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

KOSIMULACE V PROSTŘEDÍ OPENMODELICA A MATLAB-SIMULINK
OPEN-MODELICA AND MATLAB-SIMULINK CO-SIMULATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Adam Szymanik

VEDOUCÍ PRÁCE
SUPERVISOR
prof. Ing. Pavel Václavek, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Adam Szymanik ***ID:*** 195437
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Kosimulace v prostředí OpenModelica a Matlab-Simulink

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s modelovacím jazykem Modelica a simulačním SW OpenModelica a proveďte literární rešerši existujících řešení kosimulace a výměny modelů mezi Matlab-Simulink a simulátory založenými na Modelice.
2. Navrhněte řešení pro kosimulaci modelů mezi OpenModelica a Matlab-Simulink v rámci jednoho počítače i při distribuovaném výpočtu na více počítačích (síťové řešení).
3. Vytvořte softwarové řešení pro kosimulaci mezi Matlab-Simulink a OpenModelica zajišťující výměnu dat, synchronizaci simulace, ošetření chybových stavů a uživatelské rozhraní.
4. Vytvořte sadu simulačních příkladů a demonstrujte na nich funkčnost navrženého řešení pro kosimulaci.

DOPORUČENÁ LITERATURA:

Fritzon, P.:Introduction to Object-Oriented Modeling, Simulation and Control with Modelica, 2012

Bastian, J: Master for Co-Simulation Using FMI, 2011

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: prof. Ing. Pavel Václavek, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

Abstrakt

Tato práce řeší kosimulaci mezi modely v prostředí OpenModelica a Matlab-Simulink. První část práce je věnována modelovacímu jazyku Modelica, simulačnímu SW OpenModelica a standardu FMI, pomocí kterého je možné exportovat modely z jednoho prostředí a importovat prostředí jiného, které tento standard podporují. Další část se věnuje návrhu vlastích bloků pro kosimulaci a jejich realizaci. V poslední kapitole jsou tří příklady pro ověření funkčnosti řešení.

Klíčová slova

Kosimulace, Matlab, Simulink, Modelica, OpenModelica

Abstract

This thesis solves co-simulation between OpenModelica and Matlab-Simulink. In the first part Modelica language, OpenModelica software and FMI standard are described. In the next part is proposal of solution through custom blocks and realization of those blocks. In last chapter are three examples to verify functionality of my solution.

Keywords

Co-Simulation, Matlab, Simulink, Modelica, OpenModelica

Bibliografická citace:

SZYMANIK, Adam. *Kosimulace v prostředí OpenModelica a Matlab-Simulink.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/127076. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce prof. Ing. Pavel Václavek, Ph.D.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma Kosimulace v prostředí OpenModelica a Matlab-Simulink jsem vypracoval samostatně pod vedením vedoucí/ho bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **7. června 2020** ……
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce prof. Ing. Pavlu Václavkovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: **7. června 2020** ……

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

ÚVOD

Tato bakalářská práce se zabývá kosimulací modelů mezi programy OpenModelica a Matlab-Simulink. V první části je popis modelovacího jazyk Modelica a prostředí OpenModelica, které využívá jazyk Modelica a je volně dostupné. V druhé částí je popsán standard Functional Mock-up Interface, který definuje rozhraní pro výměnu modelů nebo kosimulaci mezi různými simulačními nástroji. Třetí kapitola popisuje návrh vlastích bloků pro komunikaci mezi oběma programy. Ve čtvrté kapitole je popsána realizace bloků a řešení vzniklých problémů. Poslední kapitola se věnuje praktické ukázce vytvořených bloků.

Kosimulace je propojení dvou nebo více simulačních nástrojů, kde každý nástroj má na starost svoji část systému a pouze si vyměňují data. Výměna dat mezi modely probíhá pouze v určených komunikačních bodech. Kosimulace může probíhat i s využitím standardu FMI, kdy se model exportuje z jednoho prostředí a vloží se do prostředí jiného.

Motivací pro tuto práci je možnost propojení akauzálních modelů v prostředí OpenModelica s Matlab-Simulinkem. Akauzalní modelování může být v některých případech jednodušší než kauzální modelování v Matlab-Simulinku. Navíc při využití distribuovaného výpočtu na více počítačích, je k dispozici větší výpočetní výkon, který je při složitých simulacích velmi důležitý.

1. JAZYK MODELICA A OPENMODELICA

Modelica je volně dostupný, objektově orientovaný jazyk, kde modely jsou popsány pomocí diferenciálních, diferenčních a algebraických rovnic. Jazyk Modelica je vhodný i pro modelování složitých, komplexních systémů, které propojují různá fyzikální odvětví [1].

Existuje řada programů, které využívají Modelicu pro modelování a simulování systémů. Mezi zpoplatněné systémy patří například Dymola od Dassault Systèmes, MapleSim od Maplesoft, Simplorer od ANSYS a další. Do skupiny volně dostupných programů se řadí JModelica.org, OpenModelica [2].

1.1 Jazyk Modelica

Na začátku kódu každého modelu se nachází klíčové slovo *model*, za kterým následuje název modelu. V další části je deklarace proměnných modelu, za kterou je klíčové slovo *equation* oddělující deklarace proměnných a rovnice popisující model. Ukončení definice modelu se provede pomocí klíčového slova *end*, které je opět následováno názvem modelu [1].

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

Proměnné můžou být různých datových typů jako je Boolean (hodnota *true* nebo *false*), Integer (celá čísla), Real (desetinná čísla), String (řetězec znaků). Pro každou proměnnou můžeme definovat řadů atributů např.: minimum, maximum, počáteční hodnotu aj. Všechny deklarované hodnoty se můžou v průběhu simulace měnit. V případě, že před datový typ přidáme klíčové slovo *constant* nebo *parameter*, potom se tato hodnota v průběhu simulace měnit nemůže. Rozdíl je v tom, že když nadefinujeme hodnotu jako *constant,* tak není určena k tomu, aby se měnila. Z tohoto důvodu některé grafické prostředí neumožní tuto hodnotu měnit nebo ji ani nezobrazí. Dalším rozdílem je, že parametry můžou obsahovat matematický výraz, podle kterého se hodnota parametru vypočítá a konstanty musí mít definovanou hodnotu [3].

```
1    model model
2        parameter Real p1 = 3;
3        parameter Real p2 = p1+1; //správně
4        constant Real c2 = p1+2;  //chyba, u konstanty nesmí být výpočet
5    end ObsahKruhu;
```

Důležitou součástí každého kódu jsou komentáře, díky kterým se program stává přehlednějším a jednodušším na pochopení. V jazyce Modelica existují dva typy komentářů. Prvním typem jsou komentáře, které jsou překladačem jazyka Modelica ignorovány. Může se jednat o komentář jednořádkový, který začíná symbolem dvojitého lomítka nebo víceřádkový který začíná symbolem „/*“ a končí „*/“[4].

Druhým typem komentáře jsou tzv. dokumentační komentáře, které překladačem jazyka Modelica nejsou ignorovány. Tyto komentáře se musí psát na konec řádku za definici modelu, proměnných atd. a přiřazují se jako jejich popis. Tento komentář napsaný za názvem modelu, může být využit při vyhledávání modelu [1].

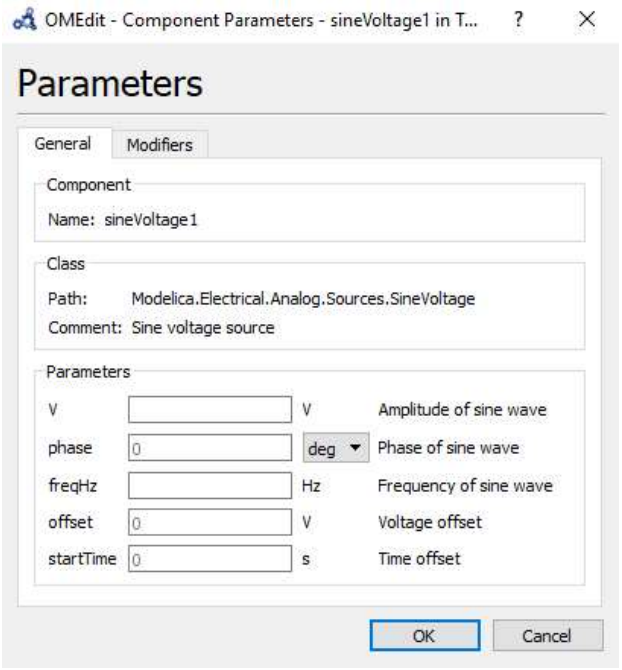
```
1   model ObsahKruhu "Popis modelu"
2       constant Real pi=3.14159265359 "konstanta pi";
3       Real r (min=0); //polomer kruhu
4       Real S "Obsah kruhu";
5   equation
6       S = pi*r*r;
7   end ObsahKruhu;
```

Rovnice v jazyce Modelica je možné zapsat několika způsoby a výsledek bude pořád stejný. Rovnici S=pi*r*r, z výše uvedeného příkladu, lze zapsat ve tvaru S/pi = r*r, protože operátor „=“ neznamená přiřazení, jak je tomu v jazyce C/C++, ale znamená rovnost výrazů na levé a pravé straně [4].

1.2 OpenModelica

OpenModelica je open-source prostředí, využívající jazyk Modelica, pro modelování a simulaci systémů. V OpenModelice se nachází několik nástrojů jako např.: OMEdit, OMNotebook, OMC, OMShell, OMOptim atd. [5].

