VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

HIL MODEL ELEKTROMECHANICKÉHO SYSTÉMU
HIL MODEL OF ELECTROMECHANICAL SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Lukáš Malík**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Blaha, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Lukáš Malík ***ID:*** 162905
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

HIL model elektromechanického systému

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s programovacím jazykem Modelica, s grafickým programovacím nástrojem LabView a možnostmi systému CompactRIO.
2. V jazyce Modelica realizujte model vícehmotového mechanického systému, který bude použit jako příklad v následujících bodech.
3. Nastudujte možnosti ukládání modelů popsaných jazykem Modelica do FMU (Functional Mockup Unit) a jejich využití v LabView.
4. Naimportujte vygenerované FMU do prostředí LabView RealTime.
5.Propojte část FMU s LabView FPGA, kde budou použity existující modely elektrických motorů, aby se CompactRIO tvářilo jako HIL model elektromechanického systému.

DOPORUČENÁ LITERATURA:

[1] VLACH, J., HAVLÍČEK, J. a VLACH, M.: Začínáme s LabVIEW. BEN - technická literatura, 2008. ISBN: 978-80-7300-245-9.

[2] SUL, S.K.: Control of Electric Machine Drive Systems. Wiley-IEEE Press., February 2011. ISBN: 978-0-4-0-59079-9.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 14.5.2018

Vedoucí práce: doc. Ing. Petr Blaha, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Tato diplomová práce se zabývá vytvořením modelu mechanického systému v jazyce Modelica a jeho použitím v prostředí LabVIEW. Úvod práce se zaměřuje na jazyk Modelica, programovací nástroj LabVIEW a standard Functional Mock-up Interface 2.0, což je standard pro výměnu a souběžnou simulaci dynamických modelů. Na základě Functional Mock-up Interface 2.0 byl vytvořen model elektromechanického systému. Následně byly prozkoumány možnosti exportování modelů do formátu Functional Mock-up Unit, do kterého byl také model uložen. Vygenerovaný model byl vložen do prostředí LabVIEW RealTime, kde byla ověřena funkčnost vytvořeného modelu. Nakonec bylo realizováno propojení instance Functional Mock-up Unit s LabVIEW FPGA tak, aby se CompactRIO jevilo jako HIL model elektromechanického systému.

KLÍČOVÁ SLOVA

Hardware-In-the-Loop, HIL, Functional Mock-up Interface, FMI, Functional Mock-up Unit, FMU, Modelica, Akauzální modelování, LabVIEW, ModelExchange, CompactRIO, FPGA, OpenModelica

ABSTRACT

This diploma thesis deals with creation of elektromechanical model in Modelica language which is subsequently imported into LabVIEW environment. The Modelica language, LabVIEW graphical programming tool and Functional Mock-up Interface 2.0 standard are described in the introduction of this thesis. Functional Mock-up Interface is a tool independent standard witch, defines a standardized interface to ModelExchange and Co-simulation of complex system components. The model of electromechanical system was created based on Functional Mock-up Interface standard. Part of the work focuses on the Functional Mock-up Unit storage possibilities and LabVIEW support to import models of this type. The imported model was simulated and tested in this environment. Finally, the instance of Functional Mock-up Unit was connected with LabVIEW FPGA target for the purpose of model HIL simulation on CompactRIO platform.

KEYWORDS

Hardware-In-the-Loop, HIL, Functional Mock-up Interface, FMI, Functional Mock-up Unit, FMU, Modelica, Acausal modeling, LabVIEW, ModelExchange, CompactRIO, FPGA, OpenModelica

MALÍK, Lukáš. *HIL MODEL ELEKTROMECHANICKÉHO SYSTÉMU*. Brno, 2018, 95 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: doc. Ing. Petr Blaha, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „HIL MODEL ELEKTROMECHANICKÉHO SYSTÉMU“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

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

Brno ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Tato práce se zabývá *Hardware-In-the-Loop* (HIL) simulací vytvořeného fyzikálního (akauzálního) modelu vícehmotového elektromechanického systému na platformě *CompactRIO* (cRIO). Hlavním cílem práce je realizace propojení modelu elektromechanického systému vygenerovaného ve formátu *Functional Mock-up Unit* (FMU) do prostředí LabVIEW RealTime. Dalším bodem práce je realizace částečného propojení naimportovaného FMU s LabVIEW FPGA (Field Programmable Gate Array) provozovaného na platformě CompactRIO.

Vytvořený model elektromechanického systému je definován v jazyce Modelica. Vytvoření systému je realizováno v softwaru OpenModelica a jejím prostředí *OpenModelica Connection Editor* (OMEdit), které je grafickým nástrojem pro modelování fyzikálních systémů z mnoha technických odvětví. Vytvořený fyzikální model je následně z prostředí OpenModelica vyexportován do formátu FMU, jenž je definován standardem *Functional Mock-up Interface* (FMI), kterému se tato práce taktéž věnuje.

Vygenerovaný model elektromechanického systému ve formátu FMU je pomocí doplňku pro LabVIEW naimportován do jeho RealTime prostředí.

Součástí práce je také řešení problémů spojených s vytvářením vlastního modelu a jeho exportem do jednotky FMU, která musí bezpodmínečně splňovat standard FMI verze 2.0.

Cílem této práce je vytváření modelů elektromechanických systémů pomocí fyzikálního modelování namísto používání kauzálního modelování. Použití akauzálních modelů může být v mnoha případech výhodnější. Příkladem může být ověření vlastností a kvality navrhované regulace pro modely, které jsou vystaveny působení svého okolí, které se téměř shoduje s podmínkami reálného prostředí. Často jsou totiž při návrzích regulátorů používány zjednodušené metody modelování soustav, v důsledku kterých je však zanedbávána spousta nemodelovaných vlastností či parametrů. Tyto, ať už cíleně, anebo z jakéhokoliv jiného důvodu opomíjené parametry, se však vždy projeví při nasazení navrženého regulátoru do reálného prostředí. Projevy mohou nastat v řízení samotném nebo jeho kvalitě, často jde však o projevy nežádoucího charakteru. Přínosem akauzálního modelování může být, na rozdíl od kauzálních metod, možnost simulace navržených regulací na modelech s nejrůznějšími podmínkami okolí, parametry modelů či jejich nelinearitami.

Nahradíme-li při tvorbě řídících algoritmů fyzickou část stroje jejím modelem, můžeme tento tento postup testování označit jako HIL simulaci. HIL simulace přináší nové možnosti testování navržených řídicích algoritmů na modelech fyzických zařízení, kterými lze ušetřit čas a materiál vynaložený na vývoj, zvýšit kvalitu konečného produktu a také zvýšit bezpečnost koncového zařízení.

1 Jazyk Modelica a OpenModelica

Modelica je nechráněný, objektově orientovaný jazyk založený na rovnicích, který umožňuje pohodlné fyzikální modelování složitých systémů obsahujících elektrické, elektronické, hydraulické, tepelné, řídicí, silnoproudé, anebo procesně orientované dílčí komponenty [1].

Jazyk Modelica je využíván celou řadou simulačních a vývojových prostředí. Mezi ty nejznámější patří bezesporu Dymola, jejíž licence je ovšem zpoplatněna, nebo například OpenModelica, která je na rozdíl od programu Dymola takzvaným Open Source Software (OSS).

Za účelem vytvoření fyzikálního modelu elektromechanického systému bylo použito právě softwaru OpenModelica. Jazyk Modelica, ze kterého program OpenModelica vychází, je popsán v této kapitole.

1.1 Jazyk Modelica

Model vytvořený v jazyku Modelica je popsán zdrojovým kódem v podobě textové reprezentace. Tento model je uložen v souboru s příponou *.mo*. Je třeba si uvědomit, že Modelica je spíše *modelovací jazyk*, než klasický programovací jazyk. K vytváření modelů je možné použít běžný textový editor, v němž se realizuje popis vytvářeného modelu elektromechanického systému. Postupy vytváření modelu jsou uvedeny v kapitole 1.2.

Každý vytvořený model má jazykem Modelica jasně definovanou strukturu. Příklad základního rozčlenění modelu je vyobrazen v následujícím popisu 1.1.

```
1 model NazevModelu
2   <PromenneModelu>
3 equation
4   <RovniceModelu>
5 end NazevModelu
```

Výpis 1.1: Základní struktura modelu

V textové reprezentaci začíná každý model klíčovým slovem *model*, za nímž následuje uživatelem zvolený název modelu. Na dalších řádcích jsou uváděny definice a deklarace jednotlivých proměnných prostředí. Pod klíčovým slovem *equation* se nachází rovnice popisující model. Pro ukončení definice modelu slouží klíčové slovo *end*, za nímž opět následuje uživatelem zvolený název modelu.

Tak jako každý programovací jazyk má svoji syntaxi, tak i jazyk Modelica má jasně definovanou strukturu, která je však mnohem bohatší, než bylo uvedeno ve

výpisu 1.1. V následující tabulce pak můžeme vidět výpis všech klíčových slov jazyku Modelica.

Tab. 1.1: Přehled klíčových slov jazyku Modelica

algorithm	discrete	false	loop	pure
and	each	final	model	record
annotation	else	flow	not	redeclare
elseif	for	operator	replaceable	block
elsewhen	function	or	return	break
encapsulated	if	outer	stream	class
end	import	output	then	connect
enumeration	impure	package	true	connector
equation	in	parameter	type	constant
expandable	initial	partial	when	constrainedby
extends	inner	protected	while	der
external	input	public	within	

Na rozdíl od běžného programovacího jazyka, kde je použito prosté přiřazování hodnot k jednotlivým proměnným, je každý fyzikální model v jazyce Modelica, popsán především diferenciálními, diferenčními nebo algebraickými rovnicemi. Sada rovnic tak tvoří část popisu z jazyku Modelica. Simulační nástroj pak může, a obvykle také musí, manipulovat s rovnicemi v podobě symbolických proměnných, a to z toho důvodu, aby bylo určeno pořadí jejich provedení a také rozlišeno, která proměnná v rovnici slouží jako vstupní a která jako výstupní argument. Jednoduchý příklad sumátoru může vypadat jako je uvedeno ve výpisu 1.2.

```
1 model Sumator
2   RealInput Cislo1;
3   RealInput Cislo2;
4   RealOutput Soucet;
5 equation
6   Soucet = Cislo1 + Cislo2;
7 end Sumator
```

Výpis 1.2: Příklad modelu sumátoru

Pod názvem modelu *Sumator* se nachází deklarace proměnných, které jsou zakončeny středníky. Středník ukončuje řádek funkčního programu podobně jako v jazyce C/C++. Deklarované nebo definované proměnné jsou vzápětí použity v sekci *equation*. Rovnice uvedená na řádku číslo 6 říká, že součet proměnných *Cislo1*

a *Cislo2* je roven proměnné *Soucet*. Je potřeba zdůraznit, že operátor „rovná se =“ zde neznamená přiřazení hodnoty jako je tomu v jazyce C/C++, ale udává rovnost výrazů uvedených na levé a pravé straně od tohoto operátoru. Uvedenou rovnici bychom mohli také vyjádřit v následující podobě:

$$Cislo1 + Cislo2 - Soucet = 0; \tag{1.1}$$

Každá rovnice je pak zakončena stejně jako jednotlivé parametry středníkem.

Analogicky k jiným jazykům jako je například C/C++ také Modelica disponuje různými dalšími nástroji, kterými lze algoritmus zpřehlednit. Za jeden ze základních nástrojů můžeme považovat komentáře. Jazyk Modelica jich nabízí několik druhů.

Prvním komentářem totožným s jazykem C/C++ je symbol dvojitého lomítka „//“. Tento symbol je překladačem jazyka Modelica ignorován a má význam jednořádkového komentáře.

Jemu obdobný typ, který však slouží naopak k zápisu víceřádkového komentáře, lze zapsat pomocí dvou párů znaků „/*“ a „*/“, které uvozují začátek a konec víceřádkového komentáře. U víceřádkových komentářů je ovšem nebezpečí vzniku tzv. vhnízděného komentáře, který může vést k pozdější chybě. Proto je výrazně lepší používat na víceřádkové komentáře definovanou funkci patřičného vývojového prostředí.

Dalším typem komentáře je dokumentační komentář. Tento druh komentáře lze uvést na konec jakéhokoliv řádku. Obsahuje–li ale příslušný řádek středník, je zapotřebí tento komentář umístit ještě před něj. Dokumentační komentáře slouží pro popis daného prvku, kterým mohou být modely, proměnné či rovnice.

Při exportu modelu do formátu Functional Mock-up Unit se tyto komentáře používají pro dokumentaci. Názorná ukázka komentářů je uvedena ve výpisu 1.3.

```
1 model Sumator "Soucet␣dvojice␣cisel"
2   RealInput Cislo1; // Scitanec 1
3   RealInput Cislo2; // Scitanec 2
4   RealOutput Soucet "Soucet";
5 equation
6   Soucet = Cislo1 + Cislo2 "Rovnice␣sumatoru";
7 end Sumator
```

Výpis 1.3: Komentáře v jazyku Modelica

Modelica disponuje také rozsáhlými knihovnami a knihovními funkcemi. Kromě funkcí se zde nacházejí různé objekty jako např. odpory, kondenzátory, pružiny, ventily, ale i snímače či jiné prvky, které se mezi sebou vzájemně propojují tak, aby vzniklo ucelené schéma vytvářeného modelu.

Vzájemnou interakci dvou komponent v modelu definuje fyzický port, který je taktéž nazýván konektorem [3]. Propojení jednotlivých komponent pomocí konektorů lze zapisovat do textového souboru. Při tvorbě modelů by ovšem textová reprezentace byla méně názorná a čitelná. Za tímto účelem proto vznikla různá grafická prostředí a nástroje, které tuto práci značně ulehčují. Takovým grafickým nástrojem může být software OpenModelica a jeho vývojové prostředí OpenModelica Connection Editor (OMEdit).

1.2 OpenModelica

OpenModelica je volně dostupným nástrojem s otevřeným kódem (OSS) založeným na jazyku Modelica. Umožňuje modelování, simulaci, optimalizaci a analýzu rozsáhlých dynamických systémů. Její dlouhodobý vývoj je podporován neziskovou organizací Open Source Modelica Consorium (OSMC), která spolupracuje se švédskou univerzitou Linköping. OpenModelica nalézá uplatnění také v automobilovém průmyslu či robotice.

Hlavním nástrojem OpenModelica se stal OpenModelica Connection Editor (OMEdit), jenž ulehčuje práci při vývoji nových modelů a návrhu algoritmů pro jejich řízení. Některé vybrané vlastnosti a způsoby ovládání OMEditoru jsou popsány dále.

1.2.1 OpenModelica Connection Editor

OpenModelica Connection Editor je open source grafické a textové uživatelské prostředí, které bylo vyvinuto pro tvorbu, editaci a simulaci modelů definovaných jazykem Modelica.

Grafické prostředí OpenModelica Connection Editor (dále jen OMEdit) se dá pomyslně rozdělit do tří hlavních částí. První částí je horní ovládací lišta s několika funkčními tlačítky a auto-rolovacími nabídkami. Ve druhé části, nalevo od bílého plátna, se nachází stromově uspořádaný výpis všech vložených knihoven do prostředí OMEditoru. Z této knihovny je možné při vytváření modelů vybírat jednotlivé prvky a vkládat je na poslední část uživatelského prostředí, jež je tvořena z počátku prázdným bílým plátnem. V jednotlivých knihovnách umístěných v levé nabídce je možné nalézt mimo jiné i ukázkové příklady zapojení jednoduchých systémů z různých technických oblastí. Podle těchto technických oblastí jsou pak logicky uspořádány jednotlivé knihovní bloky.

Vložení libovolného prvku z knihovny do vlastního modelu se provede metodou „drag and drop“, tedy táhni a pusť. Po vložení komponentu do pracovní oblasti je vždy vyvoláno dialogové okno a uživatel je vyzván k zadání libovolného pojmenování

příslušného prvku. Po naskládání jednotlivých komponentů vytvářeného modelu je nutné jednotlivé bloky propojit. To se provádí pomocí terminálů zobrazených u každého, ať už vstupního, anebo výstupního konektoru příslušného bloku.

Po propojení jednotlivých bloků schématu je nutné nechat kompilátorem OpenModelica Compiler (OMC) zkontrolovat správnost zapojení. OpenModelica Compiler přeloží vytvořený model do jazyka C s tabulkou symbolů, která obsahuje definice tříd, funkcí a proměnných daného modelu. V případě chyby je v dolní části pracovní plochy zobrazen Message Browser, ve kterém je zobrazen výpis jednotlivých chyb a varování. Při úspěšném překladu modelu je do tohoto okna vypsána zpráva o úspěchu překladu a času jeho uskutečnění.

Tvorba modelu je možná také po přepnutí se do textového módu v horní části pracovního plátna. V textovém módu je zobrazen popis modelu v jazyce Modelica, který je možné upravovat či doplňovat o další prvky nebo části algoritmů za použití správné syntaxe. Zde je rovněž možné shlédnout definici a deklaraci jednotlivých proměnných prostředí, vzájemná propojení jednotlivých stavebních bloků modelu a také sadu rovnic, které model popisují.

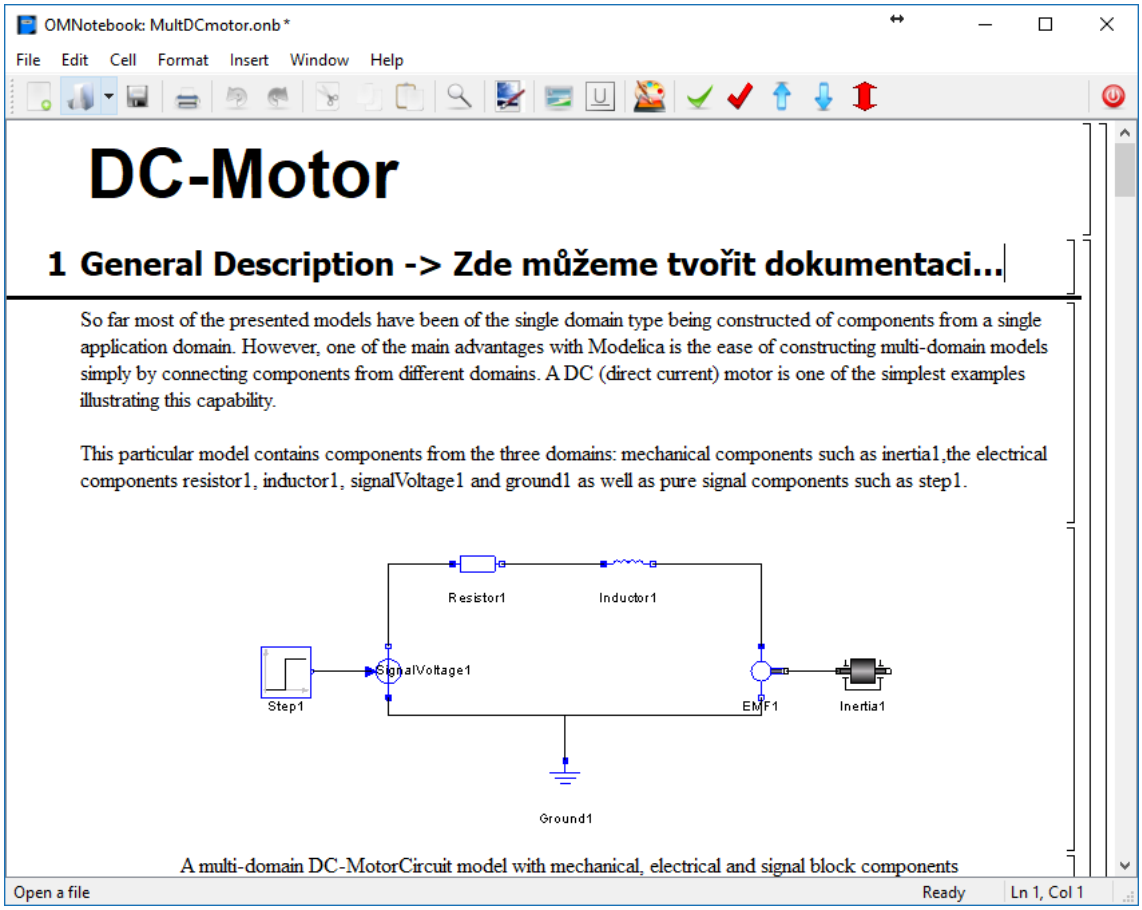
Po vytvoření modelu je nutné provést vestavěným kompilátorem kontrolu modelu. Kompilace samotná nezabere mnoho času, ale v případě syntaktické chyby nemůže být kompilace dokončena a je nuté chybu opravit a kontrolu modelu spustit znovu. Případné syntaktické chyby a jejich detailní popis se zobrazí v nově aktivovaném okně, které je umístěno ve spodní části vývojového prostředí. Po úspěšné kompilaci modelu můžeme spustit simulaci, a to v několika režimech. Na výběr máme klasickou simulaci, simulaci s podporou překládacího ladícího nástroje, simulaci s nástrojem pro ladění algoritmu nebo simulaci s animací. Před spuštěním samotné simulace v kterémkoliv režimu je vhodné nastavit její parametry. Mezi hlavní parametry simulace patří počáteční a konečný čas simulace, dále pak použitý solver pro výpočet simulace a jeho krok. Více parametrů a jejich popis lze nalézt v dokumentaci OMC dostupné na [6].

Kromě kontroly syntaxe modelu je možné také použít funkci Instantiate Model, jež převede vytvořený kód z jazyka Modelica do jazyka C/C++. Tento převedený algoritmus je možné následně využít v jiných vývojových prostředích k dalšímu zpracování.

OpenModelica obsahuje také OpenModelica Shell (OMShell), což je prostředí ve stylu příkazového řádku, které umožňuje analyzovat a interpretovat příkazy jazyku Modelica za účelem výpočtů, simulace a vykreslování vytvořených modelů.

Další částí OpenModelica je také OpenModelica Notebook (OMNotebook). Jedná se o editor podobný programu Mathematica, který implementuje tvorbu WYSIWIG interaktivních knih. Tyto knihy mohou být používány jako učební pomůcky pro výuku programování v jazyce Modelica. Další využití nalezne OMNotebook při tvorbě

tutoriálů či dokumentací postupů v Modelica. Tento interaktivní zápisník tak sjednocuje programování, jeho dokumentaci a vizualizaci dosažených výsledků. Použití OMNotebooku zachycuje obrázek 1.2.1.



Obr. 1.1: Příklad použití dokumentačního nástroje OMNotebook

Mezi další nástroje, kterými OpenModelica disponuje, patří také OpenModelica Python Interface (OMPython) a OpenModelica Development Tooling Eclipse Plugin (MDT). OMPython je rozhraní vyvíjené společností Python, pro jeho stejnojmenný programovací jazyk, které umožňuje uživateli přistupovat a využívat modelovací a simulační schopnosti OpenModelica [4]. Druhý nástroj OpenModelica Development Tooling Eclipse Plugin, jak už jeho samotný název napovídá, do sebe integruje kompilátor OpenModelica, a poskytuje tak prostředí pro práci s vývojovými projekty Modelica a MetaModelica. Tento plugin je určen především vývojářům nástrojů, nikoliv však pro aplikaci a vývoj modelů popsaných jazykem Modelica [5]. Proto jsou tyto dílčí nástroje OpenModelica z hlediska cílů této práce nezajímavé a nebudou již dále rozváděny.

OpenModelica, kromě řady jiných funkcí, umožňuje exportovat a importovat modely formátu Functional Mock-up Unit (FMU), které budou spolu se standardem FMI 2.0 popsány dále v kapitole 2 a 4. Samotný proces exportu nebo importu modelu z/do formátu FMU se provede výběrem příslušné položky z autorolovací nabídky v horní paletě menu, která je označena zkratkou FMI.

Když srovnáme výsledek tvorby modelu stejnosměrného motoru v OpenModelica Connection Editoru (obrázek 1.3) například s hojně používaným Simulinkem (obrázek 1.4), vidíme mezi technikou tvorby a jejich výsledkem mnohé odlišnosti. Rozdíly jsou způsobeny tím, že OpenModelica na rozdíl od MATLAB-Simulink umožňuje kromě kauzálního modelování také fyzikální modelování, které je v některé literatuře označováno jako akauzální.

1.3 Fyzikální modelování

Největší výhodou metody fyzikálního modelování je, že není nutné mít fyzický přístup k zařízení, pro které bude řízení navrhováno a pro které bude později použito. Pro to, abychom nebyli omezováni použitými technologiemi na daném zařízení, je vhodné vytvořit modely, na kterých bude odzkoušeno základní nastavení regulačních smyček bez toho, aby zařízení muselo být reálně vyrobeno. Účelem fyzikálního modelování je navrhnout algoritmy řízení pro zařízení již ve fázi projektování. Velkým přínosem je úspora času a peněžních prostředků vynaložených na tvorbu prototypu zařízení.

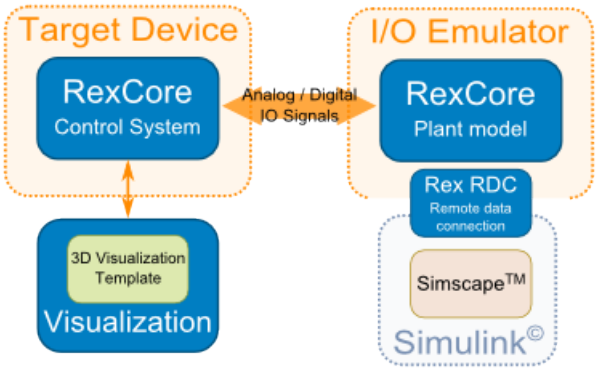
Vzhledem k tomu, že při modelování elektromechanických systémů dochází téměř vždy ke zjednodušování, obsahuje pak model oproti reálnému prostředí značné nepřesnosti. Některé parametry však mohou být zanedbávány zcela účelově, a to např. z důvodu složitosti modelu, jeho výpočetní náročnosti či samotné možnosti realizace modelu.

V první fázi návrhu řídicích algoritmů se používá Model-In-the-Loop (MIL) simulace, jejímž cílem je návrh regulace pro modely elektromechanických soustav. V tomto případě se však jedná pouze o idealizovaný návrh, který nebere v potaz vzájemné propojení hardwarového a softwarového prostředí, výpočetní výkon ani rychlost komunikačních sběrnic. Návrh regulace může být proveden např. v programovém prostředí MATLAB-Simulink. (Odstavec čerpá z [7]).

Další částí vývoje je proces, ve kterém jsou testovány algoritmy řízení navržené v první fázi, a to na reálné soustavě pomocí Hard Real Time zařízení. HRT může být realizováno například pomocí vhodné I/O karty, která je vložena do počítače. Tato karta poskytne deterministické, a tedy i skutečné RealTime spojení řídicího algoritmu s fyzickým zařízením či simulátorem. Při práci v Simulinku je možné použít toolboxy pro překlad programu do jazyka C/C++. Následnou kompilací tohoto

programu do cílového hardwaru je možné provést testování navrženého řídicího algoritmu v reálném zařízení. (Odstavec čerpá z [7]).

HIL simulace je jedním z posledních kroků vývoje návrhu řídicích algoritmů. Po něm následuje už pouze připojení k reálnému zařízení. Při HIL simulaci je chování fyzického zařízení simulováno pomocí softwarových a hardwarových modelů. Reálné komponenty zařízení (jeho fyzické části) jsou připojeny ke komunikačnímu rozhraní do simulátoru, který reprodukuje chování fyzického zařízení v reálném čase. Názorný příklad HIL simulace je ukázán na obrázku 1.3, který byl převzat z [9]. (Odstavec čerpá z [7]).



Obr. 1.2: Příklad realizace Hardware-In-the-Loop simulace [9]

Na obrázku 1.3 je schematicky znázorněna HIL simulace. V pravé polovině obrázku se nachází počítač, na němž je spuštěn MATLAB-Simulink s toolboxem Simscape. PC komunikuje pomocí RDC s emulátorem I/O, který je připojen k reálnému řídicímu systému pomocí fyzických IO.

Použití HIL simulace s sebou přináší mnoho výhod. Mezi hlavní patří následující:

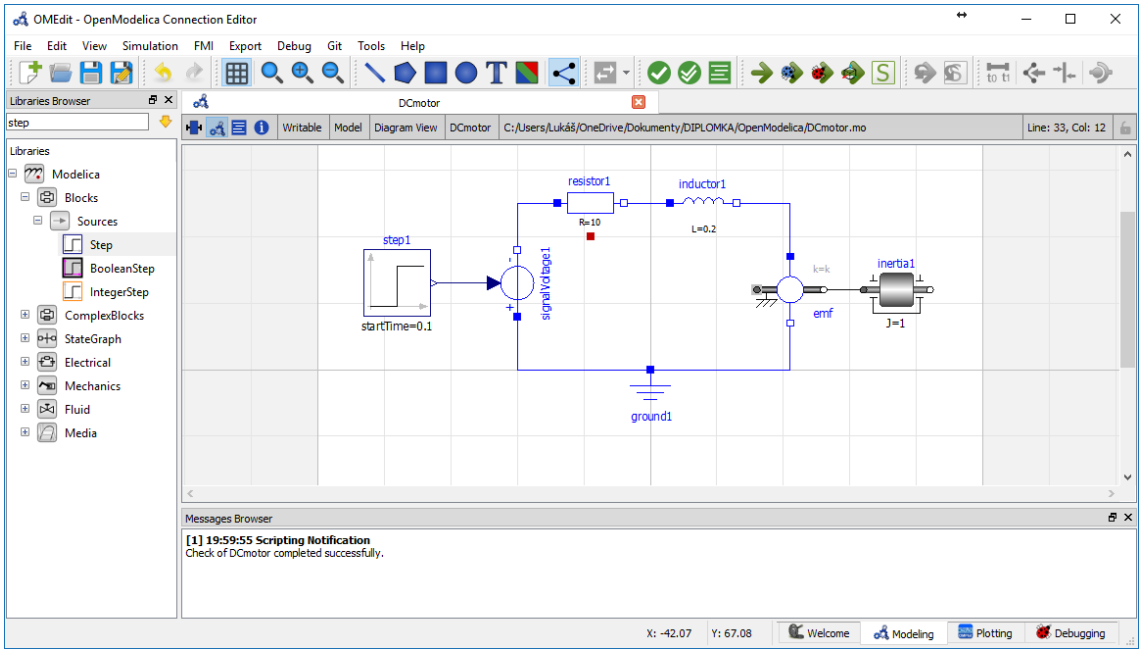
- Řídicí a regulační funkce mohou být testovány v počátcích vývoje ještě před realizací prototypu. Elektronika proto dosahuje vysoké úrovně vyspělosti a poskytuje dobrý základ pro další vývojové fáze.
- Jejím využitím se dá dosáhnout snížení času potřebného pro vývoj zařízení.
- Nákladné pokusy prováděné v terénu a v nebezpečných podmínkách mohou být částečně nahrazeny laboratorními nebo počítačovými experimenty.
- V modelu je možné nastavit extrémní nebo neobvyklé podmínky (typicky velmi nízké teploty), kterým je zařízení vystaveno.
- Systematické testování chyb a chybových hlášení, které by za podmínek přímého připojení řídicího algoritmu k reálnému zařízení vedly k devastujícím nebo katastrofickým výsledkům.

