České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCA

Model-Based Design pre programovateľné automaty

Praha, 2011 **Autor: Michal Andrejco**

Prehlásenie

Prehlasujem, že som svoju diplomovú prácu vypracoval samostatne a použil som len podklady (literatúru, projekty, softvér atď.) uvedené v priloženom zozname.

V Prahe dňa _____ _____

podpis

Poďakovanie

V prvom rade by som chcel poďakovať za vstriecnosť pánom z Humusoftu, hlavne p. Byronovi, vďaka ktorému bola prístupná 30 dňová trial verzia PLC Codera, ktorá sa na študentské účely normálne neposkytuje a bez ktorej by táto práca nemohla vzniknúť. Vďaka patrí tiež mojím kamarátom, ktorí pomohli v najhorších chvíľach a svojími pripomienkami ma inšpirovali k novým nápadom. Veľká vďaka patrí aj mojim rodičom, ktorí ma podporovali v každej chvíli a dopomohli až k vzniku tejto práce. Vďaka patrí aj katedre riadiacej techniky ktorá vypísala tému na túto diplomovú prácu a môjmu vedúcemu práce za pomoc pri riešení problémov, ktoré sa pri jej tvorbe vynorili.

Abstrakt

Cieľom diplomovej práce je demonštrovať novodobú návrhovú metódu Model-Based Design v oblasti automatizačnej techniky. Využité budú nástroje PLC-Coder systému Matlab a vývojový nástroj Mosaic pre programovateľné automaty Tecomat. V práci bude ukázaný spôsob vytvorenia modelu v grafickom prostredí Simulink a jeho následné prenesenie na platformu programovateľného automatu. Ústredným bodom je vytvorenie sady ukážkových modelov a ich prenesenie do Mosaicu. Nakoniec bude predstavená možnosť praktického využitia modelu pre diagnostiku v automatizačnom systéme.

Abstract

The aim of this thesis is to demonstrate a new progressive developing method Model-Based Design on the field of automation techniqe. It will be used Matlab tools PLC-Coder and developing tool Mosaic for Tecomat programable logic controlers. Thesis will show the kind of implementation of the system model in a grafical interface of Simulink. After that the model will be implemented on the programable logic controler platform. The main idea is creating a set of simple demonstrative models by using the tool PLC-Coder of Matlab system and portage of the generated code to the Mosaic, Tecomat PLC's software developing tool. Finaly will be introduced kind of usefullnes the model for the diagnostic task in an automation system.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Michal Andrejco**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu: **Model-Based Design pro programovatelné automaty**

Pokyny pro vypracování:

1. Seznamte se s vlastnostmi poslední verze systému Matlab/Simulink 2010 a především s možností přeložit simulační schéma do jazyka ST pro programovatelné automaty, prostudujte normu IEC/EN 61131-3 pro programovatelné automaty (PLC), syntaxi jazyka ST (Structured Text) a zhodnoťte jejich možnou implementací ve vývojovém systému Mosaic.
2. Prostudujte a zhodnoťte možnosti komunikace mezi systémem Matlab/Simulink a vývojovým systémem Mosaic pro PLC a přenosu vygenerovaného programu v ST.
3. Vytvořte soubor jednoduchých a různorodých příkladů schémat v Simulinku a prověřte kompatibilitu vygenerovaného programu ST s překladačem ST v systému Mosaic, případně upozorněte možné případy nesouladu, analyzujte jejich podstatu, případně navrhněte programová opatření k nápravě.
4. Navrhněte typický příklad využití Model-Based Design v praxi.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Martin Hlinovský, Ph.D.

Platnost zadání: do konce letního semestru 2011/2012

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 19. 1. 2011

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Výpočtové a simulačné matematické programové nástroje boli v minulosti využívané zväčša v sfére výskumu a na akademickej pôde. Postupne tieto programy prenikajú do bežného praktického života a ako príklad stojí za to uviesť priemyselné a ekonomické odvetvia. Obzvlášť tieto oblasti sú náročné na flexibilnú optimalizáciu výroby resp. stratégie, ktoré sú ovplyvňované nesmiernym množstvom externých faktorov. Prípadné nepresnosti a chyby vo výrobných algoritmoch alebo strategických postupoch môžu mať za následok obrovské investičné straty, čo v môže mať v konečnom dôsledku dopad na lokálnu podnikovú ekonomiku a zároveň prispieva ku kolapsu ekonomiky globálnej. Takémuto scenáru sa samozrejme snaží vyhnúť každá rozumná firma a do procesu vývoja a ekonomického manažmentu zapája aj matematické modely chovania sa systému vo všeobecnosti. Pomocou nich je umožnená predikcia správania sa systému v závislosti na externých podnetoch a zároveň je vytvorená možnosť flexibilnejšie reagovať na aktuálne zmeny systému, čím sa zvyšuje celková efektivita podniku. Tento spôsob návrhu je nazývaný Model-Based Design (ďalej len MBD).

Táto progresívna metóda sa dostáva už aj na pole priemyselnej automatizácie, kde riadenie výrobného procesu je zabezpečované pomocou riadiacich automatov PLC. Využitím výpočetného nástroja Matlab® a jeho súčasti slúžiacej na vytváranie matematických modelov systémov v grafickej podobe Simulink® je možné zefektívniť vývoj riadiacich algoritmov a predchádzať tak zbytočným časovým a ekonomickým nákladom na vývoj experimentálnou stratégiou pokus-omyl.

Zavedením Matlabu do procesu vývoja riadiaceho algoritmu PLC bola vytvorená vyššia virtuálna vrstva, ktorá dovoľuje vyskúšať návrh priamo na matematickom popise sústavy. To šetrí jednak prostriedky časové (je rýchlejšie sadnúť si za počítač a odsimulovať nejaké správanie sa algoritmu na PC, než je jazdenie do firmy objednávateľa a

skúšanie na fyzickom hardvére, nehovoriac o nevrelých pohľadoch domáceho personálu) a taktiež náklady v prípade zásadnej chybe v algoritme, ktorá by mohla spôsobiť katastrofu. Okrem toho je lepšie sa oprieť o nejaký kokrétny výsledok, ktorý vyprodukuje matematický model, než navrhnúť algoritmus podľa papierových špecifikácií a veriť, že pri rôznych externých faktoroch nenastane porucha. Výhoda je aj v tom, že táto návrhová metóda oddeľuje etapu špecifikácie od iplementačnej etapy návrhu systému. To znamená, že developer nie je nútený sústrediť sa pri kódovaní aj na funkcionalitu algoritmu, ale špecifikuje požiadavky funkčnosti algoritmu na vyššej vrstve a samotné prekódovanie do programovacieho jazyka je automatické. To prináša výhody hlavne pri zmene zadania počas vývoja algoritmu pre daný projekt. Zmena sa deje väčšinou len na jednotlivých moduloch, ktoré sa dajú zakomponovať do grafického návrhu pohodlnejšie než je to v prípade kódovania. V prípade bez modelu by to znamenalo prepísať a doplniť stovky riadkov kódu, čo zaberie viac času než je intuitívne skladanie grafických blokov, nehovoriac o vnášaní chýb do vytvoreného kódu. Okrem toho sa stáva, že sa špecifikácia mení s odstupom času a k vyprodukovanému programu je nútený postaviť sa iný človek než je pôvodný autor. V takom prípade je grafický návrh intuitívnejší z hľadiska nového užívateľa, než je lúštenie tisícov riadkov sekvenčného kódu, čo v prípade programov pre PLC nie je až tak ojedinelé.

Hlavným bodom tejto práce je vytvoriť sadu demonštratívnych jednoduchých príkladov ako s touto návrhovou metódou pracovať. Pre dosiahnutie cieľa bude práca štrukturovaná na celky, ktoré sa budú snažiť postupne rozobrať zahrnutú podproblematiku. Hneď na začiatok bude potrebné definovať pojmy s ktorými sa bude pracovať, aby bolo jasné na čo sa myslí a nedošlo tak k možným nedorozumeniam. Následne bude priblížená norma IEC/EN 61131-3 zaoberajúcu sa zjednotením programovacích jazykov pre riadiace automaty. Po výklade normy budú zhodnotené komunikačné možnosti medzi programovými nástrojmi Matlab® a Mosaic®. Komunikačné možnosti potom budú vyskúšané pomocou jednoduchých príkladov. Ako zhrnutie bude nakoniec navrhnutá komplexnejšia aplikácia ktorá bude názorne ukazovať užitočnosť návrhovej metódy v praxi.

Práca má poslúžiť ako inšpirácia a priblíženie možností nových návrhových metód, ktoré v automatizačnej praxi ešte nie sú až tak bežné. Obsahovo by mala byť zrozumiteľná aj pre osoby mimo akademickú pôdu a mala by byť podkladom pre zoznámenie s problematikou nasadzovania modelov sústav do praxe. Okrem toho by mohla byť inšpiráciou pre rozvoj nových doposiaľ nenasadených metód diagnostiky.

1.1 Softvérové prostriedky

Podkapitola je určená hlavne na bližšie predstavenie softvéru, ktorý bude v práci využívaný. Jej cieľom nie je propagácia produktov tretích strán. Má poslúžiť hlavne čitateľom, ktorí sa s týmto typom softvéru nedostávajú dennodenne do styku. Práve tomuto typu záujemcov o túto prácu sú určené nasledujúce riadky, ktoré by mali jednoducho a jasne predstaviť spomínané prostriedky a možno aj motivovať k ich nasadeniu do hľadaného riešenia ich problému. Ak je čitateľ s prostriedkami zoznámený, bude pre neho táto kapitola určite bezpredmetná.

1.1.1 Matlab

Matlab® je, slovami výrobcu Mathworks, vysokoúrovňový programovací jazyk s interaktívnym prostredím, ktorý umožňuje riešiť výpočetne náročné úlohy rýchlejšie a efektívnejšie, ako to bolo v prípade jazykov C, C++ alebo Fortran. Programovací jazyk je optimalizovaný hlavne na maticové počty, takže riešenie zložitých algoritmov s maticovými parametrami je efektívnejšie, ako by to bolo v prípade vyššie spomínaných jazykoch. Túto stručnú definíciu si možno vyložiť jednoduchšími slovami. Ide o programovací jazyk s vlastne definovanou syntaxou. Zo spomínaných jazykov je tá podobná najmä Fortranu a je optimalizovaná hlavne na zápis vektorových premenných. Už to napovedá, že Matlab® bude nasadzovaný hlavne pre zložitejšie výpočetné operácie, kde je nutné uvažovať ako parametre vektorové premenné. V samej podstate ide o veľmi chytrú maticovú kalkulačku, ktorá s využitím rôznych naimplementovaných knižníc a toolboxov dokáže veľmi efektívne spracovať akúkoľvek matematicko-výpočtovú úlohu. Od ostatných programovacích jazykov sa matlab líši tým, že poskytuje interaktivitu pomocou príkazového riadku, ktorý okamžite vráti výsledok zapísaného príkazu. To znamená, že pri vývoji algoritmu pre výpočet sa dá postupovať dvoma spôsobami. Prvým je postupné zadávanie príkazov do príkazového riadku s potvrdením každého príkazu, s tým že užívateľ vidí hneď medzivýsledok a môže podľa toho spätne korigovať výpočetný algoritmus. Výpis z takého to postupu, ktorý tiež možno nazvať ako „Step By Step“ , je uvedený v príklade 1.1. Druhý spôsob je klasický dávkový postup písania programu ako funkciu do editoru a následne jeho skompilovanie[1] a spustenie. Súbor s takto definovanými príkazmi je možné uložiť ako takzvaný m-file. Na ten je možné sa potom odkazovať v ďalších m-filoch, alebo ho volať priamo z príkazového riadku ako funkciu

[1] Kompilácia sa robí v prostredí Matlab® automaticky.

alebo ako sekvenciu príkazov, podľa toho ako bol nadefinovaný. Výpis typického m-file ako funkcie je uvedený v príklade 1.2. Grafické prostredie akým Matlab disponuje je znázornené na obrázku 1.1. Okrem základných funkcií ktoré boli popísané je softvér vybavený rôznorodými nadstavbami, ktoré mu okrem iného dovoľujú spracovávanie signálu, simuláciu modelov, prezentáciu výsledkov vo formách grafov (2D,3D) a pod.

Základné priblíženie tohoto nástroja je demonštrované na jednoduchých príkladoch a pre prehľad neznalého čitateľa by malo byť dostačujúce. Presný popis možností softvéru by bol ďaleko nad rozsah tejto práce. K tomuto účelu slúži oficiálna dokumentácia k nástroju (MathWorks, 2011*b*), ktorá je dostupná z webových stránok firmy Mathworks.

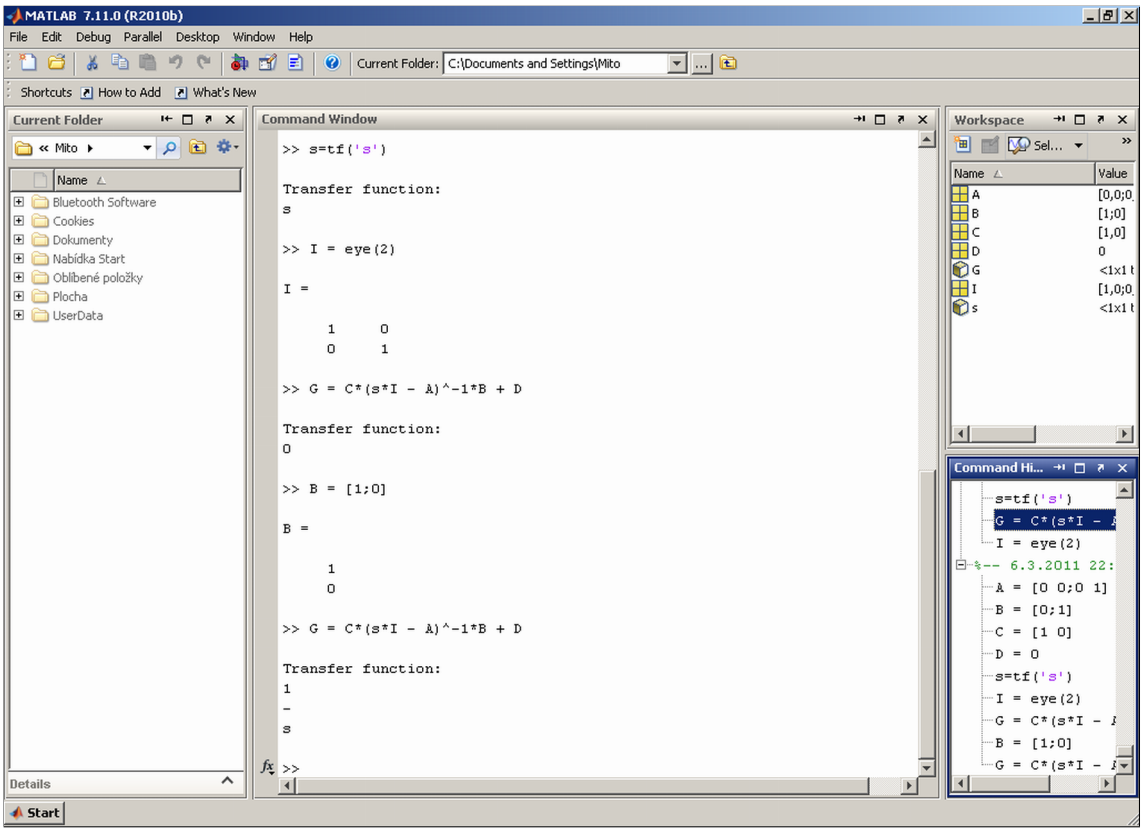
Príklad 1.1 („Step By Step“): Ukážka interaktívneho zadávania príkazov do príkazového riadku Matlabu. V tomto prípade spočítanie prenosu systému integrátora.

```
----------------------- príkaz na definíciu premenných --------------------
>> A = [0 0;0 1], B = [1;0], C = [1 0], D = 0, I = eye(2), s = tf('s')

----------------------- odpoved' programu ----------------------------------
A =  0     0
     0     1

B =  1
     0

C =  1     0

D =  0

I =  1     0
     0     1

Transfer function:
s

------------- príkaz na spočítanie prenosu systému ------------------------
>> G = C*(s*I - A)^-1*B + D

----------------------- odpoved' programu ----------------------------------
Transfer function:
1
-
s
---------------------------------------------------------------------------
```

Príklad 1.2 (Výpis m-filu): Výpis jednoduchej funkcie vytvorenej ako m-file. Funkcia spočíta zo zadaných parametrov systému jeho prenos.

Poznámka: Ide o zjednodušenú demonštračnú funkciu, ktorá nevykonáva kontrolu dimenzií vstupných parametrov.

```
------------------- Funkcia uložená v m-file ------------------
function G = getTf(A,B,C,D)

n = length(A);
I = eye(n);
s = tf('s');
G = C*(s*I - A)^-1*B + D;
---------------------------KONIEC-----------------------------

---------------- Volanie funkcie z Matlabu --------------------
>> G = getTf(A,B,C,D)

Transfer function:                //odpoved' programu
1
-
s
----------------------------------------------------------------
```



Obr. 1.1: Hlavné okno výpočtového nástroja Matlab

1.1.2 Simulink

Simulink® je nadstavbovou súčasťou systému Matlab®. Ide o multidoménový simulačný nástroj s grafickým prostredím pre tvorbu matematických modelov fyzikálnych systémov. Pre modelovanie využíva výrobcom implementované knižnice, ktoré obsahujú základné stavebné bloky umožňujúce grafický prepis diferenciálnych rovníc, ktorými je systém popísaný. Simulink® dovoľuje navrhovať, simulovať, implementovať a testovať rôznorodé dynamické systémy. Naproti čistému matematickému modelovaniu ktoré dovoľuje Matlab sám o sebe formou diferenciálnych rovníc, resp. stavovoým popisom fyzikálneho systému, v Simulinku je modelovanie viacej intuitívne. Užívateľovi je prístupná sada grafických blokov s rôznorodými funkciami, ktorými sa vhodným vzájomným prepojením dá namodelovať ľubovoľný systém. V samotnej podstate Simulink využíva algoritmov ktoré sú implementované v Matlabe, ktorý je pre neho výpočetným jadrom, a užívateľovi sprístupňuje tieto funkcie na vyššej vrstve. Tým že sa zaviedlo grafické prostredie a funkcie zložité na použitie sa sprístupnili formou čiernej skrinky *Black Box* (BB). Znížila sa tým hranica abstrakcie oproti modelovaniu len čisto matematickým popisom a v textovej podobe. Modelovanie sa stalo prehľadnejším. Okrem základných funkcií ktoré Simulink implementuje, je vybavený aj sadou rôznorodých už preddefinovaných systémových nelineárnych blokov, matematických funkcií, funkcií na spracovanie signálu, obrazu atď. Výhodou je, že prostredie dovoľuje namodelovaný systém pohodlne odsimulovať na zadanom časovom horizonte. Ako výstup simulácie sú prístupné výstupné dáta sledovaných veličín a to vo forme grafu alebo výstupného vektoru, ku ktorému prislúcha časový vektor. Časový vektor slúži na to, aby bol presne jasný vývoj sledovanej funkcie v každom časovom okamžiku simulácie na nastavenom časovom horizonte. Tieto dáta sú potom prístupné v Matlabe, alebo sú ukladané do súboru, ktorý je znova využiteľný pre neskoršiu reprodukciu výsledkov. Práve možnosť simulácie robí zo Simulinku veľmi mocný nástroj a taktiež je to zázemie pre Model-Based Design. Ako jednoduchý príklad modelovaného systému môže poslúžiť model DC elektromotora s permanentným budením. Matematický popis a simulácia je uvedená v príklade 1.3.

Príklad 1.3 (Simulácia sústavy 2. rádu v Simulinku): Predmetom záujmu je DC elektromotor s permanentným budením. Príklad s modelom motora bol prebratý z (Hoefling, T. a Isermann, R., 1996). Jedná sa o motor s príkonom 550W. Motor sa ako celok dá rozložiť do dvoch subsystémov. **Elektrický subsystém** pozostávajúci z riadiaceho (napájacieho) napätia U_a, ktoré má za následok prúd I_a pretekajúci indukčnosťou cievky rotora L_a s činným odporom R_a. Magnetický tok statora je značený ψ. Pretože motor

rotuje, na indukčnosti rotora sa indukuje napätie pôsobiace proti napájaciemu napätiu, ktoré je priamo úmerné otáčkam rotora ω. Matematický popis elektrického subsystému je zapísaný rovnicou 1.1.

Druhý podsystém je **mechanický subsystém** pozostávajúci z momentu zotrvačnosti J rotora, elektromechanického momentu M_e, záťažového momentu M_l a trecieho momentu M_f vznikajúceho na ložiskách motora. Popis výstupných otáčok motora ω je vyjadrený rovnicou 1.2.

Zo systémového hľadiska sú ako stavy systému uvažované prúd statorom I_a a výstupné otáčky motora ω.Ako vstupy sú uvažované vstupné napätie U_a a záťažný moment M_l.

$$L_a \dot{I}_a(t) = -R_a I_a(t) - \psi\omega(t) - K_b|\omega(t)|I_a(t) + U_a \tag{1.1}$$

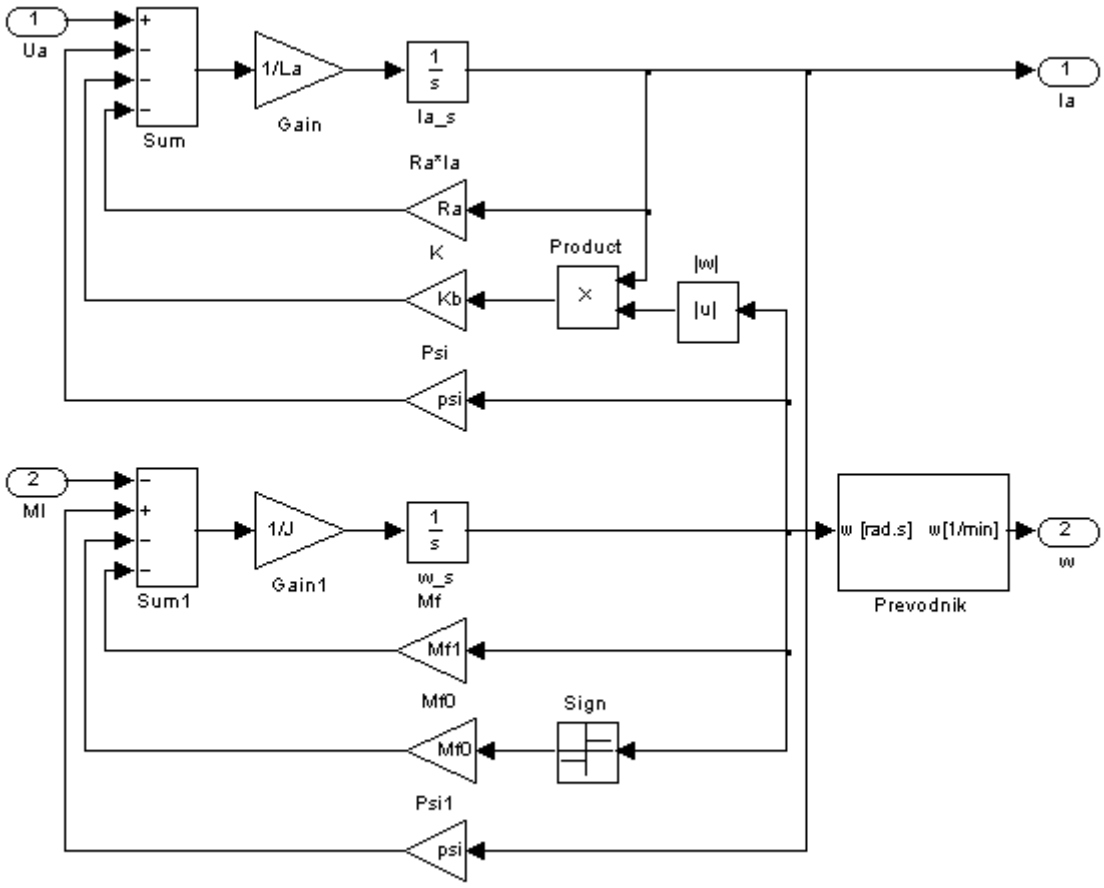
$$J\dot{\omega}(t) = \psi I_a(t) - (M_{f1}\omega(t) + M_{f0}\ sign(\omega(t))) - M_l \tag{1.2}$$

Druhý člen v rovnici 1.1 s konštantou K_b je nelineárny člen, ktorý popisuje úbytok napätia na kefách motora vznikajúceho pri napájaní pomocou PWM. Druhá nelinearita je v rovnici 1.2 vyjadrená v druhom člene s konštantou M_{f0} ktorá značí suché trenie a konštantou M_{f1} označujúcou viskózne trenie. Nelinearity boli zámerne zahrnuté do modelu kôli presnejšej aproximácii správania sa modela. Tento prístup sa nazýva „Grey-Box modeling“ a je popísaný v (Bohlin, T., 1994). Parametre elektromotora boli zisťované meraním na fyzickom modele a aproximovaním pomocou metódy najmenších štvorcov. Identifikované hodnoty parametrov sú uvedené v tabuľke 1.1.

veličina	**značka**	**hodnota**	**jednotka**
odpor vinutia kotvy	R_a	1,52	Ω
indukčnosť kotvy	L_a	$6,82 \cdot 10^{-3}$	Ωs
magentický tok	ψ	$0,33$	Vs
faktor úbytku napätia	K_b	$2,21 \cdot 10^{-3}$	Vs/A
viskózne trenie	M_{f1}	$0,36 \cdot 10^{-3}$	Nms
suché trenie	M_{f0}	$0,11$	Nm
zotrvačná konštanta	J	$1,92 \cdot 10^{-3}$	kgm^2

Tabuľka 1.1: Tabuľka identifikovaných parametrov DC motora

Schématické znázornenie už namodelovaného systému v simulinku je na obrázku 1.2.



Obr. 1.2: Simulačná schéma modelu DC motoru vytvoreného v Simulinku

Namodelovaný systém je možno simulovať pre rôznorodé pracovné podmienky. Čo je podstatné, k takto namodelovanému systému je možné rôznymi spôsobami navrhnúť regulátor. Správanie sa systému s regulátorom je možné taktiež simulovať a sledovať zmenu parametrov oproti systému bez regulátora. To dovoľuje regulátor dobre odladiť a potom ho použiť na reálnej sústave. Výhodou je že takto odladený regulátor je možné zo Simulinku prekompilovať do iného programovacieho jazyka a implementovať ho na inú platformu než je PC a Simulink. To je cieľom tejto práce a tomuto postupu sa bude zvyšok práce venovať.

