České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

DIPLOMOVÁ PRÁCE

Processor In the Loop simulace

Praha, 2007 **Martin Rapavý**

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 12.1.2007

podpis

Poděkování

Na tomto místě bych rád poděkoval Ing. Liborovi Waszniowskému, Ph.D., vedoucímu mojí diplomové práce, za jeho cenné rady, odbornou pomoc a snahu vyhovět veškerým mým dotazům a prosbám. Můj dík patří také mé rodině, přítelkyni a přátelům za jejich psychickou pomoc a pevné nervy.

Abstrakt

Tato diplomová práce se zabývá návrhem podpory pro processor in the loop simulaci v Processor Expert Embedded Real-Time targetu, který je vyvíjen na Katedře řídicí techniky. V práci jsou diskutovány výhody automaticky generovaného kódu při návrhu software pro vestavné řídicí systémy a principy generování kódu v Matlab Simulinku.

Je uveden přehled existujících řešení a přehled možných způsobů, jak zajistit podporu real-time processor in the loop simulace.

Navržené řešení využívá nástrojů Matlab, Simulink, Real-Time Workshop a Processor Expert. Je popsána architektura implementovaného řešení a vzájemná spolupráce programů při použití simulace.

Možnosti výsledného řešení a způsob práce s ním jsou prezentovány na demonstrační úloze. Jsou porovnány výsledky model in the loop a processor in the loop simulace. Nastíněny jsou možnosti dalšího vývoje.

Abstract

This gradution thesis describes the design of the processor in the loop (PIL) simulation support in Processor Expert Embedded Real-Time target, which has been developed by Department of Control Engineering. The advantages of an automatically generated code in the development of a control embedded sofware and the principles of a code generation in Matlab Simulink are discussed in the thesis.

Summary of existing solutions of PIL simulation is mentioned. Possible ways to provide a real-time processor in the loop simulation support are introduced.

The designed solution utilizes Matlab, Simulink, Real-Time Workshop and Processor Expert tools. The architecture of the implemented solution and the cooperation of the tools in the simulation are described.

The potential of the final solution and its usage are presented in the demo task. Results of the model in the loop simulation and the processor in the loop simulation are discussed. The possibilities of a future development are outlined.

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky *Školní rok:* 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Martin Rapavý

Obor: Technická kybernetika

Název tématu: Processor In the Loop simulace

Zásady pro vypracování:

1. Seznamte se s principy generování kódu ze simulinku pomocí nástroje Real-time workshop.
2. Implementujte podporu pro simulaci Procesor In the Loop pro vybraný HW.
3. Vytvořte demonstrační úlohu.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Libor Waszniowski

Termín zadání diplomové práce: zimní semestr 2005/2006

Termín odevzdání diplomové práce: leden 2007

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

v zast.

prof. Ing. Vladimír Kučera, DrSc.
děkan

V Praze dne 20.01.2006

Obsah

Seznam příloh

Seznam obrázků

Seznam obrázků v přílohách

Seznam tabulek

Seznam tabulek v přílohách

Kapitola 1

Úvod

Validace řídicího algoritmu a nastavování (ladění) parametrů regulátoru je nepostradatelnou součástí každého návrhu řídicího systému. Model in the loop simulace (MIL) je používána před implementací řídicího algoritmu na cílový hardware, slouží k ověření správnosti dosavadního návrhu softwarovou simulací modelu uzavřené smyčky soustavy a regulátoru.

Dalším krokem v návrhu řídicího systému je spuštění kódu implementujícího řídicí algoritmus na daném hardwaru. Software řízení je obvykle vyvíjen souběžně s hardwarem řídící jednotky nebo se samotnou řízenou soustavou. **Processor in the loop (PIL) simulace** je využívána pro validaci řídicí aplikace v okamžiku, kdy řízená soustava nemusí být ještě dokončena (není k dispozici).

V závěrečné fázi vývoje je možné využít hardware in the loop (HIL) simulaci, při které je již zapojena elektronická řídicí jednotka (ECU) s reálnou řízenou soustavou.

PIL simulace využívá zapojení mikroprocesoru v uzavřené smyčce se simulačním softwarem (simulátorem). Procesor je přitom cílovým hardwarem, pro který je řídicí algoritmus vyvíjen a na němž bude jeho konečná podoba implementována.

Kód řídicího algoritmu je spuštěn na cílovém mikroprocesoru (MCU) a model řízené soustavy na simulátoru. Komunikačním kanálem jsou přenášena vypočtená data mezi MCU a simulátorem. PIL simulací je např. možné zjistit paměťové a časové požadavky, tedy to, zda-li je zvolený procesor vhodný pro vyvíjený řídicí systém.

Výsledky získané PIL simulací nemohou být úplně přesné, ale jsou nejlepší, jaké se dají získat v této fázi vývoje, kdy řízená soustava není k dispozici. Nahrazení reálné soustavy jejím modelem, přenos informace po komunikačním kanálu a úprava kódu nutná k přesměrování fyzických periferií MCU na komunikační kanál - to jsou znaky PIL, které způsobují odlišnosti ve výsledcích simulace [1].

Cílem této práce je vytvoření podpory processor in the loop simulace pro PEERT (Processor Expert Embedded Real-time Target [1] vyvíjený na Katedře řídicí techniky).

Přesnější požadavky jsou přitom následující:

- použít Matlab Simulink, Real-Time Workshop
- vytvořit target a block set pro Simulink s podporou PIL simulace
- vytvořit demonstrační úlohu

Uživatel výsledného řešení bude moci po vytvoření modelu s regulačním obvodem vygenerovat kód regulátoru nástrojem Real-Time Workshop. Ten se poté nahraje do procesoru a spustit. Stejně tak bude možné vygenerovat kód soustavy a nahrát jej do simulátoru. Umožní se tak testování regulátoru v simulaci bez hotového hardware ECU a bez reálné soustavy.

Motivací k práci je i možnost použít výsledné řešení v laboratořích katedry řídicí techniky pro snažší tvorbu aplikací v rámci průmyslových a vědeckých projektů, diplomových prací, případně při výuce. Po úplném odzkoušení funkčnosti bude PIL simulace tvořit součást komerčního produktu firmy UNIS, spol. s r. o.

Úvod do rychlého vývoje aplikací (RAD) a přehled souvisejicích prací je obsahem 2. kapitoly. Kapitola 3 popisuje použité nástroje při tvorbě podpory PIL simulace, dále je zde popsána motivace použití automaticky generovaného kódu při návrhu vestavných řídicích systémů a jakým způsobem lze generovat kód ve vývojovém prostředí Matlab.

Ve 4. kapitole jsou po přehledu existujících řešení diskutována nová vlastní řešení PIL simulace. Porovnáním jednotlivých řešení byla vybrána nejvhodnější varianta k implementaci. Obsahem 5. kapitoly je popis zvolené architektury vlastního řešení PIL simulace.

Pro vytvoření podpory processor in the loop simulace bylo nutné upravit PEERT. Vznikl tak nový target PEERT PIL. Všechny provedené modifikace jsou obsaženy v kapitole 6. V 6.5 je popsán vytvořený demonstrační model a v 6.5.1 jsou porovnány výsledky simulací MIL a PIL.

Nakonec jsou v 6.6 nastíněny možnosti dalšího vývoje, závěrečná zhodnocení pak v 7. kapitole.

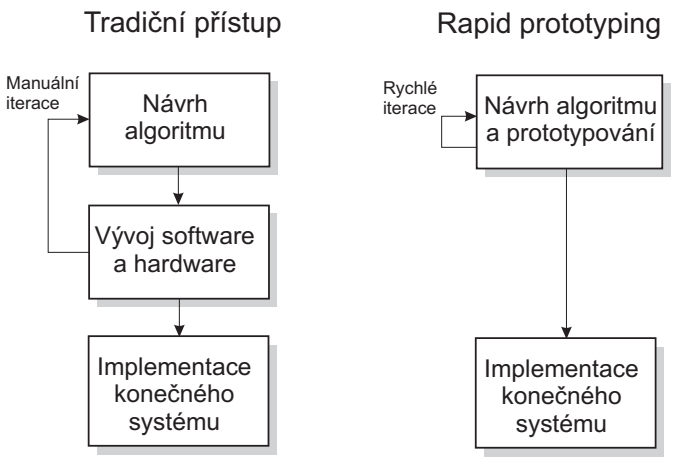
Kapitola 2

Popis problematiky

Tato kapitola si klade za cíl představit metodu návrhu systémů zvanou rapid prototyping. Rychlé prototypování je využíváno vývojovými pracovníky především v automobilovém průmyslu. Je zde popsán i nástroj TargetLink, součást dSpace [3] (nejlepší nástroj pro rychlé prototypování v automobilovém průmyslu).

2.1 Rapid Prototyping

Na obrázku 2.1 [2] lze vidět porovnání tradičního přístupu v návrhu systému a rapid prototypingu. Návrh řídicího systému (ŘS) **klasickou metodou** vede na potřebu mít typicky k dispozici několik týmů inženýrů. Jakmile je týmem pro návrh algoritmu řízení vypracován funkční simulační model ŘS, tým softwarových specialistů manuálně implementuje algoritmus na simulátor, a poté specifikuje požadavky na hardware.



Obrázek 2.1: Tradiční návrh vs. rapid prototyping

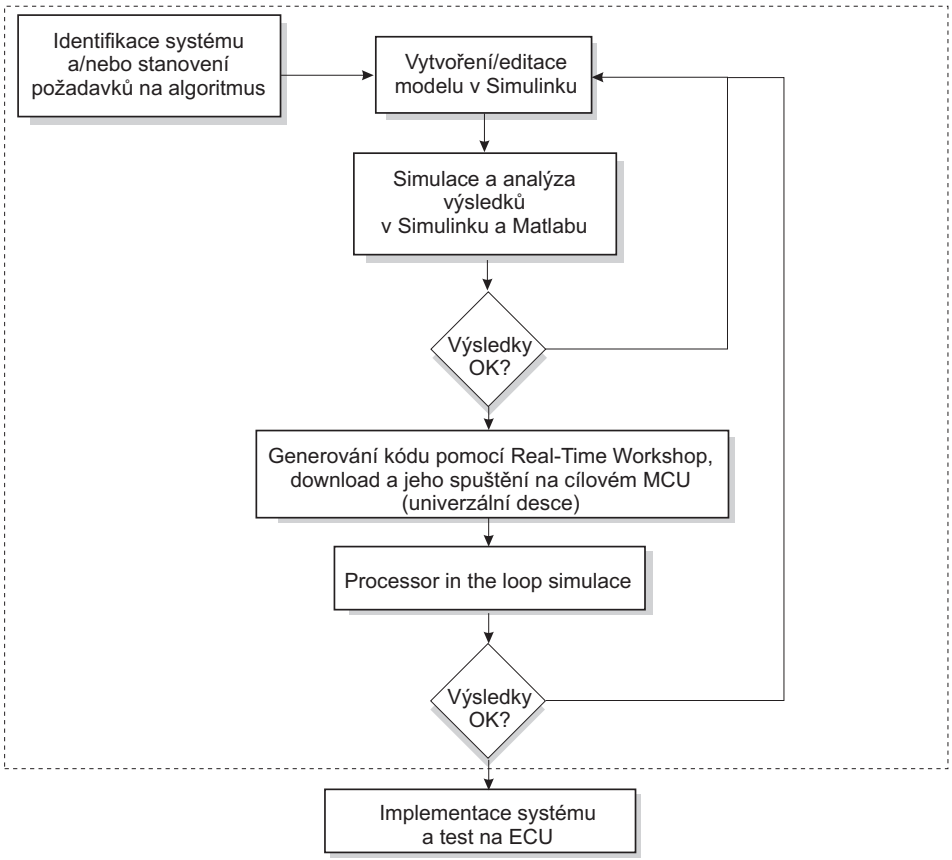
Implementace řídicího algoritmu je však prováděna pomocí jiných nástrojů, než které byly používány pro simulaci modelu řídicího systému. Ztratí se tak vazba mezi si-

mulačním modelem a výslednou aplikací řízení na cílovém hardware. Další nutné modifikace řídicího algoritmu (např. nastavování parametrů regulátoru) se poté stávají složitými. Nevýhodou jsou časově náročné změny ve specifikaci algoritmu řízení a jeho implementaci, veškeré změny jsou prováděny ručně (týmem specialistů).

Naproti tomu **rapid prototyping** (rychlé prototypování) využívá automatického generování kódu. Použití automaticky generovaného kódu umožňuje provádět návrh řídicího systému vcelku. Není zde oddělena část návrhu samotného algoritmu řízení a implementace na hardware. Zachová se tak vazba mezi simulačním modelem a cílovou aplikací. Generátor kódu generuje kód ze simulačního modelu. Vygenerovaný kód je přitom dobře čitelný, získán v kratším čase, než je tomu v případě ručně psaného kódu a oprostěn od možných zdrojů chyb.

Tým pro návrh řídicího systému se může soustředit na návrh algoritmu, protože vytvoření kódu není časově náročnou operací. Stejně tak případné změny v požadavcích na řídicí algoritmus mohou být rychle propagovány do výsledného kódu díky jeho automatickému generování.

Jedním z nástrojů využitelných při rapid prototypingu řídicích systémů je i processor in the loop simulace.



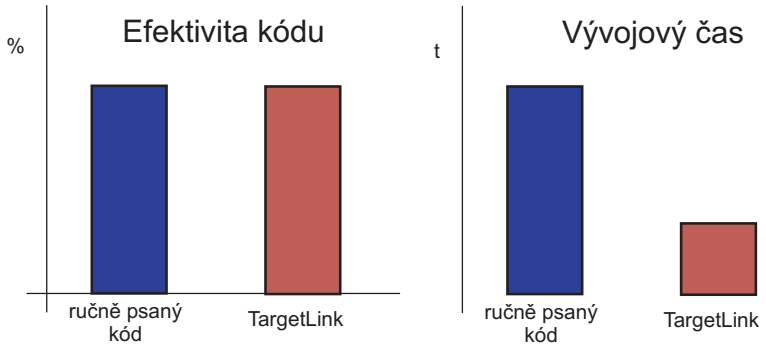
Obrázek 2.2: Rapid prototyping s využitím PIL simulace

Obrázek 2.2 ukazuje strukturu možného způsobu rychlého prototypování s využitím Matlabu, Simulinku, Real-Time Workshopu a vlastního řešení PIL simulace.

2.2 Související práce

Kompletní komerční systém zabývající se problematikou rapid prototypingu a automatického generování kódu je systém fy dSpace [3]. Produkt TargetLink je generátorem produkčního C kódu z Matlab/Simulink/Stateflow modelů. Umožňuje využívat MIL, SIL, PIL i HIL simulaci. Obsahuje block set, knihovnu TargetLink bloků, který rozšiřuje možnosti standardních bloků Simulinku.

Na obrázku 2.3 je možné vidět, že automaticky generovaný kód z TargetLinku dosahuje kvalit ručně psaného kódu, a přitom je získán v mnohem kratším čase [3].



Obrázek 2.3: Ručně psaný kód vs. automaticky generovaný kód

TargetLink podporuje rapid prototyping při vývoji řídicích systémů. Firma dSpace pro tuto podporu vyvinula několik hardwarových systémů, příkladem může být systém RapidPro.

Možnosti systému fy dSpace se staly inspirací pro rozšíření PEERT [1] o PIL.

Kapitola 3

Použité nástroje

Kromě nástrojů Matlab Simulink a Real-Time Workshop je pro automatické generování kódu pro vestavné aplikace využíván nástroj Processor Expert a target PEERT. Knihovna PESLIB zprostředkovává komunikaci mezi programy Simulink a Processor Expert a lze za její pomoci nastavovat jednotlivé periferie MCU.

3.1 Vývojové prostředí Matlab

Ve vlastní implementaci PIL simulace je z vývojového balíku Matlab využíván nástroj Simulink pro simulaci modelů, Real-Time Workshop (RTW) pro generování kódu soustavy (vygenerovaný kód nemusí být efektivní) a Real-Time Workshop Embedded Coder (RTW EC) pro generování kódu regulátoru (efektivní kód).

Při implementaci PIL simulace byl využíván Matlab verze 7.0.1 (R14SP1) a 7.0.4 (R14SP3).

3.1.1 Simulink

Simulink umožňuje modelovat a simulovat běh dynamických systémů. Nejdříve je vytvořeno simulační blokové schéma pro systém pomocí Simulink model editoru. Blokové schéma je grafickou reprezentací matematického modelu dynamického systému. Poté lze simulovat běh modelu po uživatelem specifikovaný čas [4].

V práci byl používán Simulink verze 6.1 (R14SP1) a 6.3 (R14SP3).

3.1.1.1 S-funkce

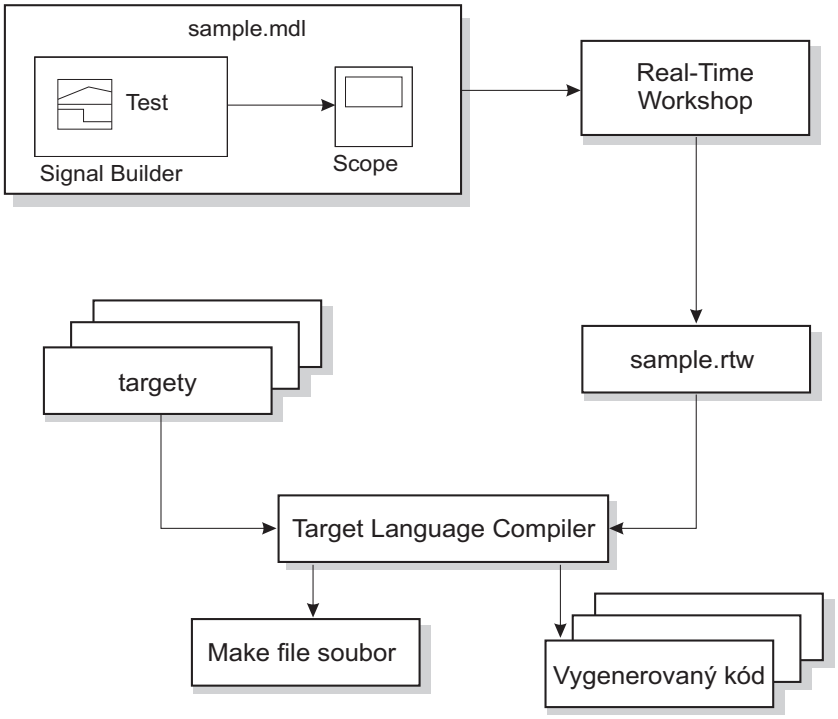
Model dynamického systému v Simulinku se skládá ze simulačních bloků a linek (signálů). **S-funkce** dovolují vytvořit uživatelské bloky pro Simulink. Jejich pomocí lze

vykonávat vlastní algoritmus či přistupovat k externím rozhraním. S-funkce může být napsána v programovacím jazyku Matlab, C, C++, Ada nebo Fortran. Využívá speciální API, které umožňuje S-funkci spolupracovat s řešitelem (solverem) Simulinku. Např. pro inicializaci bloku je volána funkce *mdlInitializeSizes* a pro výpočet výstupu bloku funkce *mdlOutputs*, kde uživatel implementuje svůj algoritmus.

Nové bloky pro block set targetu PEERT byly implementovány jako S-funkce. Více o S-funkcích je možné se dovědět v její dokumentaci, viz [5].

3.1.1.2 Generování kódu v Simulinku

Obrázek 3.1 popisuje, jakým způsobem je generován kód z modelu v Simulinku.



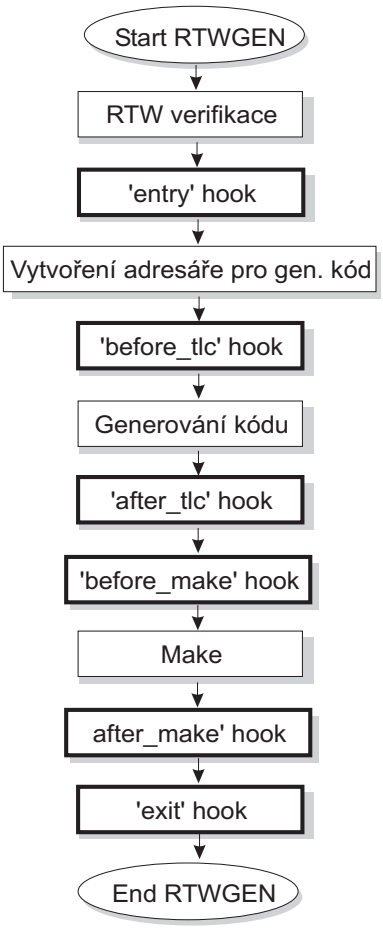
Obrázek 3.1: Generování kódu ve vývojovém prostředí Matlab

Nástrojem Real-Time Workshop (RTW) je z modelu *modelname.mdl* vytvořen soubor *modelname.rtw*. Pak při zvoleném targetu je pomocí Target Language Compileru (TLC) vygenerován výsledný kód spolu s make file souborem. *Modelname.rtw* je textovým popisem simulačního modelu a TLC převádí tento popis do kódu.
O struktuře *modelname.rtw* lze zjistit více v dokumentaci TLC [6]. RTW je blíže popisován v další části této kapitoly (viz 3.1.2). Pojem *target* je zde také vysvětlen (viz 3.1.2.1).

Target Language Compiler [6] je vnitřní částí RTW. Umožňuje během procesu generování kódu z modelu kód modifikovat podle uživatelových požadavků. Využívá

k tomu TLC soubory, textové soubory, které explicitně říkají, jakým způsobem bude kód z modelu vytvořen. TLC je tedy takovým textovým procesorem, jenž využívá TLC soubory a *modelname.rtw* k vygenerování C kódu. Editací TLC souboru lze změnit způsob, jakým bude vytvořen kód pro daný blok v modelu.

Během generování kódu je generován kód pro každý blok v simulačním modelu. V procesu automatického generování kódu existuje několik míst (tzv. hooků), ve kterých lze volat uživatelské funkce. Takto lze využívat vývojové nástroje specifické pro daný hardware.



Obrázek 3.2: Proces generování kódu

Na obrázku 3.2 je blokovým schématem tento proces popsán. Tučnými rámečky jsou zvýrazněna místa, ve kterých lze měnit způsob generování kódu. Existuje několik "hook souborů", kterými lze ovlivnit výsledný kód. Velmi důležitým je soubor *STF_make_rtw_hook.m* implementující funkci, kterou je možné spouštět uživatelský kód ve všech tučně označených místech v obrázku 3.2. STF je zkratka pro systémový TLC soubor (viz tabulka 3.1). Pro název souboru *STF_make_rtw_hook.m* je využit název systémového TLC souboru bez jeho přípony.

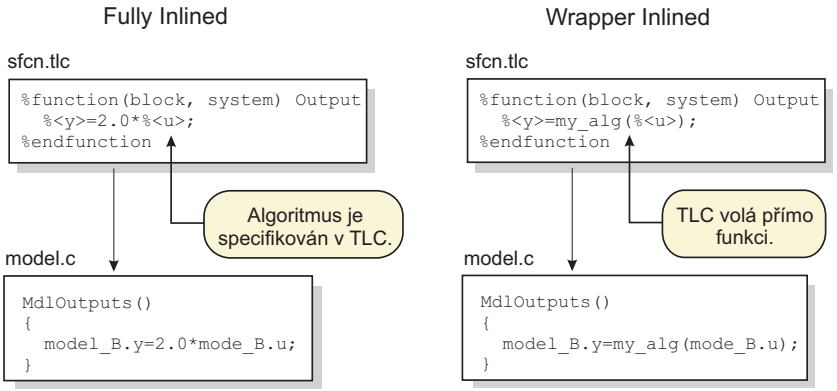
Tento soubor je vždy automaticky volán při vytváření kódu. Není nutné implementovat všechny hooky, ale pouze ty potřebné. Hook *'entry'* může být využit pro inicializaci, v době, kdy ještě není žádný kód vygenerován. Pro volání externích nástrojů je možné použít *'before_tlc'*, *'after_tlc'*, *'before_make'*, *'after_make'* a *'exit'*. Více viz dokumentace Real-Time Workshop Embedded Coder [7].

Pro každý blok v modelu je generován kód popsaný TLC souborem. Pro uživatelské S-funkce je generován kód podle jejich typu. Uživatelské S-funkce mohou být:

- vkládané (inlined)
  - plně vkládané (fully inlined)
  - vkládané s voláním funkcí (wrapper inlined)

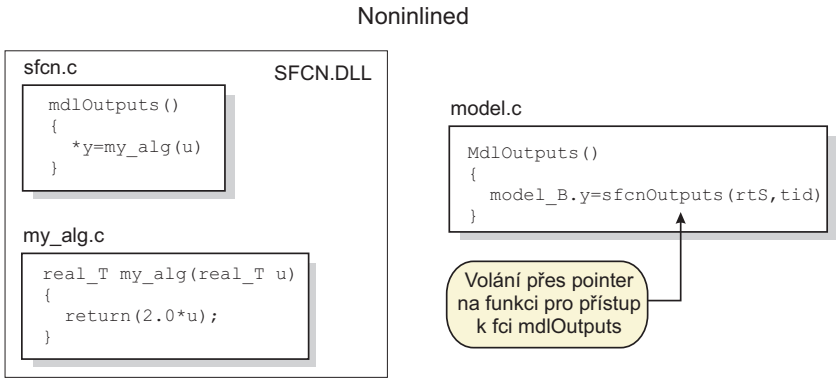
- nevkládáné (noninlined)

Vkládané S-funkce jsou dvojího druhu. U plně vkládaných je celá implementace výsledného generovaného kódu vložena v TLC souboru dané S-funkce. Druhý typ se odlišuje tím, že místo vkládání kódu celého algoritmu volá C funkce (viz porovnání na obrázku 3.3). Ke každé vkládané S-funkci přísluší TLC soubor, pomocí kterého se při generování kódu nahradí simulační kód. Existuje několik výhod proč používat vkládané S-funkce s voláním funkcí (viz [5]). Obě verze jsou efektivnější než nevkládané S-funkce.



Obrázek 3.3: Vkládané S-funkce

Jestliže S-funkce nemá svůj TLC soubor, pak je nazývána S-funkcí nevkládanou. Její použití ve vygenerovaném kódu vede na méně efektivní kód. Způsob, jakým jsou volány její vnitřní funkce, je ilustrován na obrázku 3.4.

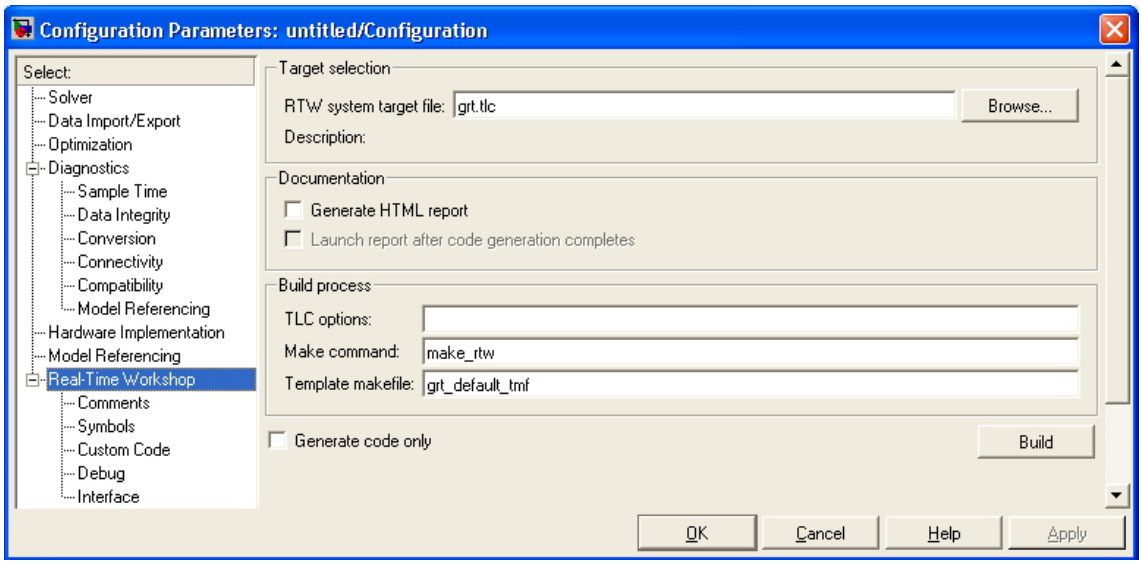


Obrázek 3.4: Nevkládané S-funkce

3.1.2 Real-Time Workshop (RTW, RTW EC)

Real-Time Workshop (RTW) generuje C kód ze simulačního modelu. **Real-Time Workshop Embedded Coder** (RTW EC) je využíván pro generování optimalizovaného produkčního kódu.

RTW EC je distribuován jako samostatný produkt, rozšiřuje možnosti RTW. Byl vyvinut za účelem generování C kódu pro vestavné systémy, kde je efektivita a čitelnost vygenerovaného kódu velmi důležitá.



Obrázek 3.5: Nastavení Real-Time Workshop

Parametry RTW lze nastavovat v konfiguračním dialogovém okně modelu v Simulinku (viz obrázek 3.5). Význam jednotlivých položek je popsán v manuálu [2], nastavení používaná při podpoře PIL simulace jsou popsána dále.

3.1.2.1 Definice targetu a jeho implementace

Target je objekt Simulinku, jímž je v simulačním programu reprezentován cílový procesor. Target je tedy závislý na typu procesoru. Targetem je specifikován CPU včetně některých jeho nastavení jako je použitý programovací jazyk, nástroje pro práci s vygenerovaným kódem či operační systém cílového hardwaru.

Obvykle přísluší ke každému targetu block set. **Block set** představuje knihovnu simulačních bloků použitých pro práci s daným procesorem. Jednotlivé bloky představují periferie daného hardware nebo jsou bloky pro komunikaci. Bloky jsou implementovány jako S-funkce a je pomocí nich definováno simulační chování bloku. K S-funkci může být přiřazen TLC soubor, kterým je předepsáno, jaký bude generován kód pro daný blok.

Target vystupuje v Simulinku jako jeden objekt (soubor), který je vybrán v nabídce *"Target Selection"*, viz obrázek 3.5. Nicméně vybíraný TLC soubor je pouze jedním z mnoha součástí, které target definují.

Target obsahuje následující součásti:

- soubory zdrojových C kódů
  zdrojové soubory využívané vygenerovaným kódem z modelu
- řídicí soubory
  - systémový TLC soubor
    je hlavní TLC soubor, který je obsažen v každém targetu
  - soubory řídicí vytvoření make file souboru nebo projektu (např. projekt pro IDE Metrowerks CodeWarrior)
  - "hook" soubory
    volitelné TLC nebo m soubory pro uživatelské úpravy způsobu generování kódu
- soubory předvoleb
  umožňují vytvořit datové objekty, které definují a ukládají parametry daného targetu
- další soubory

V dokumentaci k RTW EC [8] je možné se dovědět více o výše zmíněných typech souborů.

3.1.2.2 Druhy targetů

Zde uvedené základní rozdělení targetů je vytvořeno podle [8]. Diskutované typy targetů nejsou vzájemně jednoznačné, může být vytvořen target s rysy více druhů targetů.

Základní target (baseline) slouží ke generování kódu pro vestavný (embedded) procesor. Obvykle integruje do RTW EC prostředí některých vývojových nástrojů (compiler/linker/debugger). Slouží jako startovací bod pro tvorbu jiných typů targetů. Použitý kód by proto měl být dobře čitelný, lehký k pochopení, dobře komentovaný a dokumentovaný.

Produkční target (turnkey production) je vytvořen pro specifický procesor, s kódem využívajícím periferie procesoru. Předpokladem je, že target již nebude dále upravován. Proto je důraz kladen na snadné použití targetu, není důležitá čitelnost kódu.

Podobně **target pro PIL simulaci** je využíván pro processor in the loop simulaci. PIL simulace je mimo jiné využívána pro validaci generovaného kódu.

3.1.2.3 Targety produkované firmou MathWorks

V tabulce 3.1 je uveden seznam targetů dodávaných firmou MathWorks. Jak je vidět v této tabulce, existuje pouze několik targetů. Není tak zdaleka pokryta většina možných CPU.