- Je možné snadno testovat řídicí systém, jak reaguje na poruchy systému, které by se na reálné soustavě obtížně realizovaly.
- Experimenty prováděné v HIL systému mohou být přesně reprodukovány a automaticky opakovány podle potřeby. (Převzato z [7].)

Kromě MIL a HIL simulace existuje také Software-In-the-Loop (SIL) a Processor-In-the-Loop (PIL) simulace, které tvoří mezistupně vývoje mezi MIL a HIL simulacemi. Za závěrečný stupeň vývoje zařízení je pak považován stav, kdy je provedena aplikace řídicího systému na reálném zařízení.

Při SIL simulaci je vygenerován kód jazyka C z řídicí části. Vygenerovaný kód je následně spuštěn a testuje se, jestli se nějakým způsobem neprojeví jiná datová reprezentace, která je použita v jazyce C. Funkční bloky řídicího systému jsou spouštěny obvykle specializovaným jádrem některého z komerčně prodávaných SIL systémů. (Odstavec čerpá z [9]).

PIL simulace stejně jako SIL simuluje matematický model v reálném čase. u PIL simulace, na rozdíl od SIL simulace, je řídicí systém provozován na cílové hardwarové platformě. Nejsou však použity žádné I/O karty, čidla ani aktuátory. Data mezi modelem a řídicím systémem se přenášejí pomocí komunikační sběrnice. PIL simulace slouží především k ověření výpočetního výkonu řídicího hardwaru, testování kritických situací a nejrůznějších podmínek. (Odstavec čerpá z [9]).



Obr. 1.3: Model stejnosměrného motoru v prostředí OpenModelica – grafický editor

Při tvorbě elektromechanických modelů máme na výběr ze dvou metod. První

metoda používá tzv. kauzální a druhá akauzální postupy modelování částí elektromechanických modelů.

Nástroje, které jsou založeny na metodě vytváření fyzikálního modelování, jsou mnohdy označovány jako akauzální nebo deklarativní. Jejím protějšem jsou blokově orientované nástroje, které pracují s hierarchicky propojenými bloky. V propojeních mezi jednotlivými bloky „tečou“ signály, které přenášejí hodnoty jednotlivých proměnných od výstupu jednoho bloku ke vstupům dalších bloků. V blocích dochází ke zpracování vstupních informací na výstupní. Takovéto propojení bloků proto odráží spíše postup výpočtu (algoritmus), namísto struktury modelované reality. U značně složitých systémů se díky tomuto přístupu pod strukturou výpočtu pomalu ztrácí fyzikální význam modelovaného systému. (Odstavec čerpá z [8]).

Proto se pro modelování složitějších a komplexních systémů používají nástroje, v nichž jsou jednotlivé části modelů definovány přímo v podobě rovnic namísto algoritmů řešení těchto rovnic. V tomto kontextu se mluví o tzv. deklarativním (akauzálním) zápisu modelů, na rozdíl od kauzálního zápisu v blokově orientovaných jazycích. V kauzálním, neboli blokově orientovaném zápisu modelu, musíme např. pomocí grafického propojení jednotlivých knihovních bloků či prvků vyjádřit kauzální popis způsobu výpočtu jednotlivých proměnných modelu. U akauzálního modelování se naopak předem neví, který pin příslušného bloku bude vstupní a který bude použit jako výstupní. Akauzální přístup umožňují také nové simulinkové knihovny Simscape a návazné doménové knihovny SimElectronics, SimHydraulics, SimMechanics aj. (Odstavec čerpá z [8]).

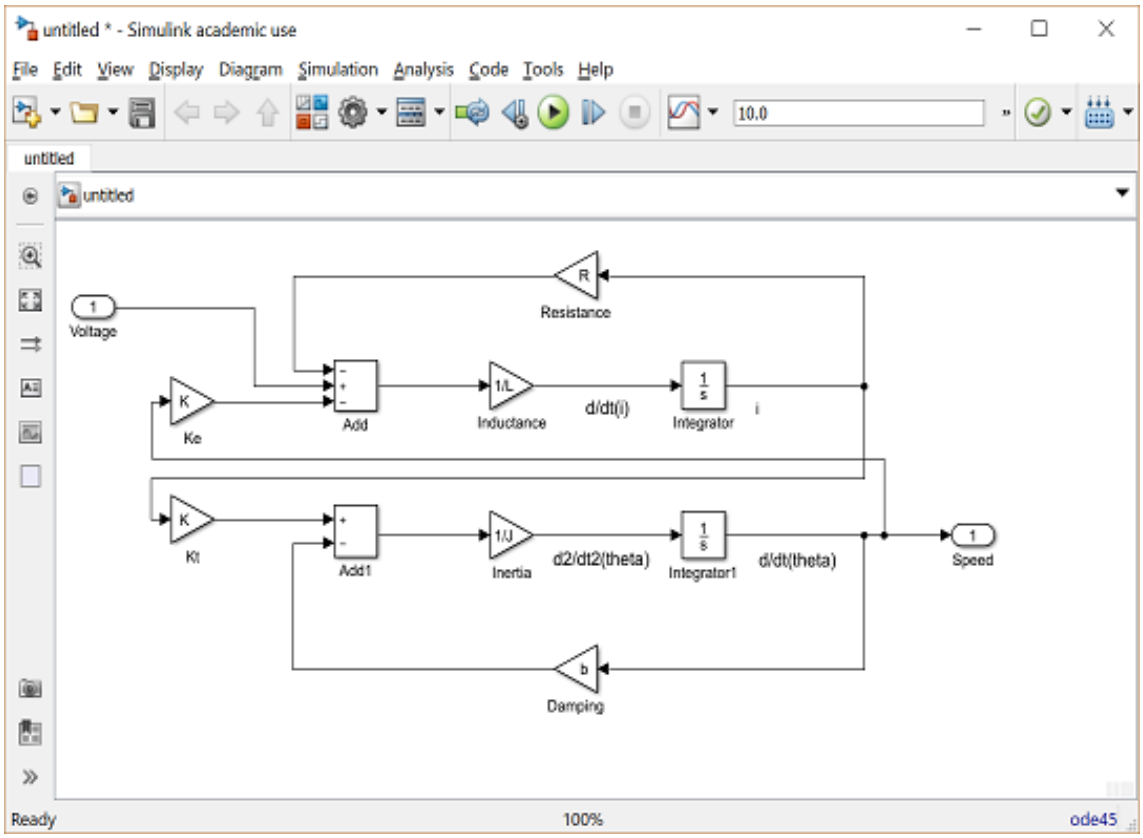
V této práci bylo k vytvoření modelu elektromechanického systému v jazyce Modelica použito vývojového nástroje OpenModelica. Realizace jednoduchého modelu stejnosměrného motoru metodou akauzálního (fyzikálního) modelování v prostředí OpenModelica je zachycena na obrázku 1.3.

Naproti tomu, model stejnosměrného motoru vytvořený metodou kauzálního modelování v prostředí MATLAB-Simulink (viz obrázek 1.4) nemusí na první pohled jasně vystihovat svoji fyzikální podstatu.

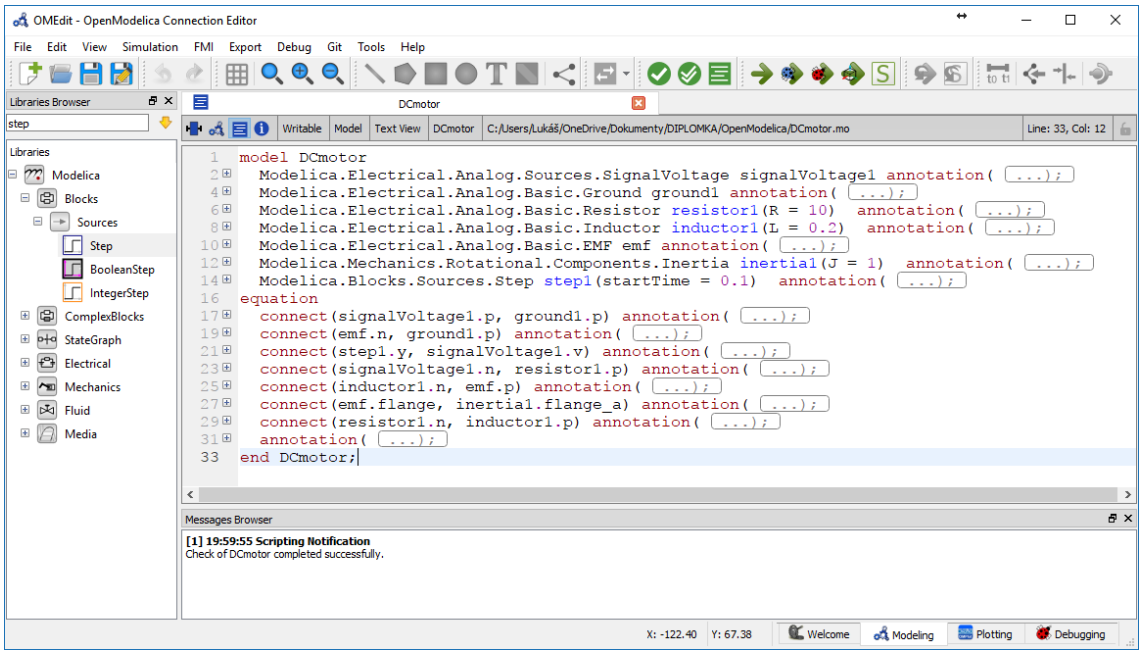
Z obrázků 1.3 a 1.4 je možné rozpoznat patrný rozdíl mezi oběma metodami modelování elektromechanických systémů. V případě použití OpenModelica a OMEditoru je na první pohled zcela zřejmý fyzikální význam, který by se mohl v případě Simulinku u složitějšího zařízení lehce vytratit.

V textovém režimu OpenModelica, jenž je na obrázku 1.5, můžeme pracovat přímo s rovnicemi – obdobně jako v Simulinku. Je tedy možné zvolit pouze jednu z nabízených variant OpenModelica, anebo při modelování kombinovat oba způsoby editace.

Při vytváření modelu je nutné dodržovat pravidla standardu FMI 2.0 (Functional Mock-up Interface). Tento standard je popsán v kapitole 2.



Obr. 1.4: Model stejnosměrného motoru v prostředí MATLAB-Simulink [10]



Obr. 1.5: Model stejnosměrného motoru v prostředí OpenModelica – textový editor

2 Functional Mock-up Interface 2.0

Functional Mock-up Interface (dále jen FMI) je otevřený standard pro výměnu simulačních modelů mezi různými nástroji ve standardizovaném formátu [11]. FMI umožňuje libovolnému modelovacímu nástroji generovat kód v jazyce C nebo binární soubor, jenž představuje model dynamického systému, který může být zcela integrován do jiného modelu nebo simulačního nástroje [12]. Funkce FMI jsou volány simulačním nástrojem nebo prostředím za účelem vytvoření jedné nebo více instancí Functional Mock-up Unit. Ty mohou být v simulačním nástroji simulovány odděleně, nebo dohromady s jinými modely. Další obsah této kapitoly vychází především z definice standardu FMI uvedeném v [13].

```
// Structure of zip file of an FMU
modelDescription.xml          // Description of FMU (required file)
model.png                     // Optional image file of FMU icon
documentation                 // Optional directory containing the FMU documentation
   index.html                 // Entry point of the documentation
   <other documentation files>
sources                       // Optional directory containing all C sources
   // all needed C sources and C header files to compile and link the FMU
   // with exception of: fmi2TypesPlatform.h , fmi2FunctionTypes.h and fmi2Functions.h
   // The files to be compiled (but not the files included from these files)
   // have to be reported in the xml-file under the structure
   // <ModelExchange><SourceFiles> … and <CoSimulation><SourceFiles>
binaries                      // Optional directory containing the binaries
   win32                      // Optional binaries for 32-bit Windows
      <modelIdentifier>.dll   // DLL of the FMI implementation
                              // (build with option "MT" to include run-time environment)
      <other DLLs>            // The DLL can include other DLLs
      // Optional object Libraries for a particular compiler
      VisualStudio8          // Binaries for 32-bit Windows generated with
                             // Microsoft Visual Studio 8 (2005)
         <modelIdentifier>.lib   // Binary libraries
      gcc3.1                 // Binaries for gcc 3.1.
        ...
   win64     // Optional binaries for 64-bit Windows
      ...
   linux32   // Optional binaries for 32-bit Linux
      <modelIdentifier>.so   // Shared library of the FMI implementation
      ...
   linux64   // Optional binaries for 64-bit Linux
      ...
resources   // Optional resources needed by the FMU
   < data in FMU specific files which will be read during initialization;
     also more folders can be added under resources (tool/model specific).
     In order for the FMU to access these resource files, the resource directory
     must be available in unzipped form and the absolute path to this directory
     must be reported via argument "fmuResourceLocation" via fmi2Instantiate.
   >
```

Obr. 2.1: Adresářová hierarchie FMU [13]

Modely, které jsou uloženy v souladu se standardem FMI, jsou označovány jako

Functional Mock-up Unit zkráceně FMU. Tento výstupní formát modelu je zabalen v pomyslném zip archivu s příponou *.fmu, v němž se nachází popis modelu, zdrojové kódy a jeho dokumentace. V tomto zip souboru se nachází, kromě zdrojových kódů modelu uložených v jazyce C, také definice binárních souborů v podobě Dynamic Link Library pro *Operační Systém* (OS) Windows. Ty bývají častěji označovány jako DLL knihovny a mají také stejnojmennou souborovou příponu *.dll. Existují ovšem také sdílené knihovny objektů určené pro Operační Systém Linux. Tyto knihovny jsou označeny, podobně jako v případě DLL knihoven, souborovou příponou *.so. Instance FMU ve své adresářové struktuře, která je zobrazena na obrázku 2.1, obsahuje mimo jiné hlavně soubor s příponou *.xml, ve kterém se nachází popis modelu.

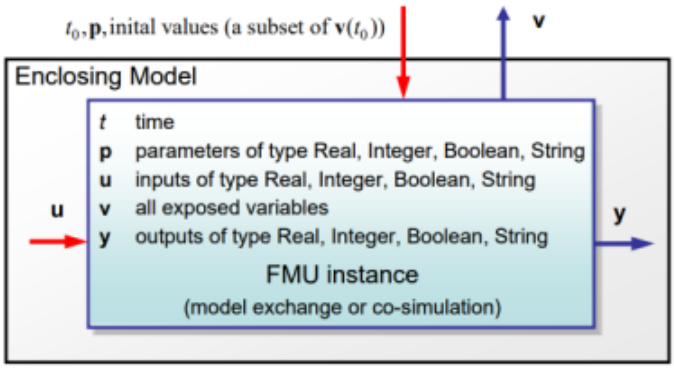
Standard FMI definuje v zásadě dvě rozhraní (formy) popisu totožného fyzikálního (akauzálního) modelu. První forma popisu modelu se nazývá ModelExchange a druhá Co-simulation. Do obou těchto rozhraní mohou být modely fyzikálních systémů vyexportovány z libovolného modelovacího nástroje, který byl použit pro jejich vymodelování, avšak za předpokladu, že tento nástroj umožňuje exportování modelu v souladu se standardem FMI verze 2.0.

Nástroje, které podporují standard FMI, jsou uvedeny na oficiálních webových stránkách: http://fmi-standard.org/tools/. Tyto webové stránky jsou provozovány skupinou společností, institutů a univerzit, které jsou organizovány pod záštitou Modelica Association a jejich projektu Functional Mock-up Interface. Mezi nástroje, ve kterých je možné vytvářet fyzikální modely v jazyce Modelica a které současně podporují standard FMI 2.0, patří komerčně dostupná Dymola. Dále je zde možné nalézt mnoho Open Source Software, které jsou legálně dostupné, jako např. JModelica.org nebo OpenModelica. OpenModelica, jež byla pro modelování fyzikálních systémů a k jejich následnému exportu v této práci použita, umožňuje exportování a importování modelů ve formě ModelExchange. Dále je možné použít tento nástroj pro Co-simulační mód, ale pouze za předpokladu, že vygenerované FMU bude použito pouze jako Slave. Implementace Master funkce je pro OpenModelicu pouze plánována a prozatím nezrealizována.

2.1 FMI ModelExchange a Co-simulation

Jednotka FMU může mít buď vlastního řešitele (FMI pro Co-simulation), nebo požadovat simulační prostředí k provádění numerické integrace (FMI pro ModelExchange). Schematické znázornění jednotky FMU je ukázáno na obrázku 2.2.

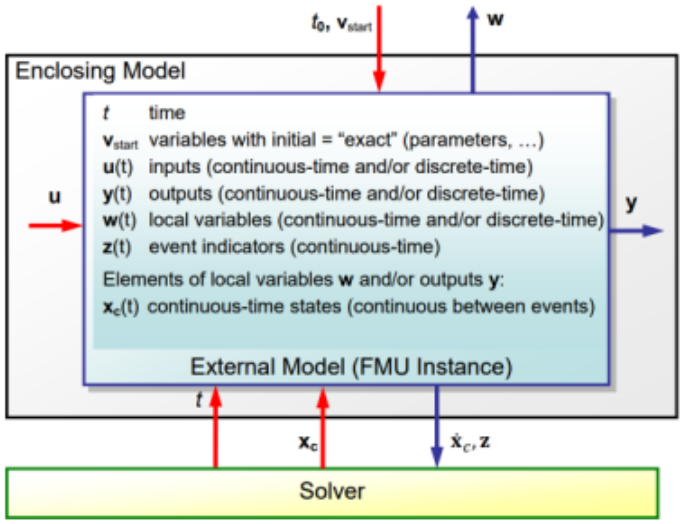
Na níže uvedených obrázcích 2.2, 2.3 a 2.4 se nachází modře a červeně vyznačené šipky, které znázorňují tok dat mezi okolním prostředím a FMU. Modré šipky



Obr. 2.2: Schematické znázornění FMU [13]

znázorňují informace poskytované FMU, červené šipky naopak informace vstupující do FMU.

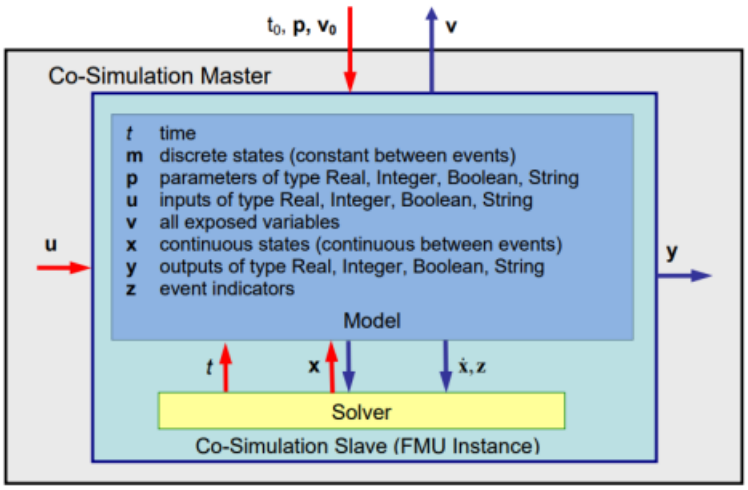
Rozhraní FMI pro ModelExchange (obrázek 2.3) definuje rozhraní k modelu dynamického systému popsaného diferenciálními, algebraickými a diskrétními časovými rovnicemi a poskytuje rozhraní pro vyhodnocování těchto rovnic v různých simulačních prostředích podle potřeby, stejně jako v integrovaných řídicích systémech s explicitními nebo implicitními integrátory a pevnou nebo variabilní velikostí kroku. (Odstavec vychází z [13]).



Obr. 2.3: Tok dat v FMI pro ModelExchange [13]

Rozhraní FMI pro Co-simulaci (obrázek 2.5) je určeno jak pro propojení jednotlivých simulačních nástrojů (tzv. simulátorová spojka nebo spojka s nástroji), tak pro spojování s modely subsystémů, které byly vyexportovány prostřednictvím

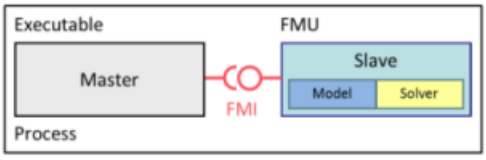
jejich simulátorů spolu s jejich solvery, a to ve formě spustitelného programu.



Obr. 2.4: Tok dat v FMI pro Co-simulation [13]

Cílem Co-Simulace je výpočet řešení časově závislých spojených systémů sestávajících z podsystémů, které jsou v čase spojité (komponenty modelu, které jsou popsány diferenciálními rovnicemi) nebo časově diskrétní (modelové komponenty, které jsou popsány diferenčními rovnicemi, např. diskrétní regulátory). V blokové reprezentaci spojitého systému jsou subsystémy reprezentovány bloky s (interními) stavovými proměnnými $x(t)$, které jsou připojeny k jiným subsystémům (blokům) pomocí vstupů subsystému $u(t)$ a výstupů subsystému $y(t)$.

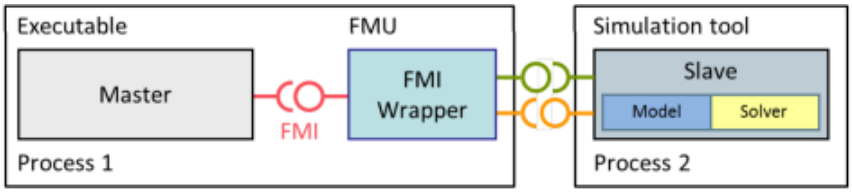
Na obrázku 2.5 se nachází FMU, které obsahuje popis modelu a solver v jediném spustitelném kódu. Toto FMU je pomocí FMI spojeno s hlavním procesem, který je označován jako master. Chování nebo podřízenost hlavnímu procesu je pak označována jako funkce slave.



Obr. 2.5: FMI pro Co-simulaci - FMU součástí spustitelného procesu [13]

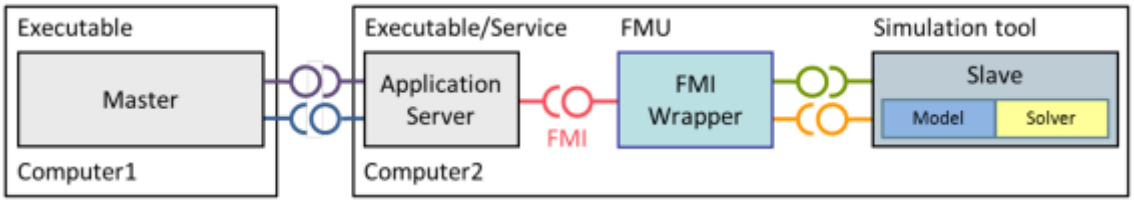
V případě, kdy je FMI použito ke spojení procesu a externího simulačního nástroje, vykonává implementace FMU funkce FMI (FMI Wrapper) jejich voláním v API (Application Programming Interface). API funkce jsou poskytovány pomocí simulačního nástroje např. COBRA API. Jak FMU, tak i simulační nástroj jsou

umístěny na jednom výpočetním zařízení a přítomnost simulačního nástroje je nezbytná ke spuštění simulace (viz obrázek 2.6).



Obr. 2.6: FMI pro Co-simulaci - FMU poskytuje komunikační rozhraní [13]

Nejobecnější formou Co-simulace (obrázek 2.7), kterou FMI 2.0 standard definuje, je případ použití dvou různých výpočetních zařízení. Použité výpočetní prostředky smí používat také různé platformy OS. Spuštěný proces je prováděn operačním systémem jednoho počítače (master), zatímco na druhém počítači (např. server nebo cluster) se stejným nebo i odlišným OS běží simulační nástroj.



Obr. 2.7: FMI pro Co-simulaci - odlišné platformy OS [13]

Vzájemné propojení obou zařízení zajišťuje síťový komunikační protokol, např. nejznámější TCP/IP, který zajišťuje spolehlivé spojení s aplikačním serverem. Navazující datové propojení aplikačního serveru a instance FMU poskytuje FMI wrapper, který používá funkce definované FMI standardem. Tento wrapper je následně propojen se simulačním nástrojem obdobně jako v předchozím případě.

2.2 Matematický popis modelu

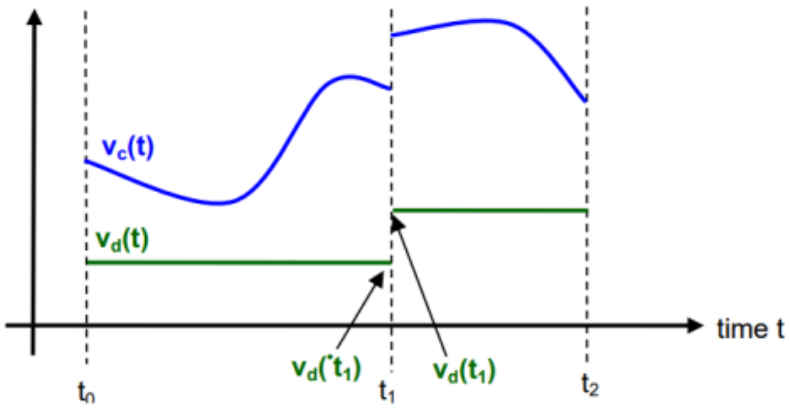
Standard FMI, jehož cílem je poskytnout jednotné prostředí pro výměnu modelů a jejich simulaci, byl navržen pro numerické řešení systému diferenciálních, algebraických a diferenčních rovnic. Tento systém bývá označován jako hybridní systém diferenciálních rovnic. V literatuře jej můžeme nalézt pod zkratkou ODE nebo hybrid ODE (Ordinary Differential Equations).

Hybridní systém diferenciálních rovnic podporovaný FMI je po částech spojitý, kde k nespojitostem může docházet pouze za předpokladu, že $t_i < t_{i+1}$. Splnění této

podmínky je v definici standardu FMI 2.0 [13] označováno jako časy událostí. Kromě toho jsou pro hybridní systém rovnic definovány následující stavové vektory:

- $x_c(t)$ je vektor spojitých stavů
- $x_d(t)$ je vnitřní vektor diskrétních stavů

Hodnoty vektoru spojitých stavů jsou spojité i mezi dvojicí po sobě jdoucích událostí. Naopak u vektoru diskrétních stavů jsou hodnoty mezi dvojicí po sobě jdoucích událostí konstantní. Příklad průběhu stavových vektorů v čase je zobrazen na 2.8. Oproti FMI 1.0 došlo ke změně pojmenování stavových vektorů.



Obr. 2.8: Průběh vektorů stavů v po částech spojitém systému [13]

V časech t_0, t_1 a t_2 se nacházejí výskyty událostí. Z obrázku průběhu vektorů stavů 2.8 je zřejmé, že v okamžiku nespojitosti t_i nabývají stavové vektory $x_c(t_i)$ a $x_d(t_i)$ dvou hodnot $v_c^-(t_i)$ a $v_c^+(t_i)$ (obdobně u diskrétního vektoru). Řešením vzniklé situace je definice limitních hodnot v čase okamžiku, a to následovně:

- $v_c^-(t_i)$ limitní hodnota zleva spojitého stavu
- $v_d^-(t_i)$ limitní hodnota zleva diskrétního stavu
- $v_c^+(t_i)$ limitní hodnota zprava spojitého stavu
- $v_d^+(t_i)$ limitní hodnota zprava diskrétního stavu

Dále smí model obsahovat další proměnné, mezi které patří vstupy, výstupy, interní proměnné a parametry, které jsou po dobu simulace konstantní. Zbylé proměnné smí po dobu simulace měnit svoje hodnoty.

2.3 Funkce modelu

Funkce FMI jsou volány prostřednictvím instancí FMU, a to za účelem vytvoření komunikačního kanálu, který slouží k přenosu dat mezi spustitelným procesem (masterem) a FMU, anebo masterem či aplikačním serverem a FMI Wrapperem.

Všechny funkce, které FMI poskytuje a které jsou spolu s funkcí *fmi2Status* volány, mají návratovou hodnotu typu enumeration. Definice funkce fmi2Status je následující:

```
1 typedef enum {  fmi2OK,
2                 fmi2Warning,
3                 fmi2Discard,
4                 fmi2Error,
5                 fmi2Fatal,
6                 fmi2Pending } fmi2Status;
```

Výpis 2.1: Definice funkce fmi2Status

Význam jednotlivých hodnot, kterých může výčtová proměnná *fmi2status* nabývat, je:

- fmi2OK – vše v pořádku
- fmi2Warning – vše není zcela v pořádku, avšak simulace může pokračovat, byla zavolána funkce „logger“ k vytvoření zprávy s popisem problému
- fmi2Discard – ModelExchange - snížit periodu vzorkování, Co-simulation - slave neposkytnul masterem požadované informace
- fmi2Error – nastala chyba, simulace nemůže dále pokračovat, je potřeba zavolat funkci „fmi2FreeInstance“ nebo „fmi2Reset“, byla zavolána funkce „logger“
- fmi2Fatal – výpočty modelu jsou nenávratně poškozeny pro všechny instance FMU, byla zavolána funkce „logger“
- fmi2Pending – vrácena pouze v případě Co-simulačního rozhraní a za podmínky, že slave provede funkci asynchronně = slave začne s výpočtem, ale okamžitě vrací hodnotu, master musí zavolat funkci „ fmi2GetStatus (. . . , fmi2DoStepStatus)“ – jedná se o novou návratovou hodnotu oproti FMI 1.0.

K vytvoření instance modelu slouží funkce *„fmi2Instantiate“*. Ke smazání jedné instance modelu slouží funkce *„fmi2FreeInstance“*. Další funkce pro inicializaci a nastavení modelu jsou *„fmi2SetDebugLogging“*, *„fmi2SetupExperiment“*, *„fmi2SetTime“*, *„mi2GetVariableType“*, *„fmi2SetVariableType“* a *„fmi2Reset“*. Po dokončení simulace je zapotřebí ji řádně ukončit, a to funkcí *„fmi2Terminate“*.

Detailní popis funkcí s příklady jejich použití lze najít v definici standardu [13]. Zde je také možné nalézt přehlednou tabulku výpisu všech dostupných funkcí pro každý typ simulace.

2.4 Popis modelu

Soubor XML (Extensible Markup Language), jenž je generován modelovým prostředím, obsahuje standardizovanou definici všech proměnných uvedených v XML

souboru. Díky tomu je možné spustit kód v jazyce C uvnitř systému, do kterého je jazyk XML integrován, bez nutnosti opětovné definice proměnných prostředí. Tento způsob šetří čas, paměťový prostor a výpočetní náročnost při simulacích.

Alternativou k této standardizované definici je uložení těchto informací do kódu jazyka C a umožnění přístupu k nim prostřednictvím volání funkcí. To ovšem není praktické ani pro vestavěné systémy, ani pro obsáhlé modely.