Okrem vyššie popísaných možností Simulink® ponúka ešte nesmierne množstvo iných aplikačných možností ktoré sú mimo rozsah tejto práce. Podrobnému popisu týchto možností je venovaná dokumentácia voľne prístupná na stránkach výrobcu (MathWorks, 2011*a*).

1.1.3 PLC Coder

PLC Coder je nadstavba pre simulačný nástroj Simulink. Ide o nástroj, ktorý dokáže z navrhnutého simulinkového modelu generovať hardvérovo nezávislý kód pre programovateľné automaty podľa normy IEC/EN 61131-3. Výsledný kód je možné generovať priamo ako projekt na platformy od rozličných výrobcov alebo ako samostatný súbor kódovaný jazykom .ST podľa vyššie uvedenej normy. To umožňuje preniesť už odskúšaný a odsimulovaný návrh na inú hardvérovú platformu, v tomto prípade platforma PLC resp. PAC, a spustiť ju na nej. Vývoj zložitých regulačných algoritmov už nepredstavuje tak komplexný problém ako v prípade kódovania priamo na platformu. Okrem toho sa naskytuje možnosť okamžitej simulácie riadenej sústavy v podobe modelu, čím sa značne eliminuje riziko poškodenia reálnej sústavy pri nesprávnom návrhu. Čo je ale hlavnou výhodou, týmto nástrojom sa vytvorilo pre programátora rozhranie pre návrh algoritmu do PLC v grafickej podobe, ktorá je omnoho intuitívnejšia čo sa v konečnom dôsledku môže prejaviť na efektivite návrhu. Vďaka tejto možnosti nie je potrebné investovať zvýšenú energiu na kódovanie pri tvorbe algoritmu a tá sa môže investovať do kvality návrhu. Algoritmus je nakoniec vytvorený intuitívne v Simulinku a kódovanie je zautomatizované pomocou tohoto nástroja.

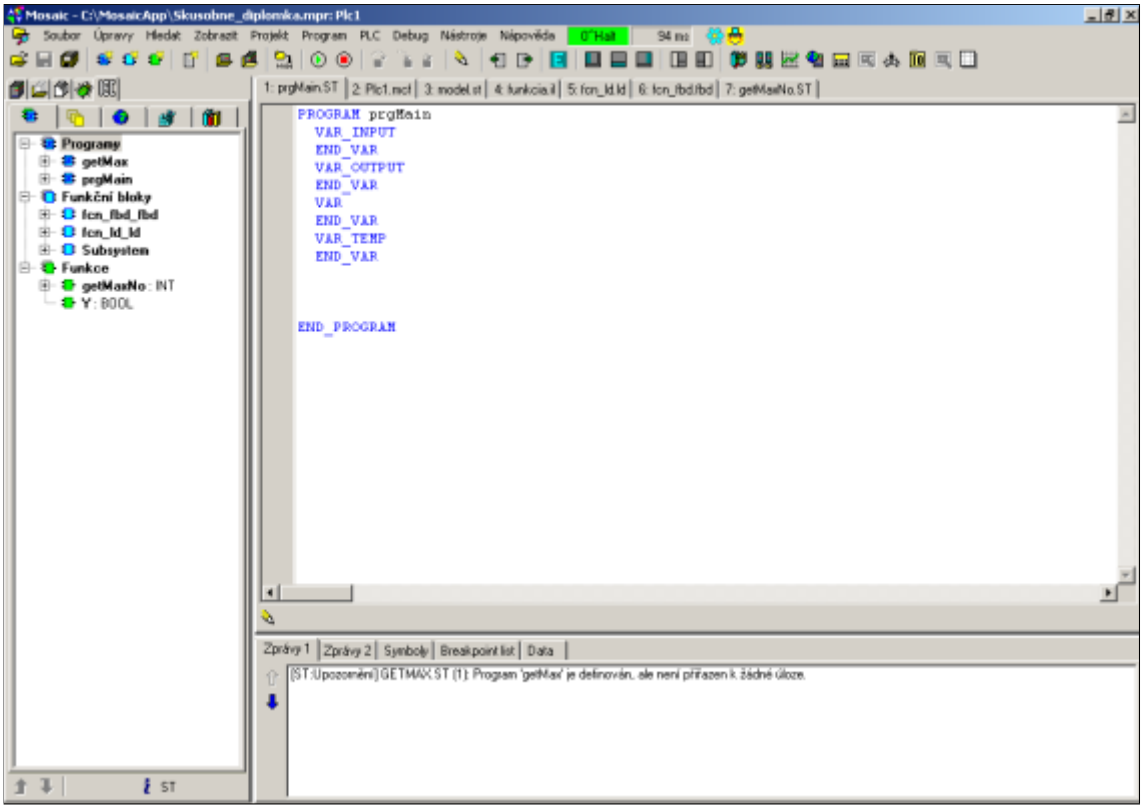
1.1.4 Mosaic

Mosaic®je vývojový nástroj vyvinutý spoločnosťou Teco a.s. pre programovanie riadiacich automatov príznačnej firmy. Mosaic pozostáva zo sady 5 nástrojov:

- Správca projektu
- PID maker
- WEB maker
- GRAPH maker
- PLCnet manažér

Pomocou tejto sady je možné v prostredí vytvoriť kompletný riadiaci algoritmus vrátane regulátorov. Podporuje tvorbu webového rozhrania pre webový server integrovaný vo väčšine riadiacich automatov od Teca. Okrem toho dovoľuje ad-hoc zber dát z riadiaceho procesu a následného zobrazenia v grafe a správu PLC na sieti. Presné

možnosti prostredia sú popísané v dokumente (Teco a.s., 2010) ktorý je dostupný na stránkach výrobcu. Pre predstavu je na obrázku 1.3 zobrazené okno vývojového nástroja.



Obr. 1.3: Hlavné okno vývojového nástroja Mosaic

1.2 Stručný výklad normy IEC/EN 61131-3

Hneď na začiatok je treba upozorniť že táto sekcia má poskytnúť rýchly úvod do problematiky a že v podkapitole budú rozobraté len základné špecifikácie, nachádzajúce sa v norme IEC/EN 61131-3. V žiadnom prípade nebude norma rozoberaná v celom rozsahu. Pre podrobné zoznámenie je nutné naštudovať IEC/EN 61131-3.

Norma IEC/EN 61131 je medzinárodne uznávanou normou pre riadiace automaty PLC. Má 8 častí (Wikipedia, 2011*b*):

- 1.časť: Všeobecné informácie.
- 2.časť: Požiadavky na zariadenia a skúšky.

- 3.časť: **Programovacie jazyky.**
- 4.časť: Užívateľské smernice.
- 5.časť: Komunikácia výmenou správ.
- 6.časť: Komunikácia cez fieldbus.
- 7.časť: Programovanie fuzzy riadenia.
- 8.časť: Smernice pre aplikácie a implementáciu programových jazykov.

Z vymenovaných častí normy je pre túto prácu dôležitá tretia časť IEC/EN 61131-3 - programovacie jazyky. Práve táto časť je predpisom pre štandardizáciu programovacích jazykov, ktoré sú pri programovaní riadiacich automatou používané. Problém bol že každý výrobca mal pre programovanie svojho automatu vlastný programovací štandard s vlastnou syntaxou, takže vznikala nekompatibilita pri prenášaní programov medzi automatami rôznych značiek. Norma je snahou o odstránenie tohoto problému a o zavedenie štandardu na pole automatizačnej techniky. Norma združuje konkrétne 5 štandardov pre programovacie jazyky. Pre rýchlejší prehľad je možné ju rozdeliť na dve časti. Prvá, definujúca spoločné prvky programovacích jazykov, a druhá, definujúca už konkrétne programovacie jazyky.

1.2.1 Spoločné prvky

Nezávisle na zápise programu bolo na začiatku nutné definovať spoločné prvky ktoré budú v programovacích jazykoch používané. To zahŕňa unifikáciu dátových typov, definíciu premenných, definíciu programových jednotiek a konfiguráciu nad programami.

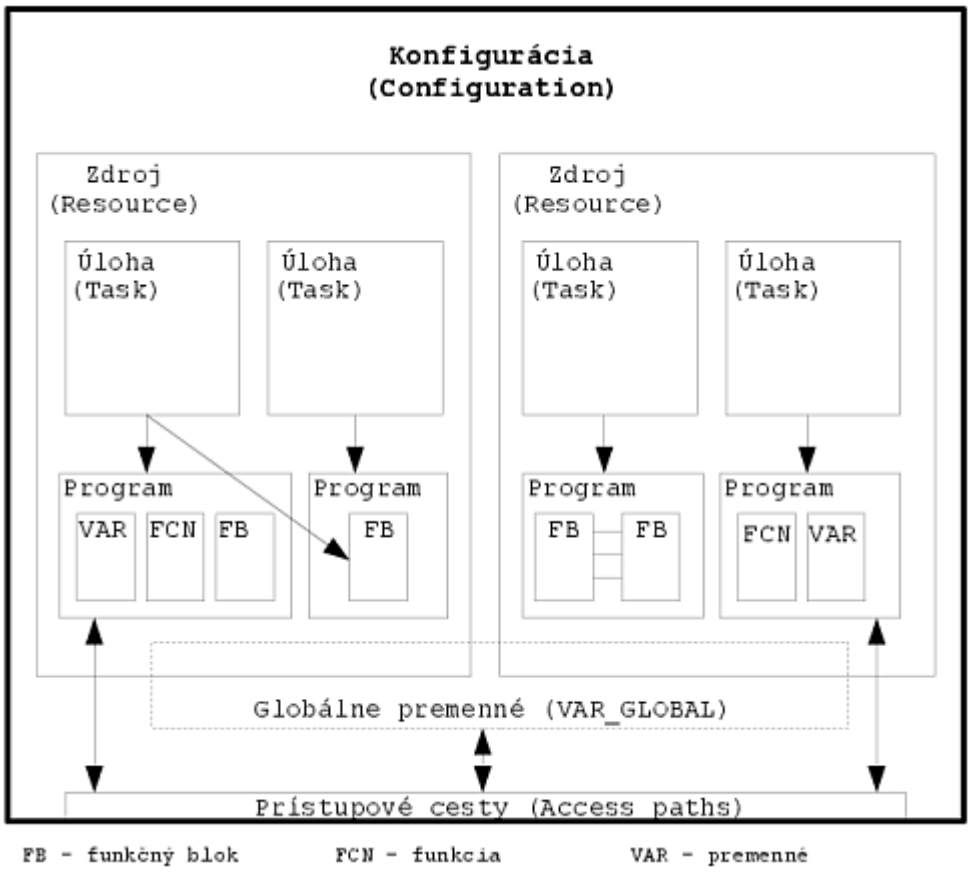
Dátové typy: Norma definuje základné dátové typy a ich rozšírenia podľa počtu bitov rezervovaných v pamäti. Kompletný prehľad s rodovým rozdelením je uvedený v (Teco a.s., 2007*a*) na strane 21-22.

Premenné: Norma popisuje spôsob akým je možné premenné deklarovať a definuje platnosť premennej podľa deklarácie. Pú to globálne premenné (`VAR_GLOBAL, VAR_EXTERNAL`), lokálne premenné (`VAR, VAR_TEMP`), vstupné premenné (`VAR_INPUT`), výstupné premenné (`VAR_OUTPUT`), vstupno-výstupné premenné (`VAR_IN_OUT`). Každá z týchto typov premenných môže mať definované ešte prídavné kvalifikátory, ktoré hovoria o tom či je

premenná zálohovaná (RETAIN), či je to konštanta (CONSTANT), nábežná hrana (R_EDGE) alebo spádová hrana (F_EDGE).

Programové organizačné jednotky: Môžeme ju definovať ako základnú stavebnú jednotku pri tvorbe algoritmu a pomocou nej je aj kompletne určený. Každý program (PROGRAM), funkcia (FUNCTION) a funkčný blok (FUNCTION_BLOCK) je programovou organizačnou jednotkou (Program Organisation Unit). Čo je dôležité podotknúť, že POU nie sú rekurzívne, to znamená že vo svojom tele nemôžu volať samé seba.

Konfigurácia: Je základný predpis pre beh programu na riadiacom automate. Priradzuje sa v nich beh jednotlivých programov na jednotlivé zdroje, tak že je možné rozdeliť výpočetnú úlohu medzi dve CPU ak sú k dispozícii. Konfigurácia (CONFIGURATION) v sebe definuje zdroj (RESOURCE), v podstate CPU, na ktorom bude vykonávaná požadovaná úloha (TASK), resp. proces v rámci ktorého bude príslušná POU vykonávaná.



Obr. 1.4: Organizácia softvérového modelu podľa IEC/EN 61131-3

1.2.2 Programovacie jazyky

Norma IEC/EN 61131-3 definuje dva grafické a dva textové programovacie jazyky.

- textové jazyky: `ST (Structured Text)`, `IL (Instruction List)`
- grafické jazyky: `LD (Ladder Diagram)`, `FBD (Function Block Diagram)`
- `Sequential Function Chart (SFC)`

Okrem nich je nadefinovaný kvázi piaty grafický jazyk[2] pre lepšie štruktúrovanie hlavného programu. Definície zahŕňajú podrobný popis s vysvetlením pre každý jazyk. V odstavci budú v skratke vysvetlené princípy programovania v každom z definovaných štandardov s ukážkami, hlavne pre jazyk ST, ktorý je predmetom tejto práce.

1.2.3 Sequential Function Chart (SFC)

Programovací jazyk definuje preostriedky pre lepšiu organizáciu a štrukturovanie programu a programových jednotiek, napísaných v hociktorom vo vyššie vymenovaných jazykoch definovaných normou. Štrukturuje ich do množiny *krokov (steps)* a *prechodov (transitions)* prepojených *orientovanými hranami (directed links)*. Každý krok je spojený s vykonávanou množinou *akcií (actions)* a pre každý prechod existuje *prechodová podmienka (transition condition)*. Dôležité je, že ak je už nejaká časť programu štrukturovaná do elementov SFC, tak zvyšný program musí byť štrukturovaný podľa SFC tiež. Pre lepšie predstavenie SFC sa využije príklad 1.4, kde sa priblíži logika programovacieho jazyka.

Príklad 1.4 (Jednoduchý program: Práčka.): Algoritmus práčky sa dá rozdeliť do viacerých samostatných celkov. Uvažuje sa základný program s nasledujúcimi krokmi:

- máčanie
- pranie
- plákanie
- žmýkanie

[2]Ide skôr o stratégiu štrukturovania programu, ale vo väčšine programovacích prostredí je možné využitie grafickej reprezentácie tejto stratégie, preto je v práci uvádzaný ako kvázi-jazyk.

Každý z vymenovaných krokov tvorí v programe samostatný podprogram. Prechod od jedného kroku k druhému je podmienený dokončením predchádzajúceho podprogramu alebo nejakou špecifickou podmienkou (napríklad napustenie vody do určitého objemu v prípade máčania môže signalizovať konečnú podmienku pre podprogram). Po jej dosiahnutí sa opustí vykonaný krok, prechod je uvoľnený a začne sa vykonávať ďalší krok pranie a jemu príslušná monožina akcií. V tomto prípade to môže byť striedavé točenie bubna na určitý časový interval. Po vykonaní určitej sekvencie pre podprogram prania sa dosiahne konečná podmienka pre opustenie tohoto kroku, čo môže byť v tomto prípade vyčerpanie špinavej vody. Ak je vyčerpané, zaháji sa proces plákania. Až dosiahne koncovú podmienku, čo môže byť znova vyčerpaná voda sa zaháji proces žmýkania. Po dosiahnutí koncovej podmienky pre žmýkanie je program ukončený. Názorné prekreslenie do SFC je uvedené na obrázku A.5 uvedeného v prílohe A na strane IV.

Pre principiálne pochopenie by mal byť takýto triviálny príklad dostatočný. Pre presný popis a definície je vhodná online dostupná literatúra: (Teco a.s., 2007*a*)

1.2.4 Ladder Diagram (LD)

Je grafický programovací jazyk. Základné logické prvky sú *ON* a *OFF*, ktoré si je možné analogicky predstaviť ako spínač v zapnutom a vypnutom stave. Zápis je analogický s rebríčkovými diagramami pužívanými v elektrotechnických schémach. Program je vytváraný *sieťami (networks)* tzv. rebríčkov, ktoré vzniknú spojením jednotlicých logických prvkov. Každá sieť je zložená z logických prvkov sériovo-paralelným zapojením a zakončená výstupom. Je možné si ju predstaviť ako grafický prepis logickej funkcie, ktorá produkuje logický výsledok. Pre lepšiu názornosť je vytvorený príklad 1.5. Celý program je vytváraný sadou za sebou idúcich sietí. To znamená, že program vytvorený v tomto jazyku je vykonávaný sekvenčne, dvoma smermi. Zľava doprava pre vyriešenie výsledku danej siete a zhora nadol pre vykonanie jednotlivého programu. Inými slovami, majme program s troma sieťami. Každá predchádzajúca sieť nech produkuje logickú hodnotu pre nasledujúcu sieť. Program bude vykonávaný zľava doprava na prvej sieti, kým sa nedôjde na výsledok, ten bude použitý v druhej sieti pre logickú hodnotu elementu a k produkcii výsledku druhej siete. Výsledok druhej siete sa použije v tretej sieti a vyprodukuje sa výsledok tretej siete. Takže program bol vykonávaný vždy dvoma smermi. Čo je dôležité uviesť, program sa vykonáva cyklicky, takže po dosiahnutí výsledku poslednej siete je vykonávaná znova prvá.

Príklad 1.5 (Logická funkcia XOR vytvorená v LD): Pre lepšiu predstavu ako fungujú v LD rebríčky sa uvedie funkcia XOR (eXclusive OR). Majme zadané dve premenné. Pravdivostná tabuľka je uvedená v tabuľke 1.2. Logická funkcia z nej odvodená je v tvare:

$$Y = (\bar{a} \wedge b) \vee (a \wedge \bar{b}) \tag{1.3}$$

a	b	Y
0	0	0
0	1	1
1	0	1
1	1	0

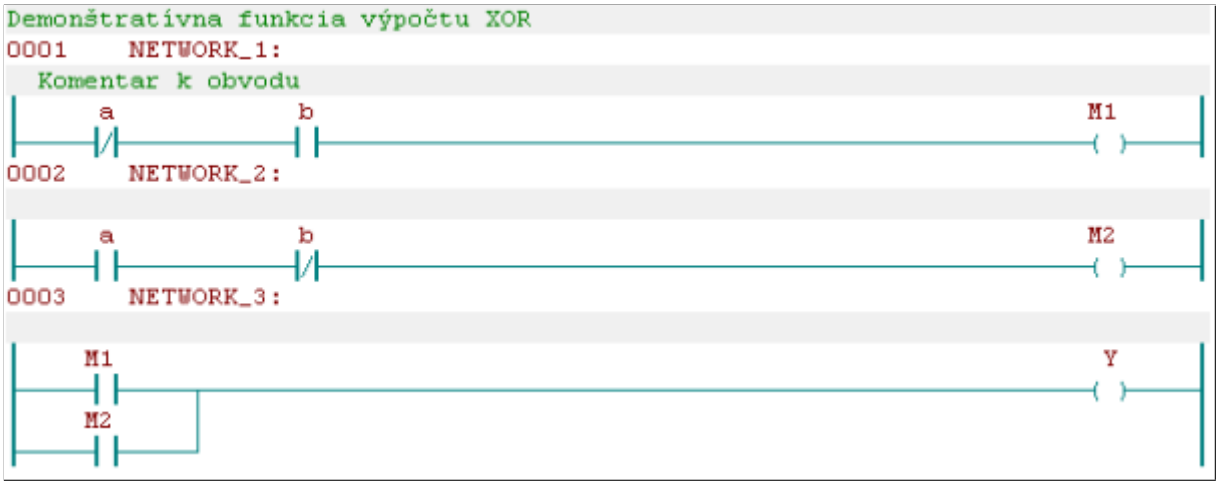
Tabuľka 1.2: Pravdivostná tabuľka logickej funkcie XOR

Prepisom do LD dostaneme štruktúru uvedenú na obrázku 1.5. Prvá a druhá sieť produkujú medzivýsledky, ktoré sú použité v tretej sieti na spočítanie výsledku.

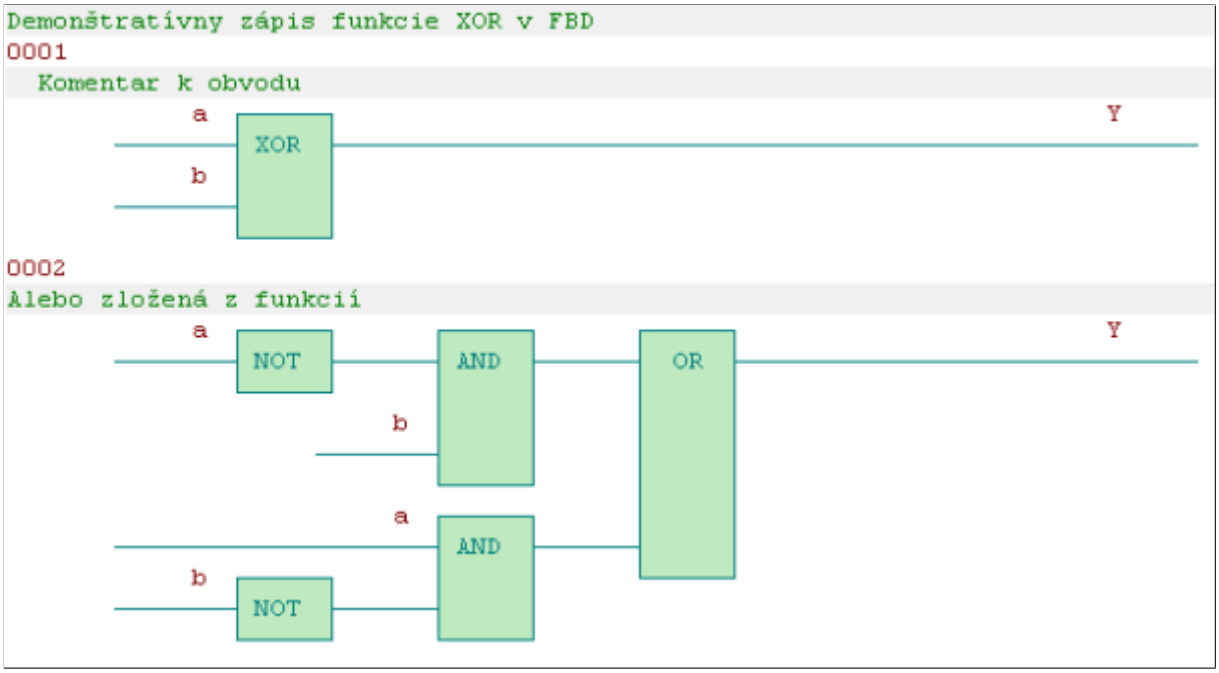
1.2.5 Function Block Diagram (FBD)

Function Block Diagram je druhý z grafických jazykov ktoré norma IEC/EN 61131-3 definuje. Podobne ako v programovacom štandarde LD tu existujú siete. Každá sieť je naproti LD zložená už z funkčných blokov ktoré majú nadefinované základné a odvodené logické funkcie. Program je potom zložený zo vzájomne spojených logických blokov. Ide v podstate o anaógiu vytvárania logických obvodov za pomoci dostupných prvkov realizujúcich základné a odvodené boolovské funkcie. Program je vykonávaný po jednotlivých sieťach podobne ako v prípade LD. Pre názornú ukážku je uvedený príklad 1.6. V porovnaní s príkladom 1.5 je vidieť hlavne úspornosť zápisu funkcie XOR, keďže tá je už zadefinovaná ako odvodená boolovská funkcia a je obsiahnuta ako blok v FBD.

Príklad 1.6 (Logická funkcia XOR reprezentovaná v FBD): Logická funkcia XOR je uvedená v tabuľke 1.2 s príslušnou logickou rovnicou (1.3). Implementácia v jazyku FBD (v prostredí Mosaic) je uvedená na obrázku 1.6.



Obr. 1.5: Demoštrácia možného zápisu funkcie XOR v jazyku LD



Obr. 1.6: Demoštrácia možného zápisu funkcie XOR v jazyku FBD

1.2.6 Instruction List (IL)

Je jeden z textových programovacích jazykov definovaných normou. Zápisom je veľmi podobný asembléru[3]. Má definovanú sadu inštrukcií *(operátorov OP)*. Program je tvorený postupnosťou riadkov. Na každom riadku je zapísaný operátor spolu s *modifikátorom* a *operandom*. Operátor určuje aká akcia bude nad operandom vykonávaná. Modifikátor určuje či sa bude inštrukcia príslušného operátora vykonávať podľa aktuálneho stavu výsledku v akumulátore, teda hodnota `true` alebo `false`. Operand je premenná nad ktorou sa daná operácia vykonáva. Je dobré podotknúť že každý výsledok z predchádzajúceho riadka je ukladaný do akumulátora. Ten slúži ako pamäť medzivýsledku nad ktorým sa budú vykonávať ďalšie operácie podľa nasledujúcich riadkov programu. Algoritmus vypočítavania programu na každom riadku je možné popísať nasledujúcou logickou funkciou:

$$vysledok := vysledok\ OP\ operand \tag{1.4}$$

Presný inštrukčný súbor je uvedený napríklad v dokumente (Teco a.s., 2007*a*) dostupnom online. Pre potreby tejto sekcie postačí jednoduchý príklad 1.7 s minimom využitých inštrukcií, ktorých význam je intuitívny.

Príklad 1.7 (Príklad výpočtu funkcie v IL): Je zadaná logická funkcia (1.5). Zápis do programovacieho štandardu IL je uvedený nižšie.

$$Y = (a \wedge b) \vee \bar{b} \tag{1.5}$$

```
------------------------- Funkcia vytvorená v prostredí Mosaic ---------------------------------
FUNCTION Y: BOOL
        ld      a                               // načítanie operandu a do akumulátora
        and     b                               // (akumulátor & b) -> akumulátor
        orn     b                               // (akumulátor & not b) -> akumulátor
        st      Y                               // akumulátor -> Y
END_FUNCTION
-------------------------------------------------------------------------------------------------
```

1.2.7 Structured Text (ST)

Je posledný z textových jazykov definovaných normou IEC/EN 61131-3. V porovnaní s jazykom IL sa jazyk ST zaraďuje do vysokoúrovňových programovacích jazykov. Program

[3]Asemblér je nízkoúrovňový programovací jazyk (na úroveni registrov) využívajúci priamo inštrukčnú sadu daného procesora.

v ňom zapísaný je štrukturovaný na bloky (funkčné, podmienené, iteračné). Je viazaný na podporu spoločných prvkov, takže v programe vytvorenom podľa tohoto štandardu (ST) je možné využívať premenné a prvky vytvorené v ostatných programovacích štandardoch (IL,FBD,SFC,LD). Názorná ukážka programu je v príklade 1.8. Syntaxou sa jazyk podobá rozšíreným programovacím jazykom ako Pascal a Fortran.