Systémový TLC soubor	Popis targetu
asap2.tlc	ASAM-ASAP2 Data Definition Target
c166.tlc	Embedded Target for Infineon C166(R) Microcontrollers
ert.tlc	RTW Embedded Coder
grt.tlc	Generic Real-Time Target
grt_malloc.tlc	Generic Real-Time Target with dynamic memory alloc
hc12.tlc	Embedded Target for Motorola HC12 and CodeWarrior (real-time)
mpc555exp.tlc	Embedded Target for Motorola MPC555 (algorithm export)
mpc555pil.tlc	Embedded Target for Motorola MPC555 (processor-in-the-loop)
mpc555rt.tlc	Embedded Target for Motorola MPC555 (real-time)
osekworks.tlc	OSEK Target for WRS OSEKWorks Implementation
proosek.tlc	OSEK Target for 3Soft ProOSEK Implementation
rsim.tlc	Rapid Simulation Target
rtwin.tlc	Real-Time Windows Target
rtwsfcn.tlc	S-function Target
ti_c2000_ert.tlc	Embedded Target for TI C2000 DSP (ERT)
ti_c2000_grt.tlc	Embedded Target for TI C2000 DSP (GRT)
ti_c6000.tlc	Embedded Target for TI C6000 DSP (GRT)
ti_c6000_ert.tlc	Embedded Target for TI C6000 DSP (ERT)
tornado.tlc	Tornado (VxWorks) Real-Time Target
xpctarget.tlc	xPC Target
xpctargetert.tlc	xPC Target (ERT)

Tabulka 3.1: Targety fy MathWorks

Bloky periferií těchto targetů nesimulují skutečné chování periferie. Pro simulaci jsou pouhým průchozím blokem, tj. co je na vstupu bloku, to se přenese na jeho výstup.

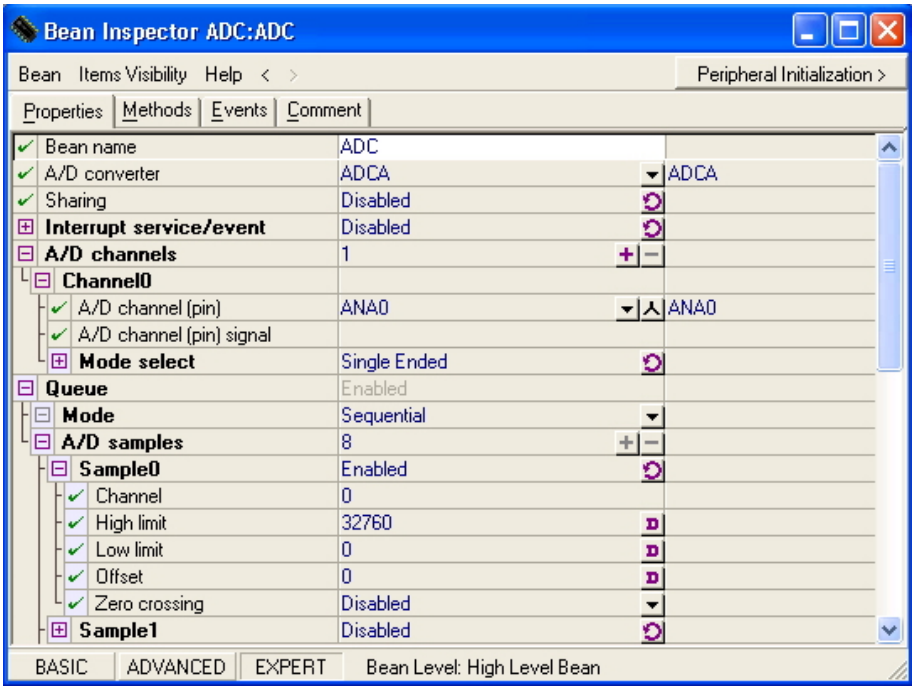
Nastavení hardware nejsou nijak kontrolována. Tyto nevýhody stávajících targetů přispívají k potřebě vytvářet nové uživatelské targety.

3.2 Processor Expert

Processor Expert (PE) [9] je komponentně orientované vývojové prostředí podporující mikrokontroléry Freescale pro rychlý vývoj vestavných aplikací a umožňuje generovat produkční C kód. PE obsahuje informace o podporovaných MCU a jejich periferií.

Pomocí tzv. *Embedded Beans* je reprezentováno jádro procesoru i jeho periferie. Specialisty je napsán kód beanů. Podle použitých beanů generuje PE kód, přitom používá vestavěnou bázi znalostí o možných nastaveních hardwaru.

Uživatelské rozhraní k beanům je tvořeno *vlastnostmi* (properties), m*etodami* (methods) a *událostmi* (events) (viz záložky na obrázku 3.6). Uživatelem provedená nastavení jsou verifikována, v případě nesprávného nebo nemožného nastavení je na toto uživatel upozorněn.

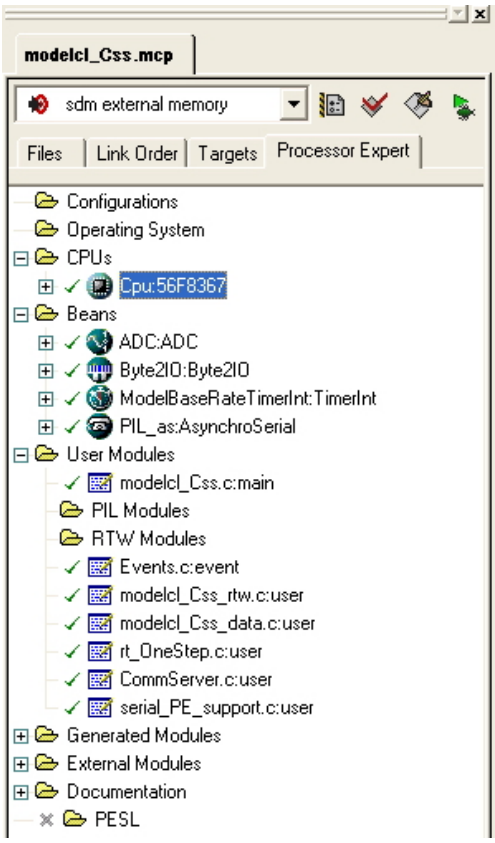


Obrázek 3.6: Okno Bean Inspector

Díky *vlastnostem* lze nastavit hardware bez detailní znalosti samotného hardware či jeho registrů. Všechny nastavení hardware může uživatel měnit, žádné z nich není přednastavené bez možnosti jeho změny. *Metody* představují interface pro uživatelský kód. *Události* jsou používány k ošetření přeřušení od periferie.

Vývojový pracovník nemusí být tedy do detailů seznámen s registry procesoru. Pro nastavení periferií stačí nastavit několik jejích parametrů a využívat její metody a

události.



Obrázek 3.7: Processor Expert projekt

PE projekt lze vytvořit jednoduše. Metodou drag&drop se přidají do projektu beany a nakonfigurují se pomocí *Bean Inspectoru*. Na obrázku 3.7 je možné vidět ukázkový projekt. Obsahuje bean procesoru a čtyři nakonfigurované periferie (v sekci *Beans*). Uživatelské zdrojové kódy využívané v projektu jsou obsaženy v části *User Modules*.

Shrnutí výhod při použití nástroje Processor Expert [9]:

- komponentně orientovaný systém pro tvorbu vestavných aplikací s 8, 16, 32 bitovými mikroprocesory
- aplikace je vytvořena z komponent *Embedded Beans*:
  - ty představují jednotlivé periferie MCU
  - jejich inicializace a práce s nimi je převedena na práci s objekty (podobně jako v OOP, využití vlastností, metod a událostí)
  - beany je možné vytvářet ve více programovacích jazycích
  - beany je snadné vytvářet použitím *Beans Wizardu*
- radikálně zrychluje vývoj aplikací použitím:
  - snadno využitelné knihovny komponent
  - expertní báze znalostí pro nastavování periferií
  - generovaného optimalizovaného zdrojového kódu pro periferie

- jednoduché správy uživatelského a generovaného kódu
- uživatelských komponent vytvořených v *Beans Wizardu*
- spolupráce s jinými vývojovými nástroji (IDE, kompilátory, emulátory atd.)

3.3 PEERT, PESLIB

Protože se nástroje Matlab Simulink a Processor Expert vzájemně doplňují, byl vyvinut *Processor Expert Embedded Real-Time Target* (PEERT) [1]. Ten integruje prostředí Processor Expert do Matlab Simulinku. Matlab byl tímto rozšířen o možnosti nastavování hardware kontrolérů. Mohou tak být využity nejlepší vlastnosti obou nástrojů pro rychlý vývoj aplikací.

PEERT obsahuje block set, díky kterému lze nastavovat periferie hardware pomocí bloků v Simulinku. *PE block set* umožňuje nastavovat veškeré vlastnosti hardware, čímž se odlišuje od targetů fy Mathworks.

PEERT obsahuje 3 části:

- PE block set - knihovna bloků periferií
- komunikační knihovna PESLIB
- RTW Embedded Coder target

PE block set Obsahuje bloky periferií MCU pro Simulink, např. AD převodník, PWM, časovače a další. Blok v knihovně odpovídá danému beanu v PE. Blok je implementován jako S-funkce a simuluje chování dané periferie. Například blok 12 bitového AD převodníku (ADC bean) vrací hodnotu s 12 bitovým rozlišením. Události jsou představovány tzv. function-call portem bloku.

Knihovna PESLIB Zprostředkovává komunikaci mezi Simulinkem a PE projektem pomocí Microsoft Component Object Model interface (COM) [10]. Zajišťuje sychronizaci modelu a PE projektu. Změny v modelu vyvolávají odpovídající změny v projektu, např. při vložení bloku periferie do simulačního modelu je vložen bean do projektu. Dvojklikem na bloku v modelu je otevřeno menu pro nastavení periferie, stejného vzhledu jako v samotném PE (viz *Bean Inspector* na obrázku 3.6).

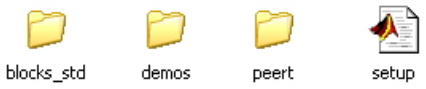
RTW Embedded Coder target Byl vytvořen pro generování C kódu. Pomocí TLC souborů bloků periferií je určeno, jaký pro ně bude generován kód. V TLC souborech jsou použity funkce PE API, bloky periferií se tak stávají nezávislé na zvoleném typu

MCU. Periodické tasky jsou prováděny na základě přerušení časovače. Asynchronní události jsou spouštěny na základě přerušení dané periferie.

PEERT target používá *peert_make_rtw_hook.m*. Tento soubor je využit pro přizpůsobení procesu generování kódu v RTW podle uživatelských potřeb (viz obrázek 3.2). Je využíván pro konfiguraci beanů, spouští generování kódu beanů a vygenerovaný kód z RTW spojuje s kódem PE do jediného PE projektu. Výsledkem automatického generování kódu pomocí PEERT targetu je PE projekt (viz obrázek 3.7), který je připraven pro kompilaci cílové aplikace.

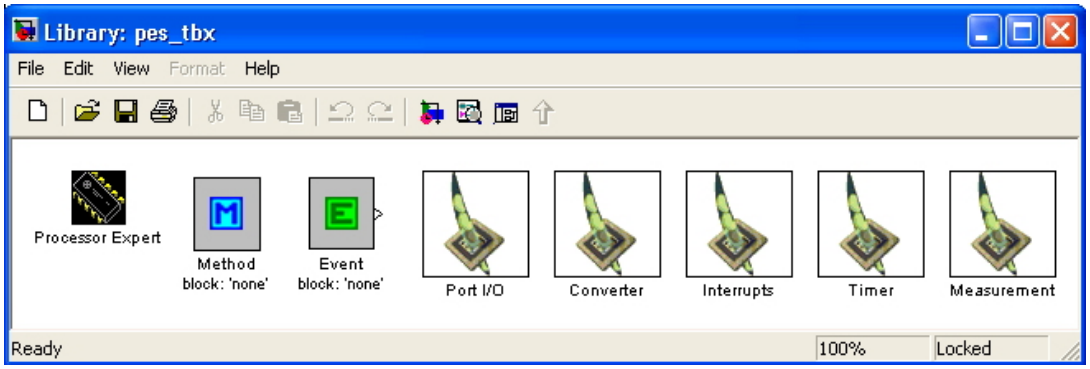
Instalace PEERT

PEERT je v současné době distribuován jako zip soubor. Po jeho rozbalení jsou vytvořeny adresáře *block_std*, *demos* a *peert* (obrázek 3.8). Pro přidání cest do Matlabu je nutné nejdříve spustit konfigurační skript *setup.m*.



Obrázek 3.8: Obsah PEERT

Jednotlivé bloky PE block setu jsou obsaženy v adresáři *block_std*. Vzhled block setu v Simulinku lze vidět na obrázku 3.9. V jeho podadresáři *tlc_c* jsou TLC soubory příslušející k blokům. Zdrojové soubory S-funkcí, které tvoří bloky, jsou umístěny v podadresáři *src*.



Obrázek 3.9: Block set targetu PEERT

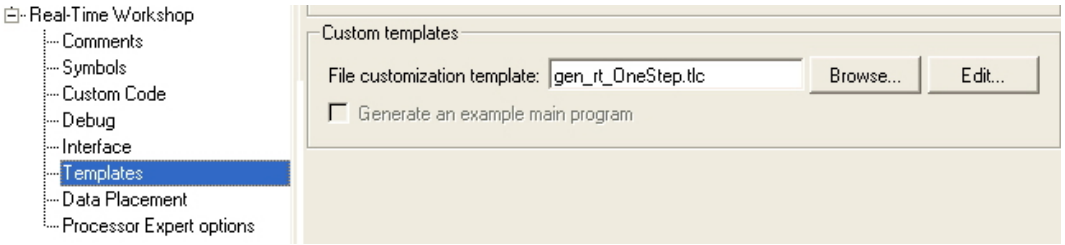
Složka *demos* obsahuje demonstrační modely pro práci s PEERT. V adresáři *peert* jsou soubory pro RTW Embedded Coder target.

Mezi nejdůležitější soubory v adresáři *peert* patří:

- *peert_make_rtw_hook.m*
  Hlavní soubor, kterým je modifikován proces generování kódu.

- *peert.tlc*
  Systémový TLC soubor, kterým je target definován (viz tabulka 3.1).

- *gen_rt_OneStep.tlc*
  Upravený soubor *example_file_process.tlc*. Slouží pro vytváření uživatelských souborů při procesu generování kódu. Ještě před vytvářením kódu z modelu v Simulinku je tento soubor nutné nastavit v konfiguračním dialogu RTW v nabídce *Templates - File Customization Template* (viz obrázek 3.10).



Obrázek 3.10: Šablona pro vytváření souborů

  Do generovaného kódu se touto šablonou přidají soubory:

  - *modelname_Model_Api.h*, kde *modelname* označuje název modelu
  - *RTW_Model_API.h*
  - *rt_OneStep.c*

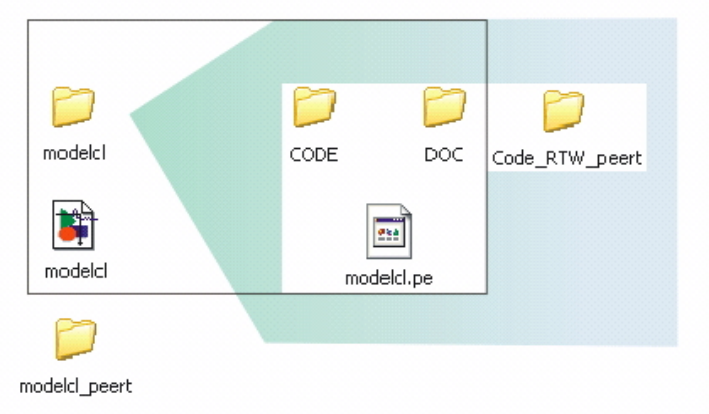
  Pokud by nebyl soubor *gen_rt_OneStep.tlc* použit, vygenerovaný kód modelu by nebyl kompletní a nebyl by ani funkční.

Složka *peert* kromě těchto uvedených souborů obsahuje ještě několik dalších TLC a m souborů.

Simulační model s PEERT

Každý model v Simulinku, který je navrhován pro PEERT target, musí obsahovat blok *Processor Expert* (viz obrázek 3.9). Tento blok je obvykle vkládán do modelu jako první (musí být vložen před jakýmkoliv blokem periferie). Po jeho vložení je vytvořen adresář *modelname* s podadresáři *CODE* a *DOC*. V adresáři *modelname* je přitom vytvořen PE projekt s názvem *modelname.pe* (*modelname* je název modelu). Vytvořené adresáře a PE projekt pro ukázkový model *modelcl.mdl* lze vidět orámované na obrázku 3.11.

Další adresáře jsou vytvářeny při generování kódu. Real-Time Workshopem vytvořený kód je uložen v adresáři *modelname_peert*. Target PEERT využívá kód z adresáře *Code_RTW_peert* v *modelname*.



Obrázek 3.11: Složky modelu pro PEERT

3.4 Překladače a hardware

Dalšími používanými nástroji při vytváření podpory PIL simulace je *Metrowerks CodeWarrior* a *Microsoft Visual Studio 2003*.

Překlad vygenerovaného kódu

IDE Metrowerks CodeWarrioru bylo využíváno jako překladač vygenerovaného kódu pro mikroprocesor. Důležitým nastavením pro bezchybný překlad kódu jsou *Access Paths* - cesty, ve kterých jsou hledány zdrojové soubory při překladu kódu. Lze je nastavit v nabídce *Edit - sdm external memory Settings - Target - Access Paths*.

Používaná verze IDE byla CodeWarrior 56800/E Hybrid Controllers version 7.1.

Překlad S-funkce a vygenerovaného kódu

Microsoft Visual Studio 2003 bylo používáno pro překlad zdrojových souborů S-funkcí. Novější verze využita být nemohla, protože Matlab verze 7.0.1 a 7.0.4 podporuje pouze verzi 2003.

Nastavení překladače v Matlabu se provádí příkazem *mex -setup*. Jeho zavoláním prohledá Matlab instalované překladače na PC a podporované poté nabídne k výběru. Příkaz *mex sfcname.c* přeloží zdrojový soubor *sfcname.c*. Ve Windows a při použití výše zmíněných verzí Matlabu má soubor přeložené S-funkce příponu *dll*.

Visual Studio bylo využíváno i pro debug S-funkce. Aby mohla být S-funkce debugovaná, musí být přeložena příkazem *mex* s parametrem *-g*. Ve Visual Studiu je pak nutné připojit se k procesu Matlabu přes nabídku *Debug - Processes*, na procesu *matlab.exe* vybrat *Attach* při nastaveném typu programu *Native*.

Dále bylo využíváno jako překladač vygenerovaného kódu z modelu soustavy pro simulační target.

Hardware

Při vývoji PIL simulace byl využíván vývojový kit MC56F8367EVMUM s procesorem MCU 56F8367. Tento procesor je jedním z řady 56800/E 16 bitových tzv. hybrid kontrolérů. Ty obsahují DSP a MCU periferie a poskytují tak "system-on-a-chip" řešení pro vestavné aplikace. Více informací o vývojové desce a procesoru lze získat na [11].

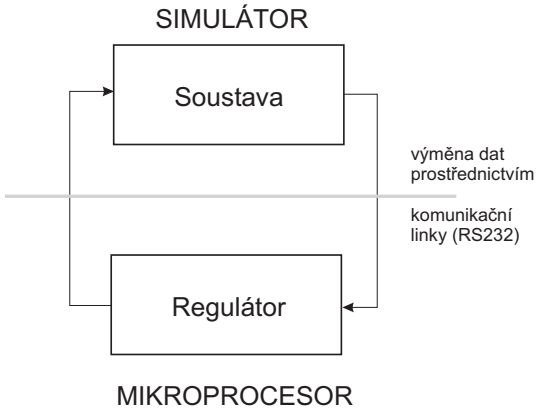
Kapitola 4

Architektura PIL

V úvodu této kapitoly jsou popsána principielní řešení návrhu processor in the loop simulace a možné návrhy modelů využitelných v PIL simulaci. Následuje přehled existujících řešení a autorem provedená analýza možných řešení PIL simulace při požadavku její práce v reálném čase.

4.1 Úvod do architektury PIL

Při PIL simulaci je využívano zapojení target mikroprocesoru se simulátorem v uzavřené smyčce (viz obrázek 4.1). Simulátor může být speciální hardware (simulační target), nebo host systém (PC) se simulačním softwarem. Možnosti a vlastnosti obou variant jsou porovnány v kapitole 4.3.7.



Obrázek 4.1: Schéma uzavřené smyčky

Obecně lze říci, že simulace na host PC bude mít delší a méně deterministickou časovou odezvu. Věrohodnost simulace je dána tím, zda probíhá v reálném čase:

- PIL simulace neprobíhá v reálném čase

1. simulátor (např. Simulink) spustí jeden krok simulace
2. simulačním blokem pro odeslání dat budou odeslána data do CPU
3. procesor přijme data a provede výpočet svého regulačního algoritmu
4. data z procesoru se odešlou pomocí komunikačních funkcí a simulátor přijme data simulačním blokem pro příjem dat
5. skok na bod 1. (tj. cyklické opakování bodů 1. - 4.)

- PIL simulace pracující v reálném čase
  - krok simulace spouští mikroprocesor na základě přerušení od časovače
  - simulátor v tomto případě čeká na příchod dat

Předložená práce se zaměřuje na real-time PIL simulaci.

Návrh modelů pro PIL simulaci lze řešit dvěma způsoby (více viz [8]):

- Více-modelový přístup
  - využívá separované modely pro simulaci MIL a generování kódu pro PIL
  - jeden model slouží k simulaci uzavřené smyčky soustavy s regulátorem (neobsahuje bloky periferií)
  - druhý model je využitelný pouze pro generování kódu, obsahuje pouze algoritmus regulátoru s bloky ovladačů nutnými pro PIL simulaci
- Jedno-modelový přístup
  - využívá stejný model pro simulaci MIL i PIL (generování kódu pro PIL a spuštění simulace)

Při použití více-modelového přístupu je uživatel nucen provádět několik kroků před samotným generováním kódu. Po provedené simulaci v Simulinku musí vykopírovat bloky regulátoru, přidat bloky pro komunikaci a teprve poté generuje kód.

Proto se předložená práce věnuje jedno-modelovému přístupu, tj. všechny nutné úpravy modelu pro PIL jsou provedeny automaticky.

4.2 Přehled existujících řešení

TargetLink - PIL simulační mód

Jedním ze simulačních módů produktu TargetLink je PIL simulace. Vygenerovaný kód je spuštěn na CPU (vývojové desce) a propojen s PC se simulačním softwarem. V případě rozdílných výsledků PIL a SIL simulace je jako nejpravděpodobnější chyba uváděn problém s kompilátorem pro CPU nebo se samotným CPU (viz [3]).

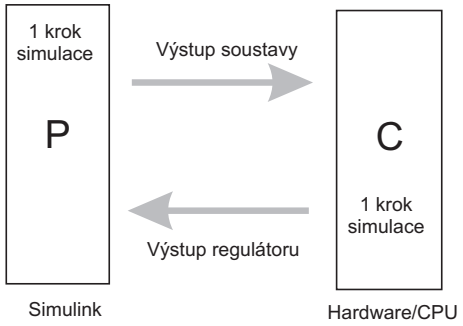
PIL kosimulace targetu mpc555pil

Processor in the loop target *mpc555pil* zprostředkovává kosimulaci, při které Simulink modeluje soustavu a vygenerovaný kód regulátoru je spuštěn na procesoru. Tento target je rozšířenou verzí základního ERT targetu (viz tabulka 3.1).

Real-Time Workshop Embedded Coder generuje efektivní kód modelu regulátoru pro CPU. Model soustavy je simulován v Simulinku, žádný kód z něj generován není. Simulace neprobíhá v reálném čase.

Jeden cyklus PIL simulace sestává z akcí (viz obrázek 4.2):

- Simulink vypočítá výstup soustavy za dobu jednoho vzorkovacího intervalu
- vypočtený výstup soustavy (P) je odeslán komunikačním kanálem k CPU
- jakmile CPU přijme signál (hodnotu) od modelu soustavy, spustí algoritmus regulátoru (C) na dobu jedné periody
- regulátor odešle svoji hodnotu akčního zásahu komunikačním kanálem zpět do modelu soustavy

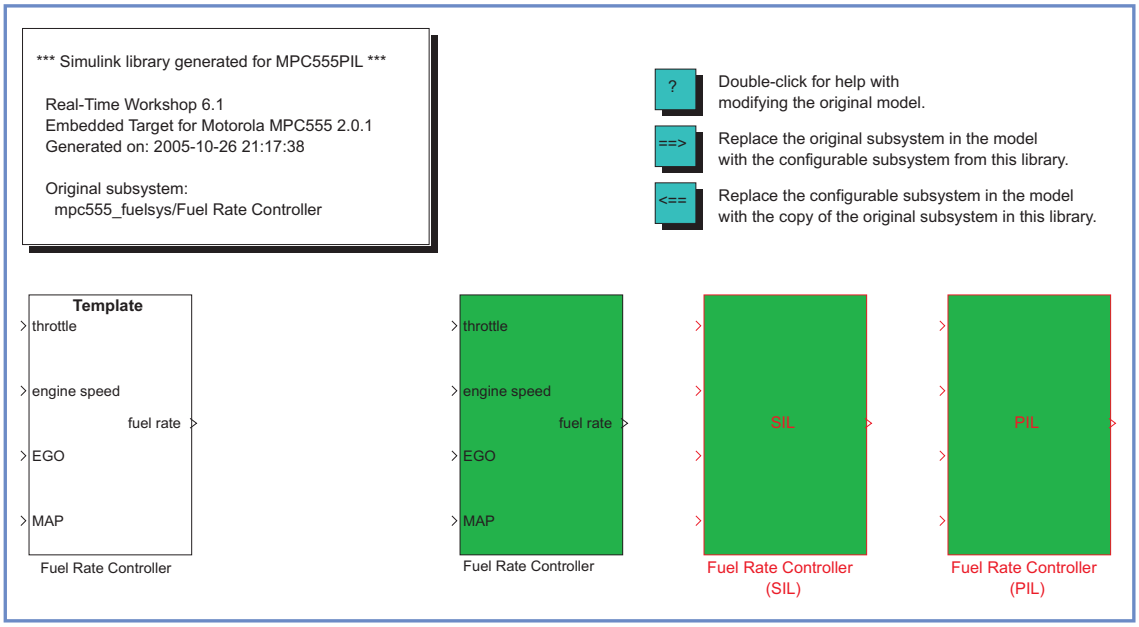


Obrázek 4.2: PIL kosimulace targetu mpc555pil

Pro komunikaci je využívána sériová linka RS232 nebo CAN. Cílový mikroprocesor, pro který je kód generován, musí být MPC555 nebo MPC56x (561-6). Jako překladač je možné využít Wind River verze 5.2.1 nebo Metrowerks CodeWarrior for Embedded PowerPC (verze 8.1).

Spolu s kódem pro CPU je procesem generování kódu regulátoru vytvořena i knihovna (obrázek 4.3) obsahující:

- původní subsystém regulátoru používaný při simulaci v Simulinku (MIL)
- blok S-funkce implementující regulátor pro software in the loop simulaci (SIL)
  - při SIL simulaci je místo použití původního bloku spouštěn C kód S-funkce
- subsystém regulátoru pro PIL simulaci
  - tento subsystém komunikuje s vygerovaným kódem na CPU
- blok typu "Configurable subsystem"
  - tímto blokem je nahrazen původní blok regulátoru
  - pomocí něho lze vybrat jeden ze tří bloků z vygenerované knihovny podle druhu žádané simulace, tj. MIL, SIL nebo PIL
  - na obrázku 4.3 označen popiskem *Template*



Obrázek 4.3: Vygenerovaná knihovna při použití mpc555pil

Knihovna také obsahuje tlačítko pro zobrazení nápovědy systému Matlab. Dalšími tlačítky lze v modelu zaměnit původní subsystém za blok typu "Configurable subsystem" a opačně "Configurable subsystem" za původní subsystém.
Jakmile je generování kódu dokončeno, PIL target automaticky spustí kompilaci a download kódu na CPU (využívá k tomu *Download Control Panel*, viz dokumentace RTW [2]).

Po každé změně modelu regulátoru musí být pro CPU znovu vygenerován kód. PIL target nemůže využívat bloky periferií z block setu targetu Embedded Target for Motorola MPC555. PIL kosimulace neprobíhá v reálném čase.

Více informací je možné získat v dokumentaci k targetu, viz [12].

4.3 Možné způsoby nového vlastního řešení

Možné varianty PIL simulace je možné rozčlenit podle obrázku 4.4.

růst schopnosti práce v reálném čase →

co se simuluje	na čem se simuluje	
	Host (PC s Windows)	Simulační target (hardware + RTOS)
Software	1	3
Model	2	4

↓ růst efektivity

Obrázek 4.4: Rozdělení variant PIL simulace

Jednotlivé způsoby (označeny čísly 1 až 4) jsou rozděleny podle toho, co je simulováno při PIL a na čem je simulace prováděna:

- 1 - simulován je model na hostu (PC)
  - je nutné zajistit chod simulace a komunikaci v reálném čase
  - více v kapitolách 4.3.2, 4.3.3 a 4.3.4
- 2 - na hostu je spuštěn vygenerovaný kód
  - snaží se být real-time, ale přesto velká latence
  - více v kapitole 4.3.5
- 3 - tuto verzi nelze realizovat, model lze spustit jen v Simulinku
- 4 - na simulačním targetu (s real-time operačním systémem) je spuštěn vygenerovaný kód
  - více v kapitole 4.3.6

4.3.1 Požadavky

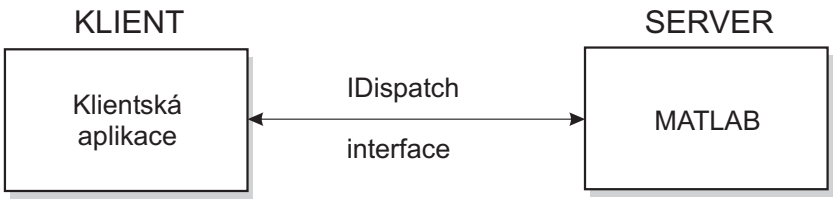
Před provedenou analýzou možných způsobů nového vlastního řešení PIL simulace byly stanoveny další požadavky:

- práce PIL v reálném čase
- využití komunikace po sériové lince RS232
  - toto rozhraní bývá dostupné na všech vývojových deskách, v aplikacích bývá častěji volné (narozdíl od CAN a dalších)
  - v simulátoru (PC) nejjednodušší přístup, není potřebná další karta s rozhraním ani další speciální ovladače

4.3.2 Řešení s Matlab COM serverem

Matlab může kontrolovat nebo být kontrolován jinými COM komponentami. Jestliže je Matlab ovládán jinou COM komponentou, vystupuje v komunikaci jako server. V opačném případě je klientem.

Z několika možných client/server konfigurací lze pro PIL simulaci využít konfiguraci, kdy je Matlab serverem a uživatelská aplikace klientem (viz obrázek 4.5). Matlab jako server může být ovládán jakýmkoliv programem ve Windows.



Obrázek 4.5: Matlab COM server

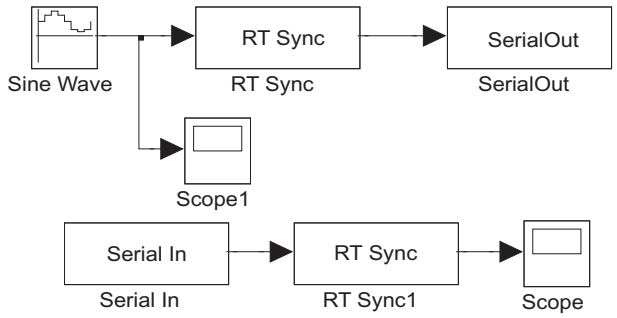
Díky Matlab serveru je možné klientskou aplikací spouštět příkazy ve workspace Matlabu a vypočtené hodnoty získávat ze serveru zpět. Seznam příkazů a více informací o Matlab COM serveru lze získat v dokumentaci External Interfaces [10].

Pro PIL simulaci lze tedy vytvořit Windows aplikaci komunikující s CPU na straně jedné a s Matlab COM serverem na straně druhé. Vypočtená data z CPU čte aplikace z komunikačního kanálu a pomocí Matlab serveru je předává simulačnímu modelu. Poté spustí krok simulace, vypočtená data si ze serveru přečte a vyšle je k CPU. Dále tyto akce cyklicky opakuje.

