České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Podpora simulace s hardware ve smyčce

Praha, 2008 **Autor: Pavel Jelínek**

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 10. 1. 2008

podpis

Poděkování

Děkuji všem, kteří mi poskytli cenné rady, především pak vedoucímu mé diplomové práce Ing. Liborovi Waszniowskému, Ph.D. za jeho odbornou pomoc.
Chtěl bych také poděkovat mé ženě Magdaleně a naší čerstvě narozené dceři Kateřině, že respektovaly můj čas strávený nad touto prací a po celou dobu mne podporovaly.

Abstrakt

Tato diplomová práce se zabývá vytvořením podpory pro generování C kódu z modelu v Simulinku, známé jako target, pro hardware s operačním systémem GNU Linux.
Práce popisuje vytvoření základního targetu a jeho následné rozšíření o práci v reálném čase, propojení se Simulinkem prostřednictvím external modu a šablonu na vytvoření bloků pro vstupně/výstupní zařízení, jejíž použití je demonstrováno na bloku asynchronní sériové linky RS232. Tím je vytvořena kompletní podpora simulace s hardware v uzavřené smyčce (hardware-in-the-loop).
Pozornost je také věnována vytvoření křížových překladačů pro překlad vygenerovaného C kódu pod operačním systémem Windows XP s procesorem x86, kde je spuštěn Simulink, pro operační systém GNU Linux s procesory x86 a PowerPC (jádro 603e). K překladu jsou využity emulátory Linuxu (Cygwin a MinGW). Je popsána také manipulace s binárním kódem mezi emulátory Linuxu, kde je kód vytvořen a operačním systémem GNU Linux na cílovém počítači.
Použití targetu je prezentováno na demonstrační úloze. Závěrem je naznačen další možný vývoj targetu.

Abstract

This graduation thesis deals with creation of a support for generation of a C code from a Simulink model, known as a target, for a hardware with the operating system GNU Linux.

The thesis describes creation of a basic target and his further extension by real time execution, external mode and a template for creation of an input/output interface used for asynchronous serial line RS232 interface.

An attention is also focused on creation cross compilers for compilation of the generated C code under operating system Windows XP with processor x86, where Simulink is running, for Linux operating system with processors x86 or PowerPC (core 603e). Linux emulators (Cygwin and MinGW) are used for compilation. The manipulation of the compiled code between Linux emulators and operating system GNU Linux is described.

The potential of the final solution and its usage are presented in a demo application. The further development of the target is suggested.

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky *Školní rok:* 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Pavel Jelínek

Obor: Technická kybernetika

Název tématu: Podpora simulace s hardware ve smyčce

Zásady pro vypracování:

1. Seznamte se s principy simulace hardware in the loop a s možnostmi její podpory v programu Matlab Simulink a real time workshop.
2. Navrhněte a implementujte podporu pro generování kódu z modelu v Simulinku a jeho spouštění pod operační systémem Linux, případně jiným operačním systémem.
3. Navrhněte a implementujte způsob využití této podpory pro simulace hardware in the loop.
4. Použití této technologie demonstrujte na vhodném příkladě.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Libor Waszniowski, Ph.D.

Termín zadání diplomové práce: zimní semestr 2006/2007

Termín odevzdání diplomové práce: leden 2008

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

v zast.

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 21.02.2007

Obsah

Seznam příloh

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato diplomová práce se zabývá vytvořením podpory pro simulace s hardware ve smyčce, která je známá v anglické literatuře jako **Hardware in The Loop Simulation** (HIL Simulation). Nejprve se tedy zmiňme, jaký je význam takové simulace a do jakého článku vývojového řetězce ji můžeme zařadit. Z tohoto pak bude patrné, co je míněno vytvořením podpory pro takovou simulaci a jaké požadavky jsou na ni kladené.

Význam simulace **HIL** (Hardware In the Loop) spočívá v zapojení hardware do regulační smyčky libovolného vyvíjeného a doposud ještě ne zcela kompletního systému. Takový hardware je pak zařízení, které bude součástí finálního systému a tedy splňuje již v době simulace požadavky na vstupně výstupní rozhraní, výpočetní výkon apod.

Simulace **HIL** jsou nejčastěji prováděny na konci vývojového řetězce, kdy je nutné verifikovat funkčnost vyvíjeného systému ještě před konečnou realizací. Této simulaci často předcházejí simulace Model in The Loop (MIL) a Processor in The Loop (PIL), jejichž porovnání je popsané v článku [1], který byl autorem publikován během řešení této diplomové práce.

Při simulaci s hardware ve smyčce je jasné, že takový hardware musí obsahovat nějaký program, který provádí specifickou činnost. Např. výpočet regulačního zásahu. Často se vybere HW s vhodnými vlastnostmi pro určitou třídu úloh (hardware s konkrétním procesorem a vstupně/výstupními rozhraními). Vlastnosti HW se poté dají jen ztěží modifikovat. Těžiště úlohy se tedy přesouvá na zmíněný program.

Variabilita takového programu vede na využití nějakého operačního systému (OS), který poskytuje množství již vytvořených nástrojů včetně ovladačů hardware. Použití OS významně zjednoduší tvorbu programu a umožní jeho přenositelnost na různý hardware.

1.1 Způsob řešení

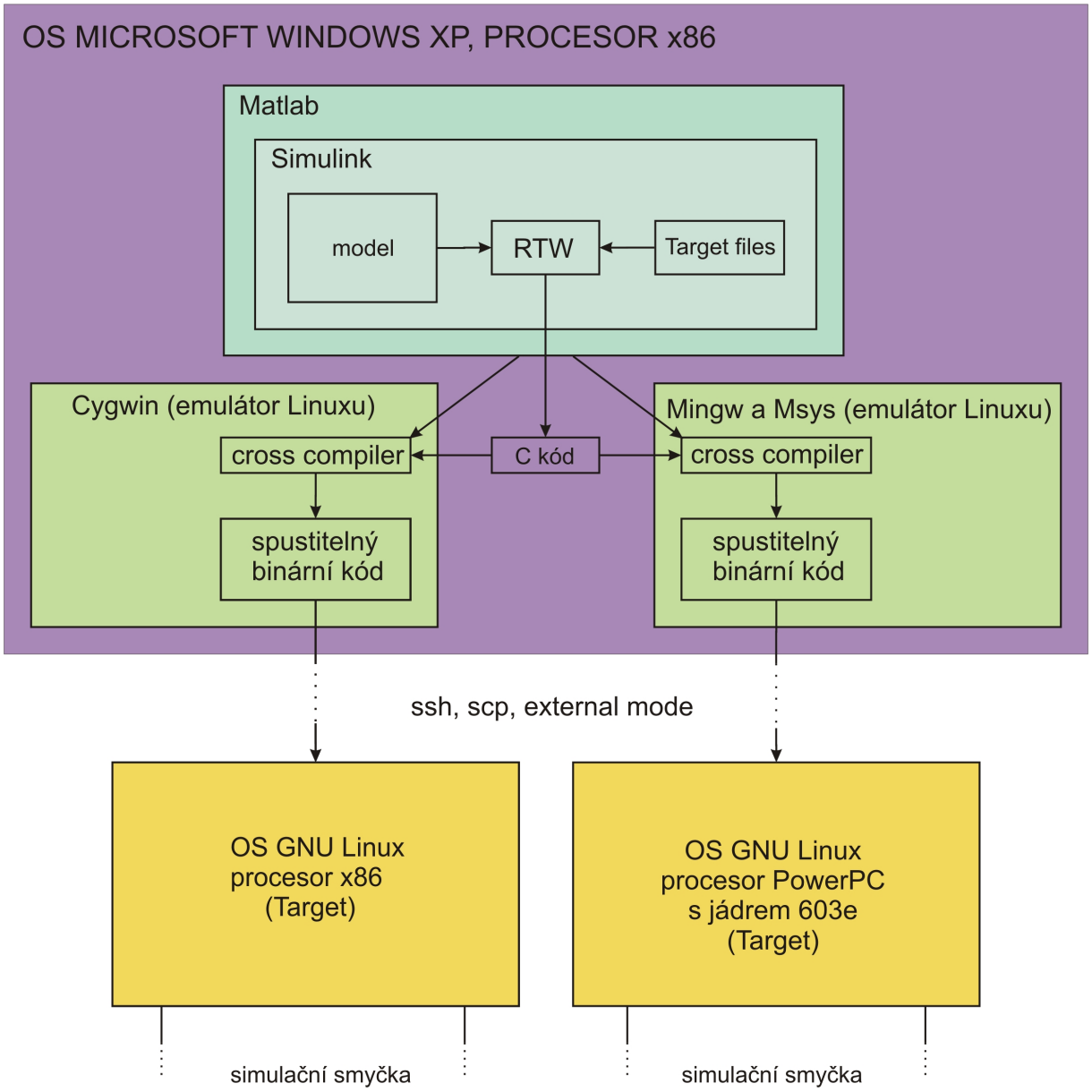
Ze zadání diplomové práce byl pro hardware v simulační smyčce použit OS GNU Linux, který poskytuje množství vytvořených a volně dostupných prostředků pro práci s hardware přes spuštěný binární kód (program), který je nutné vytvořit. Co tedy využít pro generování programu pro OS GNU Linux, aby bylo možné jako vstup zadávat dostatečnou míru abstrakce řešeného problému (např. model) a nemuseli tak pracovat přímo s programovacími jazyky? Ze zadání diplomové práce byl použit software od firmy The Mathworks, Inc. (viz. [2]) a to **Simulink** s nástrojem **Real Time Workshop**(RTW).