Príklad 1.8 (Príklad nájdenia maximálneho prvku v poli): V komplexnom riadiacom algoritme je zadané globálne pole, do ktorého sa ukladajú data z riadeného procesu. Periodicky s určitým časovým intervalom, je potrebné nájsť štatistické údaje z poľa, napríklad maximálnu hodnotu. Pre nájdenie maxima slúži jednoduchá funkcia uvedená vo výpise programu.

```
------------------------- Funkcia pre nájdenie maxima v poli ------------------------------------
FUNCTION getMaxNo : INT                                       // deklarácia funkcie

  VAR_INPUT
           numbers  : array [0..100] of int;                  // vstupný parameter
  END_VAR

  VAR_TEMP
          maximum        : int := 0;                          // pomocná premenná
          i              : int;
  END_VAR                                                     // iteračná premenná

  for i:= 0 to sizeof(numbers) do
      if maximum < numbers[i] then                            // porovnanie starej a novej hodnoty
         maximum := numbers[i];                               // priradenie novej hodnoty
      end_if;
  end_for;
  getMaxNo := maximum;                                        // vrátenie hodnoty funkcie
END_FUNCTION
------------------------------------------------------------------------------------------------
```

1.3 Motivácia na využitie MBD v automatizácii

Základné aspekty, ktoré slúžia na motiváciu pre využite metódy MBD už boli stručne spomenuté v úvode tejto práce. K lepšiemu priblíženiu sa problému je dobré si predstaviť situáciu v priemysle, hlavne prístup investora a dodávateľa. Práve pri nasadzovaní tejto metódy do praxe stoja na pozadí dve zásadné otázky. Cena projektu a časová efektivita realizácie. Z pohľadu investora je spôsob realizácie nezaujímavý. Základným kritériom na realizáciu projektu je funkčnosť a spoľahlivosť za čo najnižšie náklady. Na opačnej strane, strane dodávateľa, tak vzniká v dôsledku časového tlaku tendencia nadmerného zjednodušovania problému. To znamená, že projektanti siahnu vždy po najjednoduchšom a najrýchlejšom riešení, ktoré splňuje požiadavky špecifikované investorom. Na zahrnutie modelovania do procesu vývoja jednoducho nie je čas a prostriedky. Projekt sa vyhotoví a vyskúša na fyzickom hardvéri väčšinou experimentálne systémom pokus-omyl. Ak všetko funguje ako má, tak nie je žiaden problém. Horším prípadom je ak realizácia projektu zlyháva na chybách v návrhu. Vtedy sa pristupuje na vytváranie modelu sústavy, aby bolo možné problém rýchlejšie a efektívnejšie identifikovať a tým znížít hospodársky dopad vyplývajúci pre obidve strany - dodávateľa a investora. V konečnom dôsledku sa potom zistí, že eliminovanie modelovania na začiatku procesu vývoja malo za následok zlyhanie navrhnutej technológie a všetok čas strávený vývojom bol v podstate zbytočný. To sa samozrejme odzrkadlí do ceny a času potrebného na vyhotovenie a suma celkových nákladov presiahne hranicu pre prípad projektu už so zahrnutým modelovaním na začiatku. Model sa koniec koncov musel zrealizovať, ale za zvýšenú cenu. Ďalším scenárom je porucha spôsobená opotrebovaním. Technológia funguje bez problémov a spoľahlivo po dobu pár rokov. V okamžiku ale vznikne výpadok ktorý zastaví prevádzku, čo môže pri sériových výrobách znamenať obrovské straty. Detekcia chyby trvá určitý čas, oprava trvá určitý čas, dodávka nového dielu trvá určitý čas. Dostatočnou simuláciou problému a porovnávaním modelovaného a reálneho systému by bolo možné túto chybu v predstihu detekovať a tým ušetriť časové náklady spojené s opravou. Tento prístup je ale rentabilný až v čase poruchy, o ktorej sa samozrejme na začiatku projektu, hlavne zo strany investora neuvažuje. Za využitie metódy MBD v automatizačnej praxi hovorí niekoľko faktov:

Prvým z nich je možnosť diagnostikovať možné nedostatky navrhnutého systému v predstihu už vo fáze vývoja. Oproti experimentálnemu návrhu to prináša možnosť testovať systém na marginálne hodnoty bez toho, aby došlo k deštrukcii skutočného systému. Okrem toho sa eliminuje aj prípadná katastrofa hroziaca pri neopatrných experimentoch

na fyzikálnej sústave.

Druhým z nich je možnosť využitia návrhového modelu na diagnostiku poruchy na reálnej sústave. Buď to priamo, alebo nepriamo. *Priama implementácia* modelu znamená, že algoritmus riadenia a model sa vypočítavaju paralelne. Dáta namerané na reálnej sústave sa porovnajú s dátami ktoré produkuje referenčný model pri rovnakých vstupných parametroch pre obe sústavy. Výsledkom sú odchylky reality od modelovanej skutočnosti. Pri vysokých hodnotách odchyliek je to znamenie, že sa s fyzikálnou sústavou niečo deje, čo môže byť indikáciou skorého výpadku v systéme. Toto dáva znamenie, že je potrebné sa pripraviť na servis a tým znížiť časové a finančné náklady spojené s nepredpokladanou chybou. *Nepriama implementácia* modelu pri diagnostike je taká, že model nie je súčasťou algoritmu riadenia. Po dátach z modelu sa siaha až keď nastane porucha. Tá je za pomoci modelu ľahšie identifikovateľná a servis sa môže pripraviť už na konkrétny problém, než je hľadanie poruchy priamo na fyzikálnom systéme. Šetrí to čas a kruté pohľady na zúfalého technika snažiaceho sa pod tlakom nájsť rýchlo príčinu výpadku. Tento spôsob nie je efektívny v rovnakej miere než je priama implementácia, ale je určite účinnejší než je prístup bez modelu.

Tretí fakt je možnosť predikcie výsledku z nastavených vstupných parametrov. Veľkú výhodu to má pri procesoch s časovými konštantami rádovo v hodinách až dňoch. V týchto prípadoch je experimentálny návrh systémom pokus-omyl skoro nemožný kôli časovej náročnosti. Tu dáva model možnosť zrýchliť proces návrhu tým že pri zmene vstupných parametrov je možnosť dopočítania očakávanej odozvy a nie je nutné čakať (v lepšom prípade) pár hodín.

Štvrtý fakt je možnosť prezentácie výsledkov vo forme prehľadných grafov ešte pred započatím projektu. Má to hlavne komerčný zmysel. Priblíži investorovi problém implementácie a nie len konečný efekt. To zvyšuje šance na úspešné získanie projektu.

Jedným z podstatných problémov na ktorý MBD naráža je počiatočná časová náročnosť pri tvorbe modelu sústavy. Predstavuje zvýšené nároky na vývojára systému, ako aj na softvérové vybavenie. Vždy je na zvážení v akom rozsahu sa model pri návrhu využije a či náklady na jeho tvorbu vyvážia vyššie spomínané výhody. T.j. využitie modelu musí implikovať efektivitu návrhu tak ako aj časovú efektivitu odstraňovania prípadných chýb navrhnutého systému. A nakoniec posledný problém je, že výhody modelu sú skryté až do prvej poruchy na systéme, takže vzniká problém, že model je väčšinou podceňovaný a pre projekt predstavuje zdanlivé zvýšenie nákladov. Práca bude smerovaná hlavne na ukázanie výhod priamej implementácie modelu na platformu PLC a možnosti jehžo využitia pre diagnostiku.

Kapitola 2

Generovanie kódu

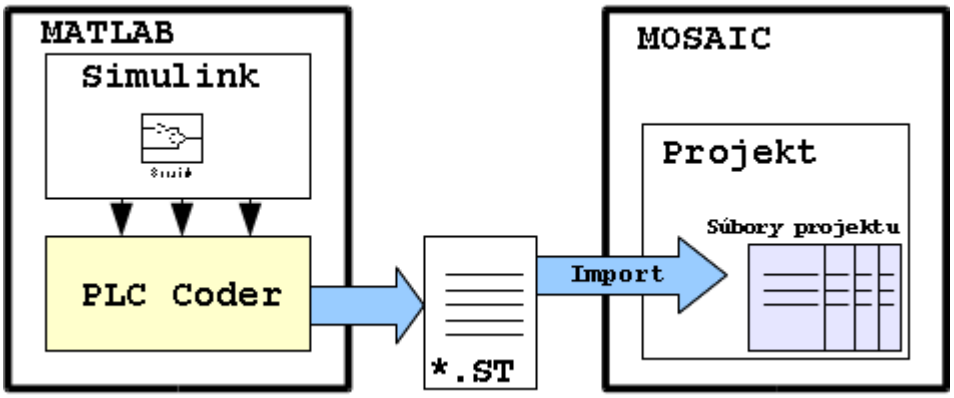
Kapitola sa bude zaoberať implementovaním rôznorodých modelov na platformu PLC. Budú ukázané simulačné schémy, simulácie a bloky generovaných kódov, ktoré môžu byť zaujímavé pre programátorov z algoritmického hľadiska. Budú diskutované a rozobraté najčastejšie problémy s kompatibilitou medzi generovaným kódom a vývojovým prostredím Mosaic.

2.1 Implementačné možnosti

Pred využitím možností PLC Codera je potrebné zistiť akým spôsobom je možné vygenerovaný kód implementovať do vývojového prostredia Mosaic. Po bližsom preskúmaní možností exportu kódu v PLC Coderi sa zistilo že neexistuje priama podpora projektových súborov aké Mosaic využíva. Ako bolo už v úvode práce spomenuté, PLC Coder podporuje projektovú sadu súborov viacerých významných výrobcov PLC (napríklad Siemens, Rockwell, WAGO atď.). To znamená že funkčný blok vytvorený v Simulinku bude prekódovaný podľa normy IEC/EN 61131-3 do jazyka ST s tým, že okrem kódu pre funkčný blok bude vygenerovaná aj sada podporných súborov vytvárajúcich projekt pre dané vývojové prostredie. Má to výhodu v ľahšom importe do vývojového prostredia. Pre potreby tejto práce táto možnosť nie je nápomocná. Pri prenose kódu medzi Matlabom a Mosaicom sa pristúpilo k trocha komplexnejšej metóde, ale o nič menej atraktívnejšej, ako to je v prípadoch pre vývojové prostrostredia renomovaných výrobcov automatov.

Súborový systém projektu vytváraného v Mosaicu je celkom komplexný. Nie je potrebné rozoberať aké súbory sa vytvárajú. Hlavné a dôležité je, že sa do projektu dajú

zahŕňať súbory s príponami podľa programovacieho jazyka definovaného normou v ktorom boli programované. To znamená že projekt v Mosaicu podporuje súbory *.IL, *.ST, *.FBD, *.SFC, *.LD. Pri nastavení PLC Codera na generovanie kódu do formátu „*Generic*“ je výsledkom jediný súbor s príponou .ST. To znamená že pri vytvorení nového projektu v prostredí Mosaic je možné do neho zaradiť priamo súbor, ktorý je výsledkom konverzie PLC Codera. Principiálne to vedie na spôsob prenosu kódu znázorneného na obrázku 2.1.

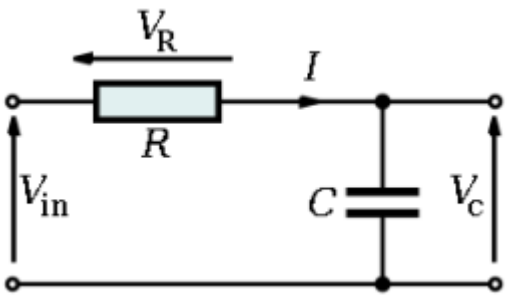


Obr. 2.1: Možnosť prenosu kódu medzi Matlabom a Mosaicom

Ide o pohodlný spôsob pridávania nových blokov k existujúcemu projektu. Po naimportovaní novo vygenerovaného súboru je nutné ešte v projekte nastaviť správne poradie prekladu súborov. Je to dôležité z dôvodu sekvenčnej kompilácie kódu ktorú vykonáva kompilátor xPRO. Možné následky a nezrovnalosti pri komplácii budú rozobraté v sekcii *Implementačné problémy* na strane 46 v tejto kapitole.

2.2 Sústava prvého rádu

Ako úvodný demonštračný príklad sa uvedie primitívny systém prvého rádu. Klasickým demonštrantom tohoto typu je pasívny filter dolná priepusť, t.j. RC-článok s obvodom uvedeným na obrázku 2.2. Využitý bol RC s charakteristickou frekvenciou $f_c = 1Hz$.



Obr. 2.2: Elektrická schéma dolnej priepusti RC

Prenos systému zo vstupu na výstup je definovaný vzťahom:

$$P(s) = \frac{1}{RCs + 1} = \frac{\omega_c}{s + \omega_c} = \frac{2\pi f_c}{s + 2\pi f_c} \tag{2.1}$$

Po dosadení charakteristickej frekvencie sa získa prenos pre danú sústavu v tvare:

$$P(s) = \frac{2\pi}{s + 2\pi} \tag{2.2}$$

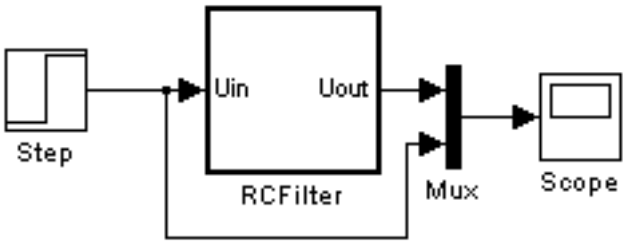
Diskretizáciou pomocou metódy „**Z**ero **O**rder **H**old“ (ZOH) sa získa prenos systému v podobe:

$$P(z) = \frac{0.4665}{z - 0.5335} \tag{2.3}$$

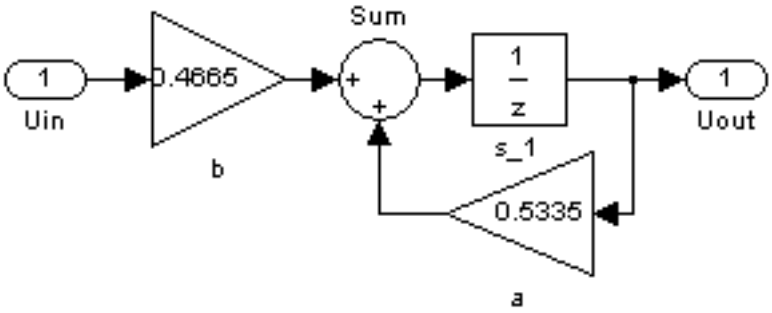
2.2.1 Popis implementácie

Pre implementáciu systému v Simulinku, definovaného priamo vzťahmi pre prenos 2.1, nie je potrebné vynaložiť veľa úsilia. Pre modelovanie systémov ktoré budú prenášané na platformu PLC je vytvorená knižnica *plclib* a vyvoláva sa rovnakým príkazom z príkazového riadka Matlabu. V tomto nástroji už existuje priamo funkcia, kde je možné zadefinovať polynómy čitateľa a menovateľa racionálnej funkcie prenosu 2.3. V tomto prípade bol pre namodelovanie prenosu systému použitý blok oneskorovacieho člena, sumátora a zosilnenia. Výhodou takéhoto modelovania, oproti matematickému poprípade algoritmickému popisu v .ST, je prehľadnosť a intuitivita pri vytváraní systémov. Schéma, ktorá bola využitá na generovanie kódu do PLC je uvedená na obrázku 2.4. Pre potreby kódera je nutné vytvoriť z namodelovaného systému funkčný blok, ktorý bude mať v tomto prípade jeden vstup a jeden výstup a vo vlastnostiach bude mať nastavený príznak „treat as atomic unit“ čo znamená že s blokom sa bude zachádzať ako s nedeliteľnou jednotkou. Subsystém je uvedený na obrázku 2.3. Po vytvorení schémy sa nechá pomocou PLC Codera vygenerovať kód podľa normy .ST. Pred samotnou generáciou kódu je nutné zvoliť v

nastaveniach PLC Codera voľbu „*Generic*“, čím nástroju povie, že výstupom generátora bude jeden súbor s príponou *.ST. Pre implementáciu do Mosaicu je potrebné tento súbor zahrnúť do projektu, v ktorom bude generovaný funkčný blok využívaný. Podrobný postup je uvedený v prílohe A na strane I.



Obr. 2.3: Funkčný blok dolnej priepusti vytvorenej v Simulinku



Obr. 2.4: Simulačná schéma dolnej priepusti vytvorenej v Simulinku

Pri výpočte hodnôt výtupu modelu môžu nastávať odchylky dynamiky spôsobené neekvidistantnou výpočtovou periódou (resp. vzorkovacou periódou T_s, pre ktorú bol systém diskretizovaný). V PLC je problematické zaručiť obslúženie programu s nižšou prioritou presne na milisekundy, ale vždy je výpočet zaťažený časovým oneskorením. To prináša malé nezrovnalosti pri porovnaní skutočnej a modelovanej hodnoty. Spôsob ako túto chybu minimalizovať je rozobratý v podkapitole *Linearizovaný model elektromotora* v sekcii *Popis implementácie* na strane 28.

2.2.2 Vygenerovaný kód

Stručné nastavenia PLC Codera a úpravy potrebné na to aby bolo možné kód vygenerovať boli stručne spomenuté v sekcii *Popis implementácie*. Pre kompletnosť sa konfigurácia uvedie znova. Pre generovanie kódu je potrebné nastaviť v konfigurácii PLC Codera platformu na ktorú bude kód generovaný. Pre potreby tejto práce a pre prenesiteľnosť do vývojového prostredia Mosaic je nutné nastavenie „*Generic*“. V tomto prípade výstupom generátora jediný súbor s príponou .ST s vygenerovaným funkčným blokom. Pomenovanie súboru je automatické podľa názvu modelu vytvoreného v Simulinku. Mená parametrov fukčného bloku, tak ako aj jeho názov (už v podobe štandardu .ST) sú generované podľa názvov použitých v simulačnej schéme. Pre lepšiu orientáciu v grafickej schéme a vo vygenerovanom kóde je dobré pomenovať bloky schémy podľa ich funkcie. Je to výhodné z toho dôvodu, že PLC Coder priraďuje mená premenným v kóde podľa ich názvu v grafickej schéme. Generovaný kód sa stáva užívateľovi transparentnejší a intuitívnejší čo je výhoda pri prípadných menších úpravách kódu. PLC Coder dovoľuje zapínanie a vypínanie komentárov ktoré taktiež môžu pomocť vytvoriť kód transparentnejší. V tomto prípade bola táto možnosť vypnutá kôli redukcii množstva kódu.

Zdrojový kód vygenerovaného funkčného bloku RC člena [1]

```
---------------------------------------- Funkčný blok RCFilter ----------------------------------
01)         FUNCTION_BLOCK RCFilter
02)         VAR_INPUT
03)             ssMethodType: SINT;
04)             Uin: LREAL;
05)         END_VAR
06)         VAR_OUTPUT
07)             Uout: LREAL;
08)         END_VAR
09)         VAR
10)             s_1_DSTATE: LREAL;
11)         END_VAR
12)         VAR_TEMP
13)         END_VAR
14)         CASE ssMethodType OF
15)             SS_INITIALIZE:
16)                 s_1_DSTATE := 0;
17)             SS_OUTPUT:
18)                 Uout := s_1_DSTATE;
19)                 s_1_DSTATE := (0.4665 * Uin) + (0.5335 * s_1_DSTATE);
20)         END_CASE;
21)         END_FUNCTION_BLOCK
-------------------------------------------------------------------------------------------------
```

[1] Čísla riadkov boli pridané z dôvodu lepšieho odkazovania.

Pri prvom porovnaní vygenerovaného kódu a generačnej schémy na obrázkoch 2.4 a 2.3 je vidieť že mená premenných a funkčného bloku neboli generované náhodne, ale kooperujú s menami priradenými v schémach. Premenné `Uin` a `Uout` predstavujú vstup a výstup systému. Premenná `s_1_DSTATE` predstavuje pamäť funkčného bloku systému RC a kooperuje s oneskorovacím členom $\frac{1}{z}$ v simulačnej schéme z obrázku 2.4. Štruktúra funkčného bloku dovoľuje prácu bloku v dvoch režimoch. Prvý inicializačný, kedy sa blok inicializuje na počiatočné podmienky, to znamená že sa vymaže pamäťová premenná (riadok 16). Druhý je výkonný, kedy sa spočítava s každým volaním bloku nová hodnota výstupu filtra (riadok 18 a 19). O prepínanie medzi týmito módami sa stará premenná `ssMethodType`. Celý algoritmus výpočtu modelu je založený na riadkoch 18 a 19. Na tomto riadku je pekne vidieť analógia medzi kódovou a grafickou interpretáciou modelu na obr. 2.4. Výstup starého stavu `s_1_DSTATE` je prenásobený konštantou **a**, čo predstavuje dynamiku systému. Vstup systému `Uin` je prenásobený konštantou **b** čo predstavuje vstupnú maticu systému, v tomto prípade skalárna veličina, keďže sa jedná o SISO (***S**ingle **I**nput **S**ingle **O**utput*) systém. Znamienko „+“ predstavuje sumačný člen *Sum* označený v schéme. Operátor priradenia „:=“ k premennej `s_1_DSTATE` je analogický vstupu do oneskorovacieho člena v schéme. Na riadkoch 15 a 17 sú makrá `SS_INITIALIZE` a `SS_OUTPUT`, ktoré boli generované automaticky ako globálne premenné. Slúžia pre definíciu rozhrania režimu v akom sa bude funkčný blok využívať. Deklarácia je uvedená nižšie v deklaračnom bloku vygenerovaného kódu.

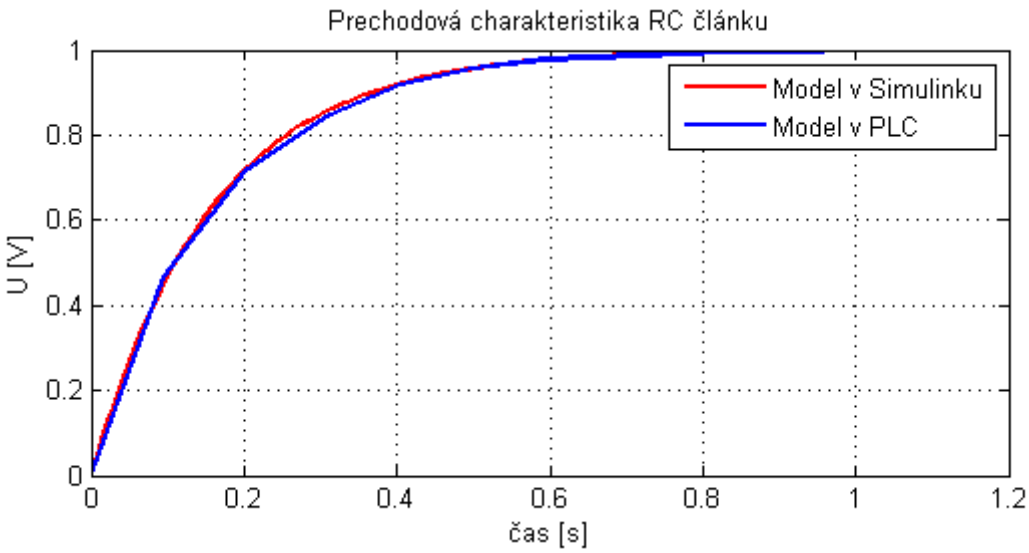
Deklarácia globálnych premenných a konštánt

```
------------------------------- Deklaračný blok vygenerovaného kódu --------------------------
VAR_GLOBAL CONSTANT
    SS_INITIALIZE: SINT := 2;
    SS_OUTPUT: SINT := 3;
END_VAR
VAR_GLOBAL
END_VAR
------------------------------------------------------------------------------------------------
```

Je dobré podotknúť, že makrá sú definované v každom vygenerovanom súbore z PLC Codera. Ak sa v Mosaicu použije viacero funkčných blokov s rovnakými deklaráciami globálnych premenných, tak je nutné tieto deklaračné bloky zmazať. V opačnom prípade bude kompilátor Mosaicu xPRO hlásiť chybu.

2.2.3 Simulácia a porovnanie

Po odstránení nekompatibilít vo vygenerovanom kóde je jeho následnej implementácii na platformu PLC bol model odsimulovaný. Na simuláciu implementovaného modelu na platforme PLC bol použitý nástroj GraphMaker. Výhoda je v tom, že dovoľuje vykresľovať priebehy globálnych premenných definovaných v projekte. Následne je možný export hodôt zmeranej charakteristiky v textovom súbore. To sprístupňuje pohodlnú cestu na porovnávanie charakteristík generovaných modelom v Simulinku a charakteristík generovaných modelom v PLC. Pre porovnávanie bol využitý nástroj Matlab. V prípade RC filtra je na demoštráciu funkcie využitá prechodová charakteristika modelu. Porovnanie výstupov oboch modelov je na obrázku 2.5, kde ako referenčný signál bol využitý jednotkový skok v čase $t = 0s$.