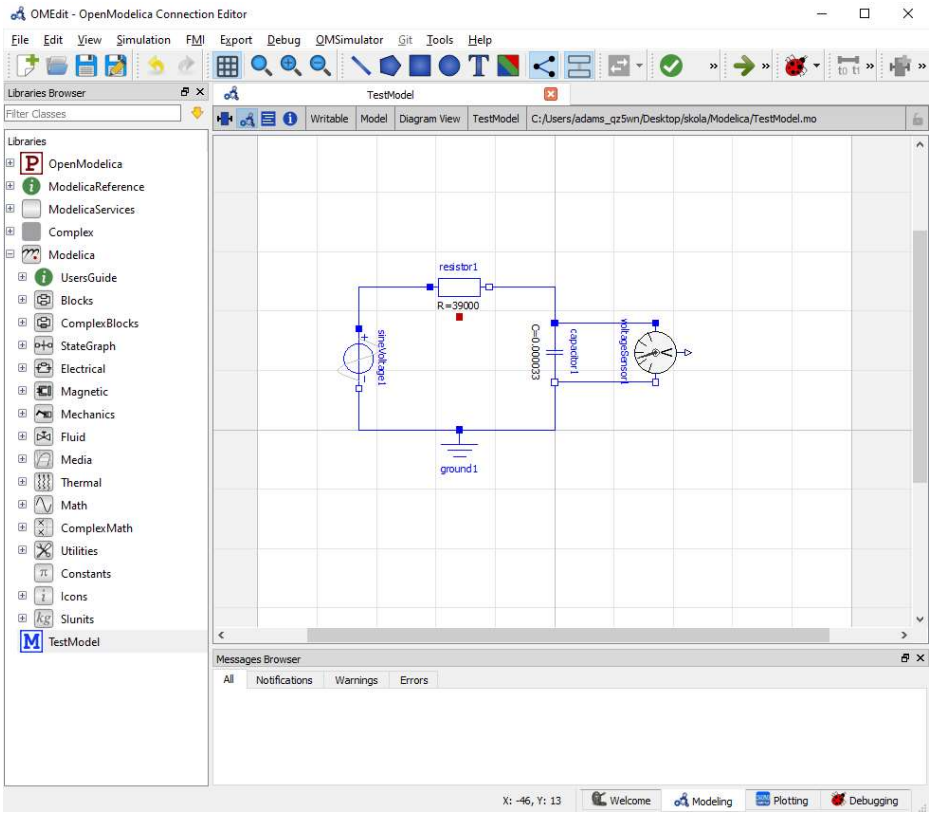
OMEdit je zkratkou pro OpenModelica Connection Editor. Je to nástroj pro grafické a textové vytváření a editaci modelů. Jak je u většiny programů zvykem, v horní části se nachází nástrojová lišta. V levé části se nachází knihovny pro výběr součástek, zdrojů, snímačů atd. Ve spodní části je prohlížeč zpráv, kde se zobrazují oznámení, výstražné a chybové zprávy. Největší část obrazovky zabírá pracovní oblast, do které přidáme prvek přetáhnutím z knihovny (pouze v grafickém zobrazení). Parametry prvku můžeme změnit po dvojkliku na vybraný prvek. Výchozí hodnoty parametrů jsou napsány šedou barvou, pokud ale u parametru žádná hodnota není, je nutné ji nastavit (viz Obrázek 1.1). V textovém zobrazení modelu za klíčovým slovem *model* a názvem modelu je vidět objekty přidané do modelu a je možné provádět jejich úpravy. Grafické a textové zobrazení je na obrázku 1.2 a 1.3.



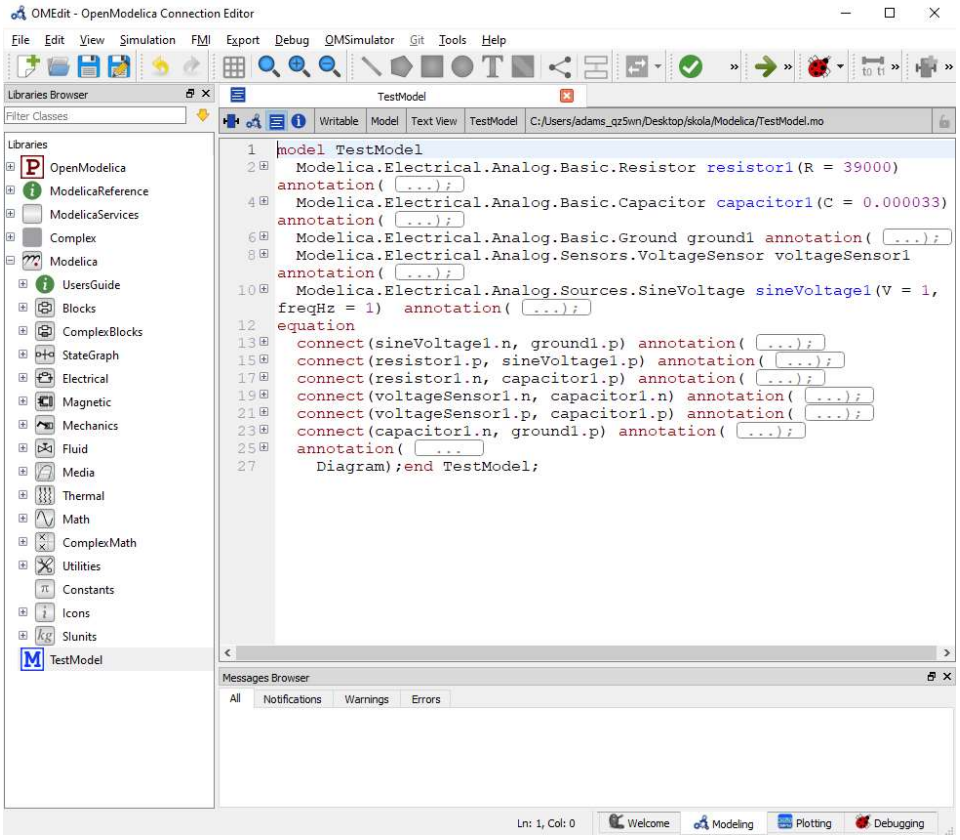
Obrázek 1.1 OMEdit – nastavení parametrů harmonického zdroje napětí

OMNotebook je program pro vytváření WYSIWYG (What You See Is What You Get) interaktivních knih. OMNotebook umožňuje do jednoho dokumentu umístit zdrojový kód programu (modelu) a dokumentaci, která usnadňuje jeho pochopení. Tato možnost je užitečná, když se učíme nový programovací jazyk. Za tímto účelem byla vytvořena vzdělávací kniha DrModelica, která je určena k naučení modelování v jazyce Modelica [5].

Advanced Interactive OpenModelica Compiler (OMC) je překladač, který s pomocí tabulky symbolů obsahující definice tříd, funkcí a proměnných překládá model z jazyka Modelica do jazyka C [6].



Obrázek 1.2 OMEdit – grafické zobrazení modelu



Obrázek 1.3 OMEdit – textové zobrazení modelu

2. FUNCTIONAL MOCK-UP INTERFACE

Functional Mock-up Interface (FMI) je standard podporující výměnu modelů a kosimulaci dynamických systému mezi různými modelačními a simulačními nástroji [7].

První verze FMI byla publikována v roce 2010, kterou v červenci roku 2014 následovala verze FMI 2.0. Vývoj standartu FMI byl zahájen firmou Daimler AG. V dnešní době vývoj pokračuje za účastí 16 firem a výzkumných ústavů. Aktuálně je FMI podporován více než 100 nástroji a je využíván řadou firem v Evropě, Asii a Severní Americe [8].

FMI podporuje dvě možnosti exportu modelu, první možnost je FMI pro výměnu modelů (FMI for Model Exchange) a druhou je FMI pro kosimulaci (FMI for co-simulation). V obou případech exportu modelu vznikne Functional Mock-up Unit (FMU). Jedná se o archiv ZIP s příponou „*.fmu“ s předdefinovanou strukturou, který obsahuje popis modelu v soubor XML a kód modelu jako zdrojový kód v jazyce C nebo v podobě binárních souborů. Dynamic Link Library (*.dll) pro operační systém Windows a Shared Object (.so) pro systémy Linux. Celá kapitola vychází z [7].

2.1 Popis Modelu

Statické informace FMU jsou uložené v XML souboru. Jedná se hlavně o definici proměnných a jejich atributy (název proměnné, jednotky, počáteční hodnota atd.). Struktura tohoto dokumentu je definována souborem *fmiModelDescription.xsd*. Popis modelu může obsahovat následující položky:

- ModelExchange
- CoSimulation
- UnitDefinition
- TypeDefinition
- LogCategories
- DefaultExperiment
- VendorAnotations
- ModelVariables
- ModelStructure

Položky *ModelExchange* a *CoSimulation* slouží pro identifikaci typu FMU a soubor XML musí obsahovat alespoň jednu z nich. Další povinné položky jsou *ModelVariables* a *ModelStructure*, ostatní položky jsou nepovinné.
ModelVariables obsahuje seznam proměnných (*ScalarVariables*), kde každá proměnná musí mít určený datový typ (*Real, Integer, atd.*) a atributy *name* a *valueReference*. Další atributy nejsou povinné.

ModelStructure definuje strukturu modelu s ohledem na rovnice modelu. Dělí se na tři části Outputs, D*erivatives* a *InitialUnknowns.* V seznamu *Outputs* se nachází proměnné, které jsou výstupem modelu. Všechny proměnné, které jsou derivaci stavu, se nachází v *Derivatives*. Proměnné, které nemají definovanou inicializační hodnotu, se nachází v *InitialUnknowns.*

UnitDefinition obsahuje definici jednotek. Každá jednotka musí obsahovat jedinečné jméno a *DisplayUnit* (zobrazovaná jednotka). Dodatečně může být pomocí *BaseUnit* nastaven převod jednotky na jednotky SI.

Pomocí *TypeDefinition* se definují datové typy jednotek, přičemž každý datový typ musí obsahovat jméno, které nesmí být stejné jako jiný datový typ nebo *ScalarVariables* a jeden ze základních datových typů (Real, Integer, Boolean, String, Enumeration).

V *LogCategories* se nachází seznam kategorií, které můžou být použité jako kategorie pro záznam zpráv a událostí pomocí funkce *„logger“.* Pokud to vybraný nástroj umožňuje, měly by se požívat standardizovaná jména kategorií (*logEvents, logStatusError, logStatusDiscard, ad.*).

V položce *DefaultExperiment* se nachází výchozí hodnoty začátku a konce simulace, tolerance a velikosti kroku. Některé nástroje můžou tuto položku ignorovat.