Simulink dovoluje modelovat dynamické systémy, vlastně je skládat z databáze vytvořených funkčních bloků a ty vhodně propojovat. Tím lze velmi jednoduše namodelovat dynamické systémy, které je možné v Simulinku spouštět, sledovat průběhy veličin a stavů takových modelů. To se však děje v rámci software Simulinku. Simulink však obsahuje nástroj, který umožní z takových modelů vygenerovat kód a tento kód přeložit zvoleným překladačem tak, aby výsledný spustitelný soubor, reprezentující model, mohl být spuštěn mimo software Simulinku a to dokonce i na jiném počítači (hardware) a operačním systému. Takový nástroj se nazývá **Real Time Workshop** (RTW). K seznámení se s tímto nástrojem doporučuji ještě před pokračováním v četbě literaturu [4].

Aby bylo možné nástrojem **RTW** (Real Time Workshop) generovat z modelu v Simulinku C kód, je nutné vytvořit tzv. **target**. Toto označení v sobě ukrývá množinu souborů, které řídí generování kódu z modelu v Simulinku a jeho překlad pro cílovou platformu. Takový target bude vytvořen pod jménem **linux_grt_target** pro platformu OS GNU Linux.

Cílovou platformou je tedy OS GNU Linux. Tato informace však nestačí a je ještě nutné vybrat okruh hardware, na kterém OS GNU Linux poběží a tím tedy i okruh hardware, který bude zapojen do simulační smyčky, tedy i do budoucích reálných aplikací. Pro své široké rozšíření a možnost připojení množství modulů s již vytvořenými ovladači byl vybrán hardware s procesory x86, tedy velké množství vestavěných jednodeskových počítačů a převážná většina osobních počítačů. Dále pak hardware s procesory PowerPC s jádrem 603e, které jsou často využívány v embedded aplikacích. Procesor PowerPC s jádrem 603e obsahuje také modul Boa5200 (viz. [5]), který je používán na Katedře řídící techniky a který bude použit v demonstrační úloze této diplomové práce. **Cílové platformy** jsou tedy dvě a to **OS GNU Linux , procesor x86** a **OS GNU Linux, procesor PowerPC s jádrem 603e**.

Termín **Target** s velkým počátečním písmenem **bude reprezentovat cílovou plat-**



Obrázek 1.1: Princip podpory HIL simulace

formu. Pokud bude nutné odlišit cílovou platformu, bude tak explicitně učiněno. Malé počáteční písmeno ve slově **target** pak **bude reprezentovat množinu souborů, které řídí generování kódu** z modelu v Simulinku **a jeho překlad** pro Target, tedy vytvářený **linux_grt_target**.

Implementace podpory pro generování C kódu z modelu v Simulinku pod OS Windows XP pro Target OS GNU Linux (x86) a OS GNU Linux (PowerPC 603e) spočívá ve vytvoření targetu **linux_grt_target**. Dále vytvoření křížových překladačů vygenerovaného C kódu pro zmíněné Targety pod OS Windows XP, protože simulink obsahuje pouze nativní překladač. Nakonec vytvoření způsobu, jak automaticky nakopírovat na Targety přeložený binární kód a jak ho spustit. Celý proces je znázorněn na obr. 1.1.

Využití podpory simulace s hardware ve smyčce spočívá v implementaci rozšířeného targetu, tedy rozšíření základního targetu o běh v reálném čase a vstupně výstupní rozhraní. To jsou dvě rozšíření, která umožní, aby hardware byl součástí reálné simulační smyčky. Tyto rozšíření tedy umožní, aby hardware ve smyčce komunikoval přes vstupně výstupní rozhraní a jeho program běžel v reálném čase.

1.2 Struktura diplomové práce

Každá kapitola obsahuje stručný teoretický úvod, případně odkaz na vhodnou literaturu. Dále pak následuje řešení tématu podle názvu kapitoly a ve vztahu k vytvářenému targetu. Kapitola 2 popisuje vytvoření základní podpory pro generování C kódu z modelu v Simulinku, který se již dá přeložit a spustit na Targetech. Samotný proces generování kódu je popsán v kapitole 3 a jeho překlad v kapitole 4. Následuje kapitola 5 o způsobu manipulace s výsledným binárním spustitelným kódem mezi emulátory Linuxu pod OS Windows XP a Targety včetně jeho spuštění na Targetech. Kapitola 6 popisuje rozšíření základního targetu **o práci v reálném čase**, rozhraní **External mode** pro vizualizaci a parametrizaci přes ethernetové rozhraní a šablonu pro **vstupně výstupní rozhraní** aplikovanou na asynchronní sériovou linku RS232.

V přílohách B a C lze nalézt základní popis využitých emulátorů OS Linux pod OS Windows XP včetně nastavení jejich prostředí pro spolupráci s nástrojem RTW a návod na vytvoření křížových překladačů (cross compilators). Příloha D popisuje zprovoznění modulu Boa5200 pro demonstrační úlohu. Příloha E popisuje demonstrační úlohu shrnující výsledky této práce na praktické ukázce s modulem Boa5200. Soupis použitého

software je v příloze F. Obsah přiloženého CD je pak v příloze G.