Variabilní definice je složitou datovou strukturou, a nástroje by proto měly mít určitou volnost ve způsobech, kterými ji lze reprezentovat ve svých programech. Tento způsob definice dovoluje nástroji ukládat a přistupovat k modelovým proměnným v programovacím jazyce simulačního prostředí jako je C++, C#, Java nebo Python. Tímto způsobem je možné se vyhnout paměťově neefektivním režiím, které by bylo nutno použít v případě standardních přístupových funkcí. [13]

Ke čtení XML souborů s nejrůznějšími datovými strukturami existuje mnoho bezplatných, ale také komerčních knihoven určených různým programovacím jazykům. Mezi výhody souborů XML formátu patří zejména jejich čitelnost pro osobu i stroj. Díky jazyku XML, který vychází z otevřeného standardu, je možné standardizovaný popis modelu po jeho otevření v kterémkoliv z mnoha volně dostupných nástrojů přečíst. Některým z těchto nástrojů pak může být například internetový prohlížeč, editor textových dokumentů nebo komerční WordPad. Ne však každý nástroj, který dokáže XML soubor zobrazit, musí být nejvhodnější pro jeho interpretaci. Ukázkový příklad výpisu XML souboru lze zhlédnout na obrázku 2.9.

Při pohledu na strukturu XML souboru 2.9 si můžeme všimnout několika položek *TypeDefinitions*, *ModelVariables* a dalších, které jsou definovány souborem *fmiModelDescription.xsd* standardu FMI verze 2.0.

Níže uvedené elementy popisu modelu lze najít pod položkou *fmiModelDescription* v příslušném XML souboru. Výčet všech dostupných elementů určených k popisu modelu je následující:

- ModelExchange
- CoSimulation
- UnitDefinitions
- TypeDefinitions
- LogCategories
- DefaultExperiment
- VendorAnnotations
- ModelVariables
- ModelStructure

Z výše uvedeného seznamu parametrů XML souboru spadá pouze jedna položka do skupiny takzvaných povinných parametrů, kterými musí být každý XML vybaven. Jedná se o jednu z položek *ModelExchange* nebo *Cosimulation*, které jsou

podkategorií popisu modelu *fmiModelDescription*.

Mezi obecné položky z kategorie *fmiModelDescription* patří atributy *fmiVersion*, *modelName*, *guid*, *description*, *author*, *version*, *copyright*, *license*, *generationTool*, *generationDateAndTime*, *variableNamingConvention* a *numberOfEventIndicators*. Význam mnohých těchto parametrů je na první pohled zřejmý. Za zmínku stojí snad jen atribut *guid*, tedy *„Globally Unique IDentifier“*, což je řetězec, který je používán ke kontrole kompatibility XML souboru s funkcemi jazyka C. Zápis do toho řetězce je prováděn automaticky, vždy při generaci XML. Význam tohoto parametru můžeme rovněž chápat jako jedinečný otisk prstu, který má v jednom parametru vypovědět o informacích obsažených uvnitř jednotky FMU.

Oproti FMI verze 1.0 byl atribut *numberOfContinuousStates* z verze FMI 2.0 odstraněn s odůvodněním, že informace v něm obsažené mohou být dedukovány ze zbývajících dat obsažených v XML souboru.

Pokud je v popisu modelu přítomen element *ModelExchange*, znamá to, že FMU je založeno na módu *ModelExchange*. Mimo jiné FMU také obsahuje model nebo komunikační rozhraní nástroje, který poskytuje prostředí pro vytváření modelu a jeho

```
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription numberOfEventIndicators="1" variableNamingConvention="structured"
generationDateAndTime="2017-12-05T15:15:41Z" generationTool="OpenModelica Compiler
v1.12.0 (32-bit)" description="" guid="{bc43967f-03ff-40b4-900e-d909675ee25b}"
modelName="DCmotor_v3" fmiVersion="2.0">
    <ModelExchange modelIdentifier="DCmotor_v3"> </ModelExchange>
  - <TypeDefinitions>
      - <SimpleType name="Modelica.Blocks.Types.Init">
          + <Enumeration>
        </SimpleType>
    </TypeDefinitions>
  + <LogCategories>
    <DefaultExperiment tolerance="1e-006" stopTime="1.0" startTime="0.0"/>
  + <ModelVariables>
  - <ModelStructure>
      - <Outputs>
            <Unknown dependenciesKind="dependent" dependencies="1" index="14"/>
            <Unknown dependenciesKind="dependent" dependencies="2" index="15"/>
        </Outputs>
      - <Derivatives>
            <Unknown dependenciesKind="dependent dependent" dependencies="1 2"
                index="3"/>
            <Unknown dependenciesKind="dependent" dependencies="1" index="4"/>
        </Derivatives>
    </ModelStructure>
</fmiModelDescription>
```

Obr. 2.9: Příklad XML souboru z prostředí webového prohlížeče

simulaci. Naproti tomu element *CoSimulation*, pokud je definován, obsahuje jak samotný model, tak i simulační nástroj, anebo komunikační rozhraní pro tento nástroj, jenž poskytuje prostředí a prostředky pro souběžnou simulaci více FMU jednotek současně.

Element *UnitDefinitions* obsahuje globální list s definicí všech fyzikálních jednotek a jednotek, které mají být v FMU zobrazeny. Tento soupis parametrů se používá například k převodu zobrazovaných jednotek na základní jednotky soustavy SI používaných v rovnicích modelu.

TypeDefinitions je element obsahující soupis všech typů proměnných použitých v FMU. Obdobně je tomu u elementu *LogCategories*, který obsahuje parametr definující rozsah logovaných zpráv při práci s FMU. Oproti verzi 1.0 standardu FMI je zde možné jednoduše nastavit rozsah nebo kategorii, které mají být zapisované do logovacího souboru.

Element *DefaultExperiment* poskytuje prostor pro definici nastavení parametrů simulace. Mezi parametry simulace patří atributy *startTime*, *stopTime*, *tolerance* a *stepSize*. Tyto parametry nejsou striktně vyžadovány, nicméně pro mód *Cosimulation* atribut *stepSize* udává preferovanou rychlost komunikace mezi modelem a simulačním nástrojem.

Méně významný element *VendorAnnotations* tvoří prostor vysvětlivek, které může libovolný simulační nástroj pod jeho unikátním identifikátorem zanechat v popisu modelu. Významnější element ModelVariables, který tvoří prostřední sekci popisu modelu, poskytuje statické informace o všech použitých proměnných. Proměnné použité v popisu modelu mohou nabývat pouze těchto hodnot: *Real*, *Integer*, *Boolean*, *String* a *Enumeration*. Více informací ohledně parametrů a příkladu jejich použití se nachází ve standardu FMI [13].

Nezávisle na tom, jak jsou rovnice modelu řešeny, definuje element *ModelStructure* strukturu parametrů, ve které zohledňuje základní rovnice modelu. Přínosem této struktury je logicky uspořádaný a seřazený soupis výstupů, stavů a derivací, které jsou použity při simulaci nezávisle na simulačním nástroji, a to vždy ve stejném pořadí. Volitelná část tohoto elementu pak definuje, jakým způsobem jsou derivace a výstupy závislé na vstupech a stavech modelu.

Rozdílů FMI verze 2.0 je oproti FMI verzi 1.0 celá řada. Jejich výčet a následné vysvětlení by svým obsahem splynul s definicí standardu FMI 2.0, proto zde byly zmíněny pouze hlavní změny, které byly do standardu FMI 2.0 implementovány. Další důležité informace o FMI 2.0 se nacházejí v definici standardu FMI [13].

3 LabVIEW a CompactRIO

Komerčně dostupný software LabVIEW, jehož název vychází ze zkratky Laboratory Virtual Instruments Engineering Workbench, je produktem společnosti National Instruments (NI). Tento programovací nástroj vznikl před třiceti lety s cílem propojit měřící zařízení s počítačem. V té době revoluční myšlenka nově vzniklého programovacího nástroje LabVIEW je založena na grafickém programování. Grafické programování je realizováno pomocí vkládání a následném propojování jednotlivých bloků do ucelených logických diagramů reprezentujících do té doby běžně používané textové zápisy.

Jednotlivé funkční bloky v diagramu se vzájemně propojují a vzniká tak mezi nimi datový tok. Datový tok zajišťuje čekání každého funkčního bloku na dostupnost všech informací přivedených na jeho vstupy. Teprve poté, co jsou všechna přivedená vstupní data příslušného funkčního bloku platná, dojde k provedení jednotlivých funkcí obsažených v bloku. Jednotlivé stavební bloky jsou nazývány virtuálními přístroji (Virtual Instruments - VI) a mohou zastávat matematické, logické či jiné sofistikované programy, které umožňují analýzy, měření, testování, řízení, zpracování a manipulaci s velkými datovými objemy.

LabVIEW velmi usnadňuje a do značné míry také zjednodušuje integraci hardwaru. Díky tomu je možné v relativně krátkém čase začít se sběrem dat a jejich následnou vizualizací a zpracováním v podstatě z jakéhokoliv I/O zařízení. Ve spojení s grafickou syntaxí programování, která zkracuje dobu vývoje řešení technického problému, LabVIEW usnadňuje návrh komplexních řídicích, měřících a testovacích systémů z mnoha technických odvětví. LabVIEW se tak stalo téměř nepostradatelným pomocníkem při řešení výzev technického rázu napříč širokým spektrem aplikačních oblastí.

3.1 LabVIEW

Vývojové prostředí, které LabVIEW nabízí, je vcelku jednoduché. Po spuštění se vytvoří nový projekt, jehož adresářová struktura bude obsahovat nově vytvořené programy, diagramy nebo vývojová schémata, která se označují jako VI. Tyto VI je možné spustit v prostředí LabVIEW, ale i exportovat do samostatně spustitelných souborů, k jejichž spuštění není licencované prostředí LabVIEW nezbytné.

Po založení projektu a vytvoření nového VI je zobrazeno okno *„Front Panel“* a *„Block Diagram“*. *Front Panel* slouží k ovládání a vizualizaci naprogramovaného řešení procesu. Je možné na něm zobrazit tlačítka, grafy, pole pro zadávání textových řetězů a mnohé další prvky, které jsou propojeny s *Blokovým Diagramem*. *Blokový Diagram* slouží ke strukturalizovanému grafickému zápisu algoritmu, podle

kterého budou vykonávány jednotlivé funkce vytvořené aplikace. *„Naprogramování“* se provádí výběrem a umístěním vhodných bloků z rozsáhlé palety funkcí na pracovní plochu VI. Propojování jednotlivých bloků je realizováno pomocí virtuálních vodičů, které reprezentují posloupnost toku dat ve vytvářeném schématu. V případě spuštění vytvořeného VI mohou být některé operace či funkce vykonány přednostně, a to z toho důvodu, že je v programu použito paralelního toku dat.

Pro účely plnění cílů této práce bylo LabVIEW použito k importování modelu elektromechanického systému definovaného jazykem Modelica. Tento model byl nejdříve vytvořen v prostředí OpenModelica a následně byl vyexportován do Functional Mock-up Unit, které podléhá standardu FMI 2.0. Jedním z cílů této práce je naimportovat vygenerované FMU právě do prostředí LabVIEW. Vzhledem k tomu, že naimportovaný model má být použit k Hardware-In-the-Loop simulaci, je zapotřebí použít dostatečně rychlé a výkonné rozhraní, které umožní spojení počítače s hardwarem. Pro připojení fyzického zařízení (hardwaru) k PC byl použit systém CompactRIO, který poskytuje možnost připojení fyzických vstupů a výstupů díky široké paletě I/O karet.

3.2 Možnosti platformy CompactRIO

Platforma CompactRIO (dále jen cRIO) se skládá ze základního rámu (častěji však nazývaného šasi), které kombinuje dva typy vestavěných průmyslových kontrolérů. První, Real-Time kontrolér, kterým je cRIO vybaveno, zajišťuje komunikaci a zpracování signálů v reálném čase. Umožňuje implementaci řídicích algoritmů a podporuje širokou škálu hodinových frekvencí, které jsou důležité pro připojení Rekonfigurovatelných I/O modulů – odtud zkratka RIO. Druhým kontrolérem v šasi je hradlové pole – Field Programmable Gate Array (FPGA), které poskytuje možnost realizace vysokorychlostního řízení a vlastního časování a spouštění přímo v hardwaru. Další výhodou FPGA, poháněného čipem firmy Xilinx, je vysokorychlostní zpracování dat. Oba kontroléry jsou však podle tvrzení National Instruments kompatibilní pouze se sériově vyráběnými I/O moduly.

I/O moduly se vkládají do šasi cRIO a vyrábí se v mnoha velikostech a provedeních. I/O moduly jsou mechanicky odolné a umožňují připojení nebo odpojení bez nutnosti vypínání jednotky cRIO. Jedná se o speciálně navržené I/O karty, které jsou určené pro připojení určitého laboratorního zařízení příslušným komunikačním rozhraním (např. GPIB, CAN, sériové linky nebo také drivery pro motory a další).

Kromě Real-Time kontroléru a hradlového pole platforma cRIO poskytuje možnost buď přímého připojení cRIO k počítači pomocí síťové karty, anebo je možné cRIO připojit do místní sítě LAN. V druhém případě je nutné konfigurovat příslušné síťové prvky LAN (například router), aby bylo zamezeno cílenému zneužití zařízení.

3.2.1 LabVIEW RealTime

Pro práci s platformou cRIO je potřeba nainstalovat následující software do PC, které bude ke cRIO připojeno:

- NI LabVIEW 2009 nebo novější
- NI LabVIEW Real-Time Module 2009 nebo novější
- NI LabVIEW FPGA Module 2009 nebo novější
- NI-RIO 3.2 nebo novější.

Nainstalované balíčky funkcí se objeví v paletě bloků v sekci Control & Simulation. Po jejich instalaci nebo kontrole je vhodné také zkontrolovat dostupné aktualizace pro Measurement & Automation Explorer (MAX) na příslušném PC. MAX nemůže být bohužel aktualizován automaticky.

K práci s platformou cRIO je nutné přivést napájení na vstupní svorky šasi. Správné zapojení napětí a jeho parametry je nutné vždy vyhledat v dokumentaci příslušného šasi. Obecně lze říci, že je možné téměř pro všechny šasi použít napájení $+24VDC$. Propojení cRIO a počítače lze realizovat pomocí rozhraní USB 2.0, nebo lépe ethernetovým kabelem. Do šasi cRIO se vloží potřebné I/O moduly, které slouží pro připojení fyzického hardware. Všechny připojené karty se společně se svými datovými komunikačním kanály objeví vždy ve stromové struktuře projektu.

Grafické programování může být realizováno v blocích VI, které svým umístěním v adresářové struktuře vypovídají o tom, pro jaké účely slouží a kterým výpočetním nástrojem budou zpracovávány. V případě, že je žádoucí, aby VI bylo vykonáváno pomocí procesoru počítače, který je s CompactRIO spojen, je nutné umístit VI pod adresář Dependencies. Chceme-li vytvořené VI nechat obsluhovat Real-Time procesorem uvnitř šasi CompactRIO, musí být VI uloženo pod adresářem šasi např. Chasis (cRIO-9074). Poslední možností je provozovat VI v prostředí FPGA, které je taktéž uvnitř šasi. Takovéto VI musí být umístěno pod adresářem FPGA Target.

Programování Real-Time smyčky se realizuje uvnitř VI, ve kterém je vložena Real-Time simulační smyčka z palety funkcí Control & Simulation. Do této smyčky lze vytvořit diagram využívající výpočetní kapacitu Real-Time procesoru uvnitř cRIO. Při použití velmi rychlého hradlového pole se musí graficky vytvořený program nejprve zkompilovat a následně přeložit do jazyku VHDL. Překlad do jazyka VHDL zabere cca 10 minut, což je doba, po kterou není možné s VI v LabVIEW pracovat (v jiném VI však ano).

3.2.2 LabVIEW FPGA

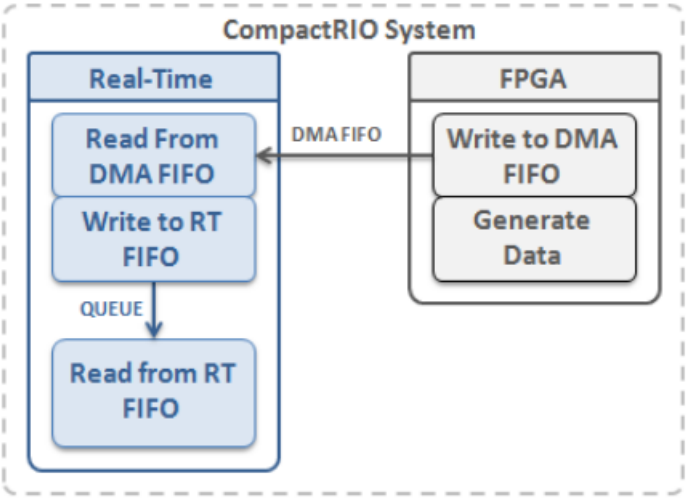
Výpočetní výkon Field Programmable Gate Array (v překladu Programovatelné Hradlové Pole) se používá pro správu a obsluhu časově kritických smyček. Díky

použití FPGA je možné dosáhnout vysoké rychlosti při komunikaci s hardwarem nebo externími hardwarovými nástroji připojenými pomocí I/O modulů.

Časově kritická smyčka, která komunikuje s RealTime smyčkou, slouží také k obsluze I/O modulů. S jednotlivými vstupy a výstupy I/O modulů lze zacházet jako s kterýmkoliv jiným knihovním blokem (VI). Jednotlivé porty (vstupy nebo výstupy) stačí přetáhnout do RealTime smyčky. Následně je možné z portů I/O modulů číst nebo naopak do nich zapisovat žádané hodnoty, jejichž rozsah je však omezen typem I/O modulu.

Vzhledem k tomu, že FPGA čip použitý k výpočtu „rychlé“ smyčky programu může být v závislosti na použité platformě CompactRIO programován pouze v jazyku VHDL nebo VERILOG, je potřeba nejdříve vytvořený program zkompilovat a následně přeložit do jednoho z těchto programovacích jazyků. K přeložení graficky editovaného programu z prostředí LabVIEW do FPGA čipu slouží šipka Run VI, stejně jako pro spuštění vytvořeného VI, které nepracuje s FPGA. Po jejím stisknutí se vytvořený program v prostředí LabVIEW automaticky přeloží do jazyka podporovaného FPGA čipem.

Jednou z nevýhod překladu do jazyka FPGA čipu je doba překladu a zpracování, která trvá řádově minuty (typicky 10 minut), která však také závisí na složitosti překládaného programu. Podle náročnosti překládaného programu se musí totiž vytvořit potřebný počet komponent cílového jazyka. Z těchto důvodů je vhodné, pokud je to možné, otestovat funkčnost vytvořeného programu na RealTime procesoru, kde není nutné nic překládat, a vývoj programu je tedy mnohem rychlejší a efektivnější. Doporučuje se tedy pracovat s více VI a otestovaný program z RealTime smyčky následně zkopírovat do smyčky časově kritické.



Obr. 3.1: Diagram architektury Real-Time a FPGA smyčky [15]

Tak jako v mnoha jiných programovací jazycích a různých vývojových prostředích je možné vytvořený program procházet a případně také debugovat a opravovat, tak i LabVIEW disponuje těmito možnostmi. V klasickém VI, které používá jako výpočetní prostředek centrální procesorovou jednotku (CPU) počítače, na kterém je VI spuštěno, je možné spustit za pomocí žárovky režim obdobný debugu v jiných vývojových prostředí. Aktivací tohoto režimu v prostředí LabVIEW se běh a vykonávání graficky vytvořeného programu zpomalí a po virtuálních spojích natažených mezi vstupy a výstupy jednotlivých bloků začnou „téct“ data. Tok dat ve schématu je tak možné sledovat a zjistit, kde při vykonávání/vytváření programu nastala chyba. Chybu je možné/nutné po zastavení debugovacího režimu opravit a spustit VI znovu. Této debugovací schopnosti je možné využít také pro RealTime smyčky, ale není možné ji použít pro opravy chyb vytvořených v časově kritické smyčce.

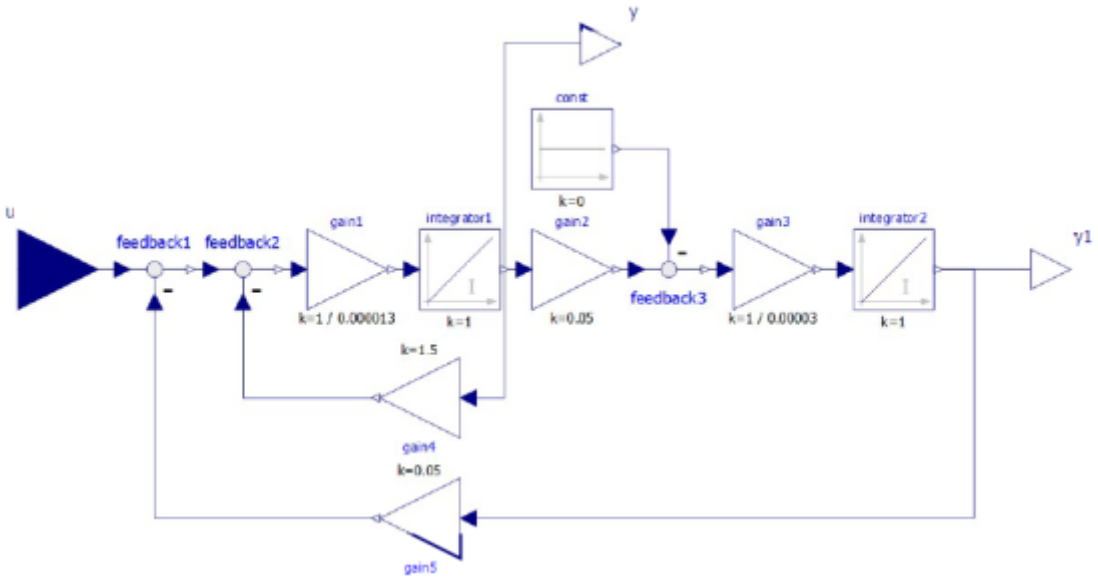
Názorná ukázka rozložení a komunikace mezi programovými smyčkami pro práci s platformou CompactRIO je uvedena na obrázku 3.1. Ze schématu je možné vyčíst, že programová smyčka pro FPGA slouží ke generaci dat, a to jak směrem do, tak i směrem z CompactRIO. Získaná data z I/O modulů jsou zapsána do DMA fronty typu FIFO (First In First Out). Tímto data opouští časově kritickou smyčku a jsou dále zpracována v Real-Time smyčce. Data jsou zde přečtena z DMA FIFO a zapsána do Real-Time fronty typu FIFO (RT FIFO). V Real-Time smyčce jsou vyzískaná data vhodně zpracována dle potřeb aplikace.

Další užitečné informace ohledně rozvržení programových smyček při práci s platformou CompactRIO jsou dostupná na webových stránkách National Instruments [15], kde je možné nalézt také srovnání výkonnosti různých platforem cRIO.

4 Modelování elektromechanického systému v jazyce Modelica

K vytvoření modelu elektromechanického systému v jazyku Modelica bylo použito programu OpenModelica. OpenModelica umožňuje akauzální modelování, čímž je garantováno, že v případě vytvoření značně složitého systému model neztrácí svou fyzikální podstatu a nestává se tak nečitelným nebo dokonce nepřehledným (více v kapitole 1.3). Navíc OpenModelica, tedy OpenModelica Connection Editor podporuje standard FMI, a to jak ukládání, tak i načítání modelů ve formátu Functional Mock-up Unit.

Model elektromechanického systému byl vytvořen v grafickém editoru OMEdit za použití základních knihovních bloků. Jako model elektromechanického systému byla vybrána konstrukce BLDC (Brushless DC electric motor) motoru. Vytvořené schéma zapojení BLDC motoru v grafickém editoru OpenModelica je ukázáno na obrázku 4.1. Funkčnost vytvořeného modelu BLDC motoru byla také otestována v simulačním nástroji vestavěném v OMEditoru.



Obr. 4.1: Model BLDC motoru v OpenModelica Connection Editoru

4.1 Simulace modelu v OpenModelice

K tomu, aby mohl být vytvořený model BLDC motoru simulací otestován, bylo potřeba provést nejprve syntaktickou kontrolu. Kontrola syntaxe modelu byla pro-

vedena pomocí tlačítka zaškrtávacího znaku, které se nachází ve vrchní části programu OpenModelica. Po odstranění drobných chyb a úspěšné kontrole správnosti modelu byly nastaveny parametry simulace v souladu s tabulkou 4.1. Teprve po na-

Tab. 4.1: Nastavení simulace BLDC motoru v OMEditoru

Parametr	**Hodnota**	**Jednotka**
StartTime	0.1	[s]
StopTime	1.0	[s]

stavení těchto parametrů byla simulace spuštěna funkčním tlačítkem ve tvaru šipky směřující doprava, která se také nachází ve vrchní liště funkčních tlačítek. Po zobrazení simulačního okna, kde nebyly z počátku vidět žádné dosažené výsledky, bylo nezbytné zapnout vykreslení jednotlivých signálů modelu.

Ve vestavěném simulátoru je možné zapnout vykreslení nasimulovaného průběhu signálu na jakémkoliv vstupním nebo výstupním konektoru libovolného bloku. Pro ověření správnosti modelu BLDC motoru však postačily dva průběhy, a to průběh elektrických otáček motoru a proud vinutím motoru. Z nasimulovaných průběhů proudu a elektrických otáček motoru bylo patrné, že odpovídají teoretickým předpokladům a konstrukčním vlastnostem BLDC motoru podle [16]. Po tomto experimentálním ověření modelu následovalo uložení vytvořeného modelu do Functional Mock-up Unit, která měla být návazně naimportována do prostředí LabVIEW.

4.2 Export modelu do formátu FMU

Export modelu formátu FMU byl proveden vestavěnou funkcí OMEditoru. Tato funkce se nachází ve vrchní liště menu pod označením standardu FMI. Po rozbalení této nabídky máme možnost výběru ze dvou variant, a to importu modelu nebo generování modelu do jednotky FMU. Výběrem možnosti *„Export FMU“* byl spuštěn automatický proces generace modelu do FMU. Průběh procesu exportování bylo možné sledovat ve spodním okně označeném jako *„Message Browser“*. Do tohoto okna hlášení programu OMEdit byla vypsána zpráva viz výpis 4.1.

```
1 The FMU is generated at C:/Users/Luk??/AppData/Local/Temp/
    BLDC.fmu
```

Výpis 4.1: Umístění vyexportovaného FMU

Z výpisu 4.1 je patrné umístění vygenerovaného modelu ve formátu FMU. Vzhledem k tomu, že při práci byl použit OS Windows 10 s českým rozhraním, který bez sebemenšího upozornění či potíží povolí vytvoření uživatelského účtu se znaky české

diakritiky, objevily se právě tyto znaky v přístupové cestě k vygenerovanému FMU. Vzhledem k tomu, že většina v této práci použitých programů nepodporuje češtinu, byly v zásadě dvě možnosti řešení vzniklého problému.

První možností bylo použití uživatelského účtu, který neobsahuje znaky s českou diakritikou. Druhou možností bylo přeinstalovat OS a použít anglické prostředí, které tyto znaky nepodporuje. Pro dočasné řešení situace byla v práci použita varianta první, a sice změna uživatelského účtu. Do budoucna je však vhodné uvažovat o použití anglického prostředí operačního systému, neboť by se situace mohla opakovat i v jiných programech, pokud by soubory byly uloženy s českou diakritikou.

Po novém exportu modelu již pod záštitou uživatelského účtu, který neobsahoval znaky české diakritiky, byl v *„Message Browseru“* obdržen výpis podobný 4.1, který již neobsahoval zástupné znaky české diakritiky ve jméně uživatelského účtu.

```
1 The FMU is generated at C:/Users/Lukas/AppData/Local/Temp/
    BLDC.fmu
```

Výpis 4.2: Opravené umístění vyexportovaného FMU

Vyexportovaný model byl uložen do výše uvedeného dočasného adresáře. Tento adresář obsahoval také další soubory, které jsou popsány v následující kapitole.

4.3 Struktura vytvořeného modelu

Do dočasného adresáře, ve kterém se nacházel také popis modelu ve formátu FMU, byla vyexportována řada dalších souborů. Adresářová struktura všech vygenerovaných souborů do dočasného adresáře je uvedena níže.

- `/` ..... kořenový adresář dočasné složky
  - `ModelName.fmu` ..... vyexportované FMU
    - `binaries` ..... adresář s binárními knihovnami
      - `win64` ..... adresář s dll knihovnami pro danou platformu OS
    - `sources` ..... adresář se zdrojovými programy jazyka C
      - `include` ..... adresář standardu FMI
      - `ModelName.c` ..... zdrojový kód modelu v jazyce C
      - `ModelName.h` ..... hlavičkové soubory k programu v jazyce C
      - `ModelName.lib` ..... knihovní funkce ke zdrojovému programu v jazyce C
      - `Makefile.in` ..... makefile
    - `modelDescription.xml` ..... XML popis modelu
  - `ModelName.libs` ..... knihovna modelu
  - `ModelName_FMU.libs` ..... knihovna modelu FMU
  - `ModelName_FMU.log` ..... logovací soubor
  - `ModelName_FMU.makefile` ..... kompilační soubor makefile
  - `ModelName_info.json` ..... JavaScriptový objektový zápis modelu

Ve vygenerovaném souboru modelu BLDC motoru se nachází jeho popis jazyce Modelica. V XML souboru se nachází popis modelu, použitá verze FMI, verze XML, název modelu, typ vyexportovaného modelu. Dále jsou zde uvedeny proměnné modelu, jejich inicializace a v neposlední řadě také rovnice, které model popisují. V XML kódu, který popisuje model BLDC motoru, se však nachází mnohem více informací, které nebyly v této fázi práce podstatné.

5 Import FMU do prostředí LabVIEW

Pro zahájení práce s modelem v prostředí LabVIEW a na platformě CompactRIO bylo nutné zjistit možnosti, kterými by bylo možné import modelu ve formátu Functional Mock-up Unit zrealizovat. Po prozkoumání možností, kterými by bylo možné vygenerované FMU naimportovat do prostředí LabVIEW, připadaly v úvahu dvě nalezené varianty řešení.

První varianta směřovala k použití Dynamic Link Library, jinak také označovaných jako DLL knihovny. Druhá varianta, která byla zvolena pro import modelu ve formátu FMU do prostředí LabVIEW, používala add-on, který byl doinstalován do prostředí LabVIEW. Druhá zmíněná varianta byla nalezena na webových stránkách diskuzního fóra výrobce programu LabVIEW dostupného z [17].

5.1 FMI add-on pro LabVIEW

Rozšíření s názvem *„LabVIEW support for FMI for Model Exchange“* obsahuje již vytvořenou knihovní funkci, která zjednodušuje import modelu do prostředí LabVIEW. Podle popisu tohoto rozšíření v [17] podporuje tento volně dostupný add-on import modelu FMU v módu ModelExchange, a to jak pro verzi standardu FMI 1.0, tak i pro verzi FMI 2.0. Tento doplněk však nedokáže vygenerovat model v Co-simulation podobě a s jeho pomocí tak není možný export FMU z prostředí LabVIEW.