4.3.3 Řešení s Real-Time Toolboxem

Real-time (RT) toolbox je produkt firmy Humusoft [13] vyvinutý pro simulace v reálném čase. Model využívající bloky tohoto nástroje není nutné překládat do cílové aplikace. Simulace je prováděna standardně pod Simulinkem a real-time toolbox zajišťuje běh v reálném čase.

Pro zjištění minimální periody nutnou pro komunikaci byl proveden test s modelem podle obrázku 4.6. Model je obsažen na přiloženém CD v adresáři *src\others\rt_toolbox*. Na CPU je spuštěn jednoduchý program, který po příjmu rámce odesílá sériovou linkou ihned rámec zpět. Při testu byl komunikován rámec délky 10B. Bloky pro práci se sériovou linkou (*Serial Out*, *Serial In*) jsou samy o sobě pouze pro simulaci bez reálného času, ale ve spolupráci s bloky *RT Sync* je lze provozovat jako real-time. Jednotlivé bloky RT block setu jsou M S-fce (viz. [5]). Je možné posílat jak zprávy v binárním tvaru, tak v textovém formátu.



Obrázek 4.6: Model pro test Real-Time Toolbox

Pro vlastní implementaci PIL simulace je nutné upravit funkci bloku *Serial Out*, tj. její zdrojový soubor *rtbsout.m*. Podobně se upraví funkce *Serial In*.

Testem zjištěná minimální vzorkovací perioda[1] je 10ms (min. hodnota 7ms). Testován byl Real-Time Toolbox verze 3.12 na počítači P4 3GHz, 512MB RAM s Windows 2000.

4.3.4 Řešení s S-funkcemi a komunikačním serverem

Toto řešení využívá S-funkce, meziprocesní komunikaci (roury) a externí program (server) pro řízení komunikace mezi CPU a PC.

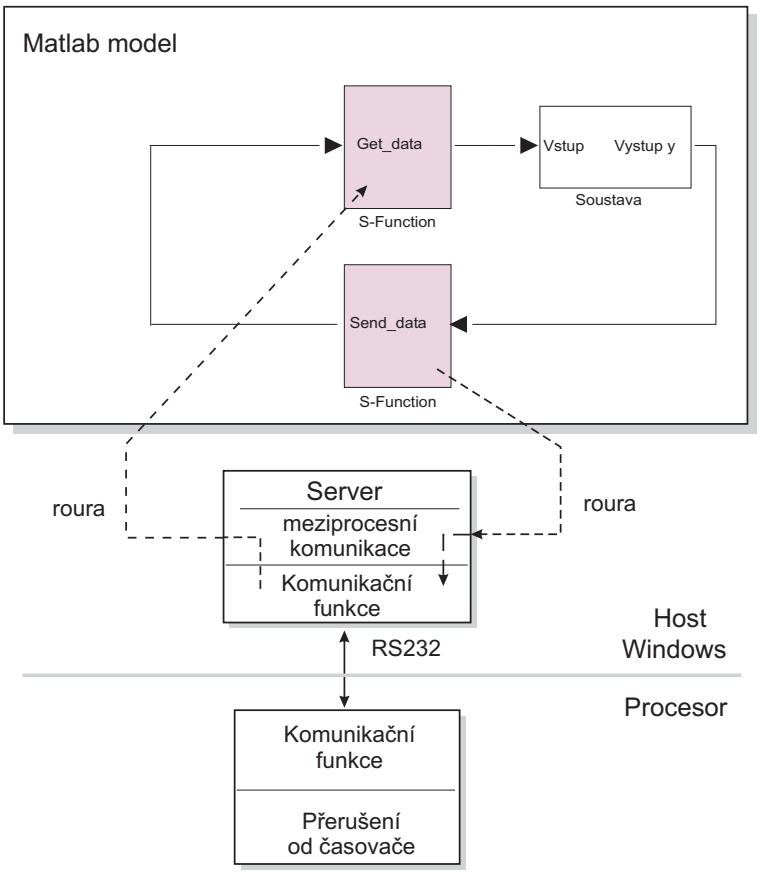
Pod Windows je spuštěn externí program, tj. server, který na jedné straně komunikuje po sériové lince s procesorem a na straně druhé se Simulinkem. Pro komunikaci serveru s Matlabem je použita meziprocesní komunikace - roury.

Strukturu řešení lze vidět na obr. 4.7. Klientskou část roury obsluhují S-funkce *Get_data* a *Send_data*. S-funkce *Get_data* čte z roury příchozí data určená pro Simulink.

Přitom operace čtení je blokující - zajistí tak čekání kroku simulace na příchod dat. Tím je zajištěno, že simulace probíhá v reálném čase, neboť krok simulace je spuštěn

[1] Minimální vzorkovací perioda. Minimální hodnota času nutná k chodu komunikace. Za tento čas se provede výpočet v Simulinku, komunikace od PC k CPU, výpočet na CPU a komunikace od CPU k PC.

z CPU po výpočtu regulátoru.



Obrázek 4.7: PIL simulace s S-funkcemi a serverem

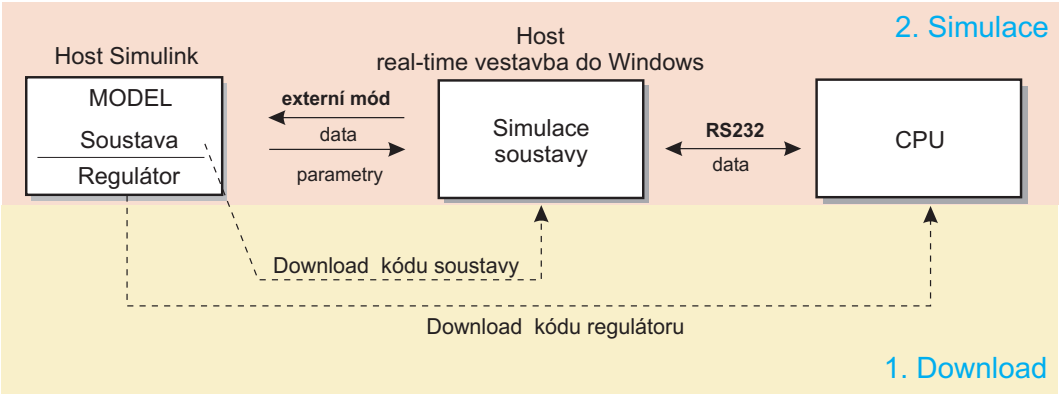
Na obrázku 4.7 je možné vidět navrhovanou strukturu modelu s podporou PIL simulace. Model v prostředí Matlab Simulinku obsahuje:

- dva základní bloky reprezentující regulátor a řízenou soustavu
- dva bloky pro komunikaci modelu s procesorem
  - blok s názvem *Send_data* posílající vypočtené hodnoty výstupů soustavy do procesoru
  - blok *Get_data* přijímá hodnoty akčního zásahu z procesoru a zprostředkovává je modelu v Simulinku

Krok simulace je spouštěn mikroprocesorem na základě přerušení od časovače. Pro otestování časové odezvy simulátoru (host s Windows) s S-funkcemi a komunikačním serverem byl vytvořen v MS Visual Studiu prototyp serveru. Stejně tak byly implementovány S-funkce pro Simulink, které se připojují k rourám vytvořenými serverem. Vše je na přiloženém CD v adresáři *src\others\sfun_server*. Tímto řešením byla dosažena minimální vzorkovací perioda (30 - 40) ms.

4.3.5 Řešení s použitím simulačního RT Win targetu

Toto řešení bylo navrženo za účelem zvýšení vzorkovací rychlosti. Hlavní změnou je, že soustava není simulována Simulinkem, ale je spuštěna jako aplikace v real-time vestavbě ve Windows. Je tedy nutné z modelu soustavy vygenerovat kód (čímž se zajistí větší efektivita). Obrázek 4.8 popisuje navrhovanou strukturu pro tento typ PIL simulace.



Obrázek 4.8: PIL real-time se simulačním targetem

PIL simulace se skládá ze dvou kroků:

1. Download
   - před spuštěním simulace je vygenerován kód pro soustavu a regulátor
   - kódy jsou downloadovány na CPU a simulační target
2. Simulace
   - po downloadu kódu lze spustit samotnou PIL simulaci
   - při simulaci je komunikačním kanálem propojen CPU (regulátor) a simulační target (soustava)
   - dalším komunikačním kanálem je propojeno PC (s Matlabem) a simulační target
     - PC s Matlabem ovládá simulační target, např. spuštění a zastavení aplikace, změnu parametrů

Real-Time Windows (RT Win) Target je určen pro prototypovaní a testovaní systémů pracujících v reálném čase.

Simulace soustavy (její aplikace) je spuštěna v externím módu v reálném čase. Externí mód je speciálním simulačním módem podporovaným v Simulinku (více viz [2]), kdy je z modelu vygenerován kód a spuštěn jinde. Integrace externího módu do tohoto targetu umožňuje využít simulační model v Simulinku pro:

- vizualizaci průběhů signálů
  - pro vizualizaci v reálném čase je možné použít bloků osciloskopů v modelu soustavy (stejné bloky jako při simulaci MIL, která neprobíhá v reálném čase)
- nastavování parametrů
  - parametry jsou nastavovány pomocí jednotlivých bloků modelu v Simulinku

Pro běh aplikace v reálném čase je využívána real-time vestavba do Windows. Její instalace a deinstalace je prováděna příkazy v Matlabu.

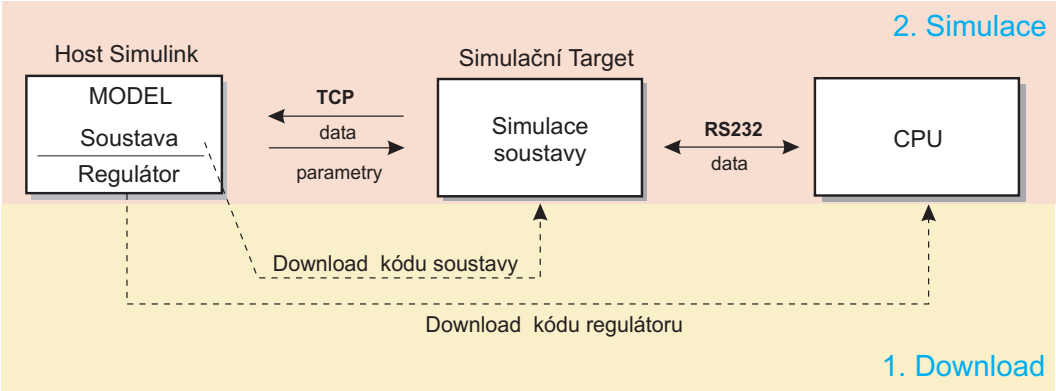
Hlavní možnosti *Real-Time Windows targetu* (viz [14]):

- Simulink modely spuštěny v reálném čase pod Windows na PC
- udávaná vzorkovací rychlost přes 10kHz
- podporuje externí mód, nastavování parametrů a práci s vypočtenými průběhy signálů
- real-time vestavba je spuštěna v Ring 0 (nejvyšší priorita) pod Windows a podporuje single nebo multitasking
- real-time simulace - simulace podle přerušení od systémových hodin
- **je složité psát ovladače**, v S-funkcích nejsou podporované žádné Win32 API funkce
- block set neobsahuje ovladače pro přístup k sériové lince

4.3.6 Řešení s použitím simulačního xPC targetu

Stejně jako řešení popisované v kapitole 4.3.5 bylo i toto řešení navrženo za účelem zvýšení vzorkovací rychlosti. Soustava není simulována Simulinkem, ale je spuštěna jako aplikace na simulačním targetu. Je tedy nutné z modelu soustavy vygenerovat kód (čímž se zajistí větší efektivita) a downloadovat jej na simulační target. Simulačním targetem může být například PC a pokud je na něm spuštěn RTOS, tak dosahuje lepší real-time vlastnosti. Obrázek 4.9 popisuje navrhovanou strukturu pro tento typ PIL simulace.

Target xPC využívá pro běh aplikací v reálném čase tzv. target PC. Target PC je jiný počítač, než na kterém je Matlab Simulink (host PC). Na obrázku 4.9 jsou tedy bloky *Host Simulink* a *Simulační Target* spuštěny na dvou různých PC.



Obrázek 4.9: PIL real-time se simulačním xPC targetem

Z modelu soustavy je vygenerován kód a downloadován na target PC, kde je spuštěno real-time jádro. Host PC a target PC lze propojit sériovou linkou RS232 nebo ethernetem s protokolem TCP/IP. Pro ovládání aplikace lze využít:

- externí mód
  - externí mód je možné používat jen pro změny parametrů v cílové aplikaci
    - platí pro verze starší než xPC Target verze 2.9 a Matlab R2006a
  - využití externího módu stejným způsobem jako u RT Win targetu
    - až od xPC Target verze 2.9 (Matlab R2006a)
- xPC Target Explorer
  - speciální nástroj xPC targetu pro ovládání aplikace na Target PC
  - spustitelný příkazem v Matlabu
  - má funkce podobné externímu módu
  - lze např. zobrazovat průběhy vypočítávaných veličin na Host PC nebo měnit parametry

Hlavní možnosti *xPC targetu* (viz [15]):

- xPC target používá Simulink modely a spouští tyto modely v reálném čase na PC kompatibilním hardwaru
- umožňuje přidat do modelu bloky I/O rozhraní, automaticky generovat kód nástrojem Real-time workshop a downloadovat kód na druhé PC, na kterém běží xPC Target real-time jádro

- dosahuje až 100kHz vzorkovací rychlosti, záleží na rychlosti CPU targetu a složitosti modelu
- monitorování signálů a nastavování parametrů z host nebo target PC
- komunikace s hostem přes RS232 nebo TCP/IP
- běh programu na targetu lze ovládat z Matlabu (grafické rozhraní nebo příkazová řádka)
- obsahuje CAN I/O drivery (podporována spousta výrobců hardwaru) + ovladače RS232
- lze získat (uložit) průběhy signálů, měnit hodnoty parametrů i monitorování signálů lze z target PC
- target lze ovládat i z web browseru, na PC pak nemusí být nainstalován Matlab ani Simulink
- programování vlastních GUI

4.3.7 Porovnání řešení

Byly provedeny testy a výsledky z testů shrnuje tabulka 4.1. Jednotlivá řešení jsou zde porovnávána z hlediska požadavků na hardware, dostupných ovladačů, možné rychlosti simulace a ceny. Taktéž jsou zaznamenány konfigurace PC, na kterých byly testy prováděny. Uvedené ceny jsou platné k 12/2006.

Pro řešení využívající Matlab COM server a Real-Time Windows Target nebyly zhotoveny prototypové funkční příklady. Proto v tabulce 4.1 nejsou uvedeny některé parametry. V případě *Matlab COM serveru* je nutné vytvořit aplikaci klienta s vlastními ovladači pro přístup ke komunikačnímu kanálu. Aplikace i Matlab jsou spuštěny ve Windows bez podpory reálného času, vzorkovací rychlost nebude největší.

Real-Time Windows Target neobsahuje ovladače pro sériovou linku. Pro vytvoření ovladačů nelze využít win32 API funkce. Proto je implementovat vlastní driver složité.

Real-Time Toolbox obsahuje ovladače sériové linky. Pro využití při PIL simulaci musí být však modifikovány. Model používaný při testech je přiložen na CD v adresáři *src\others\rt_toolbox*.

Vzorkovací perioda (30-40) ms u řešení s *S-funkcemi a komunikačním serverem* se nejeví jako dostatečně malá. Komunikace mezi CPU a PC nejde výrazně zrychlit, Simulink je spuštěn pod Windows jako proces. Nepomáhá ani přiřazení vyšší priority

procesu pomocí *Task Manageru* ve Windows. Prototyp komunikačního serveru a implementované S-funkce pro Simulink využívané při testech jsou přiloženy na CD v adresáři *src\others\sfun_server*.

	Matlab COM server	RT Toolbox	S-funkce a komunikační server	RT Win Target	xPC Target
Hardware (PC)	Nutné 1 PC.	Nutné 1 PC.	Nutné 1 PC.	Nutné 1 PC.	Nutná 2 PC.
Ovladače pro RS232	Lze v C.	Nutné úpravy.[c]	Lze v C.	Nejsou.[a]	K dispozici.[b]
Rozsah dokumentace	Matlab user's guide.	User manual.	Vlastní řešení.	User's guide.	Getting started. User's guide. I/O reference. API guide.
Dosažené vzorkovací periody	Netestováno.	10 ms	(30-40) ms	Nelze.	3 ms
Konfigurace PC	Netestováno.	P4 3GHz, 512MB RAM	AMD 1GHz, 512MB RAM	Netestováno.	P1 150MHz, 16MB RAM (SimTarget)
Cena SW potřebného pro PIL v Kč	0	52 980 [d]	0	72 980[e]	143 980 [f]
Celková [g] cena SW v Kč	619 920	672 900	619 920	692 900	763 900

Tabulka 4.1: Srovnání programových řešení PIL v reálném čase

[a] Nelze využít win32 API, driver lze psát programováním v ring zero - registrech.
[b] Drivery pro RS232 jsou k dispozici v knihovně xPC target.
[c] Ovladače jsou k dispozici, nutné úpravy.
[d] 52 980 Kč = Real Time Toolbox.
[e] 72 980 Kč = Real-Time Windows target.
[f] 143 980 Kč = xPC target.
[g] Uvedená celková cena software za 1 licenci pro firmy. Započtena cena nástrojů nutných pro generování kódu regulátoru: Matlab (69 980 Kč), Simulink (99 980 Kč), RTW (269 980 Kč), RTW EC (179 980 Kč) a PEERT, tj. celkem 619 920 Kč.

Největší vzorkovací rychlosti je dosaženo použitím *xPC targetu* jako simulačního targetu. Ovladače z block setu tohoto targetu již není nutné upravovat. Obsahuje ovladače pro RS232 i CAN. Produkt xPC target je distribuován s rozsáhlou dokumentací a podporou I/O karet.

Proto bylo pro vlastní řešení PIL simulace vybráno řešení s použitím simulačního xPC targetu (vše vytvořené v rámci této práce s využitím xPC targetu je přiloženo na CD v adresáři *src\peslib_pil*).

Kapitola 5

Architektura PIL v PEERT

Na základě porovnání výsledků testů (viz tabulka 4.1) bylo pro implementaci real-time PIL v rámci této práce vybráno řešení s xPC targetem. Z možných způsobů lze použitím xPC získat nejvýhodnější systém. Dosahuje nejvyšší vzorkovací rychlosti. Není nutné upravovat ovladače z jeho block setu. Pro budoucí využití lze použít i dostupné CAN ovladače.

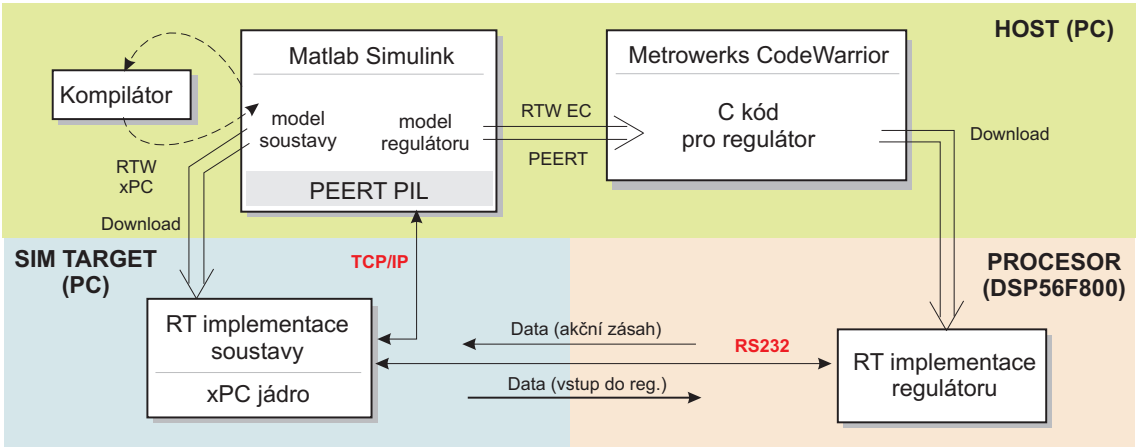
V této kapitole je popsána architektura zvoleného řešení. Jsou zde popsány modifikace bloků simulujících chování periferií. Bloky beanů pro generování kódu (periferií CPU) simulují chování skutečné periferie v simulačním modelu. Pro PIL simulaci musí být do modelu soustavy přesunuty i simulační bloky periferií. Tím se zajistí, že CPU bude pracovat s hodnotami stejnými, jako by získal při využití skutečných periferií. Před odesláním hodnoty výstupu soustavy je tedy např. hodnota zpracována blokem AD převodníku. Odeslaný rámec tak obsahuje hodnotu z výstupu AD převodníku. Přečtením příchozího rámce na CPU se získá stejná hodnota, jako by byla hodnota změřená na vstupu reálné periferie. Analogicky hodnota pro výstupní periferii CPU není touto periferií použita, ale je odeslána z CPU rámcem k simulačnímu PC. Tato hodnota je poté zpracována simulačním blokem periferie a využita jako akční zásah pro soustavu.

Je definován komunikační protokol a časování simulace. Musí být vytvořeny nové bloky pro stavbu a dekódování rámce v požadovaném formátu.

Vzájemnou spolupráci programů při vytváření podpory PIL simulace je možné vidět na obrázku 5.1. Navržená PIL simulace využívá tři různé kusy hardware, které jsou na obrázku barevně rozlišeny. Na **host PC** je používán nástroj Matlab Simulink, Metrowerks CodeWarrior a kompilátor (MS Visual Studio). Na host PC je generován kód regulátoru i soustavy. **Sim Target PC** je další počítač, který je využíván pro simulaci soustavy (je na něm spuštěn přeložený kód soustavy). **Procesor** je použit pro

simulaci regulátoru, je na něm implementován řídicí algoritmus.

Nově vytvořený PEERT PIL target (viz kapitola 6) používá PEERT a xPC target. PEERT PIL target využívá skripty pro vytvoření modelů soustavy a regulátoru ze simulačního modelu řídicího systému (viz obrázek 5.1).



Obrázek 5.1: Architektura řešení real-time PIL simulace

Po vytvoření modelu regulátoru z modelu řídicího systému spouští skript generování efektivního C kódu regulátoru pomocí nástroje Real-Time Workshop Embedded Coder a targetu PEERT. Vygenerovaný kód regulátoru je poté Metrowerks CodeWarriorem přeložen a nahrán do procesoru.

Jakmile je skriptem vytvořen model soustavy z modelu řídicího systému, je z něj generován C kód soustavy pro simulační target nástrojem Real-Time Workshop a targetem xPC. Díky výkonu simulačního PC nemusí být kód soustavy efektivní, proto pro něj není použit RTW EC, který generovaný kód optimalizuje. Po vygenerování kódu soustavy je kompilátorem vytvořena aplikace, která je poté nahrána na simulační PC.

Spojení mezi procesorem a aplikací na simulačním targetu je zajištěno RS232 sériovou linkou. Obsluha je prováděna xPC target bloky a komunikačními funkcemi na CPU . Aplikace pro xPC target je ovládána v Matlab Simulinku, spojení je zajištěno ethernetem s protokolem TCP/IP.

5.1 Bloky pro PIL simulaci

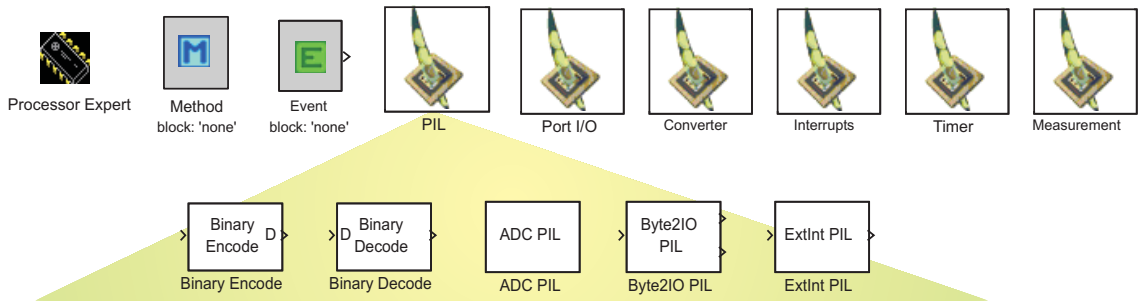
Pro podporu PIL simulace byly vytvořeny nové simulační bloky pro stavbu a dekódování rámců využívaných při komunikace v rámci PIL simulaci. Dále byly modifikovány bloky simulující chování periferií.

Bloky byly vytvořeny jako S-funkce. Nemají svůj TLC soubor pro generování kódu, nejsou tedy vkládanými S-funkcemi. Pro automatické vytváření kódu z modelu nemusí

mít S-funkce TLC soubor. V tom případě je vygenerovaný kód neoptimalizovaný a s větší paměťovou náročností, v kódu se objeví přímo volání vnitřních funkcí S-funkce (viz kapitola 3.1.1.1 a [5]).

Tento přístup byl zvolen pro usnadnění práce. Na simulačním targetu nemusí být kód efektivní, navíc vygenerovaný kód provádí tu samou funkci (ty samé výpočty) jako blok v simulačním modelu bez nutnosti psát speciální TLC soubor pro každý nově vytvořený blok.

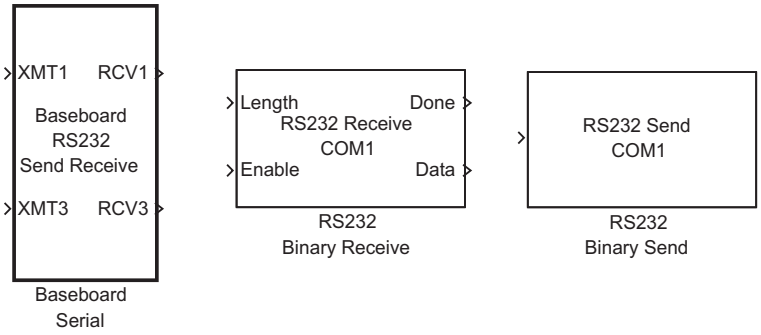
Bloky podpory PIL simulace byly uloženy do podknihovny PIL ve stávající knihovně *Processor Expert Blockset*. Byl tak vytvořen block set pro PIL. Vytvořeno bylo celkem 5 nových bloků. Obsah knihovny je vidět na obrázku 5.2. Bloky podknihovny PIL nejsou určeny pro uživatele. Využívají je skripty při vytváření modelu regulátoru a soustavy pro PIL simulaci. Jelikož vývoj bloků periferií targetu PEERT neustále probíhá, byly zatím implementovány pouze bloky *ADC PIL*, *Byte2IO PIL* a *ExtInt PIL*.



Obrázek 5.2: PEERT block set s podknihovnou PIL

5.1.1 Ovladač sériové linky

Pro práci se sériovou linkou lze využívat ovladače obsažené v knihovně xPC targetu (viz obrázek 5.3).



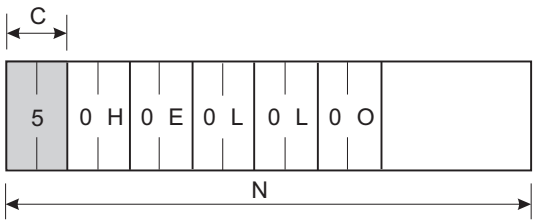
Obrázek 5.3: xPC knihovní bloky Baseboard Serial, RS232 Binary Receive a Send

Ovladače jsou reprezentovány bloky v Simulinku. Pro PIL simulaci je možné použít bloky *Baseboard Serial* nebo *RS232 Binary Send* spolu s *RS232 Binary Receive*.

Bloky *RS232 Binary Send* a *RS232 Binary Receive* jsou již zastaralé a v Matlabu je uživatel upozorněn, aby je v nových aplikacích již nepoužíval. Proto nebyly při návrhu PIL simulace dále uvažovány. Blok ovladače *Baseboard Serial* (obrázek 5.3) je využíván pro nastavení parametrů sériové linky a formátu přijímaných/odesílaných dat. Protože jsou nové bloky PIL simulace (bloky pro stavbu a dekódování rámců) navrženy pro spolupráci s tímto blokem *Baseboard Serial*, je zde popsána jeho konfigurace.

Tento ovladač je subsystém, který je složen z několika vnitřních bloků. I proto se jeho nastavení provádí ve 4 skupinách:

- Board Setup
  - výběr portů pro příjem a odesílání dat
- Basic Setup
  - nastavení základních parametrů portu (přenosová rychlost, parita, počet datových bitů, počet stop bitů, ...)
- Transmit Setup
  - nastavení velikosti softwarového FIFO bufferu pro odesílání dat
  - výběr odesílaného datového typu
    pro nastavení tohoto důležitého parametru lze vybrat ze 2 skupin:
    * *8 bitový datový výstup* vyžaduje nulou zakončený řetězec jako vstupní vektor
    * *16 a 32 bitový výstup* používá vstup takový, že první prvek udává počet prvků, kterými je tvořen vstupní vektor
      · přitom je odesílán pouze nižší byte z každého datového prvku
      · pokud odesílaná data obsahují nulový byte, musí být vybrán tento datový typ



Obrázek 5.4: 16 bitový datový vstupní vektor

· na obrázku 5.4 je každý prvek vstupního vektoru tvořen 2 byty, C je počet platných dat, N je šířka vstupního vektoru a odeslány na sériovou linku jsou pouze nižší byty z každého prvku, tj. *HELLO*

- Receive Setup
  - nastavení velikosti softwarového FIFO bufferu pro odesílání dat
  - pro určení vstupního datového typu platí to samé jako v případě výstupního typu u nastavení odesílání
  - nastavení minimálního počtu přijatých znaků
    - blok vrací nulový výstup, když přijal menší počet znaků než minimální
  - nastavení maximálního počtu přijatých znaků (nastavuje datovou šířku výstupu bloku)
  - je možné určit i zakončovací znak, kterým je definován konec rámce

Více informací o bloku lze získat v dokumentaci xPC targetu [16]. Na přiloženém CD jsou v adresáři *src\others\baseboard_serial* obsaženy demonstrační modely pro práci s blokem *Baseboard Serial*.

5.1.2 Bloky pro práci s rámci

Novými bloky pro práci s rámci využívaných pro komunikaci mezi procesorem a simulačním PC jsou bloky *Binary Encode* a *Binary Decode* (viz obrázek 5.2). Slouží ke stavbě a dekódování rámců.

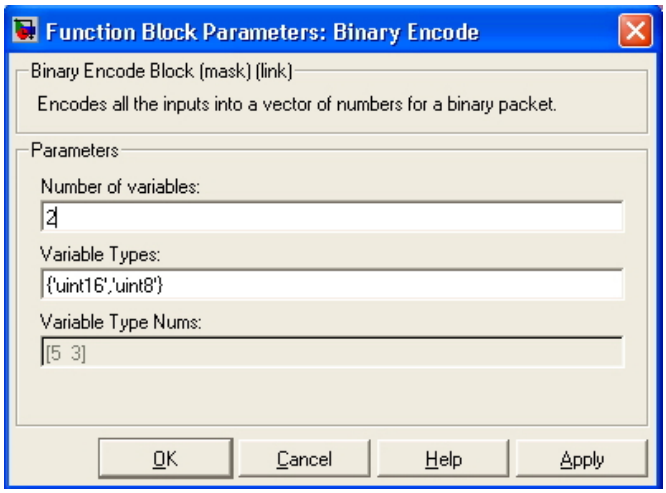
Binary Encode

Binary Encode blok vytváří rámec pro odeslání vypočtených hodnot ze soustavy, tj. z aplikace spuštěné na simulačním PC směrem k CPU.

Vytvořený rámec musí splňovat požadavky bloku *Baseboard Serial*. Jedním z vnitřních bloků *Baseboard Serial* je FIFO fronta pro odesílání dat. Tato FIFO fronta může zpracovat pouze vektor prvků, jehož první hodnota udává počet prvků celého vektoru (viz kapitola 5.1.1).

Binary Encode blok tedy ze vstupních hodnot vytvoří rámec v požadovaném formátu. Formát rámce je blíže popsán v kapitole 5.2. Blok je implementován i pro více vstupů, tj. z více vstupních hodnot vytvoří jeden rámec.