Kapitola 2

Základní target

V této kapitole je popsáno vytvoření několika základních souborů, které umožní z modelu v Simulinku vygenerovat C kód, přeložit ho pod emulátorem Linuxu a přes ethernetové rozhraní pomocí programů *ssh* a *scp* nakopírovat na Target a tam ho spustit. Takový proces ukáže získání kódu, který lze spustit na Targetu. Žádná další hodnota přidaná není. Spuštěný kód na Targetu sice poběží, ale jeho užitečnost je malá. Přesto je to funkční základ, který dovolí doimplementování reálného času a vstupně výstupních rozhraní. Těmto vylepšením je věnovaná kapitola 6.

Vytvoření kostry targetu popisuje část 2.1. Ta sice dovoluje generovat C kód, ale při překladu ho překládá nativním překladačem Matlabu. Překlad je ale nutné provádět křížovým překladačem pro Target včetně dopravení přeloženého kódu na Target a jeho spuštění. Proto je v části 2.4 popsáno vytvoření skriptu go pro Linuxový interpret příkazů, který obsahuje příkazy pro samotný překlad a následné kopírování přeloženého kódu na Target včetně jeho spuštění. Aby mohl být proveden překlad, je ještě nutné upravit cesty ve vygenerovaném *makefile* podle části 2.3. Nejen zde se využije tzv. hookfile popsaný v části 2.2. Všechny soubory targetu se kterými tato kapitola pracuje obsahuje přiložené CD v adresáři */linux_grt_target/linux_grt_target*.

2.1 Kostra targetu

Způsob vytvoření kostry targetu je podobný postupu [11], který ze zabývá vytvořením kostry *ERT-based* (Embedded Real Time) targetu. Kostru targetu tvoří dva soubory **.tmf* a **.tlc*. Následuje postup vytvoření kostry *linux-grt-targetu*.

1. Vytvoření nové adresářové struktury (.../linux_grt_target/linux_grt_target) pro vytvářený target a přidání cesty k tomuto adresáři do prostředí Matlab. Je nutné dodržovat konvence ve jménech adresářů a souborů.

2. Nakopírování obecných souborů *grt_unix.tmf* a *grt.tlc* do vytvořeného adrešáře a přejmenování souboru *grt.tlc* na *linux_grt_target.tlc*. Zdrojové soubory lze nalézt v adresáři
matlabroot/rtw/c/grt/.

3. Modifikace souboru *linux_grt_target.tlc* podle následujícího zápisu. Informace se dědí ze souboru *grt.tlc* (GenericRealTime.tlc).

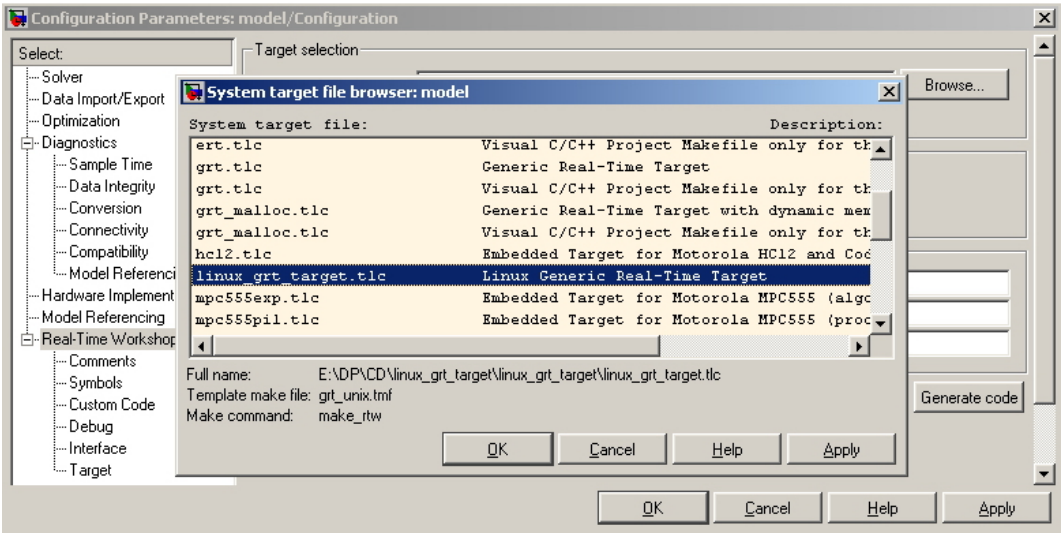
```
%selectfile NULL_FILE

%assign TargetType = "RT"
%assign Language   = "C"
%assign GenRTModel = 1
%assign _GRT_      = 1

%assign MatFileLogging = 1

%include "codegenentry.tlc"

/%
  BEGIN_RTW_OPTIONS
   %----------------------------------------%
  % Configure RTW code generation settings %
  %----------------------------------------%

  rtwgensettings.BuildDirSuffix = 'linux_grt_rtw';
  rtwgensettings.DerivedFrom = 'grt.tlc';
  rtwgensettings.Version = '1';
  END_RTW_OPTIONS
  %/
```

4. Modifikace části souboru *grt_unix.tmf* podle následujícího zápisu.

```
SYS\_TARGET\_FILE = linux\_grt\_target.tlc
MAKEFILE_FILESEP = /
```

5. Dále je nutné vyhledat všechny podmínky *ifeq ((OPTOPTS),(DEFAULT OPT OPTS))* v souboru *grt_unix.tmf* a odstranit je s tím, že se v kódu ponechají jenom části *else* a tím se všechny runtime knihovny budou překládat pro OS GNU Linux.

Po těchto úpravách je nabídka *Simulation: Configuration Parameters: Real-Time Workshop:Target selection* z menu v modelu v Simulinku rozšířena o *linux_grt_target.tlc*, jak je to vidět na obr. 2.1.