Vzhledem k tomu, že se však práce zabývá problémem, jak vytvořený model naimportovat do prostředí LabVIEW, nejsou výše uvedené nedostatky doplňku překážkou, pro kterou by toto řešení nemohlo být použito. Při dalším studiu použití tohoto rozšíření byly nalezeny video návody, na kterých byla demonstrována jednoduchost importu modelu ve formátu FMU do prostředí LabVIEW.

Další kroky práce směřovaly tedy k instalaci a následné implementaci rozšíření, které se na první pohled zdálo být jednoduše použitelné a realizovatelné.

5.1.1 Instalace FMI ad-onu do prostředí LabVIEW

V rozšíření FMI pro LabVIEW, které bylo vystaveno na stránkách diskuzního fóra společnosti NI [17], se nacházely celkem tři archivační soubory typu *.zip. První soubor s názvem *„NI Labs FMI for Model Exchange Tutorial with sample FMU.zip“* obsahoval dokument ve formátu **.pdf*, který popisoval postup práce s nainstalovaným rozšířením. Součástí tohoto adresáře byl také ukázkový model ve formátu FMU, na kterém byl podle tutoriálu uveden postup jeho importu. Druhý archivační soubor s názvem *„NI Labs–FMI For Model Exchange.zip“* obsahoval instalační soubory

k rozšíření. Třetí a poslední archiv s názvem *„NI Labs–FMI For Model Exchange v2.zip“* obsahoval stejně jako předešlý archiv instalační soubory rozšíření, jen s tím rozdílem, že se jednalo o novou verzi rozšíření. V nové verzi rozšíření označené 2.0 bylo vypnuto detailní protokolování a opraveno nesprávné nastavení v konfiguračním souboru s názvem „EMI_FMI Configuration.txt“.

Po úspěšné instalaci nastal problém se zobrazením ikony nainstalovaného add-onu. I když byl add-on správně nainstalován, nezobrazil se v paletě knihovních bloků a dostupných VI. Při hledání řešení bylo nejprve nutné zjistit, do které palety knihovnách bloků měl být doplněk pro LabVIEW zobrazen. Ukázalo se, že doplněk měl být zobrazen v paletě *„External Models“* pod knihovnou *„Control&Simulations“*. Zobrazena měla být ikona s názvem „FMI“. V knihovně *„External Models“* se blok FMI však nenacházel.

Po prozkoumání složek programu LabVIEW v adresáři *„Program Files/National Instruments“* bylo zjištěno, že rozšíření je sice nainstalováno v LabVIEW správně, ale je umístěno ve špatném adresáři. Namísto toho, aby bylo rozšíření nainstalováno do aktuální verze LabVIEW (tedy 2017), bylo nainstalováno ve verzi 2012. Oprava této chyby byla provedena vyjmutím rozšíření a souvisejících souborů z chybného adresáře a umístěním vyjmutých položek do adresáře s aktuální verzí LabVIEW 2017.

Po restartování počítače a opětovném spuštění vývojového prostředí LabVIEW již byla paleta doplňku zobrazena v paletě knihovních funkcí *„Control&Simulations“*. Po úspěšném vyřešení tohoto problému byl dále importován ukázkový model podle nalezeného návodu.

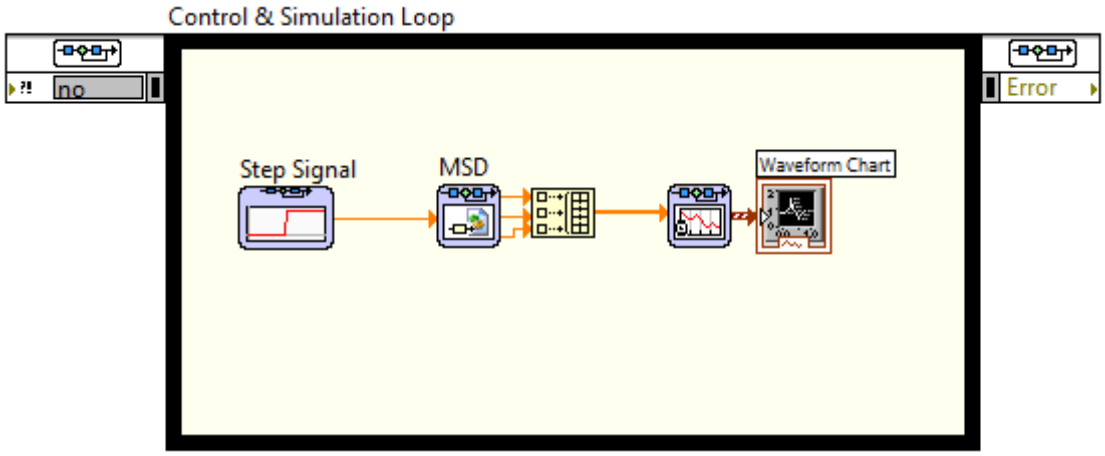
5.1.2 Import ukázkového FMU

Poté, co byly odstraněny potíže se zobrazením palety doplňku, byl prostudován návod obsahující postup importování modelu FMU do prostředí LabVIEW.

Postup ověření správné funkčnosti nainstalovaného rozšíření byl následující:

1. Vytvoř nové VI a ulož ho
2. Do grafického prostředí vlož „Control&Simulation Loop“ z palety „Control Design&Simulation“
3. Z palety „Simulation“ zvol „External Models“ a přetáhni blok „FMI“ do vytvořené smyčky
4. Automaticky je vyvoláno dialogové okno, které vyzývá k výběru modelu uloženého ve formátu FMU v defaulním systémovém průzkumníku souborů
5. Procházením složek vyber model “MSD.fmu„
6. V případě potřeby vyber možnost nastavení některého parametru modelu jako „tunable“
7. Potvrď import FMU

Ještě před potvrzením importu FMU lze vybrat, hodnotu kterých proměnných modelu bude možné v navazující práci v prostředí LabVIEW měnit. Po potvrzení importu modelu FMU byly do smyčky „Control&Simulation Loop“ vloženy bloky *StepSignal*, *BuildArray*, *SimTime Waveform* a *Waveform Chart*. Výsledné propojení ukázkového modelu „MSD.fmu“ je zobrazeno na obrázku 5.1.



Obr. 5.1: Vytvoření ukázkového modelu MSD.fmu

Spuštěním VI s naimportovaným modelem *„MSD“* byla provedena jeho simulace a vypočítané průběhy byly zobrazeny v grafu na *„FrontPanelu“*. Nasimulované průběhy odpovídaly průběhům uvedeným v [17]. Detailní postup importu modelu včetně názorných obrázků a dosažených výsledků je možné najít v tutoriálu dostupném z [17].

Úspěšně se tedy podařilo naimportovat vzorový model ve formátu FMU do prostředí LabVIEW. Další kroky směřovaly k importu vlastního modelu BLDC motoru pomocí tohoto rozšíření.

5.1.3 Import vlastního FMU

Vlastní model BLDC motoru byl již vytvořen, zbývalo importovat jej do prostředí LabVIEW. Při pokusu o import vlastního modelu FMU však došlo při potvrzení vložení vytvořeného modelu vždy k chybě, po jejímž zobrazení bylo okno, které umožňovalo výběr modelu FMU, zavřeno a nedošlo k výměně předchozího vzorového modelu za model nový.

Bylo zjištěno, že chyba, která nastala, byla způsobena použitým typem FMU. Použitý doplněk totiž umožňuje import modelů FMU pouze ve formátu *ModelExchange*. Po detailním prozkoumání vygenerovaného XML souboru vlastního modelu

bylo zjištěno, že typ FMU, ve kterém je současný model BLDC motoru vyexportován, je nastaven jako *„Co-simulation“* namísto *„ModelExchange“*. Pokud by byl ponechán Co-simulační mód jednotky FMU, nebylo by možné model do prostředí LabVIEW tímto nástrojem naimportovat. Na základě tohoto zjištění byla provedena oprava výstupního formátu modelu, který je generován z prostředí OpenModelica Connection Editor.

V OMEditoru byl nastaven v menu „Tools → Options → FMI“ typ výstupního formátu modelu na *„ModelExchange“*. Dále zde byla nastavena verze FMI z 1.0 na FMI 2.0. Změny byly uloženy a program znovu spuštěn. Model elektromechanického systému byl znovu otevřen v prostředí OMEditu a následně znovu vyexportován jako FMU v add-onem podporovaném formátu.

Znovu vyexportovaný model ve formátu FMU verze 2.0 typu *„ModelExchange“* byl následně použit k importu do prostředí LabVIEW. Úspěšně se podařilo poupravený model načíst do prostředí dialogového okna add-onu (doinstalovaného FMI bloku). V tomto dialogovém okně se také zobrazily veškeré vstupní a výstupní terminály modelu, které byly definovány jazykem Modelica v prostředí OMEditoru. Nicméně ani po úspěšném vyřešení dosavadních nesnází neproběhla po potvrzení tlačítka *„OK“* náhrada původního vzorového modelu za model nový.

Po smazání knihovního bloku FMI z kontrolní smyčky a jeho opětovném vložení přes rozhraní add-onu se již ve vytvořeném modelu terminály zobrazily. Avšak ani v tomto případě nebylo možné úkol importu vlastního modelu prohlásit za zcela splněný. Blok VI sice obsahoval vlastní model FMU a také zobrazil („viděl“) vstupy a výstupy modelu definované jazykem Modelica v dialogovém okně zobrazeném při importu modelu, nicméně doplněk LabVIEW z neznámých důvodů nezobrazil vstupní a výstupní terminály na bloku naimportovaného modelu. Bez zobrazení terminálových svorek na tomto bloku nebylo možné dále s modelem v LabVIEW pracovat.

Při řešení problémů spojených se zobrazením terminálových svorek na bloku naimportovaného modelu a po mnoha provedených experimentech zaměřených na úpravu vlastního modelu došlo k pochybnostem o tom, zda je vlastní model vůbec vyexportován v souladu se standardem Functional Mock-up Interface 2.0, neboť jeho XML popis obsahoval formálně vše, co vzorový model *„MSD.fmu“*. Jako vhodný nástroj pro kontrolu souladu vlastního vyexportovaného FMU se standardem FMI 2.0 je organizací OpenModelica doporučován *„FMU compliancy checker“*, který je volně dostupný z [13] v sekci *Downloads*. Následující kapitola 5.2 se zaměřuje na to, zda vlastnoručně vytvořený model FMU opravdu vyhovuje požadavkům standardu FMI. Kapitola 5.2 tak vysvětluje a demonstruje použití nástroje *„FMU compliancy checker“*.

5.2 FMU Compliance Checker 2.0

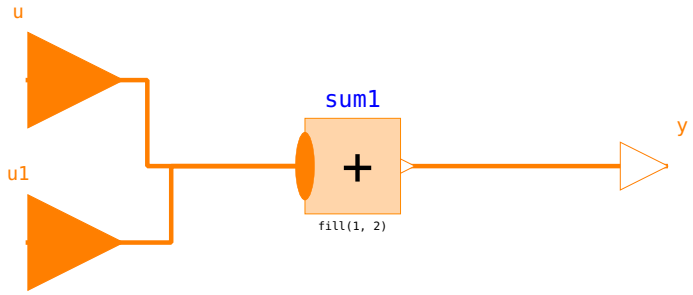
Stěžejním bodem této práce je importování vlastního modelu, který je definován jazykem Modelica, do prostředí LabVIEW. Import vzorového modelu s názvem *MSD.fmu* proběhl zcela úspěšně. Při importování vlastního modelu elektromechanického systému se však nedařilo zobrazit terminálové svorky na knihovním bloku add-onu v prostředí LabVIEW, ve kterém měl být vytvořený model dále rozdělen tak, aby mohly jednotlivé části modelu spolupracovat s Real-Time i s FPGA čipy osazenými v platformě CompactRIO.

Problém mohl pravděpodobně nastat z těchto důvodů:

- Nesprávný popis modelu vytvořený v OMEditoru
- Model byl vyexportován do formátu FMU správně, ale nevyhovuje FMI 2.0
- Model i FMU je v pořádku, chyba doplňku LabVIEW
- Kombinace výše uvedených případů
- Další možnosti

Při hledání řešení problému se zobrazením terminálových svorek na knihovním bloku naimportovaného modelu byla nejprve věnována pozornost doplňku doinstalovanému do LabVIEW. Na serveru společnosti OpenModelica [23] byly nalezeny hotové modely ve formátu FMU. Tyto modely byly společností OpenModelica otestovány a prohlášeny za vyhovující standardu FMI. Se získanými modely byla provedena celá řada experimentů, které se zaměřovaly hlavně na jejich import. Bohužel se však nepodařilo ani jeden ze získaných modelů úspěšně naimportovat add-onem do takové míry, aby byl problém se zobrazením terminálů na bloku LabVIEW odstraněn nebo alespoň částečně vyřešen.

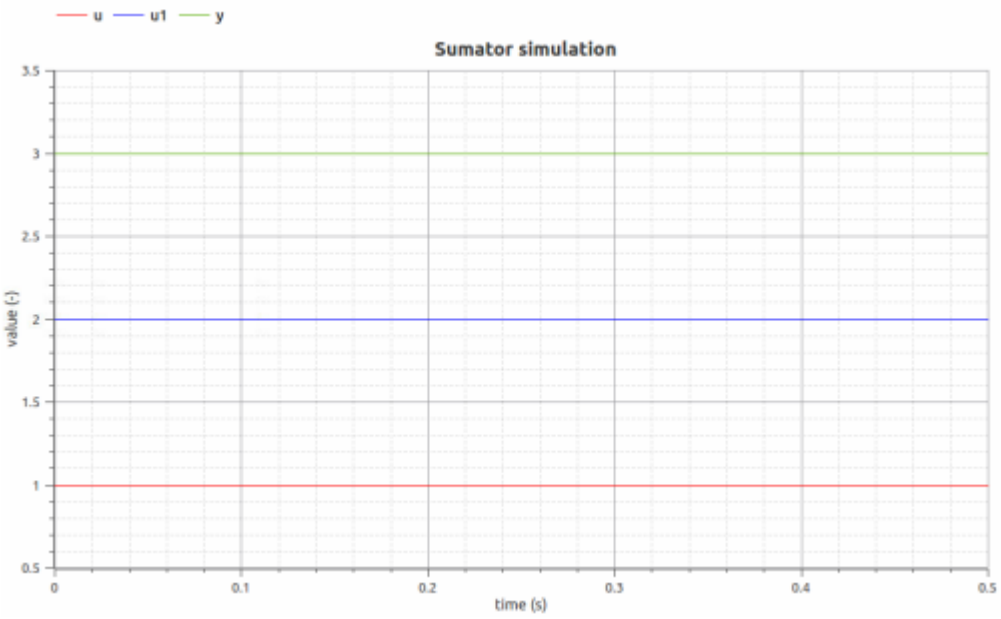
Po řadě neúspěšných pokusů o nalezení řešení problémů spojených s importem FMU se pozornost práce začala zaměřovat spíše na popis vlastního modelu. Vytvořený model elektromechanického systému obsahoval větší množství bloků, a proto byl vytvořen nový a jednodušší model za účelem minimalizace případných chyb, které by mohly být způsobené větším množstvím různorodých bloků.



Obr. 5.2: Model Sumator v OpenModelica Connection Editoru

Byl vytvořen nový experimentální model (viz obrázek 5.2), který byl pojmenován *Sumator*. Jednalo se o popis prostého součtu dvojice čísel. Vstupem modelu byly dvě číselné hodnoty. Výstupem modelu byla suma vstupních hodnot. K vytvoření tohoto modelu bylo použito pouze čtyř knihovních bloků z OMEditoru s tím, že každý z těchto bloků byl celočíselného datového typu *Inteeger*, a to proto, aby se předešlo případným dalším problémům s reprezentací desetinného čísla v různých vývojových prostředích.

Jednoduchý model sumátoru byl v prostředí OMEditoru úspěšně odsimulován (viz obrázek 5.3) a následně vyexportován do jednotky FMU. Byl proveden pokus o naimportování modelu sumátoru ve formě FMU do prostředí LabVIEW, který se však nezdařil.



Obr. 5.3: Simulace modelu Sumatoru v OpenModelica Connection Editoru

Následně byla provedena kontrola další výše nastíněné možnosti řešení vzniklé situace, a sice zda je vyexportovaný model sumátoru zcela v souladu se standardem FMI verze 2.0. Za tímto účelem byl z oficiálních webových stránek věnujících se standardu FMI [13] ze sekce *Downloads* získán nástroj *FMU Compliance Checker*.

Tento volně dostupný software, který poskytuje Modelica Association, byl touto společností vytvořen především pro navrhování a vývoj nových modelů. *FMU Compliancy Checker* disponuje možnostmi kontroly a ověření správnosti vyexportovaného modelu ve formátu FMU v souladu se standardem FMI 1.0 i FMI 2.0. Tento program případné nalezené chyby hlásí uživateli, a to podle nastavení podmínek kontroly.

V současné době je k dispozici FMU Compliance Checker 2.0, který kontroluje, zda je předložený model FMU v souladu se standardem FMI. Výhodou FMU Compliance Checker verze 2.0 je, že umožňuje kontrolu předloženého FMU podle standardu FMI 1.0 i FMI 2.0.

Mezi základní funkce FMU Compliance Checker 2.0 patří:

- Automatické rozbalení FMU do dočasně vytvořeného adresáře
- Kontrola XML popisu modelu
  - Chyby syntaxe
  - Pořadí prvků
  - Křížové odkazy
  - Sémantická konzistentnost
- Kontrola binárních souborů v FMU
  - Načtení binárního modulu
  - Kontrola dostupnosti všech požadovaných funkcí
  - Test, zda může být FMU typu ModelExchange simulováno dopřednou Eulerovou metodou
  - Test, zda může být FMU typu Co-Simulation simulováno s pevnou velikostí kroku simulace
  - Logování vypočítaných řešení do CSV souboru
  - Poskytnutí vstupních hodnot uložených v CSV souboru simulaci
- Zápis logovacích zpráv do zvláštního souboru

V následující kapitole je uvedeno, jakým způsobem je možné s programem Compliance Checker 2.0 pracovat.

5.2.1 Práce s FMU Compliance Checker 2.0

FMU Compliance Checker 2.0 je dostupný pro celou řadu OS včetně OS Linux, Mac a také Windows. FMU Compliance Checker 2.0 byl nejprve stažen pro 64b verzi OS Windows. Vzhledem k tomu, že tento nástroj je distribuován ve formě zip souboru, bylo nutné tento soubor nejprve rozbalit. Obsah rozbaleného adresáře checkeru:

```
/ ..... kořenový adresář dočasné složky
├── include ..... adresář se standardem FMI
│   ├── FMI1 ..... adresář s hlavičkovými soubory FMI 1.0
│   └── FMI2 ..... adresář s hlavičkovými soubory FMI 2.0
├── ACKNOWLEDGEMENTS.md ..... poznámky
├── BUILD.md ..... pokyny pro aplikaci FMU Compliance Checker 2.0
├── fmuCheck.win64.exe ..... spustitelný program FMU Compliance Checker 2.0
├── HOW-TO-RELEASE.md ..... pokyny pro uvolnění nové verze
├── LICENCE.md ..... licenční podmínky
├── README.md ..... informace o použití FMU Compliance Checker 2.0
└── RELEASE-NOTES.md .... změny provedené ve verzích FMU Compliance Checker 2.0
```

Kromě spustitelného programu *fmuCheck.win64.exe* obsahuje rozbalený archiv také soubor README, ve kterém je uveden návod, jak nástroj spustit a použít. V souboru *README.md*, který lze otevřít v libovolném textovém editoru, je uveden detailní popis všech funkcí, kterými tento nástroj disponuje. Nedílnou součástí tohoto souboru je také popis jednotlivých přednastavení, které je možné při kontrole modelu FMU použít.

Základní předpis použití Compliace Checkeru je následující:

```
1 fmuCheck.<platform> [options] <model.fmu>
```

Výpis 5.1: Předpis použití FMU Compliance Checker 2.0

Ke spuštění kontroly předloženého modelu FMU nebylo nutné díky spustitelnému programu Checkeru cokoliv instalovat. Spuštění Checkeru bylo však nutné realizovat z prostředí příkazového řádku. Ještě před samotným spuštěním bylo nutné se přesměrovat do adresáře, kde byl Checker rozbalen a kde se také nacházel jeho spustitelný program. Do stejného adresáře bylo na základě obsahu souboru README.md zkopírováno dříve vyexportované FMU modelu sumátoru. Teprve nyní bylo možné provést kontrolu samotného FMU.

Pro spuštění kontroly modelu *Sumator.fmu* byla podle [18] zadána následující sekvence:

```
1 fmuCheck.win64 -e log.txt -o result.csv -c , -s 2 -h 1e-3
     -l 5 -t . Sumator.fmu
```

Výpis 5.2: Použití FMU Compliance Checker 2.0

Použité nastavení kontroly modelu sumátoru zajistily, že model *Sumator.fmu* byl zkontrolován na 64b platformě. Při kontrole byl vytvořen a uložen logovací soubor *log.txt*. Výsledky simulace byly uloženy v souboru *result.csv*, ve kterém byly jednotlivé hodnoty odděleny znakem čárky *","*. Předložené FMU bylo simulováno po dobu 2 s s pevným krokem simulace 1 ms. Byly zobrazeny podrobné zprávy o průběhu kontroly. Všechny dočasně vytvořené soubory byly vytvořeny v aktuálně používaném adresáři.

Při testování modelu bylo možné volit z celkem 6 úrovní zápisu chybových zpráv do logovacího souboru. Byly použity všechny dostupné úrovně zápisu do logovacího souboru, avšak dosavadní výsledky testování jednoduchého modelu *Sumator.fmu* nikterak nenasvědčovaly tomu, že by model byl jakkoliv v rozporu se standardem FMI 2.0. Jestliže byl tedy model v souladu se standardem FMI 2.0, jedinou možnou příčinou, pro kterou by nebylo možné naimportování vlastního modelu do prostředí LabVIEW, mohlo být nesprávné fungování doplňku.

Řešení problému se zobrazením terminálů na bloku VI, které nastalo v souvislosti s použitím původně zamýšleného add-onu *LabVIEW support for FMI for Model*

Exchange, bylo zaměřeno na ověření funkčnosti samotného doplňku. Je nutné podotknout, že až do této chvíle nebyly žádné důvody k pochybnostem o funkčnosti tohoto doplňku.

Mimo ověřování funkčnosti samotného add-onu byl pomocí nástroje FMU Compliance Checker 2.0 otestován vzorový model *MSD.fmu*, který se podařilo add-onem úspěšně naimportovat do prostředí LabVIEW. Výsledky testů vzorového modelu *MSD* byly překvapivé. FMU Compliance Checker 2.0 sice nevyhodnotil žádné chyby, nicméně při detailním rozboru obsahu XML popisu vzorového a vlastního modelu FMU bylo zjištěno, že XML popis vzorového modelu, na rozdíl od popisu vlastního modelu, neobsahuje mnohé parametry. Tyto parametry sice nejsou podle FMI standardu povinné, OpenModelica je však při exportu vlastního modelu FMU vkládá do jeho XML popisu. Protože parametry, které OpenModelica vkládá do vlastního modelu nejsou podle standardu FMI povinné, FMU Comliance Checker nemohl chybějící parametr vyhodnotit jako chybu.

Další nesrovnalost, která byla nalezena, se nacházela v záhlaví XML popisu modelu, kde byl jako generationTool (tvůrčí nástroj vzorového modelu *MSD*) uveden řetězec *"By Hand"*. Uvedením tohoto řetězce napovídalo tomu, že tento, byť vzorový model, byl vytvořen ručně, namísto toho, aby byl vygenerován z některého nástroje jako je například OpenModelica nebo častěji používaná Dymola. Přestože add-on ve svém popisu zaručoval možnost použít i pro verzi FMI 2.0, byla ve vzorovém modelu *MSD.fmu* překvapivě použita verze FMI 1.0. Následně byl v diskuzním fóru National Instrument nalezen příspěvek dostupný z [24], kde jeden z uživatelů popisoval totožný problém se zobrazením terminálů na knihovním bloku add-onu v souvislosti s jeho použitím a importem vlastního modelu FMU.

Vzhledem k množství překážek spojených s aplikací doplňku *LabVIEW support for FMI for Model Exchange* bylo nakonec od na první pohled jednoduchého řešení importu vlastních modelů FMU do prostředí LabVIEW za použití tohoto doplňku, od této možnosti upuštěno. Mezi hlavní důvody, které vedly k tomuto kroku patří absence úplné dokumentace add-onu, problémy s diakritikou českých znaků, které byly vkládány do přístupových cest jednotlivých souborů, a také ztráta důvěryhodnosti add-onu vlivem „podezřelého“ popisu modelu *MSD.fmu*, který byl uveden v jeho XML souboru.

Řešením všech výše popsaných problémů s importováním vlastních modelů FMU do prostření LabVIEW by mohl být společností National Instrument komerčně nabízený doplněk *FMI Add-On for NI VeriStand & LabVIEW*, který je však určen výhradně pro vývojové prostředí NI VeriStand, nikoliv pro LabVIEW.

Jako nejvhodnější řešení importu vlastního FMU do prostředí LabVIEW Real-Time bylo za vzniklé situace vyhodnoceno použití vývojového prostředí Eclipse, jehož zásuvný modul umožňoval překlad libovolného programu jazyka C/C++ do po-

doby spustitelného souboru, který bylo následně možné spustit a ladit na RealTime platformě systému cRIO.

5.3 OpenModelica na OS Linux

Po dosavadních zkušenostech s OpenModelicou provozovanou pod operačním systémem Windows, která měla mimo jiné problémy s vkládáním české diakritiky do přístupové cesty některých souborů, byla dále v této práci použita OpenModelica na systému Linux. Konkrétně se jednalo o poslední dostupnou verzi operačního systému Ubuntu 16.04 LTS, která byla dostupná z [25]. Vzhledem k tomu, že systém Linux je open-source, tak i jeho distribuce jsou nelicencované, a tedy volně dostupné. Aby bylo možné na jedné výpočetní jednotce, na které se nacházel operační systém Windows, simultánně pracovat i na operačním systému Ubuntu, byl nainstalován pod operační systém Windows program VirtualBox od společnosti Oracle.

5.3.1 Oracle VM VirtualBox a instalace Ubuntu 16.04 LTS

VirtualBox je multiplatformní virtualizační nástroj, který byl po jeho instalaci použit pro virtualizaci výše zmíněného operačního systému Ubuntu. Prostředí VirtualBoxu je přehledné, díky čemuž je ovládání naprosto intuitivní. K vytvoření virtualizovaného operačního systému Ubuntu bylo nutné provést několik kroků:

1. Spustit Oracle VM VirtualBox
2. V Menu/Počítač vybrat Nový
   (a) Zadat vlastní název virtualizovaného OS
   (b) Vybrat Typ: Linux
   (c) Vybrat Verzi: Ubuntu (64-bit)
3. Přiřadit velikost operační paměti pro OS
4. Vybrat Vytvořit nyní virtuální disk
5. Vybrat typ souboru s pevným diskem jako VDI
6. Přiřadit velikost úložiště na fyzickém pevném disku
   (a) Vybrat Pevná velikost
   (b) Zvolit umístění nového virtuálního pevného disku
   (c) Nastavit velikost nového virtuálního pevného disku minimálně 10 GB
7. Potvrdit vytvoření virtualizovaného OS

Poznámka: V případě již vytvořeného virtuálního obrazu disku je možné na místě položky č.4 zvolit „Použít existující virtuální disk“ namísto „Vytvořit nyní virtuální disk“. Poté je nutné z adresářové struktury vybrat umístění virtuálního disku.

Výsledkem posloupnosti těchto kroků je v levé části okna VirtualBoxu vytvořený operační systém, který bylo následně možné virtualizovat. Spuštění virtualizace se

provede dvojklikem na vytvořenou ikonu operačního systému umístěnou v levé části okna. Po spuštění virtualizovaného operačního systému Linux musí být z adresářové struktury současného operačního systému vybrán image soubor **.iso*, který obsahuje instalační soubory operačního systému Ubuntu. Po výběru místa uložení image souboru OS Ubuntu proběhne spuštění samotného uživatelského prostředí.

Následně je možné vybrat příslušný jazykový balíček, který má být použit. Dále je na výběr ze dvou variant spuštění operačního systému:

1. Vyzkoušet Ubuntu
2. Nainstalovat Ubuntu

První varianta ihned spustí OS Ubuntu. Je možné pracovat se všemi funkcemi OS, ale při práci s programy není možné ukládat některá data. Příkladem může být vytvořený projekt v Eclipse studiu, který po opětovném spuštění vývojového prostředí neuchová informace o jeho nastaveních. Výhodnější je zvolit možnost *Nainstalovat Ubuntu* a pokračovat kompletním nainstalováním Ubuntu, které zmíněný nedostatek zcela odstraňuje.

Během instalace OS Ubuntu je možné vybrat, zda mají být nainstalovány nejnovější dostupné aktualizace pro systém. Dále je nutné vybrat možnost, aby před samotnou instalací byl disk zformátován. Jedná se o formátování pouze virtuálního disku přiřazeného k tomuto virtualizovanému operačnímu systému, tudíž ke ztrátě dat na disku ani k poškození OS Windows nedojde. Mezi poslední kroky úplné instalace OS Ubuntu patří výběr místní časové zóny, rozložení klávesnice a založení uživatelského účtu. Po potvrzení zadaných údajů systém stáhne svoji nejaktuálnější verzi a proběhne instalace, po níž se virtuální OS restartuje.

Následně bylo možné pod virtuální OS Ubuntu 16.04 LTS nainstalovat OpenModelicu a vygenerovat opět FMU sumátoru.

5.3.2 Instalace OpenModelicy na OS Ububtu

Operační systémy Linux k instalaci softwaru používají na rozdíl od operačních systémů Windows správce balíků. Tento správce je připojen k repozitáři, který si lze představit jako server, kde jsou uloženy některé balíčky, které je možné do OS Linux doinstalovat. Pokud si uživatel vybere některý z balíků, správce ho sám stáhne a nainstaluje. Uživatel se nemusí o nic starat.

Vzhledem k tomu, že bylo nutné doinstalovat OpenModelicu do Ubuntu, musel být správce balíků prohledán. Správce balíků však požadovaný software ve svých repozitářích nenašel a nebylo tedy možné tímto nejbezpečnějším možným způsobem OpenModelicu doinstalovat. Další možností, jak doinstalovat software, který není v aktuálním repozitáři, je rozšíření repozitáře například o adresu některého serveru.

Správu repozitářů lze provést v položce Software & Akutalizace v menu Nastavení systému.