Pro blok byla vytvořena tzv. maska (pomocí *Mask Editoru*, viz [4]), kterou jsou vytvořeny parametry pro blok a vzhled dialogového okna pro nastavení parametrů. Výsledné okno je možné vidět na obrázku 5.5.



Obrázek 5.5: Dialogové okno bloku Binary Encode

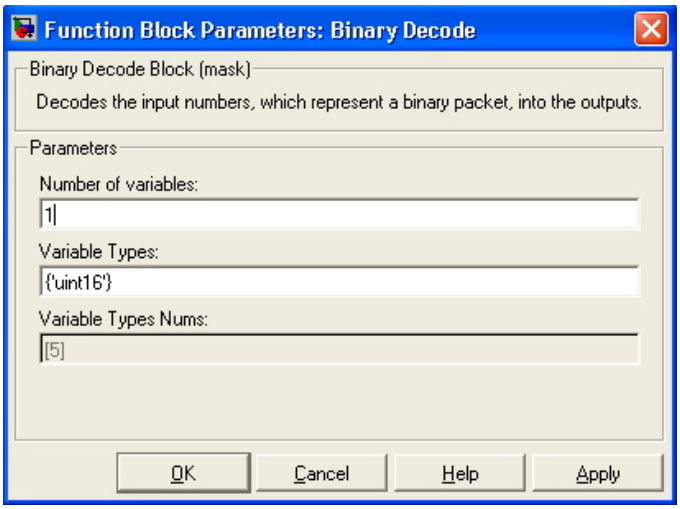
Parametry bloku jsou:

- Number of variables
  - definuje počet vstupních portů
  - každá veličina, jejíž hodnoty mají být odesílány do procesoru (výpočet regulátoru), je k bloku *Binary Encode* připojena jedním vstupním portem
- Variable Types
  - pro každý vstupní port musí být nastaven jeho datový typ
  - přípustné hodnoty jsou: *double*, *single*, *int8*, *uint8*, *int16*, *uint16*, *int32*, *uint32* a *boolean*
- Variable Type Nums
  - hodnota tohoto parametru je automaticky nastavovaná podle parametru *Variable Types*
    - automatické nastavení zařizuje tzv. *dialog callback*, funkce spuštěná po každé změně parametru *Variable Types*
    - více o dialog callback viz [4]
  - parametr je unikátní číslo pro každý datový typ

Šířka výstupního vektoru je nastavována dynamicky, uživatel ji nezadává. K bloku je vytvořena nápověda, kterou lze spustit tlačítkem *Help*. Zdrojový kód bloku je umístěn na přiloženém CD v adresáři *src\peslib_pil\blocks_std*.

Binary Decode

Binary Decode blok dekóduje přijatý rámec od CPU (regulátoru) do výstupních hodnot. Datový typ výstupní hodnoty je určen parametrem bloku. Blok je implementován i pro více výstupů. Je blokem s opačnou funkcí než blok Binary Encode. Dialogové okno bloku Binary Decode je možné vidět na obrázku 5.6.



Obrázek 5.6: Dialogové okno bloku Binary Decode

Parametry bloku jsou:

- Number of variables
  - definuje počet výstupních portů
  - každá veličina, jejíž hodnoty jsou posílány od regulátoru, musí mít svůj výstupní port
- Variable Types
  - pro každý výstupní port musí být nastaven jeho datový typ
  - přípustné hodnoty jsou stejné jako v případě bloku Binary Encode
- Variable Type Nums
  - hodnota tohoto parametru je automaticky nastavovaná podle parametru *Variable Types*

Šířka vstupního portu je nastavována dynamicky. K bloku je vytvořena nápověda, kterou lze spustit tlačítkem *Help*. Zdrojový kód bloku je umístěn na přiloženém CD v adresáři *src\peslib_pil\blocks_std*.

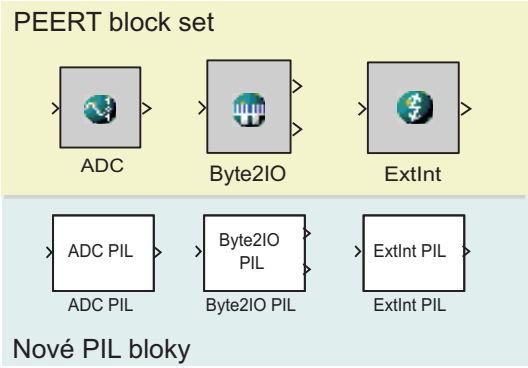
5.1.3 PIL bloky beanů

Bloky beanů pro generování kódu (periferií CPU) simulují chování skutečné periferie v simulačním modelu. Pro PIL simulaci musí být do modelu soustavy přesunuty i simulační bloky periferií. Tím se zajistí, že CPU bude pracovat s hodnotami stejnými, jako by získal při využití skutečných periferií. Před odesláním hodnoty výstupu soustavy je tedy např. hodnota zpracována blokem AD převodníku. Odeslaný rámec tak obsahuje hodnotu z výstupu AD převodníku. Přečtením příchozího rámce na CPU se získá stejná hodnota, jako by byla hodnota změřená na vstupu reálné periferie. Analogicky hodnota pro výstupní periferii CPU není touto periferií použita, ale je odeslána z CPU rámcem k simulačnímu PC. Tato hodnota je poté zpracována simulačním blokem periferie a využita jako akční zásah pro soustavu.

Každý blok periferie má svůj TLC soubor, a proto se ve vygenerovaném kódu objeví příkazy pro práci s reálnou periferií (např. příkaz pro přečtení hodnoty na vstupu AD převodníku). Tyto bloky jsou implementovány jako tzv. vkládané S-funkce (viz 3.1.1.1).

Pro PIL simulaci je ale nutné, aby měly bloky beanů stejnou simulační funkci (simulace periferie) i ve vygenerovaném kódu. Pro tvorbu vlastních bloků periferií určených pro PIL simulaci byly využity S-funkce jednotlivých beanů. Nové **PIL verze bloků beanů** nemají TLC soubor. Jsou vytvořeny jako tzv. nevkládané S-funkce (viz 3.1.1.1) a mají tak stejnou funkci při simulaci jako ve vygenerovaném kódu.

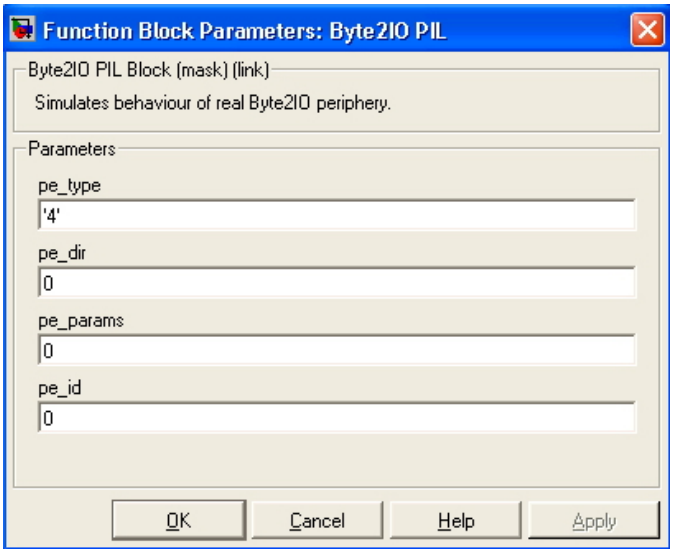
Bloky periferií a jejich verze pro PIL simulace jsou na obrázku 5.7. Všechny jsou obsaženy v PEERT block setu (nové v PIL podknihovně).



Obrázek 5.7: Bloky periferií a jejich verze pro PIL simulaci

S-funkce periferií využívají pro nastavení parametrů nástroj Processor Expert (integrovaný do Simulinku pomocí knihovny PESLIB, viz kapitola 3.3). Nové S-funkce periferií pro PIL simulaci nemohou tento nástroj využívat, protože z vygenerovaného kódu spuštěném na simulačním PC nelze volat PE.

Proto musely být vytvořeny nové verze vycházející z původních S-funkcí. Modifikace kódu jsou okomentovány přímo ve zdrojových souborech. Každá PIL S-funkce má svoje dialogové okno pro nastavení parametrů (např. na obrázku 5.8 pro Byte2IO PIL blok). Parametry jsou shodné s parametry původních bloků periferií.



Obrázek 5.8: Dialogové okno - parametry Byte2IO PIL bloku

Nicméně o parametry PIL bloků periferií se uživatel nemusí starat. Uživatel totiž tyto bloky při návrhu řídicího systému nepotřebuje používat, pracuje pouze s bloky beanů využívajících PE. PIL bloky se užijí až při procesu generování kódu soustavy pro simulační target, kde se i automaticky nastaví jejich parametry (více viz kapitola 6.3).

Block set PEERT obsahuje více periferií než je na obrázku 5.7. Nejsnažší bloky pro použití při PIL simulaci jsou ale bloky z tohoto obrázku, a proto právě pro ně byly vytvořeny nové PIL bloky. Zdrojové kódy PIL bloků periferií jsou umístěny na přiloženém CD v adresáři *src\peslib_pil\blocks_std*.

5.2 Komunikační protokol

Komunikace (výměna dat mezi CPU a simulačním PC) při PIL simulaci je spuštěna odesláním rámce s vypočtenými hodnotami ze simulačního PC (simulace soustavy). CPU přijme rámec od simulačního PC, dekóduje jej a využije přijaté hodnoty pro výpočet svého regulačního algoritmu. Poté odešle do simulačního PC rámec s hodnotami regulačního zásahu.

Jestliže nepřijde na CPU rámec s aktuálními hodnotami výstupu soustavy pro daný krok simulace, tak CPU použije hodnoty staré (dosud nejnovější). Komunikace je zabezpečena kontrolou celkového počtu bytů při příjmu rámce na CPU. Pokud není od

soustavy odeslán rámec celý, na CPU není tento rámec dekódován. Na CPU nejsou nastaveny žádné timeouty pro příchod hlavičky rámce či celého rámce.

Pokud není soustavě odeslán rámec celý, blok *Binary Decode* rámec nedekóduje. Timeouty pro příchod rámců na simulačním targetu nastovat nelze. Příjem rámců není potvrzován.

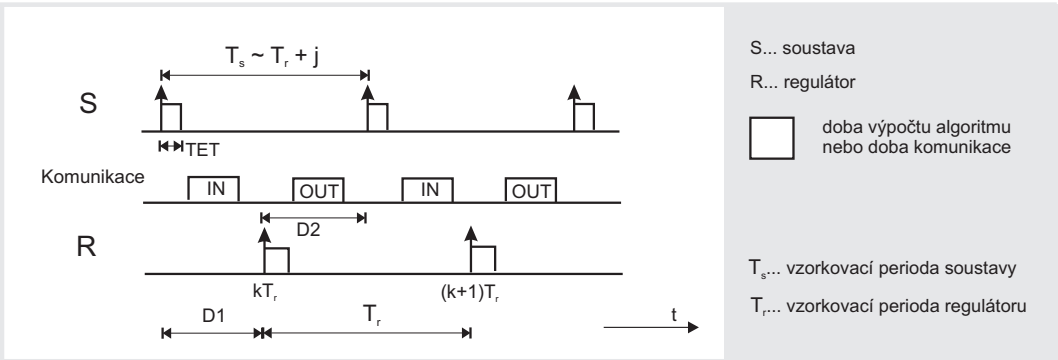
Bloky pro komunikaci (Binary Encode, Binary Decode) využívají nově navržený tvar datového rámce. Pro komunikaci mezi simulačním PC a CPU se používá datový rámec:

- rámec je složen z *hlavičky*
  - 1B: *typ rámce* (zatím implementován pouze datový rámec s hodnotou typu *1*, v budoucnu lze rámce rozlišovat podle jednotlivých vzorkovacích period, pro které budou určeny - multirate modely)
  - 1B: *délka rámce* (délka celého rámce včetně hlavičky)
- a *dat* proměnné délky: xB

Časování simulace

Výpočet regulátoru je spuštěn až po výpočtu první hodnoty výstupu soustavy. V jedné periodě je vždy odeslána hodnota a přijata nová hodnota - platí pro simulační PC i CPU. V případě, že nestihne na CPU přijít hodnota před dalším tikem časovače, je použita hodnota stará (dosud nejnovější).

Na obrázku 5.9 je možné vidět časování komunikace PIL simulace při periodě $T_s = T_r$.



Obrázek 5.9: Časování komunikace

Doba nutná pro komunikaci je označena *IN* a *OUT*. Pro start regulátoru je potřeba zvolit doby *D1* a *D2* tak, aby se stihly komunikace *IN* a *OUT* včetně jitteru *j* v rámci jedné periody.

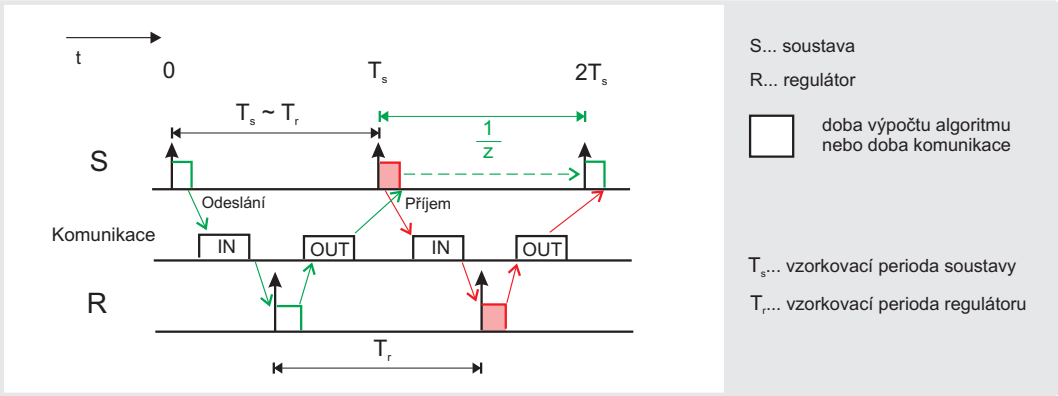
V navržené PIL simulaci čeká CPU na příjem prvního rámce, tj. čas *D1* je roven času *IN + TET*.

V případě časování komunikace pro $T_s < T_r$ je soustava regulačním zásahem aktualizována rychleji než v případě $T_s = T_r$. Pro použití asynchronních událostí v PIL simulaci je při $kT_s = T_r$ hodnota $k > 1$. Jestliže nastane asynchronní událost, je v rámci jedné periody regulátoru T_r možno *k-krát* vyvolat přerušení na CPU (více viz kapitola 6.6).

Při použití xPC targetu je časování simulace soustavy real-time. U xPC target bloků ovladačů RS232 nelze nastavit čekání na příchod rámce před dalším výpočtem algoritmu soustavy.

Na začátku kroku simulace soustavy je nejdříve vypočten výstup soustavy a až poté přijat rámec od CPU. Tím je do modelu uzavřené smyčky přidán 1 krok zpoždění. Přijatá hodnota na soustavě se využije při výpočtu algoritmu až v dalším kroku.

Toto je ilustrováno na obrázku 5.10. Regulační zásah není využit k výpočtu výstupu soustavy v dalším kroce ($t = T_s$). Časovým zpožděním je využit až v čase $t = 2T_s$. Časování je zajištěno pořadím a chováním bloků soustavy a implementací regulátoru.



Obrázek 5.10: Časování komunikace s xPC targetem

Výsledky PIL simulace v porovnání se simulací v Simulinku jsou obsaženy v kapitole 6.5.1.

Kapitola 6

PEERT PIL target

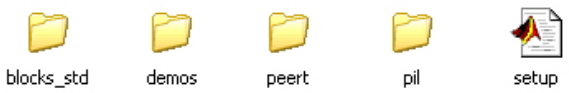
Byl vytvořen nový target PEERT PIL pro automatické generování kódu pro PIL simulaci. Nereprezentuje nový typ procesoru pro Simulink. Je pomocí něj volán target PEERT pro generování kódu regulátoru a xPC target pro kód soustavy.

V současném stavu vývoje neumožňuje PEERT target generovat kód subsystému regulátoru ze simulačního modelu řídicího systému. Proto je nutné vytvořit nový model obsahující pouze regulátor a z něj vygenerovat kód pro CPU. Proto navržený způsob PIL simulace využívá více-modelový přístup, ale modely pro generování kódu jsou vytvářeny automaticky ze simulačního modelu řídicího systému. Z hlediska uživatele jde tedy o jedno-modelový přístup.

Pro zajištění podpory PIL simulace musely být upraveny zdrojové soubory targetu PEERT. Provedené modifikace a způsob práce s výsledným řešení PIL simulace jsou obsahem této kapitoly.

6.1 Instalace targetu

PEERT PIL target je distribuován jako součást targetu PEERT. Po instalaci modifikovaného PEERT (více o instalaci v kapitole 3.3) je možné používat prostředky PIL simulace.



Obrázek 6.1: Obsah modifikovaného PEERT

Novou složkou v PEERT targetu s podporou PIL simulace je složka *PIL*. Ta obsahuje zdrojový soubor *serial_PE_support.c* a hlavičkový *serial_PE_support.h*. Tyto soubory jsou využívány při generování kódu regulátoru, více viz kapitola 6.4.

Dále obsahuje soubor *project.mcp* - projekt pro Metrowerks CodeWarrior. Mcp projekt slouží ke spuštění IDE CodeWarrioru, kde je vygenerovaný kód regulátoru překládán a nahráván na CPU. Soubor *project.mcp* je šablona, která má přednastavené cesty pro vyhledávání zdrojových souborů pro úspěšný překlad kódu. O použití tohoto souboru více v kapitole 6.4.

Po instalaci PEERT je nutné nastavit přístupové cesty v *project.mcp* podle aktuálního umístění Matlabu a PEERT na disku. Nastavované cesty jsou:

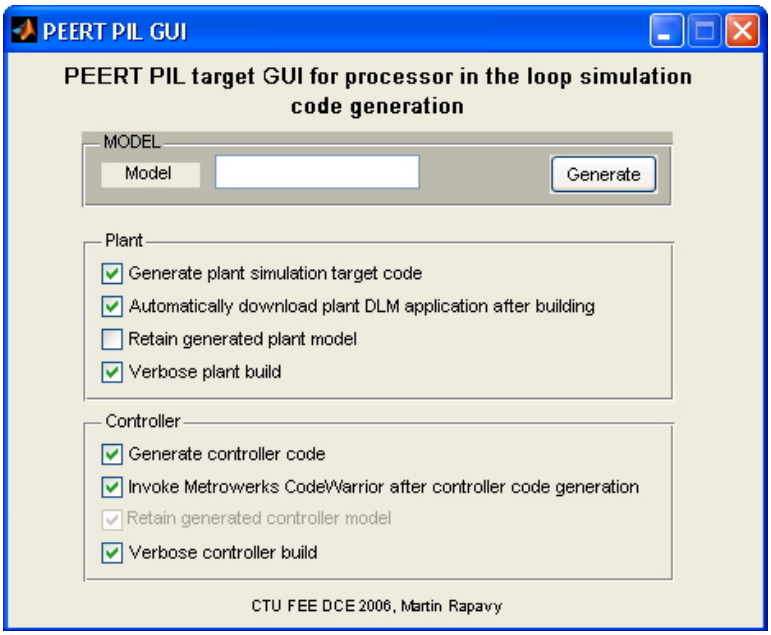
- *matlabroot\simulink\include*
- *matlabroot\rtw\c\libsrc*
- *matlabroot\extern\include*

Matlabroot je cesta k umístění Matlabu. Modifikace cest v CodeWarrioru je popsána v kapitole 3.4 v části *Překlad vygenerovaného kódu.*

6.2 Grafické uživatelské rozhraní targetu

Pro práci s novým targetem bylo zhotoveno grafické uživatelské rozhraní targetu. Je pomocí něj ovládáno generování kódu pro PIL simulaci.

Okno GUI je možné vyvolat příkazem *peert_pil* v Matlabu (viz obrázek 6.2). Nastavení parametrů pro generování kódu se provádí ve 2 skupinách:



Obrázek 6.2: GUI generování kódu pro PIL simulaci

- kód soustavy
  - lze zapnout/vypnout generování kódu soustavy pro simulační xPC target
  - volba automatického downloadu aplikace na simulační target
    - v případě navázaného spojení mezi host PC a simulačním PC je aplikace na simulační PC automaticky nahrána
  - přepínač pro zachování/smazání vygenerovaného modelu soustavy
  - je možné při generování kódu soustavy zapnout zobrazování výpisů do konzole Matlabu
- kód regulátoru
  - lze zapnout/vypnout generování kódu regulátoru
  - volba automatického spuštění Metrowerks CodeWarrior pro práci s kódem regulátoru
    - využívá registrovaných typů souborů ve Windows, pro správnou funkci musí být registrován typ souboru *mcp* (Metrowerks CodeWarrior Project)
  - přepínač pro zachování/smazání vygenerovaného modelu regulátoru
    - pevné nastavení zachování modelu (v současném stavu vývoje PESLIB nelze automaticky smazat model pro PEERT target)
  - je možné při generování kódu regulátoru zapnout zobrazování výpisů do konzole Matlabu

Do pole *Model* je nutné napsat název simulačního modelu celého řídicího systému. Tlačítkem *Generate* se spustí generování kódu soustavy a regulátoru podle nastavených parametrů.

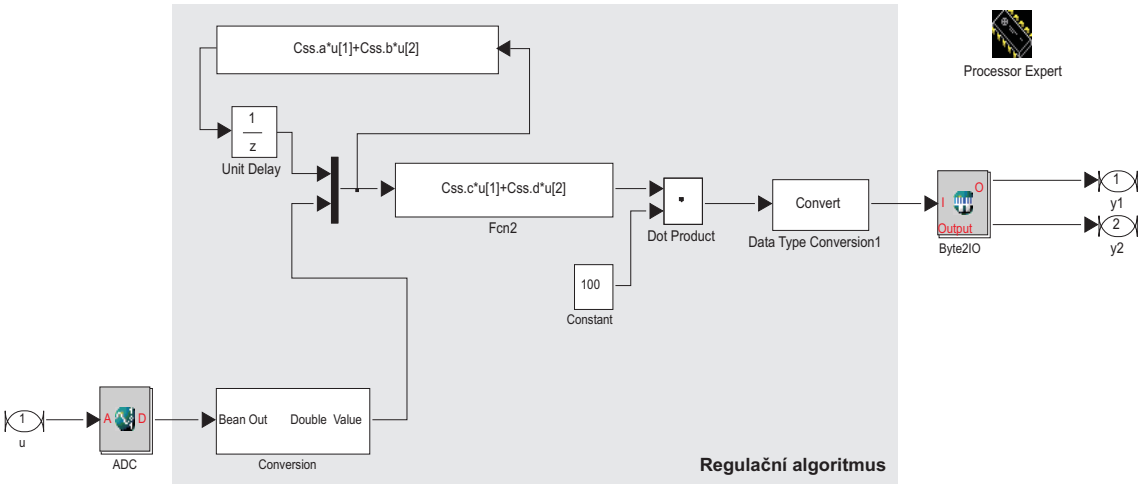
6.3 Generování kódu soustavy - xPC target

6.3.1 Model soustavy

Automatické vytvoření modelu soustavy pro simulační xPC target obstarává vnitřní funkce PEERT PIL targetu *gen_code_plant.m*. Vstupní parametry jsou funkci předávány pomocí GUI targetu PEERT PIL.

Změnou oproti simulačnímu modelu (obrázek 6.3) je to, že subsystém regulátoru je v modelu pro xPC target nahrazen bloky pro komunikaci s CPU (obrázek 6.4),

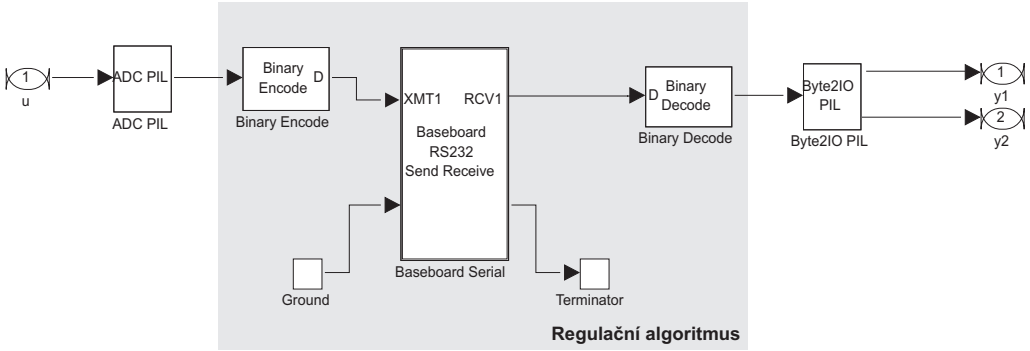
na kterém je spuštěn výpočet regulátoru při PIL simulaci. Dále jsou bloky periferií nahrazeny svými bloky periferií pro PIL simulaci.



Obrázek 6.3: Simulační subsystém regulátoru

Funkce *gen_code_plant.m* zajišťuje následující kroky:

- provede kontrolu, zda-li je v simulačním modelu regulačního obvodu obsažen *Processor Expert* (PE) blok
- vyhledá v modelu subsystém regulátoru (podle umístění PE bloku)
- uloží model pod novým jménem *modelname_xpc*, kde *modelname* je název simulačního modelu
- přejmenuje subsystém regulátoru *ctrlname* na *ctrlname_xpc*, kde *ctrlname* je název subsystému regulátoru



Obrázek 6.4: Struktura subsystému regulátoru pro xPC target

- ze subsystému regulátoru *ctrlname_xpc* smaže všechny bloky kromě PE bloku, periferních bloků, vstupních a výstupních portů

- vymaže všechny spoje zbylé po smazaných blocích
- zjistí vstupní a výstupní datové typy jednotlivých periferních bloků
  - zamění bloky periferií za bloky periferií pro PIL simulaci
  - nastaví parametry bloků periferií pro PIL simulaci podle parametrů bloků periferií
- smaže PE blok
- přidá blok *Binary Encode* pro sestavení rámce, nastaví jeho parametry a propojí ho se všemi vstupními bloky periferií
- přidá blok *RS232 Baseboard Serial* pro komunikaci po sériové lince a nastaví jeho parametry
- přidá blok *Binary Decode* pro dekódování rámce, nastaví jeho parametry a propojí ho se všemi výstupními bloky periferií
- nastaví parametry pro generování kódu:
  - systémový TLC soubor na *xpctargetert.tlc*
  - zapne položku *Generate Code Only*
  - parametry řešitele (solver)
- podle vstupních parametrů:
  - nastaví parametr *RTWVerbose* - výpisy do konzole Matlabu při generování kódu
  - nahraje výslednou aplikaci na simulační target
  - smaže vytvořený model soustavy *modelname_xpc*

Zdrojový soubor je přiložen na CD v adresáři *src\peslib_pil\peert*.

6.3.2 Konfigurace xPC targetu

6.3.2.1 Požadavky na hardware

Jako target xPC počítač, tj. počítač, na kterém je spuštěna simulace soustavy, je možné použít PC s konfigurací:

- CPU: Intel 386/486/Pentium nebo AMD K5/K6/Athlon

- periferie:
  - 3.5” disketová jednotka, pevný disk není vyžadován
  - sériový COM port a síťová karta
- paměť RAM: alespoň 8MB

V případě TCP/IP komunikace mezi simulačním počítačem a target xPC počítačem je nutné použít na target xPC počítači síťovou kartu s podporovaný čipem (více viz [17]). Podporované 10Mb/s a 100Mb/s síťové karty lze vidět v tabulce 6.1. *V práci byla využívána síťová karta Micronet (10Mb/s, ovladač NE2000).*

Čip	Název ovladače v xPC	Rychlost [Mb/s]	Verze xPC targetu
řady Intel 8255X	I82559	100	2.6.1
řady AMD 79C97x	RTLANCE	100	2.6.1
Realtek RTL8139D	R8139	100	3.0
National Semiconductor DP83815	NS83815	100	3.0
3Com 3C90x	3C90x	100	3.0
NE2000	NE2000	10	2.6.1
SMC91C9X	SMC91C9X	10	2.6.1

Tabulka 6.1: Podporované síťové karty pro xPC target

Používané verze xPC targetu při vývoji PIL simulace byly 2.6.1 (Matlab R14SP1) a 2.7.2 (Matlab R14SP3). Target verze 3.0 je používán s Matlabem verze 2006a+.

6.3.2.2 Nastavení prostředí xPC targetu

Parametry xPC targetu lze nastavit v prostředí xPC targetu. Nastavení prostředí je možné provést příkazy *setxpcenv* a *updatexpcenv* nebo pomocí *xPC Target Exploreru* (viz kapitola 6.3.4).

Příkazem *setxpcenv* se nastavují především parametry týkající se komunikace mezi target PC (simulace soustavy) a host PC (s Matlabem ovládající aplikaci soustavy). Dále pak překladače generovaného C kódu. V Matlabu verze R14SP1 a R14SP3 je možné použít překladače:

- Microsoft Visual C/C++ Microsoft (Visual Studio C/C++ Verze 6.0 or 7.1 (.NET 2003))
- Open Watcom C/C++

Ukázkové nastavení xPC prostředí lze provést příkazem:

```
setxpcenv('CompilerPath','c:\program files\microsoft visual studio .net 2003',...
'HostTargetComm','TcpIp','TcpIpTargetAddress','198.168.1.2',...
'TcpIpSubNetMask','255.255.255.0','TcpIpTargetPort','22222',...
'TcpIpGateway','255.255.255.255','TcpIpTargetDriver','NE2000')
```

Ten nastaví:

- překladač kódu MS Visual Studio 2003
- komunikaci mezi target PC a host PC na TCP/IP
- IP adresu target PC
  - IP adresa target PC musí být statická (host PC může mít dynamickou IP adresu přidělovanou DHCP serverem)
- masku sítě
- port pro TCP/IP komunikaci
- IP adresu brány
  - IP adresa brány 255.255.255.255 znamená, že brána není používána (target PC a host PC jsou propojeny přímo, zpravidla kříženým UTP kabelem)
- ovladač síťové karty na target PC

Po každém použití příkazu *setxpcenv* musí být zavolán příkaz *updatexpcenv*, který nastavení aktualizuje.

Jakmile je takto xPC target nakonfigurován, je možné příkazem *xpcbootdisk* vytvořit bootovací disketu pro target PC. Disketa je vytvářena s ohledem na nastavení targetu. Vždy po jakékoliv změně v nastavení je potřeba znovu vytvořit bootovací disk pro target a znovu vygenerovat kód soustavy pro simulační xPC target.

Z této diskety nabootuje real-time jádro na simulačním počítači. Po spuštění real-time jádra je připraven simulační počítač pro použití v PIL simulaci.

6.3.3 Test komunikace se simulátorem

Po spuštění target PC je možné otestovat komunikaci s host PC (Matlabem). K tomu slouží příkazem *xpctest*:

```
### xPC Target Test Suite 2.6.1
### Host-Target interface is: TCP/IP (Ethernet)
### Test 1, Ping target system using standard ping: ... OK
### Test 2, Ping target system using xpctargetping: ... OK
### Test 3, Reboot target using direct call: ... SKIPPED
### Test 4, Build and download xPC Target application using model xpcosc: ... OK
### Test 5, Check host-target communication for commands: ... OK
### Test 6, Download xPC Target application using OOP: ... OK
### Test 7, Execute xPC Target application for 0.2s: ... OK
### Test 8, Upload logged data and compare it with simulation: ... OK
### Test Suite successfully finished
```

Z uvedeného výpisu po spuštění testu lze poznat, co jednotlivé části testují. Test 3 zkouší softwarový restart target PC. Vzdálený reboot počítače nemusí hardware umožňovat, a tak lze tento test vypnout. Ve 4. testu se generuje kód a nahrává přeložená DLM aplikace z testovacího modelu *xpcosc*.