Obr. 2.5: Porovnanie výstupov modelov z rozličných platforiem

Porovnaním oboch charekteristík je vidieť, že modely súhlasia. Zlomové časti v modrej charakteristike boli spôsobené interpoláciou vzorkov skokovej funkcie, ktorá bola výsledkom simulácie v PLC. Malé odchylky v nábehu v oblasti v čase $t = 0s$ až $t = 0.4s$ sú spôsobené práve problémom „kĺzavosti“ periódy medzi výpočtami modelu v PLC. V PLC totiž nie je možné na 100 % zaručiť presný okamžik výpočtu. To sa prejaví práve v odchylkách dynamiky modelu oproti skutočnému systému. Je možné si to predstaviť ako desynchronizáciu medzi modelom a reálnou sústavou. Odchylky sú rádovo v milisekundách, čo nepredstavuje veľký problém. S prihliadnutím na to, že modely sa budú

využívať hlavne na diagnostiku pomalých systémov s časovými konštantami rádovo v sekundách. To znamená, že posuny dynamiky budú v porovnaní s rýchlosťou zmeny výstupu sústavy zanedbateľne malé. Koncové hodnoty zostávajú zachované, čoho je možné využiť v diagnostike koncového stavu. Aplikovateľné to môže byť hlavne u rýchlych sústav, ako je napríklad elektromotor. Zmena v dosiahnutí koncového stavu systému môže predikovať nežiadané zmeny v štruktúre fyzikálnej sústavy (napríklad zadieranie ložiska u elektromotora). Podľa toho je možné predikovať možnú závadu. Viacej o praktickom probléme diagťnózy chžyby v systéme bude rozobraté v kapitole 3.

2.3 Linearizovaný model elektromotora

V odstavci bude využitý upravený model elektromotora z motivačného príkladu 1.3. Naproti predchádzajúcemu prípadu sa bude uvažovať lineárny model elektromotora. Ten sa získa jednoducho tak, že v rovniciach 1.1 a 1.2 sa zanedbajú nelineárne členy, takže vzniká lineárne priblíženie systému s rovnicami:

$$L_a \dot{I}_a(t) = -R_a I_a(t) - \psi\omega(t) + U_a \tag{2.4}$$

$$J\dot{\omega}(t) = \psi I_a(t) - M_{f1}\omega(t) - M_l \tag{2.5}$$

Po prevedení rovníc do stavového popisu sa získa:

$$\begin{bmatrix} \dot{I}_a \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{R_a}{L_a} & -\frac{\psi}{L_a} \\ \frac{\psi}{J} & -\frac{M_{f1}}{J} \end{bmatrix} \cdot \begin{bmatrix} I_a \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L_a} & 0 \\ 0 & -\frac{1}{J} \end{bmatrix} \cdot \begin{bmatrix} U_a \\ M_l \end{bmatrix} \tag{2.6}$$

$$\boldsymbol{y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} I_a \\ \omega \end{bmatrix} \tag{2.7}$$

2.3.1 Popis implementácie

Pre simuláciu a implementáciu boli použité konštanty identifikovaného motora z tabuľky 1.1. Po dosadení parametrov do stavového popisu (rovnice 2.6 a 2.7) sa získali matice potrebné pre simuláciu, ale spojitého systému. Pre výpočet modelu v PLC je potrebný diskrétny popis sústavy čo implikuje následnú diskretizáciu systému. Okrem

toho PLC Coder podporuje len vybrané prvky, kde mimochodom nie je možné aplikovať komponenty spojitého systému[2]. Prvky podporované PLC Coderom sú obsiahnuté v spomínanej knižnici *plclib* (volaná rovnakým príkazom v matlabe). Pre diskretizáciu spojitého systému bola využitá metóda *ZOH* a vzorkovacia perióda $T_s = 0.1s$, čo priemerne odpovedlo času jedného cyklu PLC.

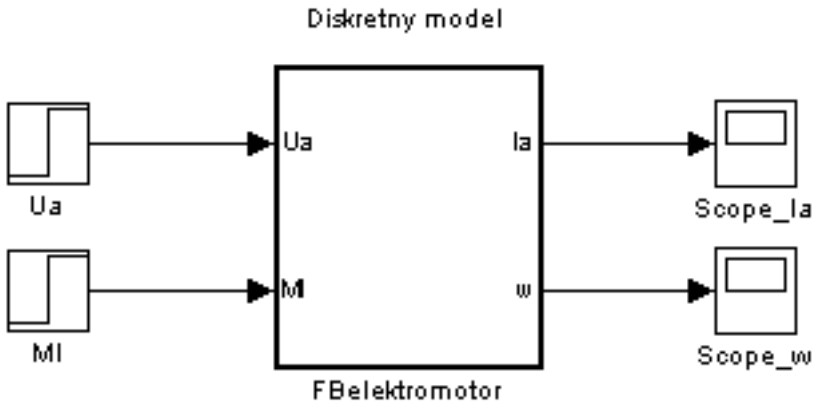
Jedným zo spôsobov ako zabezpečiť konštantnú periódu medzi dvoma výpočtami výstupov funkčného bloku modelu bolo nechať spočítať výstup raz za cyklus. To znamená, že na diskretizáciu by sa použil čas periódy otočky programu, ktorý sa označí T_c. Nevýhoda ale je že podľa náročnosti aplikácie sa čas T_c skracuje alebo predlžuje. Pre diskretizáciu by teda nebolo možné presne povedať aké T_c bude použité. Možný by bol len odhad zmeraním aktuálneho času otočky na PLC pred diskretizovaním a aplikovaním funkčného bloku. Na prvé priblíženie systému tento čas vystačil a aplikoval sa čas vzorkovania $T_s = T_c$. Logicky by sa pri pridávaní výpočtových blokov do programu PLC čas T_c predĺžil priamo úmerne s narastajúcim počtom úloh pre výpočet. Modely ktoré by boli aplikované na začiatku procesu vývoja by sa stali neaktuálne, pretože boli diskretizované pre menšie časy otočky. To by viedlo k myšlienke znova aktualizovať model, pre nový čas T_c. Pri finalizovaní aplikácie v PLC by sa už nepredpokladal rapídny nárast času otočky T_c, a ako referenčná vzorkovacia perióda T_s, pre všetky modely, by sa využil čas otočky T_{cfin} meraný už pri finálnej verzii programu v PLC. To by ale viedlo k nutnej aktualizácia všetkých použitých modelov na vzorkovaciu periódu $T_s = T_{cfin}$. Čo by znamenalo, že všetky generované programy by bolo potrebné vymeniť za nové, ktoré by boli generované už s upravenou vzorkovacou periódou T_s. Samozrejme by to bola jedna z možností ako zbezpečiť kvázi konštantnú periódu výpočtu. Veľkou nevýhodou by ale bolo, že pri nečakaných prerušeniach alebo preskokoch medzi programovými úsekami by tento čas mohol variovať a tým pádom by sa došlo už k spomínaným veľkým hodnotám „kĺzania“ periódy spomínanej v predchádzajúcom príklade s RC článkom. V tomto prípade ale tento postup využitý nebol.

Ako druhá možnosť bol využitý druhý (sofistikovanejší) postup a to využitie hodín reálneho času, ktorými PLC disponuje. To znamená, že perióda vzorkovania pre diskretizáciu modelu sa už počas vývoja nemenila a snaha je zaručiť obsluženie funkčného bloku s požadovanou periódou. PLC rady TECOMAT sú vybavené sadou registrov, ktoré dokážu vyvolávať prerušenia na obslúženie podprogramu, ktorým je v tomto prípade funkčný blok modelu. Hodinami reálneho času sa vyvolá s periódou T_s prerušenie a obslúži

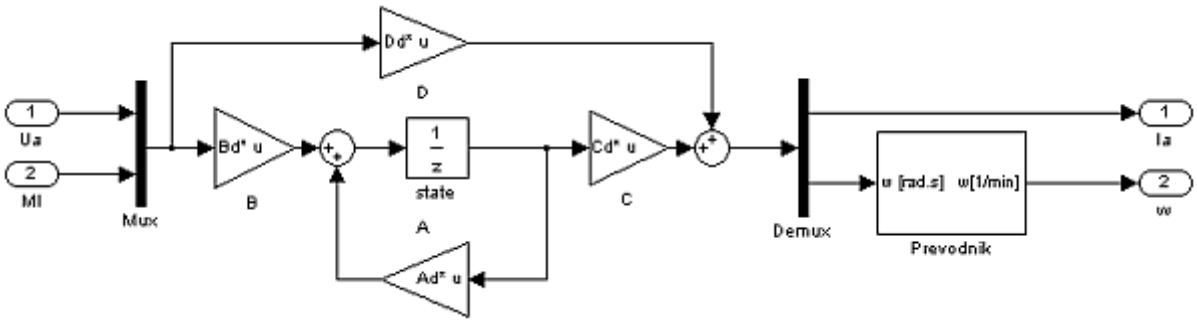
[2]Čo je logické, keďže PLC vypočítava parametre v cykle s periódou otočky programu T_c.

sa funkčný blok. To zabezpečí konštantú hodnotu periódy medzi výpočtami výstupov modelu a vznikne menšia desynchronizácia medzi reálnou sústavou a modelom. Zabezpečí sa tým minimálna chyba dynamiky modelu oproti reálnej sústave.

Pre odsimulovanie a prevod do štandardu .ST bol v simulinku vytvorený grafický model sústavy elektromotora. Vytvorená grafický popis systému je na obrázku 2.7. Model je vytvorený z prvkov dostupných v spomínanej knižnici *plclib*. Pre samotné generovanie kódu schéma v podobe z obrázku 2.7 nestačí. Pre generovanie kódu do PLC pomocou kódera je nutné vytvoriť schému ako funkčný blok, ktorý má vstupné a výstupné parametre. V jazyku Simulinku to znamená, že zo schémy je potrebné vytvoriť subsystém a označiť ho ako samostatnú jednotku. Model je týmto pripravený na generovanie kódu pre PLC. Subsystém využitý v tomto prípade je na obrázku 2.6 znázornený už aj so zdrojami vstupných signálov použitých na simuláciu.



Obr. 2.6: Funkčný blok elektromotora pre generovanie kódu do PLC



Obr. 2.7: Vnútorná štruktúra funkčného bloku elektromotora

2.3.2 Vygenerovaný kód

Konfigurácia pre PLC Coder je ako v prípade jednoduchého RC člena na strane 25. Výstup PLC Codera nastavený na „*Generic*“. Originál kód s komentármi je uvedený v prílohe na strane 28. V nasledujúcich riadkoch budú rozobraté len jednotlivé sekcie generovaného kódu, a ukázané súvislosti medzi generačnou schémou 2.6 a generovaným kódom.

Generovaný kód presne splňuje vyššie popísané závislosti medzi názvami vstupov, výstupov a stavov v generovanom kóde a v generačnej schéme. Vstupné parametre uvedené v bloku `VAR_INPUT` odpovedajú grafickým vstupom na obrázku 2.6 resp. 2.7. Premenná `ssMethodType` bola vytvorená generátorom automaticky kôli možnosti prepínania medzi režimami funkčného bloku. Je hlavným prepínačom medzi inicializáciou funkčného bloku na počiatočné podmienky a normálnym režimom, kedy sa zo vstupných parametrov spočítava výstup. Výhoda je že funkčný blok je možné nastaviť na rôzne počiatočné podmienky. Blok `VAR_OUTPUT` definuje výstupy funkčného bloku v súlade s generačnou schémou. Blok premenných `VAR` obsahuje stavovú premennú `state_DSTATE`. Premenná v sebe ukladá informáciu o aktuálnom stave. V tomto prípade boli použité parametre pre simulačnú schému 2.7 v podobe stavových matíc. Systém elektromotora má dva stavy. Premenná bola z toho dôvodu vytvorená ako pole s dvoma prvkami, kde každý prvok poľa obsahuje informáciu o aktuálnom stave. Premenná `unnamed_idx` v bloku `VAR_TEMP` je premenná slúžiaca k pomocným výpočtom pri počítaní hodnoty aktuálnych stavov.

Zdrojový kód vygenerovaného funkčného bloku elektromotora

```
----------------------- Deklaračná časť funkčného bloku FBelektromotor ------------------------
FUNCTION_BLOCK FBelektromotor
VAR_INPUT
    ssMethodType: SINT;
    Ua: LREAL;
    Ml: LREAL;
END_VAR
VAR_OUTPUT
    Ia: LREAL;
    w: LREAL;
END_VAR
VAR
    state_DSTATE: ARRAY [0..1] OF LREAL;
END_VAR
VAR_TEMP
    unnamed_idx: LREAL;
END_VAR
--------------------------------------------> 1 <--------------------------------------------
```

```
---------------------------------------------> 1 <---------------------------------------------
------------------------- Výkonná časť funkčného bloku FBelektromotor -------------------------
CASE ssMethodType OF
    SS_INITIALIZE:
        Ia_z_DSTATE[0] := 0;
        Ia_z_DSTATE[1] := 0;

    SS_OUTPUT:
        unnamed_idx := (0.60254153513739728 * Ia_z_DSTATE[0]) +
                       (0.78734271379520571 * Ia_z_DSTATE[1]);

        Ia := Ia_z_DSTATE[0];

        w := Ia_z_DSTATE[1] / 0.10471975511965977;

        Ia_z_DSTATE[0] := ((0.51472902397003173 * Ua) + (0.639212236539013 * Ml)) +
                          ((0.0066718455076774071 * Ia_z_DSTATE[0]) +
                          (-0.16963046150495639 * Ia_z_DSTATE[1]));

        Ia_z_DSTATE[1] := ((0.639212236539013 * Ua) + (-4.7701337414445373 * Ml)) +
                          unnamed_idx;
END_CASE;
END_FUNCTION_BLOCK
-----------------------------------------------------------------------------------------------
```

Vo výpise výkonnej časti je uvedený výpočet dynamiky systému. Vychádza sa z prenosových funkcií systému. Podľa nich sú spočítavané jednotlivé príspevky jednotlivých vstupov k daným výstupom. Práve tento výkonný blok prislúcha grafickému bloku s maticami a sumačnými členami na obrázku 2.7. Blok prevodníka je zaradený do algoritmu priamo faktor, ktorým je podelený výstup `w`.

Deklarácia globálnych premenných a konštánt

```
------------------------- Deklarácia globálnych premenných a konštánt --------------------------
VAR_GLOBAL CONSTANT
    SS_INITIALIZE: SINT := 2;
    SS_OUTPUT: SINT := 3;
END_VAR
VAR_GLOBAL
END_VAR
-----------------------------------------------------------------------------------------------
```

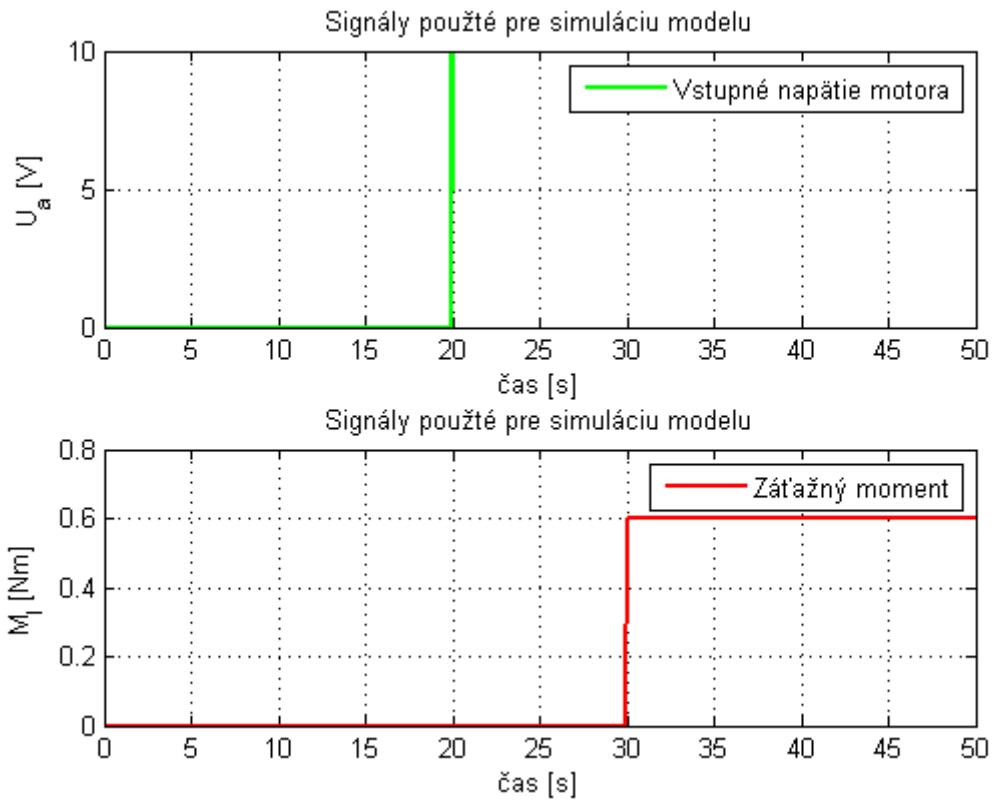
V grafickej schéme neboli definované žiadne globálne premenné ani konštanty. PLC Coder aj napriek tomu generoval dve globálne konštanty. Tie sú potrebné na špecifikáciu rôznych režimov vygenerovaného funkčného bloku. V systéme sa uvažuje inicializácia funkčného bloku na jeho počiatočné podmienky. To znamená, že pri prvom volaní funkčného

bloku je potrebné ho volať s parametrom `SS_INITIALIZE`. S každým ďalším volaním bloku kôli zisteniu vývoja jeho výstupných premenných sa funkčný blok volá s parametrom `SS_OUTPUT`. Vyplýva to zo štruktúry generovaného funkčného bloku, ktorý je uvedený vo výpise výkonnej časti funkčného bloku elektromotora.

2.3.3 Simulácie a porovnanie

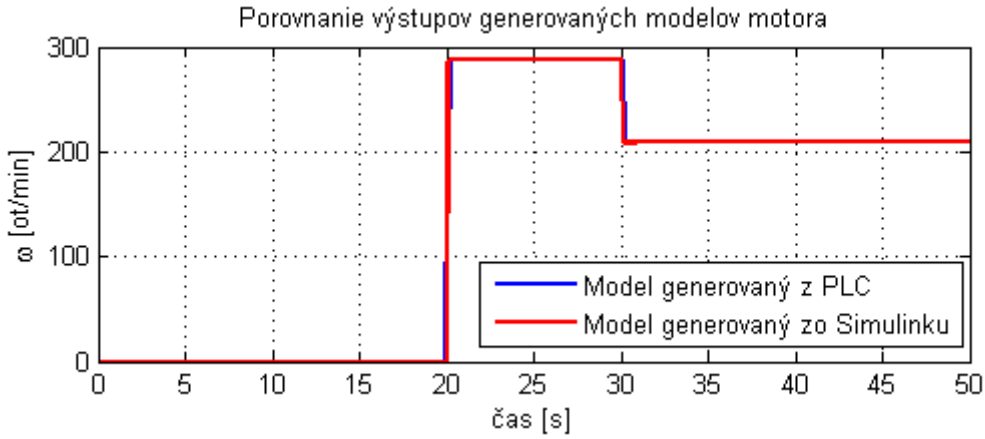
Simulácia bola prevádzaná na dvoch platformách. Po vygenerovaní kódu a odstránení nekompatibilít pre prekladač xPRO prostredia Mosaic, bola nakoniec vyskúšaná funkčnosť modelu na PLC. Pre simuláciu na oboch platformách boli použité skokové signály s parametrami:

- $U_a = 10V$ v čase 20 sekúnd
- $M_l = 0.6Nm$ v čase 30 sekúnd

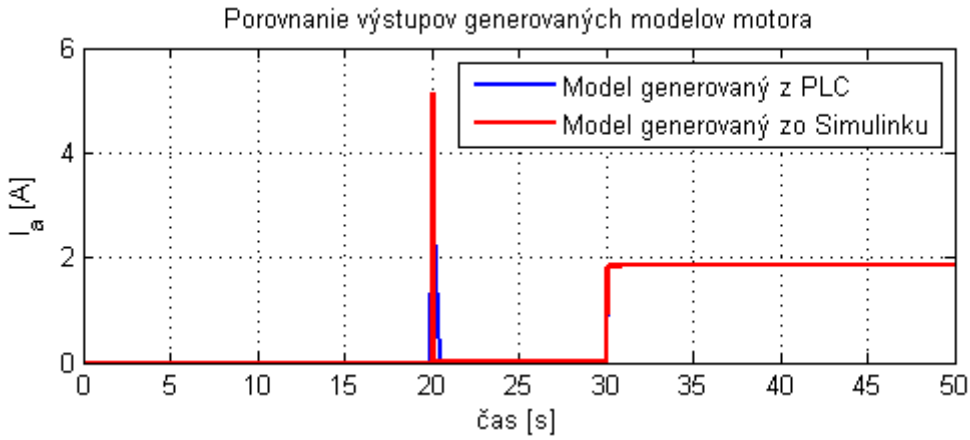


Obr. 2.8: Vstupný signál použitý na simuláciu

Pre lepšiu predstavu sú vstupné signály použité pre simuláciu uvedené na obrázku 2.8. Časové oneskorenia boli volené zámerne pre pohodlnejšiu simuláciu na platforme PLC. Na simuláciu bol v PLC vytvorený špeciálny program. Pre zobrazenie výstupu PLC bol v prostredí Mosaic použitý nástroj GraphMaker. Ten dovoľuje export údajov v textovej podobe, takže nie je problém s prenesiteľnosťou dát do Matlabu. Výsledky simulácií sú uvedené na obrázkoch 2.10 a 2.9. Ako je vidieť z výsledku simulácií obidva modely produkujú rovnaké výsledky na rovnaké vstupné parametre. Možné odchylky modelu počítanom na platforme PLC môžu byť spôsobené odchylkami v čase otočky T_c, rozoberaného v sekcii implementácie. Konečné hodnoty sú v poriadku. Tie sú z hľadiska diagnostiky na PLC zaujímavejšie, keďže dynamika systému je rýchla a nie je ju možné kontrolovať vo veľkom rozsahu, kôli reletívne veľkým časom T_c.



Obr. 2.9: Porovnanie výstupu otáčok modelu motora



Obr. 2.10: Porovnanie výstupu prúdu modelu motora

2.4 Nepriamy výmenník tepla

Naporoti predchádzajúcim predstaveným modelom sa v tomto prípade bude uvažovať sústava s pomalou dynamikou. V tomto prípade sa bude uvažovať nepriamy výmenník tepla. Pred samotným popisom je nutné dodať, že pri popise modelu sa uvažujú priemerné teploty, takže sa predpokladá ich homogénne rozloženie. To dovolí uvažovať model ako sústavu so sústredenými parametrami a rovnice pre popis sa značne zjednodušia. Popis modelu výmenníka je prebratý z (Noskievič, P., 1999).

Výmenník tepla je zariadenie, v ktorom látka teplejšia predáva teplo látke chladnejšej. Uvažuje sa jednoduchý výmenník tepla uvedeného na obrázku 2.11 bez tepelnej kapacity a s prepážkou na výmenu tepla o ploche S. Uvažuje sa prípad, kedy $T_1 > T_2$, $T_{10} > T_{20}$. Teploty vytekajúcich látok T_{10} a T_{20} sú určené rovnicami

$$\rho_1 c_{P1} V_1 \frac{dT_{10}}{dt} = \Phi_1 - \Phi_{10} - \Phi_{12} \tag{2.8}$$

$$\rho_2 c_{P2} V_2 \frac{dT_{20}}{dt} = \Phi_2 - \Phi_{20} + \Phi_{12} \tag{2.9}$$

kde Φ_i pre $i = \{1, 2\}$ je privedený tepelný tok rovný

$$\Phi_i = \rho_i c_{Pi} T_i q_i \tag{2.10}$$

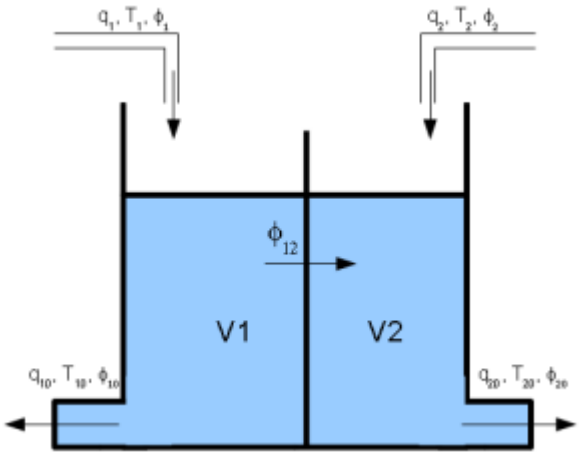
a Φ_{12} je tepelný tok prestupu tepla cez prepážku a spočíta sa ako

$$\Phi_{12} = k_s S (T_{10} - T_{20}) \tag{2.11}$$

Dosadením rovnice 2.10 a 2.11 do rovníc 2.8 a 2.9 a ich následným prepísaním do maticového tvaru sa získa stavový popis tepelného výmenníka v tvare

$$\begin{bmatrix} \dot{T}_{10} \\ \dot{T}_{20} \end{bmatrix} = \begin{bmatrix} -(k_s \frac{S}{\rho_1 c_p 1 V_1} + \frac{q_{10}}{V_1}) & k_s \frac{S}{\rho_1 c_p 1 V_1} \\ k_s \frac{S}{\rho_2 c_p 2 V_2} & -(k_s \frac{S}{\rho_2 c_p 2 V_2} + \frac{q_{20}}{V_2}) \end{bmatrix} \cdot \begin{bmatrix} T_{10} \\ T_{20} \end{bmatrix} + \begin{bmatrix} \frac{q_1}{V_1} & 0 \\ 0 & \frac{q_2}{V_2} \end{bmatrix} \cdot \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \tag{2.12}$$

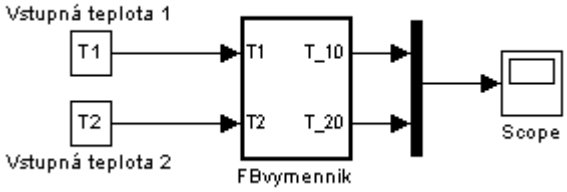
$$\boldsymbol{y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} T_{10} \\ T_{20} \end{bmatrix} \tag{2.13}$$



Obr. 2.11: Nepriamy výmenník tepla

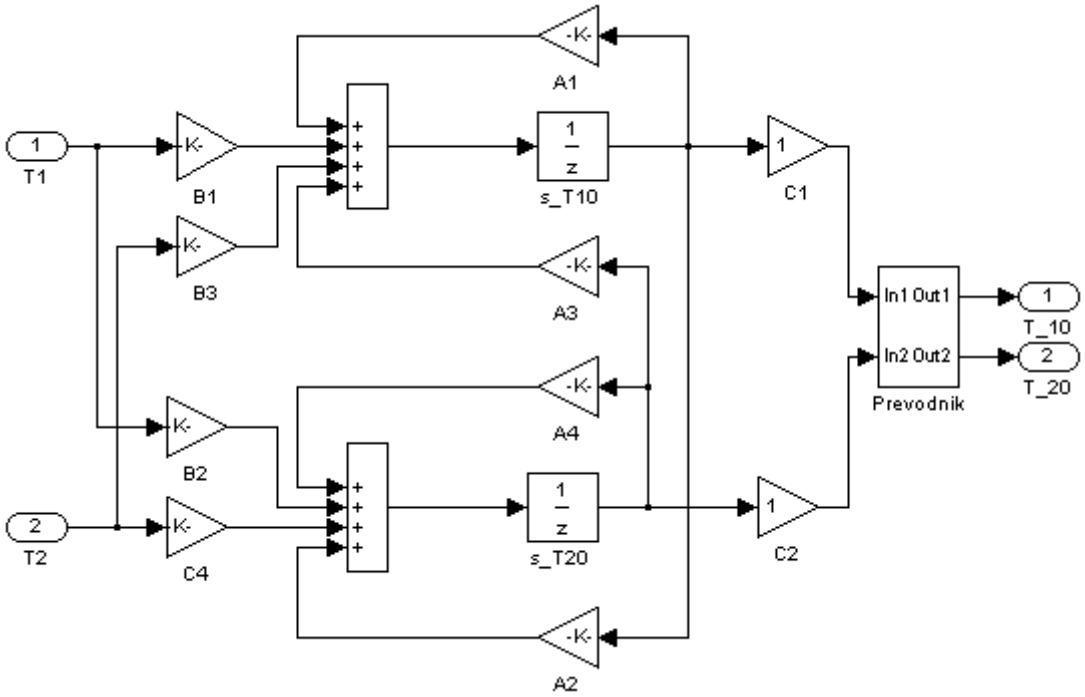
2.4.1 Popis implementácie

Pre simuláciu a zostavenie stavového popisu boli využité parametre uvedené v tabuľke A.1 v prílohe A.4. Postup pri aplikácii modelu do PLC je analogický s predchádzajúcima dvoma v sekciách 2.2.1 a 2.3.1. V prvom kroku bola prevedená diskretizácia stavového popisu modelu. Znova bola použitá metóda diskretizácie *ZOH* s periódou vzorkovania $T_s = 5s$. Vzorkovacia perióda s takou dĺžkou je dostačujúca, pretože sa jedná o pomalý systém. Pre simuláciu bola vytvorená simulačná schéma uvedená na obrázku 2.13 z ktorej bol následne vytvorený funkčný blok. V tomto prípade bola oproti simulačnej schéme elektromotora uvedenej na obrázku 2.7 vytvorená schéma s jednotlivými stavami, z dôvodu zistenia rozdielov v generovanom kóde, oproti schéme s maticovými parametrami. Ako bude vidieť vo výpise kódu, rozdielov v algoritme veľa nebude. Označenie zosilnení `A1,B1` až `A4,B4` korešponduje s prvkami stavovej matice diskrétneho popisu tohoto systému taktiež uvedenej v prílohe A.4.