Vzhledem k tomu, že OpenModelicu je možné nainstalovat také pro OS Linux, byla otevřena webová stránka dostupná z [26], na které byl uveden postup instalace. Instalování tohoto softwaru je zjednodušeno do takové míry, že stačí otevřít terminál systému a do něj překopírovat uvedené příkazy.

Prvním příkazem je výběr buildu (česky sestavení) programu. Na výběr je ze tří možností, a sice možnosti: *nightly*, *stable* a *release*.

```
1 for deb in deb deb-src; do~echo "$deb␣http://build.
     openmodelica.org/apt␣`lsb_release␣-cs`␣nightly"; done |
     sudo tee /etc/apt/sources.list.d/openmodelica.list
2 for deb in deb deb-src; do~echo "$deb␣http://build.
     openmodelica.org/apt␣`lsb_release␣-cs`␣stable"; done |
     sudo tee /etc/apt/sources.list.d/openmodelica.list
3 for deb in deb deb-src; do~echo "$deb␣http://build.
     openmodelica.org/apt␣`lsb_release␣-cs`␣release"; done |
     sudo tee /etc/apt/sources.list.d/openmodelica.list
```

Výpis 5.3: Příkazy výběru verzí sestavení programu OpenModelica

V této práci byl vybrán řádek s položkou *release*, který byl následně zkopírován pomocí pravého tlačítka myši do terminálu. Po vložení příkazu do terminálu byl příkaz potvrzen klávesou *Enter*.

Druhou sérií příkazů je importování GPG (GNU Privacy Guard) klíče, kterým byla podepsána příslušná verze sestavení instalovaného programu.

```
1 wget -q http://build.openmodelica.org/apt/openmodelica.asc
     -O- | sudo apt-key add -
2 # To verify that your key is installed correctly
3 apt-key fingerprint
4 # Gives output:
5 # pub   2048R/64970947 2010-06-22
6 #       Key fingerprint = D229 AF1C E5AE D74E 5F59  DF30 3
     A59 B536 6497 0947
7 # uid                  OpenModelica Build System
```

Výpis 5.4: Příkaz importování GNU Privacy Guard klíče

Třetím a posledním krokem instalace OpenModelicy do prostředí OS Ubuntu byla následující série příkazů:

```
1 sudo apt update
2 sudo apt install openmodelica
```

```
3 sudo apt install omlib-.* # Installs optional Modelica
    libraries (most have not been tested with OpenModelica)
```

Výpis 5.5: Příkaz aktualizace a instalace OpenModelicy pro OS Ubuntu

Kteroukoliv z výše uvedených sekvencí je možné zadat do terminálu naráz, a není tudíž nutné zadávat příkazy postupně, což značně urychluje instalaci. Na druhou stranu, pokud je zadána sekvence příkazů v jednom bloku, který je následně potvrzen klávesou enter, je nutné mít na paměti, že kterýkoliv dotaz položený uživateli rozlišuje malá a velká písmena (case sensitive). Tento případ nastane obzvlášť při zadávání poslední sekvence příkazů, kdy je uživatel vyzván k souhlasu s provedením změn a stažením některých knihovních souborů OpenModelicy. Uživatel má v tu chvíli na výběr z možností *[Y/n]*, pokud však zadá pouze znak *'y'* nikoliv *'Y'*, který následně potvrdí klávesou enter, dojde k přerušení vykonávání příkazu uvedeného na řádku 2 ve výpisu 5.5 a následující příkaz (příkazy) pokračující od řádku 3 a dále je ignorován.

Po úspěšné instalaci bylo prostředí OpenModelica připraveno k vytvoření jednoduchého modelu sumátoru pod OS Linux.

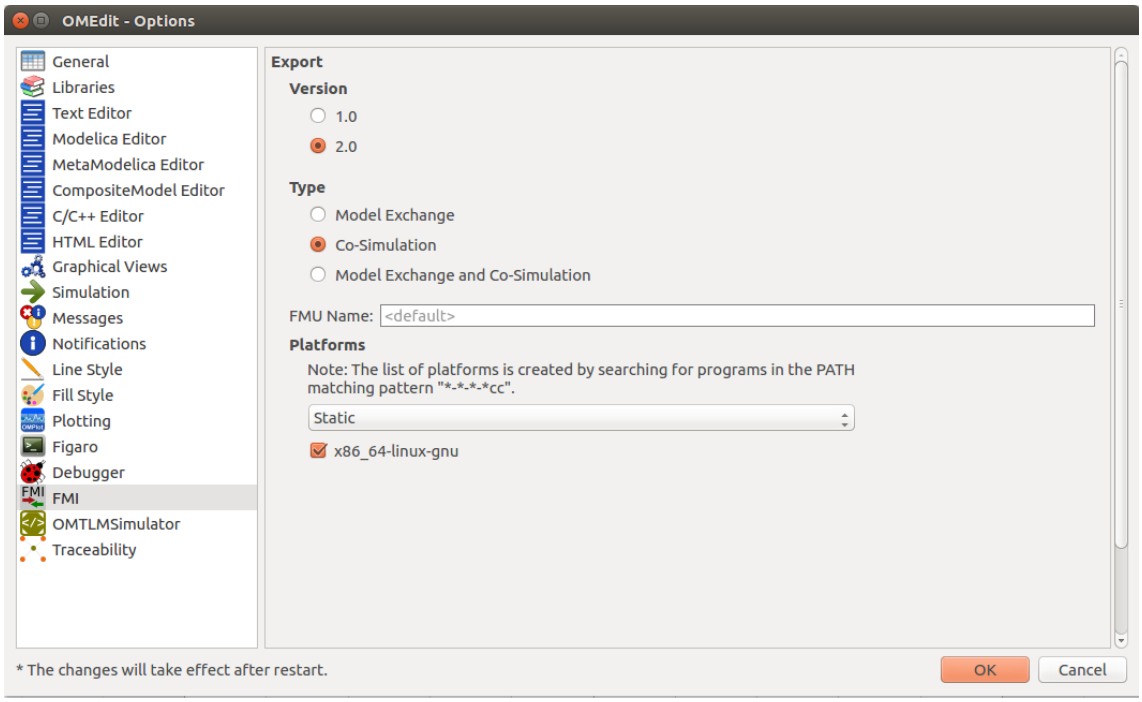
5.3.3 Model sumátoru v OpenModelica verze Linux

Zadrátovaný model sumátoru měl totožnou podobu, jaká je znázorněna na obrázku 5.2. Vytvořený model byl pro kontrolu odsimulován ve vestavěném simulačním prostředí OMEditoru. Nasimulované průběhy korespondovaly s 5.3. Před vygenerováním samotného modelu do Functional Mock-up Unit bylo nutné v menu OMEditoru - položka *Tools/Options* - záložka FMI nastavit verzi exportu FMU na hodnotu 2.0 a také změnit typ FMU na Co-simulation.

Po vyexportování modelu sumátoru pod operačním systémem Linux byly v adresářové struktuře modelu ve složce *sources* vyexportovány zdrojové soubory *ModelName.c, ModelName.h, ModelName.lib, Makefile.in* a další, které byly následně použity ke kompilaci FMU pro cílovou platformu.

5.4 Kompilace vygenerovaného FMU pro cílovou platformu

Cílovou platformou pro vyexportované FMU se stalo osmislotové CompactRIO se sériovým označením NI cRIO 9068. Uvnitř této platformy se nachází Zynq 7020, který disponuje dvoujádrovým procesor ARM Cortex A9 taktovaným na 667 MHz a především FPGA. Dále je k dipozici 512 MB operační paměti typu DRAM a 1 GB



Obr. 5.4: Nastavení exportu modelu Sumator

úložného místa. Nedílnou součástí cRIO 9068 je také Linux RealTime operační systém.

Aby bylo možné na této platformě simulovat FMU, bylo nutné nejdříve vyexportované FMU zkompilovat. OpenModelica při exportu FMU používá svůj přednastavený kompilátor, který bohužel žádným nastavením v prostředí OpenModelica není možné zaměnit za jiný typ. Bez toho, aniž by bylo vygenerované FMU zkompilováno pro cílovou platformu, není možné vytvářet instance z FMU, a ty následně testovat na reálném HW.

Ke kompilaci FMU pro jinou platformu bylo nutné použít odpovídající cross compiler. Jako vhodný cross compiler pro platformu NI cRIO 9068 byl zvolen *GNU C & C++ Compilers for ARMv7 Linux (Linux host) 2014-2016*, který je dostupný z [27]. Tento kompilátor je uložen rovněž na datovém nosiči, který je přiloženém v této práci.

Z uvedeného odkazu byl stažen soubor *oecore-x86_64-armv7a-vfp-neon-toolchain-2.0.sh*. Jedná se o Bourne Shell soubor, který interpretuje programovací jazyk tím, že čte příkazy z terminálu nebo souboru a provádí je. Posloupnost příkazů v souboru se také označuje jako skript. Spuštění staženého skriptu bylo provedeno z terminálu.

Po spuštění terminálu bylo nutné se nejdříve přesměrovat do místa uložení skriptu. Poté byl skript spuštěn příkazem uvedeným ve výpisu 5.6. Po potvrzení příkazu bylo možné nastavit umístění cílové složky, kam měl být Software Devolpment Kit (cross

compiler) nainstalován.

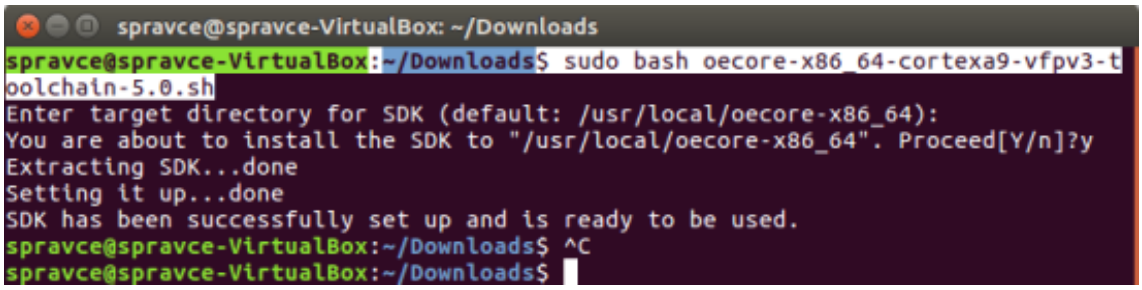
```
1 sudo bash oecore-x86_64-armv7a-vfp-neon-toolchain-2.0.sh
```

Výpis 5.6: Příkaz instalace GNU C & C++ cross compileru

V případě že by nebylo možné nainstalovat SDK cross compileru z důvodů nedostatečného oprávnění, je možné příslušné oprávnění změnit pomocí příkazu *chmod*, který je uveden ve výpisu 5.7.

```
1 chmod -x oecore-x86_64-armv7a-vfp-neon-toolchain-2.0.sh
```

Výpis 5.7: Změna přístupových práv k souboru cross compileru



Obr. 5.5: Instalace GNU C & C++ cross compileru z terminálu

5.4.1 Nastavení Eclipse studia

Vývojových prostředí, kterými lze zkompilovat vyexportované FMU pro jinou platformu, je velké množství. V zásadě všechny však používají stejnou metodiku a liší se pouze svým uživatelským rozhraním, které do značné míry závisí na použitém vývojovém prostředí.

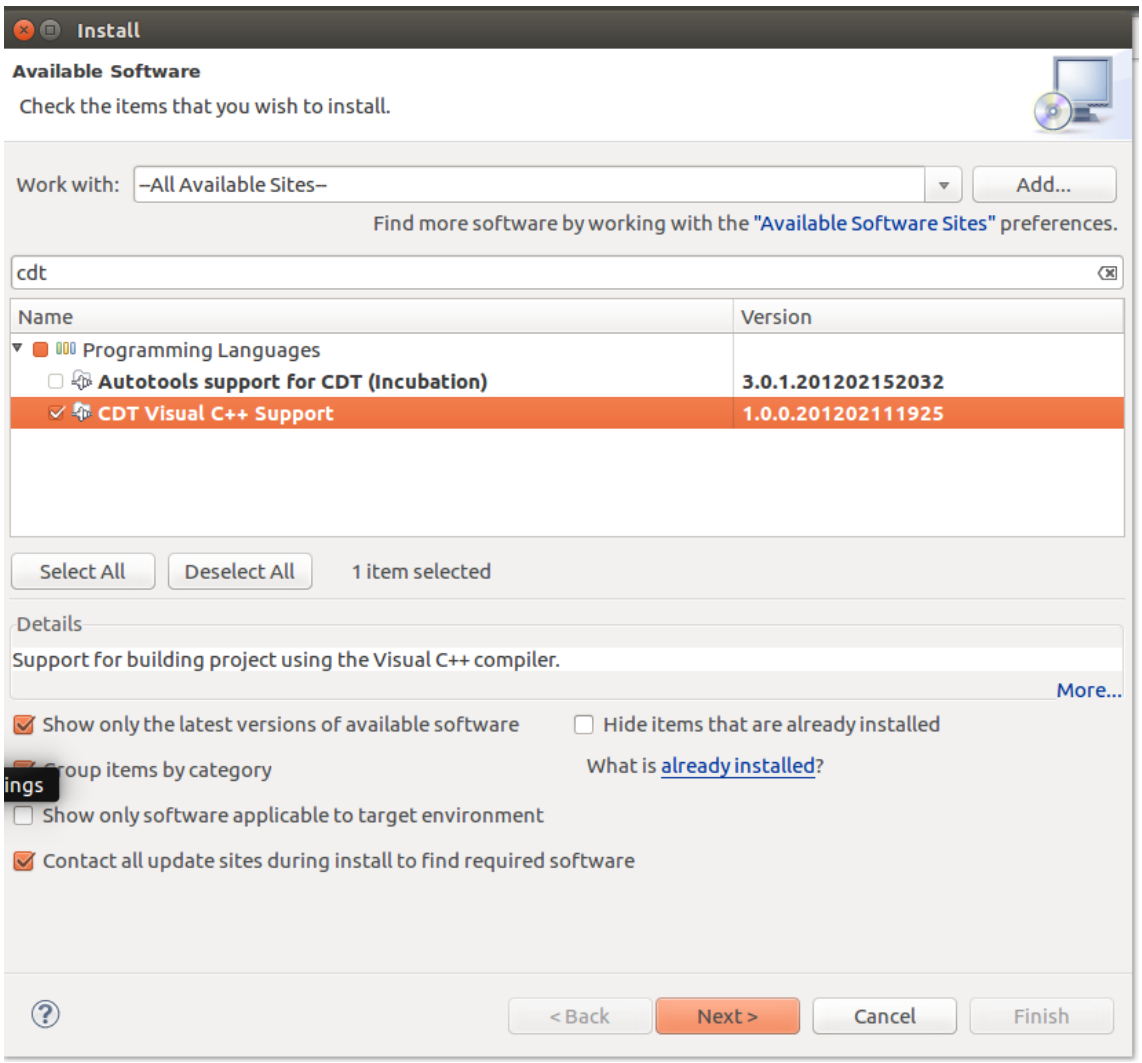
Pro prvotní zkompilování vygenerovaného FMU bylo použito vývojové prostředí *C/C++ Development Tools for NI Linux Real-Time, Eclipse Edition 2014-2016*, které je založeno na Eclipse studiu. Na stránkách NI byl nalezen návod dostupný z [28], který se zabývá úvodem do problematiky vývoje vlastních aplikací pro platformy NI Linux RealTime. Na této stránce je možné nalézt také pokyny pro instalaci Eclipse studia, jeho konfiguraci a postup vytvoření a spuštění vzorového projektu *Hello world*.

Před započetím instalace Eclipse studia pro OS Windows je nutné nejdříve nainstalovat Java SE 6 nebo její novější verzi. Tímto krokem je možné předejít mnohým problémům s instalací a provozem Eclipse studia. Eclipse studio 2014 je možné volně stáhnout z [29]. V případě této práce bylo zprvu provozováno Eclipse studio pod OS Linux Ubuntu. Instalace vývojového prostředí byla právě díky OS Linux

velmi usnadněna díky Ubuntu software center. Software center pracuje na základě dříve zmíněných repozitářů, ve kterých se požadovaná verze Eclipse studia také nacházela. Stačilo tedy vyhledat požadovanou verzi Eclipse studia a nechat si software automaticky nainstalovat. Díky dependency systému implementovaném v Linuxových systémech nebylo nutné potřebnou Javu 6 SE ručně doinstalovávat, neboť tento systém sám rozpoznává případný dodatečný software potřebný k instalaci hlavního SW, který potom sám doinstaluje.

Poté, co byl potřebný software nainstalován bez ohledu na OS, bylo nutné z prostředí Eclipse studia provést instalaci a aktualizaci CDT. Instalace a aktualizace CC++ Development Tool byla provedena podle části dokumentace nástroje Eclipse dostupné z [30]. Zjednodušený postup instalace CDT lze shrnout do těchto kroků:

1. Menu→Help→Install new software



Obr. 5.6: Instalace CDT Visual C++ Support z Eclipse studio

2. V rolovacím okně Work with vybrat –All Available Sites–
3. Do vyhledávacího pole zadat CDT → Hledat
4. Zaškrtnout CDT Visual CC++ Support
5. Přejít na Next
6. Potvrdit Instalaci

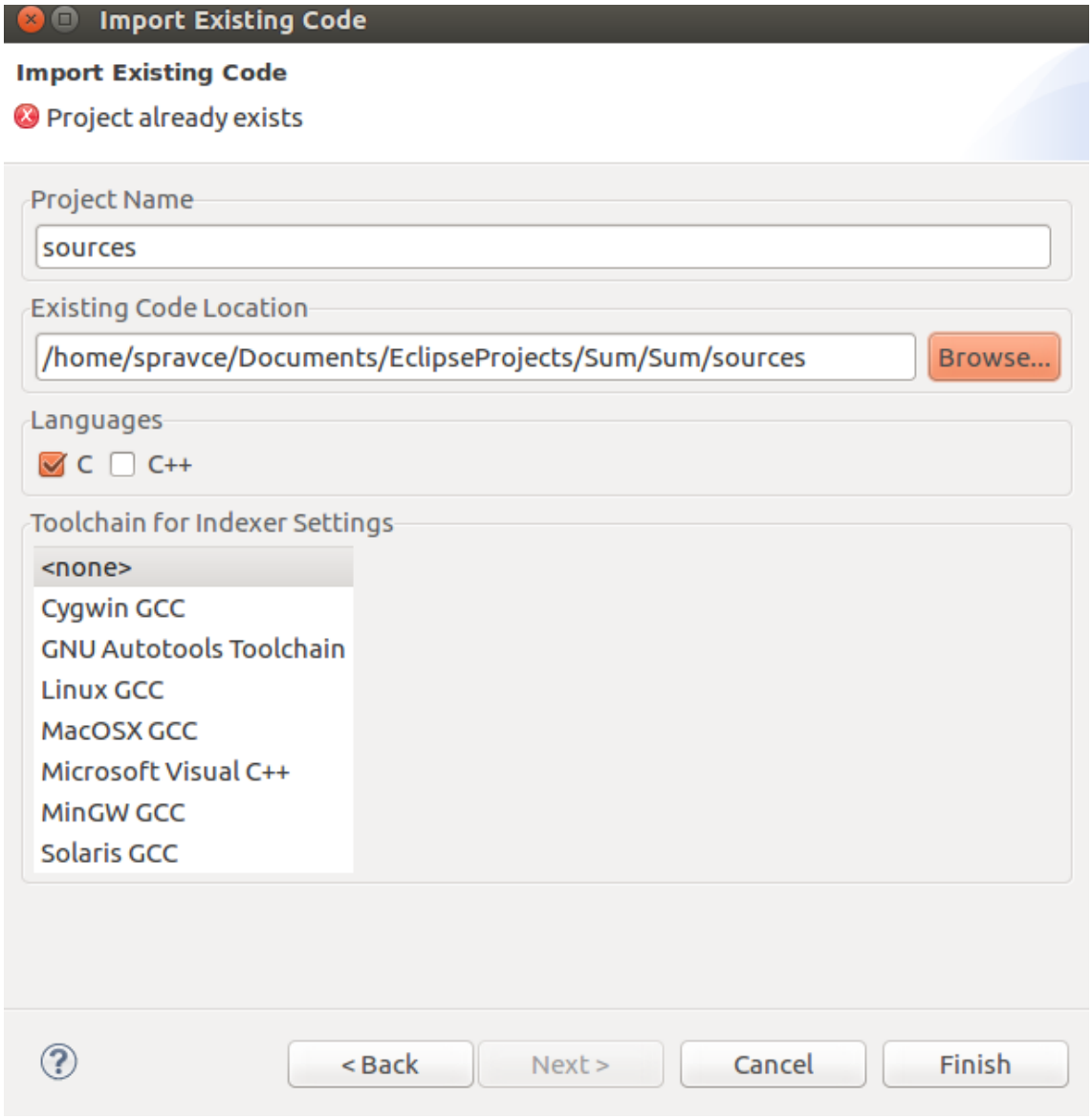
Po instalaci byla podle pokynů uvedených v návodu otestována funkčnost vzorového projektu *Hello world ANSI C project*, který byl dodán spolu s vývojovým studiem Eclipse. Před jeho sestavením (buildem) bylo nutné provést několik nastavení, která jsou detailně popsána v návodu.

5.4.2 Kompilace FMU v Eclipse studiu

Po úspěšném otestování vzorového příkladu *Hello world*, při kterém byla mimo jiné vytvořena SSH komunikace mezi PC a platformou cRIO, byla vytvořena kopie vygenerovaného FMU. U této kopie byla následně změněna přípona z **.fmu* na **.zip*. Zip archiv byl extrahován a všechny zdrojové soubory se tak staly přístupnými. Rozbalení archivu je nutné provést i v případě OS Linux, který má vestavěný průzkumník zip souborů v defaultním správci souborů, a to z důvodu importu některých souborů do nového projektu.

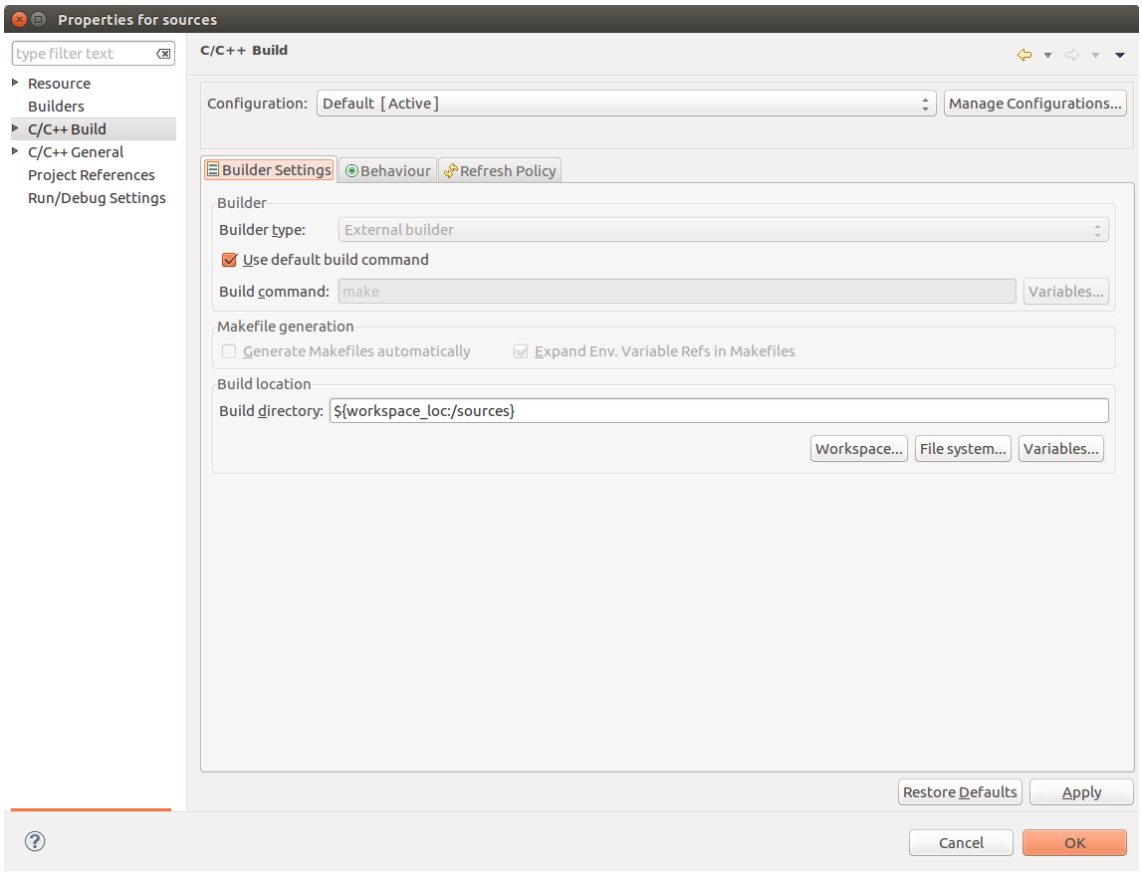
Po vytvoření kopie modelu FMU byl v Eclipse studiu vytvořen nový projekt typu *Makefile Project with Existing code*. Při založení projektu bylo nutné vybrat extrahovanou složku *sources* z modelu FMU, ze které byl proveden import zdrojových souborů do nového projektu. Názorný příklad je uveden na obrázku 5.7. Po vytvoření projektu a naimportování souborů z adresáře *sources* obsaženém v extrahovaném adresáři kopie modelu FMU bylo nastaveno chování kompilace překladu a slinkování projektu. Veškerá nastavení byla provedena v okně nastavení projektu, které je možné zvolit výběrem položky *Project settings* v rolovací nabídce vyvolané kliknutím pravého tlačítka myši na složku aktivního projektu umístěného v hierarchii adresářové struktury samotného vývojového prostředí.

V položce *C/C++ Build* v kartě *Builder Settings* bylo zaškrtnuto nastavení *Use default build command* (viz obrázek 5.8). Na kartě *Behaviour* bylo zaškrtnuto *Stop on first build error*, *Build (Incremental build)* a položka *Clean* (viz obrázek 5.9). Pro zrychlení kompilace je možné použít více vláken a využít tak více jader procesoru PC. Poslední nastavení projektu, které bylo nutné provést, se nacházelo v položce *C/C++ Build* → *Settings* v kartě *Binnary parsers*. Zde byla zatržena pouze položka *Elf Parser* (viz obrázek 5.10). To, jak bude makefile projekt přeložen, čím bude zkompilován, a jakým způsobem bude slinkován, řídí soubor makefile, který byl vyexportován spolu se zdrojovými soubory z nástroje OpenModelica. Vyexportovaný soubor z nástroje OpenModelica nese název *Makefile.in* a jeho obsah



Obr. 5.7: Import zdrojových souborů do Makefile projektu

zachycuje obrázek 5.11. Kromě souboru *Makefile.in* byl ve zdrojových souborech obsažen také soubor *NazevModelu.makefile*, který importuje soubor Makefile pomocí příkazu *include Makefile*. Aby mohlo Eclipse studio při sestavení projektu pracovat s makefile, bylo nutné původně vygenerovaný soubor *Makefile.in* buď přejmenovat na *Makefile* bez přípony, anebo lépe vytvořit kopii původního souboru *Makefile.in* a následně vytvořenou kopii přejmenovat na *Makefile* bez přípony. Pokud by nebyl soubor přejmenován, nemohlo by vývojové prostředí tento soubor nalézt a provést kompilaci projektu. Kromě přejmenování původního souboru makefile bylo nutné provést ještě několik změn uvnitř samotného souboru. Popis všech změn, které byly před kompilací v makefile souboru provedeny, jsou uvedeny v kapitole 5.4.4.



Obr. 5.8: Nastaveni Makefile projektu 1 z 3

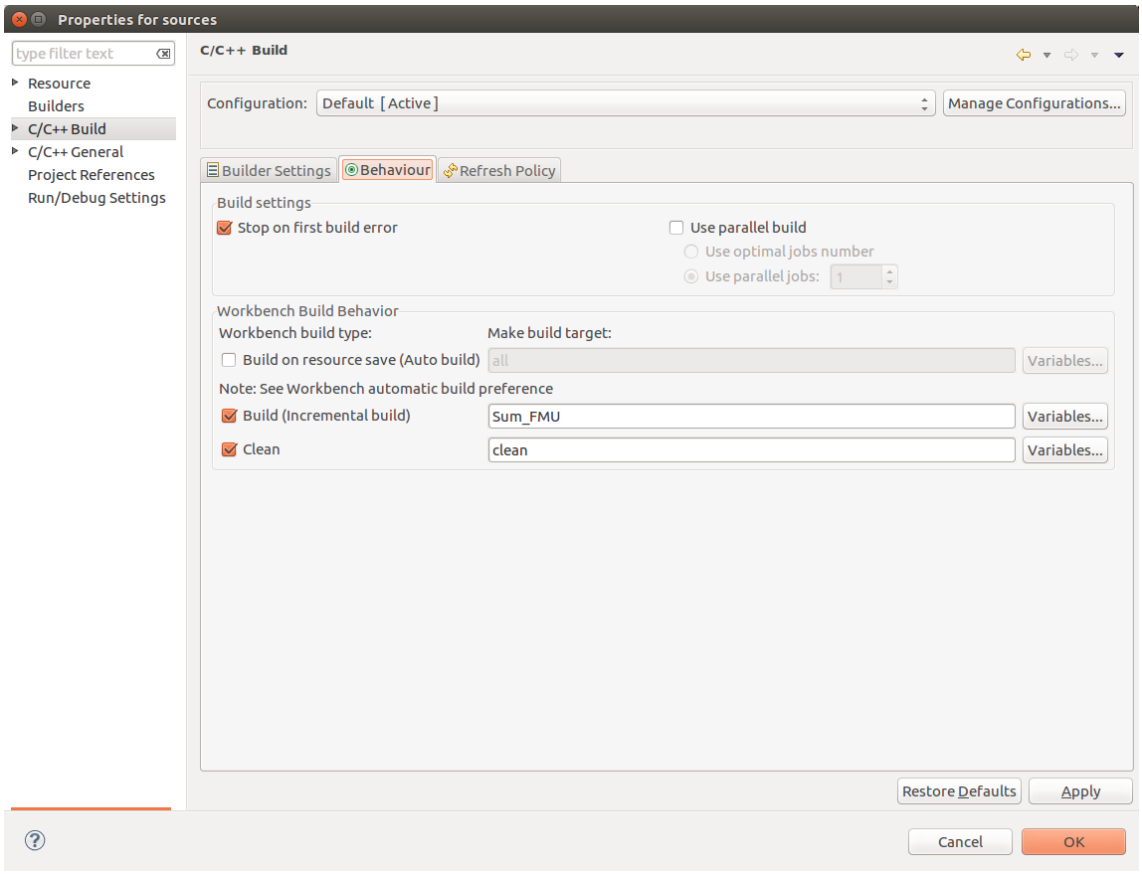
Další ruční úprava, která byla nezbytná ke kompilaci FMU, spočívala v přejmenování souboru *config.log* na název *config.sub*. Do tohoto souboru byly při sestavování projektu zapisovány některé důležité informace.