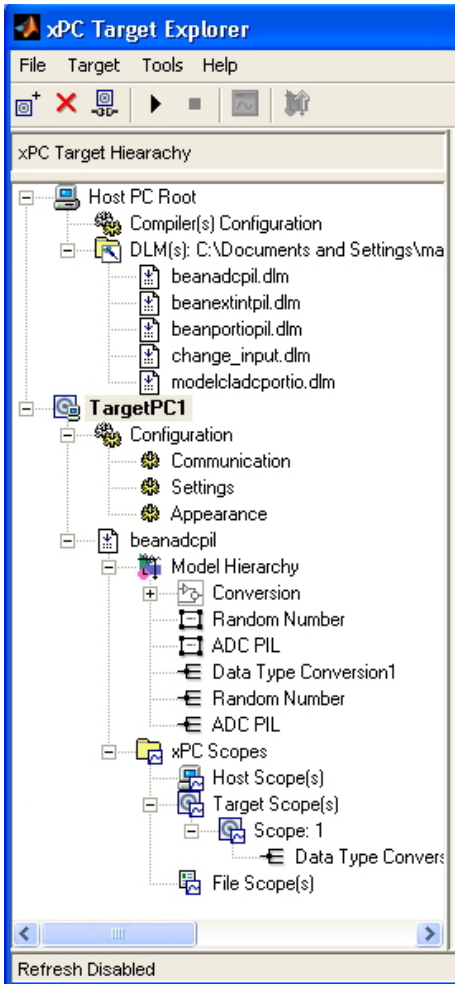
6.3.4 xPC Target Explorer

Aplikace xPC Target Explorer (TE) je program spouštěný na PC s Matlabem (host PC), pomocí kterého lze ovládat cílovou aplikaci na simulačním targetu (obrázek 6.5). Je tak možné spustit cílovou aplikaci, zastavit ji, nahrát novou. Umožňuje také sledování vypočtených průběhů a nastavování parametrů za běhu simulace. Alternativou TE je používání příkazů přímo v Matlabu.

xPC Target Explorer umožňuje:

- přidávat a konfigurovat target PC (až do počtu 64 PC)
- vytvořit bootovací disk pro zvolený target PC
- downloadovat vytvořený DLM soubor (aplikace pro simulační PC) bez opětovného generování kódu
- odebrat nahranou aplikaci z target PC
  - nastavovat parametry použitých bloky a pracovat se signály

- monitorovat signály
  - přidávat xPC osciloskopy, spouštět je a zastavovat
  - 3 druhy těchto osciloskopů:
    1. host - okno osciloskopu je spustitelné na host PC
    2. target - osciloskop je zobrazen na obrazovce target PC
    3. file - data z osciloskopu jsou nahráváno do souboru
  - přidávat/odebírat signály do/z osciloskopů
- měnit parametry simulace - vzorkovací periodu, čas simulace atd.



Obrázek 6.5: Okno xPC Target Exploreru

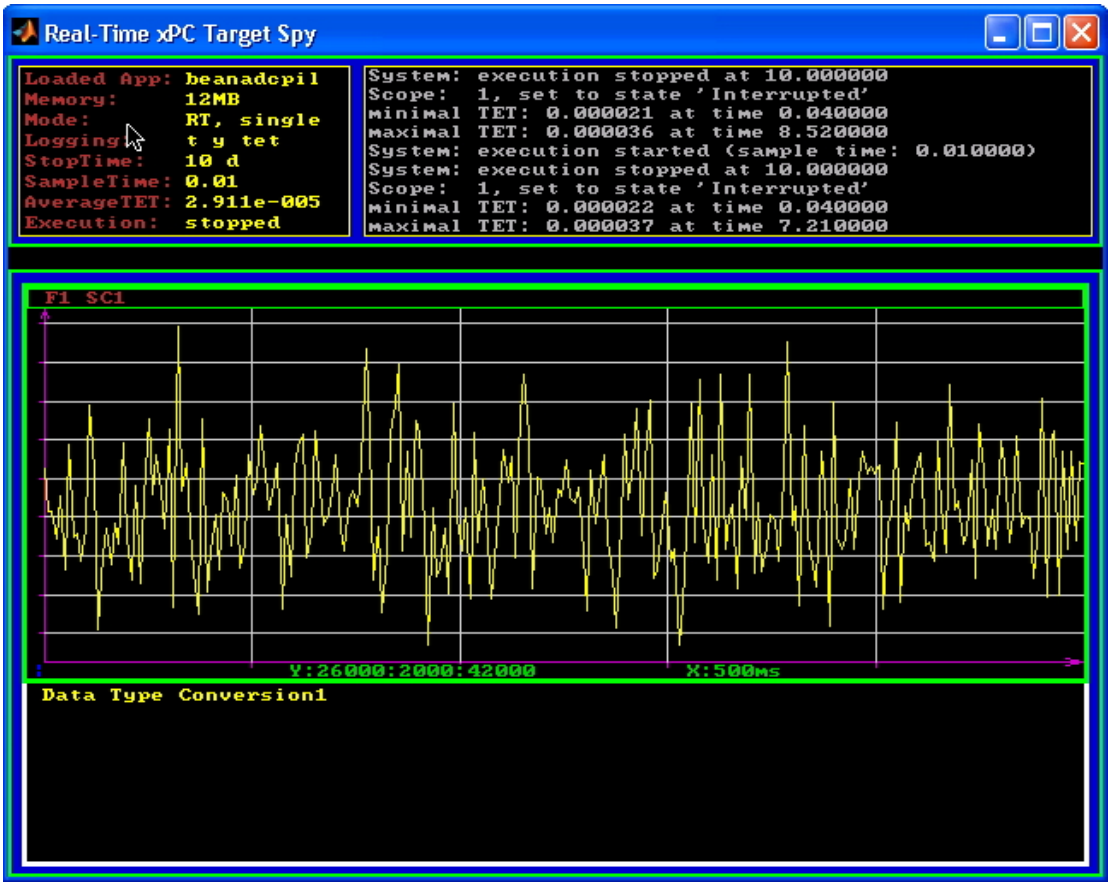
Okno xPC Target Exploreru se vyvolá příkazem *xpcexplr*. Díky xPC TE je možné měnit parametry jednotlivých bloků za běhu simulace v reálném čase. Není tak potřeba používat externí mód, všechny jeho funkce má i xPC TE.

Lze přidat nový target a v položce *Configuration - Communication* nastavit parametry komunikace.

6.3.5 GUI na simulátoru

Grafické rozhraní na simulačním PC je možné vidět na obrázku 6.6. U simulačního PC nemusí být připojen monitor či displej, z host PC lze příkazem *xpctargetspy* získat

snímek obrazovky simulačního PC. Takto byl pořízen i obrázek 6.6.



Obrázek 6.6: Grafické rozhraní na simulačním PC

Target PC GUI je rozděleno do 3 částí. Jsou zde zobrazeny informace o nahrané aplikaci, dále vypisovány informace v průběhu spuštěné real-time simulace a vykreslovány průběhy vypočtených hodnot.

6.3.6 Zajímavé příkazy pro xPC

Příkaz xpc

Příkazem *xpc* lze získat objekt typu xpc po ztrátě spojení se simulačním PC. Po zavolání příkazu $tg = xpc$ je navázáno spojení (uvedena část výpisu jeho struktury):

```
xPC Object

Connected = Yes
Application = beanadcpil
Mode = Real-Time Single-Tasking

SampleTime = 0.010000
AvgTET = NaN
```

```
TimeLog = Vector(0)
StateLog = Off
OutputLog = Matrix (0 x 3)
TETLog = Vector(0)
```

TET (zkratka z *Task Execution Time*) je doba, za kterou jsou vypočteny hodnoty signálů během každé vzorkovací periody. Po ukončení simulace je možné na host PC uploadovat *TETlog*.

Pokud simulovaný model obsahuje subsystémy, které jsou spouštěny pouze za určitých okolností, lze vykreslením průběhu *TETlog* v závislosti na čase zjistit, kdy byly tyto subsystémy spouštěny.

Vytížení procesoru simulačního PC v procentech je možné spočítat jako *AvgTET* lomeno hodnota vzorkovací periody (*AvgTET* je průměrná hodnota *TET*).

Příkaz xpcbench

Příkaz *xpcbench* spustí benchmark test k otestování výkonu simulačního PC. Při testu jsou na PC nahrávány testovací modely a zjišťována nejmenší možná vzorkovací perioda, než dojde k přetečení procesoru na PC.

Na obrázku 6.7 je vidět výsledek testu pro počítač s CPU Pentium I 150MHz.

Minimal achievable sample times in µs

	Minimal	F14	F14*5	F14*10	F14*25
Intel P4 3GHz .NET03	9	11	14	19	35
AMD 2800+ .NET03	14	16	20	25	41
Intel P4 3GHz .NET	10	12	19	29	55
AMD 2800+ .NET	14	16	21	29	60
AMD Athlon XP 1.6GHz	14	16	25	36	80
AMD Athlon 1GHz	10	13	24	41	109
Intel PIII 1.0GHz	10	13	25	45	124
Intel P4 2.0GHz	13	16	30	52	119
Intel P4 1.5GHz	10	15	31	56	133
Intel PIII 600MHz	8	13	39	97	363
xPC TargetBox 108	11	17	42	87	236
Intel PII 400MHz	8	16	58	135	502
xPC TargetBox 107	10	19	62	126	340
AMD K6-2 400MHz	10	24	75	159	755
xPC TargetBox 106	12	34	120	287	1303
Intel PI 166MHz	11	44	253	567	1681
This Machine	14	61	261	645	1747
Intel PI 90MHz	27	100	590	1400	4600
Intel 486DX 40MHz	43	375	1900	3800	12500
MachZ 128Mhz	56	254	1439	3185	7939

Obrázek 6.7: xpcbench pro Pentium I 150MHz

Testované modely jsou:

- Minimal - model obsahují pouze 3 bloky

- F14 - obsahuje 62 bloků a 10 stavů
- F14*5 - 5 modelů F14 v subsystémech, 310 bloků a 50 stavů
- F14*10 - 620 bloků a 100 stavů
- F14*25 - 1550 bloků a 500 stavů

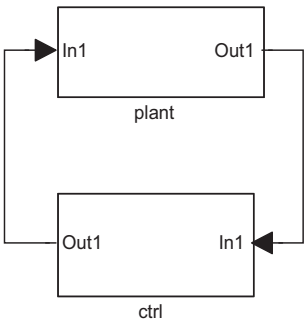
6.4 Generování kódu regulátoru

6.4.1 Model subsystému regulátoru

Automatické vytvoření modelu regulátoru obstarává vnitřní funkce PEERT PIL targetu *gen_code_ctrl.m*. Vstupní parametry jsou funkci předávány pomocí GUI targetu PEERT PIL.

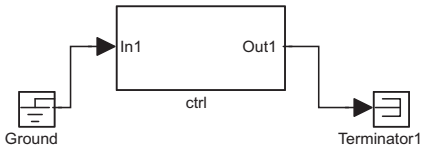
Funkce *gen_code_ctrl.m* zajišťuje několik kroků:

- provede kontrolu, zda-li je v simulačním modelu (obrázek 6.8) regulačního obvodu obsažen PE blok



Obrázek 6.8: Model regulačního obvodu *modelname.mdl*

- vyhledá v modelu subsystém regulátoru (podle umístění PE bloku)
- uloží model pod novým jménem *modelname_ctrlname*, kde *modelname* je název simulačního modelu a *ctrlname* je název subsystému regulátoru
- z modelu *modelname_ctrlname* smaže všechny bloky kromě regulátoru
- vymaže všechny spoje zbylé po smazaných blocích (viz obrázek 6.9)
- nastaví parametry pro generování kódu:
  - systémový TLC soubor na *peert.tlc*



Obrázek 6.9: Vygenerovaný model regulátoru *modelname_ctrname.mdl*

- nastaví parametr *UseModelInPIL*
  - zajistí generování kódu pro PIL simulaci
- zapne položku *Generate Code Only* (vygenerovaný kód nebude překládán, pouze se vytvoří zdrojové kódy)
- parametry řešitele (solver)
- uživatelskou šablonu *gen_rt_OneStep.tlc*
- nastaví parametr *TargetHWDeviceType* na *16-bit Generic*
- ze simulačního modelu zkopíruje do modelu *modelname_ctrlname* nastavení hodnoty kroku simulace
- nastaví parametr *CustomHeaderCode* (více v kapitole 6.4.2.6)
  - parametr je řetězec
  - představuje uživateský kód, který je vložen do hlavičkového souboru *model.h*
- nastaví parametr *CustomInitializer* (více v kapitole 6.4.2.6)
  - uživateský kód, který je vložen do funkce *model_initialize* v souboru *model_rtw.c*

- automaticky spustí generování kódu regulátoru pomocí targetu PEERT
- se složky *peslib\pil* zkopíruje soubor *project.mcp* do složky *modelname_ctrlname*
- přejmenuje *project.mcp* na *modelname_ctrlname.mcp*
- podle vstupních parametrů funkce:
  - nastaví parametr *RTWVerbose* - výpisy do konzole Matlabu při generování kódu
  - spustí *modelname_ctrlname.mcp* v Metrowerks CodeWarrioru

6.4.2 Úprava targetu PEERT

Pro integraci komunikačních funkcí do periferních bloků bylo nutné modifikovat target PEERT i příslušné bloky na několika místech:

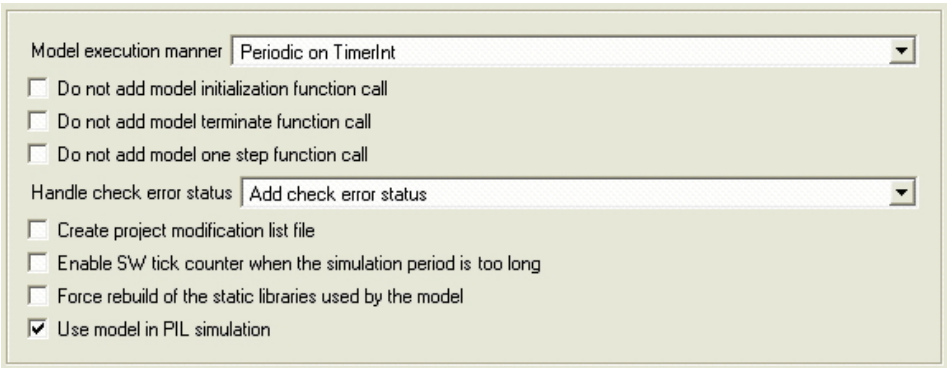
- TLC soubory periferních bloků
- systémový TLC soubor targetu *peert.tlc*
- šablonu *gen_rt_OneStep.tlc*
- m-soubory zajišťující generování kódu pro CPU
- vytvoření souboru funkcí komunikačního ISR serveru *CommServer.c*

6.4.2.1 Generování kódu pro PIL

Přidáním parametru *UseModelInPIL* do konfigurace PEERT targetu lze rozlišit 2 případy pro proces generování kódu:

- generování kódu bez PIL simulace
  - využívají se fyzické periferie
- generování kódu pro PIL simulaci
  - přístup k fyzickým periferiím je nahrazen přístupem ke komunikačnímu kanálu

Na obrázku 6.10 je možné vidět přidaný parametr *UseModelInPIL* (popisek *Use model in PIL simulation*) na upraveném panelu *Processor Expert Options*. Tento panel je součástí nastavení targetu PEERT (viz obrázek 3.5).



Obrázek 6.10: Panel Processor Expert Options

Parametr byl přidán úpravou souboru *peert.tlc* v části kódu *RTW_OPTIONS*, který specifikuje možná nastavení.

6.4.2.2 Přesměrování periferií na komunikační kanál

Při PIL simulaci nejsou využívány reálné periferie CPU, ale místo toho jsou periferie přesměrovány na komunikační kanál. Proto musely být upraveny TLC soubory bloků periferií a vytvořen nový zdrojový soubor *serial_PE_support.c*, kde jsou implementovány funkce pro přístup ke komunikačnímu kanálu.

Modifikace TLC souborů periferií

Aby mohlo být přistupováno ke komunikačnímu kanálu, musely být upraveny TLC soubory bloků periferií. TLC soubory byly modifikovány tak, že využívají funkce pro práci s rámci *SerialPortSetValue* a *SerialPortSetPacket*.

Příklad úpravy souboru **vstupní periferie** AD převodníku (soubor *pes_adc.tlc*, podtržená část kódu je původní, nepodtržená část je nová pro PIL):

```
%if UseModelInPIL == 0
   %<BeanName>_Measure(1); /* Start measurement and wait for the result */
   ...
%else
   /* Received value is assigned to the output of the block by Communication
   ISR Server */
%endif
```

V TLC souboru je testována hodnota parametru *UseModelInPIL* (příznak generování kódu pro PIL simulaci). Pokud je generován kód pro PIL, je do výsledného kódu pouze vložen komentář. Změřená hodnota na vstupu AD převodníku je nahrazena použitím příchozí hodnoty z rámce od soustavy. Příjem rámců, jejich parsování a ukládání hodnot do proměnných obstarává komunikační server (více v kapitole 6.4.2.4).

Protože AD převodník je vstupní periferií, nejsou v generovaném kódu použity funkce pro odesílání rámců.

Výstupní periferie využívá funkce *SerialPortSetValue* a *SerialPortSetPacket*. Pro každý svůj vstup zavolá funkci *SerialPortSetValue* a začlení tak vypočtenou hodnotu na svém vstupním portu do bufferu pro odesílání hodnot. Po zpracování poslední výstupní periferie je zavolána funkce *SerialPortSetPacket*, která odešle rámec z CPU na simulační PC, kde je spuštěn algoritmus soustavy. Soustava obsahuje bloky simulující periferie, proto jsou zde vstupy periferie dále zpracovány.

Příklad úpravy souboru **výstupní periferie** Byte2IO (soubor *pes_portio.tlc*, podtržená část kódu je původní, nepodtržená část je nová pro PIL):

```
%if UseModelInPIL == 0
   /* Put value to %<BeanType> bean %<BeanName> */
```

```
    %<BeanName>_PutVal((%<datatype>)%<u>);
    ...
%else
    %foreach port = block.NumDataInputPorts
        SerialPortSetValue((void*)%<u0_adr>,sizeof(%<u_name>));
        %assign ::Outdone = Outdone + 1
    %endforeach
    %assign result = FEVAL("peert_num_inports","%<modelname>")
    %if result == Outdone
        SerialPortSetPacket();
    %endif
%endif
```

Funkce pro odeslání dat z CPU

V *serial_PE_support.c* jsou implementovány 2 funkce pro odesílání dat z CPU:

- `boolean_T SerialPortSetValue(void *src, int size)`
  - vstupními parametry je ukazatel na hodnotu a velikost datového typu předávané hodnoty
  - předanou hodnotu zapíše do bufferu pro odeslání dat na pozici *write pointeru*, který je při každém volání této funkce inkrementován podle velikosti datového typu zapisované proměnné
  - v bufferu je tak tvořen rámec pro odeslání vypočtených dat z regulátoru do soustavy
- `boolean_T SerialPortSetPacket(void)`
  - odešle rámec z bufferu pro odeslání dat

Soubor *serial_PE_support.c* musí být přidán do generovaného PE projektu, aby mohly být využívány jeho funkce.

Funkce *peert_num_inports* vrací počet vstupních portů všech výstupních periferií použitých v modelu regulátoru. Je to m funkce, a tak musí být v TLC volána přes funkci *FEVAL* (více viz [6]). Každý vstupní port inkrementuje hodnotu proměnné *Outdone* (proměnná *Outdone* je globální proměnná definovaná v systémovém TLC souboru *peert.tlc*). Porovnáním hodnoty *OutDone* s hodnotou z funkce *peert_num_inports* je možné zjistit, zda-li je zpracovávána poslední periferie. Pak je totiž rámec složen a musí být odeslán funkcí *SerialPortSetPacket*.

Přidání souboru do PE projektu

Pro přidání souboru nutného při PIL simulaci byly provedeny následující úpravy:

- v systémovém TLC souboru *peert.tlc* je volán *gen_pil_filelist_mpf.tlc*
- vytvořen soubor *gen_pil_filelist_mpf.tlc*
  - při generování kódu vytváří soubor *pil_filelist.mpf*, ve kterém je uložen seznam přidávaných souborů do PE projektu
- v *peert_make_rtw_hook.m* je volána funkce *addPILFilesToPe* pro přidání souborů do PE projektu
- vytvořena *m* funkce *addPILFilesToPe*
  - vstupem funkce je vytvořený soubor *pil_filelist.mpf* se seznamem přidávaných souborů

Tímto je zajištěno přidávání souborů do PE projektu. V případě generování kódu pro PIL simulaci jsou takto vloženy do projektu soubory *serial_PE_support.c* a *CommServer.c* (o *CommServer.c* více v kapitole 6.4.2.4).

6.4.2.3 Informace o periferiích v modelu pro PIL

Informace o blocích periferií použitých v modelu jsou využívány při generování kódu komunikačního serveru (o komunikačním serveru více v kapitole 6.4.2.4).

Tyto informace jsou v průběhu procesu generování kódu regulátoru uloženy v TLC strukturách typu *record* (záznam). Pro kontrolu funkčnosti je vytvářen soubor *peripheral_list.xml*, který má shodnou strukturu jako použité záznamy. Soubor *peripheral_list.xml* není po vygenerování kódu smazán. Je tak možné zkontrolovat, podle jakých vstupních hodnot byl kód komunikačního serveru generován.

V souboru *peert.tlc* je na začátku procesu generování kódu vytvořen záznam *Model*:

```
%if UseModelInPIL == 1
   %createrecord Model {NumIn 0 ; NumOut 0}
%endif
```

Při vytváření záznamu *Model* jsou inicializovány jeho proměnné *NumIn* a *NumOut* na nulu. Tyto proměnné představují celkový počet vstupních a výstupních portů periferií obsažené v modelu. Protože *peert.tlc* je systémový TLC soubor, proměnné v něm vytvořené jsou globální proměnné. To znamená, že k nim lze přistupovat i v ostatních TLC souborech spouštěných při generování kódu.

Při zpracování TLC souborů jednotlivých bloků periferií se ukládá jejich záznam do *Model* záznamu. Proto musely být upraveny funkce *BlockInstanceSetup* v těchto TLC souborech. Přidány byly příkazy:

- pro každý výstupní port vstupní periferie (např. ADC)
  - `%addtorecord Model OutPort {num "%<i>"; varname "%<yname>"; ... datatype "%<y_t>"; TID "%<block.TID>"}`
  - příkaz přidá záznam *OutPort* s proměnnými:
    - *num* - pořadové číslo výstupního portu v rámci jednoho bloku
    - *varname* - název proměnné (signálu) daného výstupního portu
    - *datatype* - datový typ signálu výstupního portu
    - *TID* - Task ID portu
  - příkaz pro přičtení hodnoty proměnné *Model.NumOut*
- pro každý vstupní port výstupní periferie (např. Byte2IO)
  - `%addtorecord Model InPort {num "%<i>"; varname "%<uname>"; ... datatype "%<u_t>"; TID "%<block.TID>"}`
  - příkaz přidá záznam *InPort* s proměnnými stejného významu jako v případě *OutPort*
  - příkaz pro přičtení hodnoty proměnné *Model.NumIn*

Po zpracování všech bloků celého modelu je záznam *Model* naplněn záznamy *OutPort*, *InPort* a jsou nastaveny hodnoty *NumOut* a *NumIn*.

Souběžně s prací se záznamy je vytvářen soubor *peripheral_list.xml*. Funkce pro vytvoření tohoto souboru *BeginPeripheralListFile* je volána v *peert.tlc* před začátkem generování kódu:

```
%if UseModelInPIL == 1
   %include "PILutilities.tlc"
   %<BeginPeripheralListFile(CompiledModel)>
%endif
```

Podobně zavření souboru je funkcí *FinishPeripheralListFile* prováděno po ukončení generování kódu. Obě funkce jsou obsaženy v souboru *PILutilities.tlc*, který je uložen v podadresáři *block_std\tlc_c* targetu PEERT.
Při zpracování bloku periferie je vytvořen buffer *PERIPHERY*, který obsahuje:

- pro každý výstupní port vstupní periferie (např. ADC)
  - `<OutPort num="%<i>" varname="%<yname>" datatype="%<y_t>" ... TID="%<block.TID>"/>`
- pro každý vstupní port výstupní periferie (např. Byte2IO)
  - `<InPort num="%<i>" varname="%<uname>" datatype="%<u_t>" ... TID="%<block.TID>"/>`

Po naplnění bufferu je funkcí *AddPeriphery* přidán do *peripheral_list.xml* záznam o periferii.

Ukázka obsahu *peripheral_list.xml* po dokončeném generování kódu:

```
<Model name="modelclfix_Css" PeslibVersion="0.1">
   <Bean id="2" name="ADC" beantype="ADC">
      <OutPort num="0" varname="modelclfix_Css_B.ADC" datatype="uint16_T" TID="0"/>
   </Bean>
   <Bean id="4" name="Byte2IO" beantype="Byte2IO">
      <InPort num="0" varname="modelclfix_Css_B.DataTypeConversion1" ...
      datatype="uint16_T"TID="0"/>
   </Bean>
</Model>
```

6.4.2.4 Komunikační server

Příjem dat (rámce od soustavy) na CPU je možné řešit dvěma způsoby:

1. číst rámce až když jsou přijaté hodnoty potřebné k dalšímu výpočtu (na začátku periody regulátoru)
   - přijatý rámec do té doby čeká ve vstupním bufferu sériové linky na CPU
2. využít komunikační server
   - příjem rámce a jeho dekódování je zajištěno na události (interruptu) příjmu znaku ze sériové linky
   - jakmile přijde poslední znak rámce, je rámec dekódován a data zapsána do proměnných, rámec dále již není v bufferu

Pro vlastní implementaci PIL simulace byl zvolen přístup s použitím komunikačního serveru. Zvolen byl pro možnou budoucí integraci asynchronních událostí do PIL simulace.

Kód komunikačního serveru je vytvářen podle modelu regulátoru, ze kterého je generován. Funkce komunikačního serveru jsou obsaženy v generovaném *CommServer.c* souboru. Pro jeho tvorbu jsou důležité informace o použitých blocích periferií, které jsou uloženy v záznamu s názvem *Model* (viz kapitola 6.4.2.3).

Soubor komunikačního serveru *CommServer.c* musí být pro správné použití vložen do PE projektu. Vkládán je stejným způsobem jako soubor *serial_PE_support.c*, viz kapitola 6.4.2.2 v části *Přidání souboru do PE projektu*.

Pro vytvoření *CommServer.c* byly upraveny soubory:

- *gen_rt_OneStep.tlc*
  - tento soubor slouží jako šablona pro generování uživatelských zdrojových souborů
- *peert_make_rtw_hook.m*
- *PILutilities.tlc*

Modifikace gen_rt_OneStep.tlc

V souboru je volána TLC funkce *CreateCommServer* pro vytvoření obsahu souboru *CommServer.c*.

```
%if UseModelInPIL == 1
   %include "PILutilities.tlc"
   %<CreateCommServer(Model)>
%endif
```

Pro PIL simulaci je vkládána deklarace funkcí komunikačního serveru do souboru *modelname_Model_Api.h*, kde *modelname* je název modelu.

```
%if UseModelInPIL == 1
   extern void CommServer_read(void);
   extern void CommServer_init(void);
%endif
```

Úprava peert_make_rtw_hook.m

V souboru je volána *m* funkce *peert_modify_events*. Je vnitřní funkcí targetu PEERT slouží k modifikaci generovaného souboru *Events.c*, který obsahuje jednotlivé implementované události.

Je nutné zajistit volání funkce *CommServer_read* z komunikačního serveru na události příjmu znaku. Proto byl do *peert_make_rtw_hook.m* vepsán kód:

```
% Get MdlBean OnRxChar function name
FunctionName = getPeValue(pebl,peMdlBeanId,otBEAN,{'Event','itm_symb="OnRxChar"',...
'itm_symb="Name"'});
% modify EventModules
EApiCall{1,1} = 'CommServer_read();';
peert_modify_events(ModuleName,ModuleList,FunctionName,EApiCode,EApiCall,pe_path,1);
```

Do výsledného souboru *Events.c* vloží kód volání funkce *CommServer_read*:

```
void PIL_as_OnRxChar(void)
{
   /*** RTW event handler. -- DON'T MODIFY THIS CODE !!! ***/
   CommServer_read();
   /*** End of RTW event handler. ***/

   /* Write your code here ... */
}
```

Modifikace PILutilities.tlc

Do souboru byla přidána funkce *CreateCommServer* pro vytvoření souboru *CommServer.c*. Vstupní parametrem je záznam *Model* nesoucí informace o použitých periferiích v modelu. Funkce je šablonou pro vytvoření funkcí *CommServer_read* a *CommServer_init*. Funkce *CommServer_init* je vždy stejná.

CommServer_read se liší podle modelu. Záleží na použitých vstupních periferiích v modelu regulátoru.

Čtení rámce je řízeno stavovým automatem. Protože je funkce pro čtení rámce zavolána vždy po příjmu jednoho znaku, stavový automat má 3 stavy:

- stav čtení hlavičky
- stav čtení velikosti datové části rámce
- stav čtení proměnné délky těla rámce

Jakmile je přečten celý rámec, jsou dekódované přijaté hodnoty uloženy do odpovídajících proměnných a proveden přesun do stavu čtení hlavičky. Tato variabilní část kódu je generována příkazy:

```
if (data_read == packet_size)
{
   %foreach idx = Model.NumOut
      %assign outportname = Model.OutPort[idx].varname
      memcpy((void*)&%<Model.OutPort[idx].varname>,(void*)&recvBuffer[offset],...
      sizeof(%<Model.OutPort[idx].varname>));
      offset+=sizeof(%<Model.OutPort[idx].varname>);
   %endforeach
   first_init = 1;
   CommServer_init();
}
```

Kód pro každý výstupní port vstupní periferie vloží do vygenerovaného souboru *CommServer.c* kód uzavřený ve *%foreach ... %endforeach*. Tím se vygeneruje kód, který uloží přijaté hodnoty do správných proměnných.

6.4.2.5 Ovladače sériové linky

Funkce *peert_make_rtw_hook.m* přidává při generování kódu modelu beany do výsledného PE projektu. PE projekt pro PIL simulace musí obsahovat bean pro obsluhu sériové linky *AsynchroSerial*. Dále musí být nastaveny vlastnosti přidaného beanu pomocí knihovny PESLIB.

Přidání beanu se provede příkazem *peslib* v souboru *peert_make_rtw_hook.m* v části *"entry"*.

```
[res,peMdlBeanId]=peslib(50,pebl,otBEAN,'<PEObject><Item name="PIL_as"...
beantype="AsynchroSerial"/></PEObject>');
```

Zároveň je prováděn test, zda-li přidání bylo úspěšné. V případě neúspěchu je zobrazena chybová hláška a generování kódu ukončeno.

Když jsou známy hodnoty parametrů *itm_symb* představující názvy vlastností beanu, je možné je programově nastavit. Pro tento účel má funkce *peslib* příkaz:

```
[rc,curval]=peslib( 54, peb, type, id, itm_symb, v)
```

Funkce se pokusí nastavit zadanou vlastnost *itm_symb* na hodnotu *v* a vrátí nastavenou hodnotu *curval*.

Následující příklad nastaví hodnotu přenosové rychlosti sériové linky:

```
% Set Baud rate
[res,curval]=peslib(54,pebl,otBEAN,peMdlBeanId,'_BdRate','115200 baud');
if res ~=0
 error(' Setting AsynchroSerial property in PE failed!');
end
```

Pro správnou funkci *AsynchroSerial* pro PIL simulaci je nutné tímto způsobem nastavit více vlastností. Tabulka A.1 shrnuje všechna potřebná nastavení.

Vlastnost[a]	**itm_symb**	**hodnota**[b]
Channel	Ser	SCI0
Interrupt service/event	IntService	Enabled
Input buffer size	InpBufferSize	2048
Output buffer size	OutBufferSize	2048
SCI output mode	SCIOutMode	Normal
Baud rate	_BdRate	115200 baud

Tabulka 6.2: Nastavení beanu AsynchroSerial pro PIL simulaci

[a]Vlastnost - uveden název vlastnosti, tak je používán v Processor Expertu na záložce Properties.

[b]Hodnota je vždy typu text.

6.4.2.6 Implementace časování simulace

Aby byla zajištěna všechna pravidla stanovená komunikačním protokolem (viz kapitola 5.2), musely být provedeny ještě další úpravy v PEERT targetu.

Zajištění čekání CPU na příchod prvního rámce

Čekání CPU na příchod prvního rámce zajišťují dva parametry modelu, které nastavuje funkce *gen_code_ctrl.m*.