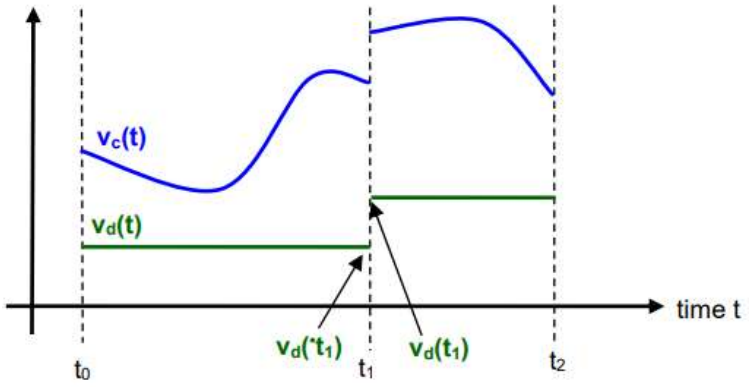
VendorAnotations obsahuje poznámky, které jsou určené pouze pro vybraný nástroj, ostatní nástroje tuto poznámku ignorují.

2.2 Matematický popis

Rozhraní FMI pro výměnu modelu bylo navrženo pro řešení systému diferenciálních, diferenčních a algebraických rovnic. Tyto systémy se nazývají hybridní systémy diferenciálních rovnic (hybrid ODE, ordinary differential equations).

Systémy podporované FMI jsou po částech spojité a nespojitosti můžou nastat v časech událostí. Mezi těmito událostmi můžou být proměnné konstantní nebo spojité. V případě, že proměnná mění hodnotu pouze v okamžiku události, je nazvána diskrétní. Jestliže mění hodnotu i mezi časy událostí je nazvaná spojitou.

Na obrázku 2.1 je znázorněn průběh proměnných v po částech spojitém systému. $v_c(t)$ označuje spojitou proměnnou a $v_d(t)$ diskrétní proměnnou. V časech t_0, t_1 a t_2 nastaly události, ve kterých můžou nastat nespojitosti, a proto se definují dvě hodnoty $v_c^+(t_i)$ (limitní hodnota zprava) a $v_c^-(t_i)$ (limitní hodnota zleva), obdobně pro v_d.



Obrázek 2.1 Průběh proměnných v po částech spojitém systému [7]

2.3 Funkce FMI

Pro definici rozhraní FMU existují tři hlavičkové soubory. V těchto souborech všechny funkce a definice datových typů začínají předponou „fmi2".

Prvním souborem je „*fmi2TypesPlatform.h*" obsahuje definice datových typů vstupních a výstupních argumentů funkcí. V „*fmi2FunctionsTypes.h*" se nachází definice všech prototypů funkcí FMU. Poslední hlavičkový soubor *„fmi2Functions.h"* obsahuje prototypy všech funkcí a vkládá předchozí dva soubory.

Všechny funkce vrací hodnotu *fmi2Status* datového typu enumeration, která označuje výsledek volání funkce. Definice *fmi2Status* je následující:

```
typedef enum {
        fmi2OK,
        fmi2Warning,
        fmi2Discard,
        fmi2Error,
        fmi2Fatal,
        fmi2Pending} fmi2Status;
```

Význam jednotlivých hodnot:

- fmi2OK – všechno je v pořádku
- fmi2Warning – všechno není úplně v pořádku, ale výpočet může pokračovat
- fmi2Discard má jiný význam pro výměnu modelu a kosimulaci
  - Výměna modelu – zmenšit velikost kroku výpočtu
  - Kosimulace – slave není schopen poskytnout požadované informace
- fmi2Error – vyskytla se chyba, simulace nemůže pokračovat, simulace může být obnovena z uloženého stavu pomocí funkce *fmi2SetFMUState*, jinak je třeba zavolat funkci *fmi2FreeInstance* nebo *fmi2Reset*
- fmi2Fatal – výpočty modelu jsou poškozené pro všechny instance FMU

- fmi2Pending – tento stav je možný pouze pro FMU pro kosimulaci a znamená, že slave začal počítat a ihned vrátil hodnotu, master musí pomocí funkce fmi2GetStatus určit, jestli slave dokončil výpočet

V případě, že fmi2Status má hodnotu fmi2Warning, fmi2Discard, fmi2Error nebo fmi2Fatal je zavolána funkce „logger“.

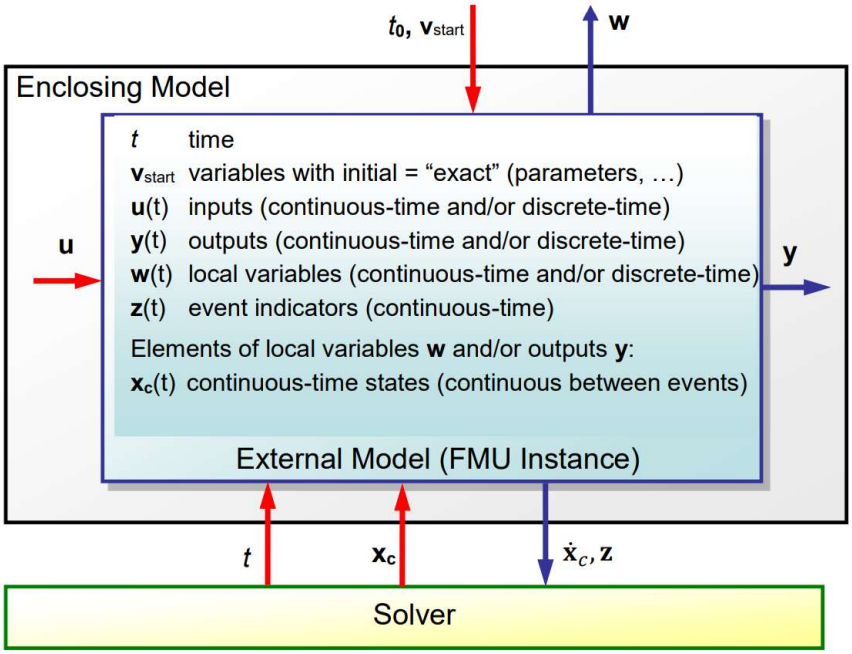
Pro vytvoření instance modelu je třeba zavolat funkci *fmi2Instantiate*. Inicializace modelu se provádí pomocí funkcí *fmi2SetupExperiment, fmi2EnterInitializationMode*, *fmi2ExitInitializationMode*. Nastavení proměnných se provede pomocí funkcí *fmi2SetReal, fmi2SetInteger* atd. Přečtení proměnných se provádí funkcemi *fmi2GetReal, fmi2GetInteger* atd. K ukončení simulace slouží funkce *fmi2Terminate*. Funkcí *fmi2Reset* resetujeme FMU do podoby stejné jak po vytvoření instance funkcí *fmi2Instantiate*. *fmi2FreeInstance* ukončí instanci FMU a uvolní alokovanou paměť.

Podrobný popis argumentů funkcí a dalších funkcí je v dokumentaci standardu FMI.

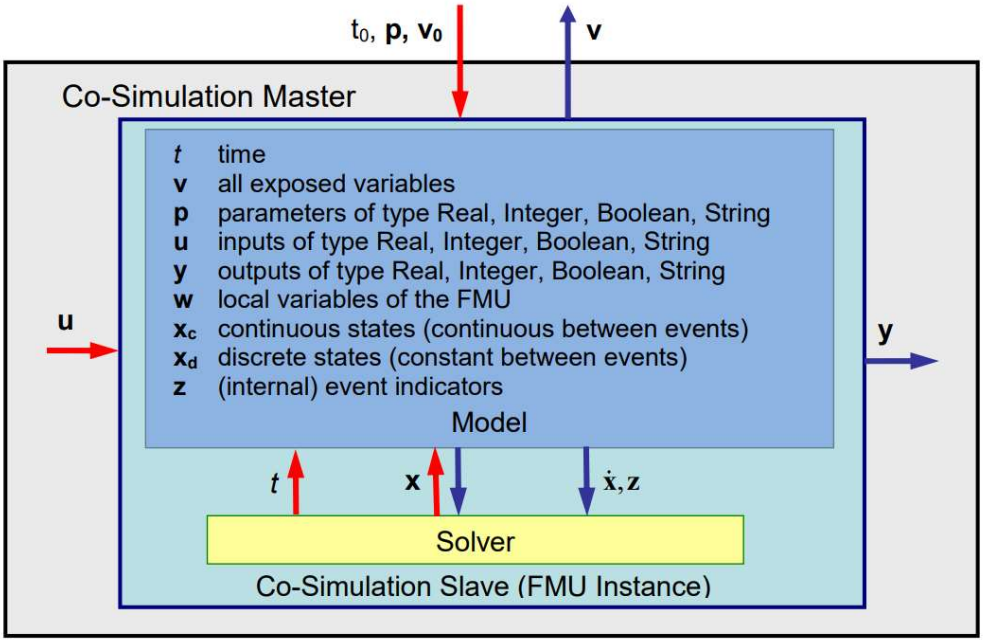
2.4 FMI pro výměnu modelů a kosimulaci

Rozdíl mezi FMI pro výměnu modelu a FMI pro kosimulaci je, že při výměně modelu probíhá výpočet diferenciálních, diferenčních a algebraických rovnic v prostředí kde je model importován. Při kosimulaci je řešitel (solver) původního prostřední exportován do FMU a použit pro výpočet rovnic nebo mezi prostředími probíhá výměna hodnot proměnných.

Na obrázcích 2.2 a 2.3 je zobrazeno schéma FMU pro výměnu modelu, resp. kosimulaci a jejich datové toky. Červené šipky označují informace vstupující do FMU a modré vystupující z FMU.