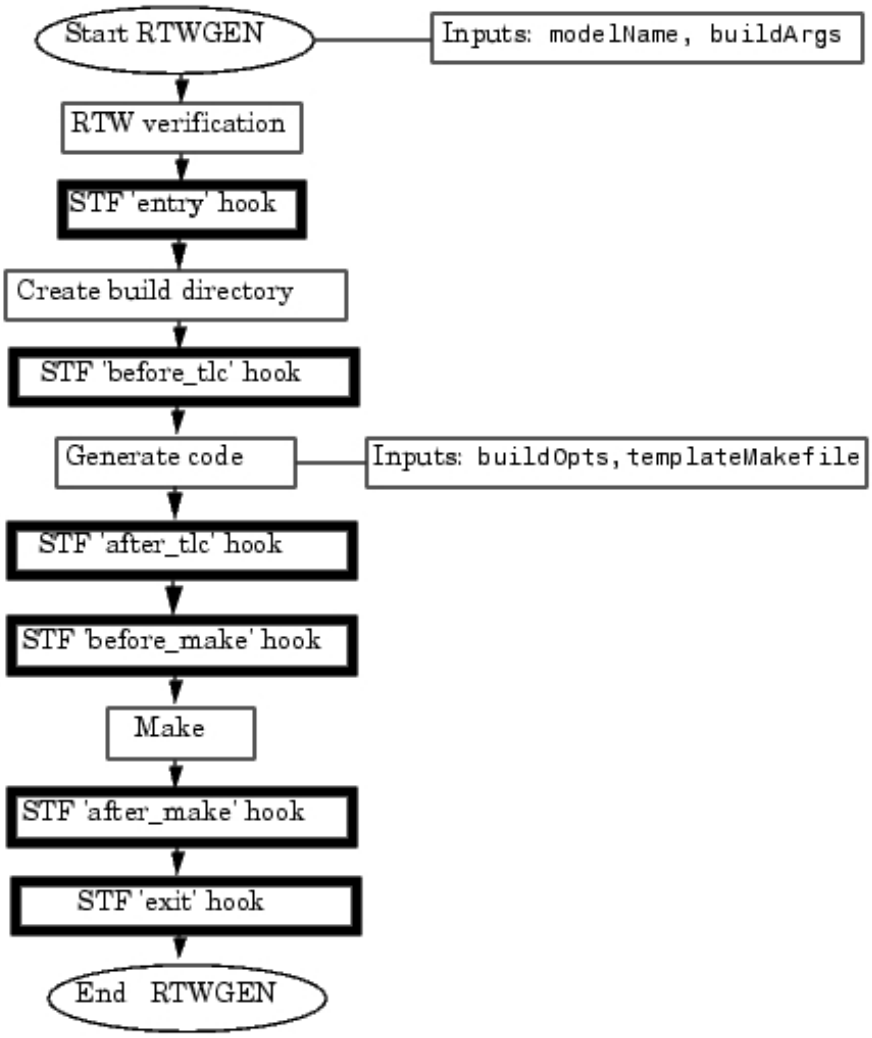


Obrázek 2.1: Výběr targetu pro model v Simulinku

2.2 Make hookfile

Zde je popsaná možnost vstupu do procesu generování C kódu nástrojem RTW v místech, které jsou na obr. 2.2 vyznačeny tučným rámečkem (viz. [10]). V těchto místech lze spouštět vlastní programy. To bude využito např. pro vygenerování skriptu *go* (viz. část 2.4) a změnu cest ve vygenerovaném *makefile* (viz. část 2.3).

Lze využít soubor *ert_make_rtw_hook.m* z adresáře *matlabroot/toolbox/rtw/targets/ ecoder/* a nakopírovat ho do adresáře targetu, kde jsou již základní targetové soubory. Dále je nutné tento soubor přejmenovat na *linux_grt_target_make_rtw_hook.m*, aby byl automaticky volán targetem. Tento soubor v sobě obsahuje přepínač, který umožní zápis vlastního kódu v místech s návěštími schodnými s tučnými rámečky na obr. 2.2.



Obrázek 2.2: Místa pro vlastní program během generování C kódu

2.3 Změna makefile

Cesty ve vytvořeném *makefile* se po procesu generování kódu odkazují na umístění zdrojových souborů pod OS Windows XP. Překlad vygenerovaného kódu se ale bude provádět v emulátorech Linuxu, do kterých budou tato umístění zpřístupněna ("namontována"). To je popsáno v přílohách B a C o emulátorech Linuxu. Cesty ve vygenerovaném *makefile* je nutné změnit z konvence OS Windows (c:\soubor) na konvenci OS GNU Linux (/soubor). Byl vytvořen skript *adapt_code.m*, který v zadaném souboru zaměňuje řetězec č. 1 za řetězec č. 2. Skript je umístěn na přiloženém CD v adresáři */linux_grt_target/linux_grt_target*. Změnu *makefile* provádí volání vytvořeného skriptu *adapt_code* z části *'exit' hook* v souboru *linux_grt_target_make_rtw_hook.m*. Část *'exit' hook* se využívá kvůli dokončení expanze *tokenů* v *makefile* před tímto místem (vyplňování proměnných a cest) během generování C kódu z modelu. Provedou se následující volání:

```
adapt_code([modelName,'.mk'],'C:/','/');
adapt_code([modelName,'.mk'],'c:/','/');
adapt_code([modelName,'.mk'],'C:\','/');
adapt_code([modelName,'.mk'],'c:\','/');
```

Tyto příkazy umožní hledání potřebných souborů při překladu C kódu v emulátorech Linuxu. Pouze tato změna je možná jenom díky shodné adresářové struktuře pod OS Windows XP a "namontovaných" adresářových struktur pod emulátory Linuxu. Jedná se o adresář instalace Matlabu, pracovní adresář a všechny adresáře s uživatelsky vytvořenými kódy, které je vhodné mít umístěné na disku *c:*.

Pokud se využívají soubory i z jiných disků jako např. *d:* (umístění musí být též "namontované" do emulátoru Linuxu), je také nutné provést změnu cest následujícími příkazy. Obdobně by se postupovalo i při využití dalších disků.

```
adapt_code([modelName,'.mk'],'D:/','/');
adapt_code([modelName,'.mk'],'d:/','/');
adapt_code([modelName,'.mk'],'D:\','/');
adapt_code([modelName,'.mk'],'d:\','/');
```

2.4 Skript go

Protože je nutné řídit práci s C kódem po jeho vygenerování (přeložení, nahrání binárního kódu na Target a jeho spuštění), bude popsáno vytvoření skriptu go, který obsahuje příkazy provádějící tyto akce. Skript go je automaticky generován spolu s C kódem z modelu. Jedná se o univerzální skript pro Linuxový příkazový interpreter. Skript *go* převezme veškerou práci nad vygenerovaným C kódem od jeho přeložení křížovým překladačem pod emulátorem Linuxu až po spuštění na Targetu.

Aby bylo možné skript go vygenerovat, je nejprve nutné vytvořit konfigurační složku v okně modelu v Simulinku, která popíše Target a způsob překladu C kódu a jeho spuštění. Toho se dosáhne editací souboru *linux_grt_target.tlc*. Hned za řádkem

```
BEGIN_RTW_OPTIONS
```

se zapíše následující kód:

```
rtwoptions(1).prompt = 'Target';
rtwoptions(1).type = 'Category';
rtwoptions(1).enable = 'on';
rtwoptions(1).default = 6; % Number of items under this category
                             % excluding this one.
rtwoptions(1).popupstrings  = '';
rtwoptions(1).tlcvariable   = '';
rtwoptions(1).tooltip       = '';
rtwoptions(1).callback      = '';
rtwoptions(1).opencallback  = '';
rtwoptions(1).closecallback = '';
rtwoptions(1).makevariable  = '';
rtwoptions(1).callback = '';

rtwoptions(2).prompt = 'Your target-user-name';
rtwoptions(2).type = 'Edit';
rtwoptions(2).default = 'jelinp4';
rtwoptions(2).tlcvariable = 'UN_target';
rtwoptions(2).tooltip = ...
```

```
['Define target-user-name for communication with target hardware.'];
rtwoptions(2).callback = '';

rtwoptions(3).prompt = 'Target IP address';
rtwoptions(3).type = 'Edit';
rtwoptions(3).default = 'rtime.felk.cvut.cz';
rtwoptions(3).tlcvariable = 'IP_target';
rtwoptions(3).tooltip = ...
['Define IP address for communication, upload and execution of the code.'];
rtwoptions(3).callback = '';

rtwoptions(4).prompt = 'Executive options';
rtwoptions(4).type = 'Edit';
rtwoptions(4).default = '-tf inf -w -port 17725';
rtwoptions(4).tlcvariable = 'EX_target';
rtwoptions(4).tooltip = ...
['How to execute the code.'];
rtwoptions(4).callback = '';

rtwoptions(5).prompt = 'Operations (for script go) over generated code';
rtwoptions(5).type = 'Popup';
rtwoptions(5).default = 'compile_copy_execute';
rtwoptions(5).popupstrings = 'compile_copy_execute|compile|compile_copy';
rtwoptions(5).tlcvariable = 'C_target';
rtwoptions(5).callback = '';
rtwoptions(5).tooltip = [''];
```

Tak se ve složce *Simulation: Configuration Parameters: Real-Time Workshop* v Simulinku vytvoří nová záložka *Target*, jak je vidět na obr. 2.3. *Your target-user-name* je jméno účtu na Targetu pod OS GNU Linux, kam se bude nahrávat a spouštět přeložený kód. *Target IP address* je adresa Targetu. *Executive options* nastavuje možné parametry pro spuštění kódu na Targetu pod OS GNU Linux (viz. např. Kapitola 6.2.1).

