RAM - více bude popsáno v podkapitole 3.1.2. Hradlová pole jsou plně překonfigurovatelná. Nachází uplatnění především v nízké a střední úrovní ovládání elektroniky, kde by se z časových a finančních důvodů nevyplatilo použít specifické integrované obvody.

Výhodou FPGA je především jejich flexibilita. Narozdíl od mikroprocesorů je možné měnit jejich funkcionalitu. Vykonávání programu v FPGA může být efektivněji, protože vlastní program je tvořen propojením určitých hradel, takže nic nebrání tomu využít ostatní nevyužitá hradla pro další výpočty - tzv. paralelní zpracování.

Vlastní program je implementován přímo v hardwarové struktuře a je tedy vynechán nadřazený operační system. Program je tudíž velice spolehlivý a může dosahovat velké rychlosti výpočtu. [4]

3.1.2 Obsah hradlového pole

Každý FPGA čip je vyroben z konečného počtu programovatelných propojení určených pro implementaci programu přenastavením těchto propojení. Specifikace čipu se skládá z následujících parametrů [1]:

- počet konfigurovatelných logických bloků (například AND, OR, NAND)
- počet funkčních logických bloku (například násobení)
- velikost operační paměti (RAM)

Na té nejnižší úrovní je vše složeno z:

- Tzv. „Flip-flops“ - což je jednotka překlápějící svou hodnotu mezi 0 a 1, je-li potřeba. Je to binární posuvný registr. Hodnota se může měnit při každém tiku procesoru a je držena do následujícího tiku.
- Tzv. „LUTs = Look-up tables“ - tabulka hodnot (pravdivostní tabulka nul a jedniček). Díky této tabulce je ušetřeno mnoho Flip-flops, které by musely být použity.

Násobičky a DSP slices - jak už název napovídá, jedná se o funkční logické bloky. Násobičky jsou v čipu obsaženy z důvodu vysokého nároku na počet hradel na sestavení algoritmu násobení. Starší verze FPGA čipů obsahují 18-bitové násobičky, což znamená, že pokud je potřeba vynásobit 32-bitové číslo, je třeba využít více než jednu. Novější verze čipů obsahují tzv. DSP slice, což je násobička uchovávající si násobenou hodnotu pro další iteraci.

Paměť RAM - uživatelsky definovaná paměť v hradlovém poli, jež je vhodná pro uchování malého množství dat či předávání dat mezi paralelními smyčkami. Podle typu FPGA je možno vytvářet 16 nebo 32kb bloky RAM. Pro uchování dat je možné také použít pole s pomocí Flip-flops, nicméně to zabere spoustu hradel. Je dobré mít na paměti, že po odpojení napájení dojde ke ztratě hodnot z paměti. [5]

3.1.3 Tok dat v FPGA

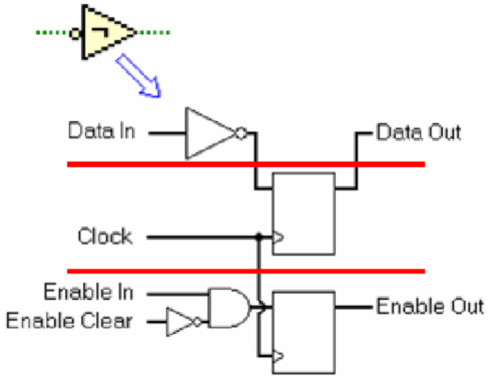
LabVIEW vykonává program postupně, jak „tečou“ data programem. Každý uzel se vykoná, pokud má přítomná data na všech svých vstupech. Když se uzel vykoná,

[1] Jsou zde uvedeny ty nejdůležitější, čip obsahuje více částí.

objeví se na výstupech data, která se předávají dále. Na obrázku číslo 3.2 je příklad takovéto operace.

LabVIEW přeloží kód pro FPGA a vzniknou tři sekce: logická, synchronizační a spouštěcí řetězec. Logická sekce odpovídá funkci z programu LabVIEW. Synchronizační sekce zaručuje vykonání programu pouze při vzestupné hraně hodin. Spouštěcí řetězec má za úkol zaručit spuštění jednotlivých bloků dle posloupnosti ve schématu v LabVIEW.

Kvůli tomuto spouštěcímu řetězci trvá vykonání každé funkce přinejmenším jeden tik procesoru. Při zpracování analogového vstupu může tato operace trvat stovky tiků. Celý program může běžet pouze tak rychle, jaký je celkový součet jednotlivých operací při dané cestě programem. Výhodou FPGA je použití paralelního programování, kdy nic nebrání tomu, aby se při daném tiku procesoru vykonalo několik na sobě nezávislých operací. Viz schéma 3.1. [4]



Obr. 3.2: Princip toku dat v FPGA [4]

3.1.4 Porovnání čipů FPGA

Testované cRIO - 9073 obsahuje FPGA čip Xilinx **Spartan-3 2000**. Specifikace dle výrobce je v tabulce 3.1, kde jsou uvedeny také další modely pro srovnání.

Tab. 3.1: Přehled FPGA čipů

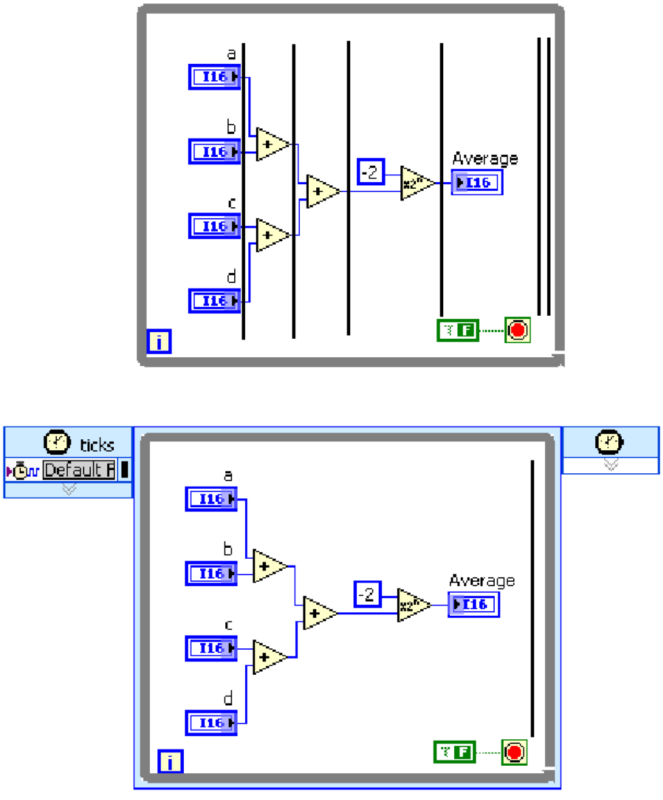
Název	Počet hradel	Flip-flops	LUTs	Násobiček	Celková velikost RAM [kbits]	Velikost paměti [kbits]
Spartan-3 2000	2 M	40960	40960	40	720	16
Virtex-II 3000	3 M	28672	28672	96	720	16
Virtex-5 LX110	—	69120	69120	64	4608	36

Pozn.: V dnešní době je model Spartan u šesté verze a Virtex u sedmé s řádově vyššími výkony. Nicméně jejich implementace do produktů NI nějakou dobu trvá.

3.2 Použití časových smyček

Kód obsažený v Timed Loop je více optimalizovaný, zabírá méně paměťového místa v FPGA a vykonává se rychleji než stejný kód obsažený ve While Loop. Timed Loop odstraňuje spouštěcí řetězec ze smyčky, aby ušetřil místo, a především také proto, aby se mohl celý proces vykonat při jediném tiku procesoru. Porovnání vykonání smyčky při použití Timed Loop a While Loop je na obrázku 3.3.

Protože se celý program vykonává při jediném tiku, musí být délka periody dostatečně dlouhá, aby se stihly všechny operace vykonat při tomto jediném tiku. Proto nelze použít některé funkce a struktury jako například analogový vstup/výstup, přerušení atd., které jsou příliš pomalé pro vykonání v jediném tiku procesoru. [4]



Obr. 3.3: Použití smyček [4]

3.3 Fixed-point aritmetika

Čísla s pevnou desetinnou čárkou (= „FXP“) tvoří soubor racionálních čísel reprezentovaný pomocí binárních číslic (tedy bitů). Narozdíl od čísel s plovoucí čárkou, kde celková velikost čísla v bitech může být pohyblivá, FXP čísla mají vždy pevně stanovenou velikost v bitech. Tato čísla jsou vyžadována při práci s hardwarem (např. FPGA), který nemá zavedené počítání s plovoucími čísly. Je možno nastavit libovolný rozsah a přesnost čísla. Velikost může být v rozmezí 1 až 64 bitů, přičemž číslo může být znaménkové nebo neznaménkové.

Rozsah FXP čísla určuje minimální hodnota, maximální hodnota a delta (nejmenší změna čísla). Tyto tři hodnoty jsou určeny použitím znaménka, délkou čísla a délkou hodnoty před desetinou čárkou. [4]

Pro neznaménkové číslo označené $< +X, Y >$, kde X je délka čísla a Y počet čísel před desetinou čárkou, lze najít maximální hodnotu dle vzorce:

$$Max = 2^Y - \frac{1}{2^{X-Y}} \tag{3.1}$$

hodnota delta:

$$\Delta = \frac{1}{2^{X-Y}} \tag{3.2}$$

U znaménkového čísla zůstává výpočet delty stejný, ale dochází ke změně výpočtu maximální hodnoty:

$$Max = 2^{Y-1} - \frac{1}{2^{X-Y}} \tag{3.3}$$

Operace s FXP čísly - při matematických operacích dochází ke změně rozměru čísla. Tuto skutečnost je nutno zohlednit do návrhu programy, aby nedošlo k nechtěné ztrátě přesnosti v důsledku prováděných matematických operací. Změnu rozměru čísla ukazuje následující přehled.

- sčítání: $< \pm A, B >$ výsledek: $< \pm A + 1, B + 1 >$
- odčítání: $< +A, B >$ výsledek: $< \pm A + 1, B + 1 >$
- násobení: $< \pm A, B >$ a $< +C, D >$ výsledek: $< \pm A + C, B + D >$
- dělení: $< \pm A, B >$ a $< \pm C, D >$ výsledek: $< \pm A + C + 1, B + C - D + 1 >$
- dělení: $< +A, B >$ a $< +C, D >$ výsledek: $< +A + C, B + C - D >$

3.4 Základy programovacích jazyků VHDL a HDL

VHDL a HDL jsou programovací jazyky sloužící pro popis hardwaru. Používají se pro návrh a simulaci digitálních integrovaných obvodů. Jejich výhodnou je schopnost popsat požadované chování a otestovat ho před kompilací pro real-time zařízení (FPGA čip). Další výhodnou je možnost souběžného chodu programu, na rozdíl od například jazyka C, který běží sekvenčně (jedna instrukce v jednom časovém okamžiku). Více-účelnost je další z jeho předností, které umožňuje využívat jednou vytvořené bloky programu ve více projektech (včetně úpravy daných parametrů).

Základní konstrukce

- **entity** - definuje rozhraní (pouze vstupy a výstupy, ne funkci)
- **architecture** - určuje chování entit (má dvě části - deklarační a příkazovou)

Pro jednu entitu může existovat více architektur (implementací). [6]

3.5 Způsob porovnání dílčích částí programu

Celou real-time aplikaci lze rozdělit do tří částí a to podle místa, kde se daná část kódu nachází. Tímto způsobem bude také aplikace rozdělena, tedy na část FPGA, část Real-time (RT) a část v Hostitelském počítači.

FPGA – tato nejnižší část programu obsahuje především interface pro ovládání digitálních a analogových vstupů/výstupů. Nicméně je zde také možnost provádět určité výpočty, pokud je možné je sestavit s pomocí dostupných (značně omezených) bloků LabVIEW. Tato skutečnost je zapříčiněná faktem, že námi vytvořený kód je převeden a optimalizován do jazyku VHDL a nahrán do hradlového pole FPGA. Jeden ze způsobů, jak porovnávat rychlost bloků je pomocí množství tiků potřebných pro výpočet dané smyčky. Tato metoda je nicméně vhodná pro použití na While Loop, protože Timed Loop proběhne vždy v jednom jediném tiku (tato skutečnost vyplývá z podstaty způsobu běhu programu v FPGA - viz 3.2). Další možností je porovnávat množství potřebných zdrojů hradlového pole pro daný program, což bude rozvedeno dále.

RT – neboli Real-Time počítač obsažený v cRIO. Do této programové části jsou přenášeny data z FPGA a jsou zde dále zpracovávány. Je zde o poznání bohatší nabídka bloků LabVIEW než v FPGA. Jedna skupina bloků nese název „RT execution trace toolkit“ a jejím hlavním smyslem je vytvoření jakéhosi monitoru jednotlivých procesů při vykonávaní programu. Z výsledného schématu je možné vyčíst dobu trvání jednotlivých částí. Obsahuje také bloček pro umístění uživatelských značek

v programu, které budou viditelně zobrazeny ve výsledném logu. Porovnávání rychlosti smyček v RT bude tedy řešeno pomocí času. Dále jsou k dispozici nástroje pro měření zatížení CPU RT procesoru a zaplnění jeho paměti - bude také předmětem práce.