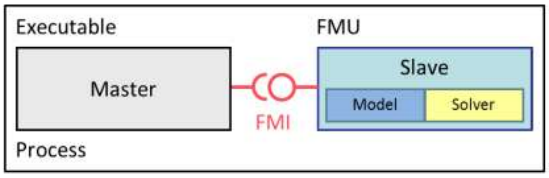


Obrázek 2.2 Datové toky mezi FMU pro výměnu modelu a simulačním prostředím [7]



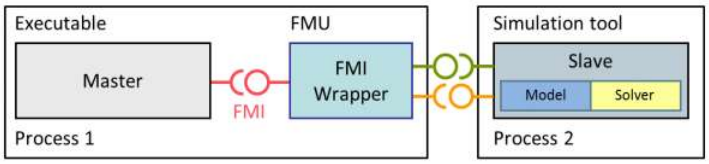
Obrázek 2.3 Datové toky mezi FMU pro kosimulaci a simulačním prostředím [7]

Obrázek 2.4 znázorňuje použití FMI pro kosimulaci pro spojování modelů, které byly exportovány spolu se solvery jako spustitelný kód.

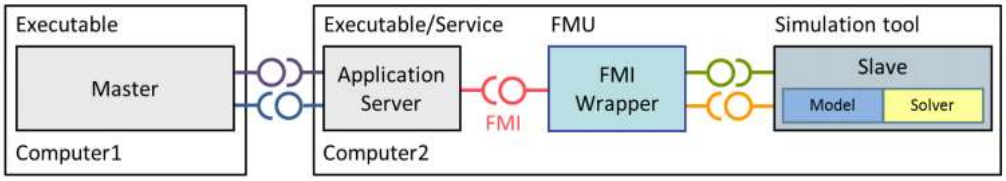


Obrázek 2.4 FMI pro kosimulaci v jednom procesu [7]

Na obrázcích 2.5 a 2.6 je FMI pro kosimulaci použito pro propojení simulačních nástrojů. Na obrázku 2.6 probíhá kosimulace na více počítačích a každý může používat jiný operační systém. Komunikaci mezi počítači zajišťuje tzv. master a může probíhat například pomocí síťového komunikačního protokolu TCP/IP. Tato komunikace není součástí standardu FMI.



Obrázek 2.5 FMI pro kosimulaci s propojováním simulačních prostředí [7]



Obrázek 2.6 FMI pro kosimulaci na více počítačích [7]

2.5 Použití FMU v Matlab-Simulink

Matlab-Simulink podporuje import modelů jako FMU pomocí bloku *FMU Import*. Tento blok má jediný parametr, kterým je soubor s příponou „*.fmu“. Vytvořený blok se přizpůsobí typu vybraného FMU (kosimulace/výměna modelů). V případě, že FMU obsahuje obě možnosti je možné vybrat, kterou verzi chceme použít. Importované FMU může být verze 1.0 i 2.0. Tento blok je možné použít od verze R2017b [11] .

3. NÁVRH ŘEŠENÍ

Pro řešení kosimulace modelů mezi Matlab-Simulink a OpenModelica jsem se rozhodl pro vytvoření bloků pro výměnu dat mezi modely pomocí síťového spojení. Toto řešení poskytne možnost kosimulace v rámci jednoho i více počítačů.

3.1 Komunikace

Komunikace mezi modely bude probíhat pomocí síťového spojení s využitím protokolu TCP. Každý blok vytvoří síťový socket typu PULL nebo PUSH podle toho, jestli bude data přijímat nebo odesílat. Nevýhodou komunikace typu PUSH/PULL je pouze jednosměrný tok dat. Pro účel této práce je tento typ spojení dostatečný a navíc rychlejší, protože vysílací blok nemusí čekat na odpověď.

Pro komunikaci využiji knihovnu ZeroMQ, která je popsána v další části kapitoly.

Pro správnou synchronizaci mezi modely je důležité, aby simulace probíhaly s pevným krokem. Při proměnlivém kroku nelze zajistit, aby obě prostředí zvolila vždy stejně velký krok simulace a při stejné délce se může lišit počet kroků.

ZeroMQ

ZeroMQ je asynchronní knihovna pro zasílání zpráv, se zaměřením na distribuované nebo souběžně běžící aplikace. Tato knihovna poskytuje frontu zpráv a může fungovat bez zprostředkovatele zpráv [10].

Knihovnu ZeroMQ můžeme využívat v různých programovacích jazycích. Její základní částí je knihovna *libzmq,* určená pro jazyk C. Pro použití v jiných jazycích existují vazby, které umožní používat funkce *libzmq*. ZeroMQ podporuje jazyk C++, C#, Erlang, F#, Java, Python, Haskell, Perl a mnoho dalších. Kdokoliv může vytvořit vazbu pro další programovací jazyky [10].

ZeroMQ podporuje řadu protokolů jako je TCP, IPC, PGM, NORM, inproc a od verze 4.2 podporuje i UDP. Základním protokolem, který podporují všechny jazyky, je protokol TCP. Pouze u některých programovacích jazyků je podpora jiných protokolů. Seznam protokolů a podporovaných jazyku je dostupný na [10].

Následující příklad ukazuje, jak vytvořit socket, který odpovídá na dotazy. Před vytvořením síťového socketu se musí vytvořit kontext. Socket se vytváří pomocí funkce *zmq_socket*, kde první parametr je ukazatel na dříve vytvořený kontext a druhým je typ socketu. K připojení na určitý port se používá funkce *zmq_bind* nebo *zmq_connect*. V nekonečné smyčce socket čeká na příchozí zprávu. Příjem zpráv zajišťuje funkce *zmq_recv*. Odpověď odešleme pomocí funkce *zmq_send*.

```
#include <zmq.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

int main(void) {
       void *context = zmq_ctx_new();
       void *responder = zmq_socket(context, ZMQ_REP);
       int rc = zmq_bind(responder, "tcp://*:5555");
       assert(rc == 0);

       while (1) {
              char buffer[10];
              zmq_recv(responder, buffer, 10, 0);
              printf("Received Hello\n");
              zmq_send(responder, "World", 5, 0);
       }
       return 0;
     }
```

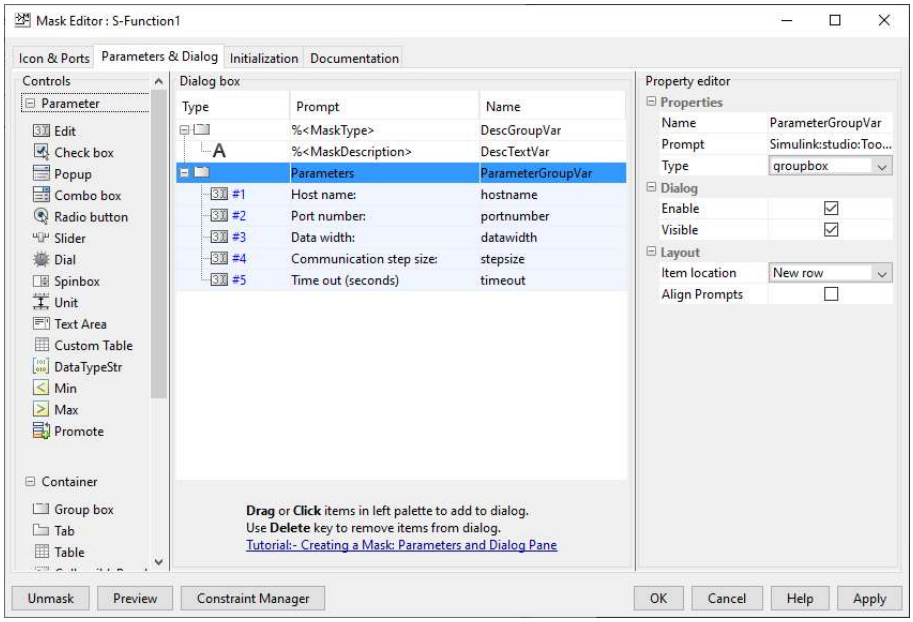
3.2 Ošetření chybových stavů

Při inicializaci simulace proběhne kontrola simulačních dat jako je délka a krok simulace, počet odesílaných, resp. přijímaných dat. V průběhu simulace se bude kontrolovat pouze spojení. V případě, že po určenou dobu nepřijdou data, bloky vyhodnotí tento stav jako chybu spojení.

3.3 Matlab-Simulink

Pomocí S-Funkce vytvořím v Simulinku samostatné bloky pro příjem a odesílání dat. Blok pro příjem dat bude mít jeden výstupní port s nastavitelnou šířkou. Počet příchozích dat se nastaví v parametrech bloku. Vysílací blok bude mít podobné vlastnosti jako přijímací.

Pro možnost zadávání parametrů do S-Funkce je potřeba vytvořit masku. Pro úpravu hodnoty parametru lze využít jednu z několika možností jako editační nebo zaškrtávací políčko, vyskakovací okno s přednastavenými možnostmi a další. Všechny možnosti jsou zobrazeny na obrázku 3.1.



Obrázek 3.1 Editor masky S-Funkce

Každý blok bude obsahovat parametry pro zadání adresy a portu pro komunikaci, počtu komunikovaných hodnot, krok simulace a čas pro ukončení simulace v případě přerušení komunikace.

3.4 OpenModelica

Podobně jako v Simulinku vytvořím v OpenModelice vlastní bloky pro komunikaci hodnot. Tyto bloky budou volat externí funkce napsané v jazyce C, pomocí kterých bude zajištěna komunikace. Volání externích funkcí v jazyce Modelica bude popsáno v další části kapitoly.

```
parameter String Address;
parameter Integer Port;
```

Komunikační bloky budou mít stejné parametry jako bloky v Simulinku. Ve vybraném bloku vytvořím parametr klíčovým *parameter*, za kterým následuje datový typ a název. Parametry patří mezi proměnné a konstanty před část *equation*. Při zadávání parametru datového typu *string* se musí tento řetězec zadat do uvozovek.

3.4.1 Externí funkce v jazyce Modelica

Jazyk Modelica podporuje externí funkce v jazyce C a FORTRAN 77. Podpora dalších jazyků jako např. C++, Fortran 90 bude možná přidána v dalších verzích. Příklad volání externí funkce v jazyce C z OpenModelicy:

```
function ExtFunkce
input Real u;       //seznam vstupů/výstupů
output Real y;

external "C" y = vynasob2(u) annotation(
Include = "#include \"funkce.c\"");
end ExtFunkce;
```

Externí funkce se volají pomocí klíčového slova *external*, za kterým následuje specifikace jazyka (C nebo FORTRAN 77) a pak samotné volání funkce. Za klíčovým slovem *annotation* se uvádí soubory, které jsou potřebné pro vykonání volané funkce. Jedná se o zdrojové soubory, hlavičkové soubory, statické nebo dynamické knihovny.