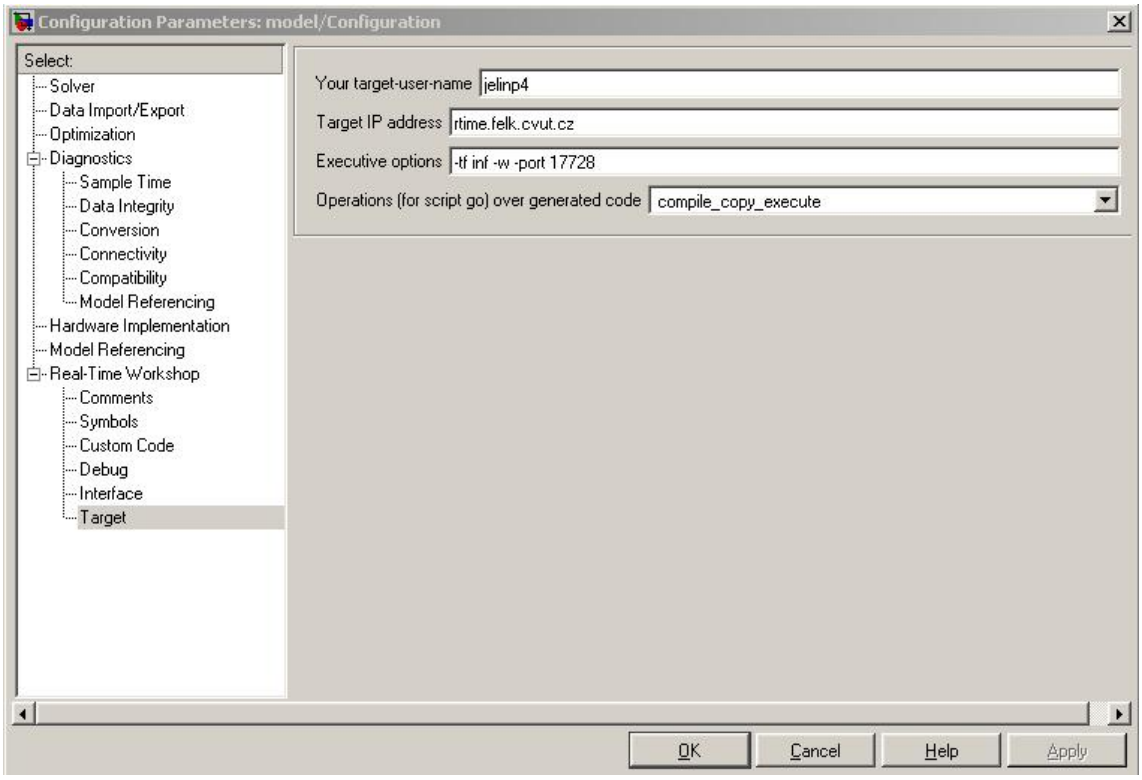
Skript go lze nastavit do tří pracovních režimů:

Compile je režim, kdy skript přeloží vygenerovaný C kód z modelu v Simulinku překladačem, který je zapsán před generováním kódu z modelu v Simulinku jako parametr v editačním poli *Make command* v záložce *Simulation: Configuration Parameters: Real-Time Workshop:Build process* jako *CC=název_překladače* (příklad je na obr. E.4).

Lze tak získat binární spustitelný soubor.

Compile_copy rozšiřuje režim *compile* o nakopírování přeloženého binárního spustitelného souboru přes ethernetové rozhraní na Target. To provede program *scp*.

Compile_copy_execute rozšiřuje režim *compile_copy* o spuštění nakopírovaného binárního souboru na Targetu s parametry *Executive options*. To provede program *ssh*.



Obrázek 2.3: Target options

Dále je nutné vytvořit skript (m-file), který z nastavení ve složce *Simulation: Configuration Parameters: Real-Time Workshop: Target* vytvoří během generování C kódu z modelu skript go. Vytvořený skript je na přiloženém CD v souboru */linux_grt_target/linux_grt_target/linux_grt_target_go.m*.

Tento skript je nutné volat během generování C kódu a musí se mu předat aktuální parametry z konfigurační složky *Target*. K tomu slouží soubor *linux_grt_target_make_rtw_hook.m* popsaný v části 2.2. Do části *'exit' hook* se zapíše následující kód:

```
%It creates script go from target options.
    rtwopt = getActiveConfigSet(gcs);
    UN_target=get_param(rtwopt,'UN_target');
    IP_target=get_param(rtwopt,'IP_target');
    EX_target=get_param(rtwopt,'EX_target');
    C_target=get_param(rtwopt,'C_target');

    linux_grt_target_go(UN_target,IP_target,EX_target,C_target,modelName);
```

Funkce *getActiveConfigSet(gcs)* získává parametry zadefinované v TLC skriptu. Volání *linux_grt_target_go(UN_target,IP_target,EX_target,C_target,modelName)* provede samotné vygenerování skriptu *go*. Ukázka skriptu *go* je v následujícím kódu.

```
#!/bin/sh
cd modellinux_grt_rtw/
./model.bat
cd ..
scp model rtw@192.168.0.100:~
ssh rtw@192.168.0.100 ./model -tf inf -w -port 17725
```

Skript *go*, zcela nahrazuje neoznačenou volbu *Generate code only* z menu v Simulinku ve složce *Simulation: Configuration Parameters: Real-Time Workshop* a s ní svázané tlačítko *Build*. V případě označeného *Generate code only* a stisknutí tlačítka *Generate* se vygeneruje kód z modelu. V opačném případě by se mělo provést nejen generování kódu z modelu, ale i jeho překlad. Pokud je ale v souboru *grt_unix.tmf* nastavena proměnná *HOST* na jiný typ systému, než na kterém je zamýšlen překlad vygenerovaného kódu (náš případ), vypíše Matlab do příkazové řádky *### Make will not be invoked - template makefile is for a different host* a proběhne jenom vygenerování kódu a jeho překlad se neprovede. Další operace pak přebírá právě skript *go*.

Kapitola 3

Generování C kódu

Tato kapitola popisuje nejjednodušší postup pro vygenerování C kódu z modelu v Simulinku. Předpokladem je již vytvořený model v Simulinku. Podrobný postup této operace s množstvím konfiguračních parametrů popisujících optimalizační nástroje, popis použitého hardware, způsob využití paměti atd. lze nalézt v literatuře [19].

Vygenerovaný C kód tvoří množina souborů, která může být různá pro různá nastavení a nelze ji jednoduše popsat. Základními soubory jsou soubory s C kódem **.c* a hlavičkové soubory **.h*, dále makefile **.mk* a skript *go*. Soubory **.c* a **.h* v sobě implementují model, z něhož byly generovány, včetně řízení jeho běhu. Kromě těchto souborů můžou být generovány html zprávy apod. Více v [19].