Obr. 2.12: Simulačná schéma funkčného bloku tepelného výmenníku

Prvky sú označené systémom, akým používa matlab na indexovanie matíc. To znamená od začiatku stĺpca po koniec, následne druhý stĺpec od začiatku po koniec atď.. Schéma s funkčným blokom využitá na simuláciu a na vygenerovanie kódu funkčného bloku je uvedená na obrázku 2.12. Po vygenerovaní kódu (spôsobom ako bol popísaný v predchádzajúcich dvoch príkladoch) z neho boli odstránené nekompatibility ktoré vznikli pri imlementovaní do prostredia Mosaic. Pre odstránenie bol použitý postprocesor vytvorený práve pre tieto účely, ktorý bol implementovaný do Matlabu a vyvoláva sa funkciou `precode()`. Popis tohoto pomocného nástroja je uvedený v sekcii 2.7 tejto kapitoly.



Obr. 2.13: Simulačná schéma nepriameho tepelného výmenníku

2.4.2 Vygenerovaný kód

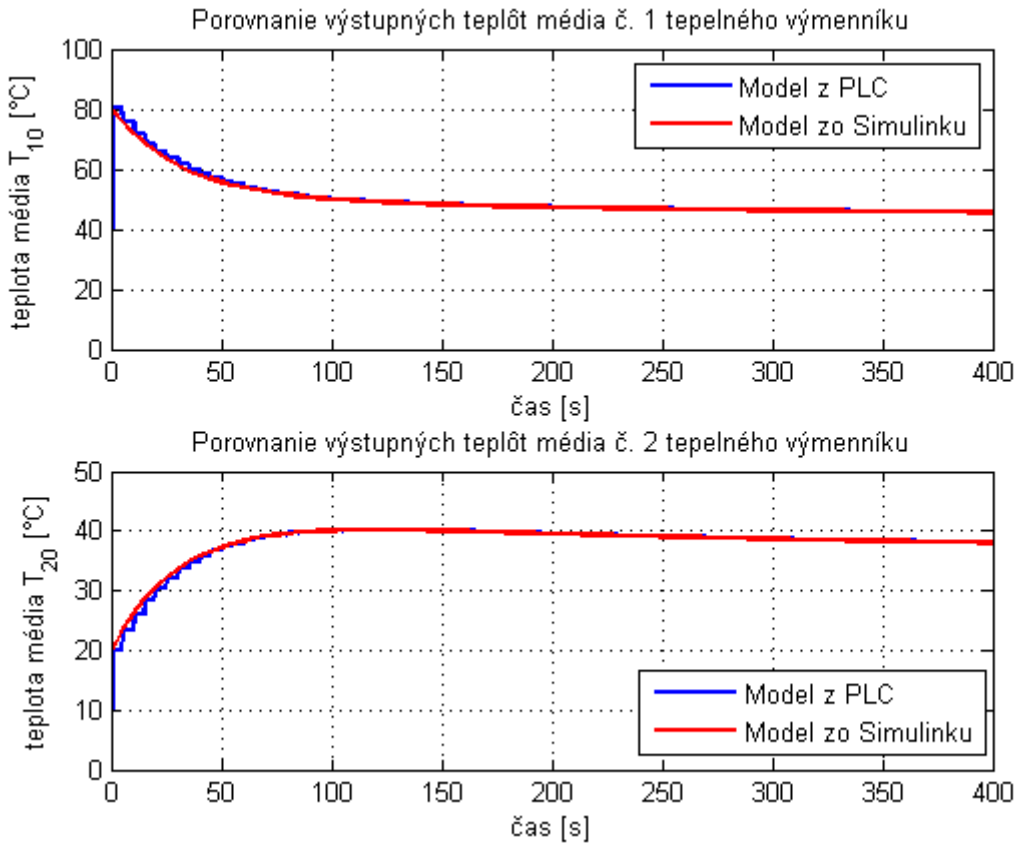
Nastavenia PLC Codera boli rovnaké ako v predchádzajúcich dvoch príkladoch. Vygenerovaný kód je uvedený nižšie. Pre tento prípad bola využitá schéma, v ktorej sa nevyužívali ako parametre matice, takže bola vymodelovaná „krok za krokom“. Výpisy kódu sú uvedené nižsie a bez vygenerovaného komentára.

```
----------------------- Výpis kódu funkčného bloku tepelného výmenníka ------------------------
FUNCTION_BLOCK FBvymennik
VAR_INPUT
    ssMethodType: SINT;
    T1: LREAL;
    T2: LREAL;
END_VAR
VAR_OUTPUT
    T_10: LREAL;
    T_20: LREAL;
END_VAR
VAR
    s_T10_DSTATE: LREAL;
    s_T20_DSTATE: LREAL;
END_VAR
VAR_TEMP
    rtb_A2: LREAL;
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:
        s_T10_DSTATE := 353.15;
        s_T20_DSTATE := 293.15;
    SS_OUTPUT:
        T_10 := s_T10_DSTATE - 273.15;
        T_20 := s_T20_DSTATE - 273.15;
        rtb_A2:= 0.054606712928954478 * s_T10_DSTATE;

        s_T10_DSTATE := (((0.91051425098448635 * s_T10_DSTATE) + (0.015902986074523631 * T1)) +
                        (0.0009820858867772642 * T2)) + (0.072600677054212664 * s_T20_DSTATE);

        s_T20_DSTATE := (((0.91975211970277493 * s_T20_DSTATE) + (0.00046900154765099365 * T1)) +
                        (0.025172165820619541 * T2)) + rtb_A2;
END_CASE;
END_FUNCTION_BLOCK
-------------------------------------------------------------------------------------------------
```

Pri porovnaní kódu funkčného bloku tepelného výmenníku a funkčného bloku elektromotora je vidieť v podobnostiach kódu že rozdiely pri vytváraní algoritmu nie sú až tak veľké. Prvým rozdielom je definícia stavov. Keďže na vygenerovanie bola použitá podrobne rozkreslená schéma a nie všeobecná schéma s maticovými parametrami, namiesto stavov definovaných v poli boli využité dve premenné `s_T10_DSTATE` a `s_T20_DSTATE`. Ďalšia pomocná premenná je `rtb_A2` ktorá označuje spätnú väzbu od stavu `T_10` do vstupu druhého oneskorovacieho člena. Je to pomocný medzivýpočet, ktorý sa pri výpočte stavu `T_20` pripočíta cez pamäťovú premennú `s_T20_DSTATE`. Ostatné premenné a konštanty vychádzajú z parametrov systému definovaných v A.4. Definícia globálnych premenných a konštánt bola uvedená pri modele elektromotora v sekcii 2.3.2.

2.4.3 Simulácia a porovnanie

Simulácia na platforme PLC bola prevádzaná na softvérovom automate, ktorý je súčasťou vývojového nástroja Mosaic. Parametre pre simuláciu boli nastavené podľa tabuľky A.1, teda počiatočné teploty $T_{10} = 20°C$ a $T_{20} = 80°C$. Po celú dobu simulácie boli uvažované konštanté vstupné teploty médií $T_1 = 20°C$ a $T_2 = 80°C$. Tak ako v predchádzajúcich príkladoch pre zozbieranie dát z modelu počítaného v PLC bol použitý nástroj WebMaker. Pre tieto potreby bolo potrebné v PLC doprogramovať jednoduchý obslužný program, ktorý po inicializácii merania nastavil funkčný blok výmenníka na požadované parametre a následne každých s periódou $T_s = 5s$ spočítaval novú hodnotu výstupu modelu výmenníka. Pre zabezpečenie periódy výpočtu boli využité hodiny reálneho času nachádzajúce sa v PLC, v prípade TECOMATU bol využitý register `S20`, slúžiaci na zachytávanie nábežných hrán od obvodu reálneho času PLC. Podrobný popis rozloženia jednotlivých bitov je k dohľadaniu v (Teco a.s., 2007*b*). Výsledky simulácií sú uvedené na obrázku 2.14.



Obr. 2.14: Porovnanie výstupov simulácie v PLC a v Simulinku

Zaujímavosťou, ktorou disponuje PLC Coder, je vygenerovanie fukčného bloku `TestBench`. Ide o funkciu, ktorá dokáže skontrolovať správnosť výpočtov vygenerovaného modelu. Funkčný blok `TestBench` v sebe implementuje instanciu funkčného bloku modelu, sadu dopredu vypočítaných výstupných hodnôt, ktoré má model generovať a sadu vstupných signálov pre ktoré bude výstup generovaný. Spustením funkčného bloku sa porovná hodnota spočítaná na PLC s hodnotou dopredu vypočítanú zo Simulinku. Ak je rozdiel generovaných hodôt väčší než 10^{-5} testovací blok vráti hodnotu testovacej premennej overenia modelu `testVerify = false`. To znamená že model pre daný cyklus nedáva presné výsledky. V opačnom prípade, ak model odpovedá zadanému kritériu presnosti, je vrátená hodnota `testVerify = true`. Výpis funkcie je uvedený v prílohe A.3.

2.5 Riadiaci modul práčky

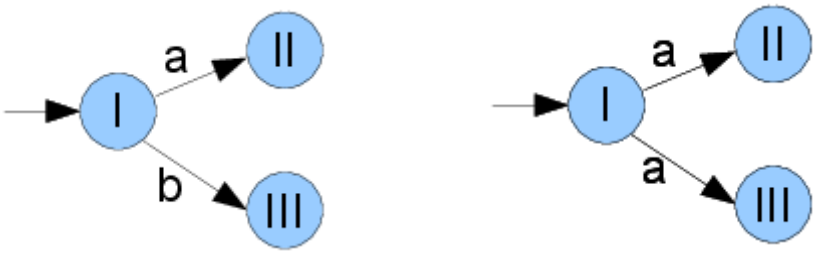
Naproti predchádzajúcim modelom, kde sa uvažoval sústava spojitého systému, ktorá bola zdiskretizovaná, v tejto časti sa bude uvažovať systém riadený diskrétnymi udalosťami. Takzvaný *Systém diskrétnych udalostí*, v literatúrach často označovaný anglickým názvom *Discret Event System*. Jedná sa o systémy s diskrétnymi stavami, ktoré sú riadené udalosťami. Vývoj stavov je priamo závislý na asynchrónne sa vyskytujúcimi diskrétnymi udalosťami v čase (Wikipedia, 2011*a*). Pre predstavu je možné uvažovať napríklad model práčky. Detailný popis modelu by bol zbytočne nad rozsah demonštračného príkladu, takže budú uvedené len subsystémý napríklad programového voliča a riadenia motora pri praní. Kompletná schéma je s podblokami je uvedená v prílohe A.5. Pre modelovanie systému tohoto typu bol využitý nástroj *State Flow Chart* ktorý je v Matlabe implementovaný a bloky vytvorené v tomto nástroji je možné použiť a simulovať v Simulinku.

2.5.1 Popis modelu

Pre detailnú demonštráciu boli zvolené dve najprehľadnejšie časti subsystému. Programový volič, ktorý tvorí rozhranie medzi užívateľom a ostatnou riadiacou logikou v práčke. Ako druhý subsystém riadenia motora, ktorý riadi otáčky a smer motora pri praní. Oba subsystémy boli namodelované ako stavový automat.

Programový volič, ktorý bol navrhnutý uvažuje výber 4 pevne daných programov pre pranie, odštartovanie programu, reset programu a zastavenie programu v priebehu

prania. Všetky tieto akcie predstavujú definované udalosti, ktoré môžu v systéme nastať. Namodelovaný automat je na obrázku 2.16. Automat má 7 stavov, neblokujúci, deterministický. Neblokujúci automat znamená, že z každého stavu je možné sa dostať do počiatočného stavu, v tomto prípade je to stav označný menom `IDE`. To že sa jedná o počiatočný stav hovorí aj šípka s bodkou priradená tomuto stavu. Deterministický automat, je taký, pre ktorého všetky stavy platí, že udalosť definovaná nad daným stavom spôsobí vždy jednoznačný prechod do stavu iného. To znamená že nie je možné aby jedna udalosť definovaná nad stavom ktorá sa vyskytne mohla spôsobiť prechody do dvoch stavov naráz. Pre lepšiu predstavu je problém determinizmu uvedený na obrázku 2.15.



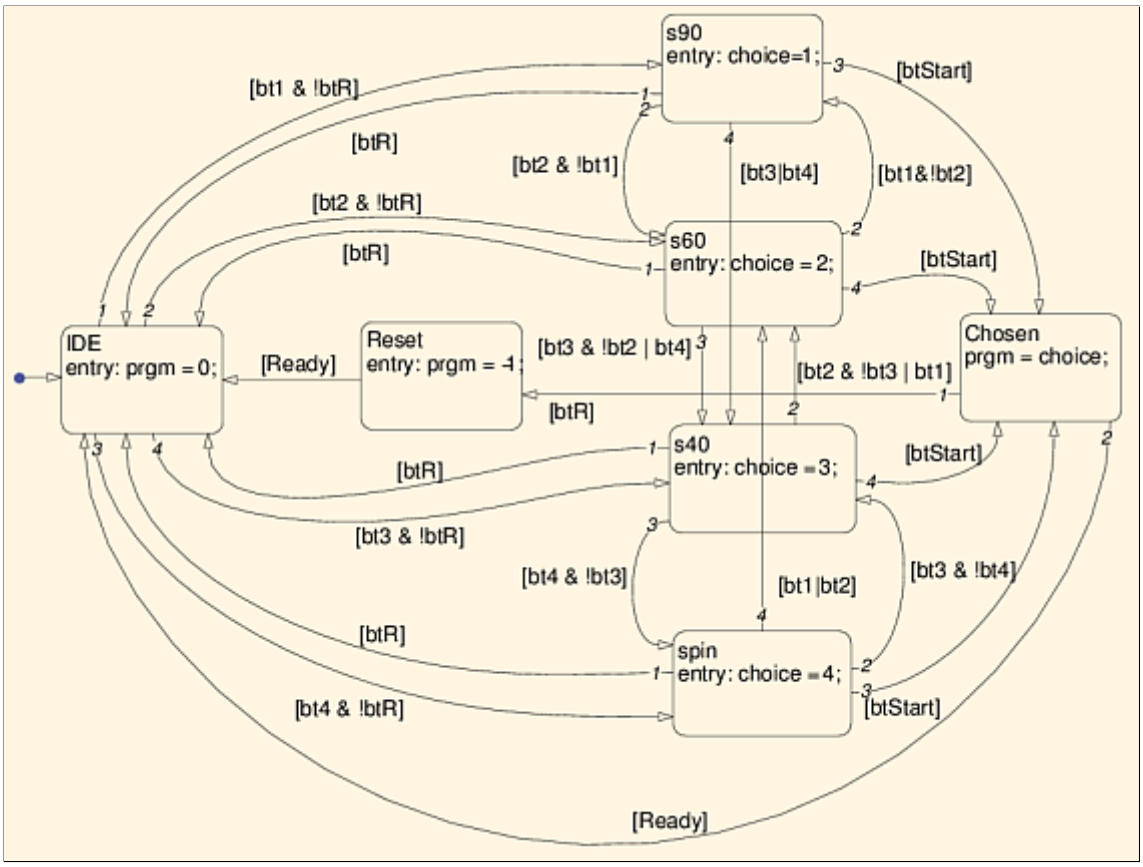
Obr. 2.15: Determinizmus automatov (vľavo deterministický)

Pre pochopenie navrhnutého systému sa popíše napríklad stlačenie tlačidla pre voľbu prvého programu[3]. Automat sa týmto dostane zo stavu `IDE` do stavu označeného `s90`. Pri vstupe systému do tohoto stavu sa nastaví premenná `choice` na hodnotu 1, čo je značené notáciou `entry:`. Nad týmto stavom sú definované ďalšie udalosti `bt2, bt3, bt4, btR, btStart` kotré môžu nastať. Výskytom danej udalosti nad týmto stavom systém prejde do ďalšieho stavu podľa prechodu na ktorom je daná udalosť definovaná. Napríklad výskytom udalosti `btStart` sa systém dostane do stavu `Chosen`, kde sa nastaví hodnota premennej `prgm = 1` a nasledujúcemu bloku sa povie, že sa má začať vykonávať program č. 1. Ak bola voľba programu raz potvrdená pomocou `btStart`, tak odvolať ju je možné len udalosťou `btR`. Aby bolo možné zvoliť nový program, tak od nasledujúcich blokov musí byť potvrdené udalosťou `Ready` že je možné započať voľbu nového programu a to tak, že sa systém dostane znova do počiatočného stavu `IDE`.

Riadiaci blok motora je taktiež namodelovaný stavovým automatom uvedeným na obrázku 2.17. Boli v ňom definované 4 stavy. Prvý stav `Off` slúži ako východzia poloha. Nastavuje sa v ňom pomocná premenná `D`, ktorá slúži k určovaniu smeru motora. Ďalšie dva stavy `OnL` a `OnR` určujú smer ktorým sa bude motor otáčať. Systém zostáva

[3]Činnosť akú má program vykonávať je možné naimplementovať ľubovoľne.

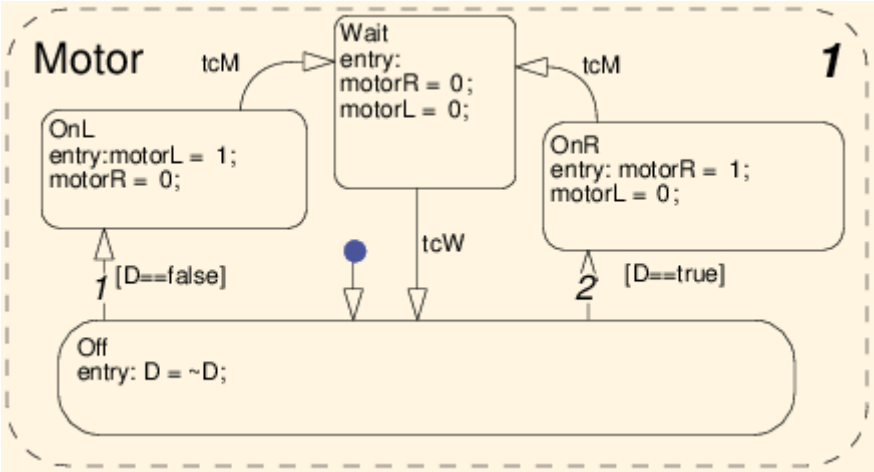
v tomto stave dovtedy kým nepríde spádová hrana externého časového signálu označená ako `tcM`. Signál predstavuje udalosť definovanú nad daným stavom. Pri zaregistrovaní tohoto signálu systém prejde do stavu čakania `W`. V tomto stave sa vynulujú oba riadiace signály `motorR` a `motorL`, ktoré určujú smer akým sa má motor otáčať. Systém sa dostáva zo stavu zaregistrovaním udalosti `tcW` ktorá je definovaná ako spádová hrana externého signálu, ktorý do riadiaceho modulu privádzaný. Opustením stavu `W` sa prejde znova do počiatočného stavu, kde sa neguje informácia o smere motora. To znamená, že ak sa v predchádzajúcom cykle motor točil vpravo, tak v ďalšom cykle bude otáčaný vľavo.



Obr. 2.16: Programový volič práčky namodelovaný automatom

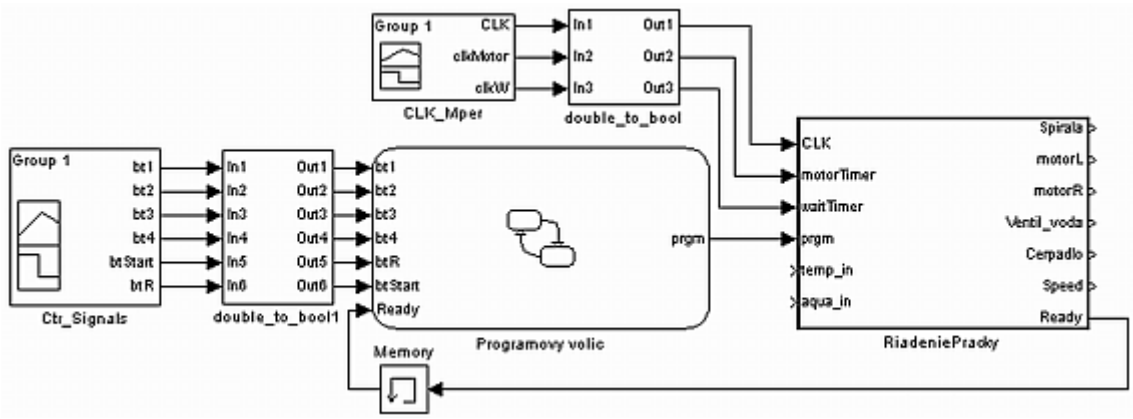
2.5.2 Popis implementácie

Ako bolo uvedené vyššie, na namodelovanie systému diskrétnych udalostí bol využitý nástroj State Flow Chart (SFC). Naproti Simulinku, vytváranie sústavy v tomto nástroji obnáša potrebu znalostí o automatoch a hlavne znalosť syntaxe tohoto nástroja. Pre zoznámenie je vhodná dokumentácia dostupná na webe: (Mathworks, 2011*c*). Schéma



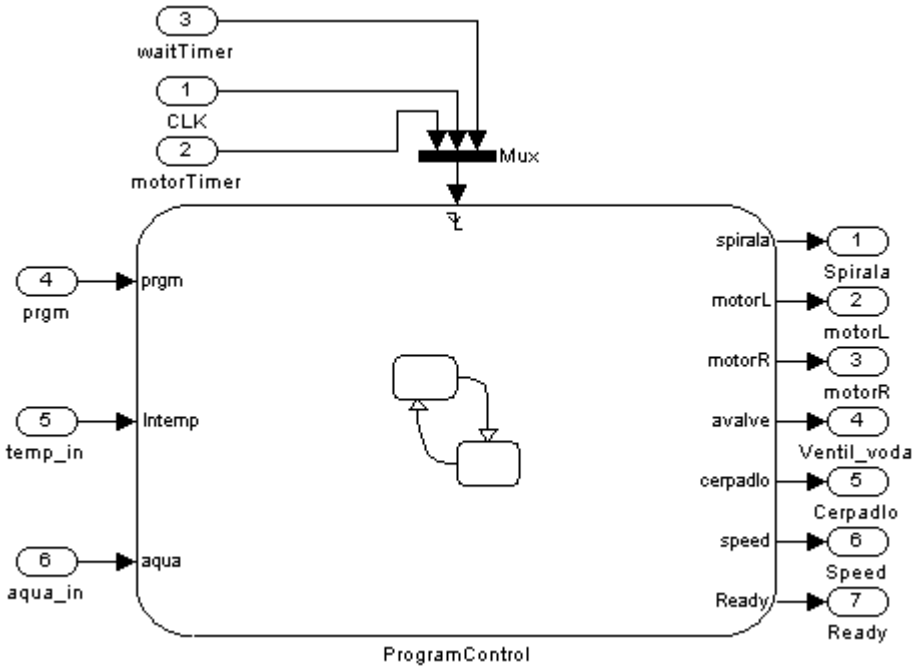
Obr. 2.17: Riadiaci blok motora práčky namodelovaný ako automat

sa pôvodne vytvára v Simulinku. V knižnici je potrebné nájsť prvok `StateFlow` a vložiť ho do simulinku ako klasický prvok. Dvojitým klikom sa otvorí nástroj na modelovanie automatov. Ako prvý krok je dobré definovať rozhranie medzi grafom a simulinkom. To znamená, že je potrebné si nadefinovať vstupy a výstupy bloku. Vstupy sú väčšinou využité na privedenie udalostí, ktoré môžu nad stavom nastať. Výstupy sú už výsledky algoritmu ktorý sa chce automatom dosiahnuť. Je to možné si predstaviť ako funkčný blok. Simuliková schéma použitá na simuláciu je uvedená na obrázku 2.18. Blok programového voliča je priamo SFC blok. Vnútorná štruktúra tohoto bloku bola rozobraná v predchádzajúcej sekcii a je znázornená na obrázku 2.16. Druhý blok, blok riadiacej jednotky produkuje výstupné riadiace signály. Bol vytvorený ako simulinkový subsystém obsahujúci SFC blok vo svojej vnútornej štruktúre. Detail subsystému je uvedený na obrázku 2.19. Zapúzdrenie SFC do simulinkového subsystému bolo vytvorené kôli hodinovým signálom, ktoré sú očakávané ako vstup do SFC bloku.