Ve výše uvedeném příkladu se návratová hodnota funkce zapíše do proměnné y. V případě, že funkce v jazyce C má více výstupních hodnot, proměnné, do kterých se zapisují tyto hodnoty, se uvádí jako argumenty funkce a v samotné funkci vystupují jako ukazatele. V následující tabulce je přehled datových typů vstupních a výstupních argumentů.

Tab. 3.1 Porovnání datových typů argumentů funkcí v jazyce Modelica a jazyce C [4]

Modelica	C	
	Vstup	Výstup
Real	double	double *
Integer	int	int *
Boolean	int	int *
String	const char *	const char **
Enumeration type	int	int *

4. REALIZACE

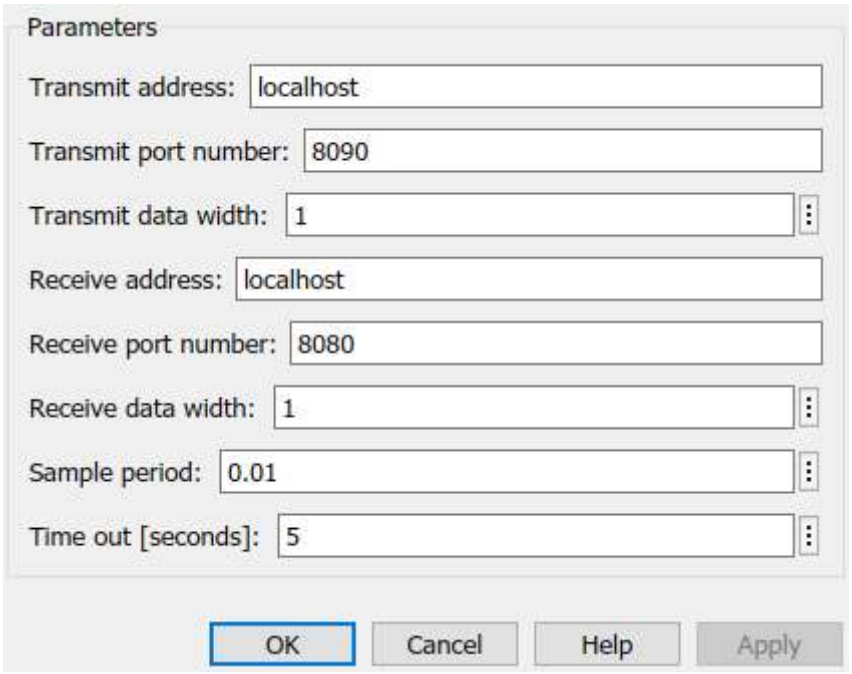
V následující kapitole je popsána výsledná realizace komunikačních bloků v Matlab-Simulinku a OpenModelice a případné odlišnosti od návrhu.

4.1 Matlab-Simulink

Při prvních testech funkčnosti vznikl problém, kdy oba simulátory vykonávaly nejdřív přijímací blok a následně vysílací blok. Toto pořadí vykonávání vedlo vždy k selhání simulace, protože nikdy nepřišla žádná data. Z tohoto důvodu jsem se rozhodl vytvořit v Matlab-Simulinku blok, který bude zajišťovat odeslání i příjem dat v předem určeném pořadí, aby nedošlo k zastavení simulace nekonečným čekáním na data.

Při inicializaci bloku jsem využil funkci *ssSetInputPortDirectFeedThrough*, aby Simulink nejdřív spočítal hodnotu na vstupním portu, protože chci nejdřív odeslat data a až potom přijmout. Využití této funkce způsobilo, že při zapojení bloku do uzavřené smyčky vznikla algebraická smyčka. Tento problém jsem vyřešil zpožděním signálu na výstupu, jak je zobrazeno na obrázku 5.5 a 5.9.

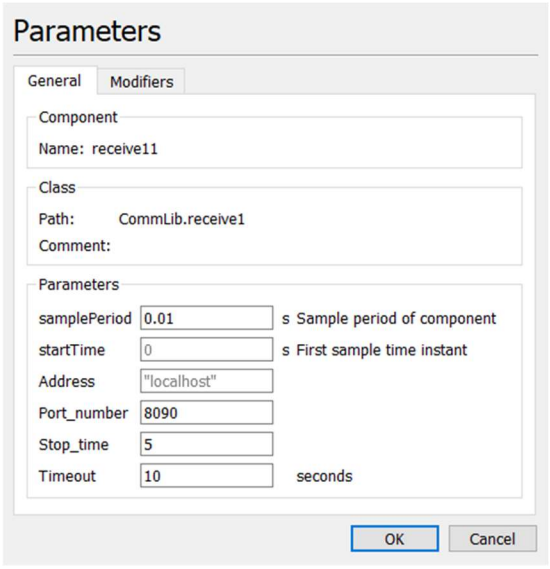
Ve vytvořené S-Funkci využívám funkce ze souborů *mdlclient.cpp* a *statcal_util.cpp*, které jsem převzal z příkladu kosimulace mezi dvěma modely v Matlab-Simulinku dostupného z [9]. Z tohoto příkladu jsem převzal i skripty *SetEnvVariable.m* a *buildCommLib.m*, které jsem upravil pro účely této práce.



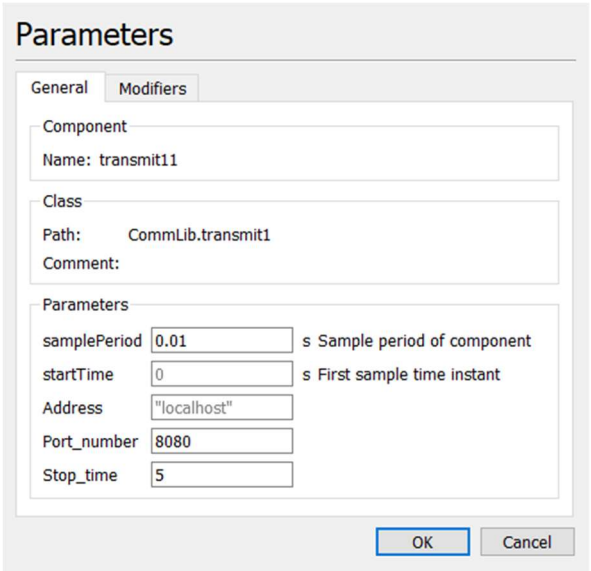
Obrázek 4.1 Parametry komunikačního bloku v Matlab-Simulink

4.2 OpenModelica

Pro kosimulaci modelu v OpenModelica s modelem v Matlab-Simulink jsem vytvořil knihovnu *CommLib*, ve které se nachází vlastní bloky pro odesílání a příjem dat a funkce, které tyto bloky využívají.



Obrázek 4.2 Parametry přijímacího bloku v OpenModelica



Obrázek 4.3 Parametry vysílacího bloku v OpenModelica

Na obrázcích 4.2 a 4.3 je zobrazeno nastavení parametrů přijímacího, resp. vysílajícího bloku. Blok pro příjem dat obsahuje navíc parametr timeout, pro vyhodnocení poruchy spojení. V porovnání s blokem v Simulinku, nemají parametr pro nastavení počtu přijímaných a odesílaných dat a obsahují navíc parametr startTime, který vznikl děděním bloku *discreteBlock*. Počet komunikovaných dat je pevně

nastaven pro každý blok podle počtu vstupů/výstupů. Další informace k této problematice jsou v kapitole 4.6.

4.3 Inicializace

Na začátku simulace proběhne inicializace, při které vytvářím sockety pro komunikaci dat a obě strany si mezi sebou vymění parametry simulace a počet komunikovaných dat. V případě, že se parametry liší simulace neproběhne.

Krok simulace, počet odesílaných a přijímaných dat se zadává v parametrech bloku. Konečný čas simulace lze v S-Funkci zjistit funkcí *ssGetTFinal*. Na obrázku 4.2 a 4.3 jsou zobrazeny parametry bloků v OpenModelice, které musí mít oproti S-Funkci navíc parametr koncového času simulace, protože jazyk Modelica nemá funkce jako *ssGetTFinal*.

Z důvodu jednosměrné komunikace parametry simulace může kontrolovat pouze přijímací blok, vysílací blok nemá přístup k parametrům druhé strany. To zapříčiňuje, že se chyba vyhodnotí pouze v jednom prostředí a druhé prostředí se snaží pokračovat v simulaci. Tento nedostatek tohoto řešení by se dal odstranit použitím obousměrné komunikace, kdy blok, který zjistí chybu, odešle zpátky informaci a výskytu chyby.

4.4 Vyhodnocení chyb

V OpenModelice vznikají chyby v externí funkci a každá chyba má přidělenou vlastní návratovou hodnotu. Tato hodnota je zapsána do proměnné v jazyce Modelica, kde probíhá i její kontrola a případné přerušení simulace.

```
if Err == 1 then
      assert(Err == 0,"Different data width", level =
AssertionLevel.error);
elseif Err == 2 then
      assert(Err == 0,"Different step time", level =
AssertionLevel.error);
elseif Err == 4 then
      terminate("End of simulation");
end if;
```

K ukončení simulace lze využít funkce *assert* a *terminate*. Funkce assert má tři argumenty, prvním je podmínka, která musí být splněna pro úspěšnou simulaci. Druhým argumentem je text, který se vypíše v případě nesplnění podmínky a posledním je úroveň s jakou tato funkce ovlivňuje simulaci. Může se jednat o varování, které neukončí probíhající simulaci nebo chybu, která simulaci ukončí. Použitím funkce *terminate* dojde k úspěšnému ukončení simulace, a to je při vyhodnocování chyb nežádoucí. Funkce *terminate* má pouze jeden parametr a to zprávu, kterou zobrazí po ukončení [4].

Ukončení funkcí *terminate* využívám pouze v případě obdržení příznaku konce simulace, který Matlab-Simulink odešle na konci simulace. Tato vlastnost, ale není plně funkční, protože v Modelice není ekvivalent funkce *mdlCleanupRuntimeResources*, která se provede při každém ukončení simulace a z OpenModelicy nelze tento příznak odeslat při chybném ukončení.