Hostitelské PC – tato část programu je vykonávána na uživatelském počítači, kde rychlost programu závisí na výkonu daného PC. Zkoumání doby trvání jednotlivých smyček tedy není tak podstatné a zajímavé pro naše účely real time řízení procesů.

Statistické metody - slouží ke statistickému hodnocení kvality regulace zvoleného algoritmu. [8]

- střední kvadratická odchylka - „MSE“

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\varphi_{ref_i} - \varphi_i)^2 \tag{3.4}$$

- střední absolutní chyba - „MAE“

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\varphi_{ref_i} - \varphi_i| \tag{3.5}$$

- maximální chyba odchylky - „MaxError“

$$MaxError = max(|\varphi_{ref_i} - \varphi_i|) \tag{3.6}$$

3.6 Přenos dat v RT

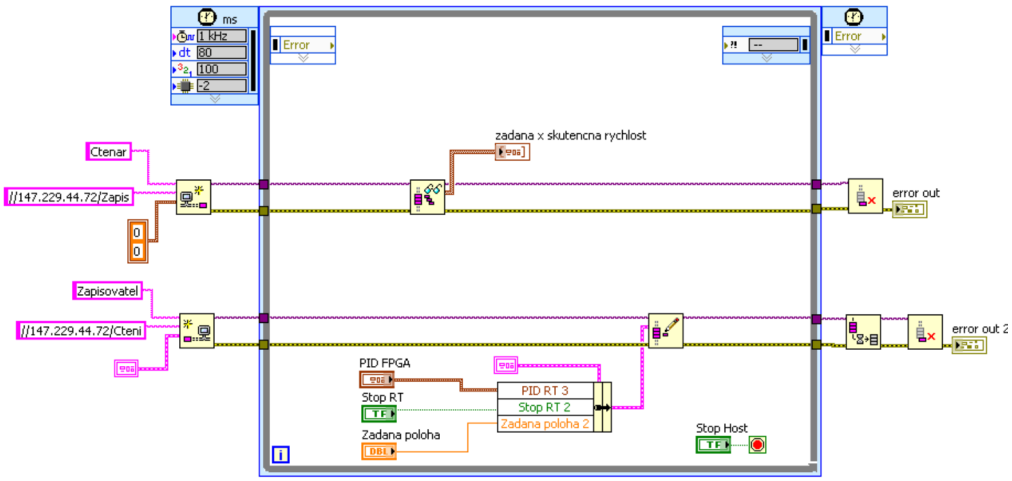
Samotný přenos dat je důležitou částí programu, protože pokud bude jeho použití neuvážené, mohou se ztrácet data, což ovlivní celou aplikaci. Jako důležitý fakt je skutečnost, že aplikace v FPGA a RT běží asynchronně. Po synchronizaci je třebné použít v RT informaci o časování z FPGA, což vyžaduje přidání synchronizačního kódu do naší aplikace. Úroveň dané synchronizace záleží na aplikaci, může být například vyžadována synchronizace, pokud časování FPGA musí ovládat časování RT aplikace. Dalším příkladem může být získávání dat z okolí. Pokud jsou data získávána s určitou frekvencí, ale rychlost přenosu je nižší, dochází ke ztrátě dat, což je nevhodné, pokud máme zájem o všechna data. V případech, kdy nám nevadí, že přicházíme o některá data, nemusíme se tímto problémem zabývat. Jedná se především o pomalejší řídící aplikace, kde rychlost smyčky v FPGA je vyšší než rychlost smyčky v RT. Konkrétně například smyčka produkující hodnotu natočení z enkodéru běží rychleji v FPGA než řídící smyčka v RT, kde tato smyčka si bere

vždy určitou hodnotu natočení v daném tiku procesoru. Není potřeba vědět natočení v každé nanosekundě, protože uspokojivého řízení lze dosáhnout i například při časování na milisekundu.

Číst data po jednotlivých hodnotách je nevhodné, protože se snadno mohou ztrácet - RT procesor nemusí být připraven číst jednotlivá data v každém okamžiku, kdy přicházejí na jeho vstupy. Nejvhodnějším řešením je použití tzn. bufferu, což je jakýsi zásobník dat v paměti HW, který je schopen ukládat různá data. Tato data lze pak přenášet s použitím tzv. Direct Memory Access (DMA) neboli přímého přístupu do paměti. Je užito tzv. FIFO procesu, kdy jsou přenášeny balíky dat z FPGA do paměti nadřazeného systému (v našem případě RT HW). Toto nabízí značnou výhodu oproti čtení indikátorů a zapisování do ovládacích prvků pomocí RT procesoru v FPGA. Tento proces běží samostatně, takže ve stejné době, kdy jsou data přenášena z FPGA, může RT vykonávat různé algoritmy. Synchronizace může být prováděna pomocí přerušení přicházející z FPGA a nebo pomocí další smyčky, která kontroluje status FPGA, zda již došlo k určitým podmínkám, které mají spustit přenos dat.

Metody přenosu dat do PC jsou následující:

- TCP/IP (klasický způsob přenosu dat sítí).
- Network stream (moderní způsob transferu dat, podporující přenos libovolných datových typů aniž by je bylo nutno sloučit a poté rozdělit jako u TCP, umožňují automatickou obnovu přenosu při výpadku sítě).



Obr. 3.4: Přenos dat z RT do Hostitelského PC s pomocí Network Stream

3.7 Vhodné zdroje informací

Jako základní zdroje informací pro práci s HW a SW od NI lze považovat následující:

- Internet, především stránky `www.ni.com` , které obsahují obrovské množství různých článků, návodů, rad, řešení problémů, řešených příkladů, uživatelské fórum a další ...
- Hotové programy dodávané s programem LabVIEW (neboli „Examples“). V této databázi lze hledat s pomocí klíčových slov a jsou k nalezení jak programy čistě pro LabVIEW v PC, tak také programy určené pro různé multifunkční karty a Real-time zařízení.
- Telefonická nebo emailová podpora firmy NI v případě problémů

3.8 SW nástroje pro interface mezi LabVIEW a Simulinkem

3.8.1 Simulation Interface toolkit

Tento jeden z mnoha toolkitů dostupných pro LabVIEW složí pro provázání modelu vytvořeného v Simulinku (potažmo z Real-time Workshopu) v systému LabVIEW. Umožňuje především samotný vývoj, tzv. prototyping a následné testovaní daného modelu v LabVIEW. Tento toolkit je možno použít pro vytvoření uživatelského prostředí v LabVIEW pro model vytvořený v Simulinku a spustit simulaci na real-time zařízení.

Výhody:

- relativní jednoduchost a pochopitelnost
- použití stejného prostředí jako v LabVIEW
- možnost vidět vygenerovaný kód, který je následně nahrán do RT (do kterého by se ale nemělo zasahovat)
- cena (pokud je třeba toto brát do úvahy)
- jednodušší program v FPGA (což ale nemusí být zase úplně výhodné)

Nevýhody:

- proces převodu a následného spuštění aplikace se skládá z mnoha kroků, které je třeba stále opakovat a některé nemusí být zrovna logické

3.8.2 VeriStand

Tento samostatný program je určen pro ovládání simulací spuštěných na různých real-time zařízeních. Pro tyto účely jsou vhodnější systémy PXI, které nabízejí vysoké výkony (oproti cRIO). Je možno ovládat jak analogové, digitální porty, tak různé komunikační linky, ukládat naměřená data, generovat budící signály, přidávat různé jednoduché algoritmy, nastavovat alarmy a další. Je možno importovat algoritmy od dalších různých výrobců a také je potom nahrávat do zařízení, která nebyla vyrobena NI.

Častým příkladem z praxe je použití při testování tzv. Hardware-in-the-loop (HIL). Při tomto testu jde o to, aby model běžel deterministicky a bylo tedy dosaženo vždy přesné časové odezvy simulovaných komponentů (například různých senzorů) se kterými se dále pracuje v simulačních smyčkách řídících algoritmů.

Vývoj aplikace se skládá z kompletního nastavení modelu, kde se specifikují vstupy a výstupy, cílové zařízení, importují se modely pro následné spuštění v RT a vše další potřebné pro nahrání a spuštění modelu v cílovém zařízení. Druhou vrstvou programu je uživatelské prostředí, které zprostředkovává vizualizaci výstupů a vstupů z modelu. Uživatel zde může snadno ovládat model s pomocí různých tlačítek, knoflíků, přepínačů atd. a vidět, co se děje na různých grafech a displejích. Snadno se zde nastaví, která data se mají ukládat pro pozdější zpracování.

Obr. 3.5: Logo VeriStand [2]

Výhody:

- celistvý softwarový nástroj pro správu real-time aplikace (netřeba vůbec LabVIEW)
- jednoduchá editovatelnost kódu (hlavě správa)
- možnost použití na různých HW (i jiných firem než NI)
- možnost importu modelů vytvořených v Simulinku, SimulationX, MapleSim, GT-POWER a jiných
- použitím FPGA je možné doprogramovat vlastní I/O HW (možnost vytvoření procesů běžících rychlostí až 25 ns a nespotřebovávající kapacitu RT aplikace)
- bohatost nástrojů pro interface mezi RT a uživatelem
- spolupráce s .NET API jazykem

Nevýhody:

- složitější na celkové pochopení, hodí se pro náročnější aplikace

3.9 Implementace VHDL do FPGA čipu cRIO

Systém NI cRIO obsahuje FPGA čip vyrobený firmou Xilinx. Pro kompilaci programů z LabVIEW do FPGA (vytvoření tzv. bitfile) se volá kompilátor firmy Xilinx, což je jiná aplikace. Z tohoto důvodu neexistuje (prozatím) žádný jednoduchý automatický převod z programů třetích výrobců do čipů FPGA (od Xilinx) umístěných v produktech NI cRIO. Nicméně je to přesto řešitelné. Knihovna v LabVIEW určená pro vytváření programů v FPGA obsahuje mimo jiné i blok pro tzv. Component level IP, což je nástroj pro vkládání VHDL kódu do LabVIEW, který nám vytvoří v projektu vstupy a výstupy k tomuto vloženému programu. Zmíněný kód lze získat dvěma způsoby a to buď stažením z internetu, kde je možno nalézt množství těchto hotových programů a pouze vložit do našeho projektu nebo si ho vytvořit sám. Pokud se vydáme touto cestou, je dobré vědět základní charakteristiky programování v tomto jazyku, není ovšem nutné umět přímo programovat, protože pokud dokážeme vygenerovat funkční kód, stačí ho jen pak vložit na správná místa. Základní vlastnosti o jazyku jsou uvedeny v kapitole 3.4. Tímto způsobem lze vytvořit aplikaci, která může běžet na FPGA. Zbývá tedy vytvořit zmiňovaný kód, který lze vygenerovat například i ze Simulinku. Jako dobrý zdroj informací lze využít diplomovou práci věnovanou této problematice [18].

3.9.1 Generování VHDL/HDL kódu

- Generování VHDL/HDL kódu s pomocí Simulink HDL Coder - Simulink sám o sobě obsahuje nástroj pro automatické vytváření VHDL/HDL kódu z modelu. Jde třeba dbát na použití jen těch bloků, které jsou podporovány pro konverzi.
- Použití Xilinx System Generator - firma Xilinx dodává nástroj, který přidá speciální knihovnu do Simulinku, která je optimalizovaná pro převod do VHDL/HDL kódu pomocí Xilinx generátoru.
- Napsání vlastního VHDL/HLD kódu - je zde vždy možnost si napsat celý kód sám, nicméně to vyžaduje vyšší programátorské dovednosti. Poté ale máme kód, který je přehledný a kterému plně rozumíme.

4 Jednoduché testovací úlohy

V této kapitole bude vytvořeno několik testovacích programů za účelem otestování výkonu cRIO. Jednotlivé programy mají různou výpočetní náročnost, jejich popis je uveden v přehledové tabulce. Kapitola je rozdělena na dvě části a to na testovaní vlastností FPGA a RT.

4.1 Testování programů pro FPGA čip

Pro účely testování byly vytvořeny programy obsahující různé dostupné bloky pokrývající široké spektrum použití. Přehled modelů s jejich popisem bude uveden v následující tabulce. Všechny programy obsahují pouze jedinou časovanou smyčku.