Obr. 2.18: Schéma riadiaceho modulu práčky vytvorená v simulinku

PLC Coder si nevie poradiť priamo s blokom SFC, ktorý má hodinový signál. To znamená že simulink zablokuje možnosť generovania kódu pre takýto blok. Riešením je pridanie multiplexora ktorý spojí všetky očakávané hodinové signály (v tomto prípade sa jednalo o vyššie rozobraté signály `tcM`, `tcW` a `clc`, ktorým je taktovaný prací cyklus). Spolu s multiplexorom sa blok SFC uzavrie do subsystému a označí sa ako atomická jednotka. Po takejto úprave je možné generovať kód pomocou PLC Codera. Je to dôležitá informácia, ktorá bohužiaľ nie je nikde v manuále pre SFC a PLC Coder nikde uvedená.



Obr. 2.19: Vnútorná štruktúra subsystému riadenia práčky s multiplexorom

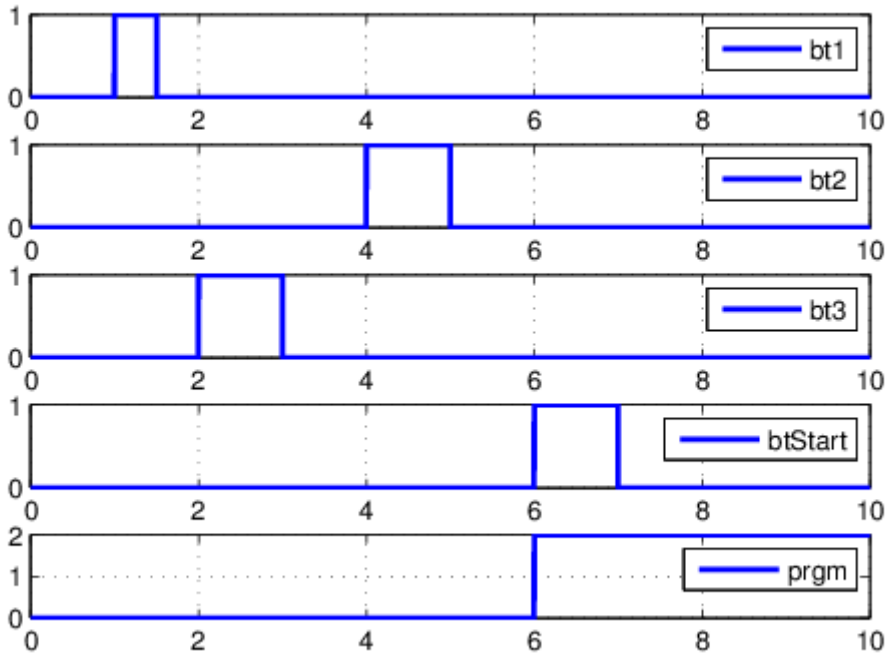
2.5.3 Vygenerovaný kód

Navrhnutý riadiaci modul má dve časti programový volič a riadiaci systém. Kód bol vygenerovaný pre oba prípady. Ukážka vygenerovaného kódu bude uvedená len pre subsystém programového voliča, z dôvodu rozsiahlosti kódu druhého modulu. Kompletný kód má približne 1300 riadkov, takže bude priložený elektronicky ako príloha CD. Ukážka vygenerovaného kódu pre programový volič je v prílohe A.6. Vygenerovaný program je zlinkovaný do `CASE` štruktúry a je riadený podmienkami `IF-ELSIF`. Takýto prístup programovania nie je práve najefektívnejší a hlavne je neprehľadný. Záver z využitia PLC Codera pre

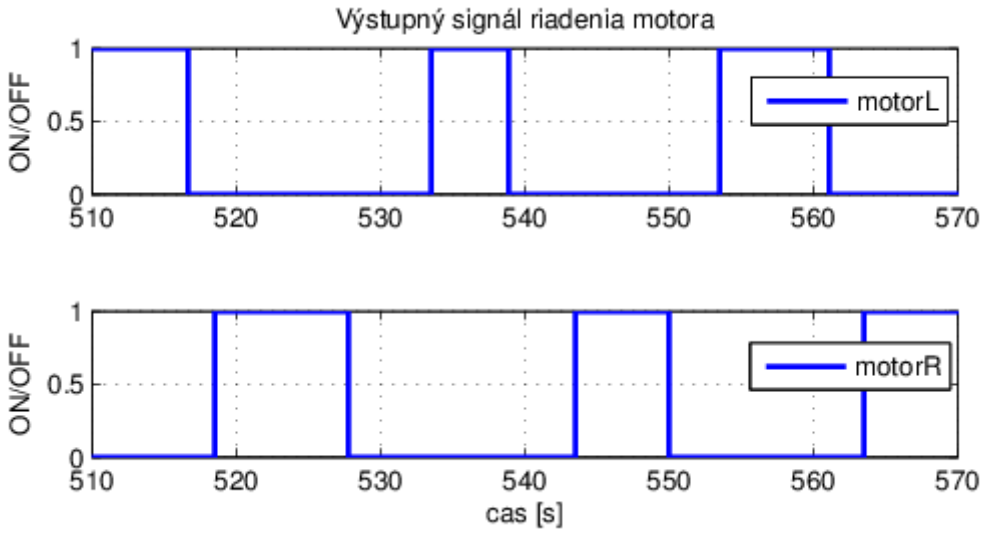
rozsiahlejšie systémy vytvorené v SFC je taký, že kóder je výhodné použiť pre jednouché systémy. Pri rozsiahlých systémoch je kód neprehľadný a s vygenerovanou štruktúrou je pochybnosť o rýchlosti a optimalite kódu. Norma IEC 61131-3 definuje grafické programovacie jazyky, ktorými sa dá daný algoritmus dosiahnuť oveľa efektívnejšie, než je prechod z grafického prostredia do textového. Takže výhliadky pre využitie v systémoch disktrétnych udalostí nie sú veľké.

2.5.4 Simulácia

Na obrázkoch 2.20 a 2.21 je uvedený výstup z blokov popísaných v sekcii 2.5.1. Na obrázky je potrebné nazerať ako na výstup z logického analyzátora. Funkcia programového voliča bola overená testami pre rôzne stavy. Pre priblíženie je uvedená jedna situácia, kedy sa prepína medzi troma stavami a to tlačidlami `bt1`, `bt2` a `bt3`. Ako je vidieť z grafu a zo štruktúry automatu voliča na obrázku 2.16, správanie odpovedá očakávanej funkcii. Výstup `prgm` sa nastaví na hodnotu zvoleného programu až po stlačení tlačidla `btStart`. Program je daný tlačítkom, ktoré bolo stlačené ako posledné. V tomto pípade je to `bt2`. Situácia pre riadenie motora je na obrázku 2.21.



Obr. 2.20: Simulácia funkcie programového voliča



Obr. 2.21: Simulácia funkcie subsystému pre riadenie motora

Po porovnaní výstupu s logickou štrukúrou je vidieť, že riadiaci modul pracuje správne. Očakávaná funkcia bola striedavá komutácia smeru motora oddelená časovým úsekom kedy sa nevykonáva nič. Podľa simulácie namodelovaný automat 2.17 pracuje podľa očakávaní.

2.6 Implementačné problémy

Ako bolo avízované u predchádzajúcich príkladov, po prenesení kódu do vývojového nástroja Mosaic nastávali problémy s kompatibilitou vygenerovaného kódu. Pri pokuse o kompiláciu importovaného súboru začal kompilátor xPRO hlásiť hneď sadu chýb súvisiacich so syntaxou, deklaráciami dátových typov a deklaráciami premenných. Prvotná kompilácia bola znemožnená na chýbajúcej definícii globálnych konštánt definovaných pre riadenie funkcie funkčného bloku, teda konštanty `SS_INITIALIZE`, `SS_OUTPUT`, `SS_UPDATE` atď.. Chyba nastávala aj napriek existujúcej deklarácii[4] týchto konštánt v zdrojovom kóde. Príčina tejto nekompatibility bola zrejmá zo spôsobu generovania jednotlivých blokov v kóde. Normálne usporiadanie blokov pri vytváraní kódu a ktoré je akceptované prekladačom xPRO je sekvenčné. To znamená, že prekladač nedokáže pracovať s kódom

[4]Výpisy deklarácií sú uvedené v predchádzajúcich príkladoch vo výpise kódu pod hlavičkou „`Deklarácia globálnych premenných a konštánt`“ napríklad v kóde RC člena na strane 22.

online-staticky, ale preferuje spôsob prekladu zvrchu nadol. Logicky sa očakáva usporiadanie blokov v poradí:

- deklarácie globálnych premenných a konštánt
- deklarácie nových typov a štruktúr
- deklarácie funkčných blokov a funkcií
- deklarácia programov

V prípade kódu z PLC Codera toto usporiadanie bolo realizované presne v opačnom poradí, tak ako to bolo zámerne uvádzané v príkladoch v predchádzajúcich sekciách tejto kapitoly. Najprv definícia funkčného bloku, potom nové typy a štruktúry a nakoniec definícia globálnych konštánt. Takýto spôsob usporiadania blokov kódu s kombináciou prekladača xPRO malo za následok výpis sady chybových hlášok ktoré sa týkali deklarácii. Premenné, ktoré boli definované na konci kódu a ktoré ešte prekladač nepoznal mali byť použité vo funkčnom bloku definovanom na začiatku kódu, ku ktorého prekladu sa dostal xPRO skôr. Riešením tohoto problému bolo, že po vygenerovaní súboru s kódom sa urobilo kompletné preusporiadanie blokov do podoby akceptovateľnej prekladačom, teda vo vyššie uvedenom poradí.

Po odstránení tohoto problému sa na povrch dostali nové nekompatibility medzi PLC Coderom generovaným kódom a prekladačom xPRO. Ďalšou z nich bolo chybové hlásenie o nekompatibilite typov hodnôt priradzovaným k premenným. Uvedená nekompatibilita bola spôsobená viacmenej PLC Coderom, ktorý, zdá sa, nerešpektuje zápis inicializačnej hodnoty reálnym dátovým typom, teda typov `ANY_REAL`. Príklad takéhoto priradenia pri inicializácii je vo výpise nižšie. Do premennej deklarovanej ako reálne číslo `LREAL` sa priradzuje hodnota celočíselného typu. Prekladač takéto priradenie spracuje ako nekompatibilitu medzi typami a vyhlási chybu. Ten istý problém sa objavoval aj pri nekompatibilite typov v podmienkových výrazoch. To znamená, že ak sa chcela kvantitatívne porovanať celočíselná hodnota s hodnotou reálnou, xPRO taktiež hlásil nekompatibilitu typov. Chyba musela byť odstránená pre všetky premenné deklarované v kóde tak, že sa priradením desatinnej čiarky hodnote definovala ako reálna. Po tejto úprave xPRO spracoval kód bez ďalších problémov.

```
------------------------------ Nekompatibilita priradzovaných typov ------------------------------
VAR
    s_1_DSTATE: LREAL;
END_VAR
==========================================================================================
    s_1_DSTATE := 0;            <----- nesprávne priradenie celočíselnej hodnoty do premennej LREAL
==========================================================================================
    s_1_DSTATE := 0.0;          <-----        správne priradenie reálnej hodnoty do premennej LREAL
==========================================================================================
IF s_1_DSTATE >= 3 THEN         <-----    nesprávne porovnávanie typu LREAL s celočíselnou hodnotou
==========================================================================================
IF s_1_DSTATE >= 3.0 THEN       <-----      správne porovnávanie typu LREAL s celočíselnou hodnotou
------------------------------------------------------------------------------------------
```

Odstránením vyššie popísaných problémov s prekladom boli programy popísané v sekciách druhej kapitoly pripravené k použitiu. No pri dlhšom testovaní generovaného kódu pre rôznorodé systémy sa narazilo na ďalšie nezrovnalosti, ktoré prekladač nevedel spracovať. Jednou z nich bola inicializácia prvkov poľa na určitú hodnotu. Vzhľadom na normou definovanú inicializáciu v takýchto prípadoch je zjavne chyba na strane prekladača xPRO. Pri inicializácii poľa na počiatočné hodnoty prekladač očakáva uzavretie prvkov do hranatých zátvoriek, tak ako je to uvedené na výpise problémovej sekcie programu nižšie. Podľa normy však takéto opatrenie nie je nutné a prvky poľa sa zapíšu jednoducho za sebou oddelené čiarkou.

```
------------------------------ Nekompatibilita pri deklarácii poľa -------------------------------
VAR
    ==========================================================================================
    pole: ARRAY [0..5] OF LREAL := 0.1,0.2,0.3,0.4,0.5;        <-----  deklarácia poľa podľa normy
    ==========================================================================================
    pole: ARRAY [0..5] OF LREAL := [0.1,0.2,0.3,0.4,0.5];      <--- deklarácia očakávaná v Mosaicu
    ==========================================================================================
END_VAR
------------------------------------------------------------------------------------------
```

Jednou z ďalších zistených nekompatibilít bola indexácia poľa. Prekladač xPRO očakáva, že premenná ktorá bude použitá pre indexovanie prvkov v poli je výhradne typu `INT`, `SINT`,`UINT`,`USINT`. PLC Coder generuje index poľa deklarovaný dátovým typom `DINT`. Tam nastáva znova nekompatibilita typov a prekladač xPRO vyhlási chybu indexu. Riešením je zmena deklaračného typu danej indexovej premennej na typ `INT` alebo na iný z vyššie vymenovaných podporovaných prekladačom. V tomto prípade bolo zvolené riešenie konverzie typu. To znamená že premnenná ktorá je deklarovaná ako `DINT` bude pri indexácii prekonvertovaná na `INT`. Výpis ukážky nekompatibility je uvedený nižšie.

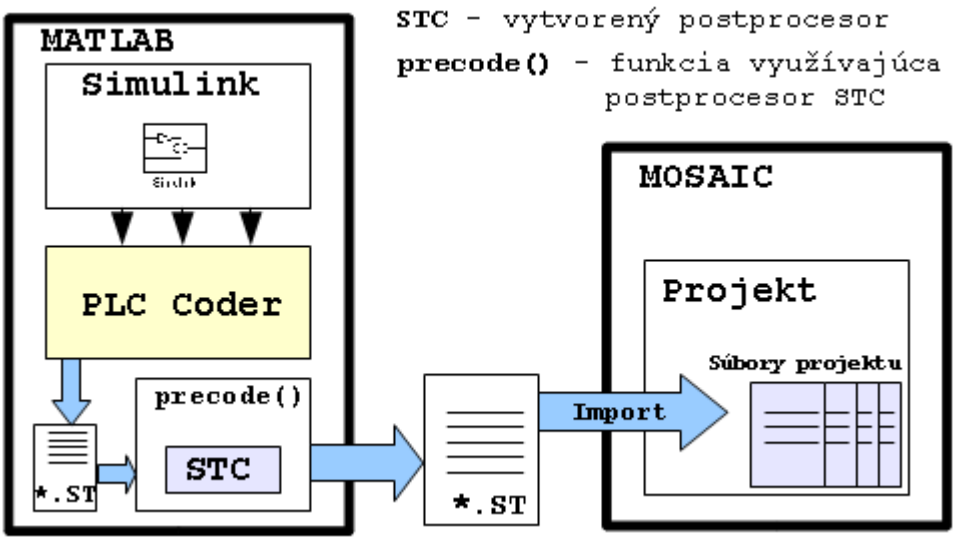
```
---------------------------- Nekompatibilita pri deklarácii indexu poľa ---------------------------
VAR
    ==========================================================================================
    indexPola : DINT;                        <-----  deklarácia z PLC Codera nekompatibilná s Mosaicom
    ==========================================================================================
    indexPola : INT;                        <-----  deklarácia očakávaná v Mosaicu
    ==========================================================================================
    pole[DINT_TO_INT(indexPola)]            <-----  konverzia typu pri použití v poli
    ==========================================================================================

END_VAR
------------------------------------------------------------------------------------------------
```

Poslednou zo zistených nekompatibilít, ktorá znova hraje v prospech prekladača xPRO je syntaktická chyba pri definovaní štruktúry. Za ukončovacím znakom `END_STRUCT` sa podľa normy očakáva bodkočiarka. Tá v prípade generovaného kódu chýba, takže je nutné ju doplniť vo všetkých definovaných štruktúrach. To znamená, že každé koncové kľúčové slovo štruktúry bude ukončené takto: `END_STRUCT;`

2.7 Postprocesor

Pre odstránenie všetkých zistených chýb bol vytvorený postprocesor pre vygenerovaný súbor s príponou .ST, ktorý urobí kontrolu vygenerovaného kódu na vyššie popísané nekompatibility a odstráni ich. Výstupom postrocesora je upravený kód, ktorý rieši nezrovnalosti medzi PLC Coderom, Mosaicom a normou IEC/EN 61131-3 tak, aby sa vygenerovaný kód stal kompatibilný s prostredím Mosaic. Posprocesor bol vytvorený v programovacom jazyku *Java*, takže týmto je nezávislý na platforme z ktorej sa spúšťa. Idea vytvorenia postprocesora nezávislého na platforme a ako samostatne spustiteľného kódu je možnosť jeho integrácie do obidvoch prostredí. Matlab je dostupný pre všetky platformy, takže sa chcela vytvoriť verzia, ktorá by problém medzi platformami riešila. Mosaic je dostupný len na platformu Windows. To že postprocesor je vytvorený ako samostatne spustiteľný program mu dáva možnosti byť volaný z jedného alebo druhúho prostredia. Jednoduchšia integrácia je v prostredí Matlab. Ten umožňuje spúšťanie externých programov z príkazoveho riadka systému, takzvaného *SHELL*. V Matlabe bola vytvorená funkcia s názvom `precode('inputName','outputName')`, ktorá tento postprocesor využíva. Vstupom funkcie je cesta k súboru s menom špecifikovaným v reťazci `inputName`, ktorý má byť upravený. Výstupom je upravený súbor s menom špecifikovaným v parametri `outputName` funkcie. Upravený súbor sa ukladá do aktuálneho pracovného adresára

nastaveného v Matlabe. Volanie z prostredia Mosaic je pre užívateľa neprípustné. No každopádne existuje možnosť integrácie systémovým vývojárom. Prenos vygenerovaného programu medzi Matlabom a Mosaicom sa tak oproti plánovanému spôsobu komunikácie, uvedeného na obrázku 2.1 zmení po použití postprocesora na spôsob uvedený na obrázku 2.22.



Obr. 2.22: Použitý spôsob prenášania kódu medzi Matlabom a Mosaicom

Kapitola 3

Diagnostika systému s využitím modelu

Ako bolo už v predchádzajúcich kapitolách naznačené, využitie modelu v automatizačnej technike je prínosné vo všetkých smeroch. Od návrhu, cez testovanie až po ad-hoc diagnostiku. Z aplikačného hľadiska a prínosnosti čím ďalej atrakttívna diagnostika prevádzaná online. Výpočtové prostriedky dnešných automatov sú na vysokej úrovni. Náročnosť riadiacich algoritmov sa v samotnej podstate nezvyšuje až tak rapídne aby bol využitý celý potenciál automatu. Preto je možné zvyšný potenciál investovať do prídavnej funkcie - paralelnej diagnostiky procesu. Tá bude vykonávaná simultánne s riadiacim algoritmom: (Šmejkal, L. a Pohl, T., 2009). Metódy pre riešenie problému diagnostiky sú v dnešnej dobe ešte vo vývoji a do priemyslu sa pretlačujú len pomaly. No napriek tomu existuje niekoľko overených a použiteľných metód v ktorých využitie modelu hrá rozhodujúcu úlohu. Model je ponímaný jednak v zmysle presného matematického popisu a vytvára vzor, nominálnu sústavu pre výpočet očakávaných výsledkov. Táto metodika bude popísaná nižšie v sekcii 3.1. No model môže byť ponímaný tiež ako modelový svet, prostredie, v ktorom je automat nasadený. Pre spoluprácu s ostatnými distribuovanými systémami v riadiacom algoritme zahŕňa ontológiu. To znamená že má popísané vzťahy medzi jednotlivými komponentami systému nachádzajúcich sa v pracovnom prostredí. Táto metodika zasahujúca už na pôdu umelej inteligencie bude pre inšpiráciu popísaná v sekcii 3.2.

3.1 Diagnostika založená na synchrónnej simulácii

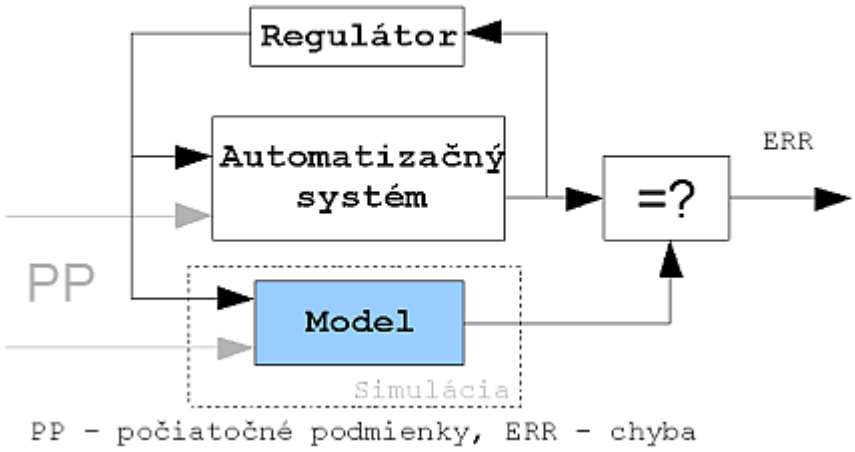
Mechatronický systém je sieť vzájomne pospájaných senzorov a aktuátorov. Mechatronické systémy nasadené v priemysle sú riadené vo viacerých úrovniach. Najnižšia úroveň pre riadenie fyzických akcií, vyššia úroveň, ktorá má za úlohu sledovanie lokálneho cieľa (každá časť mechatronického systému slúži k istému účelu, ktorého dosiahnutie sa sleduje) a najvyšia úroveň, ktorá sleduje globálny cieľ (výsledok práce). Pre správnu funkciu a s tým spojenú efektívnosť je potrebná diagnostika odchyliek a chýb v systéme už na najnižšej úrovni. Na vyššej úrovni je vhodná analýza týchto deviácií a ich následná oprava. Popísaná metodika má za úlohu odhaliť vzniknutú chybu, popr. klasifikovať ju, ale nerieši jej odstránenie. Metodika je popísaná v (Kain, S. et al., 2010) a popisuje spôsob diagnostiky a dosiahnuté výsledky.

Na diagnostiku je využitá synchrónna simulácia analytického modelu riadenej sústavy. Metodika kombinuje využitie *Hardware In the Loop* (ďalej HIL) a využitie simulácie modelu na ktorý je aplikované modelové riadenie (model sústavy je riadený vlastným regulátorom). Pre kompletnosť je dobré uviesť čo sa myslí pod pojmom HIL. HIL je metóda ktorá sa využíva pri testovaní funkčnosti vyvinutého hardvérového zariadenia. Zariadenie nie je aplikovaná priamo na sústavu pre ktorú bolo vyrobené, ale jeho funkčnosť sa skúša na modele sústavy. Prípadné chyby ktoré navrhnutý systém obsahuje tak nespôsobia škodu na zariadení.

Diagnostika založená na princípe využitia simultánne spočítavaného modelu dokáže odhaliť dva druhy prípadných chýb vyskytujúcich sa v systéme. Prvý druh je riaditeľná chyba. Zjednodušene sa dá vnímať ako chyba, ktorú je možné kompenzovať pomocou regulátora tak, že výstup sústavy zostane na sledovanej referencii. Druhá je neriaditeľná chyba, čo znamená že chyba tohoto druhu je pre regulátor „neviditeľná“. Aj napriek jej výskytu v systéme, regulátor o nej nemá informáciu a tým neaplikuje na systém žiadnu akciu na jej kompenzáciu.

Idea diagnostiky chýb je založená na porovnávaní výstupov reálnej sústavy a výstupov modelu. Model sústavy by mal byť čo najpresnejší aspoň v pracovnom bode v ktorom sa bude využívať. Dôvodom je, že model bude považovaný za nominálnu sústavu oproti ktorej sa bude porovnávať výstup reálnej sústavy. To znamená, že simulačný model bude emulovať správanie sa systému. Za predpokladu že obe sústavy (model a reálna sústava) budú inicializované na rovnaké počiatočné podmienky. Ak sústava a modelu budú budené rovnakým vstupom, bude model a reálna sústava vykazovať rovnaké hodnoty výstupu. V zmysle odchyliek medzi simuláciou a reálnym systémom môže byť detekovaná porucha.

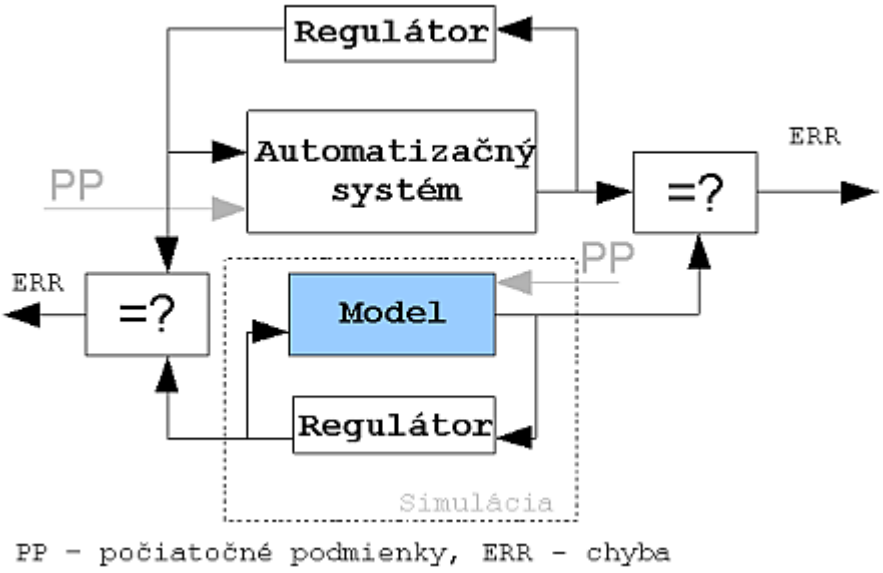
Z myšlienky paralelného porovnávania výstupov vychádza potom podmienka, že model musí byť spočítavaný paralelne s vývojom reálnej sústavy. Princíp využitia simultánne spočítavaného modelu v otvorenej slučke je na obrázku 3.1.