Parametr *CustomHeaderCode* vkládá uživateský kód do hlavičkového souboru *model.h*

```
cHeaderCode = ['/* first received packet flag */',newLine,'extern int first_init;'];
set_param(hCtrl,'CustomHeaderCode',cHeaderCode);
```

Parametr *CustomInitializer* vkládá uživateský kód do funkce *model_initialize* v souboru *model_rtw.c*

```
cInitializer = ['/* wait for first packet, then start compute */',newLine,...
'while (!first_init)',newLine,'{',newLine,'}'];
```

```
set_param(hCtrl,'CustomInitializer',cInitializer);
```

Ve funkci *model_initialize* ve vygenerovaném souboru *model_rtw.c* se čeká na příjem prvního rámce od soustavy.

```
/* Model initialize function */
void model_initialize(boolean_T firstTime)
{
    ...
    /* user code (Initialize function Body) */
    /* wait for first packet, then start compute */
    while (!first_init)
    {
    }
}
```

Musí být ale ještě vypnuty přerušení od časovače *ModelBaseRateTimerInt*. Přerušení časovače *ModelBaseRateTimerInt* spouští krok výpočtu algoritmu regulátoru. Až po příjmu prvního rámce se zapnou přerušení tohoto časovače.

V souboru *peert_make_rtw_hook.m* je příkazem *peslib* nastaveno generování kódu metod *Enable* a *Disable*. Ty je pak možné v kódu využít. Ve funkci *placeMainCode.m* je zajištěno vkládání těchto metod do kódu souboru *model.c*.

Kód v *model.c* pro PIL simulaci volá funkci *RTW_initialize*, kde se čeká na příjem rámce od soustavy, a pak až jsou zapnuty přerušení od časovače, který spustí výpočet algoritmu.

```
ModelBaseRateTimerInt_Disable();
RTW_initialize(1);
ModelBaseRateTimerInt_Enable();
```

Ošetření příjímání rámce na CPU

Na CPU je potřeba ošetřit možný souběh v použití hodnoty proměnné při výpočtu algoritmu regulátoru a zápisu hodnoty do proměnné v události vyvolané příjmem rámce. Proto je po dobu výpočtu algoritmu vypnuto přerušení od sériové linky.

V souboru *peert_make_hook_rtw.m* je zapnuto generování kódu metod *Enable* a *Disable* beanu *AsynchroSerial* pomocí příkazu *peslib*. Ty lze pak v generovaném kódu používat.

V TLC souborech vstupních periferií je umístěn kód, který při zpracování první vstupní periferie vypne přerušení sériové linky.

```
%if AfterFirstIn == 0
   %assign ::AfterFirstIn = 1
   PIL_as_Disable();
%endif
```

Využívá k tomu globální proměnnou *AfterFirstIn* (deklarovaná v *peert.tlc*). Po zpracování první vstupní periferie inkrementuje její hodnotu, a tak již další vstupní periferie tento příkaz do kódu nevkládají.

V TLC souborech výstupních periferií je umístěn kód zapnutí přerušení sériové linky. Vkládán je před příkazem pro odeslání rámce z CPU (viz kapitola 6.4.2.2 v části *Modifikace TLC souborů periferií*).

Tento způsob není funkční pro případ, kdy model obsahuje blok se stavy. Takový blok může být zpracován dříve než bloky vstupních periferií.

V budoucím použití PIL simulace by bylo vhodné využít jiného ošetření možného souběhu. Např. použít cyklický buffer pro přijímání rámců od soustavy.

6.4.2.7 Výpis modifikací ve zdrojových souborech PEERT

Úpravy provedené ve zdrojových souborech je možné zjistit porovnáním obsahu výchozí verze PEERT a upravené verze pro PIL simulaci. Dobrým nástrojem pro porovnání obsahu souborů může být například program Total Commander.

Pro snažší orientaci v provedených změnách v jednotlivých souborech slouží tabulky 6.3 a 6.4. Podtržené soubory jsou soubory nové, nepodržené jsou soubory modifikované.

Soubor	Adresář	Provedená změna
serial_PE_support.c serial_PE_support.h	pil	funkce pro odesílání dat z CPU
project.mcp	pil	šablona CW projektu
pes_adc.tlc	block_std	generování kódu pro PIL simulaci vložení záznamu do *Model* vložení záznamu do peripheral_list.xml vypnutí přerušení od sériové linky

Tabulka 6.3: I. Modifikace zdrojových souborů PEERT

Soubor	Adresář	Provedená změna
pes_portio.tlc	block_std	generování kódu pro PIL simulaci vložení záznamu do *Model* vložení záznamu do peripheral_list.xml zapnutí přerušení od sériové linky
PILutilities.tlc	block_std	funkce pro vložení záznamu do peripheral_list.xml vytvoření souboru komunikačního serveru
addPILFilesToPe.m	peert	přidá soubory PE projektu
gen_code_ctrl.m	peert	generování modelu a kódu regulátoru
gen_code_plant.m	peert	generování modelu a kódu soustavy
gen_pil_filelist_mpf.tlc	peert	pro vytvoření seznamu souborů PIL podpory přidávaných do PE projektu
gen_rt_OneStep.tlc	peert	volání funkce pro tvorbu zdrojového souboru komunikačního serveru vložení prototypu funkcí serveru do modelname_Model_Api.h
peert.tlc	peert	parametr pro generování kódu pro PIL simulaci volání gen_pil_filelist_mpf.tlc vytvoření záznamu *Model* vytvoření a ukončení peripheral_list.xml
peert_make_rtw_hook.m	peert	volání funkce pro přidání souborů podpory PIL do PE projektu volání funkce pro modifikaci *Events.c* přidá ovladač sériové linky *AsynchroSerial* *Enable* a *Disable* metody časovače *ModelBaseRateTimerInt*
peert_num_inports.m	peert	vrací počet vstupních portů výstupních periferií
peert_pil.fig	peert	GUI targetu PEERT PIL
peert_pil.m	peert	funkce GUI targetu PEERT PIL
peert_settings.tlc	peert	parametr *UseModelInPIL*
placeMainCode.m	peert	vypnutí a zapnutí přerušení časovače *ModelBaseRateTimerInt*

Tabulka 6.4: II. Modifikace zdrojových souborů PEERT

6.5 Demonstrační úloha

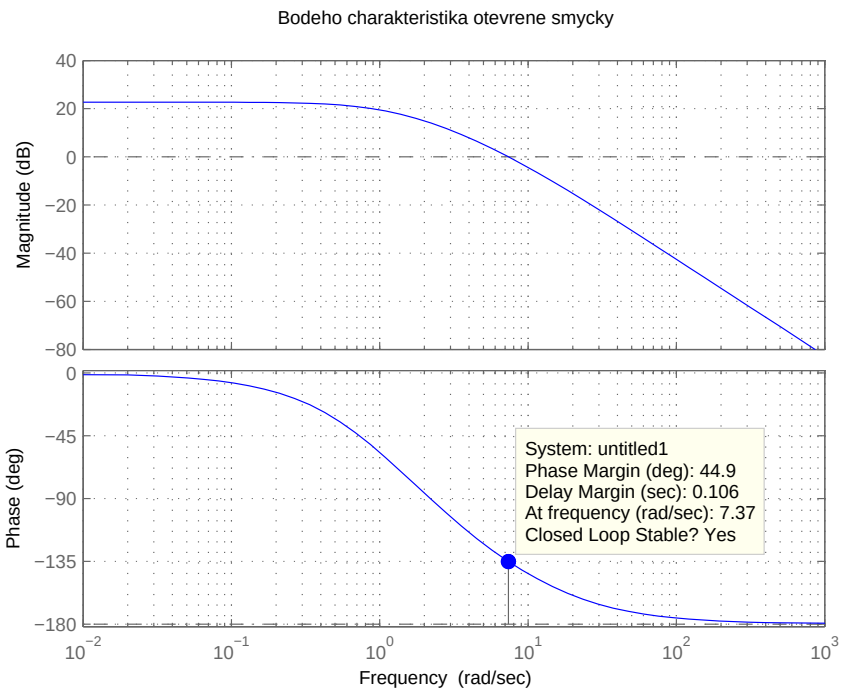
Výsledné řešení PIL simulace je prezentováno na demonstrační úloze (soubor *modelcl-fix.mdl*, na CD umístěn v adresáři *src\peslib_pil\demos\pil\modelcl_fixedpoint*). Byl vytvořen jednoduchý model řídicího systému.

Pro soustavu s přenosem

$$P = \frac{50}{(s+1)(s+4)(s+10)} \tag{6.1}$$

byl navržen PD regulátor s fázovou bezpečností $PM = 45°$

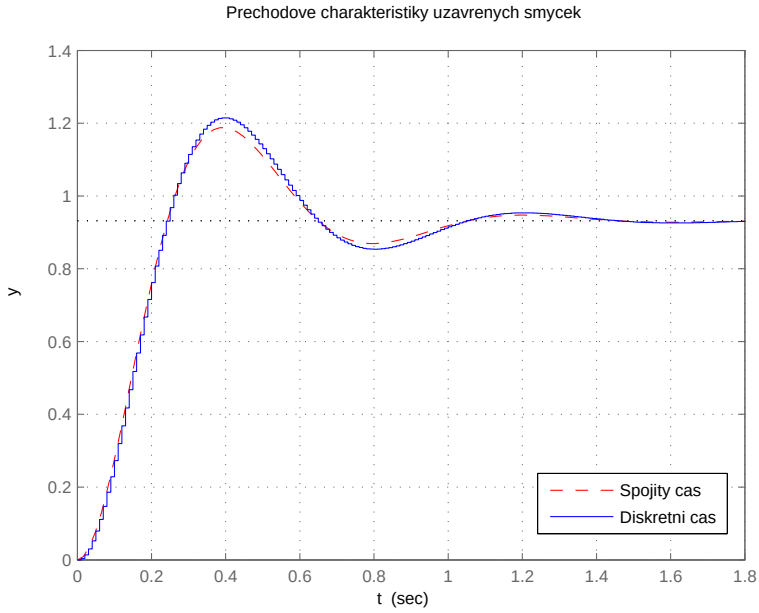
$$C = 1.49s + 10.95 \tag{6.2}$$



Obrázek 6.11: Bodeho charakteristika otevřené smyčky

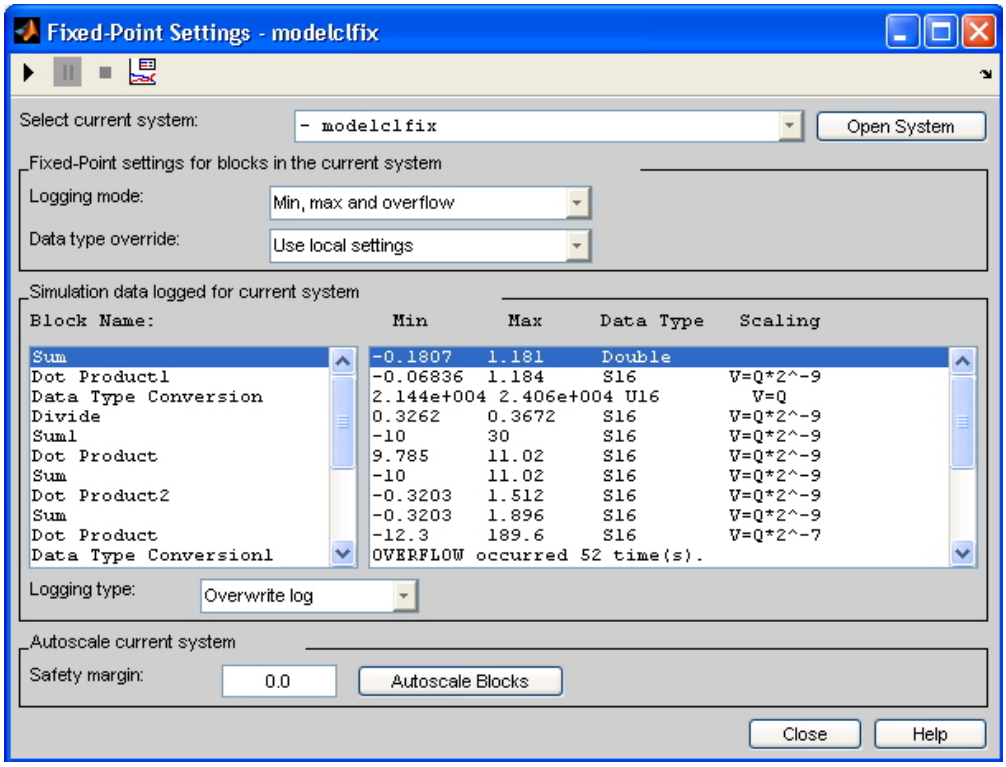
Soustava P i regulátor C byly diskretizovány s periodou vzorkování $T_s = 0.01s$. Porovnání spojité a diskrétní přechodové charakteristiky uzavřené smyčky lze vidět na obrázku 6.12.

Byl vytvořen model řídicího systému v Simulinku (viz obrázek 6.15). Regulátor i soustava jsou SISO. Na obrázku 6.15 mají tyto bloky dva výstupy (resp. dva vstupy), ale to je jen díky použití Byte2IO bloku, který vstupní hodnotu transformuje do dvou jednobytových čísel. Subsystém regulátoru využívá bloky periferií ADC a Byte2IO



Obrázek 6.12: Přechodové charakteristiky uzavřených smyček ($w- > y$)

z PE block setu, které simulují chování reálných periferií. Implicitní datový typ *double* používaný v Simulinku není vhodný pro 16 bitové procesory bez FPU.

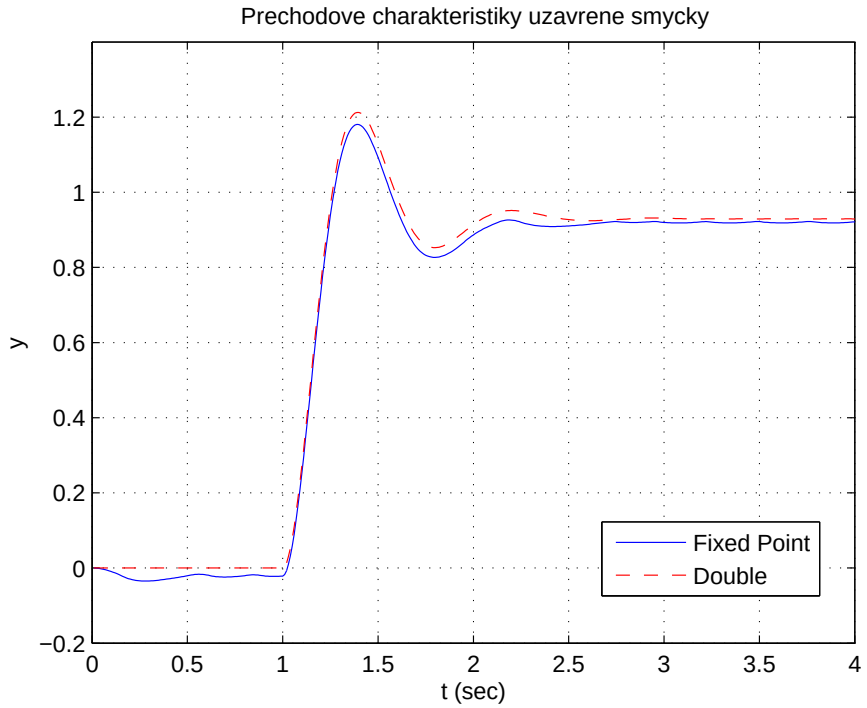


Obrázek 6.13: Grafické rozhraní pro Fixed Point

Proto byl použit *Fixed-point toolbox* [18] pro nastavení datových typů s pevnou řádovou

čárkou. Pro nastavování datových typů s pevnou řádovou čárkou je možné použít *Fixed Point GUI* (viz obrázek 6.13).

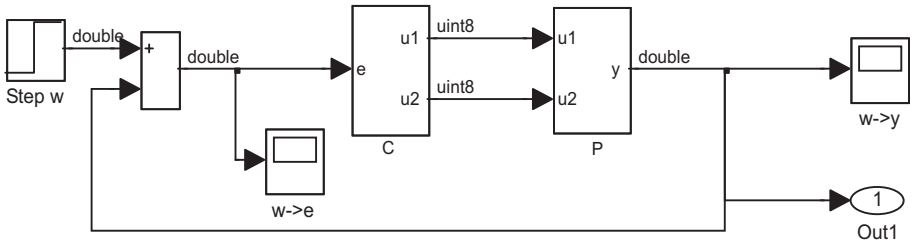
To umožňuje ve spuštěné simulaci ukládat minimální a maximální hodnoty na vstupech jednotlivých bloků modelu, a tak může pomoci při výběru datových typů.



Obrázek 6.14: Přechodové charakteristiky uzavřených smyček MIL

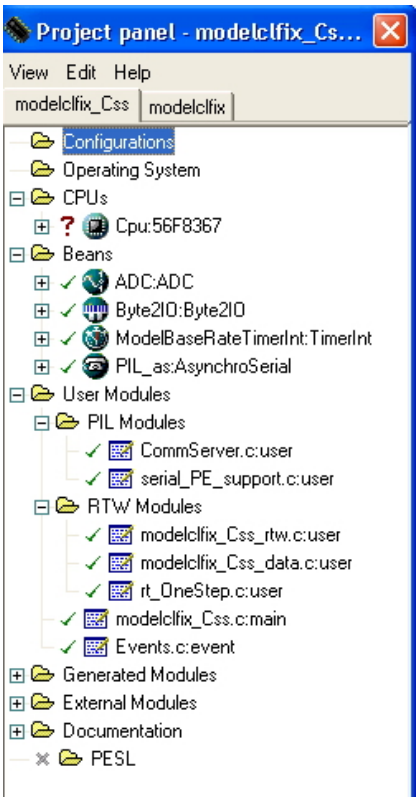
Subsystém regulátoru využívá blok periferie AD převodníku na svém vstupu. Příchozí hodnota má na výstupu rozlišení 12 bitů. Výstup AD bloku tak není nulový i v případě nulového vstupu na AD. Dalšími výpočty při použití *Fixed Point* vzniká zaokrouhlovací chyba. Rozdíl v přechodových charakteristikách uzavřených smyček je možné vidět na obrázku 6.14.

Demonstrační model (obrázek 6.15) obsahuje blok *Outport*. Ten se využívá pro nahrání vypočtených dat PIL simulace ze simulačního PC, na kterém je spuštěn výpočet algoritmu soustavy.



Obrázek 6.15: Demonstrační model

Po ověření simulací v Simulinku je vygenerován kód regulátoru i soustavy pomocí PEERT PIL grafického rozhraní (viz kapitola 6.2). Kód regulátoru je přeložen, nahrán na vývojovou desku a spuštěn. Algoritmus regulátoru čeká na příchod prvního rámce s vypočtenými hodnotami od soustavy.
Obsah vygenerovaného PE projektu je na obrázku 6.16. PE projekt pro PIL simulaci vždy obsahuje *AsynchroSerial* bean pro obsluhu sériové linky s názvem *PIL_as*. Obsažena je dále složka PIL se soubory *CommServer.c* a *serial_PE_support.c*.



Obrázek 6.16: Vygenerovaný projekt regulátoru

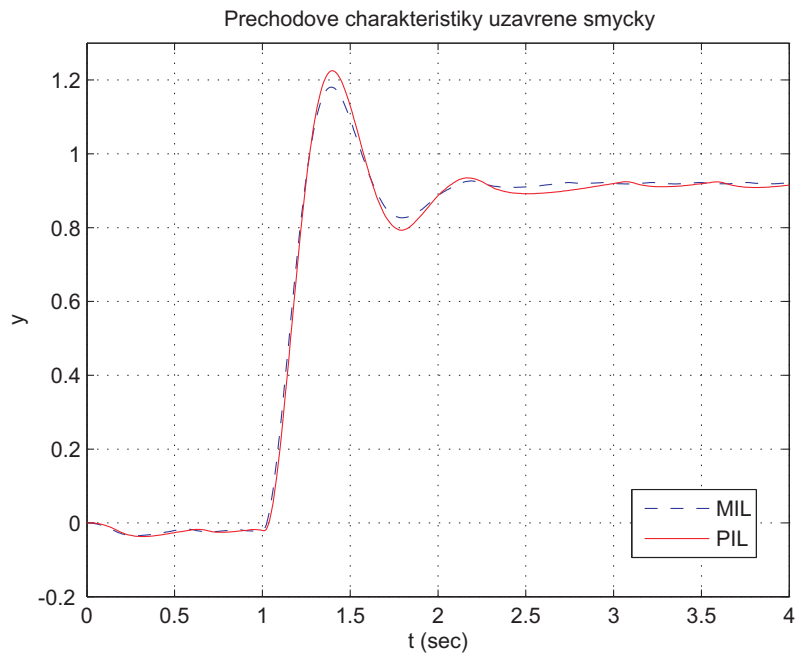
Kód soustavy je přeložen a nahrán na simulační PC. Běh aplikace soustavy je ovládán z host PC s Matlabem. Aplikace (a tím i PIL simulace) se spustí příkazem

$$tg.start \qquad \text{nebo} \qquad +tg.$$

Díky použití bloku *Outport* lze po dokončení PIL simulace uložit simulovaná data do Matlab příkazy:

$$time = tg.TimeLog;$$
$$values = tg.OutputLog;$$

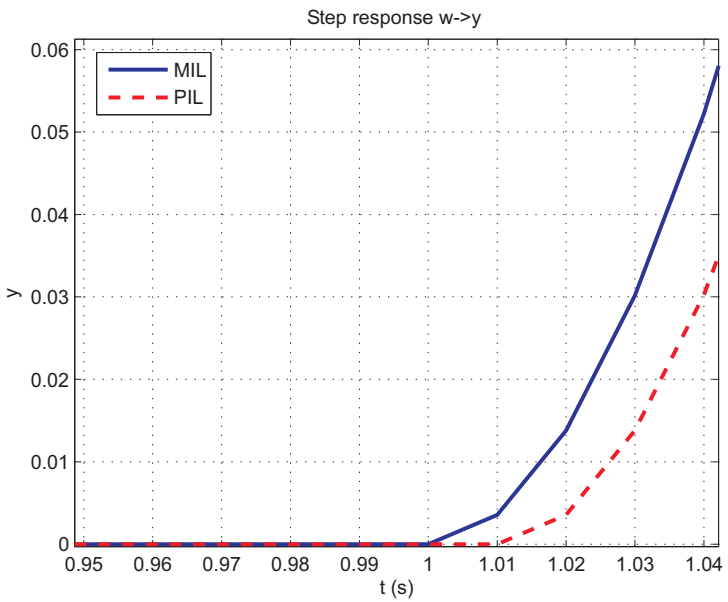
Vypočtená data se tímto ze simulačního PC přes ethernet propojení uploadují do host PC. Průběhy přechodových smyček z MIL (Simulink) a PIL simulace je možné vidět na obrázku 6.17.



Obrázek 6.17: Průběhy ze MIL a PIL simulace

6.5.1 Porovnání MIL a PIL simulace

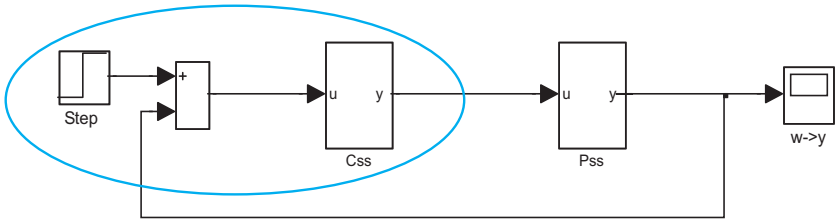
PIL simulace byla vyzkoušena na simulaci zpětnovazebního regulačního obvodu. Pohledem na detail náběhu přechodové charakteristiky na obrázku 6.18 lze vidět časové zpoždění 1 kroku simulace (na obrázku není detail z přechodové charakteristiky 6.17, ale jiný).



Obrázek 6.18: Detail náběhu přechodových charakteristik

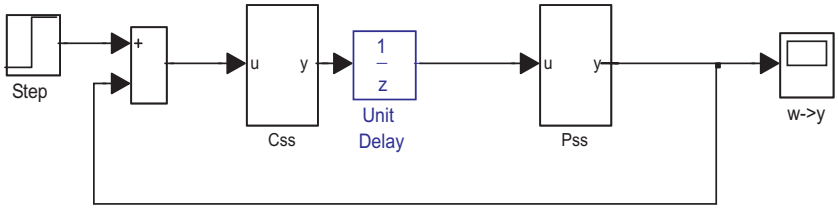
Toto zpoždění je způsobeno nutností odeslat data sériovou linkou do regulátoru, v regulátoru provést výpočet a v dalším kroku simulace vypočtená data z regulátoru přijmout.

Naproti tomu při softwarové simulaci v Simulinku zpoždění nevzniká. Jestliže je regulátor bez zpoždění, je možné získat výstup regulátoru reagující na změnu vstupu regulátoru v jednom kroku simulace. Na obrázku 6.19 se tedy výstup ohraničené oblasti spočítá v tom samém kroku simulace, jako nastala změna v referenci.



Obrázek 6.19: Model regulačního obvodu

Naproti tomu při PIL simulaci se projeví změna v referenci až v dalším kroku simulace. Jeden krok simulace je nutný pro komunikaci mezi CPU a simulačním PC. Toto odpovídá modelu z obrázku 6.20. MIL simulací modelu z obrázku 6.20 se získají výsledky shodné s výsledky PIL simulace modelu z obrázku 6.19.



Obrázek 6.20: Model regulačního obvodu s přidaným zpožděným

Při simulaci uzavřené smyčky je nutné, aby buď regulátor, nebo soustava obsahovaly zpoždění. Pokud by zpoždění nebylo obsaženo, schéma by obsahovalo algebraickou smyčku (více viz [4]). V tom případě by nešla spustit ani SIL simulace pod Simulinkem.

Časové zpoždění jednoho kroku simulace se projeví i v přenosu regulační odchylky na výstup soustavy.

Při návrhu regulačního obvodu pro PIL simulaci je potřeba s tímto 1 krokem časového zpoždění počítat. PIL simulaci nelze jinak implementovat.

6.6 Možnosti dalšího vývoje

Použití více vzorkovacích period v PIL simulaci

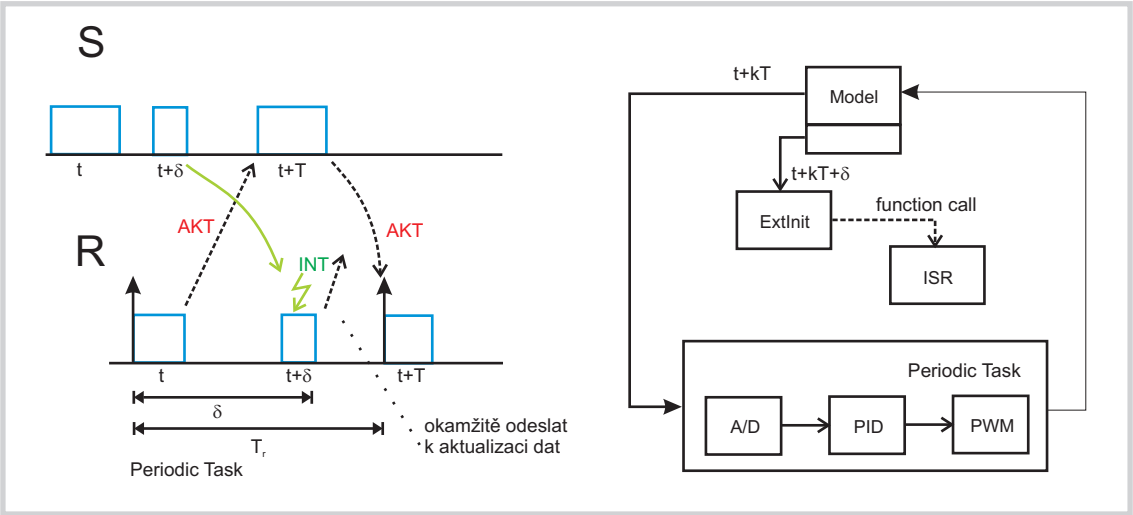
Model, který používá více vzorkovacích period, se nazývá multirate. V tomto případě musí být upraven blok pro stavbu rámce tak, aby odesílal hodnoty příslušné dané periodě.

Datový rámec bude obsahovat hlavičku se specifikací, jaké vzorkovací periody jsou posílány hodnoty.

Například při použití vzorkovacích period 0.01s a 0.1s je v čase simulace od 0s do 0.1s poslán 10xkrát rámec s hodnotami pro periodu 0.01s a 1xkrát rámec s hodnotami pro periody 0.01s i 0.1s.

Asynchronní události v PIL simulaci

Asynchronní události jsou v modelu v Simulinku reprezentovány function call portem (viz obrázek 6.21 vpravo). Pokud nastane asynchronní událost, musí být odeslán rámec k CPU (na obrázku označen *INT*). Příjem asynchronního rámce na CPU nastává v čase δ. Příjem asynchronních rámců umožňuje implementovaný komunikační server, viz kapitola 6.4.2.4.



Obrázek 6.21: Asynchronní události v PIL simulaci

Po dekódování rámce na CPU je spuštěn kód dané asynchronní události a odeslán rámec zpět k aktualizaci dat soustavy.

Linux real-time target

Motivací do budoucna je implementovat vlastní Linux real-time target s možností blokování dalšího výpočtu soustavy. Výpočet by byl blokován do té doby, než by přišel rámec od regulátoru (CPU). Toto ovladače sériové linky xPC targetu neumožňují.

Real-time časování u simulace soustavy není žádoucí, rychlé odezvy soustavy ano. Řešení s použitím S-funkcí a serveru (viz kapitola 4.3.4) umožňuje blokování, ale

dosažitelná rychlost komunikace je příliš nízká. Linux real-time target by tyto problémy vyřešil.

Výhodná je také možná budoucí rozšiřitelnost tohoto targetu díky otevřenosti systému Linux.

Kapitola 7

Závěr

Hlavním cílem předložené práce bylo vytvoření targetu a block setu implementující do Simulinku podporu processor in the loop (PIL) simulace.

PIL simulace využívá zapojení mikroprocesoru v uzavřené smyčce se simulačním softwarem (simulátorem). Procesor je přitom cílovým hardwarem, pro který je řídicí algoritmus vyvíjen a na němž bude jeho konečná podoba implementována.

Kód řídicího algoritmu je spuštěn na cílovém mikroprocesoru (MCU) a model řízené soustavy na simulátoru. Komunikačním kanálem jsou přenášena vypočtená data mezi MCU a simulátorem.

K řešení projektu byl využit PEERT (Processor Expert Embedded Real-time Target vyvíjený na Katedře řídicí techniky).

V úvodu práce byl uveden přehled souvisejících prací spolu s popisem výhod používání automaticky generovaného kódu při vývoji softwaru pro vestavné řídicí aplikace.

Následoval popis využitých nástrojů při implementaci vlastního řešení PIL simulace. Z vývojového balíku Matlab je využíván nástroj Simulink pro simulaci modelů, Real-Time Workshop (RTW) pro generování kódu soustavy a Real-Time Workshop Embedded Coder (RTW EC) pro generování kódu regulátoru. Dále je využíván nástroj Processor Expert, jehož použitím lze radikálně zrychlit vývoj vestavných aplikací. Target PEERT slouží k automatickému generování kódu pro vestavné aplikace, použitá knihovna PESLIB zprostředkovává komunikaci mezi programy Simulink a Processor Expert a lze za její pomoci nastavovat jednotlivé periferie hardwaru z prostředí Matlabu.

Principy generování kódu v Simulinku a používané S-funkce jsou taktéž v práci popsány.

Popsána byla principielní řešení processor in the loop simulace při použití Matlab Simulinku. Následoval přehled hotových řešení a provedená analýza možných řešení

PIL simulace při požadavku její práce v reálném čase.

Na základě porovnání výsledků testů s jednotlivými možnými řešeními PIL simulace bylo pro implementaci real-time PIL v rámci této práce vybráno řešení s xPC targetem. Z možných způsobů lze použitím xPC získat nejvýhodnější systém. Dosahuje nejvyšší vzorkovací rychlosti. Není nutné upravovat ovladače z jeho block setu. Pro budoucí využití lze použít i dostupné CAN ovladače. Produkt xPC target je distribuován s rozsáhlou dokumentací.