V Matlab-Simulinku taky existují dvě možnosti pro ukončení simulace. Funkce *ssSetStopRequest* nastaví příznak pro zastavení simulace a po skončení aktuálního kroku se simulace zastaví. Pro vyhodnocení chyb je vhodnější využít funkci *ssSetErrorStatus*, která umožňuje vypsat i chybové hlášení.

4.5 Způsob komunikace dat

Při návrhu komunikace jsem vycházel z příkladu kosimulace mezi dvěma modely v Matlab-Simulinku, který je dostupný z [9]. V tomto příkladu komunikace probíhá pomocí knihovny ZeroMQ. Pomocí této knihovny lze odeslat pouze řetězec a při simulaci chceme většinou posílat reálná čísla případně celá čísla. Tento problém lze vyřešit pomocí funkce *memcpy*. Touto funkcí můžeme zkopírovat určený počet bytů ze zdrojového do cílového umístění. Prvním argumentem je cílový ukazatel, druhým je zdrojový ukazatel a posledním je počet bytů, které se mají zkopírovat.

Na následujícím příkladu je zkopírování dvou hodnot typu int a jedné hodnoty typu double do řetězce.

```
int type = 3;
int len = 1;
double val = 2.57;
char *buffer = malloc(2 * sizeof(int) + len *
sizeof(double));
memcpy(buffer, &type, sizeof(int));
memcpy(buffer + sizeof(int), &len, sizeof(int));
memcpy(buffer + 2 * sizeof(int), &val, sizeof(double));
```

4.6 Komunikace více hodnot

K odeslání nebo příjmu více hodnot v Matlab-Simulinku stačí vstupnímu nebo výstupnímu portu nastavit určitou šířku. Pro vstupní port se nastavuje funkcí *ssSetInputPortWidth*, výstupní pak obdobně funkcí *ssSetOutputPortWidth*. Obě funkce mají tři parametry. Prvním parametrem je struktura reprezentující blok S-funkce, druhým je index portu a poslední je šířka vybraného portu.

Na straně OpenModelicy jsem se rozhodl pro vytvoření bloků obsahující více vstupních/výstupních portů, kde na každém portu je pouze jedna hodnota, protože jazyk Modelica nepodporuje vrácení pole hodnot z externí funkce. Ve vytvořené knihovně se nachází bloky pro komunikaci až tří hodnot. V případě, že kosimulace vyžaduje

komunikaci více než tří hodnot, jsou v knihovně dostupné šablony pro vytvoření vlastních bloků.

Postup vytvoření bloku v OpenModelice pomocí šablony

K vytvoření nového bloku slouží šablony *transmitN*, *receiveN* a k nim odpovídající funkce *transmitNFcn* a *receiveNFcn.* Tyto funkce volají externí funkce v jazyce C, nacházející se v souborech *transmit.c* a *receive.c*.

Prvním krokem je přidání vstupních nebo výstupních portů do bloků, následně je potřeba upravit ikonu bloku, tak aby porty nebyly na sobě a bylo je možné připojit k dalším blokům. Zobrazit ikonu lze vedle tlačítek na přepnutí mezi grafickým a textovým zobrazením modelu.

Po upravení ikony je potřeba přidat všechny porty do rovnice, kde se volá funkce pro komunikaci. Výstupní porty jsou zapisovány před znak přiřazení a vstupní k argumentům volané funkce.

Ve funkci se musí přidat stejný počet vstupů jako portů ve vytvářeném bloku. Je důležité, aby všechny vstupy a výstupy byly zapsané ve stejném pořadí jako při volání funkce. Následně je třeba přidané vstupy/výstupy doplnit do volání externí funkce a stejně upravit i hlavičku funkce v jazyce C.

Posledním krokem je úprava funkcí v souborech *receive.c* a *transmit.c*. Zde je potřeba inicializovat proměnnou *len* na počet hodnot, které chceme komunikovat. Následně se musí přidat další použití funkce *memcpy* pro zkopírování hodnot do odesílaného řetězce nebo přečtení hodnot z řetězce. Pro použití funkce je v souborech uveden příklad.

4.7 Diskretizace bloků

K nastavení vzorkovací periody se v S-Funkcích v Simulinku využívá funkce *ssSetSampleTime*, která má tři parametry.

```
ssSetSampleTime(S, 0, *stepSizeP);
ssSetOffsetTime(S, 0, 0.0);
ssSetModelReferenceSampleTimeDefaultInheritance(S);
```

Druhá funkce z uvedeného příkladu nastavuje časový offset. Poslední funkce umožní modelu, který obsahuje tuto S-Funkci zdědit vzorkovací periodu z modelu rodiče. Výše zmíněné funkce se používají ve funkci *mdlInitializeSampleTimes.* Před použitím těchto funkcí je nutné definovat pomocí funkce *ssSetNumSampleTimes* počet vzorkovacích period.

Pro vytvoření diskrétního bloku v OpenModelice jsem využil vlastnosti již vytvořeného bloku *DiscreteBlock* v knihovně Modelica. Dědění se provádí klíčovým slovem *extends*, za kterým následuje název bloku nebo modelu, který chceme dědit.

V případně, že se nachází v jiné knihovně je potřeba uvést i cestu k vybranému bloku/modelu.

```
extends Modelica.Blocks.Interfaces.DiscreteBlock;
```

Použití proměnné *sampleTrigger* z bloku *DiscreteBlock* ve vysílacím bloku:

```
when {sampleTrigger,initial()} then
  Err := CommLib.transmit1Fcn(initial(), samplePeriod,
Stop_time, u1, time, Address, String(Port_number));
end when;
```

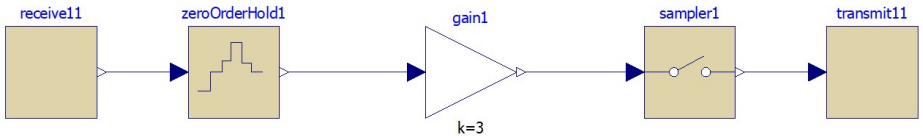
Protože vytvořené bloky jsou diskrétní, je potřeba tyto bloky oddělit pomocí tvarovače a vzorkovače od části modelu se spojitým časem. V Matlab-Simulinku oddělení komunikačního bloku není nutné, o tuto funkci se postará samotná S-Funkce po nastavení periody vzorkování.

5. PŘÍKLADY POUŽITÍ

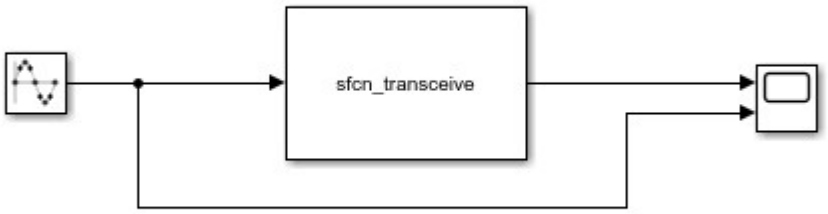
V této kapitole jsou uvedeny příklady použití komunikačních bloků.

5.1 Zesílení signálu

Jako první příklad jsem vybral jednoduchý model, který zesílí vstupní signál. Tento příklad slouží jako ověření funkčnosti komunikace mezi modely v OpenModelice a Matlab-Simulinku.



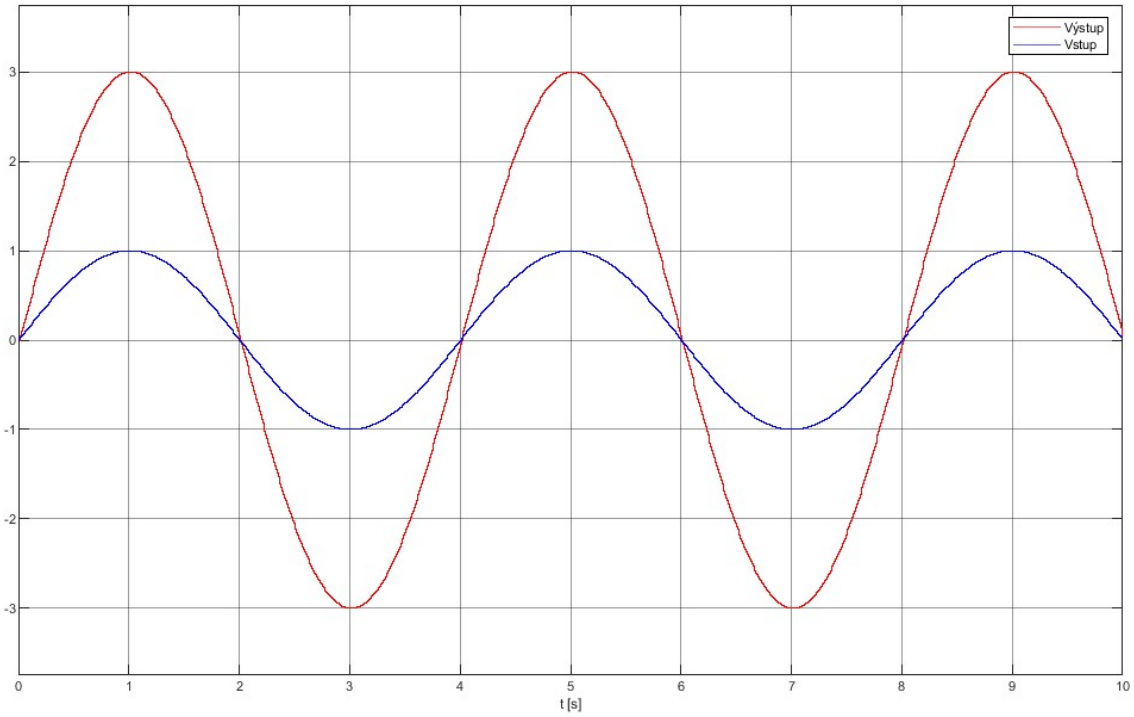
Obrázek 5.1 Schéma zapojení v prostředí OpenModelica



Obrázek 5.2 Schéma zapojení v Matlab-Simulinku pro ověření komunikace

Na obrázku 5.1 je zapojení v programu OpenModelica, protože se jedná o diskrétní bloky, musí se tyto bloky od spojité části modelu řádně oddělit. Na obrázku 5.2 lze vidět, že schéma zapojení v Matlab-Simulinku neobsahuje bloky jako vzorkovač a tvarovač, protože tyto vlastnosti zajistí S-Funkce, po nastavení periody vzorkování.