Obr. 3.1: Diagnostika za pomoci modelu v otvorenej slučke

Ak sa v takomto systéme objaví riaditeľná chyba, zareaguje regulátor sústavy. Regulátor sa bude snažiť udržať svoj výstup na vhodnej hodnote tak, aby udržal výstup sústavy na požadovenej hodnote. Riaditeľná chyba je tak kompenzovaná zvýšením hodnoty výstupu regulátora. Okrem toho že je tento výstup aplikovaný na riadenú sústavu, je paralelne aplikovaný aj na nominálny model. Nominálny model vznik chyby neuvažuje, takže aplikáciou rovnakého akčného zásahu vznikne rozdiel medzi výstupom modelu a výstupom reálnej sústavy. Porovnaním sa zistí, že v systéme sa deje niečo, čo nie je v súlade s modelovou situáciou. Na obrázku 3.1 je táto situácia naznačená ako `ERR`.
V prípade výskytu neriaditeľnej chyby v systéme je situácia odlišná. Výstup regulátora, naproti predchádzajúcemu prípadu, nebude ovplyvnený. Neriaditeľná chyba v systéme nie je modelovaná a pôsobí len na reálnu sústavu. Pre regulátor je transparentná, takže výstup regulátora nebude ovplyvnený. Ovplyvnený nebude ani model, no napriek tomu je výsledkom deviácia hodnoty nominálneho výstupu modelu od reálneho výstupu sústavy. To diagnostikuje chybu v systéme. Ohodnotením závažnosti výskytu a podaním do vyššej vrstvy môže byť realizovaná komplexná diagnostika.
Pri využití modelu v otvorenej slučke je možné detekovať oba druhy chýb. Menší problém nastáva pri metodike podľa obrázku 3.1 nastáva vtedy, ak by bolo potrebné chyby z nejakého dôvodu klasifikovať (napr. pre potreby vyššej diagnostickej vrstvy). To daná

metodika nie je schopná, ale s malými obmenami nie je problém dosiahúť jednoduchého klasifikačného mechanizmu. Spomínanou úravou je jednouché využitie modelu ale v uzavretej slučke. To znamená použije sa kompletný model sústavy vrátane riadenia. To znamená, že model je riadený vlastným regulátorom ktorý je zhodný s reálnym regulátorom. Principiálne usporiadanie je na obrázku 3.2.



Obr. 3.2: Diagnostika pomocou modelu s vlastným regulátorom

Na systém a na model je aplikovaná rovnaká referencia. Oba nezávislé regulátory sa snažia o sledovanie referencie. Ak na systém začne pôsobiť riaditeľná chyba, regulátor sa snaží chybu kompenzovať nastavením vhodnej hodoty akčného zásahu na jeho výstupe. Model informáciou o chybe nedisponuje takže regulátor riadiaci model bude držať na svojom výstupe vždy hodnotu odpovedajúcu akčnému zásahu pre dosiahnutie referenčnej hodnoty. Na reálnej sústave bude sa bude regulátor snažiť chybu kompenzovať. To znamená že výstup regulátora sa nastaví na inú hodnotu. Porovnaním výstupov oboch regulátorov sa zistí nezrovnalosť je to znamenie výskytu riaditeľnej chyby (na obrázku 3.2 blok porovnania vľavo). Výstupy systému sa líšiť nebudú. Naopak pri situácii, kedy sa v systéme objaví neriaditeľná chyba sa pri porovnaní akčných zásahov regulátorov nezistí nič. Regulátor fyzického systému nemá informáciu o jej výskyte a tým nemá možnosť jej kompenzácie. Model nemá taktiež žiadnu informáciu o chybe, takže nie je čo kompenzovať. Napriek tomu sa porovnaním výstupov zistí deviácia, ktorej výskyt implikuje neriaditeľnú chybu. Ukážka využitia je na príklade popísanom v (Kain, S. et al., 2010).

3.2 Priemyselný systém s modelom prostredia

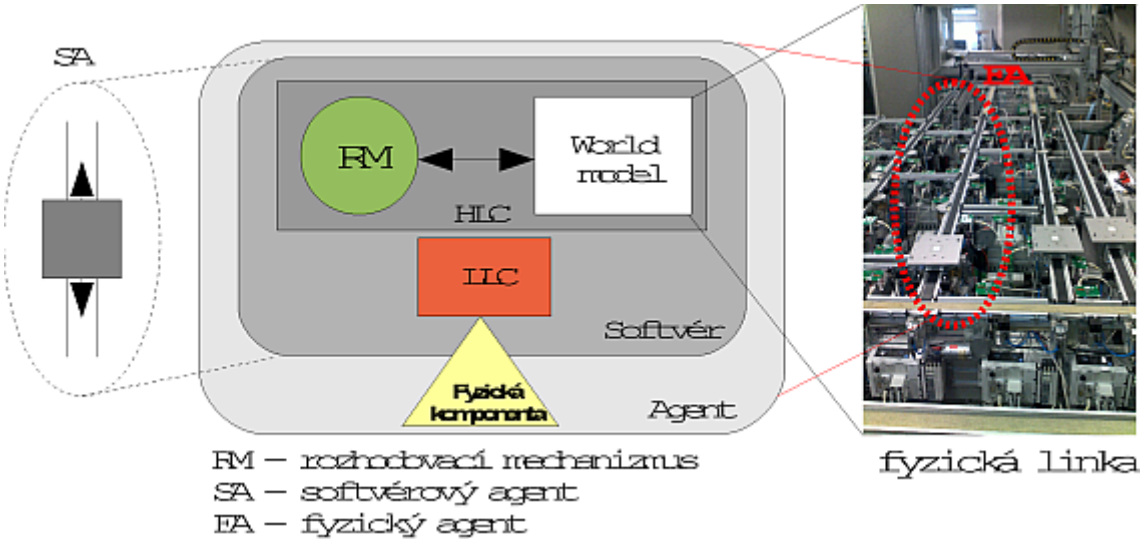
Naproti predchádzajúcemu príkladu využitia modelu simulovaného synchrónne, táto metodika poníma modelovanie z opačnej stránky. Nezaoberá sa analytickým modelom, ale zaoberá sa modelom prostredia v ktorom systém pracuje. Taktiež naproti predchádzajúcej metodike diagnostiky poruchy, táto je využitá v decentralizovaných riadiacich systémoch. Využívanie decentralizovaného riadenia má čoraz väčší význam. Naproti centralizovaným systémom majú decentralizované systémy výhodu v tom, že úloha riadenia a diagnostiky sa sústreďuje na konkrétny subsystém. To že sa úloha sústreďuje len na vybranú časť systému dovoľuje značne redukovať algoritmy, čím sa zvýši výkon systému. Okrem toho výpadok subsystému neparalizuje kompletný riadiaci proces ako by to bolo v prípade centrálneho riadenia. Ako sľubná cesta pri decentralizácii sa javia riadiace systémy s využitím automatizačného agenta.

Agent obsahuje informácie o svojom prostredí a okolí v ktorom pracuje. Má možnosť reagovať na anomálie v tomto prostredí a lokálne zasahovať do tohoto okolia. Podľa (Jennings, N. et al., 1998) zavedením agenta do decentralizvaného riadenia sa získajú minimálne 4 výhody. Prvou je, že sa zredukuje komplexnosť pri rozhodovaní v decentralizácii a zaistí sa lepšia zhoda modelu s realitou. Druhou je že pomocou kooperačných techník agentov je sprístupnené riadenie akcií na vysokej úrovni abstrakcie, čím sa systém stáva flexibilnejší v zmysle konfigurácie a veľkosti. Tretia je že rôzni agenti môžu mať priradené rôzne hodnotiace kritéria pre situáciu, ktorá nastala. Štvrtá je adaptácia systému na neočakávané situácie. Jedným z príkladov použitia decentralizovaného riadenia s využitím agenta a modelu systému v priemysle je popísaná v (Vallée, M. et al., 2009). V tejto práci bude reprodukovaná len myšlienka aplikácie s malými obmenami.

Agent je tvorený dvoma komponentami. Hardvérovou komponentou, čo predstavuje fyzickú časť sústavy, kotrá je riadená. Druhá je softvérová komponenta, ktorá má za úlohu zabezpečiť riadenie fyzického agenta a zabezpečiť komunikáciu medzi ostatnými agentami. Architektúra je uvedená na obrázku 3.3[1]. Vysvetlenie bude urobnené na modele prepravníka paliet. Fyzickú časť agenta tvorí v tomto prípade časť linky (jeden pás), slúžiaci na prepravu palety medzi dvoma pásami. Na obrázku 3.3 je znázornený červenou prerušovanou čiarou a popisom `FA`. Jeho softvérová komponenta je zložená z riadenia *Low Level Control* (LLC) na nižšej úrovni a riadenia *High Level Control* (HLC) na vyššej úrovni. LLC má za úlohu sledovať a riadiť základné funkcie dopravného pásu ako je pohyb určitou rýchlosťou a určitým smerom, podľa situácie ktorú vyhodnocuje vyššia vrstva.

[1]Obrázok fyzickej linky bol použitý s povolením spoluautora článku (Vallée, M. et al., 2009)

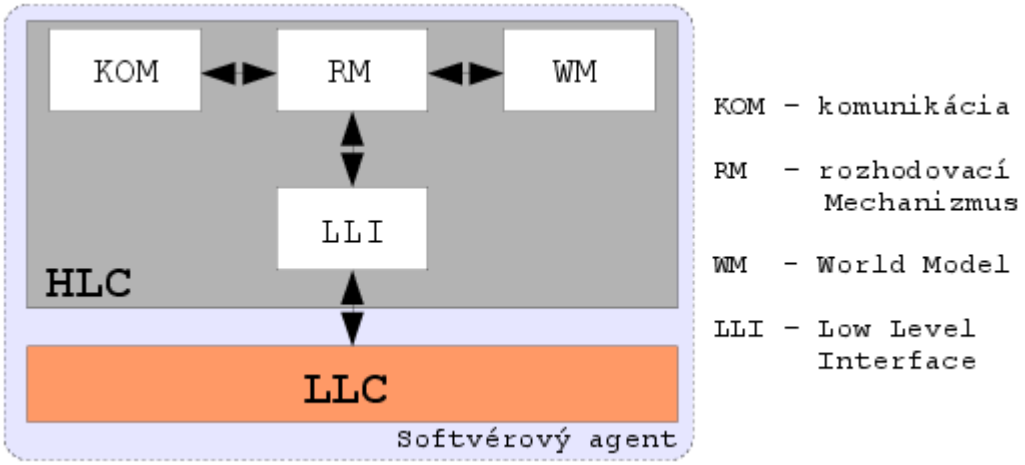
To znamená, že má priame spojenie na hardvér. Dokáže rozpoznávať jednoduché udalosti ktoré sa v systéme nastali (motor beží, detekčné čidlo zadetekovalo objekt a pod.) a túto znalosť zúžitkovať dosiahnutie svojho cieľa (riadenie fyzikálnej sústavy v zmysle udržiavania povolených trajektórií systému). Takto získané informácie už ďalej nespracováva a predáva ich mechanizmu do vyššej vrstvy. Vo vyššej vrstve je implementovaný model prostredia v ktorom daný agent pracuje. Model obsahuje popisy susedných agentov nachádzajúcich sa v tomto prostredí a taktiež ontológiu vzťahov medzi nimi. Napríklad pri vstupe na pás sa nachádza podávač paliet. Podávač je namodelovaný vlastným agentom. Agent pásu má teda informáciu o agentovi podávača a naopak. Agent podávača má informáciu že podáva palety na pás a agent pásu má informáciu že dostáva palety od podávača. Ak agent podávača má vo svojom repozitáre zapísanú akciu že paleta bola odovzdaná na pás a pás nemá informáciu o tom že by paletu dostal, znamená to chybu niekde medzi týmito dvoma agentami. Z predchádzajúceho popisu je zrejmé, že HLC vrstva musí obsahovať aj nejaký mechanizmus pre ukladanie akcií ktoré nastali a majú nastať. V architektúre agenta sú tieto informácie o prostredí a o akciách nazvané ako *World Model* (WM). Ďalší blok `RM` má za úlohu z dostupných informácií od okolitých agentov diagnostikovať problém a urobiť rozhodnutie na jeho riešenie.



Obr. 3.3: Architektúra automatizačného agenta znázornená na modele transportného pásu paliet

HLC má za úlohu diagnostikovať chybu a rozhodnúť aká akcia nastane aby bola chyba odstránená a aby bolo možné dosiahnuť globálny cieľ. V prípade prepravy paliet

je to dopravenie palety na konečné miesto. Zaujímavosťou je, že WM agenta obsahuje aj informáciu sám o sebe. T.j. základné parametre a vzťahy k ostatným agentom. Na obrázku 3.3 je to znázornené popisom `SA`. Autori konceptu sa o tomto druhu agenta vyjadrujú ako o agentovi s reflexívnym modelom sveta. Aplikácia konceptu bola prevádzaná, ako bolo pre predstavu vyššie popísané, na transportnom páse paliet. Podrobnejší popis je uvedený v článku (Vallée, M. et al., 2010) kde bol tento postup nasadený. Úloha softvérového agenta je teda zrejmá, no pre lepšiu predstavu je uvedená na obrázku 3.4.



Obr. 3.4: Úloha softvérového agenta pri diagnostike

Z hľadiska tejto práce zaoberajúcej sa využitím modelov v automatizačnej praxi je zaujímavý práve koncept zahrnutia modelu prostredia do riadiaceho algoritmu. Tento model predstavuje symbolickú reprezentáciu okolia agenta v ktorom pracuje. Je vybavený mechanizmom pre detekciu stavov, ktoré môže agent dosahovať. To otvára dvere pre využitie modelovacích nástrojov využitých v predchádzajúcej kapitole. Konkrétne mechanizmus detekcie stavov sa dá interpretovať ako systém disktrétnych udalostí. Fyzického agenta je taktiež možno interpretovať na dvoch úrovniach. Na nízkej ako dynamický spojitý systém, ktorý je popísaný diferenciálnymi rovnicami a reprezentuje vývoj systému v čase ale z mikroskopického hľadiska. Na vysokej úrovni sa dá agent znova vnímať ako vyvíjajúci sa v čase, ale z makroskopického hľadiska, to znamená že je možné ho reprezentovať ako systém disktrétnych udalostí. Je ho možné modelovať ako stavový automat a teda pre prenos na platformu PLC by mohol byť využitý StateFlow a následne PLC Coder.

Kapitola 4

Záver

Na prácu je možno prihliadať ako na motivačné dielo pre projektantov z praxe. Cieľom práce bolo v prvom rade sprostredkovať nový pohľad na metódu Model-Based Design, ktorá sa do priemyslu dostáva pomaly. Integrácia a propagácia tejto metódy bola smerovaná hlavne na hardvérové komponenty značky Tecomat. Predstavený bol softvér potrebný pre programovanie týchto automatov, ako je Mosaic, a softvér pre priamu podporu návrhovej metódy MBD ako je Matlab-Simulink-PLC Coder. Vzhľadom na to, že práca bola obmedzovaná licenčnými problémami[1], tak sa podarilo využiť dostupné prostrieky v maximálnej miere aby bolo možné sledovať ciele zadania práce. V práci bolo prezentovaných pár jednoduchých príkladov, ktoré demonštrovali využitie metódy MBD pre realizáciu návrhu modelu systému a možnosť jeho následnéhžo využitia.

4.1 Zhodnotenie dosiahnutých cieľov

Začiatok práce je určený hlavne čitateľom, ktorí s rozoberanou problematikou a softvérom neprichádzajú do styku každodenne. Stručne boli predstavené softvérové prostriedky na ktorých je práca postavená. Keďže práca je postavená na možnosti využitia generácie kódu do normovaného jazyka pre programovateľné automaty, bolo vhodné predstaviť možnosti a obmedzenia, ktoré norma ponúka. Krátky úvod do programovacích štandardov využívajúcich sa v oblasti automatizačnej techniky bol demonštrovaný na jednoduchých príkladoch. Tie dávajú čitateľovi náhľad na mieru náročnosti programovania v tom kto-

[1] Pre vytvorenie práce bola poskytnutá 30-dňová skúšobná verzia Matlabu s komponentou PLC-Coder, na ktorej je celá práca postavená.

rom jazyku v porovnaní s jazykom .ST, ktorý je predmetom tejto práce. Nakoniec po prehľade normy je kapitola zakončená pasážou popisujúcou výhody a nevýhody návrhovej metódy a tvorí úvod k praktickej realizácii súboru jednoduchých systémov, na ktorých je návrh demonštrovaný.

Druhá kapitola je ťažiskom práce a nehľadiac na predstavenie a preštudovanie normy, ktoré bolo rozobraté v prvej kapitole, pokrýva tri prvé body zadania práce. Na začiatku je rozobratý problém implementácie vygenerovaného kódu a tým aj komunikačné mohžnosti medzi programami. Pristúpilo sa k riešeniu priamej implementácie súboru do projektu pre programovateľný automat. Boli vytvorené štyri demonštračné príklady s primeranou obtiažnosťou pre ukázanie simulácie modelu systému na riadiacom automate. Je treba poznamenať, že model z dôvodu absencie hardvéru programovateľného automatu bol simulovaný na softvérovom emulátore automatu. Zhodnosť proramového a skutočného automatu bola overená pri tvorbe bakalárskej práce (Andrejco, M., 2008), kedy sa pracovalo na skutočnom hardvéri. Voľba modelov bola zámerná. Jednoduchý RC-člen bol použitý ako štart do problematiky a bol navrhnutý zámerne na nízke frekvencie aby bolo vidieť že pri simulácii modelu tohoto typu na inej platforme nevznikajú problémy. Ďalším modelovaným systémom bol linearizovaný model elektromotora. Systém má rýchlu dynamiku, je druhého rádu. Pri tomto type systému vznikali menšie problémy v simulácii, ktoré boli rozobraté v sekcii 2.3. Tretím demonštrantom bol príklad zjednodušenéhžo tepelného výmenníku. Išlo skôr o principiálne vysvetlenie problému a hlavne demoštráciu systému s veľkými časovými konštantami. Výhodou je, že takýto model je možné zrýchlene simulovať, čím sa šetrí čas. Pre vyskúšanie bol model v PLC simulovaný v reálnom čase. Ako posledný bol uvedený model práčky. V práci je kôli rozsahu uvedený len podsystém riadiaceho modulu a to programový volič a modul riadenia motora[2]. Model bol využitý zámerne ako upozornenie na možnosť generovať kód aj zo stavových automatov.
Priebežne pri riešení problémov boli konzultované možnosti nesúladu generovaného kódu s normou a kompatibilita generovaného kódu. Kód nebol po preklade kompatibilný, s čím súvisel následný návrh postprocesora pre vygenerovaný súbor. Postprocesor bol testovaný na viacerých kódoch a funkcie pre odstraňovanie nekompatibilít boli pridávané postupne po objavení sa chyby. Funkčnosť postprocesora je týmto obmedzená na zistené chyby rozobraté v sekcii 2.6.
Posledná kapitola sa nakoniec zaoberá problémami a príkladami možného využitia návrhovej metódy v praxi. Sú rozobraté dva problémy diagnostiky s využitím modelu vytvoreného

[2]Model sa nesnaží o inováciu technológie. Koncept bol navrhnutý intuitívne a má slúžiť pre demoštráciu možnosti využitia automatu v generovaní kódu.

vyššie popísaným spôsobom. Boli popísané dva prístupy ako sa dá na model nazerať. Kapitola sa nesie v teoretickej rovine a dáva inšpiráciu na príklad použitia MBD v praktickom živote. Kapitola sa opiera hlavne o dosiahnuté výsledky pri vývoji diagnostických metód, ktoré boli aplikované na reálnu sústavu. Otvára tým cestu k prípadnej novej téme na diplomovú prácu, ktorá by sa mohla zaoberať len konkrétnym praktickým problémom napríklad pri HVAC aplikáciách. Diagnostika s využitím metodiky popísanej v (Vallée, M. et al., 2009), (Kain, S. et al., 2010) by zohľadňovala problémy týkajúce sa diagnostikovateľnosti systému (Sapath, M. et al., 1995).

Aj napriek nemalým licenčným prekážkam sa podarilo v práci dosiahnúť zadané ciele. Dúfam že práca poslúži svojmu demonštračnému účelu a bude inšpiráciou pre ľudí zainteresovaných do tejto problematiky.

Literatúra

Andrejco, M. (2008), *Sada pro dlouhodobé měření a zviditelnění procesů ve vytápěných prostorech.*

Bohlin, T. (1994), 'A case study of grey box identification', *Automatica* **30**(2), 307 – 318.

Hoefling, T. a Isermann, R. (1996), 'Fault detection based on adaptive parity equations and single parameter tracking', *Control Eng. Practice* **4**(10), 1361 – 1369.

Jennings, N., Jennings, N.R. a Wooldridge, M.J. (1998), *Agent technology: foundations, applications, and markets*, Springer. `http://books.google.com/books?id=D5pnZ_fUplUC`. ISBN 9783540635918.

Kain, S., Schiller, F. a Frank, T. (2010), 'Monitoring and diagnostics of hybrid automation systems based on synchronous simulation', *IEEE Conferences* pp. 260–265.

MathWorks (2011*a*), 'Online documentation for simulink', Webové stránky. `http://www.mathworks.com/help/toolbox/simulink/`.

MathWorks (2011*b*), 'R2010b documentation', Webové stránky. `http://www.mathworks.com/help/techdoc/`.

Mathworks (2011*c*), 'Stateflow user's guide', webové stránky. `http://www.mathworks.com/help/pdf_doc/allpdf.html`.

Šmejkal, L. a Pohl, T. (2009), 'Plc, řízení a technická diagnostika', *Automatizace* **52**(5), 313.

Noskievič, P. (1999), *Modelování a identifikace systémů*, Montanex. `http://books.google.com/books?id=heOuAAAACAAJ`. ISBN 9788072250301.

Sapath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K. a Teneketzis, D. (1995), 'Diagnosability of discrete-event systems', *IEEE Conferences* pp. 1555 – 1575.

Teco a.s. (2007*a*), 'Programování PLC podle normy IEC 61 131-3 v prostředí Mosaic', Webové stránky. `http://tecomat.com/index.php?ID=318`.

Teco a.s. (2007*b*), 'Příručka programátora PLC TECOMAT', Webové stránky. `htto://www.tecomat.cz/docs/cze/Tecomat/txv00109.pdf`.

Teco a.s. (2010), 'Začínáme s mosaicem', Webové stránky. `http://www.tecomat.com/index.php?ID=365`.

Vallée, M., Keindl, H., Lepuschitz, W., Arnautovic, E. a Vrba, P. (2009), 'An automation agent architecture with a reflective world model in manufacturing systems', *IEEE Conferences* pp. 305 – 310.

Vallée, M., Merdan, M. a Vrba, P. (2010), 'Detection of anomalies in a transport system using automation agents with a reflective world model', *IEEE Conferences* pp. 489–494.

Wikipedia (2011*a*), 'Discrete event dynamic sytems', webové stránky. `http://en.wikipedia.org/wiki/Discrete_event_dynamic_system`.

Wikipedia (2011*b*), 'IEC 61131', webové stránky. `http://en.wikipedia.org/wiki/IEC_61131`.

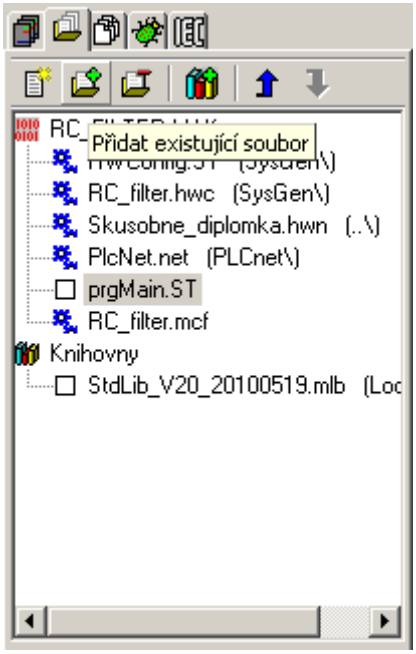
Príloha A

Návody, kódy, parametre

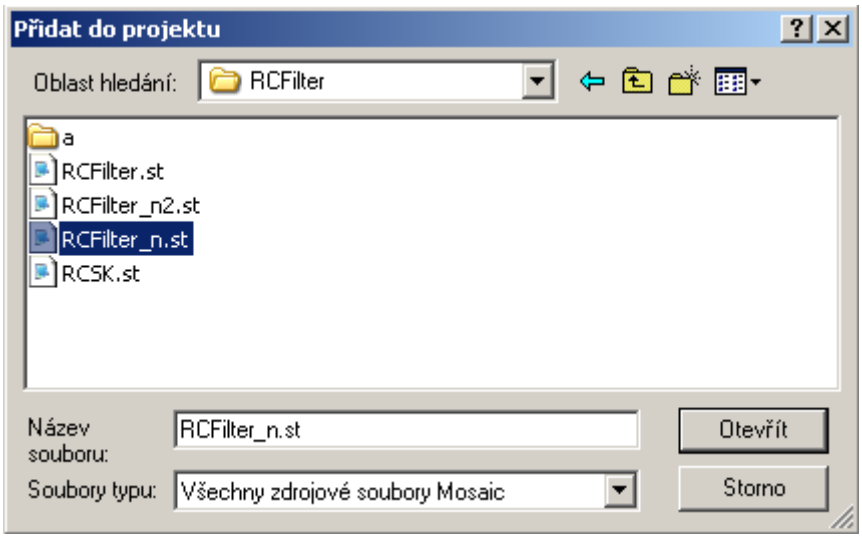
A.1 Návod na implementáciu kódu v Mosaicu

Princíp importu vygenerovaného súboru je jednoduchý. V Mosaicu je potrebné si vytvoriť nový projekt, ak sa model pridáva do nového projektu resp. súbor pridať do existujúceho projektu v ktorom chce byť model vygenerovaný súbor použitý.