Aby se v předešlo pozdějším komplikacím při překladu projektu, které se při vypracování této práce později vyskytly, bylo důležité zajistit dostatečná uživatelská oprávnění k adresáři projektu. Změna uživatelského oprávnění složky makefile projektu byla provedena pomocí příkazu uvedeného ve výpisu 5.8. Ještě před zadáním tohoto příkazu do terminálu bylo nutné se přesměrovat do místa uložení adresáře makefile projektu s použitím známých příkazů *ls, ls -l, cd* a popřípadě i *cd...*

```
1 sudo chmod 777 Nazev_slozky
```

Výpis 5.8: Změna přístupových práv k adresáři makefile projektu

Teprve po přejmenování makefile a konfiguračního souboru a po úpravě přístupových práv ke složce makefile projektu, bylo možné zdrojové soubory bez problémů zkompilovat pro cílovou platformu. Při kompilaci byla v adresářové struktuře projektu ve vývojovém prostředí vytvořena dočasná složka *binaries*, která obsahovala



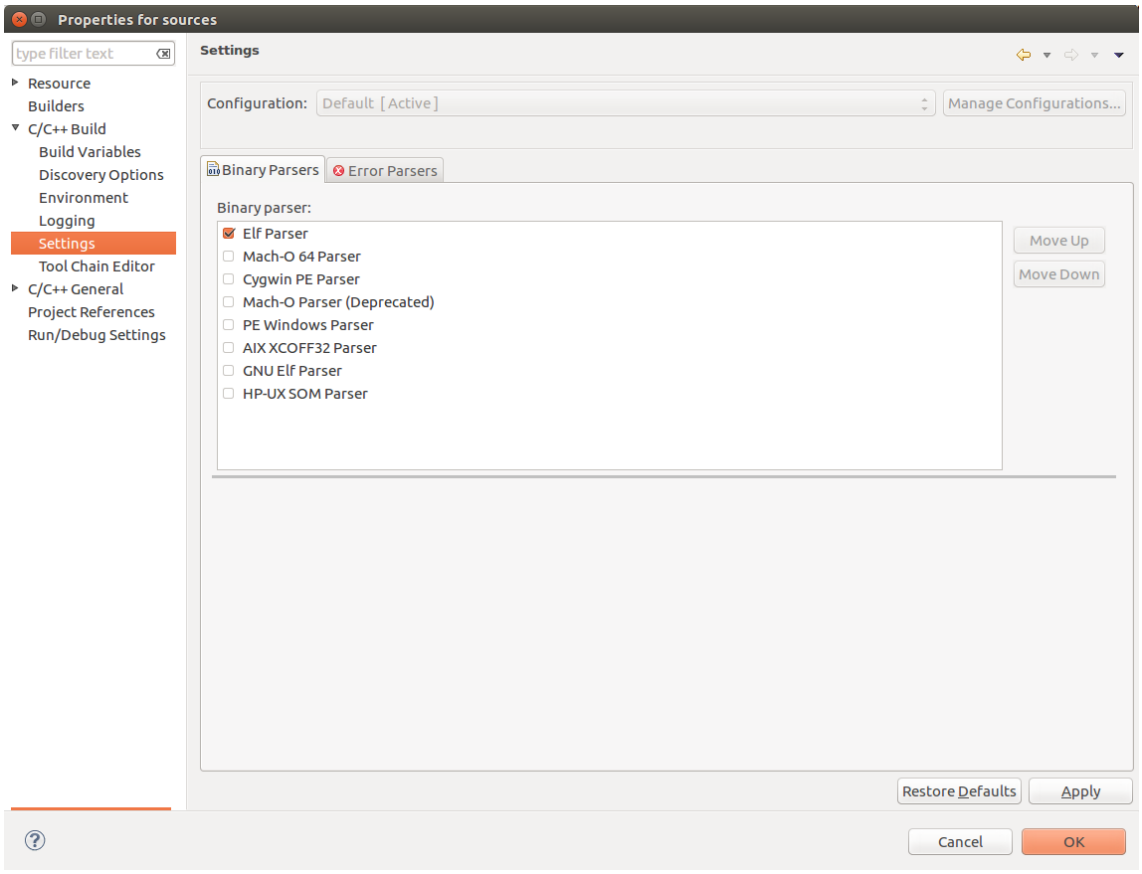
Obr. 5.9: Nastaveni Makefile projektu 2 z 3

soubor ve formátu *Nazev_Modelu.so*. Tento soubor používá pouze samotné vývojové prostředí Eclipse za účelem ladění a spuštění spustitelného souboru na cílové platformě.

Je důležité podotknout, že kromě vytvořených binárních souborů v prostředí Eclipse workspace, byly také aktualizovány binární soubory v adresáři dříve extrahovaného archivu modelu FMU. Proto bylo nutné změnit přístupová práva, a proto bylo taktéž důležité před vytvořením makefile projektu vytvořit kopii FMU, tu posléze přejmenovat na zip archiv, a ten následně extrahovat. Tento postup zajistil, že bude zachován původní soubor FMU obsahující model beze změny s tím, že všechny provedené změny budou provedeny pouze v jeho kopii.

5.4.3 Kompilace FMU pomocí make

Druhou možností, jak zkompilovat vyexportované FMU pro jinou cílovou platformu, bylo spuštění samotného souboru makefile ze systémového terminálu. Tento způsob měl oproti kompilaci FMU z prostředí vývojového studia Eclipse uvedeného v ka-



Obr. 5.10: Nastavení Makefile projektu 3 z 3

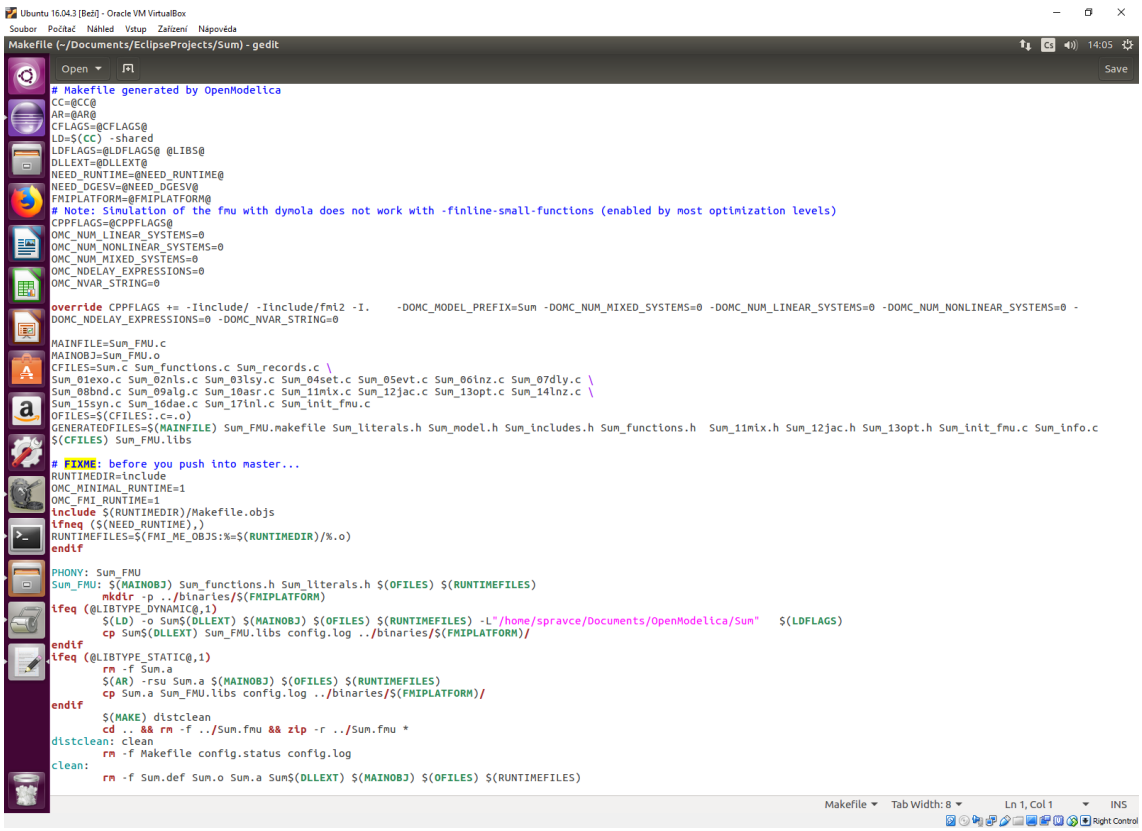
pitole 5.4.2 několik výhod, ale zároveň i nevýhod. V některých případech by bylo přesnější označit některé nevýhody spíše za vlastnosti.

Jedním ze základních předpokladů pro práci s makefile z prostředí terminálu je alespoň základní znalost Linuxových systémů. Dalším předpokladem by měla být také znalost terminálových příkazů, které je možné najít také v tzv. Linux man pages.

Jak již bylo uvedeno dříve, soubor makefile vždy řídí a automatizuje proces kompilace, tudíž nezáleží na tom, kterým vývojovým nástrojem nebo prostředím je spuštěn. Z prostředí terminálu je soubor Makefile (bez přípony) možné spustit příkazem *make*.

Výhodou spuštění makefile souboru z prostředí terminálu je, že není nutné instalovat jiný software. Stačí vyexportované FMU znovu uložit a extrahovat v průzkumníku souborů a následně se přesměrovat v terminálu do extrahované složky *sources*, která obsahuje zmíněný soubor *Makefile.in*. Soubor *Makefile.in* musí být stejně jako v případě kompilace v prostředí Eclipse přejmenován, aby neobsahoval příponu.

Posledním krokem kompilace FMU z prostředí terminálu bylo spuštění makefile



Obr. 5.11: Vyexportovaný Makefile.in z nástroje OpenModlica

souboru pomocí příkazu uvedeného v následujícím výpisu 5.9.

```
1 make
```

Výpis 5.9: Spuštění kompilace makefile projektu z terminálu

Za příkazem *make* může následovat parametr v podobě kteréhokoliv řetězce, který je uveden v Makefile souboru, a za nímž se nachází dvojtečka. Jedním ze způsobů, kterými lze přiblížit význam parametru za příkazem make, je návěští. Navěští, které je voláno po zadání příkazu *make* bez parametru, se nazývá *all:*. V něm je v podobě příkazů určeno, jakým způsobem kompilace proběhne.

Výsledek kompilace z nástroje Eclipse a ze systémového terminálu v případě použití příkazu *make* je totožný. Výhodou vývojového studia Eclipse může být větší přehlednost projektu a lepší report případných chyb vzniklých při kompilaci, které jsou pro méně znalé uživatele procesu *make* výhodné. S použitím Eclipse studia je ale nutné počítat s větším množstvím nastavení, než v případě kompilace FMU prostřednictvím systémového terminálu a příkazu *make*.

Ani jedna z uvedených metod kompilace by však nebyla funkční, aniž by byl soubor *Makefile* adekvátně upraven. Úpravě *Makefile* souboru se věnuje odstavec 5.4.4,

ve kterém jsou popsány jednotlivé změny procesu kompilace FMU pro platformu cRIO 9068.

5.4.4 Úprava souboru makefile

Původně vygenerovaný soubor *Makefile.in* je zachycen na obrázku 5.11 a tvoří také obsah přiloženého CD. Při bližším pohledu do souboru *Makefile.in* bylo možné v první části souboru spatřit deklaraci několika proměnných.

```
1  # Makefile generated by OpenModelica
2  CC=@CC@
3  AR=@AR@
4  CFLAGS=@CFLAGS@
5  LD=$(CC) -shared
6  LDFLAGS=@LDFLAGS@ @LIBS@
7  DLLEXT=@DLLEXT@
8  NEED_RUNTIME=@NEED_RUNTIME@
9  NEED_DGESV=@NEED_DGESV@
10 FMIPLATFORM=@FMIPLATFORM@
11 # Note: Simulation of the fmu with dymola does not work
      with -finline-small-functions (enabled by most
      optimization levels)
12 CPPFLAGS=@CPPFLAGS@
13 OMC_NUM_LINEAR_SYSTEMS=0
14 OMC_NUM_NONLINEAR_SYSTEMS=0
15 OMC_NUM_MIXED_SYSTEMS=0
16 OMC_NDELAY_EXPRESSIONS=0
17 OMC_NVAR_STRING=0
```

Výpis 5.10: Původní deklarace proměnných v souboru Makefile.in

První proměnná, která byla v přejmenované kopii souboru *Makefile* modifikována, se nacházela na třetím řádku tohoto souboru a také na tomtéž řádku výpisu 5.10. Její původní tvar byl upraven tak, aby proměnná *CC* měla v makefile význam přístupové cesty k externímu compileru. Tento compiler byl již v OS Linux nainstalován podle postupu uvedeného v kapitole 5.4, pomocí příkazu uvedeného ve výpise 5.6 shodné kapitoly.

Deklarace proměnné *AR* byla odstraněna a zůstala zde pouze definice této proměnné. V proměnné *CFLAGS* byly nastaveny parametry kompilace. Proměnná *LD* určovala, jakým způsobem budou načteny knihovny. V *LDFLAGS* byly nastaveny parametry pro načtené knihovny. Proměnná *DLLEXT* určovala, jakého typu bude externí dynamicky linkovaná knihovna. Hodnoty proměnných *NEED_RUNTIME*

a *NEED_DGESV* byly nastaveny na *1* z důvodů řízení procesu kompilace. Proměnná *FMIPLATFORM* určovala OS cílové platformy, pro který mělo být FMU zkompilováno. V *CPPFLAGS* byla provedena nastavení precompileru. Zbytek proměnných zůstal v původní nezměněné podobě. Hodnoty jednotlivých proměnných z první sekce souboru makefile jsou uvedeny ve výpisu 5.11.

```
1  # Makefile generated by OpenModelica
2  CC=/usr/local/oecore-x86_64/sysroots/x86_64-nilrtsdk-linux
      /usr/bin/armv7a-vfp-neon-nilrt-linux-gnueabi/arm-nilrt-
      linux-gnueabi-gcc
3  AR=
4  CFLAGS=-g -s -O2 -fno-stack-protector -Wno-parentheses-
      equality -Wno-unused-variable -fPIC
5  LD=$(CC) -shared -g
6  LDFLAGS= -shared -static-libgcc -lm -lpthread -shared
7  DLLEXT=.so
8  NEED_RUNTIME=1
9  NEED_DGESV=1
10 FMIPLATFORM=linux64
11 # Note: Simulation of the fmu with dymola does not work
      with -finline-small-functions (enabled by most
      optimization levels)
12 CPPFLAGS= -fdollars-in-identifiers -DOMC_MINIMAL_RUNTIME=1
       -DCMINPACK_NO_DLL=1
13 OMC_NUM_LINEAR_SYSTEMS=0
14 OMC_NUM_NONLINEAR_SYSTEMS=0
15 OMC_NUM_MIXED_SYSTEMS=0
16 OMC_NDELAY_EXPRESSIONS=0
17 OMC_NVAR_STRING=0
```

Výpis 5.11: Modifikovaná deklarace proměnných v souboru Makefile

Poslední změny, které byly v makefile provedeny, se týkaly především řízení kompilace FMU. Původní proces kompilace měl podobu uvedenou ve výpisu 5.12.

```
39 PHONY: Sum_FMU
40 Sum_FMU: $(MAINOBJ) Sum_functions.h Sum_literals.h $(
      OFILES) $(RUNTIMEFILES)
41   mkdir -p ../binaries/$(FMIPLATFORM)
42 ifeq (@LIBTYPE_DYNAMIC@,1)
43   $(LD) -o Sum$(DLLEXT) $(MAINOBJ) $(OFILES) $(
        RUNTIMEFILES) -L"/home/spravce/Documents/OpenModelica/
        Sum"   $(LDFLAGS)
```

```
44    cp Sum$(DLLEXT) Sum_FMU.libs config.sub ../binaries/$(
         FMIPLATFORM)/
45  endif
46  ifeq (@LIBTYPE_STATIC@,1)
47    rm -f Sum.a
48    $(AR) -rsu Sum.a $(MAINOBJ) $(OFILES) $(RUNTIMEFILES)
49    cp Sum.a Sum_FMU.libs config.sub ../binaries/$(
         FMIPLATFORM)/
50  endif
51    $(MAKE) distclean
52    cd .. && rm -f ../Sum.fmu && zip -r ../Sum.fmu *
53  distclean: clean
54    rm -f Makefile config.status config.log
55  clean:
56    rm -f Sum.def Sum.o Sum.a Sum$(DLLEXT) $(MAINOBJ) $(
         OFILES) $(RUNTIMEFILES)
```

Výpis 5.12: Původní proces kompilace FMU v souboru Makefile

Změny provedené na řádcích 42 a 46 ve výpisu 5.13 byly provedeny z důvodu provedení části kompilace, kterou bylo nutné provést vždy při spuštění kompilace. Použití komentáře na řádcích 47 a 54 totožného výpisu způsobilo přeskočení příkazů uvedených na těchto řádcích při spuštění procesu kompilace. Důležité bylo především zabránění odstranění souborů *Makefile*, *config.status* a *config.log*. Tyto soubory by včetně modifikovaného souboru makefile byly po spuštění kompilace smazány a pro další kompilaci by bylo nutné vytvořit a následně modifikovat nový makefile. Upravená část procesu kompilace FMU je uvedena ve výpisu 5.13.

```
39  PHONY: Sum_FMU
40  Sum_FMU: $(MAINOBJ) Sum_functions.h Sum_literals.h $(
       OFILES) $(RUNTIMEFILES)
41    mkdir -p ../binaries/$(FMIPLATFORM)
42  ifeq (1,1)
43    $(LD) -o Sum$(DLLEXT) $(MAINOBJ) $(OFILES) $(
         RUNTIMEFILES) -L"/home/spravce/Documents/OpenModelica/
         Sum"   $(LDFLAGS)
44    cp Sum$(DLLEXT) Sum_FMU.libs config.sub ../binaries/$(
         FMIPLATFORM)/
45  endif
46  ifeq (0,1)
47    # rm -f Sum.a
48    $(AR) -rsu Sum.a $(MAINOBJ) $(OFILES) $(RUNTIMEFILES)
```

```
49    cp Sum.a Sum_FMU.libs config.sub ../binaries/$(
          FMIPLATFORM)/
50  endif
51    $(MAKE) distclean
52    cd .. && rm -f ../Sum.fmu && zip -r ../Sum.fmu *
53  distclean: clean
54    # rm -f Makefile config.status config.log
55  clean:
56    rm -f Sum.def Sum.o Sum.a Sum$(DLLEXT) $(MAINOBJ) $(
          OFILES) $(RUNTIMEFILES)
```

Výpis 5.13: Modifikované proces kompilace FMU v souboru Makefile

Poté, co bylo vygenerované FMU z nástroje OpenModelica zkompilováno pro cílovou platformu cRIO 9068, byl zkompilován binární soubor *Sum.so* následně použitý k vytvoření a simulaci instance FMU na cílové platformě cRIO.

5.5 Vytvoření a simulace instance FMU

K vytvoření a následné simulaci instance FMU na RealTime platformě cRIO bylo použito stejného vývojového prostředí Eclipse jako v případě kompilace FMU. V tomto vývojovém prostředí byl založen nový spustitelný projekt jazyka C. Vytvořený projekt bylo dále nutné správně nakonfigurovat, aby bylo následně možné vytvořit instanci FMU a spustit ji na RealTime části platformy cRIO. Část nastavení projektu vychází z návodu s titulem *Getting Started with C/C++ Development Tools for NI Linux Real-Time, Eclipse Edition* nalezeného na webových stránkách NI dostupných z [28].

5.5.1 Založení spustitelného projektu jazyka C

Postup vytvoření nového spustitelného projektu jazyka C korespondoval se sekcí číslo 4 označené jako Creating a C/C++ Project, která byla uvedena v návodu umístěného na webových stránkách společnosti NI. Návod je dostupný z [28].

Oproti postupu uvedeném v sekci číslo 4 tohoto návodu byla provedena změna typu projektu, kde byla pod složkou *Executable* zvolena položka *Empty project*, namísto *Hello World ANSI C Project*.

V bodě číslo 13, který se věnuje výběru *Cross compiler prefix* uvedenému v téže sekci, byla vybrána volba *ARM-based targets, 2014 software stack*. Její hodnota *arm-nilrt-linux-gnueabi-* včetně pomlčky na konci byla vložena do stejnojmenného řádku dialogového okna *Cross GCC Command*.

Jako *Cross compiler path* byla vybrána stejná verze jako v případě *Cross compiler prefix*, tedy *ARM-based targets, 2014 software stack*, a podle následující sekvence `\<NationalInstruments>\Eclipse\14.0\arm\sysroots\i686-nilrtsdk-mingw32\usr\bin\armv7a-vfp-neon-nilrt-linux-gnueabi` bylo v adresářové struktuře PC označeno umístění compileru.

Výběrem a zadáním jiných parametrů v dialogovém okně *Cross GCC Command* je možné provést nastavení pro kteroukoliv jinou RealTime platformu.

5.5.2 Nastavení projektu

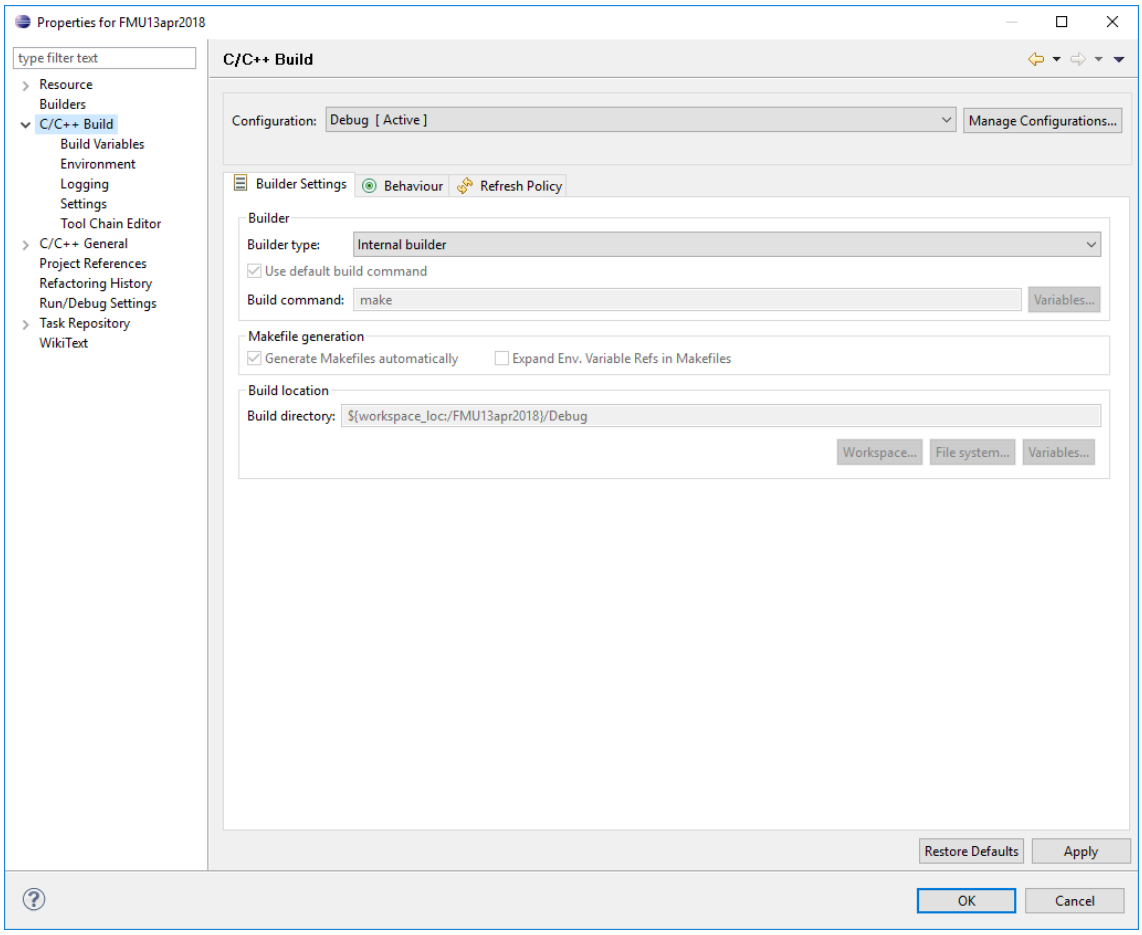
Ještě před spuštěním překladu uživatelem vytvořeného řídicího programu bylo nutné provést některá nastavení vytvořeného projektu. Kontextová nabídka nastavení projektu byla vyvolána označením vrcholové složky projektu z okna *Project Explorer* pravým tlačítkem myši, kde byla vybrána položka *Properties*. Jiný způsob, kterým je možné vyvolat okno nastavení projektu, je stisknutím kombinace kláves *Alt+Enter*.

V levém sloupci nabídky nastavení projektu byla vybrána položka *C/C++ Build*. Na kartě *Builder Settings* byl z rozbalovací nabídky *Builder type* vybrán *Internal builder* (viz obrázek 5.12). Pro zrychlení procesu překladu projektu je možné v záložce *Behaviour* navolit maximální počet vláken, které smí Eclipse studio pro překlad projektu využít. V případě, že je k dispozici výkonný PC, je vhodné povolit více vláken. Při řešení této práce však využití tohoto nastavení nebylo nezbytné.

Další velmi důležitá nastavení překladu projektu byla provedena v položce *Settings*, která je dostupná po rozbalení rolovací nabídky položky *C/C++ Build* v levém sloupci nastavení projektu. Po otevření nabídky *Settings* pod položkou *C/C++ Build* je k dispozici adresářový strom, jehož jednotlivé položky představují nastavení překladu projektu. Adresářový strom byl rozdělen na celkem čtyři části. První část, která je zachycena na obrázku 5.13, se nazývá *Cross Settings*. Atributy této položky říkají, jaký překladač má být pro překlad projektu použit, a také definují jeho absolutní přístupovou cestu na disku. Výběr použitého Cross Compileru bylo možné provést již při založení projektu. Pokud by však byl, ať už omylem, anebo cíleně vybrán jiný typ compileru, na tomto místě je možné použitý compiler libovolně změnit.

Z druhé části adresářového stromu nazvané *Cross GCC Compiler* byla důležitá úprava atributu *Other flags* pod položkou *Miscellaneous* (viz obrázek 5.14).

Za původní řetězec *-c -fmessage-length=0* byl přidán další *-mfpu=vfpv3 -mfloat-abi=softfp*, kterému musí nutně předcházet mezera. Přidaný řetězec zlepšuje výkonnost provádění operací s plovoucími desetinnými čísly. Zlepšení výkonnosti operací s plovoucími desetinným čísly je možné provést pouze v případě použití platformy založené na procesoru ARM.



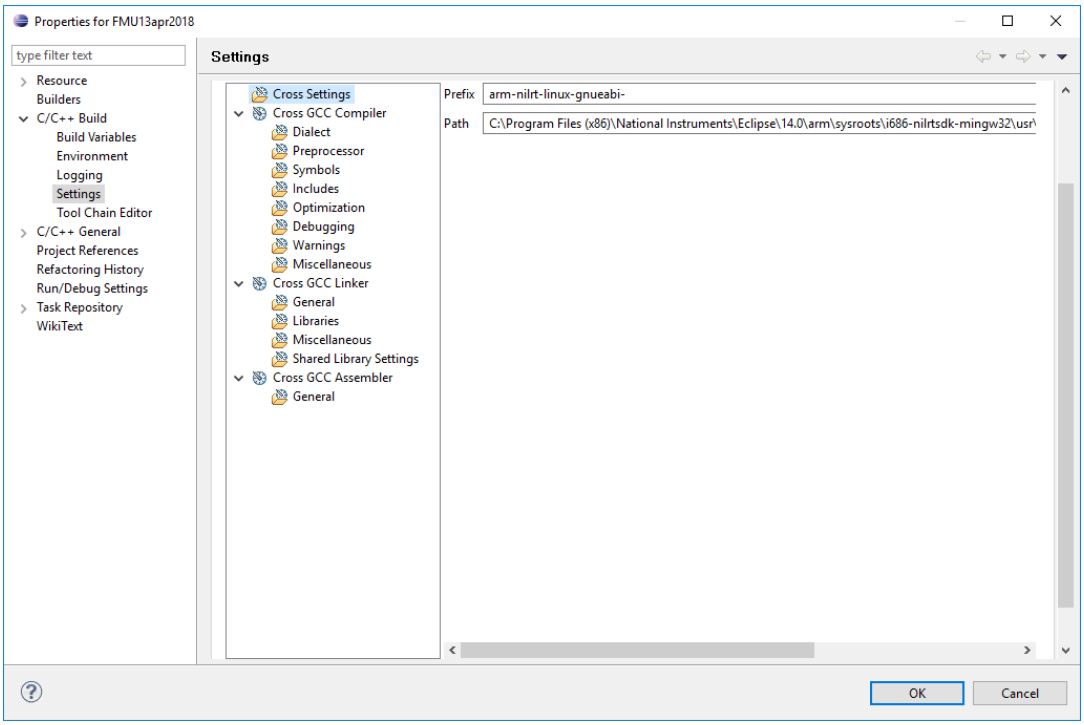
Obr. 5.12: Obecné nastavení projektu

Pro účely překladu programu uvedeného ve výpise 5.14, který byl vytvořen k otestování funkčnosti SSH komunikace mezi PC a cRIO, byla všechna potřebná nastavení projektu hotova. K přenosu dat mezi PC a vzdáleným systémem byl vytvořen SSH komunikační kanál, jemuž se věnuje následující bod kapitoly.