1. V okně s otevřeným modelem v záložce *Simulation: Configuration Parameters: Real-Time Workshop* v oblasti *Target selection* vybrat z nabídky *linux_grt_target.tlc*. V oblasti *Build proccess* do editačního pole *Make command* připsat argument ***CC=jméno překladače***, kterým bude překlad vygenerovaného kódu pod emulátorem Linuxu prováděn.

2. V okně s otevřeným modelem v záložce *Simulation: Configuration Parameters: Real-Time Workshop:Target* zapsat do editačního pole *Your Target-user-name* jméno uživatelského účtu na Targetu, do *Target IP address* IP adresu Targetu a do *Executive options* parametry pro spuštění na Targetu. Dále zvolit v *Operations (for script go) over generated code* pro jaký druh akce s vygenerovaným C kódem se má vygenerovat skript *go*.

3. V okně s otevřeným modelem v záložce *Simulation: Configuration Parameters: Solver* nastavit *Type* v oblasti *Solver options* na hodnotu *Fixed-step*.

4. Nastavit pracovní adresář Matlabu do míst, kde je uložen model. C kód je totiž generován do pracovního adresáře. Doporučené je umístění v adresáři na disku *c:* (viz. Kapitola 2.3).

5. V okně s otevřeným modelem v záložce *Simulation: Configuration Parameters: Real-Time Workshop* označit volbu *Generate code only* a stisknout tlačítko *Generate code*. Tak se provede vygenerování C kódu z modelu do pracovního adresáře Matlabu.

Kapitola 4

Překlad C kódu

Překlad vygenerovaného C kódu se provádí v emulátorech Linuxu, které májí zpřístupněné ("namontované") adresářové struktury (adresář instalace Matlabu a pracovní adresář Matlabu s modelem) z OS Windows XP. Využívá se skriptu go, který v sobě sdružuje příkazy pro Linuxový interpreter (shell) a je generován spolu s C kódem. Skript go je popsán v kapitole 2.4. Při správném nakonfigurování emulátorů Linuxu včetně zpřístupnění adresářových struktur, jak to popisují přílohy B a C stačí tento skript spustit z místa, kam byl vygenerován (pracovní adresář Matlabu) následujícím příkazem:

```
./go
```

Pro Target **OS GNU Linux s procesorem x86** se využívá emulátor Linuxu **Cygwin** a pro Target **OS GNU Linux s procesorem PowerPC s jádrem 603e** emulátor Linuxu **MinGW**.

Dva emulátory jsou vyžity kvůli různé náročnosti získání křížových překladačů na Targety.

Kapitola 5

Práce s binárním kódem

Využívá se skriptu go, který v sobě sdružuje příkazy pro Linuxový interpreter (shell) a je generován spoulu s C kódem.

Prací s binárním kódem je myšleno:

- Nakopírování binárního kódu programem *scp* z emulátoru Linuxu na Target, který je jednoznačně definován parametry *Your target-user-name* a *Target IP address* popsanými v kapitole 2.4.
- Spuštění binárního kódu pomocí programu ssh s parametry *Executive options*, které je možné zapisovat podle kapitoly 2.4. Samotné parametry spuštění popisuje dokumentace [20].

Pokud jsou požadavky na tyto práce zadány (viz. kapitola 2.4), provedou se bezprostředně po překladu C kódu automaticky.

Kapitola 6

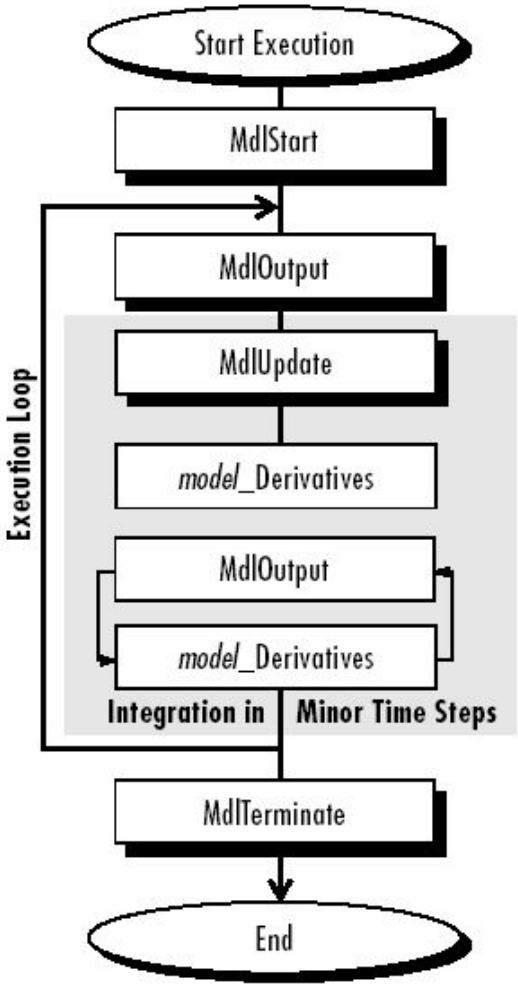
Rozšířený target

6.1 Reálný čas

Protože použité Targety poskytují dostatečně velký výpočetní výkon, je běh v reálném čase implementován v *single tasking* módu. Podrobnější informace o možnostech spouštění jsou popsány v [23].

Pro běh programu v reálném čase ve vytvářeném targetu pro OS GNU Linux je využito softwarové přerušení a na ně navázaná obslužná funkce, tzv. *Interrupt Service Routine* (ISR). OS GNU Linux podle dokumentace [22] poskytuje intervalové časovače *ITIMER_REAL*, *ITIMER_VIRTUAL* a *ITIMER_PROF*, které zasílají s nimi svázané signály. Pro spouštění ISR je využit časovač *ITIMER_REAL*, který zasílá signál *SIGALRM* po vypršení specifikované dávky času bez ohledu na to, v kterém vlákně či procesu se program nachází a to i v případě, že OS obsluhuje jiný program.

Aby vygenerovaný kód z modelu v Simulinku běžel v reálném čase, je nutné periodicky spouštět jeden krok výpočtu kódu modelu, tedy kód ve smyčce podle obr. 6.1. Tuto smyčku reprezentuje vygenerovaná funkce *rt_OneStep(S)*, je tedy nutné periodicky spouštět tuto funkci. Periodu spuštění (*base sampling rate*) lze zjistit funkcí *rtmGetStepSize(S)* a je největším společným dělitelem všech period v modelu.



Obrázek 6.1: Vykonávání kódu modelu

6.1.1 Implementace

Aby bylo možné implementovat běh v reálném čase, je nutné vytvořit vlastní funkci *main*. K tomu lze využít soubor */matlabroot/rtw/c/grt/grt_main.c*. Tento soubor v sobě obsahuje základní šablonu běhu kódu modelu (také funkci *main*) a je tedy vhodné ho využít a rozšířit o běh v reálném čase. Následuje postup vytvoření vlastního souboru s *main* funkcí tak, aby se automaticky generoval z modelu v Simulinku a byl použit během překladu vygenerovaného kódu.

1. Nakopírovat soubor */matlabroot/rtw/c/grt/grt_main.c* do adresáře vytvářeného targetu a přejmenovat na soubor *linux_grt_target_main.c*.