Tab. 4.1: Přehled použitých programů pro FPGA

název modelu	popis
FFT	FPGA obsahuje generátor sinusového signálu, který se ukládá do FIFO paměti. Z paměti jsou pak tyto hodnoty čteny a vstupují do bloku pro FFT analýzu (v časované smyčce), výsledné zpracování je ukládáno do FIFO paměti a přenášeno do RT, kde je vykreslováno v grafu.
FILTR	FPGA obsahuje generátor sinusového signálu a generátor šumu. Tyto signály vstupují do filtru dolní propust a výsledek je ukládán do FIFO paměti, která je čtena v RT a vykreslována.
8xFILTR	Totožný jako program FILTR s tím rozdílem, že v tomto programu je 8 bloků pro filtraci signálu v FPGA.
ČÍTAČ	FPGA obsahuje časovanou smyčku, která počítá vzestupné hrany a dále smyčku, která měří periodu. Je zde také jednoduchý PWM generátor pomocí časovače.
ENKODÉR	FPGA obsahuje zpracování 16-bitového kvadratického čítače, kdy se zpracovává i indexovací kanál.
NORMALIZACE VEKTORU	FPGA obsahuje jednoduchou matematiku pro normalizaci 3D vektoru, ukládání do FIFO paměti.
GENERÁTOR SIN	FPGA obsahuje jednoduchý generátor dvou sinusových signálů.
HDL BLOK	FPGA obsahuje blok pro vkládání VHDL kódu, kde je naprogramováno sčítání dvou 32-bitových čísel (integer).
PRÁZDNÝ MODEL	FPGA obsahuje pouze jednu prázdnou časovou smyčku a jednu While smyčku - určeno pro porovnání.

Programy použité pro testování byly čerpány ze souboru vypracovaných příkladů dodávaných společně s LabVIEW. Následující tabulka popisuje zužitkování bloků v FPGA. Vysvětlení bloků popisovaných v tabulce lze nalézt v kapitole 3.1.2.

Tab. 4.2: Přehled náročnosti programů na prostředky FPGA čipu

Název modelu	Slices	%	Flip-flops	%	LUTs	%	Bloků RAM	%
FFT	3058	14,9	2322	5,7	4192	10,2	16	40
FILTR	1678	8,2	1761	4,3	2021	4,9	5	12,5
8xFILTR	12103	59,1	6978	17,0	20083	49	32	80
ČÍTAČ	827	4,0	943	2,3	1072	2,6	0	0,0
ENKODÉR	525	2,6	512	1,2	665	1,6	0	0,0
NORM. VEKTORU	1381	6,7	1320	3,2	1784	4,4	2	5,0
GENERÁTOR SIN	1441	7,0	1522	3,7	1492	3,6	2	5,0
HDL BLOK	568	2,8	563	1,4	656	1,6	0	0,0
PRÁZDNÝ MODEL	377	1,8	378	0,9	520	1,3	0	0,0

Po kompilaci také dostaneme informaci o časové náročnosti daného programu - vidíme reálnou hodnotu frekvence, na které daná smyčka poběží. Dalším údajem je hodnota frekvence smyčky, které by šlo maximálně dosáhnout. Pro výše uvedené programy jsou hodnoty v následující tabulce:

Tab. 4.3: Časování programů v FPGA

Název modelu	Čas (MHz)	Max. možný čas (MHz)
FFT	40,02	40,66
FILTR	40,02	41,62
8xFILTR	40,02	38,66
ČÍTAČ	40,02	76,81
ENKODÉR	40,02	90,64
NORM. VEKTORU	40,02	70,73
GENERÁTOR SIN	40,02	70,64
HDL BLOK	40,02	112,98
PRÁZDNÝ MODEL	40,02	142,45

Průměrná doba kompilace programů je 10 minut (nejnáročnější je kompilace programu FFT, který trval 15 minut), nicméně ale také záleží na daném počítači, na kterém je kompilace prováděna (v tomto případně procesor Intel Core2 Duo E8400 3GHz, 3 GB RAM).

4.2 Testování programů pro Real-time procesor

Pro potřeby testování RT procesoru byly sestaveny programy, na kterých mají být ukázány možnosti a schopnosti HW a SW. Část programů byla převzata z hotových ukázkových programů, které jsou součástí instalace LabVIEW. Jiné byly ručně sestaveny. Jednotlivé programy obsahují dvě smyčky a to jednu časovou, ve které je umístěna celá zkoumaná aplikace a jednu While smyčku s dlouhou periodou, která zjišťuje statistická data.

Tab. 4.4: Přehled použitých programů pro RT

Název modelu	Popis
CPU	Blok diagram obsahuje pouze jednu While smyčku, ve které jsou umístěny bloky pro měření zatížení CPU a množství paměti použité RT. Tato smyčka slouží jako referenční pro ostatní měření.
FFT	V této smyčce je blok pro generování periodického signálu a blok pro FFT analýzu, který zjišťuje amplitudu a fázi signálu.
DYN. FFT	Dynamicky počítaná FFT analýza. Za generovaným signálem je filtr. Tento model je zde pro porovnání s předchozím, kdy při použití vhodné filtrace lze docílit mnohem rychlejšího výpočtu.
NYQUIST	Smyčka počítá frekvenční odezvu a vykresluje Nyquistovu charakteristiku.
FIR	Generovaný signál s přidaným šumem vstupuje do filtru FIR (finite impulse response) a je zobrazován v grafu.
IIR	Generovaný signál s přidaným šumem vstupuje do filtru IIR (infinite impulse response) a je zobrazován v grafu.
DC-RMS	Měření stejnosměrné a efektivní hodnoty přiváděného periodického signálu (využití při měření střídavého proudu). V programu se generují signály a přivádějí se do bloku pro výpočet DC-RMS hodnot.
THD	Výpočet činitele harmonického zkreslení (THD) - míra zkreslení signálu. Generovaný signál spolu se šumem vstupuje do bloku pro analýzu.
TÓN	Extrakce informace o tónu z komplexního vstupního signálu. Generovaný signál spolu se šumem vstupuje do bloku pro detekci frekvence a amplitudy.
DATA	Z FPGA jsou čteny 4 analogové a 4 digitální porty a pomocí DMA přenosu posílány do RT, kde jsou zobrazovány.
STREAM	Pomocí Network stream je posíláno jedno 64-bitové číslo (double) do PC a přijímáno jiné číslo.
SÍŤ	Pomocí sdílené síťové proměnné je posíláno jedno 64-bitové číslo (double) do PC a přijímáno jiné číslo.
PRÁZDNÝ	Model obsahuje pouze jedinou While smyčku s nástroji pro měření výkonu CPU a zaplnění paměti.

Smyčka obsahující pouze nástroje pro měření výkonu CPU a zaplnění paměti

(ekvivalentní s programem PRÁZDNÝ), který běží s periodou 1 s a zatěžuje celkově procesor 2,3 %. Tuto hodnotu lze považovat za referenční. Nicméně je třeba mít na paměti, že v RT je obsaženo spoustu aplikací, které mohou výsledek ovlivňovat, ale pro reálnou představu o náročnosti jednotlivých modelů je to dostačující. Před každým spuštěním cRIO byl proveden restart.

Tab. 4.5: Zatížení CPU [%] v RT pro dvě periody smyčky

Název modelu	Zaplnění paměti	20 ms		5 ms		Minimální perioda smyčky
		Timed loop	Celkem	Timed loop	Celkem	
FFT	45 MB	35 %	53 %	– %	– %	8 ms
DYN. FFT	49 MB	34 %	86 %	35 %	87 %	1 ms
NYQUIST	46 MB	37 %	64 %	45 %	78 %	1 ms
FIR	45 MB	15 %	54 %	60 %	95 %	3 ms
IIR	45 MB	16 %	54 %	61 %	95 %	3 ms
DC-RMS	45 MB	30 %	54 %	60 %	90 %	1 ms
THD	45 MB	70 %	85 %	– %	– %	15 ms
TÓN	45 MB	70 %	89 %	94 %	99 %	1 ms
DATA	42 MB	2 %	9 %	5 %	13 %	1 ms
STREAM	40 MB	7 %	26 %	10 %	34 %	1 ms
SÍŤ	41 MB	2 %	47 %	6 %	59 %	1 ms
PRÁZDNÝ	39 MB	0,8%	8 %	3 %	10 %	0,1 ms

4.3 Zhodnocení výkonu cRIO

Testované programy ukázaly, že FPGA čip disponuje stále s dostatkem volných prostředků v čipu, což umožňuje vytvářet složité programy. Z tabulky 4.2 je vidět, že použití datového typu boolean má nejnižší nároky na zdroje. Z důvodu optimalizačního procesu při kompilaci neplatí, že použití dvou, například FFT programů, bude spotřebovávat dvojnásobek zdrojů, nýbrž bude spotřebovávat méně zdrojů.

Z výsledků získaných z testování programů pro RT je vidět, že obslužné programy přítomny v paměti RT zabírají mnohem více místa než samostatný vyvíjený program. Z přehledu zatížení CPU je vidět, že při periodě smyčky 20 ms je rozdíl mezi testovými programy velký, při periodě 5 ms je rozdíl znatelně nižší, což je způsobeno vyššími nároky obslužných aplikací programu při této vysoké periodě.

U některých programů (například DYN. FFT) nedochází ke změně zatížení CPU při změně periody smyčky. Příčinou je použití nejjednodušších bloků při návrhu programu.

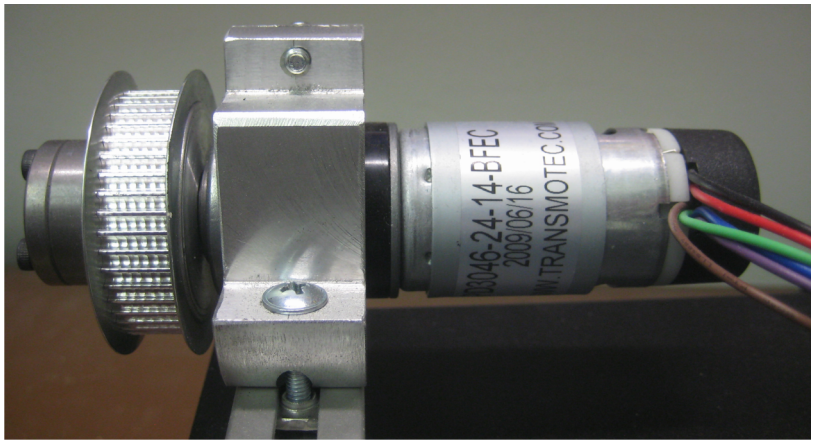
5 Testování reálných modelů

V následující kapitole budou vytvořeny programy pro dva testovací systémy a to řízení rychlosti DC motoru a řízení otevření škrtící klapky. Zkoumané vlastnosti budou shodné jako u jednoduchých testovacích modelů v kapitole 4.

5.1 Použité testovací modely

5.1.1 DC motor

Pro testování algoritmů bylo zvoleno využití již hotového výukového modelu dvou DC motorů s optickými enkodéry s navzájem spojenými řemenicemi pomocí ozubeného řemene. Tento model slouží pro prvotní reálné seznámení s PID regulací.



Obr. 5.1: DC motor - reálný model

Motor je vyroben firmou TRANSMOTEC s označením PD3046–24–14–BFEC.

Specifikace motoru jsou následující:

Tab. 5.1: Parametry DC motoru

parametr	hodnota
napájení	24 V
převodový poměr	14:1
jmenovité otáčky pohonu	445 ot/min

Specifikace enkodéru jsou následující:

Tab. 5.2: Parametr enkodéru

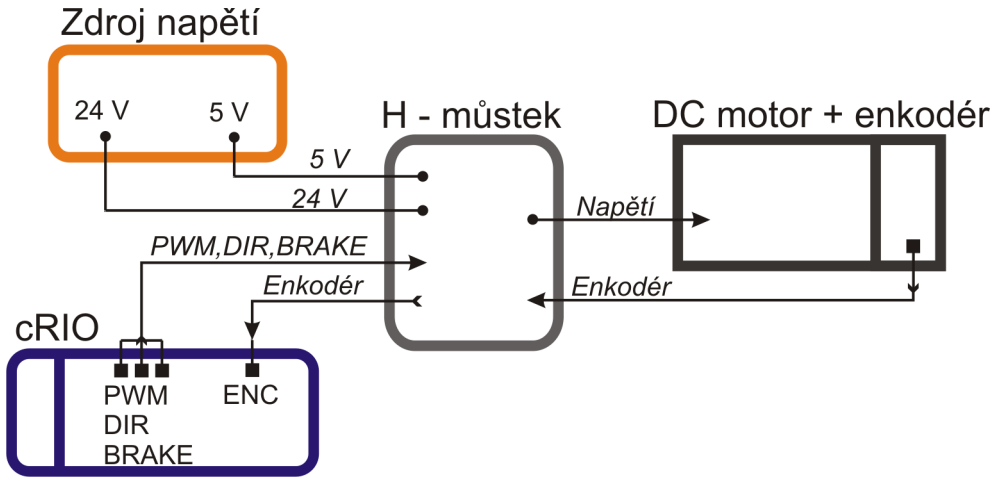
parametr	hodnota
magnetických pólů	14

Při měření motoru byli zjištěny následující parametry:

Tab. 5.3: Změřené parametry DC motoru

parametr	hodnota
R	6,7 Ω
L	4,3 mH

Následující obrázek ukazuje schématické zapojení DC motoru:



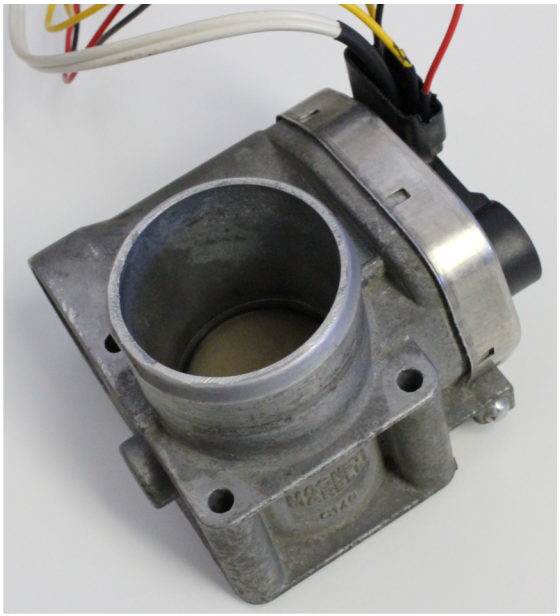
Obr. 5.2: Schéma zapojení DC motoru

5.1.2 Škrtící klapka z automobilu

Škrtící klapka je součástí tzv. konceptu řízení po drátě („x-by-wire“), kdy z pedálu řízení vede pouze kabel nikoliv táhlo, kde elektrický signál ovládá otevírání klapky a tím reguluje množství vzduchu přiváděného do motoru. Klapka je elektromechanickým systémem s DC motorem a převodovkou pro ovládání ventilu, torzní pružinou pro zavření klapky v případě nutnosti a potenciometru pro zjištění aktuální polohy natočení ventilu. Z důvodu celkové ceny za produkt je kvalita tohoto dílu poměrně nízká, tudíž vykazuje značnou nelinearitu způsobenou použitím pružiny a

přítomností suchého třením. Obvyklé použití klasického PID nemusí být dostačující a je záhodno využít například kompenzace tření. Tímto „spravíme“ nižší technickou kvalitu a dosáhneme přesnějšího řízení, kterého bychom dosáhli také použitím kvalitnější = dražších dílů. Tato práce si neklade za cíl nalézt dokonalé řízení, větší pozornost bude naopak věnována porovnání různých metod řízení v závislosti na rychlosti smyčky.

Testovaná klapka má označení Magneti marelli C146 a je používaná v autech značky FIAT.



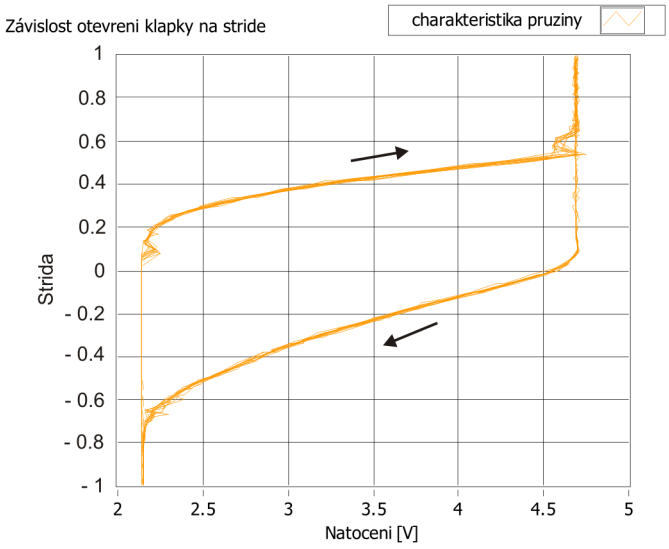
Obr. 5.3: Reálná součástka škrtící klapka z automobilu

Tab. 5.4: Parametry škrtící klapky

parametr	hodnota
Napájecí napětí	24 V
R	3,6 Ω
L	1,2 mH

Nelineární chování pružiny je možné pozorovat ze závislosti otevření klapky na dané střídě viz graf 5.4. Natočení klapky je zde vyjádřeno za pomocí napětí na potenciometru (2,2 V - zavřeno, 4,7 V - otevřeno). Na střídu byl přiveden trojúhelníkový signál.

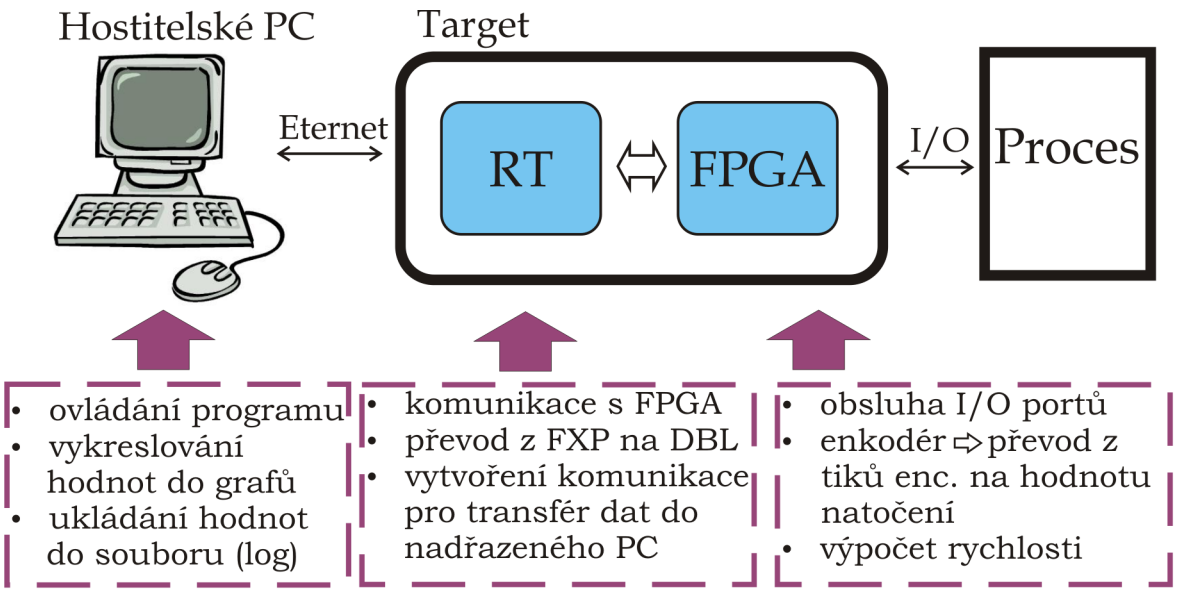
Více informací o škrtící klapce v automobilu je možno nalézt zde [7].



Obr. 5.4: Charakteristika pružiny el. klapky

5.2 Struktura programu pro systém cRIO

Celý program je komplexní záležitost, která se skládá z několika částí. Tři hlavní části programu jsou FPGA, RT a Hostitelské PC (jak již bylo zmíněno v kapitole 3.5). Při vytváření programu je nutné vědět, kterou část programu umístit do jaké vrstvy. Pro bližší představu o struktuře programu byl vytvořen následující diagram:



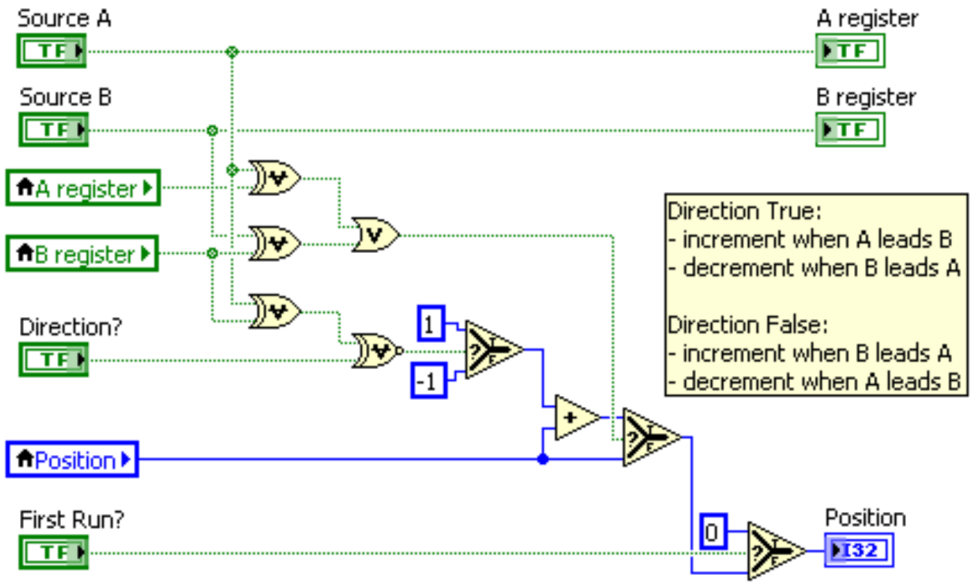
Obr. 5.5: Struktura real-time aplikace

5.3 Zpracování dílčích částí programu

V této části budou podrobněji rozepsány části programu důležité pro běh celé aplikace. Jedná se především o kód obsažený v FPGA.

5.3.1 Enkodér a rychlost otáčení

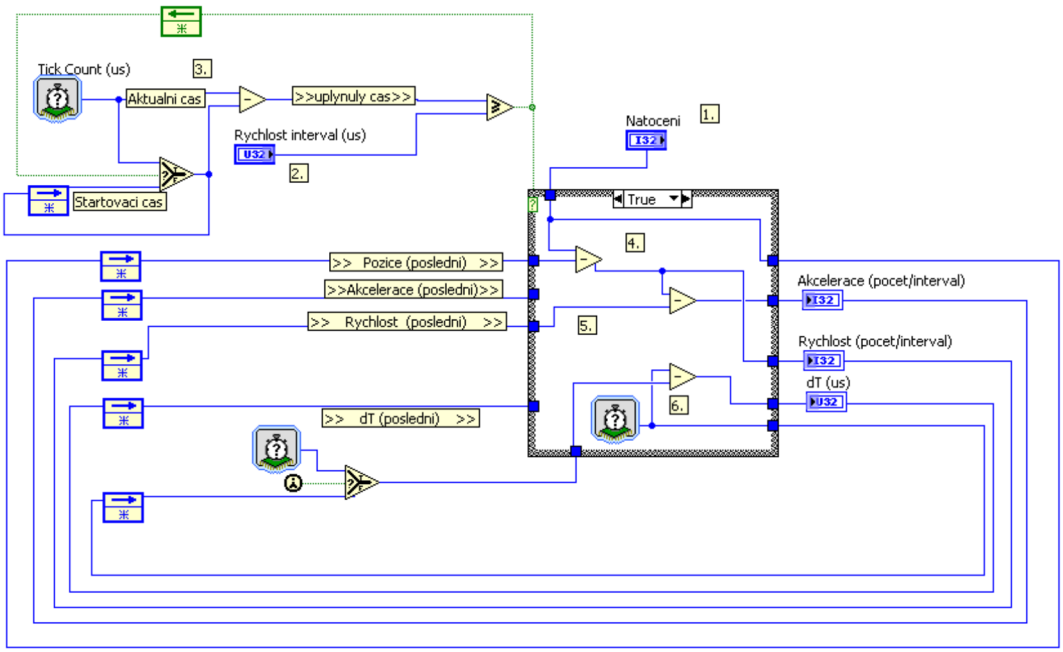
Inspirací pro návrh programu pro enkodér byl především tutoriál ze stránek NI [12]. Z obrázku 5.6 je vidět použití pouze logických prvků (a to **XOR**, negace **XOR** a **OR**). Registry dávají informaci o předcházejícím stavu, který je dále vyhodnocován. Program provede krok při vzestupné hraně signálu z enkodéru.



Obr. 5.6: Realizace enkodéru pomocí booleovské logiky

Výstupem tedy máme hodnotu natočení, nicméně bylo zamýšleno řízení rychlosti otáčení motoru, k čemuž je nutná hodnota rychlosti motoru. Tato hodnota je získána jako podíl počtu tiků enkodéru v nastaveném intervalu. Je možné provést tento výpočet v RT, protože smyčka v RT probíhá znatelně pomaleji než v FPGA, nestíhají se počítat všechny hodnoty (pokud je měřený interval příliš krátký). Pokud je interval dlouhý, RT smyčka nám dává nezkreslenou hodnotu, nicméně je nedostatečně přesná. Z toho plyne jasný závěr pro výpočet rychlosti (měření počtu tiků v intervalu) v FPGA.

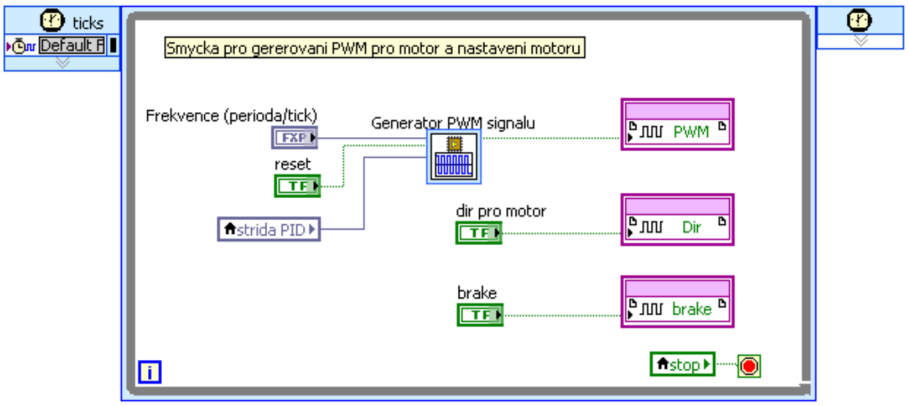
Počítání tiků v Timed loop je zde proto, jelikož nelze tuto smyčku zpomalit (vyplývá z toku dat v FPGA viz 3.1.3). Bylo by možné použít While loop, kde je možné použít časovače, nicméně bychom ztratili přesnost.