Byla popsána architektura zvoleného řešení. Bylo nutné vytvořit simulační bloky jeho periferií pro xPC target. Byl definován komunikační protokol a stanovena komunikační pravidla. Byly vytvořeny nové bloky pro stavbu a dekódování rámce v požadovaném formátu. Simulační aplikace na xPC targetu při PIL simulaci musí obsahovat model soustavy, ale i simulační bloky periferií. Pro tvorbu vlastních bloků periferií určených pro PIL simulaci byly využity zdrojové kódy bloků periferií targetu PEERT. Nové bloky tvoří PIL block set, který rozšířil stávající knihovnu targetu PEERT. V práci byl popsán způsob práce s novými bloky a jejich nastavení.

Byl vytvořen nový target PEERT PIL pro automatické generování kódu pro PIL simulaci. Je pomocí něj volán target PEERT pro generování kódu regulátoru a xPC target pro kód soustavy.

Pro zajištění podpory PIL simulace musely být upraveny zdrojové soubory targetu PEERT. Provedené modifikace a způsob práce s výsledným řešení PIL simulace byly v dokumentu popsány.

Práce s navrženým řešením PIL simulace byla demonstrována na ukázkové úloze. Simulací regulačního obvodu byla ověřena funkčnost navrženého prototypu PIL simulace. Byly porovnány výsledky PIL a MIL simulace a nastíněny možnosti dalšího vývoje.

V budoucnu je možné doplnit řešení PIL simulace o možnost použití modelů s více vzorkovacími periodami či využít asynchronní události. Na základě výsledků předkládaného řešení je již vyvíjen Linux real-time target pro simulaci soustavy.

Byl tedy navržen prototyp PIL simulace, který může být využit při vývoji vestavných řídicích systémů pro nastavování parametrů různých typů regulátorů při absenci reálných řízených soustav.

Uživatel předloženého řešení po vytvoření modelu s regulačním obvodem vygeneruje kód regulátoru. Ten poté nahraje do procesoru a spustí. Stejně tak vygeneruje kód soustavy a nahraje jej do simulátoru. Umožní se tak testování regulátoru v simulaci bez hotového hardware ECU a bez reálné soustavy.

Literatura

[1] Bartosinski, R.; Hanzálek, Z.; Waszniowski, L.; Stružka, P.; *Integrated Environment for Embedded Control System Design*.

[2] The MathWorks Inc. *Real-Time Workshop - User's Guide*. Version 6, 2005.

[3] dSpace TargetLink [online]. Poslední revize 2006-04-25 [cit. 2006-12-23], http://www.dspaceinc.com.

[4] The MathWorks Inc. *Simulink: Simulation and Model-Based Design - Using Simulink*. Version 6, 2005.

[5] The MathWorks Inc. *Simulink - Writing S-Functions*. Version 6, 2005.

[6] The MathWorks Inc. *Real-Time Workshop - Target Language Compiler*. Version 6, 2005.

[7] The MathWorks Inc. *Real-Time Workshop Embedded Coder - User's guide*. Version 4, 2005.

[8] The MathWorks Inc. *Real-Time Workshop Embedded Coder - Developing Embedded Targets*. Version 4, 2005.

[9] Processor Expert [online]. Poslední revize 2006-12-20 [cit. 2006-12-23], http://www.processorexpert.com.

[10] The MathWorks Inc. *MATLAB External interfaces*. Version 7, 2005.

[11] Freescale Semiconductor [online]. Poslední revize 2006-12-20 [cit. 2006-12-23], http://www.freescale.com.

[12] The MathWorks Inc. *Embedded Target for Motorola MPC555 User's Guide*. Version 2, 2005.

[13] HUMUSOFT s.r.o. [online]. Poslední revize 2006-10-27 [cit. 2006-12-23], http://www.humusoft.cz.

[14] The MathWorks Inc. *Real-Time Windows Target - User's Guide*. Version 2, 2006.

[15] The MathWorks Inc. *xPC Target - User's Guide*. Version 3, 2006.

[16] The MathWorks Inc. *xPC Target - I/O Reference*. Version 3, 2006.

[17] The MathWorks Inc. *xPC Target - Getting Started*. Version 3, 2006.

[18] The MathWorks Inc. *Simulin Fixed Point - User's guide*. Version 5, 2006.

Přílohy

Příloha A

Externí mód

Externí mód je speciálním simulačním módem podporovaným v Simulinku, jenž umožňuje spouštět výpočet simulací na externím procesoru. Simulace tak může probíhat v reálném čase. Externí mód není však využitelný pro PIL simulaci.

Přestože externí mód není využíván v PIL simulaci, byl v rámci práce implementován pro PEERT. Nabyté poznatky s vytvářením podpory externího módu a s prací v něm jsou obsahem této kapitoly.

A.1 Úvod do externího módu

Externí mód [2] je módem Real-Time Workshopu. Zajišťuje komunikační spojení mezi modelem v Simulinku a kódem spuštěném na procesoru. Je využit pro monitorování signálů a změny parametrů modelu.

Počítač se spuštěným Matlabem a Simulinkem je označován jako *host*. *Target* je počítač nebo procesor, na kterém je spuštěn kód.

Host odesílá zprávy s požadavky na target. Příkladem žádosti je změna parametrů či upload dat. Po příjmu zprávy target provádí požadovanou akci. Komunikace v externím módu je založena na *klient/server* architektuře. Klientem je přitom Simulink a target je server.

Externí mód umožňuje:

- Měnit parametry bloků modelu v reálném čase. Když je změněna hodnota parametru bloku, Simulink ji nahraje do spuštěného programu na targetu. Není tak nutné znovu kompilovat vygenerovaný kód při změně parametru.

- Monitorovat a ukládat vypočtená data z programu bez nutnosti vytváření speciálního interface.

Externí mód vytváří komunikační kanál mezi Simulinkem a programem na targetu. Transportní vrstva a kód využívající její funkce jsou umístěny v separovaných programových modulech. Targety tak mohou využívat různé transportní vrstvy. Většina targetů podporuje TCP/IP a RS232 sériovou komunikaci. Transportní vrstva pro sériovou komunikaci je implentována pouze pro 32-bitové Windows.

A.1.1 Současná podpora externího módu v jiných targetech

Externí mód není podporován ve všech targetech produkovaných firmou Mathworks. Dostupné targety lze rozdělit do 2 skupin:

- *s podporou externího módu*
  GRT, GRT malloc, ERT, Tornado, RSIM, RT Win, xPC
- *bez externího módu*
  c166, hc12, mpc555 [12], osekworks, TI C2000, TI C6000

V uživatelských příručkách jednotlivých targetů lze najít, zda je externí mód podporován či nikoliv. Z hlavičky systémového TLC souboru je možné zjistit stav podpory externího módu. Níže je uvedena hlavička targetu *Embedded Target for Infineon C166 Microcontrollers*:

```
%% SYSTLC: Embedded Target for Infineon C166(R) Microcontrollers
%% TMF: c166.tmf MAKE: make_rtw EXTMODE: no_ext_comm
%%
%% $RCSfile: c166.tlc,v $
%% $Revision: 1.12.6.7 $
%%
%% Copyright 1994-2004 The MathWorks, Inc.
%% Abstract: Embedded real-time system target file.
```

Pokud není externí mód implementován, je u *EXTMODE* uvedeno *no_ext_comm*. Navíc v TLC souboru nastavení (*target_settings.tlc*) je testováno, zda je zvoleno generování kódu s podporou externího módu. Pokud je vybráno a target přitom externí mód neumožňuje, je v průběhu generování kódu zobrazena chybová hláška. Pro ilustraci je zde tento test uveden:

```
%if ExtMode == 1
  %assign ExtMode = 0
  FEVAL("warndlg","External mode is not supported. This setting will be ignored.")
%endif
```

Společný rys skupiny targetů nepodporujících externí mód je ten, že jsou to tzv. embedded targety. V [8] se uvádí, že je netypické implementovat externí mód v embedded

targetu, protože externí mód je především rys rapid prototypingu. Nicméně je zde taktéž řečeno, že je možné vytvořit podporu externího módu použitím standardního API, o kterém lze zjistit více v dokumentaci k Real-Time Workshopu [2].

PEERT target je embedded target, implementací externího módu se bude lišit od ostatních embedded targetů, které obyčejně tuto možnost nenabízí.

A.1.2 Zdrojové soubory externího módu

Pro použití externího módu jsou k dispozici zdrojové kódy od MathWorks. Popis zdrojových souborů je uveden v uživatelské příručce Real-time Workshopu [2].

Real-Time Workshop dává k dispozici zdrojové kódy externího módu jak serveru, tak i klienta. Hlavní zdrojový soubor je:

- pro server:
  matlabroot\rtw\c\src\ext_mode\common\ext_svr.c
- pro klienta:
  matlabroot\rtw\ext_mode\common\ext_comm.c

Tyto soubory obsahují funkce, které volají funkce transportní vrstvy. Od MathWorks je již implementovaná transportní vrstva TCP/IP a sériová transportní vrstva. Dále bude popis omezen pouze na případ sériové transportní vrstvy, kterou využívá externí mód pro PEERT. Více informací o možnosti TCP/IP obsahuje [2].

Funkce sériové transportní vrstvy jsou umístěny v souborech:

- pro server:
  matlabroot\rtw\c\src\ext_mode\serial\ext_svr_serial_transport.c
- pro klienta:
  matlabroot\rtw\ext_mode\serial\ext_serial_transport.c

A.1.2.1 Popis zdrojových souborů klienta

Většina zdrojových souborů klienta je umístěna v adresáři *matlabroot\rtw\ext_mode*:

- *common\ext_comm.c*
  Tento soubor je jádrem komunikace externího módu. Komunikuje se Simulinkem pomocí datové struktury *ExternalSim*. S targetem komunikuje voláním funkcí transportní vrstvy. Zajišťuje spojení s targetem, download parametrů a ukončení spojení s targetem.

- *serial\ext_serial_transport.c*
  Obsahuje funkce transportní vrstvy. Využívá *ext_serial_utils.c*, který obsahuje funkce společné pro klienta i server.

- *common\ext_main.c*
  ext_main.c je interfacem pro Simulink. Používá funkce, které předávají požadavky Simulinku dále k *ext_comm.c*.

- *common\ext_convert.c* a *common\ext_convert.h*
  ext_convert.c obsahuje funkce pro konverzi datových formátů hosta a serveru. Dále provádí konverzi mezi big endian a little endian[1].

- *common\ext_sim.h*
  Definuje datovou strukturu *ExternalSim* a k ní přístupová makra.

- *common\ext_transport.h*
  Definuje funkce, které musí být implementovány transportní vrstvou.

A.1.2.2 Popis zdrojových souborů serveru

Zdrojové soubory serveru jsou umístěny v adresáři *matlabroot\rtw\c\src\ext_mode*:

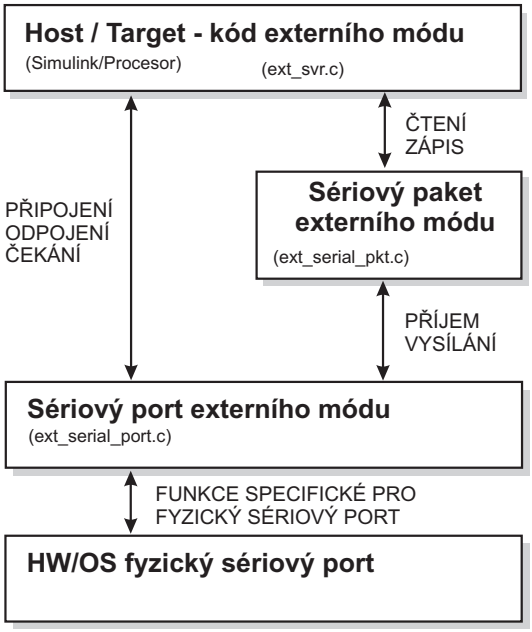
- *common\ext_svr.c*
  Je analogií k *ext_common.c* u hosta (klienta), ale je odpovědný za více úkolů. Navazuje a ukončuje spojení s klientem. Obsahuje funkce, které provádí změny v nastaveních hodnot parametrů modelu a funkce nutné pro odesílání vypočtených dat zpět Simulinku.

- *serial\ext_svr_serial_transport.c*
  Implementuje funkce transportní vrstvy. Využívá funkce *ext_serial_utils.c*

- *serial\ext_serial_utils.c*
  Obsahuje funkce a datové struktury pro komunikaci, funkce sériového komunikačního protokolu.

- *common\updown.c*
  updown.c ovládá model na targetu. Model na targetu je reprezentován vygenerovaným kódem. Při downloadu nových hodnot parametrů na target provádí zápis těchto nových hodnot. Poskytuje funkce pro upload dat z targetu.

[1] Endian definuje pořadí bytů u vícebytových čísel. Big endian ukládá vícebytová slova v pořadí od nezvýznamějšího bytu (tzv. MSB) k nejméně významnému (tzv. LSB). Nejvýznamější byte je uložen na nejnižší adrese v paměti. Opačné pořadí používá little endian. Více viz http://en.wikipedia.org/wiki/Endianness.



Obrázek A.1: Struktura kódu externího módu

- *serial\ext_serial_pkt.c* a *serial\ext_serial_pkt.h*
  Vkládá data k odeslání do objektu sériového paketu. Přijatá data transformuje taktéž do objektu sériového paketu. Na obrázku A.1 lze vidět strukturu kódu externího módu na straně serveru.

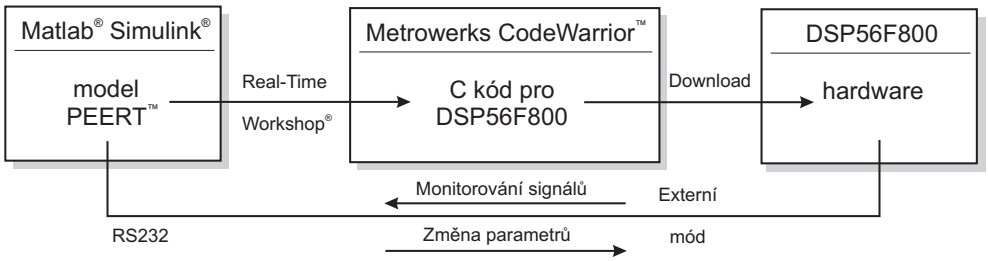
- *serial\ext_serial_port.c*[2]
  Je interfacem mezi kódem externího módu a fyzickým sériovým portem, viz obrázek A.1.

A.2 PEERT a externí mód

Na obrázku A.2 lze vidět strukturu řešení externího módu pro PEERT. Kromě Matlab Simulinku a Real-Time Workshopu je využito vývojového prostředí Metrowerks CodeWarrior s nástrojem Processor Expert a knihovny PEERT. Knihovna PESLIB zprostředkovává komunikaci mezi programy Simulink a Processor Expert a s její pomocí je možné nastavovat jednotlivé periferie hardwaru.

Z modelu se vygeneruje C kód, který se nahraje do 16 bitového procesoru řady DSP56800 (konkrétně typ 56F83674) a pomocí externího módu je vytvořeno spojení mezi procesorem a modelem v Simulinku.

[2] *ext_serial_port.c* je pouze zástupným názvem souboru. Tento soubor je odlišný pro použitý typ operačního systému a portu. Příkladem může být soubor *ext_serial_win32_port.c*, který obsahuje funkce pro přístup k sériovému portu PC, a přitom využívá funkce win32API.



Obrázek A.2: Architektura použitého řešení externího módu pro PEERT

V následujícím textu je popsán postup implementace externího módu pro PEERT. Text taktéž obsahuje popis práce s grafickým uživatelským rozhraním externího módu.

A.3 Úprava targetu PEERT

Tato podkapitola popisuje všechny provedené změny v targetu PEERT tak, aby podporoval externí mód. Jednotlivé modifikace jsou přitom uváděny v souvislosti se zajištěním nových funkcí nutných pro externí mód.

A.3.1 Generování kódu s podporou externího módu

Pro správně vygenerovaný kód využívající externí mód je nutné:

- umožnit použití externího módu - odstranění testu na výběr externího módu, který v případě generování kódu s externím módem doposud zastavoval proces generování kódu
- nastavit příznak generování kódu s externím módem pro preprocesor v CodeWarrioru

V targetu PEERT byl odstraněn test na výběr externího módu. V kapitole A.1.1 na str. 85 bylo již popisováno, že v targetech nepodporujících externí mód tento test způsobí zastavení generování kódu modelu. Zakomentování této části kódu (viz níže) v souboru *peert_settings.tlc* je tedy prvním krokem ke zprovoznění externího módu.

```
%%if ExtMode == 1
 %%assign msg = "External Mode is not currently supported. Unselect the \"External
 mode\" checkbox under ERT code generation options."
 %%exit %<msg>
%%endif
```

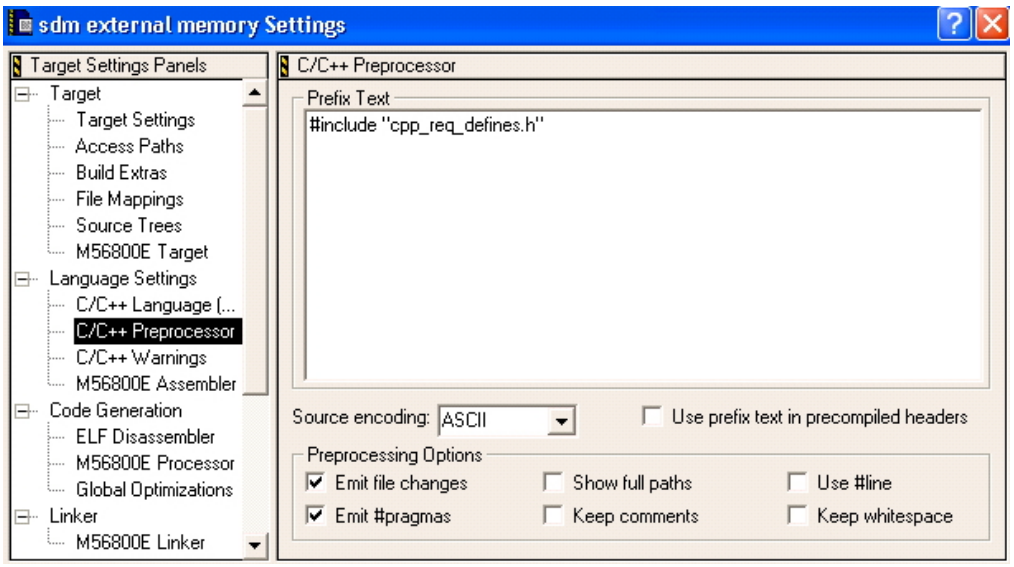
Po provedení této úpravy lze již spustit generování kódu s externím módem.

Generovaný hlavičkový soubor *cpp_req_defines.h* umožňuje CodeWarrior projektu přistupovat k flagům preprocesoru. Tento soubor je vždy obsažen v generovaném kódu.

Aby byl správně vygenerován kód modelu s podporou externího módu, musí být nastaven flag *EXT_MODE* na hodnotu 1. Proto byl upraven *gen_cpp_req_defines_h.tlc*, který vytváří soubor *cpp_req_defines.h*. Provedená modifikace

```
%if ExtMode == 1
  #define EXT_MODE 1
%endif
```

nastaví odpovídajícím způsobem příznak *EXT_MODE*. Nastavení preprocesoru v CodeWarrioru pro použití *cpp_req_defines.h* je možné vidět na obrázku A.3.



Obrázek A.3: Nastavení preprocesoru

A.3.2 Přidání zdrojových kódů externího módu

Pro přidání zdrojových kódů externího módu do PE projektu byly provedeny následující úpravy:

- v systémovém TLC souboru *peert.tlc* je volán *gen_as_filelist_mpf.tlc*
- vytvořen soubor *gen_as_filelist_mpf.tlc*
  - při generování kódu vytváří soubor *as_filelist.mpf*, ve kterém je uložen seznam přidávaných souborů do PE projektu
  - přidávány jsou soubory:

  * *ext_svr.c*
  * *updown.c*
  * *ext_work.c*
  * *ext_serial_pkt.c*
  * *ext_svr_serial_transp.c*
  * *ext_serial_PE_port.c*

- v *peert_make_rtw_hook.m* je volána funkce *addASFilesToPe* pro přidání souborů do PE projektu
- vytvořena *m* funkce *addASFilesToPe*
  - vstupem funkce je vytvořený soubor *as_filelist.mpf* se seznamem přidávaných souborů

Tímto je zajištěno přidávání zdrojových kódů externího módu do PE projektu.

A.3.3 Komunikační funkce na CPU

Ze šablony *matlabroot\rtw\c\src\ext_mode\custom\ext_serial_custom_port.c* byl vytvořen soubor *ext_serial_PE_port.c*, ve kterém jsou implementovány komunikační funkce pro CPU. V tomto zdrojovém souboru je nutné implementovat těla následujících funkcí:

- *ExtSerialPortCreate*
  Vytvoří objekt typu *ExtSerialPort*. Tento objekt je interfacem pro přístup k fyzickému sériovému portu.
- *ExtSerialPortConnect*
  Provádí připojení kódu externího módu s objektem typu *ExtSerialPort* a připojení k fyzickému sériovému portu.
- *ExtSerialPortDisconnect*
  Opak *ExtSerialPortConnect*, vše odpojí.
- *ExtSerialPortSetData*
  Pošle daný počet bytů na sériovou linku.
- *ExtSerialPortDataPending*
  Tato funkce vrací počet bytů čekajících na příjem ze sériové linky.
- *ExtSerialPortGetRawChar*
  Přijme jeden byte ze sériové linky.

Kód jednotlivých funkcí *ext_serial_PE_port.c* bude využívat bean *AsynchroSerial* a jeho metody. Využití funkcí tohoto zdrojového souboru lze vidět v obrázku A.1 (soubor *ext_serial_PE_port.c* je reprezentován popiskem *ext_serial_port.c*).

A.3.4 Ovladače sériové linky na CPU

Funkce *peert_make_rtw_hook.m* přidává při generování kódu modelu beany do výsledného PE projektu. Projekt pro externí mód musí obsahovat bean pro obsluhu sériové linky, tj. bean *AsynchroSerial*. Dále musí být nastaveny vlastnosti přidaného beanu pomocí knihovny PESLIB.

Bean pro sériovou linku se přidá pouze v případě externího módu (viz proměnná *val_ExtMode*):

```
if (strcmp(val_ExtMode,'on'))
 % Get FolderId
 peFolderId = char(getObjectId(getPeInfoData(pebl,otFOLDER),['name="',peAS,'"']));
 if isempty(peFolderId)
  % Add <peAS> folder to the PE project
  [res,peFolderId] = peslib(50,pebl,otFOLDER,['<PEObject><Item name="',peAS, ...
  '" foldertype="otUSR_PROG"/></PEObject>']);
  if res ~=0
   error([' Adding "',peAS,'" folder in PE failed!']);
  end
 end
end
```

Přidání beanu se provede příkazem *peslib* v souboru *peert_make_rtw_hook.m* v části *"entry"* . Zároveň je prováděn test, zda-li přidání bylo úspěšné. V případě neúspěchu je zobrazena chybová hláška a generování kódu ukončeno.

Když jsou známy hodnoty parametrů *itm_symb* představujících názvy vlastností beanu, je možné je programově nastavit. Pro tento účel má funkce *peslib* příkaz:

```
[rc,curval]=peslib( 54, peb, type, id, itm_symb, v)
```

Funkce se pokusí nastavit zadanou vlastnost *itm_symb* na hodnotu *v* a vrátí nastavenou hodnotu *curval*.

Následující příklad nastaví hodnotu přenosové rychlosti sériové linky:

```
% Set Baud rate
[res,curval]=peslib(54,pebl,otBEAN,peMdlBeanId,'_BdRate','57600 baud');
if res ~=0
 error(' Setting AsynchroSerial property in PE failed!');
```

```
end
```

Pro správnou funkci AsynchroSerial v režimu externího módu je nutné tímto způsobem nastavit více vlastností. Tabulka A.1 shrnuje všechna potřebná nastavení.

Přenosová rychlost nastavená na 57600 baud je maximální možná rychlost pro externí mód (viz [2]).

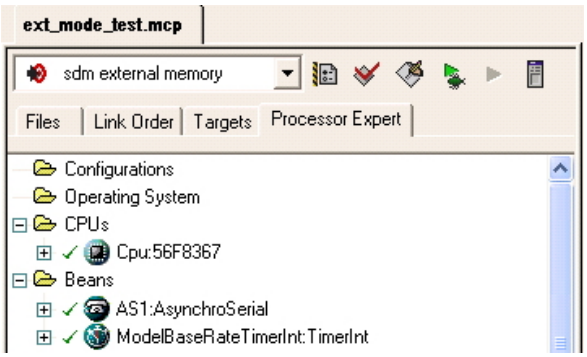
AsynchroSerial bean pro obsluhu sériové linky pod externím módem je tímto nastaven.

Vlastnost[a]	**itm_symb**	**hodnota**[b]
Channel	Ser	SCI0
Interrupt service/event	IntService	Enabled
Input buffer size	InpBufferSize	2048
Output buffer size	OutBufferSize	2048
SCI output mode	SCIOutMode	Normal
Baud rate	_BdRate	57600 baud

Tabulka A.1: Nastavení beanu AsynchroSerial pro externí mód

[a]Vlastnost - uveden název vlastnosti, tak je používán v Processor Expertu na záložce Properties.

[b]Hodnota je vždy typu text.



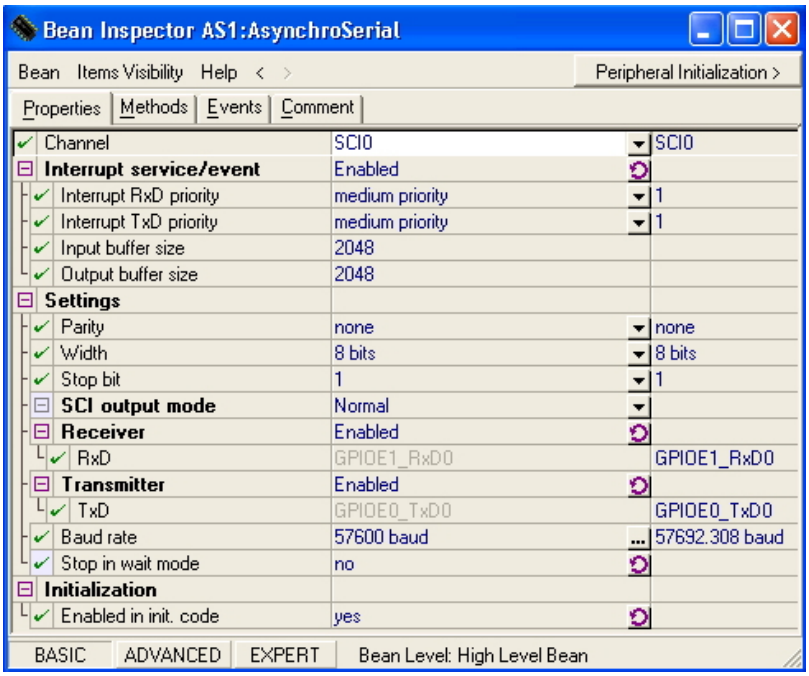
Obrázek A.4: AsynchroSerial bean v PE projektu

Na obrázku A.4 lze vidět přidaný bean *AsynchroSerial* pro obsluhu sériové linky AS1. Provedená nastavení beanu pro práci v externím módu jsou na obrázku A.5.

A.3.5 Komunikační protokol a kód pro PEERT

Kód generovaný targetem PEERT musí volat další funkce tak, aby byla zajištěna komunikace v externím módu. Proto musí být modifikován target PEERT:

- pro zajištění čekání CPU na příchod prvního rámce od Simulinku



Obrázek A.5: Nastavení AsynchroSerial beanu pro externí mód

- pro komunikaci nastavením priority přerušení hlavního časovače

Zajištění čekání CPU na příchod prvního rámce

Aby CPU čekal na příchod prvního rámce, musí být vypnuty přerušení časovače *ModelBaseRateTimerInt*. Přerušení časovače *ModelBaseRateTimerInt* spouští krok výpočtu aplikace spuštěné na CPU. Až po příjmu prvního rámce se přerušení tohoto časovače zapnou, a tak se spustí výpočet.

V souboru *peert_make_rtw_hook.m* je příkazem *peslib* nastaveno generování kódu metod *Enable* a *Disable*. Ty je pak možné v kódu využít. Ve funkci *placeMainCode.m* je zajištěno vkládání těchto metod do kódu souboru *model.c*.

Kód v *model.c* pro PIL simulaci volá funkci *RTW_initialize*, kde se čeká na příjem rámce od soustavy, a pak až jsou zapnuty přerušení od časovače, který spustí výpočet algoritmu.

```
ModelBaseRateTimerInt_Disable();
RTW_initialize(1);
ModelBaseRateTimerInt_Enable();
```

Priorita přerušení hlavního časovače

Pro zajištění komunikace mezi CPU a PC se Simulinkem je nutné nastavit přerušení časovače *ModelBaseRateTimerInt* na hodnotu *minimal priority*. Pokud je totiž nasta-

vena priorita vyšší, komunikace není bezchybná. Nastavení priority přerušení časovače je prováděnou v souboru *peert_make_rtw_hook.m*:

```
% Set ModelBaseRateTimerInt Interrupt_priority
if (strcmp(val_ExtMode,'on'))
  [res,curval]=peslib(54,pebl,otBEAN,peMdlBeanId,'CmpInitPriority',...
  'minimal priority');
  if res ~=0
    error(' Setting ModelBaseRateTimerInt Timer property in PE failed!');
  end
else
  [res,curval]=peslib(54,pebl,otBEAN,peMdlBeanId,'CmpInitPriority',...
  'medium priority');
  if res ~=0
    error(' Setting ModelBaseRateTimerInt Timer property in PE failed!');
  end
end
```

V případě generování embedded kódu je ponecháno přerušení na hodnotě *medium priority*. U externího módu je ale nastaveno na hodnotu *minimal priority*.

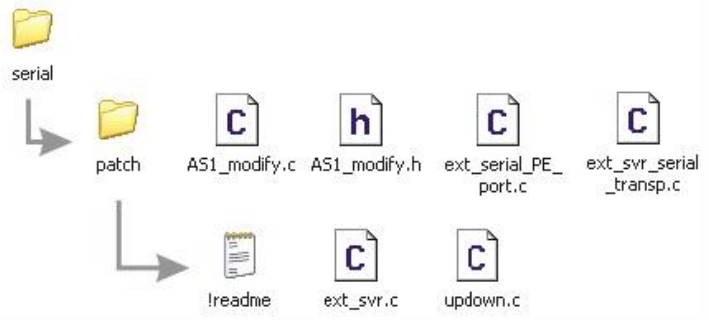
A.3.6 Výpis modifikací ve zdrojových souborech PEERT

Úpravy provedené ve zdrojových souborech je možné zjistit porovnáním obsahu výchozí verze PEERT a upravené verze pro PIL simulaci. Dobrým nástrojem pro porovnání obsahu souborů může být například program Total Commander.

Na obrázku A.6 lze vidět obsah modifikovaného targetu PEERT s podporou externího módu.

Obrázek A.6: Obsah PEERT s podporou externího módu

Novou složkou je složka *EXT_MODE*. Ta obsahuje podsložku *PATCH* se soubory, které je pro běh externího módu ve spolupráci PEERT nutné přehrát přes soubory původní (*ext_svr.c* a *updown.c*). Dále pak *AS1_modify.h*, *AS1_modify.c*, *ext_serial_PE_port.c* a *ext_svr_serial_transp.c*.



Obrázek A.7: Obsah adresáře serial

Pro snažší orientaci v provedených změnách v souborech slouží tabulka A.2. Modifikace souborů v adresáři PIL je popsána v kapitole A.4.1. Soubory nové jsou podtržené, soubory modifikované nejsou podtržené.