2. Aby se soubor *linux_grt_target_main.c* automaticky generoval z modelu v Simulinku do pracovního adresáře Matlabu spolu s dalšími soubory, je nutné přidat do souboru *linux_grt_target_make_rtw_hook.m* (viz. Kapitola 2.2) za návěští *case 'after_tlc'* následující kód, který provede jeho kopírování:

```
f1=which('linux_grt_target_main.c');
    f2 = pwd;
    copyfile(f1,f2,'f');
```

3. V souboru *grt_unix.tmf* v adresáři vytvářeného targetu změnit řetězec *grt_main.c* za *linux_grt_target_main.c*. Tak se dosáhne toho, že bude soubor *linux_grt_target_main.c* použit během překladu vygenerovaného kódu.

Implementace běhu kódu z modelu v Simulinku v reálném čase zahrnuje editaci souboru *linux_grt_target_main.c* s využitím časovače *ITIMER_REAL*, který zasílá signál *SIGALRM* na který je navázaná obslužná rutina *rt_OneStep_handler*. Nejdůležitější části úpravy jsou popsány v následujícím postupu. Upravený soubor *linux_grt_target_main.c* je obsažen na přiloženém CD v adresáři */linux_grt_target/linux_grt_target*.

1. Zápis obslužné rutiny ISR, která je spouštěna příchodem signálu *SIGALRM*. Proměnná drive řídí spouštění funkce *rt_OneStep(S)* v hlavní smyčce programu.

```
void rt_OneStep_handler(int sig){
drive=drive+1;
}
```

2. Následující kód nastavuje ISR pro signál *SIGALRM*. Dále se nastavuje časovač *ITIMER_REAL* podle *base sampling rate* a spouští se. Nakonec se odblokuje signál *SIGALRM*, aby se po jeho příchodu automaticky spouštěla ISR.

```
/* Install timer_handler as the signal handler for SIGALRM. */
memset (&sa, 0, sizeof (sa));
sa.sa_handler = &rt_OneStep_handler;
sigaction (SIGALRM, &sa, &oldsa);
   /* Configure the timer to expire after ... */
timer.it_value.tv_sec = (int)rtmGetStepSize(S);
```

```
timer.it_value.tv_usec =(int)((rtmGetStepSize(S)-timer.it_value.tv_sec)\
*1000000);
   /* ... and every ... after that. */
timer.it_interval.tv_sec =(int)rtmGetStepSize(S);
timer.it_interval.tv_usec =(int)((rtmGetStepSize(S)-timer.it_value.tv_sec)\
*1000000);
   /* Start a virtual timer. It counts down whenever this process is
      executing. */
setitimer (ITIMER_REAL, &timer, NULL);

/* Unblock signal SIGALRM for use by handler */
sigemptyset(&mask);
sigaddset(&mask, SIGALRM);
sigprocmask(SIG_UNBLOCK,&mask,&oldmask);
```

3. Následuje kód hlavní smyčky, která v případě podmínky *drive>1* spouští krok výpočtu kódu modelu v podobě funkce *rt_OneStep(S)*. Podmínky *drive>1* způsobí ukončení programu s chybou *ISR overrun - base sampling rate is too fast*. To znamená, že se krok výpočtu kódu modelu v podobě funkce *rt_OneStep(S)* nestihnul vypočítat během základní periody (*base sampling rate*). Tzn., že přišel od časovače *ITIMER_REAL* další signál *SIGALRM*, aniž by byl krok výpočtu kódu modelu v podobě funkce *rt_OneStep(S)* dokončen a mohl tak být znovu spuštěn.

```
while (!GBLbuf.stopExecutionFlag &&
           (rtmGetTFinal(S) == RUN_FOREVER ||
            rtmGetTFinal(S)-rtmGetT(S) > rtmGetT(S)*DBL_EPSILON)) {

if(drive==1){

rt_OneStep(S);
if(drive>1){
//  Disable ITIMER_REAL
timer.it_value.tv_sec = 0;
timer.it_value.tv_usec =0;
```

```
timer.it_interval.tv_sec =0;
timer.it_interval.tv_usec =0;

setitimer (ITIMER_REAL, &timer, NULL);
GBLbuf.isrOverrun=1;
GBLbuf.stopExecutionFlag = 1;
break;
};
drive=drive-1;
};

if(drive>1){
//  Disable timer.
timer.it_value.tv_sec = 0;
timer.it_value.tv_usec =0;

timer.it_interval.tv_sec =0;
timer.it_interval.tv_usec =0;

setitimer (ITIMER_REAL, &timer, NULL);
GBLbuf.isrOverrun=1;
GBLbuf.stopExecutionFlag = 1;
break;
};
    if (rtmGetStopRequested(S)) break;
}
```

4. Po regulerním ukončení hlavní smyčky programu je nutné ukončit časovač. To provede následující kód:

```
//  Disable timer.
timer.it_value.tv_sec = 0;
timer.it_value.tv_usec =0;
```

```
timer.it_interval.tv_sec =0;
timer.it_interval.tv_usec =0;

setitimer (ITIMER_REAL, &timer, NULL);
```

6.1.2 Ladící prostředky

Pro účel zjišťování skutečného chování programu v RT běhu je možné využít následující kód, který vypíše na standardní výstup aktuální časovou značku. Takový kód je potom možné umístit na začátku kroku výpočtu kódu modelu a na jeho konci, čímž lze zjistit nejenom délku trvání prováděného kódu, ale i čas, v který byl kód vyvolán.

Toto uvádím proto, že generování signálů od časovačů v OS GNU Linux není tak přesné, jak požadujeme. Nepoužívá se totiž RT Linux. Jako příklad můžu uvést požadované generování signálu v OS GNU Linux s periodou 1 ms a skutečnou periodou doručování 4 ms.

```
// - JUST FOR DEBUGGING USE -
/* Obtain the time of day, and convert it to a tm struct. */
gettimeofday (&tv, NULL);
ptm = localtime (&tv.tv_sec);
/* Format the date and time, down to a single second. */
strftime (time_string, sizeof (time_string), "%Y-%m-%d %H:%M:%S", ptm);
/* Compute milliseconds from microseconds. */
milliseconds = tv.tv_usec / 1000;
/* Print the formatted time, in seconds, followed by a decimal point
and the milliseconds. */
printf ("stop step: %s.%06ld\n", time_string, tv.tv_usec);
fflush(stdout);
```

6.1.3 Shrnutí

Výše popsanými postupy bylo dosaženo běhu kódu vygenerovaného z modelu v Simulinku v reálném čase v *single tasking* módu. Protože jsou využity Targety s velkým výpočetním výkonem, není zatím nutné provádět implementaci běhu v reálném čase v *multi tasking* módu. Vzhledem k tomu, že OS GNU Linux není RT Linux (Real Time

Linux) se stává, že při využití kratších period než 4 ms dochází k nesprávnému chování programu zapřičiněnému vlastnostmi jádra OS GNU Linux. Nejzřetelnější chybou je pak neukončení programu při časové náročnosti výpočtu jednoho kroku modelu podle obr. 6.1 (funkce *rt_OneStep(S)*) větší, než povoluje zvolená vzorkovací perioda. V případě standardního chování tuto chybu oznamuje výpis *ISR overrun - base sampling rate is too fast* na standardním výstupu OS GNU Linux. To je pravděpodobně způsobeno tím, že ISR tvoří nepřerušitelnou část programu a po dokončení ISR již čeká na obsluhu další signál a ISR je tedy znovu spuštěna, aniž by proběhl další kód programu, který by vohodnotil chybu *ISR overrun - base sampling rate is too fast*.