1. Krok: pridanie súboru do existujúceho projektu



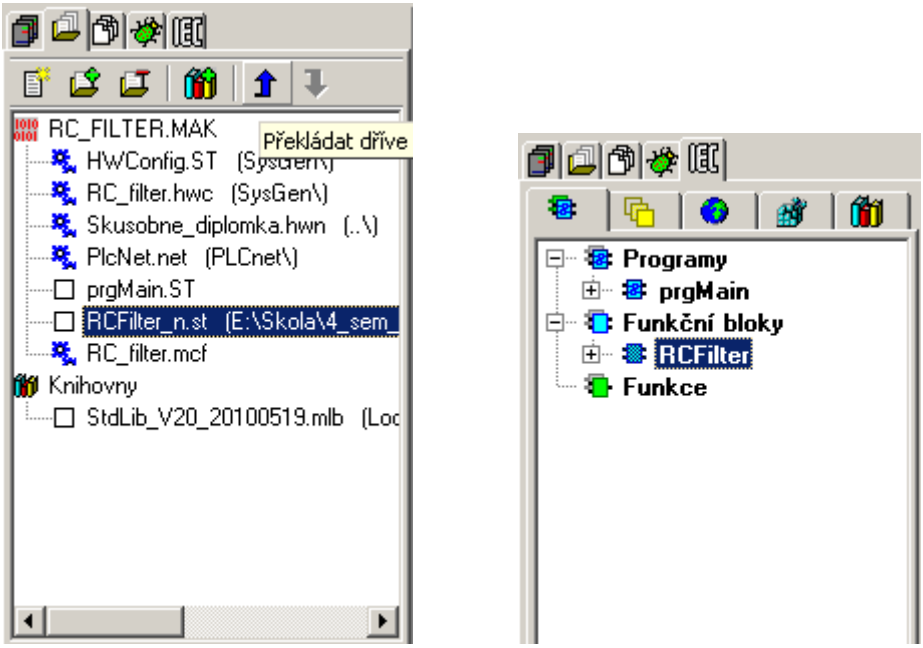
Obr. A.1: Pridanie nového súboru do projektu



Obr. A.2: Výber súboru upraveného postprocesorom

2. Krok: nastavenie poradia prekladania súborov

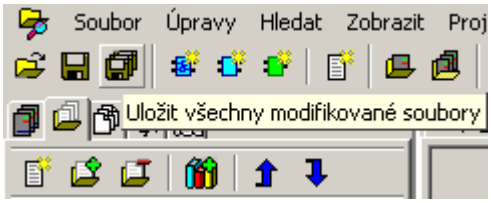
Pri nesprávnom nastavení prekladania súborov vzniká problém že deklarácie globálnych premenných a celkovo deklarácie funkčných blokov použitých vo vygenerovanom súbore nie sú pre prekladač xPRO známe a tak hlási chybu.



Obr. A.3: Zmena poradia kompilácie súborov pre prekladač

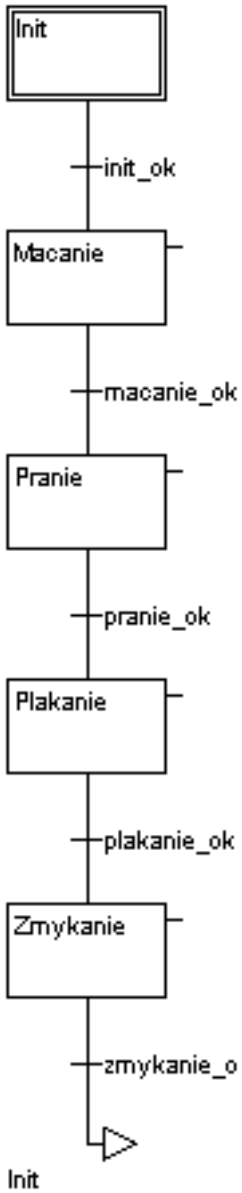
Po pridaní súboru sa v manažérovi IEC objavia všetky funčné bloky, ktoré sú v súbore nadafinované. Znázornené na obrázku A.3 vpravo.

<u>3. Krok:</u> pridanie prázdného riadku na koniec importovaného súboru
Z neznámych príčin xPRO potrebuje pred kompiláciou projektu urobiť v naiportovanom súbore zmenu, aby bolo možné projekt preložiť. Najlepšie je pridanie prázdného riadku do naiportovaného súboru a následné uloženie všetkých zmien v projekte.



Obr. A.4: Uloženie zmien v editovaných súboroch

A.2 SFC program práčky



Obr. A.5: Program práčky vytvorený v SFC

A.3 Testovací funkčný blok testBench

```
------------------ Výpis kódu funkčného bloku testBench pre tepelný výmenník ------------------
------------------------------------- DEKLARAČNÁ ČASŤ ---------------------------------------
FUNCTION_BLOCK TestBench
VAR_INPUT
END_VAR
VAR_OUTPUT
    testVerify: BOOL := TRUE;
    testCycleNum: INT := 0;
END_VAR
VAR
    _i0_FBvymennik: FBvymennik;
END_VAR
VAR_TEMP
    tb_T1: ARRAY [0..40] OF LREAL :=[ 353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,
      353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,
      353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,353.15,
      353.15,353.15,353.15,353.15,353.15,353.15,353.15];
    cycle_T1: LREAL;
    tb_T2: ARRAY [0..40] OF LREAL :=[ 293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,
      293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,
      293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,293.15,
      293.15,293.15,293.15,293.15,293.15,293.15,293.15];
    cycle_T2: LREAL;
    tb_T_10: ARRAY [0..40] OF LREAL :=[ 80,75.585034223540617,71.8050570160803,
      68.566490453826759,65.789597456470972,63.406433529348021,61.359101643968131,
      59.598265392480755,58.0818821915766,56.774123968561582,55.6444575822959,
      54.66686133829478,53.819157456113658,53.0824433281399,52.440606948723882,
      51.879914056501434,51.388656376416463,50.956851918751568,50.575989630790218,
      50.238811836983416,49.939128874983396,49.671661162616317,49.431904636075387,
      49.216016100452521,49.020715545641224,48.843202916793075,48.681087200110483,
      48.532326001361525,48.395174064248749,48.26813940558759,48.149945940058331,
      48.039501634127646,47.935871370872405,47.838253828542463,47.745961778877529,
      47.658405299102469,47.575077466423295,47.495542167659437,47.41942371101743,
      47.3463979733325,47.276184855572467];
    cycle_T_10: LREAL;
    out_T_10: LREAL;
    tb_T_20: ARRAY [0..40] OF LREAL :=[ 20,23.304542868596343,26.102816407890259,
      28.470122296918248,30.4706094317703,32.15892471574756,33.581619528481212,
      34.778348033647887,35.782888133283791,36.624011317108341,37.326223770497393,
      37.910397794923369,38.394309774707892,38.793098521321269,39.119655779435561,
      39.384958934872373,39.598354478642364,39.767799515249294,39.900067524780525,
      40.000923669291581,40.075274150998268,40.127293462673265,40.160532802264811,
      40.178012439493955,40.1823004095977,40.175579557858441,40.159704659060253,
      40.136251080838406,40.1065562424792,40.071754935496529,40.032809414493272,
      39.9905350323553,39.945622079266172,39.898654387424017,39.850125180183852,
      39.800450573495539,39.749981077142422,39.699011391853162,39.6477887545405,
      39.596520046584942,39.545377848274256];
    cycle_T_20: LREAL;
    out_T_20: LREAL;
END_VAR
------------------------------------------> 1 <------------------------------------------
```

```
---------------------------------------------> 1 <---------------------------------------------
------------------ Výpis kódu funkčného bloku testBench pre tepelný výmenník ------------------
---------------------------------------- VÝKONNÁ ČASŤ ----------------------------------------
IF testCycleNum < 41 THEN
    (* TEST CYCLE SETUP  *)
    cycle_T1 := tb_T1[testCycleNum];
    cycle_T2 := tb_T2[testCycleNum];
    cycle_T_10 := tb_T_10[testCycleNum];
    cycle_T_20 := tb_T_20[testCycleNum];
    IF testCycleNum = 0 THEN
        (* INIT  *)
        _i0_FBvymennik(ssMethodType := SS_INITIALIZE, T1 := cycle_T1, T2 := cycle_T2);
        out_T_10 := _i0_FBvymennik.T_10;
        out_T_20 := _i0_FBvymennik.T_20;
    END_IF;
    (* OUTPUT  *)
    _i0_FBvymennik(ssMethodType := SS_OUTPUT, T1 := cycle_T1, T2 := cycle_T2);
    out_T_10 := _i0_FBvymennik.T_10;
    out_T_20 := _i0_FBvymennik.T_20;
    (* UPDATE  *)
    _i0_FBvymennik(ssMethodType := SS_UPDATE, T1 := cycle_T1, T2 := cycle_T2);
    out_T_10 := _i0_FBvymennik.T_10;
    out_T_20 := _i0_FBvymennik.T_20;
    (* VERIFY  *)
    IF testVerify AND (ABS(out_T_10 - cycle_T_10) > 1.0E-5) THEN
        testVerify := FALSE;
    END_IF;
    IF testVerify AND (ABS(out_T_20 - cycle_T_20) > 1.0E-5) THEN
        testVerify := FALSE;
    END_IF;
    testCycleNum := testCycleNum + 1;
END_IF;
END_FUNCTION_BLOCK
-----------------------------------------------------------------------------------------------
```

A.4 Parametre výmenníka tepla

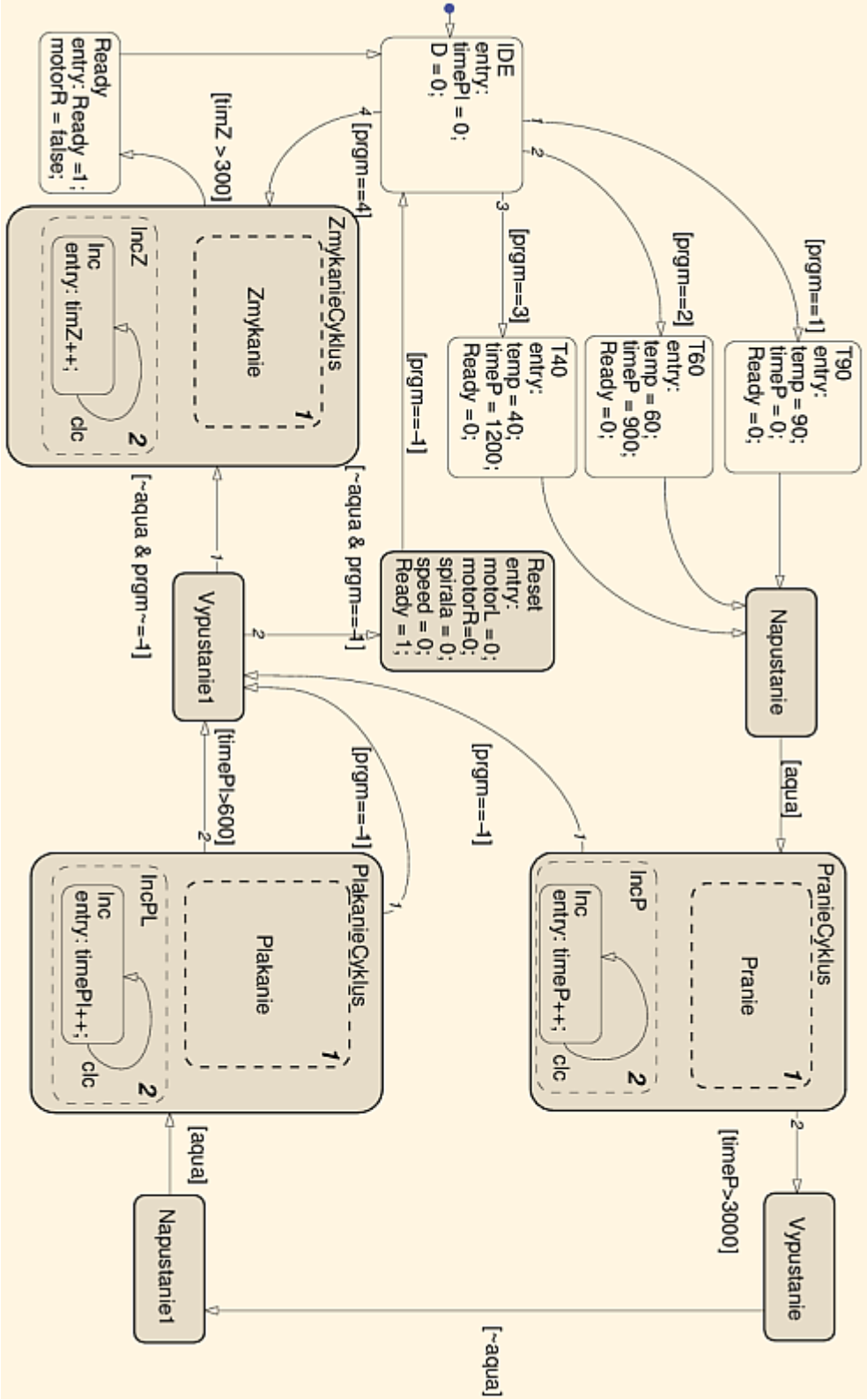
veličina	**značka**	**hodnota**	**jednotka**
hustota média	$\rho_{1,2}$	1000	$kg.m^{-3}$
súčiniteľ prestupu rovinnou stenou	k_s	1000	Wm^2K^{-1}
objem prvej nádrže	V_1	0.03	m^3
objem druhej nádrže	V_2	0.04	m^3
merné teplo pri konštantnom tlaku	c_{P1}	4195	$kJkg^{-1}K^{-1}$
merné teplo pri konštantnom tlaku	c_{P2}	4183	$kJkg^{-1}K^{-1}$
objemový prietok na vstupe 1. nádrže	ρ_1	10^{-4}	m^3s^{-1}
objemový prietok na vstupe 2. nádrže	ρ_2	$2.1 \cdot 10^{-4}$	m^3s^{-1}
objemový prietok na výstupe 1. nádrže	ρ_{10}	10^{-4}	m^3s^{-1}
objemový prietok na výstupe 2. nádrže	ρ_{20}	$2.1 \cdot 10^{-4}$	m^3s^{-1}
teplota média na vstupe do 1. nádrže	T_1	353.15	K
teplota média na vstupe do 2. nádrže	T_2	293.15	K
počiatočná teplota na výstupe 1. nádrže	T_{10}	353.15	K
počiatočná teplota na výstupe 2. nádrže	T_{20}	293.15	K

Tabuľka A.1: Tabuľka využitých simulačných parametrov nepriamého výmenníka tepla

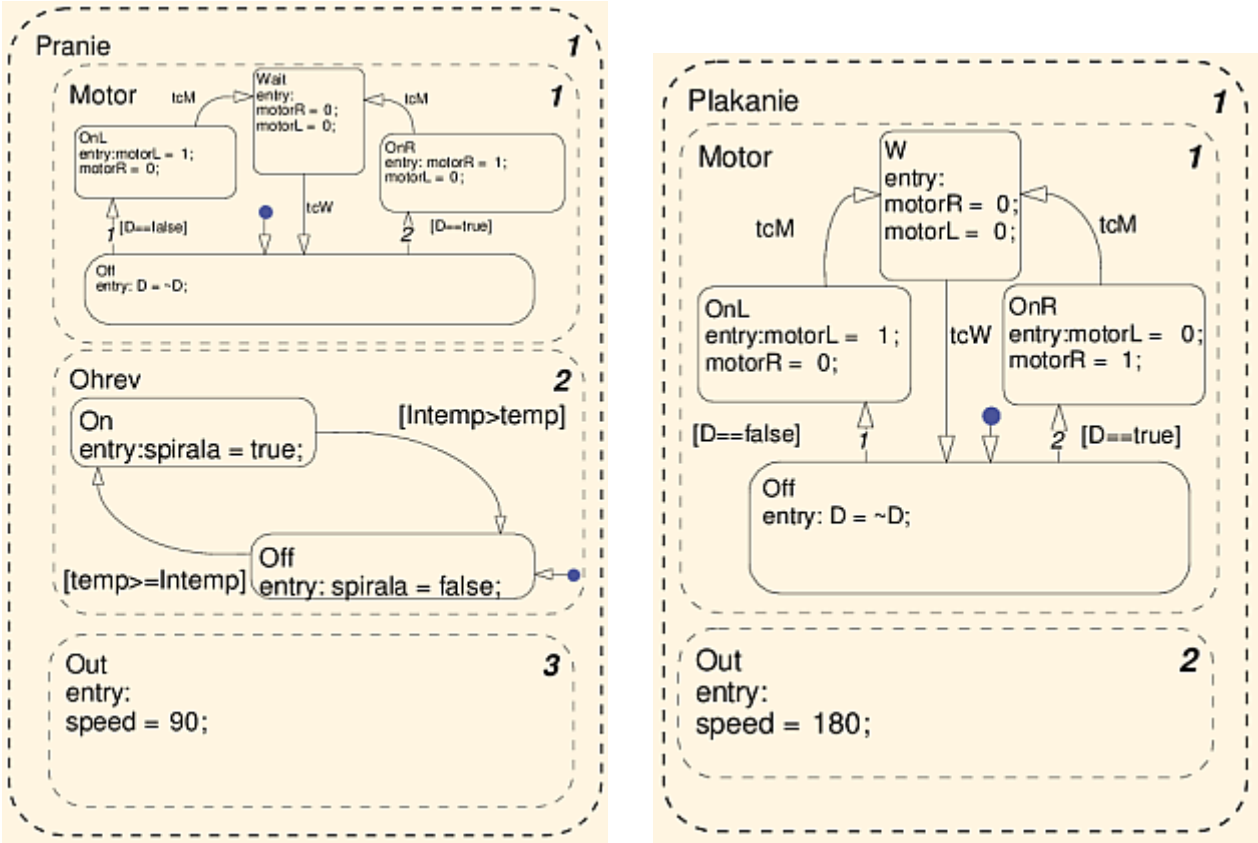
Matice diskretizovaného systému pre vyššie uvedené parametre, metóda ZOH, perióda vzorkovania $T_s = 5s$:

$$\boldsymbol{A_d} = \begin{bmatrix} 0.9105 & 0.0726 \\ 0.05461 & 0.9198 \end{bmatrix} \qquad \boldsymbol{B_d} = \begin{bmatrix} 0.0159 & 0.0009821 \\ 0.000469 & 0.02517 \end{bmatrix} \tag{A.1}$$

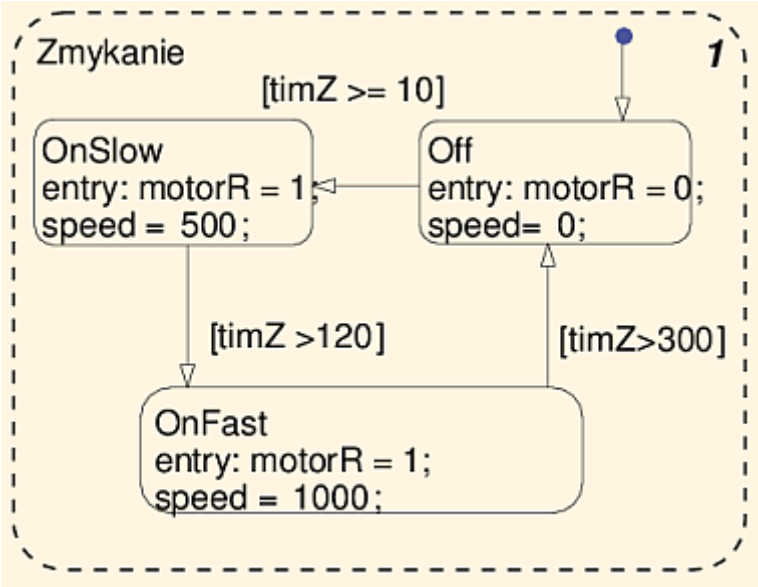
A.5 Riadiaci modul práčky



Obr. A.6: Kompletný algoritmus práčky namodelovaný v SFC



Obr. A.7: Detail subgrafov použitých v algoritme



Obr. A.8: Detail subgrafov použitých v algoritme

A.6 Vygenerovaný kód programového voliča práčky

```
------------------------------------ Vygenerovaný kód ---------------------------------------
VAR_GLOBAL CONSTANT
    Programovy_IN_IDE: USINT := 2;
    Programovy_IN_s90: USINT := 6;
    Programovy_IN_s60: USINT := 5;
    Programovy_IN_s40: USINT := 4;
    Programovy_IN_spin: USINT := 7;
    Programovy_IN_Chosen: USINT := 1;
    Programovy_IN_Reset: USINT := 3;
    SS_INITIALIZE: SINT := 2;
    SS_OUTPUT: SINT := 3;
END_VAR
VAR_GLOBAL
END_VAR
FUNCTION_BLOCK Programovy
VAR_INPUT
    ssMethodType: SINT;
    bt1: BOOL;
    bt2: BOOL;
    bt3: BOOL;
    bt4: BOOL;
    btR: BOOL;
    btStart: BOOL;
    Ready: BOOL;
END_VAR
VAR_OUTPUT
    prgm: SINT;
END_VAR
VAR
    b_prgm: SINT;
    is_active_c1_Programovy: USINT;
    is_c1_Programovy: USINT;
    choice: SINT;
END_VAR
VAR_TEMP
END_VAR
CASE ssMethodType OF
    SS_INITIALIZE:

        is_active_c1_Programovy := 0;
        is_c1_Programovy := 0;
        choice := 0;
        b_prgm := 0;
    SS_OUTPUT:

        IF USINT_TO_INT(is_active_c1_Programovy) = 0 THEN
            is_active_c1_Programovy := 1;
            is_c1_Programovy := Programovy_IN_IDE;
            b_prgm := 0;
----------------------------------------------> 1 <---------------------------------------------
-------------------------------------------------------------------------------------------------
```

```
---------------------------------------------> 1 <---------------------------------------------
------------------------------------- Vygenerovaný kód ---------------------------------------
        ELSE
            CASE is_c1_Programovy OF
                Programovy_IN_Chosen:
                    IF btR THEN
                        is_c1_Programovy := Programovy_IN_Reset;
                        b_prgm := -1;
                    ELSIF Ready THEN
                        is_c1_Programovy := Programovy_IN_IDE;
                        b_prgm := 0;
                    END_IF;
                Programovy_IN_IDE:
                    IF bt1 AND ( NOT btR) THEN
                        is_c1_Programovy := Programovy_IN_s90;
                        choice := 1;
                    ELSIF bt2 AND ( NOT btR) THEN
                        is_c1_Programovy := Programovy_IN_s60;
                        choice := 2;
                    ELSIF bt4 AND ( NOT btR) THEN
                        is_c1_Programovy := Programovy_IN_spin;
                        choice := 4;
                    ELSIF bt3 AND ( NOT btR) THEN
                        is_c1_Programovy := Programovy_IN_s40;
                        choice := 3;
                    END_IF;
                Programovy_IN_Reset:
                    IF Ready THEN
                        is_c1_Programovy := Programovy_IN_IDE;
                        b_prgm := 0;
                    END_IF;
                Programovy_IN_s40:
                    IF btR THEN
                        is_c1_Programovy := Programovy_IN_IDE;
                        b_prgm := 0;
                    ELSIF (bt2 AND ( NOT bt3)) OR bt1 THEN
                        is_c1_Programovy := Programovy_IN_s60;
                        choice := 2;
                    ELSIF bt4 AND ( NOT bt3) THEN
                        is_c1_Programovy := Programovy_IN_spin;
                        choice := 4;
                    ELSIF btStart THEN
                        is_c1_Programovy := Programovy_IN_Chosen;
                        b_prgm := choice;
                    END_IF;
---------------------------------------------> 2 <---------------------------------------------
-------------------------------------------------------------------------------------------------
```

```
---------------------------------------------> 1 <---------------------------------------------
------------------------------------- Vygenerovaný kód -------------------------------------
                Programovy_IN_s60:
                    IF btR THEN
                        is_c1_Programovy := Programovy_IN_IDE;
                        b_prgm := 0;
                    ELSIF bt1 AND ( NOT bt2) THEN
                        is_c1_Programovy := Programovy_IN_s90;
                        choice := 1;
                    ELSIF (bt3 AND ( NOT bt2)) OR bt4 THEN
                        is_c1_Programovy := Programovy_IN_s40;
                        choice := 3;
                    ELSIF btStart THEN
                        is_c1_Programovy := Programovy_IN_Chosen;
                        b_prgm := choice;
                    END_IF;
                Programovy_IN_s90:
                    IF btR THEN
                        is_c1_Programovy := Programovy_IN_IDE;
                        b_prgm := 0;
                    ELSIF bt2 AND ( NOT bt1) THEN
                        is_c1_Programovy := Programovy_IN_s60;
                        choice := 2;
                    ELSIF btStart THEN
                        is_c1_Programovy := Programovy_IN_Chosen;
                        b_prgm := choice;
                    ELSIF bt3 OR bt4 THEN
                        is_c1_Programovy := Programovy_IN_s40;
                        choice := 3;
                    END_IF;
                Programovy_IN_spin:
                    IF btR THEN
                        is_c1_Programovy := Programovy_IN_IDE;
                        b_prgm := 0;
                    ELSIF bt3 AND ( NOT bt4) THEN
                        is_c1_Programovy := Programovy_IN_s40;
                        choice := 3;
                    ELSIF btStart THEN
                        is_c1_Programovy := Programovy_IN_Chosen;
                        b_prgm := choice;
                    ELSIF bt1 OR bt2 THEN
                        is_c1_Programovy := Programovy_IN_s60;
                        choice := 2;
                    END_IF;
                ELSE
                    is_c1_Programovy := Programovy_IN_IDE;
                    b_prgm := 0;
            END_CASE;
        END_IF;
        prgm := b_prgm;
END_CASE;
END_FUNCTION_BLOCK
-------------------------------------------------------------------------------------------------
```

Príloha B

Obsah priloženého CD

K práci je priložené CD s obsahom zdorojových modelov využitých pre simulácie. Okrem toho sú tam uložené originál zdrojové kódy, ktoré boli výstupom nástroja PLC-Coder a kódy spracované postprocesorom. Je priložený balíček s postprocesorom, obsahujúci funkciu `precode()` vytvorenú práve na spracovanie súborov využiých v tejto práci.

- Adresár 1: Modely
- Adresár 2: Postprocesor
- Adresár 3: Diplomova_Praca