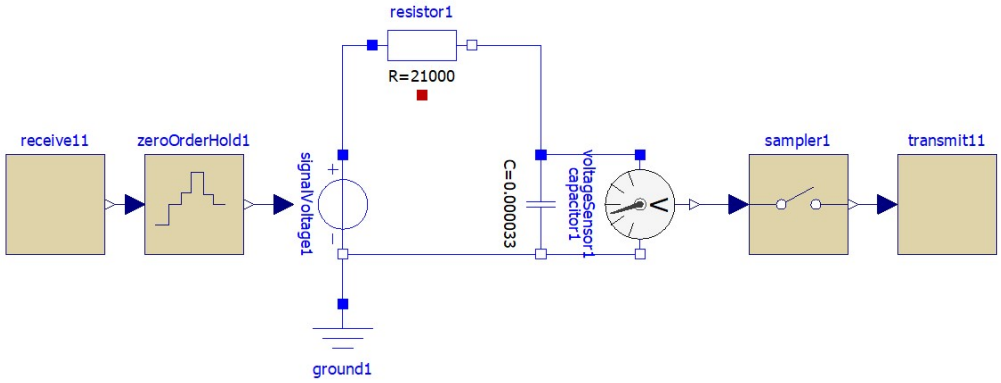
Výsledek kosimulace obou modelu je zobrazen na obrázku 5.3. Tento průběh odpovídá předpokladu, kde výstupní sinusový signál je trojnásobně zesílen oproti signálu vstupnímu.



Obrázek 5.3 Průběh vstupního a výstupního signálu z komunikačního bloku v Matlab-Simulinku

5.2 Regulace napětí RC článku

Druhým příkladem použití je řízení napětí na RC článku pomocí PI regulátoru. Tento příklad je zaměřen na ověření funkčnosti komunikace v uzavřené smyčce.



Obrázek 5.4 Zapojení RC článku s komunikačními bloky v prostředí OpenModelica

Přenos RC článku jsem určil z rovnice:

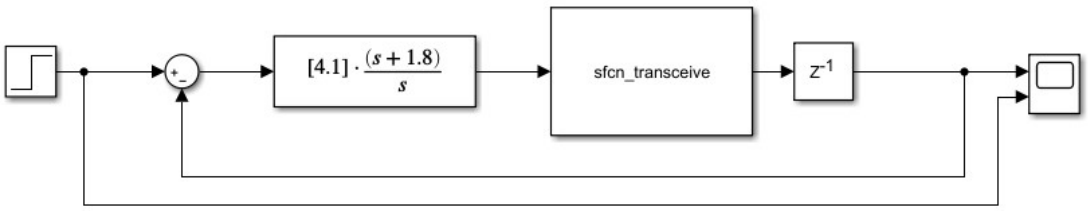
$$F_S(p) = \frac{K}{Tp + 1} \tag{5.1}$$

Pro RC článek platí, že K = 1 a T = R*C. Po dosazení do rovnice 5.1 vyjde přenos soustavy ve tvaru:

$$F_S(p) = \frac{1}{0{,}69p + 1} \tag{5.2}$$

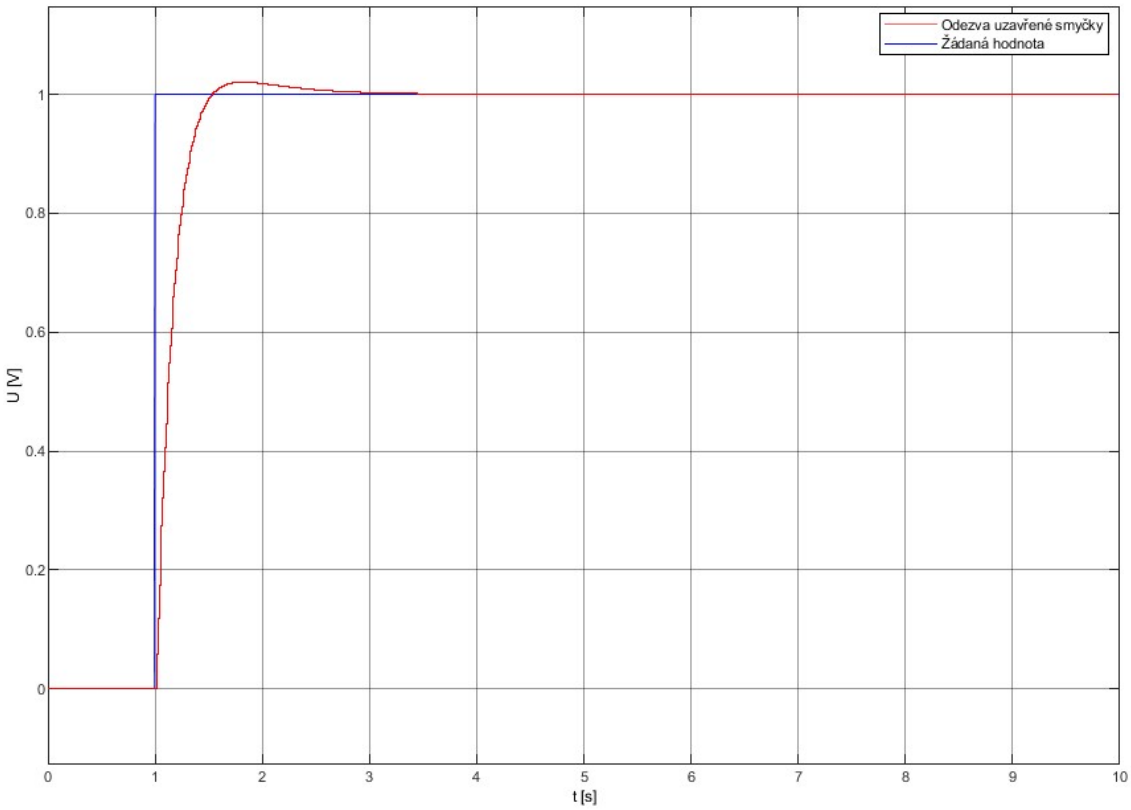
Přenosu RC článku jsem ověřil pomocí přechodové charakteristiky, kterou lze vytvořit nahrazením přijímacího bloku v OpenModelice za jednotkový skok. Pro tuto soustavu byl pomocí nástroje automatického ladění regulátoru v programu Matlab navržen PI regulátor s přenosem:

$$F_R(p) = 4{,}1\frac{(1 + 0{,}55p)}{1} \tag{5.3}$$



Obrázek 5.5 Simulink schéma zapojení PI regulátoru s komunikačním blokem

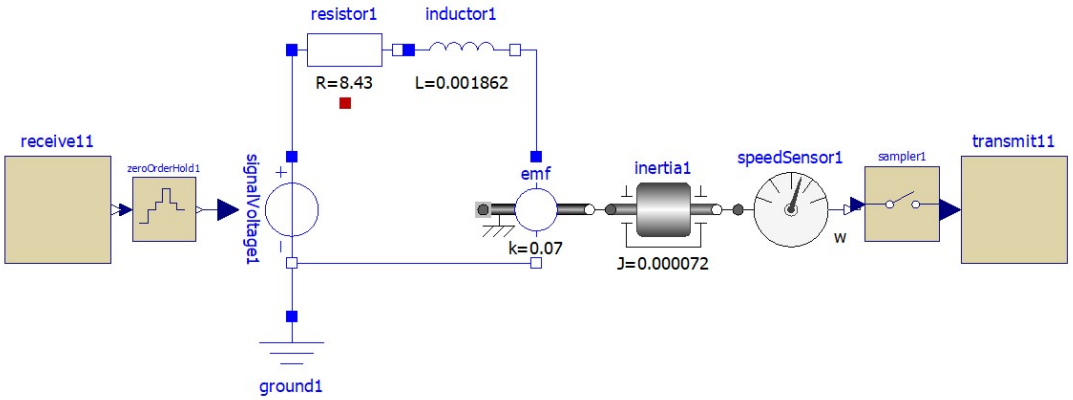
Výsledek kosimulace modelu RC článku a PI regulátoru zobrazen na následujícím obrázku.



Obrázek 5.6 Odezva uzavřené smyčky PI regulátoru a RC článku na skokovou změnu žádané hodnoty

5.3 Regulace otáček stejnosměrného motoru

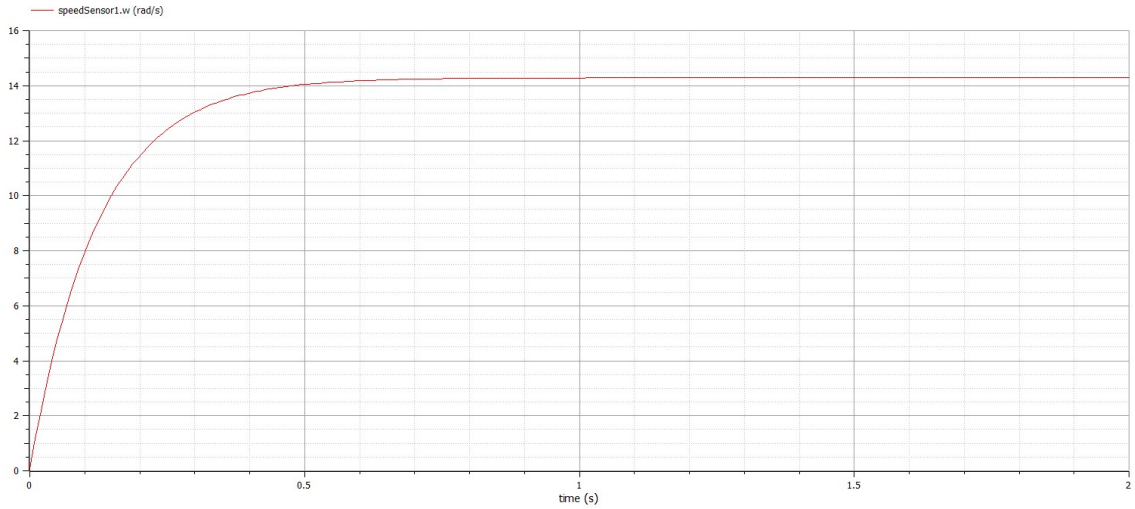
Jako poslední příklad jsem zvolil regulaci otáček stejnosměrného motoru PI regulátorem.