Obr. 5.7: Realizace měření rychlosti z hodnoty natočení enkodéru

5.3.2 Generování PWM

Způsobů generování PWM signálu se nabízí několik možností, ať už s pomocí porovnávání signálu se sinusovým signálem, použití časovačů v sekvenční smyčce nebo také s pomocí bloku, který generuje obdélníkový signál (PWM). Poslední metoda byla použita v návrhu.



Obr. 5.8: Realizace generování PWM signálu a ovládání dig. výstupů

5.4 Implementační poznámky

Následující poznámky se týkají programu pro real-time aplikace.

FPGA

- použití Scan enginu nebo přistupovat přímo na porty FPGA a definovat tyto vstupy a výstupy vlastním programem
- nastavení pinů na modulu na vstup a nebo výstup
- dodržovaní zákonitostí popsaných v kapitole 3.1
- program bude zkompilován externím kompilátorem (Xilinx) a vytvořený soubor obsahují nastavení propojení jednotlivých hradel a bude přenesen do cRIO a následně do FPGA

RT

- prostředník mezi FPGA a Hostitelským PC
- způsob řešení přenosu dat v a přes RT
- LabVIEW kód je převeden pro program spustitelný na RT procesoru v cRIO

5.5 Řízení DC motoru

Nyní bude navrženo a porovnáno několik způsobů pro řízení motoru. Cílem bude regulovat rychlost otáčení motoru.

5.5.1 Popis navržených programů pro DC motor

Motor 10:

- **FPGA** - program obsahuje čtení digitálních signálů z enkodéru, které převádí na hodnotu natočení. Z této hodnoty se vypočítá rychlost, na kterou je použit plovoucí filtr. Druhá smyčka obsahuje blok pro generování PWM signálu a nastavení H-můstku.
- **RT** - zde program čte hodnotu filtrované rychlosti z FPGA a zapisuje hodnotu střídy pro generování PWM v závislosti na vypočítané hodnotě z PID bloku. Tato časová smyčka je řídící a deterministická. Pro dodržení determiničnosti jsou hodnoty do této smyčky předávány s pomocí RT FIFO. Dále program obsahuje dvě komunikační smyčky pro přenos dat mezi RT a Hostitelským PC. Jedna je určena pro přenos parametrů nastavení a to s pomocí síťových sdílených proměnných. Druhá má na starost přenos hodnot zadané a skutečné rychlosti a je realizována s pomocí Network Stream.
- **Hostitelské PC** - v tomto programu jsou čteny hodnoty o rychlostech s pomocí Network Stream a zapisovány parametry nastavení do sdílených síťových proměnných. Jsou zde ovládací panely, kontrolky a graf pro rychlost.

Motor 11:

- **FPGA** - program je zde totožný jako u Motoru 10.
- **RT** - zde je struktura programu totožná jako u Motoru 10, ale s tím rozdílem, že přenos parametru z Hostitelského PC je zde pomocí Network Stream a ne sdílené proměnné. Motivací je porovnání v použití a nepoužití síťových sdílených proměnných.
- **Hostitelské PC** - stejný jako u Motoru 10 (bez přenosu síťových sdílených proměnných).

Motor 12:

- **FPGA** - program obsahuje části jako u Motoru 10, tj. smyčky pro enkodér, přepočet rychlosti a dále smyčku pro generování PWM. Je zde While smyčka obsahují blok pro PID řízení.
- **RT** - protože FPGA obsahuje v podstatě vše potřené pro řízení motoru, RT zajišťuje pouze transfér dat mezi FPGA a Hostitelským PC. Pro komunikace je užito Network stream.
- **Hostitelské PC** - program zde je totožný jako u Motoru 11.

5.5.2 Změřené charakteristiky

Jednotlivá náročnost kódu na zdroje pro FPGA je ukázána v následující tabulce:

Tab. 5.5: Přehled náročnosti programů na prostředky FPGA čipu

Název modelu	Slices	%	Flip-flops	%	LUTs	%	Bloků RAM	%
Motor 10	1633	8,0	1543	3,8	2138	5,2	0	0,0
Motor 11	1671	8,2	1543	3,8	2138	5,2	0	0,0
Motor 12	2491	12,2	2340	5,7	3351	8,2	0	0,0

Z následující tabulky plyne, že smyčky byly kompilací optimalizovány tak, že se již nenabízí další prostor ke zvýšení frekvencí těchto smyček.

Tab. 5.6: Časování programů na FPGA

Název modelu	Čas (MHz)	Max. možný čas (MHz)
Motor 10	40,02	40,71
Motor 11	40,02	41,47
Motor 12	40,02	40,19

Kompilace programů Motor 10 a Motor 11 trvala přibližně 5 minut, Motor 12 trval 7 minut (závisí na vybavení daného PC - v tomto případně procesor Intel Core2 Duo E8400 3GHz, 3 GB RAM).

Tab. 5.7: Zatížení CPU [%] v RT pro dvě periody smyčky

Název modelu	Zaplnění paměti	20 ms		5 ms		Minimální perioda smyčky
		Timed loop	Celkem	Timed loop	Celkem	
Motor 10	45 MB	30 %	56 %	75 %	99 %	1 ms
Motor 11	44 MB	34 %	52 %	80 %	95 %	1 ms
Motor 12	46 MB	37 %	64 %	45 %	78 %	1 ms

5.5.3 Porovnání kvality řízení DC motoru

Použité programy slouží k regulaci rychlosti otáčení DC motoru. Porovnání kvality řízení bylo provedeno s pomocí spočítaných statistických charakteristik z naměřených dat (viz kapitola 3.5) a vytvořena přehledová tabulka.

Tab. 5.8: Statistické charakteristiky programů pro DC motor

Název modelu	MSE	MAE	MaxError
Motor 10	0,0911	0,2257	1,6418
Motor 11	0,1111	0,2508	1,8036
Motor 12	0,3494	0,3278	2,0000

Specifikace měření:

- Perioda smyčky v RT 1 ms.
- U Motoru 12 v FPGA byl počet tiků PID smyčky 15.
- Z důvodu malého rozlišení enkodéru a z toho plynoucí malé rozlišení rychlosti otáčení motoru jsou výsledné statistické údaje u programů pro Motor vyšší. U Motoru 12 jsou nejvyšší, protože filtrace rychlosti je prováděna v FPGA na datovém typu integer číslech než u ostatních, kde filtrace rychlosti je prováděna v RT na datovém typu double.

5.6 Řízení škrtící klapky

Program má klasickou strukturu, kde byl navržen program pro FPGA, RT a Hostitelský PC. FPGA obsahuje smyčku pro generování PWM (princip stejný jako u DC motoru) a nastavení H-můstku. Dále je zde While smyčka, ve které je čtena analogová hodnota napětí na potenciometru.
V RT je dodržena podobná struktura jako u řízení DC motoru.

- Řídící smyčky s nejvyšší prioritou obsahují PID regulátor s/bez kompenzace tření s dodržením vlastností pro determinismus.
- Komunikační smyčka pro přenos údajů o natočení klapky. Využito Network stream.
- Komunikační smyčka pro přenos údajů parametrů nastavení s využitím Network stream.

V Hostitelském PC jsou přenášené hodnoty zobrazovány do grafu a umístěny kontrolky pro nastavení parametrů.

5.6.1 Popis navržených programů pro škrtící klapku

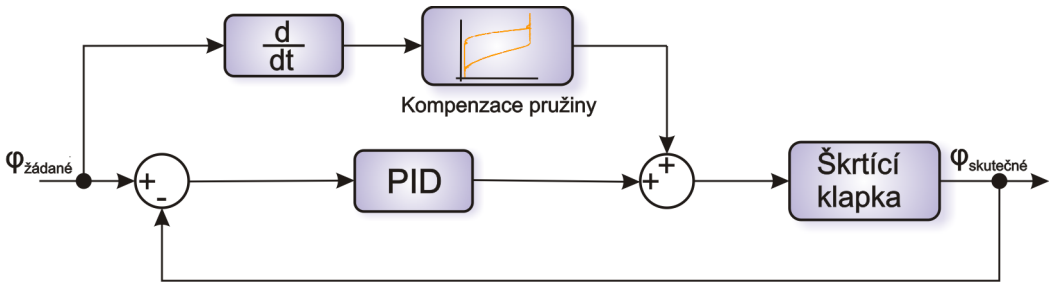
Klapka 1

- **FPGA** - program obsahuje časovanou smyčku s generátorem PWM a s ovládáním směru a brzdy na H-můstku. Dále While smyčku pro čtení analogové hodnoty napětí z potenciometru.

- **RT** - řídící smyčka s nejvyšší prioritou obsahuje čtení dat z FPGA a PID regulátor. Pro zachování determinismu jsou hodnoty předávány do komunikačních smyček s pomocí proměnných s RT FIFO. Dále jsou zde dvě komunikační smyčky. Přenos dat mezi Hostitelským PC je řešen pouze přes Network Stream. Celkově je použita o něco jednodušší struktura než u řízení motoru, nicméně pro vývoj aplikace je dostačující.
- **Hostitelské PC** - v tomto programu jsou čteny hodnoty zadaného a skutečného natočení klapky a vykreslovány v grafu. Dále jsou zde umístěny ovládací prvky pro RT.

Klapka 2

- **FPGA** - naprosto totožné jako u Klapka 1.
- **RT** - struktura programu je stejná jako u Klapka 1 pouze s tím rozdílem, že je zde umístěna kompenzace nelinearity pružiny. Kompenzace pružiny je zde provedena z naměřeného chování klapky v závislosti na střídě. Na základě požadovaného směru klapky a pozice je přičtena daná hodnota k akčnímu zásahu vycházejícímu z regulátoru. Tímto je docíleno jakési pomoci při pohybu. Při ustálení je kompenzace vždy nulová, protože se již neprojevuje efekt odporu pružiny.
  Schéma řízení je možné vidět na následujícím obrázku.



Obr. 5.9: Schéma regulace s kompenzací pružiny

- **Hostitelské PC** - totožný s programem v Klapce 1.

Klapka 3

- **FPGA** - vychází opět z programu pro Klapku 1, ale je dále rozšířen o PID řízení ve vlastní While smyčce.
- **RT** - pouze zprostředkovává komunikaci mezi FPGA a Hostitelským PC.
- **Hostitelské PC** - stejný opět jako u Klapka 1.

5.6.2 Změřené charakteristiky

Jednotlivá náročnost kódu na zdroje pro FPGA je ukázána v následující tabulce:

Tab. 5.9: Přehled náročnosti programů na prostředky FPGA čipu

Název modelu	Slices	%	Flip-flops	%	LUTs	%	Bloků RAM	%
Klapka 1	1664	8,1	1712	4,2	2103	5,1	0	0,0
Klapka 2	1679	8,2	1712	4,2	2102	5,1	0	0,0
Klapka 3	2652	12,9	2482	6,1	3232	7,9	0	0,0

Možnosti zrychlení FPGA smyčky je možné vidět v následující tabulce:

Tab. 5.10: Časování programů na FPGA

Název modelu	Čas (MHz)	Max. možný čas (MHz)
Klapka 1	40,02	46,05
Klapka 2	40,02	46,81
Klapka 3	40,02	43,78

Využití CPU při běhu programu v RT porovnáno v následující tabulce:

Tab. 5.11: Zatížení CPU [%] v RT pro dvě periody smyčky

Název modelu	Zaplnění paměti	20 ms		5 ms		Minimální perioda smyčky
		Timed loop	Celkem	Timed loop	Celkem	
Klapka 1	44 MB	31 %	48 %	67 %	82 %	1 ms
Klapka 2	44 MB	31 %	48 %	70 %	87 %	1 ms
Klapka 3	44 MB	34 %	50 %	72 %	86 %	1 ms

5.6.3 Porovnání kvality řízení škrtící klapky

Použité programy slouží k regulaci natáčení ventilu škrtící klapky automobilu. Porovnání kvality řízení bylo provedeno s pomocí spočítané statistické charakteristiky z naměřených dat (opět bližší vysvětlení viz kapitola 3.5) a vytvořena přehledová tabulka.