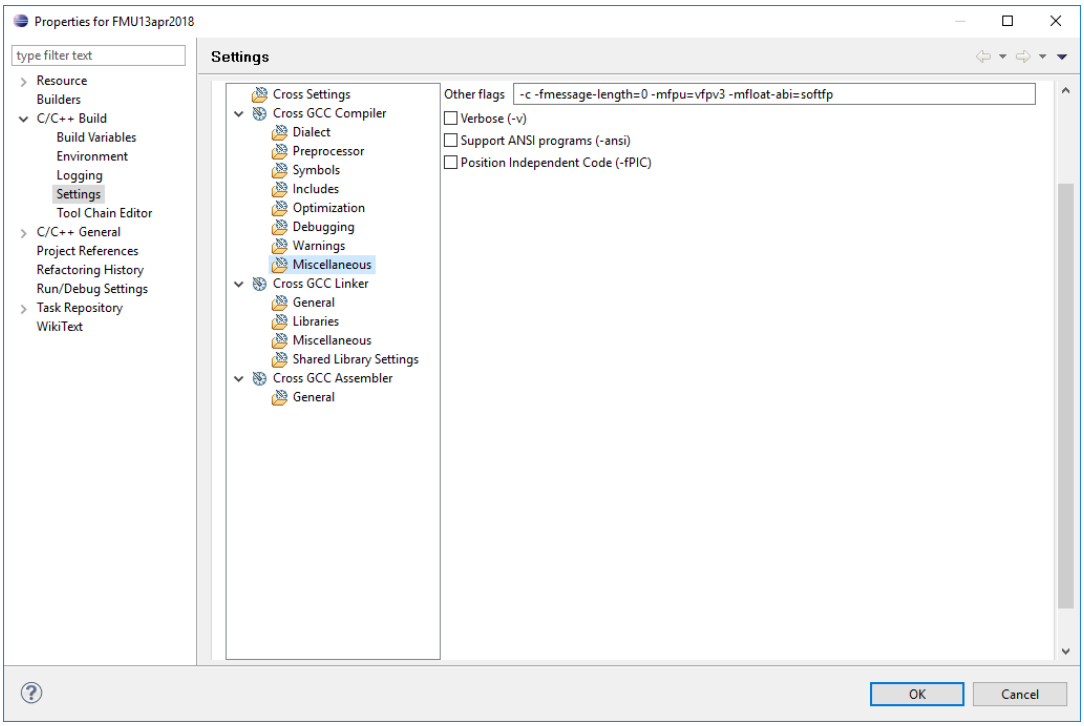
5.5.3 Konfigurace SSH komunikace s cRIO

Vytvoření SSH komunikace mezi Elcipse studiem a RealTime částí cRIO 9068 bylo provedeno na základě 6. sekce výše uvedeného návodu. Vytvoření SSH komunikace lze shrnout do sekvence následujících kroků:

1. Připoj cRIO ethernetovým kabelem k PC
2. Pomocí NI-MAX ověř komunikaci PC s cRIO
3. Z prostředí NI-MAX nastav uživatelské jméno a heslo
4. Dále pokračuj s konfigurací v Eclipse
5. V Menu Window→Open Perspective→Other vyvolej dialogové okno Open



Obr. 5.13: Výběr Cross Compileru projektu



Obr. 5.14: Nastavení zlepšení výkonnosti operací s pohyblivou desetinnou čárkou

Perspective

6. Z dialogového okna Open Perspective vyber Remote System Explorer
7. Vyber položku Define a connection to remote system
8. Vyber SSH only
9. Pojmenuj připojení a vyber hosta (platforma cRIO)
10. Pravým tlačítkem myši klikni na vytvořené připojení a z kontextového menu vyber connect
11. Pokud budeš dotázán, zadej uživatelské jméno a heslo pro přístup do platformy cRIO

Po vytvoření SSH komunikačního rozhraní byl vytvořen nový soubor *main.c*, ve kterém byl vytvořen jednoduchý testovací program, který je uveden ve výpisu 5.14.

Jak je možné vidět z výpisu 5.14, jednalo se o program, který měl při spuštění vypsat řádek textu do terminálu vývojového prostředí. Smyslem vytvoření tohoto programu bylo jednoduché otestování funkčnosti komunikace mezi Eclipse studiem a platformou cRIO.

Nyní byl vytvořený testovací program sestaven (build) a mohl být dále otestován na reálném HW. Před spuštěním testovacího programu musela být provedena nastavení ve vzdálené platformě cRIO pomocí procesu vzdálené konfigurace.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(viod){
5   printf("\nSSH␣communications␣test\n");
6   return 0;
7 }
```

Výpis 5.14: Program testu SSH komunikace mezi Eclipse studiem a cRIO

5.5.4 Spuštění projektu jazyka C na NI Linux RealTime

Vzdálená konfigurace cRIO byla provedena podle sekce číslo 7 *Running a C/C++ Executable on Your NI Linux Real-Time Target* uvedené v návodu *Getting Started with C/C++ Development Tools for NI Linux Real-Time, Eclipse Edition*, který je dostupný z [28].

Při vytvoření nové konfigurace spuštění testovacího programu bylo vybráno již nakonfigurované SSH spojení se stanicí cRIO. V adresářově struktuře vzdálené stanice byl vybrán adresář *My Home*, ve kterém byla vytvořena nová složka pojmenovaná stejně jako název projektu. Tato složka slouží při procesu spuštění testovacího programu jako úložiště zdrojového nebo zdrojových souborů ve vzdálené stanici. Při

každém novém spuštění testovacího programu jsou soubory, které byly změněny ve vytvořeném adresáři v systému cRIO, aktualizovány.

Kromě spuštění vytvořeného programu na cílové platformě bylo možné vytvořený program také ladit. Pro účely ladění vytvořeného programu bylo vytvořeno nové nastavení vzdálené konfigurace z menu *Run→Debug Configurations*, která byla pojmenována *FMU13apr2018 Debug*. V kartě konfigurace režimu ladění byl v poli *GDB debugger* učiněn výběr debugeru *arm-nilrt-linux-gnueabi-gdb.exe* ze systémové složky `<NationalInstruments>\Eclipse\14.0\arm\sysroots\i686-nilrtsdk-mingw32\usr\bin\armv7a-vfp-neon-nilrt-linux-gnueabi`. Tento debugger byl podle výše zmíněného návodu určen pro *ARM-based targets, 2014 software stack*, a tedy i pro cílovou platformu cRIO 9068, která byla použita v této práci.

Po spuštění vytvořeného programu pomocí tlačítka *Run* byl do okna terminálu vývojového prostředí Eclipse proveden výpis řetězce uvedeného v testovacím programu 5.14. S úspěchem byla otestována SSH komunikace mezi vývojovým prostředím Eclipse (PC) a systém cRIO.

Při spuštění debugovacího režimu testovacího programu došlo ve vývojovém prostředí Eclipse k automatickému přepnutí perspektivy z CC++ do perspektivy Debug. Rozložení uživatelského rozhraní perspektivy debugu bylo obdobné perspektivě CC++. Navíc se zde nacházely kontrolní prvky jako krokování programu, procházení programu, nastavení break pointů a tlačítka spuštění programu, které byly použity k ovládání krokování programu. Díky této vlastnosti mohl být program prováděn po jednotlivých řádcích a s pomocí monitorovacích nástrojů bylo možné zobrazit stav a hodnoty jednotlivých proměnných vytvořených v programu.

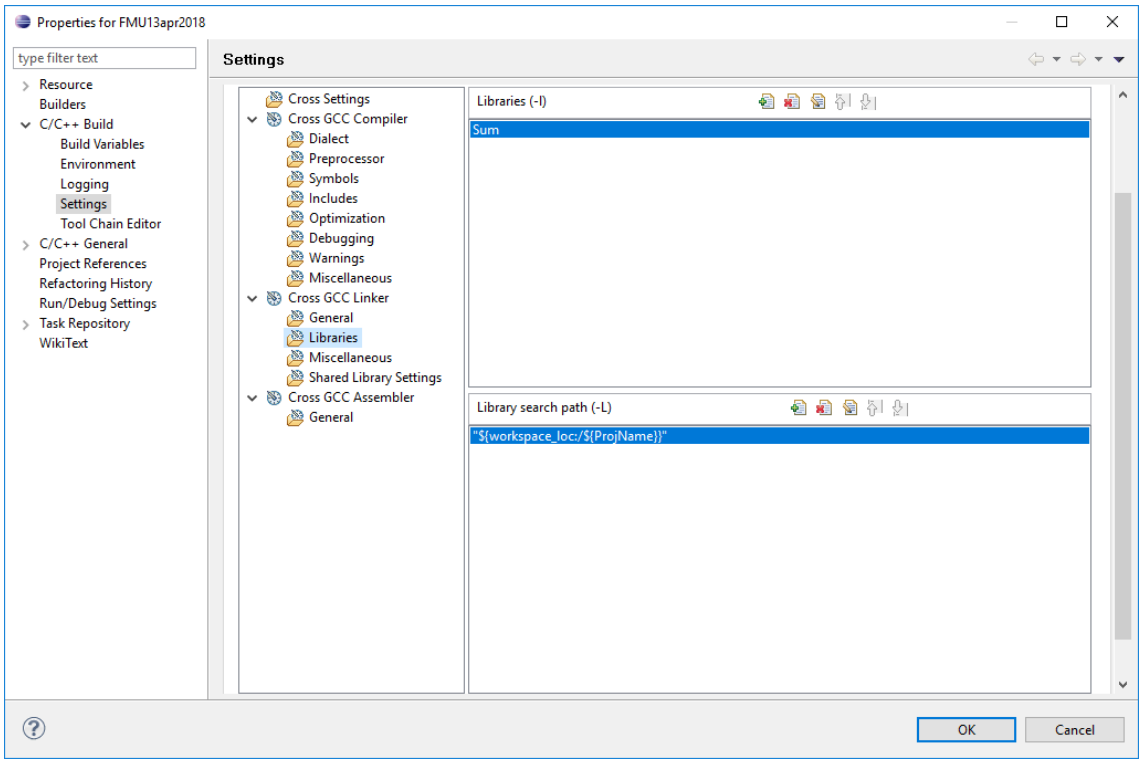
Testovací program byl tímto krokem úspěšně otestován. Další část této práce byla věnována vytvoření instance FMU a její simulaci.

5.5.5 Vytvoření instance FMU a její simulace

Vytváření instancí FMU je obecně svázáno s binárními soubory, které jsou obsaženy v každém FMU. Tyto binární soubory musí být zkompilovány pro konkrétní cílovou platformu, na které bude vytvořená aplikace provozována. Vytvořená aplikace pak bude využívat zkompilované binární soubory k účelům tvorby a simulace jednotlivých instancí vlastního modelu.

Vytvoření instance FMU bylo provedeno úpravou souboru *main.c*, který byl původně použit pro otestování funkčnosti SSH komunikace mezi PC a cRIO. V kapitole 5.4 byl popsán postup kompilace FMU pro cílovou platformu. Jejím výsledkem byl právě binární soubor *Sum.so*, který byl zkopírován do složky projektu. V projektu byl tento soubor pro řádné označení, že se jedná o sdílený objekt OS Linux, přejmenován na *libSum.so*.

Přejmenovaná shared object knihovna musela být překopírována také do cílové platformy. Aniž by byla překompilovaná sdílená knihovna fyzicky přítomna v adresáři vzdáleného systému (cRIO), do kterého je vlastní spustitelný program při debugu nebo jeho spuštění zkopírován, nebylo možné program na cílové platformě spustit a vzdálený systém zahlásil *segmentation fault*.



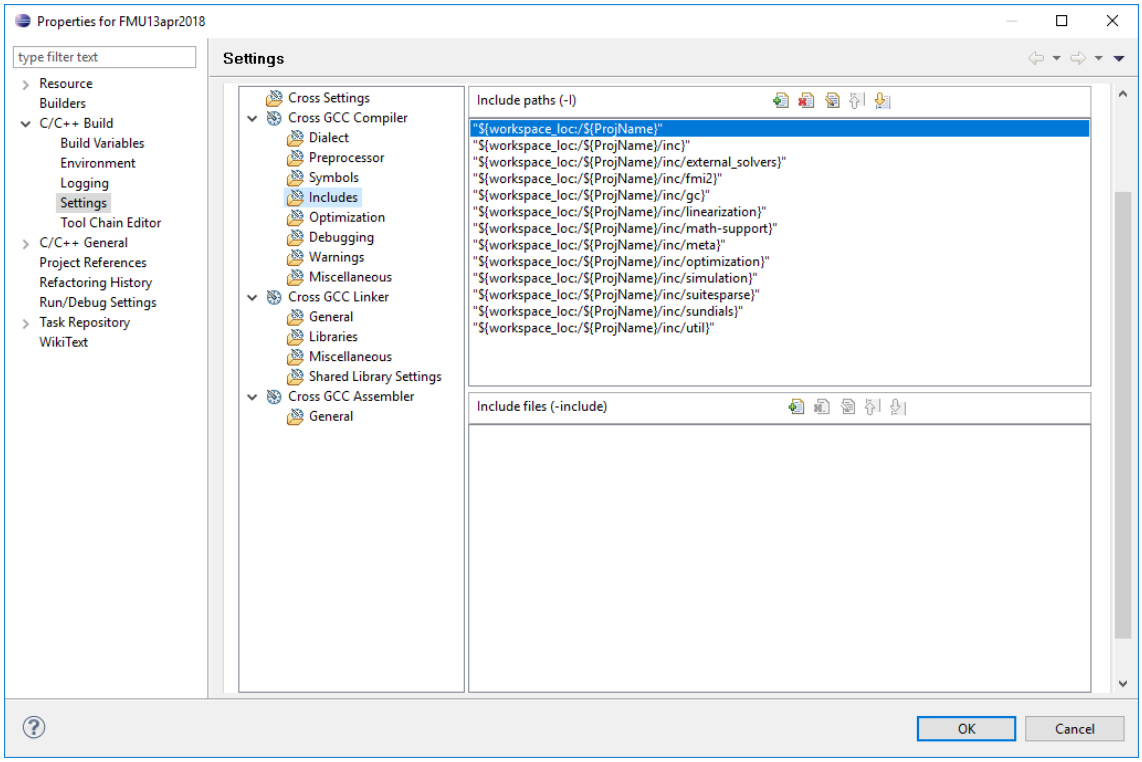
Obr. 5.15: Nastavení linkeru projektu

Mimo to, že byla sdílená knihovna vložena do projektu a následně překopírována do cílové platformy, bylo nutné linkeru oznámit, že má sdílenou knihovnu přilinkovat k ostatním výstupním souborům. Nastavení linkeru je ukázáno na obrázku 5.15.

Přestože sdílená knihovna *libSum.so* obsahovala všechny funkce definované standardem FMI, nebylo možné použít našeptávač (Ctrl + mezerník) vývojového prostředí pro zefektivnění práce při psaní programu. Mimo to nebyly v projektu přístupné ani definice, popřípadě deklarace funkcí, které popisuje samotný FMI standard. Bez toho, aby byla známa alespoň deklarace jednotlivých funkcí, nebylo možné vytvářet program jinak, než na základě dokumentace standardu. Často se však stává, že dokumentace obsahuje chyby, které by mohly následně vést k chybám vlastního programu.

Při tvorbě vlastního programu nebylo možné tento nedostatek pro komplikovanost řešení ignorovat, a proto musel být odstraněn. Následně bylo zjištěno, že všechny

funkce FMI standardu jsou zakomponovány v FMU vygenerovaném z nástroje OpenModelica. Funkce standardu FMI byly umístěny v FMU v adresáři *Jmeno_modelu/sources/include*. Obsah složky byl překopírován do dočasného adresáře. V dočasném adresáři byly vyfiltrovány všechny zdrojové soubory s příponou **.c* a **.cpp* tak, aby zbyly pouze hlavičkové soubory. Tyto hlavičkové soubory byly vloženy do nového adresáře *inc* vytvořeného v projektu Eclipse studia.



Obr. 5.16: Vložení hlavičkových souborů FMI standardu do projektu

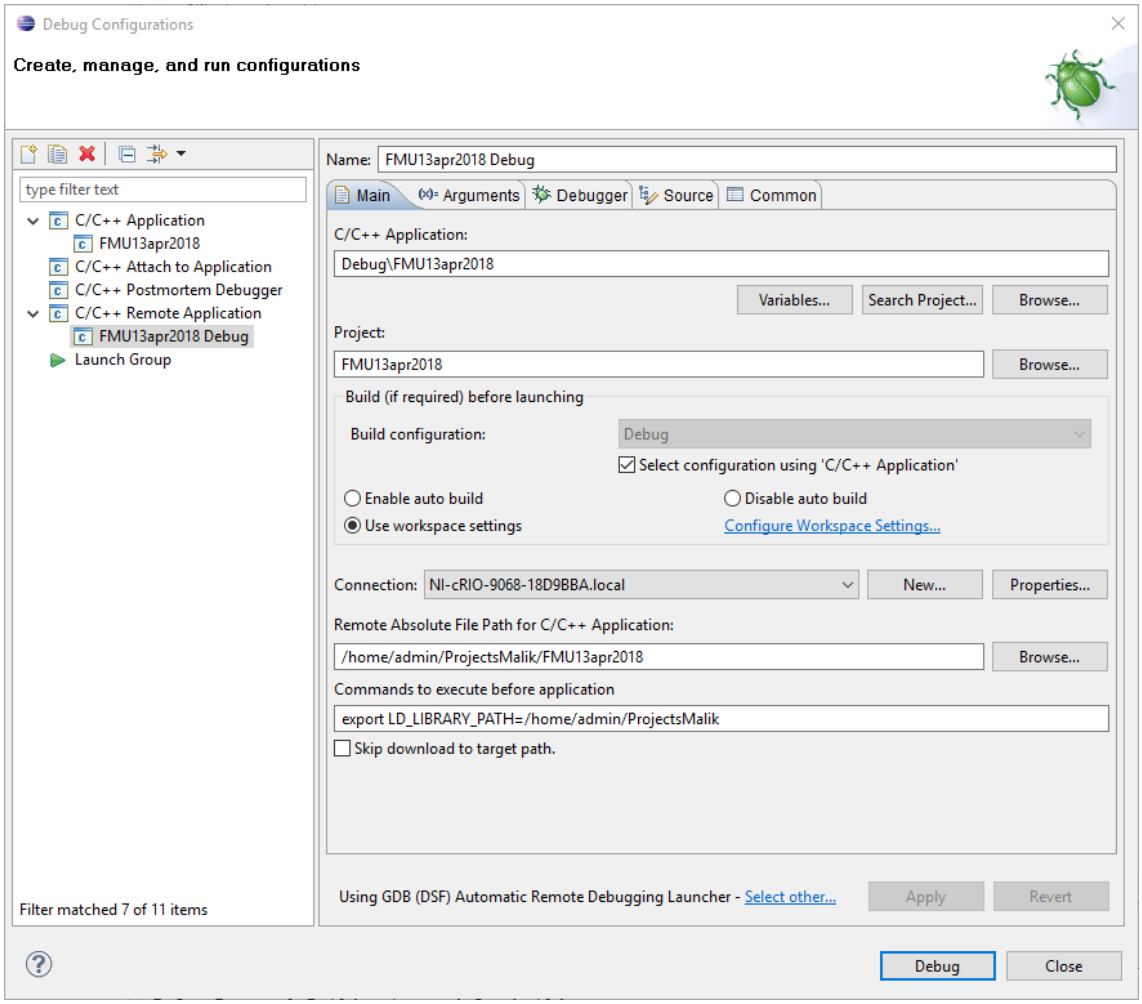
Po vložení hlavičkových funkcí FMI standardu do vlastního projektu bylo provedeno další nastavení v *Project settings* (klávesová zkratka Alt+Enter). Na obrázku 5.16 je možné vidět, jak byly přidány vložené hlavičkové soubory do zdrojových cest cross compileru.

Tímto byla funkce našeptávače zprovozněna a nebylo nutné dále používat dokumentaci FMI standardu, ve které se mohly vyskytovat chyby. Navíc byl umožněn přístup k deklaracím jednotlivých funkcí ve vývojovém prostředí Eclipse.

K tomu, aby bylo vůbec možné jakoukoliv funkci z FMI standardu použít ve vlastním programu, musela být příslušná funkce před jejím použitím prvně deklarována. Tím, že byly deklarace funkcí FMI standardu vloženy do projektu, bylo možné deklarace použitých funkcí ve vlastním programu do vlastního programu zkopírovat.

Mimo doposud provedených nastavení byla nastavena ještě proměnná prostředí

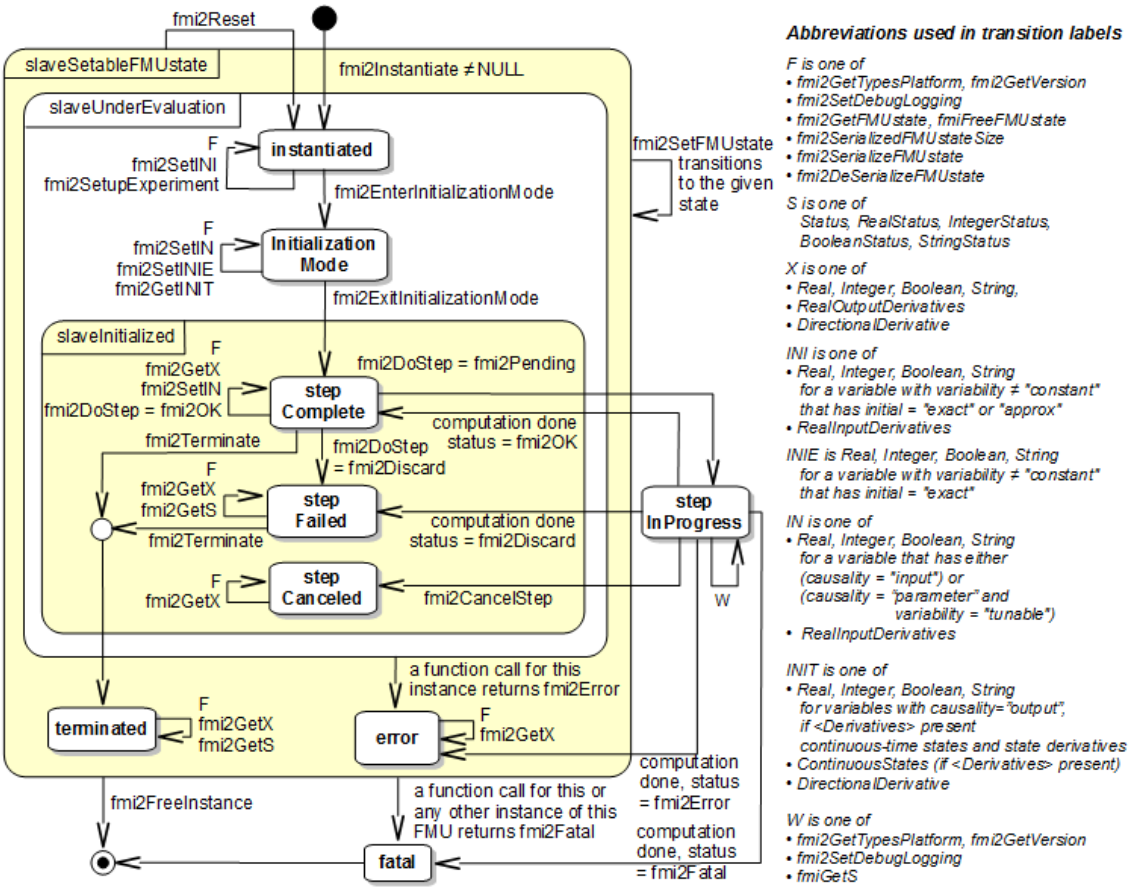
na cílové platformě, a to z důvodu chyby *segmentation fault*, kterou hlásilo cRIO po spuštění vlastního programu, který používal funkce FMI standardu. Hlášení chybového stavu bylo logické, neboť RealTime OS Linux, který běžel na cílové platformě, hledal binární soubor *libSum.so* na jiném (defaultním) místě ve své adresářové struktuře.



Obr. 5.17: Nastavení proměnné prostředí na OS Linux

Po nastavení všech doposud zmíněných parametrů projektu, vložení knihovních souborů FMI standardu, vložení překompilovaného shared object souboru, jak do projektu, tak i do cílového zařízení, a po nastavení hodnoty proměnné cesty prostředí na cílové platformě byl vytvořen jednoduchý stavový automat pro simulaci modelu sumátoru (viz obrázek 5.18).

Vytvořený stavový automat vycházel z obrázku stavového automatu uvedeného ve standardu FMI [13]. Samozřejmě nebylo použito všech dostupných funkcí, protože model sumátoru obsahoval dvě vstupní proměnné a jednu výstupní proměnnou.



Obr. 5.18: Obecný stavový automat FMU pro CoSimulaci [13]

Všechny proměnné přitom byly celočíselného datového typu integer. Vytvořený stavový automat je možné rozdělit do třech částí, tak jako je uvedeno na obrázku 5.18.

V první části vlastního stavového automatu vytvořeného v souboru *main.c*, která je ve stavovém automatu označena jako *slaveSetableFMUstate*, byly inicializovány proměnné, které byly použity pro řízení stavového automatu. Dále byla vytvořena jedna instance modelu sumátoru pomocí funkce *fmi2Instantiate*. Návratovou hodnotou funkce *fmi2Instantiate* byla nově vytvořená proměnná *s1* datového typu *fmi2Component*. Pro kontrolu korektnosti vytvoření instance FMU slouží podmínka testující návratovou hodnotu z funkce *fmi2Instantiate*. Pokud je návratová hodnota funkce *fmi2Instantiate* rovna *NULL*, potom nebyla instance vytvořena a program musí být ukončen.

Ve stejné části programu byla nastavena hodnota proměnných *startTime*, *stopTime* a *h*. První dvě proměnné slouží pro nastavení počátečního a koncového času simulace. Proměnná *h* slouží pro nastavení kroku simulace. K nastavení počátečního a koncového času simulace vytvořené instance byla použita funkce *fmi2 Setup*

Experiment z FMI standardu.

Ke vstupu do inicializační části instance modelu byla použita funkce *fmi2 Enter Initialization Mode (s1)*. V tomto stavu byly nastaveny hodnoty proměnných instance modelu sumátoru v čase nastaveném proměnnou *startTime*. Po nastavení hodnot vstupních proměnných v čase rovném hodnotě proměnné *startTime* byl funkcí *fmi2 Exit Initialization Mode (s1)* opuštěn stav inicializace instance.

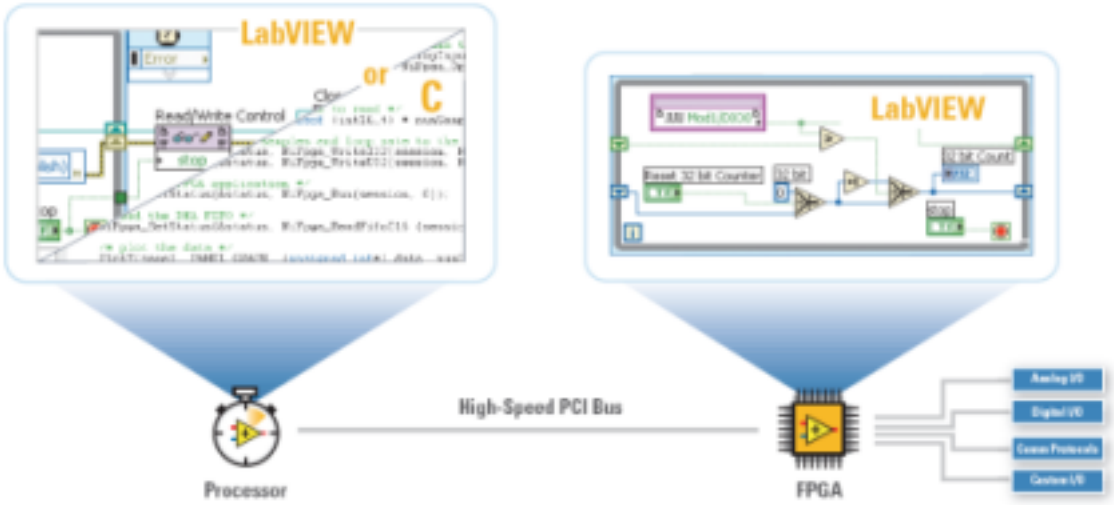
Ve třetí části stavového automatu, v obrázku 5.18 označené *slaveInitialized*, byla s použitím case struktury a smyčky while vytvořena část programu, která zajišťuje simulaci instance FMU. Ve smyčce while jsou cyklicky volány funkce pro vyčítání výstupů, nastavení vstupů a provedení jednoho kroku simulace. K získání hodnot výstupu modelu sumátoru byla použita funkce *fmi2GetInteger*. Pro nastavení nových vstupních hodnot instance modelu byla použita funkce *fmi2SetInteger*, a pro vykonání jednoho kroku simulace s velikostí kroku definovaného proměnnou *h* byla použita funkce *fmi2DoStep*. Návratová hodnota funkce *fmi2DoStep* byla dále pomocí case struktury testována na všechny možné návratové hodnoty, které mohla tato funkce po jejím zavolání vrátit. Pokud hodnota návratové funkce *fmi2DoStep* byla rovna hodnotě *fmi2Error*, *fmi2Discard* nebo *fmi2Fatal*, bylo nutné okamžitě řádně ukončil simulační část a ukončit, popřípadě i odalokovat paměť vytvořené instance.

Vytvořený stavový automat modelu sumátoru byl uložen společně s celou složkou projektu studia Eclipse na CD nosič, který byl přiložen k této práci. Ve standardu FMI byl rovněž vysvětlen způsob použití všech ostatních funkcí, které nebyly ve vytvořeném stavovém automatu použity.

Ke splnění posledního bodu zadání, který měl realizovat propojení výše vytvořené instance modelu, která běžela v RealTimové části platformy cRIO s její FPGA částí, byl vytvořen komunikační kanál mezi oběma částmi platformy cRIO, jemuž se věnuje kapitola 6.

6 C API komunikační rozhraní

Jako C API (Application Programming Interface) je označováno rozhraní, které je využíváno pro programování heterogenních aplikací. API obsahuje funkce, třídy anebo procedury nějaké knihovny, které mohou být programátorem použity. Application Programming Interface popisuje, jak se daná funkce volá a jaký parametr po jejím zavolání funkce vrátí v návratové hodnotě. Rozhraní založené na stejném principu bylo implementováno pro vytvoření komunikace mezi RealTime procesorem a FPGA na platformě cRIO.



Obr. 6.1: C interface pro FPGA [19]

Na obrázku 6.1 je znázorněno propojení RealTime procesoru a FPGA, které jsou osazeny uvnitř platformy cRIO. Programová výbava musí být vytvořena individuálně pro každý procesor. Tvorba programu pro RealTime procesor může být realizována buď pomocí grafického programování nebo programováním v jazyce C. Programování FPGA je možné realizovat pouze pomocí grafického uživatelského prostředí LabVIEW, které musí být následně přeloženo do jazyka VHDL. Po překladu vytvořeného programu pro FPGA musí být přeložený program do FPGA nahrán a následně spuštěn.

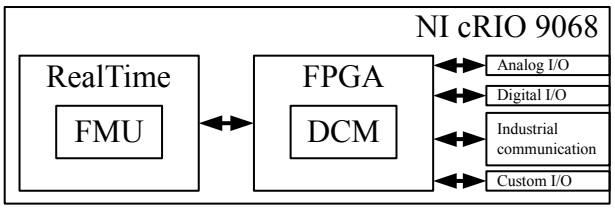
6.1 Popis rozhraní C API

Fyzickou vrstvou komunikace mezi RealTime procesorem a FPGA tvoří systémová sběrnice AXI (Advanced Extensible Interface). Aby bylo možné po této sběrnici posílat data mezi procesorem a FPGA, musí být vytvořen komunikační kanál, kterým

budou posílány data v podobě zpráv mezi procesorem a hradlovým polem.

Při řešení této práce musela být vyřešena situace, kdy byl vytvořen řídicí algoritmus pro práci s instancí FMU, který měl být schopen zpracovávat naměřená data z reálného prostředí. V systémech cRIO je možné pro zvýšení spolehlivosti měření fyzikálních veličin použít FPGA, ke kterému jsou připojeny vstupně výstupní karty. Vstupně výstupní karty slouží pro interakci mezi okolním prostředím a FPGA a také pro převod fyzikálních veličin do digitální podoby. Vstupně výstupní karty mohou být v libovolné konfiguraci a pořadí připojeny do šasi platformy cRIO. K těmto kartám jsou připojeny jednotlivé senzory, které snímají sledované fyzikální veličiny. Tímto je zajištěno měření fyzikálních veličin na reálném systému.

Naměřené hodnoty, jejichž spolehlivé snímání zajišťuje FPGA, však bylo nutné zpracovat v RealTime procesoru, ve kterém se nacházel vlastní algoritmus instance FMU. K předání naměřených hodnot z FPGA do RealTime procesoru (nebo naopak) slouží spojení pomocí DMA rozhraní nebo Peer-to-Peer spojení. V případě použití DMA rozhraní musí být v projektové složce LabVIEW vytvořena fronta typu FIFO, do které je následně možné například z VI pro FPGA zapisovat naměřené hodnoty. Naměřené hodnoty je pak možné v nadřazené RealTime aplikaci dále podle potřeby zpracovávat a vyhodnocovat.

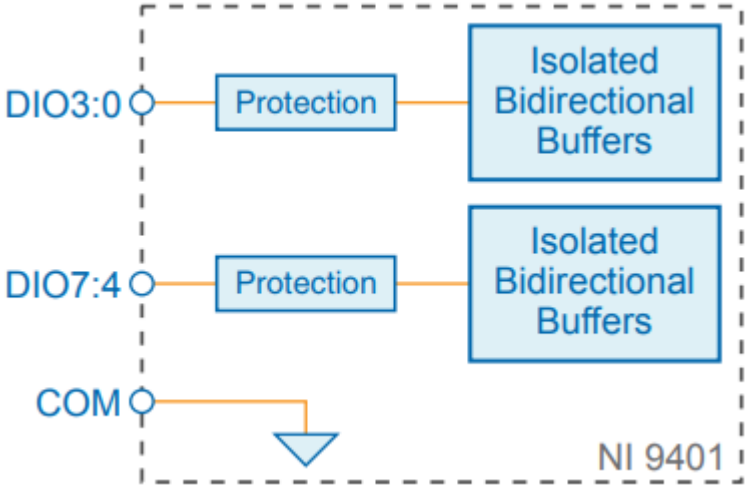


Obr. 6.2: Propojení FMU s LabVIEW FPGA

Pro splnění posledního bodu zadání této práce bylo nutné realizovat propojení vytvořené instance FMU (umístěna v RealTime části cRIO) s LabVIEW FPGA. Za tímto účelem bylo použito FPGA C API rozhraní, které podle [20] poskytuje téměř totožnou funkcionalitu, jako grafický nástroj LabVIEW. Výhodou FPGA C API rozhraní je, že přináší možnost automatického generování graficky vytvořeného programu v prostředí LabVIEW přímo do programu jazyka C. Vygenerovaný program jazyka C je možné následně vložit do programu pro RealTime procesor a zpracovávat v něm naměřená data z FPGA.

6.2 Propojení FMU s LabVIEW FPGA

Pro realizaci propojení FMU s LabVIEW FPGA byl nejprve vytvořen nový projekt v prostředí LabVIEW. V projektu bylo vytvořeno nové VI s názvem *FPGA.vi*, které bylo ve stromu projektu umístěno pod zařízením FPGA. Před samotným vytvořením programu pro FPGA v grafickém prostředí byla do šasi stanice cRIO vložena vstupně výstupní karta s označením *NI-9401*.

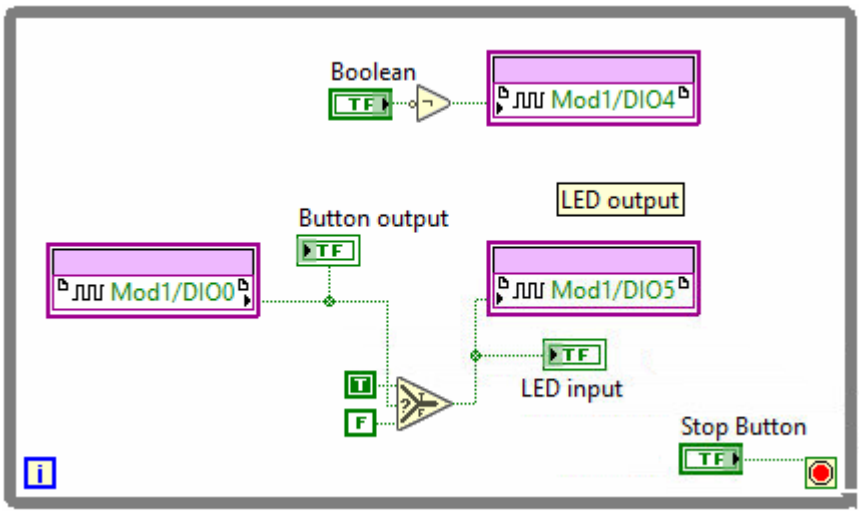


Obr. 6.3: Vnitřní struktura digitální vstupně výstupní karty NI 9401 [21]

Karta *NI-9401* disponovala 8 DIO s TTL logikou. Uvnitř karty je 8 DIO seskupeno do dvou portů, které mohou být nakonfigurovány buď jako vstupy nebo jako výstupy. Aby mohla být jednoduše otestována funkčnost propojení FMU s LabVIEW FPGA, byl první port nakonfigurován jako vstupní a druhý port jako výstupní. Na jeden vstupní pin prvního portu bylo připojeno tlačítko a ke dvěma výstupním pinům druhého portu byly připojeny LED diody s předřadným rezistorem. Hodnota předřadného rezistoru byla stanovena pomocí Ohmova zákona na základě maximálního dovoleného proudu, který měl diodou téct v sepnutém stavu.

Poté byl v prostředí LabVIEW pomocí knihovních bloků vytvořen program (viz obrázek 6.4), který realizuje rozsvícení jedné LED diody po stisknutí tlačítka. Tento program byl přeložen do jazyka VHDL. Při spuštění překladu programu pomocí tlačítka *Run*, došlo k chybě, která vznikla kvůli nenainstalovanému překladač VHDL jazyka. Na stránkách National Instruments byl jako kompatibilní překladač pro platformu NI cRIO 9068 nalezen *LabVIEW 2017 FPGA Module Xilinx Compilation Tool for Vivado 2015.4*. Získaný překladač byl po zavření všech otevřených projektů nainstalován. Po opětovném spuštění vytvořeného projektu a spuštěním kompilace vytvořeného programu proběhl překlad do jazyka VHDL v pořádku a vzniklý binární

soubor byl nahrán do FPGA. Po úspěšném otestování funkčnosti rozsvícení jedné LED diody pomocí tlačítka byl program modifikován tak, aby druhá LED dioda byla ovládaná pomocí binární proměnné na front panelu VI pro FPGA. Program byl opět přeložen do jazyka VHDL a nahrán do FPGA, jako v předchozím případě.



Obr. 6.4: Vytvořený program v FPGA.vi

Z informací uvedených v [20] vyplynulo, že C API rozhraní používá k vytvoření zdrojových souborů jazyka C generátor, který tvoří zdrojové soubory na základě binárního souboru vytvořeného při překladu grafického programu do jazyka VHDL. Na základě informací uvedených v [20] byla spuštěna generace zdrojových souborů jazyka C. Po jejím spuštění nastala chyba, která byla způsobena absencí SW *FPGA Interface C API 2.0* dostupného z [31].

Po doinstalování *FPGA Interface C API 2.0* a znovu spuštění procesu generování zdrojových souborů jazyka C, byl v projektové složce vytvořen nový adresář s názvem *FPGA Bitfiles*. Tento adresář obsahoval soubory *NiFpga.c*, *NiFpga.h*, *NiFpga_FPGA.h* a *NiFpga_FPGA.lvbitx*.

Je nutné zdůraznit, že C API generátor nepodporuje všechny datové typy, které grafické programovací prostředí LabVIEW nabízí. Mezi C API generátorem podporované datové typy patří:

- Boolean
- Integer
- Floating Point
- LabVIEW FPGA controls/indicators
- DMA FIFOs
- a přerušení

Ostatní datové typy jako například Fixed point, což je nativní datový typ pro FPGA, a clustery, které používá prostředí LabVIEW, nejsou podle [22] podporovány.

6.3 Práce s rozhraním C API

C API generátorem vytvořené soubory byly vloženy do Eclipse projektu, ve kterém byla vytvořena instance FMU. Bylo nutné přidat složku s umístěním těchto souborů do cesty linkeru v *Project properties*. K tomu, aby mohly být využity funkce C API rozhraní, musel být vložen soubor *NiFpga_FPGA.h* do programu *main.c* pomocí klíčového slova *#include "NiFpga_FPGA.h"*. Kromě toho musela být do Eclipse projektu přidána také knihovna standardních symbolických konstant a typů pomocí příkazu *#include <unistd.h>*.

V souboru *NiFpga_FPGA.h* se nachází funkce rozhraní C API, které je možné použít při vytváření vlastní aplikace. Kromě samotných funkcí je v tomto souboru definována proměnná *NiFpga_FPGA_Bitfile*, která říká jak je pojmenován bitový soubor **.lvbitx*. Je zde také uvedena proměnná *NiFpga_FPGA_Signature* jejíž hodnota definuje digitální podpis binárního souboru.

Na základe tutoriálu nalezeného v [22] byl do programu *main.c*, který řešil simulaci instance FMU, vytvořen program, který zprovoznil komunikaci mezi RealTime procesorem a FPGA.