Obrázek 5.7 Model motoru v prostředí OpenModelica

Pro určení přenosu stejnosměrného motoru jsem využil přechodovou charakteristiku. Na vstup komunikačního bloku v Simulinku jsem připojil jednotkový skok a výsledek zobrazil na osciloskopu. Přenos z přechodové charakteristiky na obrázku 5.8 jsem určil ve tvaru

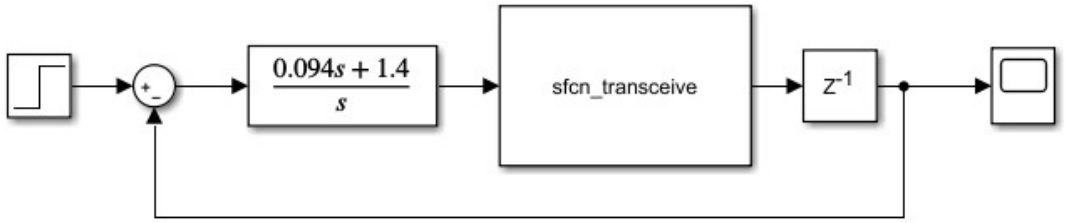
$$F_S(p) = \frac{14{,}28}{0{,}13p + 1} \tag{5.4}$$



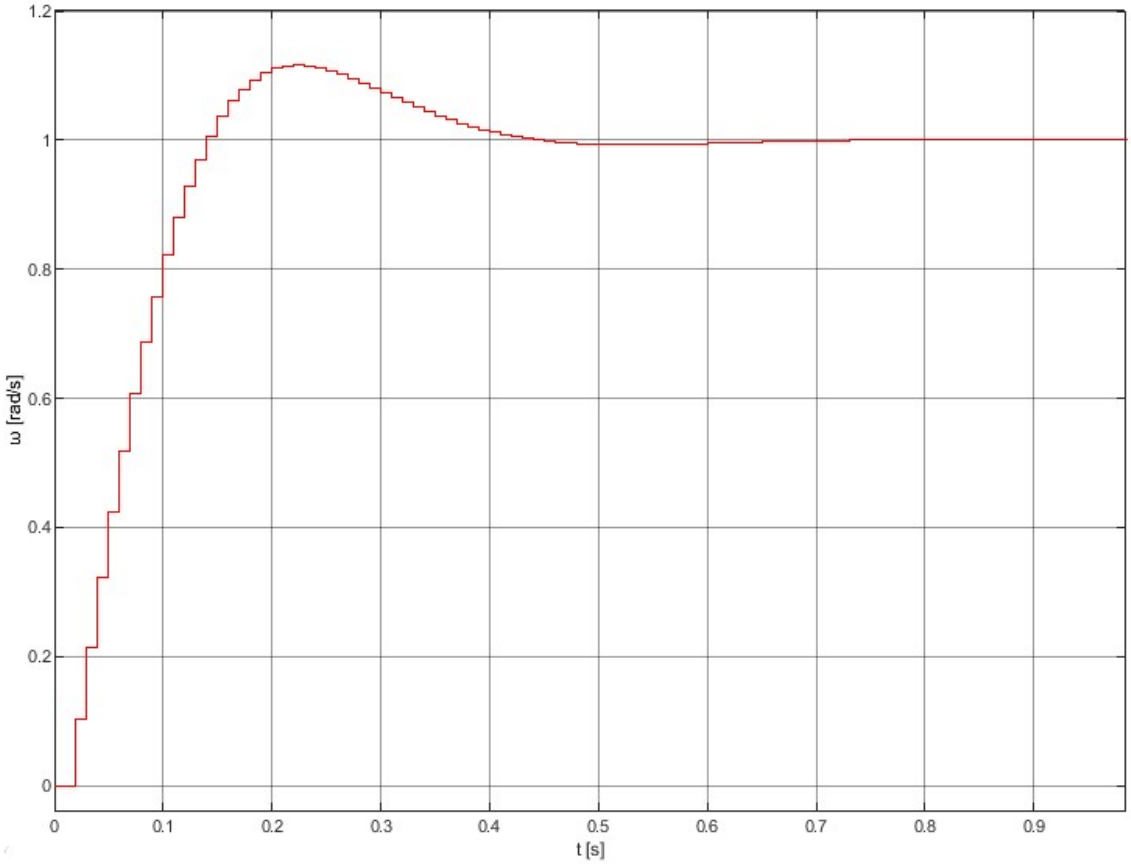
Obrázek 5.8 Přechodová charakteristika motoru

Pro návrh PI regulátoru pro tuto soustavu jsem opět využil nástroj automatického ladění regulátoru v Matlabu. Výsledný přenos PI regulátoru má následují tvar

$$F_R(p) = 4{,}1\frac{(1 + 0{,}55p)}{1} \tag{5.5}$$



Obrázek 5.9 Simulink schéma zapojení regulace otáček stejnosměrného motoru



Obrázek 5.10 Odezva uzavřené smyčky PI regulátoru a stejnosměrného motoru na jednotkový skok žádané hodnoty

6. ZÁVĚR

Tato bakalářská práce se zabývá kosimulaci mezi simulačními SW OpenModelica a Matlab-Simulink.

V první kapitole je popsán modelovací jazyk Modelica, který je využíván programem OpenModelica pro modelování systémů a samotný program OpenModelica a jeho části.

Druhá kapitola se věnuje standardu Functional Mock-up Interface, pomocí kterého je možné exportovat model a importovat do jiného prostředí, které tento standard podporuje.

Třetí kapitola se věnuje návrhu vlastního řešení v podobě komunikačních bloků. Kapitola popisuje způsob komunikace mezi programy OpenModelica a Matlab-Simulink. Komunikace využívá knihovnu ZeroMQ, která je zde taky popsána.

Následující kapitola se věnuje realizaci navrženého řešení a odstranění problémů vzniklých při vytváření bloků. Jedním z problémů bylo, že oba simulátory nejprve vykonávaly přijímací blok, ale v tom případě nikdy data nepřišla a simulace neproběhla. Z tohoto důvodu jsem vytvořil společný blok pro příjem i odesílání dat. Tento blok způsobil při zapojení do uzavřené smyčky algebraickou smyčku, kterou jsem odstranil použitím zpoždění na výstupu komunikačního bloku.

Vytvořené bloky jsem otestoval na třech příkladech, které jsou popsány v páté kapitole. Výsledné průběhy odpovídají očekávaným průběhům. Tyto příklady byly otestovány i při kosimulaci mezi dvěma počítači.

Literatura

[1] TILLER, Michael. *Modelica by Example* [online]. [cit. 2019-12-12]. Dostupné z: https://mbe.modelica.university/

[2] *Modelica Tools* [online]. Modelica Association, ©2019 [cit. 2019-12-12]. Dostupné z: https://www.modelica.org/tools

[3] TILLER, Michael. *Introduction to physical modeling with Modelica*. Boston: Kluwer Academic Publishers, c2001. ISBN 0-7923-7367-7.

[4] MODELICA ASSOCIATION. *Modelica - A Unified Object-Oriented Language for Systems Modeling: Language Specification*, 2017.

[5] *OpenModelica User's Guide* [online]. Open Source Modelica Consortium 2019 [cit. 2019-12-10]. Dostupné z: https://openmodelica.org/doc/OpenModelicaUsersGuide/v1.9.3/index.html

[6] *Advanced Interactive OpenModelica Compiler (OMC)* [online]. OpenModelica, 2019 [cit. 2019-12-10]. Dostupné z: https://openmodelica.org/?id=51:open-modelica-compiler-omc&catid=10:main-category

[7] Modelica Association, *Functional Mock-up Interface for Model Exchange and Co-Simulation* [online]. 2019 [cit. 2019-12-21]. Dostupné z: https://fmi-standard.org/downloads/

[8] Modelica Association Project "FMI". *Functional Mock-up Interface* [online]. 2019 [cit. 2019-12-21]. Dostupné z: https://fmi-standard.org/

[9] ROULEAU, Guy. *Example implementation od Co-Simulation using Simulink* [online]. 2018 Dostupné z: https://www.mathworks.com/matlabcentral/fileexchange/66969-example-implementation-of-co-simulation-using-simulink

[10] *ZeroMQ Documentation* [online]. The ZeroMQ authors, ©2020 [cit. 2020-01-04]. Dostupné z: https://zeromq.com/get-started/

[11] *Include Functional Mockup Unit (FMU) in model* [online]. The MathWorks, ©1994-2020 [cit. 2020-06-06].
Dostupné z:
https://www.mathworks.com/help/simulink/ref_extras/fmu.html

Seznam symbolů, veličin a zkratek

FMI	-	Functional Mock-up Interface
FMU	-	Functional Mock-up Unit
ODE	-	Ordinary differential equations
SW	-	Software
XML	-	Extensible Markup Language

Seznam příloh

Příloha 1 – Obsah CD

Na přiloženém CD se nachází:

- Elektronická verze této práce ve formátu PDF
- Adresář se všemi zdrojovými kódy použitými v této práci
- Adresář Matlab obsahující:
    - projekt v programu Matlab,ve kterém se nachází knihovna s komunikačním blokem, zdrojové kódy použité k vytvoření bloku, modely příkladů
    - knihovny potřebné k fungování bloku
- Adresář Modelica obsahující:
    - knihovnu CommLib s vytvořenými komunikačními bloky
    - modely použitých příkladu
    - adresář resource obsahující soubory a knihovny potřebné pro komunikaci
- Adresář „ZdrojoveKody“ se všemi zdrojovými kódy použitými v této práci