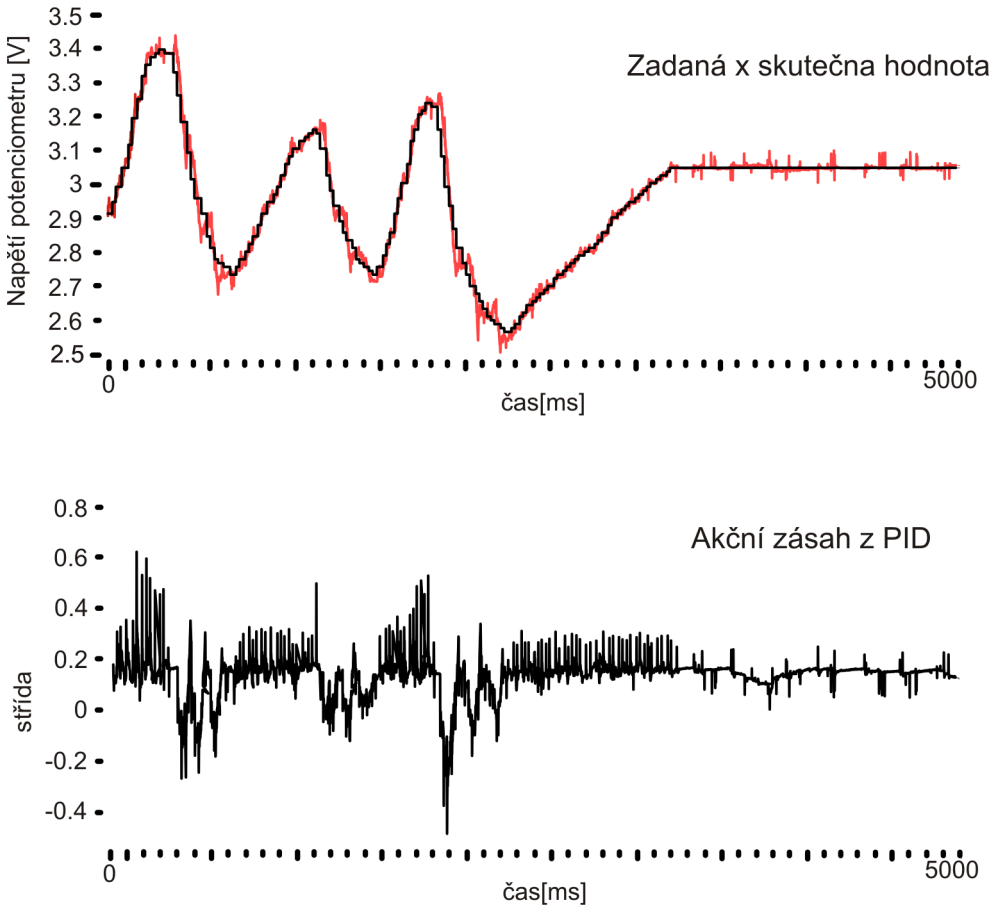
Tab. 5.12: Statistické charakteristiky programů pro škrtící klapku

Název modelu	MSE	MAE	MaxError
Klapka 1	0,0106	0,0527	0,6085
Klapka 2	0,0007	0,0161	0,1460
Klapka 3	0,0164	0,1133	0,5247

Specifikace měření:

- Perioda smyčky v RT 1 ms.
- U Klapky 3 v FPGA byl počet tiků PID smyčky 180.
- Doba načítání jedné hodnoty analogového signály byla 180 tiků.

Na ukázku grafy z měření Klapky 2



Obr. 5.10: Odezva škrtící klapky na navržené řízení

5.7 Celkové zhodnocení programů pro reálné aplikace

5.7.1 Doba trvání jednotlivých částí programu

V následující tabulce bude ukázána časová náročnost vybraných částí programu - slouží především k obecné představě. Tyto výsledky byly získány s pomocí aplikace Real-time trace toolkit, která umožní změřit reálný časový okamžik mezi dvěma body, které si v programu vybereme. Dává také celkový přehled o celém chodu programu z pohledu spuštěných úkolů v daném čase. Nicméně bloky pro označení místa v programu viditelné následně v reportu lze vložit pouze tam, kde je veden error cluster.

Tab. 5.13: Doba trvání různých částí programu

Perioda smyčky 1 ms	
Část programu	Doba procesu
Přenos DMA	60 us
Kompenzace pružiny	27 us
PID	18 us
RT filtr	15 us
Trace toolkit	18 us
Network Stream do Host	2 ms

Pozn.: Hodnota přenosu dat z RT do Hostitelského PC je různá, záleží na množství přenášených dat - zda je použit zásobník atd.

5.7.2 Hodnocení náročnosti tvorby jednotlivých programů

V této části bude zhodnocena uživatelská náročnost vytváření jednotlivých programu pro FPGA, RT a Hostitelské PC pro systémy postavené na SW LabVIEW. Protože výsledná aplikace se skládá ze všech tří částí, je i toto hodnocení postavené na této skutečnosti. Nejsou zde tedy posuzovány tři různé programy, ale jeden komplexní.

- **FPGA** - základem je dobré vědět, co je požadováno a co je možno realizovat. Nabídka bloků z knihovny v LabVIEW je nejchudší, proto se vytvořené programy mohou jevit jako jednoduché, nicméně i právě z toho důvodu a při snaze dosáhnout nějaké vyšší logiky, je možné vytvořit program, který bude na pohled složitý, ale vykonávat jednoduchý algoritmus, který by bylo možno jednoduše vyřešit například v RT. Při použití datového typu FXP je potřeba rozhodnou o velikosti jednotlivých proměnných. Komunikace a zpracování signálů z I/O portů je jednoduchá.
  Střední obtížnost programování
- **RT** - program v RT může narůst až do obřích rozměrů a vysoké složitosti, proto je třeba pracovat uváženě a program co nejvíce strukturalizovat a rozdělovat do různých smyček. Z toho plyne problém v komunikaci mezi smyčkami a také komunikace mezi Hostitelským PC. Vytvoření aplikace, která přijímá data z FPGA a provádí s nimi nějaké úkony je jednoduchá, obtížnější je zajistit determiničnost celé smyčky, pokud je to vyžadováno. Tento fakt vyžaduje hlubší znalosti a také zkušenosti. Je záhodné věnovat této problematice čas, protože výsledek je mnohem lepší, i když se na to první pohled nemusí zdát. Knihovna je zde bohatší o různé komplexní algoritmy, nicméně je třeba mít stále na paměti omezení plynoucí z běhu programu v RT.
  Vysoká obtížnost programování

- **Hostitelské PC** - neboli nadřazený počítač. Zde je k dispozici plnohodnotné LabVIEW, což umožňuje až téměř neomezené funkční možnosti. Nesmí se ale zapomínat, že největší motivací je provozovat hlavní výpočet v RT, kde máme k dispozici rychlý deterministický systém. Proto se PC využívá především k ovládání programu, ukládání dat atd., což nevyžaduje velké znalosti, proto je práce snadná. Je zde ale možno se setkat se zadáním vytvořit uživatelsky příjemné prostředí, což je zcela odlišná práce od vytváření výpočetních algoritmů.
  Nízká obtížnost programování

6 Spuštění simulinkovského modelu v prostředí LabVIEW

V této kapitole bude uvedeno, jaké možnosti se nabízí při využití softwaru Simulink při práci s cRIO (lze to zobecnit i na ostatní hardware od NI) a programováním v LabVIEW. Motivací je ukázat, že využití této spolupráce nemusí být zase až tak složité a naskýtá určité výhody. Nicméně hodně záleží na zkušenostech, schopnostech a zvyklostech uživatele. Je to tedy otázka, která nemá jednoznačnou odpověď.

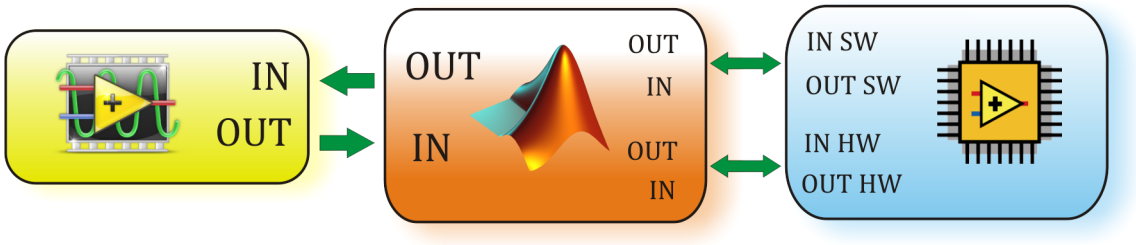
6.1 Vytvoření jednoduché aplikace v Simulinku určenou pro běh v RT

V této kapitole budou ukázány oba přístupy při vytváření aplikací pro RT HW. Při použití VeriStandu ale nebude výsledný produkt spuštěn na cRIO a to z důvodu nedostatečné velikosti operační paměti cRIO potřebné pro běh RT VeriStand enginu na cRIO.

K dispozici jsou 3 části programu:

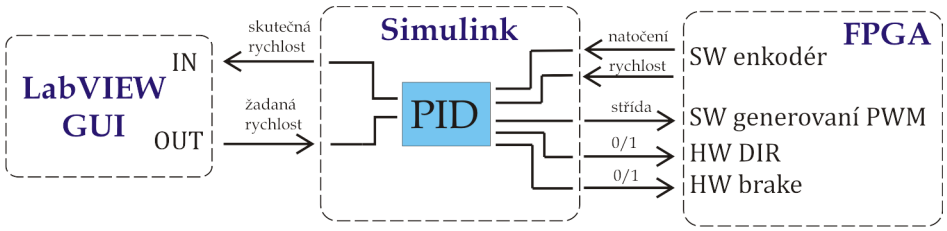
- FPGA - možnost dosáhnout nejrychlejší časové smyčky, vhodné pro zpracování (generování) signálů ze senzorů, neovlivňuje RT.
- Simulink - kód, který bude následně nahrán do RT prostředí, vhodný pro řídící algoritmy.
- LabVIEW GUI - interface mezi uživatelem a programem, zobrazování hodnot a ovládání programu.

Příklad bude uveden na následujícím schématu. Poté podle toho schématu bude doplněn program pro FPGA o požadované algoritmy a nastavení vstupů/výstupů. V Simulinku budou taktéž doplněny vstupy a výstupy pro úspěšné provázání programu (zmiňované tři vrstvy).



Obr. 6.1: Schéma vrstvení programu

Pro konkrétní aplikace řízení DC motoru může schéma programu vypadat takto:



Obr. 6.2: Schéma aplikace DC motor

6.1.1 Program v Simulinku

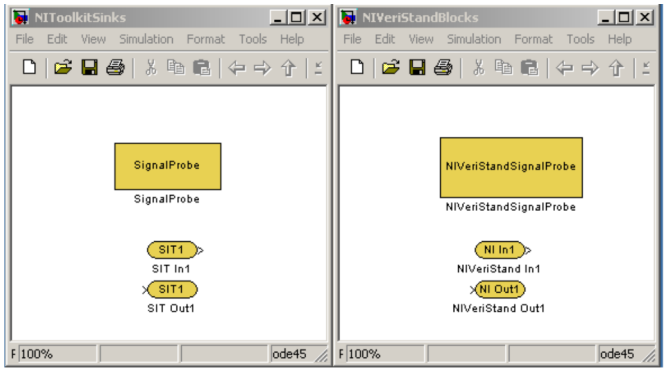
V Simulinku je sestaven řídící program, který obsahuje klasické bloky knihovny a dále také z důvodu následného rozpoznání v LabVIEW, vstupní a výstupní bloky získané z knihovny SIT. Použitím těchto vstupů a výstupu získáme po namapování přímé provázání simulinkovského modelu s HW vstupy a výstupy. Pro zobrazení dat tekoucí programem je potřebné do modelu vložit zesílení, na které se následně v LabVIEW naváže indikátor.

Na obrázku 6.8 je ukázka jednoduchého modelu pro řízení DC motoru. Model musí obsahovat zmiňované bloky pro provázání s NI SW.

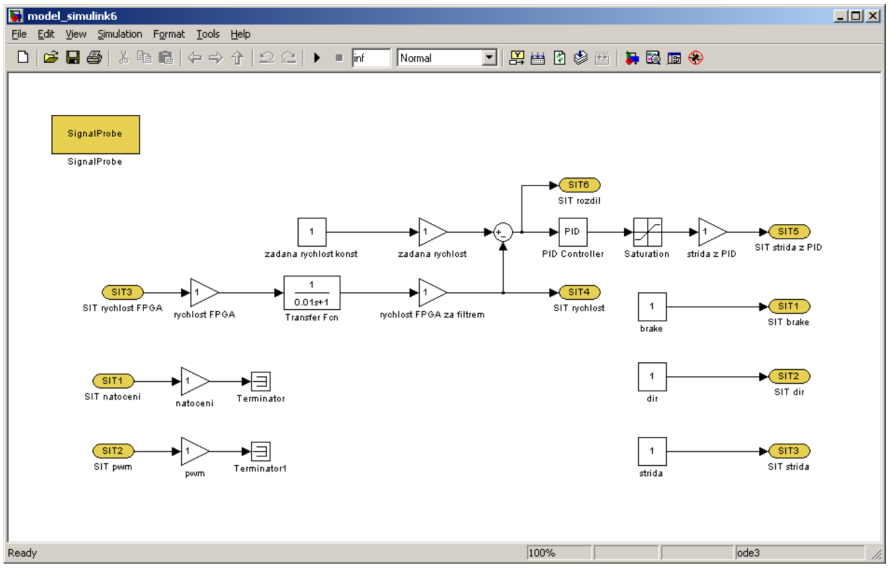
Po zkompletování modelu je třeba vytvořit .dll [1] soubor, který se poté použije dále v LabVIEW. K tomuto slouží v Simulinku Real-time Workshop, kde se jako cílové zařízení vybere NI Real-time Target. Pro použití na zařízení PXI je to vše, co je potřeba, pro použití na cRIO je potřeba další komplice, která vytvoří soubor pro operační systém VxWorks [2], který je potom nahrán do cRI0.