Soubor	Adresář	Provedená změna
addASFilesToPe.m	peert	přidá soubory PE projektu
gen_cpp_req_defines_h.tlc	peert	nastavení flagu EXT_MODE
peert.tlc	peert	volání gen_as_filelist_mpf.tlc
peert_make_rtw_hook.m	peert	volání funkce pro přidání souborů podpory externího módu do PE projektu přidá ovladač sériové linky *AsynchroSerial* *Enable* a *Disable* metody časovače *ModelBaseRateTimerInt* nastavení priority přerušení časovače *ModelBaseRateTimerInt*
peert_settings.tlc	peert	zakomentování kódu pro přerušení procesu generování v případě externího módu
placeMainCode.m	peert	vypnutí a zapnutí přerušení časovače *ModelBaseRateTimerInt*
AS1_modify.c AS1_modify.h	pil\serial pil\serial	úprava PE API beanu AsynchroSerial
ext_serial_PE_port.c	pil\serial	komunikační funkce na CPU
ext_svr_serial_transp.c	pil\serial	ext_svr_serial_transport.c a ext_serial_utils.c
ext_svr.c	pil\serial \patch	modifikovaný ext_svr.c pro PEERT
updown.c	pil\serial \patch	modifikovaný updown.c pro PEERT

Tabulka A.2: Modifikace zdrojových souborů PEERT pro externí mód

A.4 Ostatní změny nutné pro funkci externího módu v PEERT

A.4.1 Změny v kódu externího módu

Kvůli použití externího módu na 16 bitovém procesoru musely být provedeny změny, které jsou obsahem této podkapitoly.

A.4.1.1 Modifikace souborů transportní vrstvy

Soubor *ext_svr_serial_transport.c* je využíván při simulaci v externím módu. V tomto souboru je použit include souboru *ext_serial_utils.c*. Protože byl problém s překladem vygenerovaného kódu v CodeWarrior, byl vložen obsah souboru *ext_serial_utils.c* přímo do souboru *ext_svr_serial_transport.c*. Soubor byl pojmenován *ext_svr_serial_transp.c* a umístěn do adresáře *pil\serial* targetu PEERT.

Použitím nového souboru byly odstraněny problémy s překladem vygenerovaného kódu.

A.4.1.2 Zajištění čekání na startovací rámec

Kromě vypnutí přerušení od hlavního časovače spouštějící výpočet na procesoru (viz kapitola A.3.5) musela být ještě provedena úprava kódu externího módu tak, aby CPU čekal na příjem prvního rámce.

Vygenerované aplikaci (kódu) je obecně možné předávat vstupní parametry s příkazové řádky při spuštění aplikace. Tímto způsobem je předávána i informace, zda-li má aplikace spustit výpočet ihned nebo až po příjmu prvního rámce. V případě targetu PEERT je ale kód překládán z PE projektu, a tak nelze tímto způsobem předávat parametry aplikaci.

Proto musela být modifikována funkce *ExtWaitForStartPktFromHost* v souboru *ext_svr_serial_transp.c*. Byl do ní vložen kód:

```
UD -> WaitForStartPkt = 1;
```

Vygenerovaná aplikace pro simulaci v externím módu tak vždy čeká na příjem prvního rámce, teprve poté spustí výpočet algoritmu.

A.4.1.3 Zjištění velikosti datových typů

Při inicializaci komunikace mezi CPU a PC se Simulinkem je zjišťována velikost datových typů na CPU. Zdrojem problému při definici délek datových typů je to, že kód

pro target (CPU) je vygenerován jako 16 bitový, a přitom kód transportní vrstvy je napsán jako 32 bitový.

Musel být změněn kód ve funkci *ProcessConnectPkt* v souboru *ext_svr.c* (soubor *ext_svr* distribuován s PEERT jako patch, kterým se musí přehrát původní soubor pro správnou funkci s PEERT). Byl zakomentován původní příkaz a použita *memcpy* pro překopírovaní obsahu paměti, kde jsou uloženy velikosti datových typů:

```
//(void)memcpy(&tmpBuf[7], dtSizes, sizeof(uint32_T)*nDataTypes);
(void)memcpy(&tmpBuf[7], dtSizes, sizeof(uint_T));
(void)memcpy(&tmpBuf[8], dtSizes+1, sizeof(uint_T));
(void)memcpy(&tmpBuf[9], dtSizes+2, sizeof(uint_T));
...
```

Touto úpravou se již správně překopírují 16 bitové hodnoty *dtSizes* do 32 bitového bufferu *tmpBuf*.

A.4.1.4 Využití modifikovaného PE API beanu AsynchroSerial

Pro správné dekódování přijatých rámců na CPU muselo být upraveno programové aplikační rozhraní beanu AsynchroSerial (viz kapitola A.4.2). Nové funkce jsou využívány ve funkci *ExtGetPktBlocking* v souboru *ext_svr_serial_transp.c* pro čtení příchozích rámců:

```
while (GetExtSerialPacket(InBuffer, portDev)!=EXT_NO_ERROR)
{
  Init_ReadPtr();
}
AS1_DeleteReadChars();
```

Novými funkcemi zde použitými jsou *Init_ReadPtr* a *AS1_DeleteReadChars*.

A.4.2 Processor Expert API

Aby se předešlo problémům s alokováním paměti na CPU, je nutné nastavit na bloku Processor Expert v modelu pro simulaci v externím módu velikost zásobníku (*Stack Size*) a haldy (*Heap Size*). Bezchybný běh simulace pro testovací model (viz kapitola A.7) zajištěn pro nastavení parametrů:

- Stack Size: 2000B
- Heap Size: 8000B

Rámce posílané při komunikaci v externím módu využívají tzv. *escape character* znaky. Protokolem definovaný rámec je zakončen dvojicí bytů s hodnotou 3. Rámec tak nemůže obsahovat ve své datové části byte s hodnotou 3. Pokud je potřeba odeslat rámec s hodnotou 3 v jeho datové části, musí být tato hodnota předem přemaskována.

Escape character znaky nenesou v rámci žádnou datovou informaci, slouží pouze ke správnému dekódovaní přijaté hodnoty. Velikost datové části rámce, kterou lze přečíst z hlavičky, neudává velikost včetně escape character znaků, ale velikost bez těchto znaků.

Pro zajištění bezchybného příjmu rámců s escape character znaky bylo upraveno API beanu AsynchroSerial. Modifikací byly vytvořeny *AS1_modify.c* a *AS1_modify.h*, které jsou umístěny v adresáři *ext_mode\serial*. Jejich použití je popsáno v kapitole A.7. Soubor implementují funkce:

- *void AS1_Init_ReadPtr(void)*
  - pointer pro čtení dat z bufferu nastaví na začátek bloku dat
- *byte AS1_ReadChar(AS1_TComData *Chr)*
  - přečte znak bez jeho výběru z bufferu, rozdíl od funkce *RecvChar*
- *byte AS1_ReadBlock(AS1_TComData *Ptr,word Size,word *Rcv)*
  - přečte blok dat bez jeho výběru z bufferu, rozdíl od funkce *RecvBlock*
- *void AS1_DeleteReadChars(void)*
  - pointer pro čtení dat z bufferu nastaví na konec bloku přečtených dat, data jsou tak vybrána z bufferu

A.5 Komunikační protokol externího módu

Počítač se spuštěným Matlabem a Simulinkem je označován jako *host*. *Target* je počítač nebo procesor, na kterém je spuštěn kód.

Host odesílá zprávy s požadavky na target. Příkladem žádosti je změna parametrů či upload dat. Po příjmu zprávy target provádí požadovanou akci. Komunikace v externím módu je založena na *klient/server* architektuře. Klientem je přitom Simulink a target je server.

A.5.1 Druhy rámců

Druhy rámců jsou definovány v *matlabroot\rtw\c\src\ext_mode\common\ext_share.h*. Pořadové číslo rámce je uvedené v závorce v hexadecimální soustavě. Toto číslo obsahuje osmý byte většiny z těchto rámců pro jejich identifikaci. Rámce je možné rozdělit do několika skupin:

- rámce k targetu (tj. k CPU)
  - rámce pro navázání a ukončení spojení
    - EXT_CONNECT *(00)*
    - EXT_DISCONNECT_REQUEST *(01)*
    - EXT_DISCONNECT_REQUEST_NO_FINAL_UPLOAD *(02)*
    - EXT_DISCONNECT_CONFIRMED *(03)*
  - rámce pro práci s parametry
    - EXT_SETPARAM *(04)*
    - EXT_GETPARAMS *(05)*
  - rámce pro upload dat
    - EXT_SELECT_SIGNALS *(06)*
    - EXT_SELECT_TRIGGER *(07)*
    - EXT_ARM_TRIGGER *(08)*
    - EXT_CANCEL_LOGGING *(09)*
    - EXT_CHECK_UPLOAD_DATA *(0A)*
  - rámce pro ovládání modelu
    - EXT_MODEL_START *(0B)*
    - EXT_MODEL_STOP *(0C)*
  - rámec požadavku dat
    - EXT_GET_TIME *(10)*
- rámce z targetu (tj. k Simulinku) - odpovědi
  - EXT_CONNECT_RESPONSE *(11)*
  - EXT_DISCONNECT_REQUEST_RESPONSE *(12)*
  - EXT_SETPARAM_RESPONSE *(13)*
  - EXT_GETPARAMS_RESPONSE *(14)*

- EXT_MODEL_SHUTDOWN *(15)*
- EXT_MODEL_SHUTDOWN_DATA_PENDING *(16)*
- EXT_GET_TIME_RESPONSE *(17)*
- EXT_MODEL_START_RESPONSE *(18)*
- EXT_UPLOAD_LOGGING_DATA *(1C)*
- EXT_SELECT_SIGNALS_RESPONSE *(1D)*
- EXT_SELECT_TRIGGER_RESPONSE *(1E)*
- EXT_ARM_TRIGGER_RESPONSE *(1F)*
- EXT_CANCEL_LOGGING_RESPONSE *(20)*
- EXT_TERMINATE_LOG_EVENT *(21)*
- EXT_TERMINATE_LOG_SESSION *(22)*

Pro nejdůležitější rámce je dále uveden popis jejich tvaru. Jednotlivé byty rámce jsou vždy vyjádřeny v hexadecimální soustavě.

Potvrzovací paket (ACK)

Jako potvrzovací rámec je využíván rámec složený pouze z hlavičky a zakončení:

```
7E 7E 02 00 00 00 00 03 03
```

Význam jednotlivých bytů:

- *7E 7E* - pevný začátek hlavičky, vždy 2B
- *02* - definice typu rámce v hlavičce, vždy 1B; rozlišuje pouze dvě hodnoty: 02 - ACK rámec, 01 - ostatní rámce
- *00 00 00 00* - specifikace velikosti datové části rámce v hlavičce, vždy 4B
- *03 03* - pevné zakončení rámce, vždy 2B

Paket pro připojení (EXT_CONNECT)

První rámec (startovací) vyslaný od Simulinku směrem k targetu před navázáním komunikace má tvar:

```
7E 7E 01 08 00 00 00 65 78 74 2D 6D 6F 64 65 03 03
```

V rámci je specifikována datová velikost rámce na 8 bytů (viz *08 00 00 00*) a rámec obsahuje řetězec *ext-mode* (hodnoty *65 78 74 2D 6D 6F 64 65*).

Odpovědi na startovací paket (EXT_CONNECT_RESPONSE)

Po příjmu startovacího rámce odešle CPU tři odpovědi směrem k Simulinku. První rámec má formát:

```
7E 7E 01 08 00 00 00 11 00 00 00 08 00 00 00 03 03
```

V tomto speciálním rámci je při jeho příjmu na PC interpretována hodnota velikosti datové části rámce jako počet bitů v 1B na targetu (CPU). Díky tomuto rámci také Simulink dedukuje endian targetu. *Typ odpovědi specifikuje osmý byte.*

Druhým rámcem je Simulinku řečeno, jak velká bude datová část dalšího rámce:

```
7E 7E 01 08 00 00 00 11 00 00 00 54 00 00 00 03 03
```

Další poslaný rámec bude mít velikost datové části 84 bytů

Ve třetím rámci odpovědi jsou poslány velikosti datových typů na targetu. Příklad tvaru při použití 16 bitového procesoru:

```
7E 7E 01 54 00 00 00 76 43 83 CE 51 0D 78 B0 37 72 DB CE 66 C6 DB 29 00 00 00 00 01
00 00 00 0E 00 00 00 04 00 00 00 04 00 00 00 01 00 00 00 01 00 00 00 02 00 00 00 02
00 00 00 04 00 00 00 04 00 00 00 01 00 00 00 04 00 00 00 02 00 00 00 02 00 00 00 04
00 00 00 08 00 00 00 03 03
```

Velikost datové části rámce je 84B (viz *54 00 00 00*). Prvních 16 bytů datové části představují čtyři 4 bytové kontrolní součty. Následuje 4 bytový parametr *intCodeOnly*, který určuje, zda-li je na targetu použit pouze typ integer. Další 4B představují *targetStatus* a další počet datových typů, zde 14 (viz *0E 00 00 00*).

Ve zbývající části jsou poslány velikosti datových typů na targetu. Pro lepší orientaci jsou vždy střídány podržené 4B a nepodtržené. Poslány jsou velikosti datových typů *real_T*, *real32_T*, *int8_T*, *uint8_T*, *int16_T*, *uint16_T*, *int32_T*, *uint32_T*, *boolean_T*, *fcn_call_T*, *int_T*, *pointer_T*, *action_T* a 2**uint32_T* (uvedeno postupně od prvních podtržených čtyřech bytech).

Paket uploadu proměnných (EXT_GETPARAMS)

Rámec vysílající požadavek k nahrání přístupných parametrů má tvar:

```
7E 7E 01 08 00 00 00 05 00 00 00 00 00 00 00 03 03
```

Simulink blokuje další chod simulace do té doby, než přijde odpověď.

Získání času simulace (EXT_GET_TIME)

Tvar rámce vyjadřující požadavek pro získání simulačního času z targetu:

```
7E 7E 01 08 00 00 00 10 00 00 00 00 00 00 00 03 03
```

Start simulace modelu (EXT_MODEL_START)

Vysláním následujícího rámce ze Simulinku je spuštěn model (vygenerovaný kód) na targetu:

```
7E 7E 01 08 00 00 00 0B 00 00 00 00 00 00 00 03 03
```

Výběr signálů pro upload (EXT_SELECT_SIGNALS)

Tento rámec je využíván je v případě, že v modelu existují tzv. tunable parametry, které lze za běhu simulace měnit. V případě nastavených tunable parametrů je ve vygenerován kódu obsažen soubor *model_data.c*, kde *model* je název modelu. Tunable parametry se nastavují v dialogovém okně nastavení parametrů modelu v nabídce *Optimization* (více viz dokumentace k Real-Time Workshopu).

Posláním následujícího rámce Simulink určuje, že další posílaný rámec ze Simulinku bude mít velikost datové části 60 bytů (viz podtržená část):

```
7E 7E 01 08 00 00 00 06 00 00 00 3C 00 00 00 03 03
```

Dalším vyslaným rámcem od Simulinku je:

```
7E 7E 01 3C 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 02
00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 01
00 00 00 00 00 00 00 C4 8C 00 00 03 03
```

Význam jednotlivých polí v rámci je popsán v souboru *updown.c* (*updown.c* je jedním ze zdrojových souborů externího módu).

Podtržené 4 byty je hodnota bufferu na targetu, o jehož alokaci Simulink žádá. Simulink žádá o alokaci bufferu velikosti 36036 bytů. Protože velikost paměti na použitém procesoru je menší, musela být modifikována funkce *UploadLogInfoInit* v souboru *updown.c*. Úpravou je funkci *UploadBufInit* předána hodnota 2048 představující velikost bufferu. Tento upravený soubor *updown.c* je distribuován s PEERT jako patch, kterým se musí přehrát původní soubor pro správnou funkci s PEERT.

Paket nastavení uploadu (EXT_SELECT_TRIGGER) a odeslání dat z targetu (EXT_UPLOAD_LOGGING_DATA)

Významy jednotlivých polí obou dvou rámců jsou popsány v souboru *updown.c*.

A.5.2 Externí mód a reálný čas

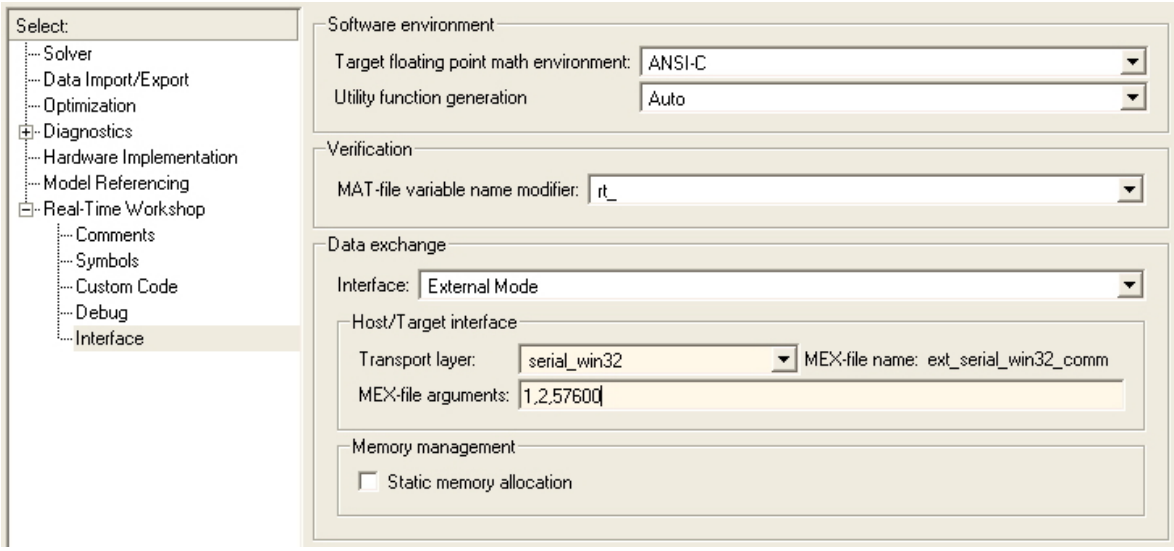
Jestliže není v průběhu simulace pod externím módem odeslán rámec odpovědi s hodnotou simulačního času *EXT_GET_TIME_RESPONSE*, simulace v Matlabu se pozastaví bez jakékoliv chybové hlášky. Simulink blokujícím čtením čeká na příchod odpovědi.

A.6 Práce s externím módem

A.6.1 Grafické uživatelské rozhraní externího módu

Generování kódu pro externí mód

Generování kódu s podporou externího módu se nastavuje v dialogovém okně nastavení parametrů modelu (viz obrázek A.8).



Obrázek A.8: Nastavení interface externího módu

V nabídce *Real-Time Workshop -> Interface* je potřeba nastavit parametry na panelu *Data exchange*.

Parametr *Interface* je nutné nastavit na *External Mode*. V případě propojení hosta a targetu sériovou linkou je nastaven parametr *Transport layer* na *serial_win32*. Volitelně je možné nastavit parametr *MEX-file arguments*:

- první argument zapíná/vypíná (hodnoty 1/0) výpisy do konzole Matlabu v průběhu simulace v externím módu
- druhý argument představuje použitý COM port (na obrázku tedy komunikace přes COM2)

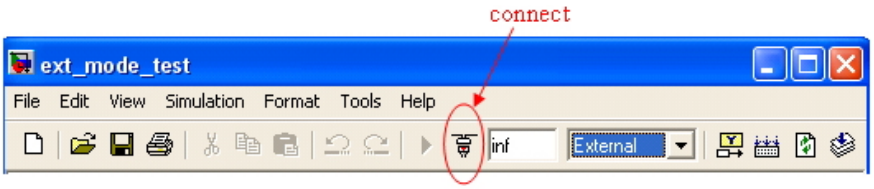
- třetí argument je použitá přenosová rychlost (57600b/s je maximální možná rychlost pro externí mód)

S nastavenými parametry podle obrázku A.8 je již možné vygenerovat kód s podporou externího módu.

A.6.2 Ovládání simulace v externím módu

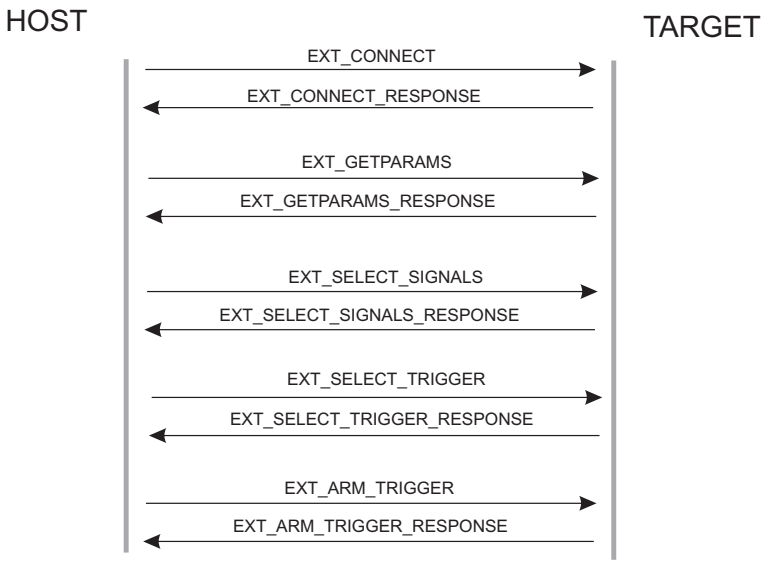
Inicializace komunikace mezi hostem a targetem

Připojení k targetu se provede stiskem tlačítka *Connect* v ovládacích prvcích modelu v Simulinku (viz obrázek A.9). Přitom musí být vybrán typ simulace *External*. Při připojení k targetu dochází k výměně několika rámců.



Obrázek A.9: Tlačítko pro připojení k targetu

Prvním rámcem je *EXT_CONNECT*. Timeout pro čekání Simulinku na odpověď targetu je nastaven na 2 minuty (v *matlabroot\rtw\ext_mode\common\ext_comm.c* lze zjistit hodnotu timeoutu). Pokud do tohoto času nepřijde odpověď, Simulink zobrazí chybovou hlášku. Přehled komunikovaných rámců při připojení k targetu je možné vidět na obrázku A.10.



Obrázek A.10: Připojení k targetu

Po příjmu posledního rámce z obrázku je host připojen k targetu. V tomto stavu si mezi sebou host a target neustále vyměnují rámce s hodnotami simulačních časů a čekají na spuštění simulace. Start simulace se spustí stiskem tlačítka *Start real-time code*.

Výpis do konzole Matlabu

V Matlabu je možné nastavit vypisování informací o externím módu do konzole Matlabu. Následující výpis představuje právě výpis při připojení hosta k targetu:

```
host endian mode: Little
byte swapping required: false
target integer only code: false
action: EXT_CONNECT
action: EXT_GETPARAMS
action: EXT_SELECT_SIGNALS
Got EXT_SELECT_SIGNALS_RESPONSE from target for upInfoIdx 0 with status OK.
action: EXT_SELECT_TRIGGER
Got EXT_SELECT_TRIGGER_RESPONSE from target for upInfoIdx 0 with status OK.
action: EXT_ARM_TRIGGER
Got EXT_ARM_TRIGGER_RESPONSE from target for upInfoIdx 0 with status OK.
```

Záznam komunikace při inicializaci

Při připojení k targetu si host a target vymění následující rámce (záznam komunikace při použití PEERT a externího módu):

Rámce od targetu (CPU):

```
7E7E02000000000303
7E7E0108000000110000000080000000303
7E7E0108000000110000000540000000303
7E7E0154000000A1460E4E384272EF41187D5E539362FCCA00000000010000000E0000000400000004000000100 ...
00000100000002000000020000000400000004000000010000000400000002000000020000000400000008000000303
7E7E02000000000303
7E7E0108000000140000000080000000303
7E7E0108000000000000040000040400303
7E7E02000000000303
7E7E02000000000303
7E7E01080000001D0000000080000000303
7E7E02000000000303
7E7E0108000000000000000000000000303
7E7E0108000000170000000040000000303
7E7E02000000000303
7E7E0104000000000000000303
7E7E02000000000303
```

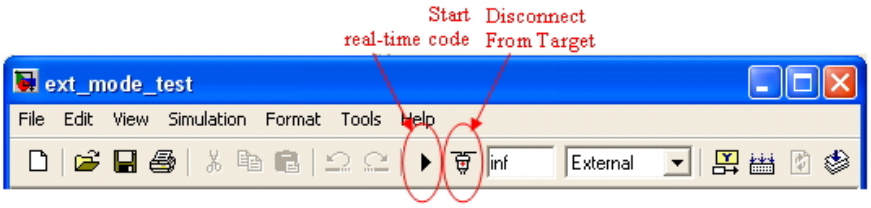
7E7E01080000001E000000080000000303
7E7E01080000000000000000000000000303
7E7E02000000000303
7E7E02000000000303
7E7E01080000001F000000080000000303
7E7E01080000000000000000000000000303
7E7E02000000000303

Rámce od hosta (Simulinku):

7E7E01080000006578742D6D6F64650303
7E7E02000000000303
7E7E02000000000303
7E7E02000000000303
7E7E01080000000500000000000000000303
7E7E02000000000303
7E7E02000000000303
7E7E01080000000600000003C0000000303
7E7E013C0000000000000000100000000000000010000000000000000200000000000000000000000000100000000000 ...
000000000000100000001000000000000000C48C00000303
7E7E01080000001000000000000000000303
7E7E02000000000303
7E7E02000000000303
7E7E01080000000700000020000000000303
7E7E02000000000303
7E7E01200000000000000000000000000E87D2300000000000000000000000000000000000000000000000000303
7E7E02000000000303
7E7E02000000000303
7E7E02000000000303
7E7E01080000000800000000400000000303
7E7E01040000000000000000000303
7E7E02000000000303
7E7E02000000000303

Spuštění a ukončení simulace

Jakmile je host připojen k targetu, změní se vzhled ovládacích tlačítek modelu (viz obrázek A.11). Tlačítkem *Start Real-Time Code* lze simulaci spustit a tlačítkem *Disconnect From Target* se lze odpojit od targetu.



Obrázek A.11: Ovládací tlačítka simulace v externím módu

V případě zapnutých výpisů do konzoli Matlabu se po stisku tlačítka *Start Real-Time*

Code vypíše:

```
action: EXT_MODEL_START
Got EXT_MODEL_START_RESPONSE packet from target.
```

Simulaci je možné ukončit dvěma způsoby:

- tlačítkem *Stop Real-Time Code*
  - výpis:

    ```
    action: EXT_MODEL_STOP
    Got EXT_MODEL_SHUTDOWN packet from target.
    action: EXT_DISCONNECT_CONFIRMED
    ```

- tlačítkem *Disconnect From Target*
  - výpis:

    ```
    action: EXT_DISCONNECT_REQUEST
    Got EXT_DISCONNECT_REQUEST_RESPONSE packet from target.
    action: EXT_DISCONNECT_CONFIRMED
    ```

Změna parametrů

Hodnoty parametrů modelu je možné za běhu simulace měnit pouze v případě, že jsou nastaveny jako *tunable*. Parametr lze měnit z příkazové řádky Matlabu nebo z dialogového okna bloku modelu v Simulinku. Po každé změně musí být proveden update zkratkovou klávesou *CTRL+D* nebo nabídkou *Edit -> Update*. Updatem se provede nahrání nové hodnoty parametru do targetu.

A.7 Demonstrační model

Instalace PEERT

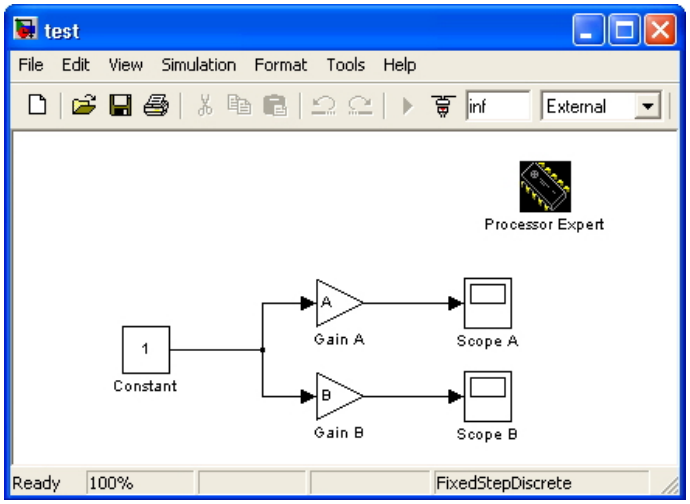
- rozbalit někam archiv peslib-2006-05-08.zip (archiv obsahuje demo "test", viz níže)
- spustit setup.m pro přidání targetu do Matlab RTW
- soubory ext_svr.c a updown.c z archivu ze složky *ext_mode\patch* zkopírovat do adresáře *matlabroot\rtw\c\src\ext_mode\common*

Generování kódu z modelu

- *výchozí stav:* smazat všechny již vygenerované soubory z modelu
  - dobré taky při přechodu z normálního módu do external mode a naopak (kódy již vygenerované - např. modename.c - se totiž při dalším generování již znovu nevygenerují, pokud existují...- a to je nežádoucí funkce, protože modelname.c pro normální a external mode je odlišný)
- pak při spuštění tohoto modelu se zobrazí hláška s informací, že chybí příslušný PE projekt; taktéž dotaz, zda-li chceme vytvořit nový PE projekt - dát ano
- v nově vytvořeném projektu je potřeba přidat bean procesoru 56F8367 a v Build options nastavit Stack size na 2000 byte, Heap size na 8000 byte a v PE compiler/linker support nastavit Tool directory k aktuálnímu umístění instalace CodeWarrior
- standardně vybrat peert.tlc jako cílový target, na záložce Interface/Data Exchange vybrat "external mode" s transportní vrstvou "serial_win32"
- dobré použít MEX argumenty: např. 1, 1, 57600 (tj. VERBOSE, COM port, baud rate - verbose na 1 pro výpis stavu do Matlab konzole)
- po skončeném generování kódu je potřeba uložit model - uloží se tak vygenerovaný PE projekt
  - také dokopírovat do adresáře soubor mcp a přejmenovat ho na stejný název jako název modelu
- spustit mcp, nastavit access paths:
  - *matlabroot\simulink\include*
  - *matlabroot\rtw\c\src*
  - *matlabroot\extern\include*
  - *..\..\ext_mode\serial*
- spustit make pro zkompilování kódů beanů - doplní kódy beanů, které byly prázdné kvůli použití standalone demíčka pod Matlabem
- projekt bude obsahovat chyby spojené s upraveným PE API beanu AsynchroSerial
- pro odstranění chyb je potřeba změnit v Processor Expert/Options/Project Options/Preserve User Changes na "yes", a pak na beanu AsynchroSerial dát Code Generation/Preserve User Changes in Generated Bean Modules

- do AS1.c v Generated Modules/Bean modules dát na konec souboru #include "AS1_modify.c"
- v model_rtw.c zakomentovat výpis do konzole $(void)printf("\backslash n**starting\ the\ model**\backslash n");$
  - výpis do konzole v CodeWarrioru většinou přeruší komunikaci po sériové lince!
- za komentář /* signal main to stop simulation */ umístit kód: $model\_M->Timing.tFinal = -1;$
  - to způsobí, že program na targetu (procesoru) bude běžet nekonečně

Demo "test"



Obrázek A.12: Testovací model pro externí mód

- demo je pro Fixed Step Size = 0.01s, změnu parametrů lze provést změnou v Matlab konzoli, a poté nabídkou Edit/Update Diagram nahrát změnu parametru do procesoru...
- pro simulaci musí být vybrána nabídka external mode
- po spuštění programu na targetu se lze připojit tlačítkem "Connect", a poté spustit simulaci tlačítkem "Start real-time code"

Příloha B

Obsah přiloženého CD

Součástí této práce je přiložené CD obsahující elektronickou verzi diplomové práce a zdrojové kódy targetu PEERT. CD obsahuje několik adresářů:

- **doc**
  - obsahuje elektronickou verzi diplomové práce
- **src**
  - **peslib_050907**
    výchozí verze targetu PEERT, na kterém byly prováděny úpravy v rámci této práce
  - **peslib_pil**
    modifikovaná verze PEERT umožňují PIL simulaci
  - **peslib_extmode**
    upravený PEERT podporující externí mód
    * **videos**
      obsahuje záznam simulace v externím módu
  - **others**
    * **rt_toolbox**
      obsahuje model pro test Real-Time Toolboxu
    * **sfun_server**
      aplikace serveru a S-funkce pro test PIL simulace s S-funkcemi a serverem
    * **baseboard_serial**
      demonstrační modely pro práci s blokem baseboard_serial

Na CD je obsažen soubor *index.html* s dalšími informacemi.