```
1  NiFpga_Session session;
2
3  /* Load FPGA interface library */
4  NiFpga_Status status = NiFpga_Initialize();
5
6  /* Open a~session with FPGA, download and run the
      bitstream on the FPGA and store any error info in "
      status" */
7  NiFpga_MergeStatus(&status, NiFpga_Open("/home/admin/
      ProjectsMalik/NiFpga_FPGA.lvbitx",
8          NiFpga_FPGA_Signature,
9          "RIO0",
10         0,
11         &session));
12 }
```

Výpis 6.1: Vytvoření relace spojení RealTime procesoru a FPGA

Po vložení vygenerovaných souborů do Eclipse projektu byl do hlavního programu vložen hlavičkový soubor *NiFpga_NazevVlastnihoFPGAprojektu*. V tomto

souboru byly definovány všechny C API rozhraním podporované proměnné, které byly vytvořeny v rámci VI umístěného v FPGA.

Před tím, než mohla být zahájena výměna dat mezi RealTime a FPGA uvnitř cRIO, musela být nejprve vytvořena relace spojení odkazující na příslušný bitový soubor, viz výpis 6.1.

K vytvoření relace bylo nutné zadat unikátní podpis bitového souboru a také název zdroje. Dalším krokem před zahájením komunikace byla inicializace proměnných, které ovládaly chování VI aplikace v FPGA, viz výpis 6.2.

```
1 /* Initialize variables that control the application
     behavior */
2   NiFpga_Bool button_pressed = 0;   /* button is not
       pushed */
3   NiFpga_Bool LED = 0;         /* LED is OFF */
4 }
```

Výpis 6.2: Inicializace řídicích proměnných C API

Po periodicky se opakující komunikaci mezi RealTime procesorem a FPGA byla vytvořena *while* smyčka s časováním 100 ms. Uvnitř této smyčky byl vytvořen program, který zpracoval událost stisknutí tlačítka z reálného prostředí a na základě této události reagoval pomocí připojené LED diody, která signalizovala stisknutí tlačítka.

```
1  while(NiFpga_IsNotError(status)){
2      NiFpga_MergeStatus(&status, NiFpga_ReadBool(session,
3          NiFpga_FPGA_IndicatorBool_Buttonoutput,
4          &button_pressed));
5      if (button_pressed){
6        LED = 1;
7      }
8      else {
9        LED = 0;
10     }
11     NiFpga_MergeStatus(&status, NiFpga_WriteBool( session,
12           NiFpga_FPGA_ControlBool_Boolean,
13           LED));
14     usleep(100000); /* 100ms delay */
15   }
16 }
```

Výpis 6.3: Komunikace mezi RealTime procesorem a FPGA v platformě cRIO

Vytvořený program, který je uveden ve výpisu 6.3, běží tak dlouho, dokud není některou z C API funkcí vrácen status *Error*. Hodnota tohoto statusu je kontrolována vždy na začátku iterace while smyčky. Další částí vytvořeného programu bylo zjištění, zda reálné tlačítko, které bylo k platformě cRIO připojeno pomocí digitální karty NI 9401, není stisknuté. Kontrola stisknutí tlačítka byla provedena pomocí C API funkce *NiFpga_MergeStatus*, jejíž druhý parametr nařizoval použít inline funkci čtení stavu příslušného tlačítka.

Na základě výsledku inline funkce *NiFpga_ReadBool* byla změněna hodnota bitové proměnné *LED*, která byla vzápětí použita pro zápis hodnoty do proměnné, která v FPGA ovládala LED diodu, připojenou k výstupnímu portu digitální karty. Aby mohla být definována proměnná typu bool v jazyce ANSI C byla ve vlastním programu použita knihovna *stdbool.h*, který tento datový typ definuje.

Vytvořený program ovládání LED diody pomocí tlačítka byl zkompilován a nahrán pomocí vzdálené správy zařízení do platformy cRIO. Ještě před spuštěním programu musel být umístěn bitový soubor vygenerovaný rozhraním C API na cílovou platformu.

Po prvním spuštění vytvořeného programu v režimu ladění byla odhalena chyba. Při volání funkce *NiFpga_Initialize()*, která je uvedena na 4. řádku výpisu 6.1, byla inicializační funkcí vrácena hodnota – 63194. Tato hodnota podle [32] říká, že verze SW na platformě NI cRIO 9068 není kompatibilní s použitou verzí rozhraní C API 17. K odstranění této chyby musí být použity stejné verze SW pro platformu cRIO, LabVIEW a C API rozhraní (generátor).

Odstraněním chyby *NiFpga_Status_VersionMismatch* byl úkol propojení RealTime části platformy cRIO s FPGA splněn, nicméně by bylo vhodné pro velké objemy přenášených dat použít DMA FIFO než jednotlivé proměnné. Před zahájením komunikace pomocí DMA kanálu je zapotřebí provést jeho inicializaci, která trvá déle, než v případě použití jednotlivých proměnných. Výhodou DMA kanálu je však rychlejší předávání dat mezi instancí FMU a FPGA.

7 Závěr

Tato diplomová práce se zabývala tvorbou fyzikálních (akauzálních) modelů v jazyce Modelica a způsoby, které lze použít k jejich implementaci do prostředí LabVIEW za účelem Hardware-In-the-Loop simulace.

Práce je rozdělena do několika částí. První část se zaměřovala na jazyk Modelica a grafický programovací nástroj OpenModelica. Byly zde popsány rozdíly mezi klasickým (kauzálním) a fyzikálním (akauzálním) modelováním. V další části se práce věnovala standardu Functional Mock-up Interface 2.0, který slouží k výměně simulačních modelů mezi různými vývojovými nástroji ve standardizovaném formátu. Byly zde rozebrány základní parametry a vlastnosti standardu. Následující kapitola se věnovala popisu vývojového prostředí LabVIEW a možnostmi, kterými disponuje platforma CompactRIO.

Praktická část práce se zabývala vytvořením modelu elektromechanického systému v prostředí OpenModelica Connection Editoru, který byl následně použit jako jeden z nástrojů pro implementaci modelu do prostředí LabVIEW RealTime. Tento úsek se věnoval také popisu způsobu, kterým byl model z OMEditoru vyexportován do formátu Functional Mock-up Unit. Byla rozebrána adresářová struktura modelu včetně popisu vygenerovaného XML souboru. Dále byly ukázány způsoby, kterými lze vygenerovaný model ve formátu FMU naimportovat do prostředí LabVIEW RealTime.

V návaznosti na původně zamýšlený způsob implementace modelu elektromechanického systému do prostředí LabVIEW RealTime pomocí add-onu *LabVIEW support for FMI for Model Exchange* byly částečně vyřešeny problémy spojené s importem vlastního modelu do zmíněného prostředí pomocí *FMU Compliancy Checkeru*. Hlavním problémem spojeným s použitím add-onu byly problémy se zobrazením terminálových svorek na knihovním bloku naimportovaného modelu v prostředí LabVIEW.

Od původně zamýšleného způsobu implementace modelu elektromechanického systému do prostředí LabVIEW RealTime pomocí add-onu bylo nakonec upuštěno. Otázka importu modelu elektromechanického systému do prostředí LabVIEW RealTime byla úspěšně vyřešena vývojem vlastního programu jazyka C pro platformu cRIO s operačním systémem NI Linux RealTime. Za tímto účelem byly vygenerované soubory z nástroje OpenModelica pomocí standardu FMI 2.0 zkompilovány pro vlastní cílovou platformu NI cRIO 9068. Ve vývojovém prostředí Eclipse studio byl vytvořen algoritmus vytvoření instance FMU, který byl založen na základě stavového automatu vycházejícího ze standardu FMI 2.0.

Po úspěšném vytvoření a odsimulování instance FMU v RealTime procesoru platformy cRIO byl ukázán způsob vytvoření komunikačního kanálu mezi RealTime

procesorem a FPGA čipem, umístěným v platformě cRIO. Komunikace mezi Real-Time procesorem a FPGA byla vytvořena na základě C API rozhraní, které pomocí grafického programování v nástroji LabVIEW vygenerovalo soubory a funkce jazyka C, které byly následně použity v programu pro simulaci instance FMU.

Při realizaci praktické části práce nastalo několik problémů, které bylo nutné překonat. Je nutné podotknout, že muselo být od původně zamýšleného řešení importu vlastního modelu FMU do prostředí LabVIEW RealTime upuštěno, ale i přes to se povedlo splnit zadání diplomové práce v plném rozsahu. Pro komplexnost vytvořeného řešení by bylo vhodné do budoucna postup generování FMU, jeho kompilaci pro cílovou platformu cRIO a také simulaci instance FMU automatizovat tak, aby byla zachována uniplatformita operačního systému.

Nad rámec zadání byl vyhotoven návod vytvoření projektu ve vývojovém studiu Eclipse, který zahrnuje postup vytvoření nového projektu, nastavení projektu, nastavení režimu ladění a vytvoření SSH komunikace mezi projektem a cílovou platformou.

Literatura

[1] FRITZSON, Peter A. *Introduction to modeling and simulation of technical and physical systems with Modelica*. 2001. Hoboken, N.J.: IEEE Press, c2011. ISBN 111801068X.

[2] TILLER, Michael. *Introduction to physical modeling with Modelica*. 2001. Boston: Kluwer Academic Publishers, c2001. ISBN 0792373677.

[3] *Modelica® - A Unified Object-Oriented Language for Systems Modeling: Language Specification* [online]. Version 3.4. The Modelica Association, 2017 [cit. 2017-12-15]. Dostupné z: <https://www.modelica.org/documents/ModelicaSpec34.pdf>.

[4] OMPython. *Openmodelica.org* [online]. [cit. 2017-12-29]. Dostupné z: <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/ompython.html>.

[5] OpenModelica MDT user guide. *Openmodelica.org* [online]. [cit. 2017-12-29]. Dostupné z: <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/mdt.html>.

[6] *OpenModelicaCompiler and Simulation settup* [online]. [cit. 2017-12-29]. Dostupné z: <https://github.com/OpenModelica/OMCompiler>.

[7] LAMBERG, Klaus a Peter WÄLTERMANN. Using HIL Simulation to Test Mechatronic Components in Automotive Engineering. *DSPACE GmbH* [online]. Paderborn, 2000 [cit. 2017-12-19]. Dostupné z: <http://www.dspaceinc.com/ftp/papers/hdt00_e.pdf>.

[8] KOFRÁNEK, Jiří, Marek MATEJÁK, Pavol PRIVITZER a Martin TRIBULA. *KAUZÁLNÍ NEBO AKAUZÁLNÍ MODELOVÁNÍ: DŘINU LIDEM NEBO DŘINU STROJŮM* [online]. Laboratoř biokybernetiky, Ústav patologické fyziologie 1. LF UK,Praha [cit. 2017-12-29]. Dostupné z: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=134E99981B37D66365A0EA09D3D8AD27?doi=10.1.1.521.9531&rep=rep1&type=pdf>.

[9] Rexcontrols: Rapid development, RT simulace, modelování. *Rexcontrols* [online]. 2017 [cit. 2017-12-29]. Dostupné z: <https://www.rexcontrols.cz/rapid-development-rt-simulace-modelovani>.

[10] DC motor scheme, In: *Control Tutorials for MATLAB and Simulink (CTMS)* [online]. MATLAB® 9.2 [cit. 2017-12-29]. Dostupné z: <http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SimulinkModeling>.

[11] *Modelon: What is FMI* [online]. 2017 [cit. 2017-12-30]. Dostupné z: <http://www.modelon.com/blog/articles/what-is-fmi/>.

[12] FUNCTIONAL MOCK-UP INTERFACE. *Dassault Systèmes* [online]. [cit. 2017-12-17]. Dostupné z: <https://www.3ds.com/products-services/catia/products/dymola/fmi/>.

[13] MODELICA ASSOCIATION, *Functional Mock-up Interface 2.0: Project Functional Mock-up Interface*, 2014. Edition 2.0. Dostupné také z: <https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf>.

[14] Getting Started with CompactRIO and LabVIEW. *National Instruments Corporation* [online]. 2009 [cit. 2018-01-02]. Dostupné z: <http://www.ni.com/pdf/manuals/372596b.pdf>.

[15] NI CompactRIO Performance Controller: Performance and Throughput Benchmarks. *National Instruments Corporation* [online]. 2015 [cit. 2018-01-03]. Dostupné z: <http://www.ni.com/white-paper/52250/en/>.

[16] BARTÍK, Ondřej. *MIL simulace elektrických motorů v reálném čase.* [online]. Brno, 2017, 70 s Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce Doc. Ing. Petr Blaha, Ph.D. [cit. 2018-01-04]. Dostupné z: <https://www.vutbr.cz/studium/zaverecne-prace?zp_id=102853>.

[17] LabVIEW support for FMI for Model Exchange. *National Instruments Corporation: Forum* [online]. 2015 [cit. 2018-01-04]. Dostupné z: <https://forums.ni.com/t5/NI-Labs-Toolkits/LabVIEW-support-for-FMI-for-Model-Exchange/ta-p/3506018>.

[18] FMU Complianci Checker. *GitHub* [online]. 2017 [cit. 2018-01-04]. Dostupné z: <https://github.com/modelica-tools/FMUComplianceChecker/blob/master/README.md>.

[19] Building a R Series FPGA Interface Host Application in C. *National Instruments* [online]. 2016, 11.07.2016 [cit. 2018-05-09]. Dostupné z: <http://www.ni.com/tutorial/8638/en/>.

[20] Introduction to the FPGA Interface C API. *National Instruments* [online]. 2017, 27.09.2017 [cit. 2018-05-09]. Dostupné z: <http://www.ni.com/product-documentation/9036/en/>.

[21] NI 9401 Datasheet. *National Instruments* [online]. 2015 [cit. 2018-05-09]. Dostupné z: <http://www.ni.com/pdf/manuals/374068a_02.pdf>.

[22] Programming with the FPGA Interface C API. *National Instruments* [online]. 2017, 15.05.2017 [cit. 2018-05-10]. Dostupné z: <http://www.ni.com/white-paper/53283/en/>.

[23] *OpenModelica: Online repozitář společnosti OpenModelica* [online]. 2018 [cit. 2018-05-12]. Dostupné z: <https://github.com/OpenModelica>.

[24] LabVIEW support for FMI for Model Exchange. *National Instruments* [online]. 2018, 04.06.2015 [cit. 2018-05-12]. Dostupné z: <https://forums.ni.com/t5/NI-Labs-Toolkits/LabVIEW-support-for-FMI-for-Model-Exchange/ta-p/3506018>.

[25] *Download Ubuntu Desktop* [online]. 2018 [cit. 2018-05-12]. Dostupné z: <https://www.ubuntu.com/download/desktop>.

[26] *OpenModelica Linux released software* [online]. 2018 [cit. 2018-05-12]. Dostupné z: <https://openmodelica.org/download/download-linux>.

[27] GNU C & C++ Compilers for ARMv7 Linux (Linux host) 2014-2016: oecore-x86_64-armv7a-vfp-neon-toolchain-2.0.sh. *National Instruments* [online]. 2018, 20.04.2014 [cit. 2018-05-12]. Dostupné z: <http://www.ni.com/download/labview-real-time-module-2014/4957/en/>.

[28] Getting Started with C/C++ Development Tools for NI Linux Real-Time, Eclipse Edition. *National Instruments* [online]. 06.02.2018 [cit. 2018-05-12]. Dostupné z: <http://www.ni.com/tutorial/14625/en/>.

[29] *C/C++ Development Tools for NI Linux Real-Time, Eclipse Edition 2014-2016: Eclipse studio 2014* [online]. 04.08.2014 [cit. 2018-05-12]. Dostupné z: <http://www.ni.com/download/labview-real-time-module-2014/4846/en/>.

[30] *Installing and updating the CDT: Eclipse studio online manual* [online]. [cit. 2018-05-12]. Dostupné z: <https://help.eclipse.org/mars/index.jsp?topic=\%2Forg.eclipse.cdt.doc.user\%2Fgetting_started\%2Fcdt_w_install_cdt.htm>.

[31] *FPGA Interface C API 2.0 - RedHat, Windows* [online]. 08.02.2011 [cit. 2018-05-12]. Dostupné z: <http://www.ni.com/download/fpga-interface-c-api-2.0/2616/en/>.

[32] *FPGA Interface C API Help: Errors* [online]. 2013, June 2013 [cit. 2018-05-13]. Dostupné z: <http://zone.ni.com/reference/en-XX/help/372928G-01/capi/errors/>.

Seznam symbolů, veličin a zkratek

API	Application Programming Interface
AXI	Advanced Extensible Interface
BLDC	Brushless DC electric motor
cRIO	CompactRIO
DIO	Digital Input Output
DLL	Dynamic Link Library
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FPGA	Field Programmable Gate Array
HIL	Hardware-In-the-Loop
HRT	Hard Real Time
HW	Hardware
LabVIEW	Laboratory Virtual Instruments Engineering Workbench
MDT	OpenModelica Development Tooling Eclipse Plugin
MIL	Model-In-the-Loop
ODE	Ordinary Differential Equations
OMC	OpenModelica Compiler
OMEdit	OpenModelica Connection Editor
OMPython	OpenModelica Python Interface
OMShell	OpenModelica Shell
OS	Operační Systém
OSMC	Open Source Modelica Consorium
OSS	Open Source Software
PC	Personal Computer
PIL	Processor-In-the-Loop
SDK	Software Devolpment Kit
SIL	Software-In-the-Loop
VI	Virtual Instrument
XML	Extensible Markup Language

Seznam příloh

A Obsah přiloženého CD

- `/` ..... Kořenový adresář přiloženého CD
  - `Datasheets` ..... Datasheety
    - `NI9401.pdf` ..... 8 DIO, 5 V/TTL karta pro cRIO
    - `NI-cRIO-9068.pdf` ..... NI cRIO-9068 Uživatelský manuál
  - `Diploma_thesis` ..... Diplomová práce - elektronická verze
    - `Latex` ..... Zdrojové soubory textového dokumentu
    - `HIL_MODEL_ELEKTROMECHANICKEHO_SYSTEMU` ..... DP elektronická verze
    - `HIL_MODEL_ELEKTROMECHANICKEHO_SYSTEMU_NAVOD` . Návod nastavení projektu
  - `Eclipse_Linux` ..... Adresář Eclipse studia pro OS Linux
    - `EclipseProjects` ..... Projektová složka Eclipse studia
      - `Sum` ..... Složka projektu sumátoru
        - `Sum` ..... Zdrojové soubory modelu Sum
        - `Sumkek` ..... Testovací zdrojové soubory modelu Sum
        - `Sum.fmu` ..... FMU modelu Sum
  - `Eclipse_WIN10` ..... Adresář Eclipse studia pro OS Windows 10
    - `FMU13apr2018` ..... Složka projektu importu FMU
      - `.settings` ..... Složka souborů s nastavení prostředí Eclipse studia
      - `Debug` ..... Složka ladění projektu
      - `inc` ..... Složka vložených souborů do projektu
      - `.cproject` ..... Soubor Eclipse studia
      - `.project` ..... Soubor Eclipse studia
      - `libSum.so` ..... Zkompilovaná knihovna pro NI cRIO 9068
      - `main.c` ..... Zdrojový program vytvoření instance FMU a C API rozhraní
      - `NiFpga.dll` ..... Knihovna rozhraní C API
  - `FMI_standard` ..... Soubory standardu FMI 2.0
    - `FMI_for_ModelExchange_and_CoSimulation_v2.0.zip` ..... FMI standard
  - `FMU_3rdParty` ..... Vzorové FMU od společnosti OpenModelica
  - `FMU_Compiler` ..... Kompiler FMU pro platformu NI cRIO 9068
  - `FMU_Compliance_checker` ..... FMU Compliance checker
    - `Compliance-Checker-3956.zip` .. Archiv s compliace checkerem pro různé OS
  - `FMU_Dymola` ..... FMU vyexportované z Dymoly
    - `model_soucet_real` ..... Model sumátoru s proměnnými typu Real
    - `model_soucet_v1` ..... Model sumátoru s proměnnými typu Integer
  - `LabVIEW_2017` ..... Projektová složka LabVIEW verze 2017
    - `C_API_LabVIEW_2017` ..... Vytvoření C API rozhraní
      - `FPGA Bitfiles` .. Bitové soubory vytvořené C API generátorem - defaultní složka
      - `FPGA Bitfiles C API 2014` Bitové soubory vytvořené C API generátorem - verze LabVIEW 2014
      - `FPGA Bitfiles C API 2017` Bitové soubory vytvořené C API generátorem - verze LabVIEW 2017
      - `C_API_proj.lvproj` ..... FPGA Projekt pro NI cRIO 9068
      - `FPGA.vi` ..... VI pro FPGA
  - `LabVIEW_AddOn` ..... Doplněk do LabVIEW

- `NI Labs FMI for Model Exchange Tutorial with sample FMU.zip` Tutoriál k doplňku
- `NI Labs-FMI For Model Exchange v2.zip` ... Instalační soubory doplňku do LabVIEW

`OpenModelica_Linux` ..... Adresář OpenModelicy pro OS Linux

- `Makefile upraveny` ..... Upravené soubory makefile pro kompilaci FMU
  - `Makefile_FINAL` ..... Finální verze makefile (tuto používat)
  - `Makefile_funkcni` . Funkční verze makefile (bez parametru -s v CFLAGS)
  - `Makefile_template` ..... Záloha funkční verze makefile
- `Sum` ..... Model sumátoru vytvořený v OpenModelica pro OS Linux

`Usefull_links.txt` ..... Odkazy na tutoriály, návody a odkazy ke stažení