[1] Dynamicky linkovaná knihovna obsahující všechny potřebné informace pro otevření modelu v LabVIEW.

[2] VxWorks je real-time operační systém vyvíjený firmou Wind River Systems z USA. Je navržen pro použití na embedded systémech. Nachází využití v průmyslových robotech, letectví, elektronice a také NI cRIO 901x, 902x, 907x.



Obr. 6.3: Knihovny přidané do Simulinku



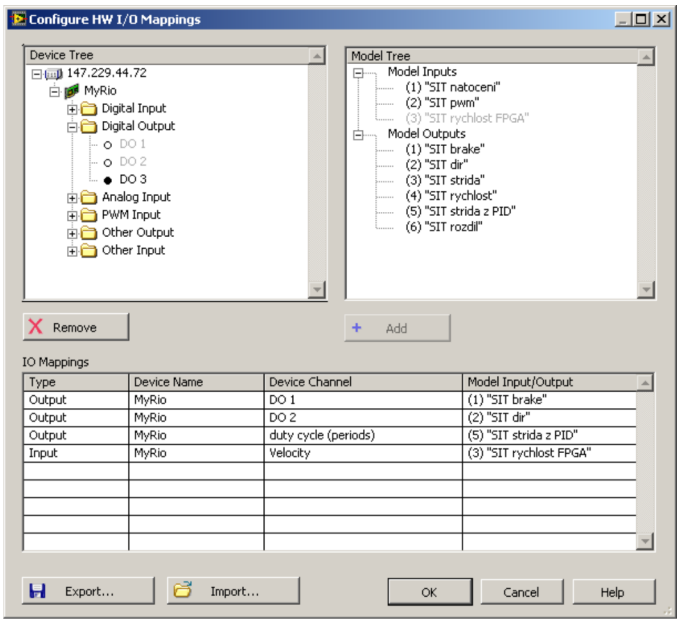
Obr. 6.4: Jednoduchý model řízení DC motoru v Simulinku

Množství použitelných bloků z knihovny Simulink je v podstatě obdobné jako při vytváření programů pro následnou kompilaci např. do jazyka C. Řešič je nastaven na pevný krok.

6.1.2 Použití LabVIEW - SIT

Před samotným spuštěním programu v LabVIEW je třeba si připravit program pro FPGA, kde se namapují porty pro daný HW. V odpovídající složce v PC, kde je LabVIEW nainstalován se nachází šablona pro namapování portů. Do tohoto programu je také možné přidat další různé algoritmy, které budou spuštěny v FPGA. Může se jednat například o zpracování signálu z enkodéru, kde na výstupu bude

hodnota natočení nebo také generovat PWM ze vstupní hodnoty střídy. Tyto bloky, pokud by byly v RT (které například na cRIO běží na 1kHz), tak nebudou stíhat zpracovávat signál. Poté je nutné program zkompilovat a vytvořit tzv. bitfile [3]. Získaný bitfile je potom potřeba správně pojmenovat a překopírovat do složky, která obsahuje již další předprogramované bitfily pro další různé zařízení. Správnost tohoto kroku se projeví v nastavení SIT v LabVIEW při mapování SW portů s HW porty.



Obr. 6.5: Mapování HW a SW portů mezi Simulinkem a FPGA

Nyní je vše potřené již nastaveno a může se přejít k vygenerování programu pro RT. Tento proces je plně automatický a není potřeba dalších úprav. Součástí vygenerovaného programu je i uživatelské ovládání, které slouží ke spuštění modelu po jeho nahrání do RT.

6.1.3 Použití LabVIEW - VeriStand

Jak již bylo zmíněno výše, je možno využít dvou nástrojů pro správu real-time modelu. Tím bohatším je právě VeriStand. V popisu instalace je uvedena nutná instalace enginu pro RT zařízení, který zabezpečuje komunikaci a správu modelu v RT. Tento program je relativně komplexní, proto pro jeho běh je požadováno minimálně 128 MB operační paměti. Zařízení cRIO 9073 obsahuje pouze 64 MB

[3] Struktura, obsahující kód pro FPGA, jak nastavit hradla. Výsledek optimalizačního procesu kompilátoru, který se může může při opakovaném kompilování lišit.

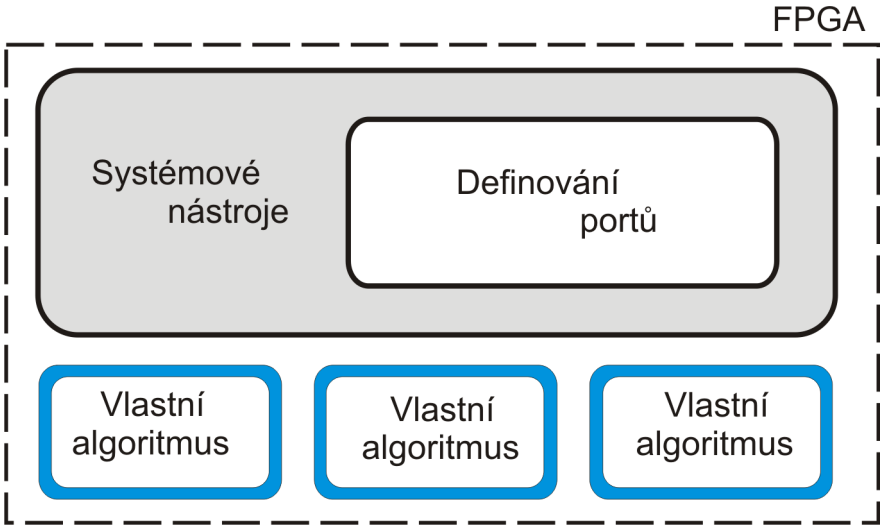
operační paměti, proto není možné spustit model v RT. Podrobnější popis programu je možno nalézt například zde [17].

6.1.4 Vytvoření bitfile souboru pro SIT (případně VeriStand)

Tento soubor je důležitý pro namapování portů v FPGA. Pokud v tomto programu nenamapujeme porty správně, neuvidíme je dále při práci se SIT nebo VeriStandem. Protože tyto systémy jsou především určeny pro práci s PXI, jsou zde již před-připraveny soubory pro mnoho verzí tohoto systému. Pro systémy cRIO je nachystána šablona, která se musí upravit podle konkrétního cRIO. SIT případně VeriStand definují program v RT, proto pokud chceme vložit nějakou část kódu do FPGA, musíme to učinit v tomto souboru a připravit jeho výstup tak, aby byl viditelný jako jakýkoliv jiný reálný I/O port. Programování je zde stejné jako při klasickém psaní programu pro FPGA. Je možné zde implementovat algoritmy vyžadující rychlý krok smyčky - například zpracování enkodéru, filtry, generování PWM a jiné. Tento soubor nelze vytvořit nebo upravit například ze Simulinku, proto se nelze vyhnout programování v LabVIEW.

Po zkompletování se program zkompiluje a výsledný vytvořený bitfile [4] se překopíruje do knihovny, odkud bude dále viditelný a použitelný.

Postup vytvoření bitfilu pro SIT lze nalézt zde [15] a pro použití cRIO pod VeriStandem zde [16].



Obr. 6.6: Mapování HW a SW portů mezi RT a FPGA

[4] soubor definující propojení hradel v FPGA

6.2 Implementační poznámky

6.2.1 Příprava pro konverzi - instalace

1. Instalace Matlabu - je nutné zkontrolovat kompatibilitu mezi Simulation Interface toolkitem a Matlabem.
2. Instalace MS Visual Basicu C++ , kontrola kompatibility mezi Matlabem a MS Visual Basicem (příkazem „mex“ lze v Matlabu ověřit přítomnost kompilátoru MS Visual Basic studio).
3. Instalace NI LabVIEW.
4. Instalace NI Veristand (pokud je vyžadováno).
5. Instalace GCC kompilátoru pro VxVorks - provést potřebná nastavení popsaná v Readme souboru ve složce Simulation Interface toolkit a podsložce VxVorks: readme_sit_vxworks.txt.
6. Vytvořit bitfile pro daný typ cRIO, kde je třeba namapovat potřebné I/O porty (rozvedeno více v kapitole 6.1.4).
7. Instalace potřebných enginů do RT zařízení.

Obecná doporučení:

- Používat anglický operační systém a veškeré programy spolupracující s LabVIEW instalovat anglicky. Mohou se vyskytovat chyby zapříčiněné použitím českých Windows, které nicméně budou v následné verzi LabVIEW opraveny, ale které znepříjemňují práci a najití řešení těchto chyb nemusí být snadné.
- Pokud je nainstalován i SW Xilinx ISE Design, dojde poté ke kolizi s kompilátorem Xilinx standardně instalovaným s LabVIEW pro kompilaci FPGA. Je nutné následně nastavit správné cesty, aby následná kompilace v LabVIEW probíhala správně.

6.3 Maximální výkon programů implementovaných pro SIT

Bylo vytvořeno několik programů v Simulinku, které slouží k otestování maximálních rychlostí smyček dosažitelných po následné implementaci v SIT SW LabVIEW.

6.3.1 Použité testovací programy pro porovnaní

Tab. 6.1: Přehled použitých programů pro SIT

Název modelu	Popis
A/D VSTUP	Program obsahuje pouze vstupní blok, zesílení a výstupní blok, na které se poté v SIT napojí analogové nebo digitální vstupy.
10x PID	Analogový vstup je rozveden pro deset PID bloků. Tyto jednotlivé bloky obsahují zesílení, integrátor a derivaci.
PID MOTOR	Reálný program pro řízení DC motoru. Obsahuje PID regulátor, potřebné vstupy a výstupy pro signály a filtr na vstupu.

6.3.2 Výsledek porovnání programů pro SIT

Tab. 6.2: Dosažitelné rychlosti smyček v SIT

Název modelu	Zaplnění paměti	Perioda smyčky		
		600 us	700 us	800 us
A/D VSTUP	53,8 MB	×	✓	✓
10x PID	53,8 MB	×	×	✓
PID MOTOR	54,0 MB	×	×	✓

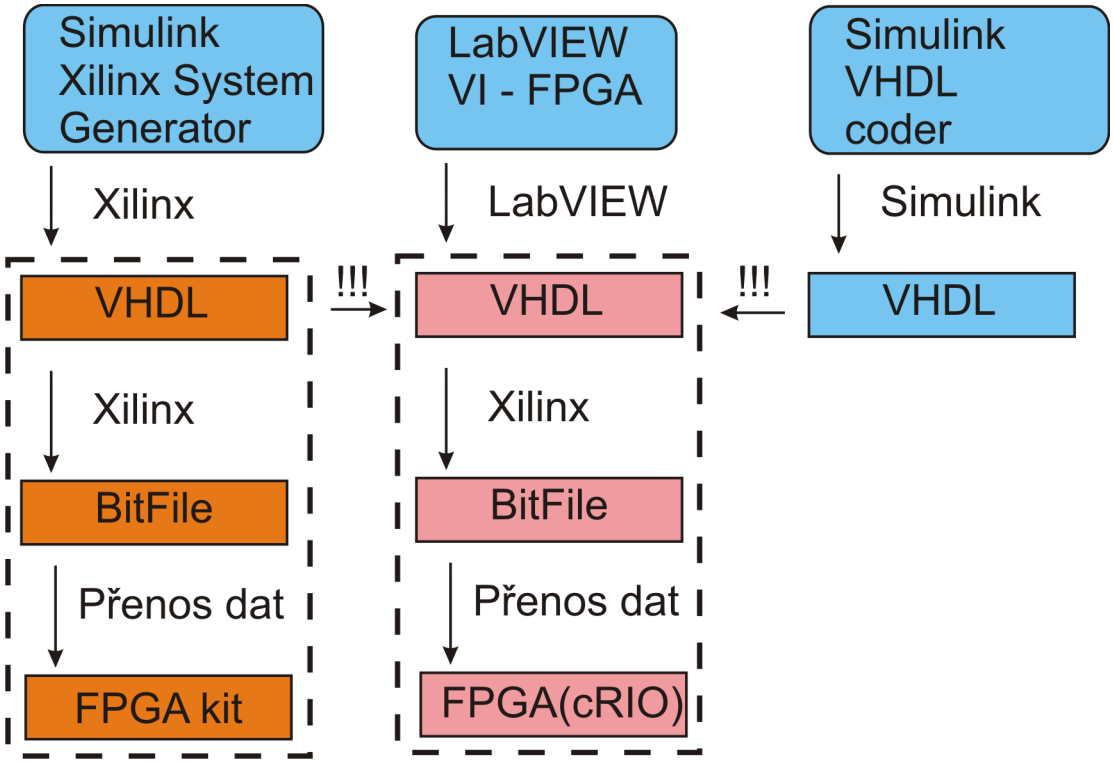
Z tabulky 6.2 je vidět, že složitější programy je možné provozovat od periody smyčky 800 us, nicméně z důvodu nemožnosti měření zatížení CPU a použití Real-time trace toolkit je obtížné stanovit, zda daná smyčka má deterministiký průběh. Doporučená doba smyčky je 1 ms, která má dostatečnou rezervu pro zachování determinismu a také prostor pro výpočet složitých algoritmů. Větší omezení může být velikost paměti cRIO. Z tabulky je vidět, že samotný program zabírá zanedbatelně málo místa oproti obslužným aplikacím SIT. Místo v paměti lze zvýšit odinstalováním nepotřebných enginů z cRIO.

6.4 Implementace jednoduchých simulinkovských příkladů do FPGA

Postup pro vytvoření CLIP[5] je možno nalézt na internetu například zde [21].

Před samotnou konverzí je nutno věnovat pozornost správnému nastavení použitého generátoru. Protože FPGA - neboli pospojované hradlové pole konají určitou činnost při detekci hrany hodin, je třeba tento signál do programu implementovat. To se nemusí provést automaticky při konverzi. Je nutné tedy už samotný program v Simulinku navrhovat s tímto faktem. Další problémem může být vygenerovaný kód, pokud nemáme potřebné znalosti pro jeho alespoň částečné pochopení, v podstatě nejsme schopni ho při případné chybě opravit. V LabVIEW je také nutné provést dodatečné nastavení a změny, aby byla následná kompilace a program funkční.

Schéma postupu práce je naznačeno na následujícím diagramu:

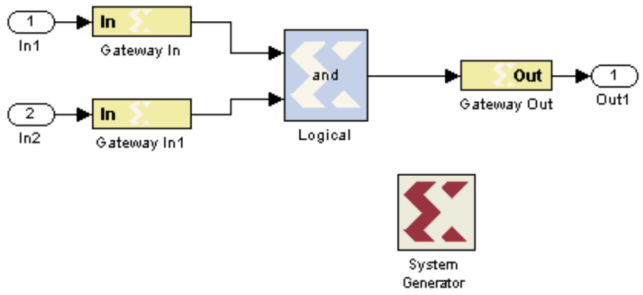


Obr. 6.7: Možnosti získání VHDL kódu pro FPGA

[5] CLIP = Component Level IP - blok obsahující vložený VHDL kód

6.4.1 Program vygenerovaný z Xilinx System generátoru

Po instalaci potřebného SW od Xilinxu se objeví v Simulinku další knihovna obsahující nové bloky pro programování. Tyto bloky jsou optimalizované pro konverzi do FPGA čipů. U generátoru lze nastavit typ cílového zařízení (může se jednat jak o samostatný čip nebo o celý vývojový kit firmy Xilinx). Při použití v LabVIEW se vygenerovaný VHDL kód implementuje do komponenty CLIP a dále se s ní pracuje jako s obyčejným funkčním blokem z LabVIEW.



Obr. 6.8: Ukázka jednoduchého modelu v Simulinku s použitím Xilinx System Generator

6.4.2 Program napsaný ve VHDL

Další možností implementování VHDL kódu do FPGA cRIO je sestavit daný kód ručně. Následující příklad sčítá dvě 16-bitové integer čísla. Je nutné docílit provádění instrukcí při detekci vzestupné hrany hodin, jak je vidět na následují části kódu, která čeká, až tento podnět nastane a poté provede součet.

```
%================================================
    begin process(aReset, clk) begin
        if(aReset = '1') then
            cAddOut <= (others => '0');
        elsif rising_edge(clk) then
            cAddOut <= std_logic_vector(signed(cPortA) + signed(cPortB));
        end if;
    end process;
%================================================
```

Vytvořený VHDL kód je pak třeba vložit do komponenty pro CLIP, která vytvoří nové bloky pro vstupy a výstupy z programu v našem projektu. Tyto pak můžeme použít v našem programu a dostaneme se takto k našemu vytvořenému

VHDL kódu. Je nutné provést ještě další dodatečná nastavení. Poté již jen celý program zkompilovat a nahrát do FPGA. Popis postupu práce je možno nalézt zde [21].

6.5 Porovnání náročnost vytvoření programů s využitím Simulinku

- **Samotná idea využití Simulinku při práci s HW a SW od NI** - již při plánování této práce je dobré zvážit, zda pro práci volit i SW nástroj Simulink, zda LabVIEW není dostačující pro všechny naše potřeby. Je určitě vhodné, pokud se bude spolupracovat se Simulinkem, aby s tím uživatel měl již nějakou předešlou zkušenost - především se samotným nastavením celého prostředí. Tento fakt může neúměrně prodloužit dobu strávenou vývojem dané aplikace. Nicméně nic není nemožné, proto pokud se přeneseme přes počáteční obtíže se správným nastavením, můžeme se plně začít věnovat samotné práci.
  Střední obtížnost programování
- **Simulinkovský model** - uživatel zběhlý s prací se Simulinkem nebude mít s touto částí problém, protože se neliší od běžné práce se Simulinkem. Jediný rozdíl je v použití jiných bloků pro definování vstupů a výstupů v modelu, které budou následně rozpoznány v LabVIEW.
  Nízká obtížnost programování
- **Simulation Interface toolkit** - pro další použití je nutno správně zkompilovat simulinkovský model, což může být obtížné správně nastavit, nicméně pak následná kompilace už není obtížná. Model pro LabVIEW se vygeneruje sám, proto je nutné mít simulinkovský model opravdu kompletní - do vygenerovaného modelu již nelze (moc) zasáhnout. Je taktéž nutno mít správně nastavený program pro FPGA, aby vše správně spolupracovalo.
  Střední obtížnost programování
- **VeriStand** - tento program je velice všestranný, proto záleží, jaký bude vstup. Poté je práce ne příliš obtížná, program je rozumně strukturovaný a intuitivní. Pokud bude používáno cRIO, je zde opět záležitost s nastavením programu pro FPGA.
  Nízká obtížnost programování
- **Implementace Simulinku do FPGA** - zde se nabízí v podstatě dva přístupy, napsat si VHDL kód s pomocí překladu ze Simulinku, což vyžaduje vyšší znalosti programování VHDL (případně HDL) a nebo využít „inteligentnější“ bloky pro Simulink, který vytvoří správný kód. Tento přístup nicméně vyžaduje složitá nastavení a i zde je třeba mít určité znalosti programování.
  Vysoká obtížnost programování

7 Závěr

V této práci jsem se zabýval především zkoumáním vlastností a výkonu real-time počítače cRIO firmy National Instrumets. Formálně lze práci rozdělit do tří částí a to testování jednotlivých částí systému (FPGA a RT) za pomocí jednoduchých programů (kapitola 4), použití cRIO pro řízení reálných soustav (kapitola 5) a také průzkum možností převodu simulinkovského programu do RT nebo FPGA cRIO (kapitola 6). Byly vypracovány porovnávací tabulky, které ukazují výkonnost daného systému. Tvorba samotných aplikací byla také porovnávána a je zhodnocena v závěru daných kapitol.

Tato práce ukazuje, že daný systém cRIO-9073 je vhodný pro použití při řízení procesů (jako například DC motor, použitý v této práci). Je nutné ale počítat s omezenými zdroji především v FPGA, kdy je potřeba program navrhovat tak, abychom se i při relativně jednoduché aplikaci nedostali na hranice systému. Na druhou stranu ale FPGA nabízí velice rychlé vykonávání programu. Získané výsledky ukazují, že v RT je možno dosahovat period smyček 1 ms i při relativně složitém programu. Vše záleží na samotném návrhu programu. Vyšší rychlosti smyček jsou možné, ale nemusí být již zaručena determinočnost celku. Perioda 1 ms je rozumným kompromisem mezi rychlostí a spolehlivostí. Při spouštění zkonvertovaných simulinkovkých programů je potřeba velké množství paměti, které je zabráno podpůrnými procesy. Výsledný vygenerovaný kód je v LabVIEW téměř needitovatelný a jsou zde omezené možnosti na jeho rozšíření o bloky z LabVIEW. Na druhou stranu ale máme způsob, jak spustit daný kód v cRIO, což je určitě výhodné. Zde opět použití programu rychlejšího než 1 ms je na zvážení. Nakonec i převod simulinkovského kódu přímo do FPGA je realizovatelný, ale vyžaduje více zkušeností a znalostí a může se zde vyskytovat mnoho omezení.

Domnívám se, že cíle této práce byly splněny a výsledky budou sloužit jako dobrý přehled o možnostech cRIO. Programy v přiloženém archivu lze použít jako materiál pro seznámení s LabVIEW a pro porozumění návrhu real-time aplikací.

8 Seznam použitých zkratek a symbolů

Tab. 8.1: Seznam použitých zkratek a symbolů

Zkratka	Popis
NI	Z anglických slov „National Instruments“, což je název firmy.
cRIO	Z anglických slov „Compact Reconfigurable I/O“, což znamená kompaktní systém nastavitelných vstupů a výstupů.
FPGA	Z anglických slov „Field-programmable gate array“, což znamená programovatelné hradlové pole.
RT	Z anglických slov „Real-time“, což znamená běh například programu v reálném čase.
FXP	Z anglických slov „Fixed-Point“, což znamená čísla s pevnou desetinou čárkou.
SIT	Z anglických slov „Simulation Interface Toolkit“, což je název toolkitu v LabVIEW, který se stará o spuštění aplikací vytvořených v programech jiných výrobců v LabVIEW.
PXI	Z anglických slov „PCI eXtensions for Instrumentation“, což je zkratka pro robustní výkonný počítač pro průmyslové použití.

9 Literatura a odkazy

[1] PXI Platform:, http://www.ni.com/pxi/, 2011-05

[2] National Instrumets:, http://www.ni.com/, 2011-05

[3] LabVIEW Real-Time Application Development Course Manual:, http://sine.ni.com/nips/cds/view/p/lang/en/nid/10292l, 2009-02

[4] LabVIEW FPGA Course Manual:, http://sine.ni.com/nips/cds/view/p/lang/en/nid/10292l, 2009-08

[5] FPGAs - Under the Hood:, http://zone.ni.com/devzone/cda/tut/p/id/6983, 2011-05

[6] Wikipedia - VHDL:, http://en.wikipedia.org/wiki/VHDL, 2011-05

[7] Electronic throttle control:, http://en.wikipedia.org/wiki/Electronic_throttle_control, 2011-05

[8] Grepl, R.: *Mechatronika - vybrané problémy*, FSI, VUT v Brně, ISBN 978-80-214-3804-0, 2007

[9] Transmotec - technický list:, www.transmotec.cz/download.php?fileid=3, 2010-06

[10] Xilinx - přehled FPGA čipů:, http://www.xilinx.com/products/silicon-devices/fpga/index.htm, 2011-05

[11] Quadrature Encoder Velocity and Acceleration Estimation with CompactRIO and LabVIEW FPGA:, http://zone.ni.com/devzone/cda/tut/p/id/3921, 2011-05

[12] LabVIEW 2010 Simulation Interface Toolkit Readme:, http://digital.ni.com/public.nsf/allkb/4D65725A055138798625776C006234B9, 2011-05

[13] Supported and Compatible Compilers:, http://www.mathworks.com/support/compilers/previous_releases.html, 2011-05

[14] Developing Shared Libraries for the cRIO-901x and Other VxWorks Targets:, http://zone.ni.com/devzone/cda/tut/p/id/5694, 2011-05

[15] Creating a Custom FPGA Bitfile:, http://zone.ni.com/reference/en-XX/help/371504F-01/lvsithowto/sit_h_custfpga/, 2011-05

[16] Creating FPGA-Based I/O Personalities for NI VeriStand:, http://zone.ni.com/devzone/cda/tut/p/id/9349, 2011-05

[17] NI VeriStand: Real-Time Testing and Simulation Software:, http://www.ni.com/veristand/, 2011-05

[18] Kindgren, O.: *Executing Simulink models on FPGA in a LabVIEW environment*, Diplomová práce, Department of Computer Science and Engineering, CHALMERS UNIVERSITY OF TECHNOLOGY UNIVERSITY OF GOTHENBURG,Göteborg, Sweden, 2009

[19] HOW TO CONNECT SIMULINK TO LABVIEW IN ORDER TO COLLECT SYSTEM DATA:, http://www1.mengr.tamu.edu/aparlos/MEEN364/Simulink2LabVIEW.pdf, 2011-05

[20] Using VHDL Code as Component-Level IP:, http://zone.ni.com/reference/en-XX/help/371599F-01/lvfpgaconcepts/using_component_ip/,

[21] Importing External IP into LabVIEW FPGA:, http://zone.ni.com/devzone/cda/tut/p/id/7444, 2011-05

[22] LabVIEW Execution Trace Toolkit User Guide:, http://www.ni.com/pdf/manuals/323738a.pdf, 2011-05