Doporučuji tedy využívání nejmenší vzorkovcí periody 4 ms.

6.2 External mode

Externí rozhraní (External mode) je způsob, jak komunikovat z prostředí modelu v Simulinku se spuštěným kódem reprezentujícím tento aktuální model ze Simulinku na Targetu. Používá se pro

- Spuštění (jedná se o spuštění vykonávání kódu, program již musí být spuštěn s patřičnými parametry, viz. dále) a zastavení kódu na Targetu.
- Vizualizaci veličin, které jsou počítané běžícím kódem na Targetu.
- Parametrizaci běžícího kódu (modelu) na Targetu.

a to vše přímo z prostředí modelu v Simulinku.

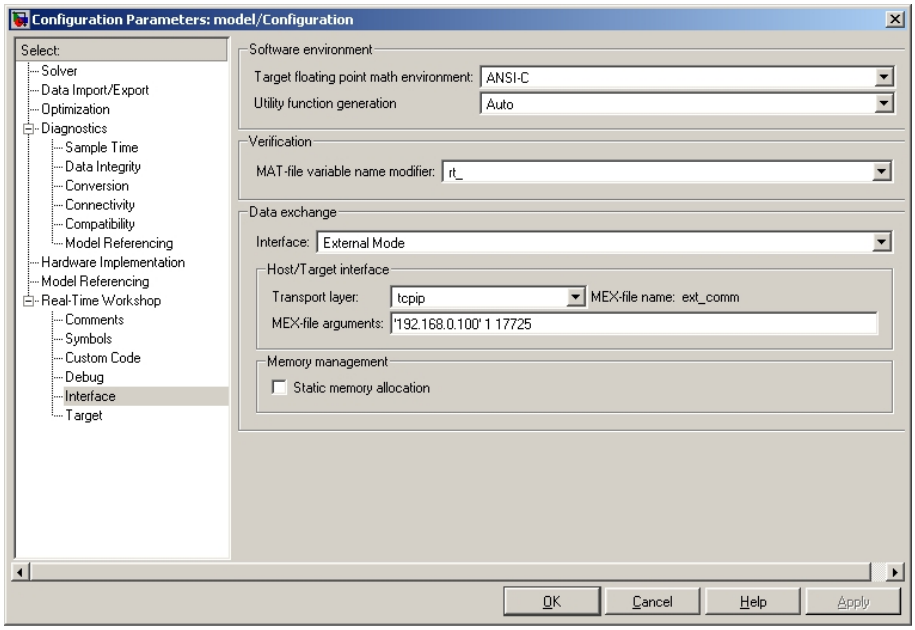
Komunikace je typu *client-server*, kde pojem *host* je využit pro Simulink a *target* pro zařízení, kde běží kód vygenerovaný z modelu v Simulinku. Využívá se protokolu TCP/IP nebo asynchronní sériové linky RS232. Pro účely této práce je využitý protokol TCP/IP. Stejný protokol je totiž využíván pro kopírování přeloženého kódu na Target a jeho spouštění (viz. Kapitola 5).

Díky tomu, že jako Target je používán OS GNU Linux s 32 bitovými procesory a External mode s protokolem TCP/IP je navržen pro 32 bitová prostředí a využívá soketovou komunikaci, není nutný žádný zásah do zdrojových souborů, které lze nalézt v adresáři */matlabroot/rtw/ext_mode*.

Podrobné informace lze nalézt v dokumentaci [24].

6.2.1 Základní znalosti pro použití

Před generováním kódu z modelu v Simulinku je nutné provést nastavení např. podle obr. 6.2 z menu *Simulation: Configuration Parameters: Real-Time Workshop:Interface*. Důležitý je rámeček *Data exchange*. Parametr *'192.168.0.100'* je IP adresa Targetu, *1* znamená zapnuté hlášení o průběhu komunikce a *17725* je port pro komunikaci. Tak je prostředí Simulink připraveno na spojení *External mode*.



Obrázek 6.2: Konfigurace External mode v Simulinku

Binární kód na Targetu pod OS GNU Linux lze spustit např. příkazem *./model -tf inf -w -port 17725* (to je prováděno automaticky skriptem *go*), kde použité parametry znamenají:

- *-tf inf* znamená, že kód nemá určenou dobu běhu (stoptime, inf=nekonečno)
- *-w* znamená čekej na příkaz z rohraní External mode ke spuštění vykonávání kódu
- *-port 17725* je port pro soketovou komunikaci

Samotné připojení k spuštěnému programu na Targetu lze provést z menu *Tool: External Mode Control Panel* v prostředí modelu v Simulinku. Další informace jsou v literatuře [24].

6.2.2 Chybové hlášení

Je možné, že se při práci s *External mode* objeví chybová zpráva *Checksum mismatch. Target code needs to be rebuilt.* Taková chybová zpráva ukazuje na to, že vygenerovaný kód z modelu na Targetu a model v Simulinku nejsou totožné. Tohoto problému je možné se zbavit znovuvygenerováním kódu z modelu, jeho přeložením, nahráním na Target a jeho spuštěním.

6.3 Vstupně výstupní rozhraní

Jedná se o vytváření bloků pro modelování v prostředí Simulink, které reprezentují komunikační rozhraní na Targetu. Tyto bloky jsou označovány jako **Block set** a jsou reprezentovány S-funkcemi, které v sobě sdružují C kódy a skripty. Bloky se ukládají pod vytvořenou knihovnou a jsou přímo přístupné z prostředí Simulink.

Existují tři druhy S-funkcí, zde je využita tzv. *Wrapper S-function*, která volá naprogramované funkce z dalšího C souboru, a ty reprezentují operace jako je otevření a zavření komunikačního rozhraní, čtění a zápis na komunikační rozhraní apod. Před započetím práce doporučuji prostudování literatury [25].

V části 6.3.1 je popsán princip jednoho ze způsobů vytvoření *Block setu*. Tento způsob je aplikován v části 6.3.2, která popisuje důležité kroky vedoucí k vytvoření vstupně výstupního bloku asynchronní sériové linky RS232.

6.3.1 Možný způsob vytvoření *Block setu*

Použije se kostra S-funkce ze souboru *matlabroot/simulink/src/sfuntmpl_basic.c*, která obsahuje funkce odpovídající jendotlivým místům na obr. 6.1, který reprezentuje způsob běhu kódu z modelu v Simulinku a jednotlivé vstupní body pro spouštění vlastního kódu z S-funkce. S-funkce se přejmenuje tak, aby reprezentovala vytvářené rozhraní. Vytvoří se nový soubor s funkcemi, které provádějí operace s požadovaným rozhraním (otevření,

zavření, zápis, čtení apod.). Tyto funkce jsou pak volány z vytvořené S-funkce v návěštích podle již zmíněného obr. 6.1.

Vytvoří se knihovna, do níž se vloží nový blok *Simulink-ser-Defined Functions-S-function* a jako parametr se zadá vytvořená S-funkce, její parametry a soubor s naprogramovanými funkcemi pracujícími s rozhraním, jak je to např. vidět na obr. 6.4.

Vytvoří se soubor *slblocks.m* ve kterém se vytvoří popis vytvořených bloků a který je nutný pro zpřístupnění vytvořeného bloku z okna *Simulink Library Browser*.

Umístění zdrojových souborů musí být nastavené v cestě Matlabu. Podrobný postup lze nalézt v literatuře [25].

6.3.2 Vytvoření RS232 Block Set

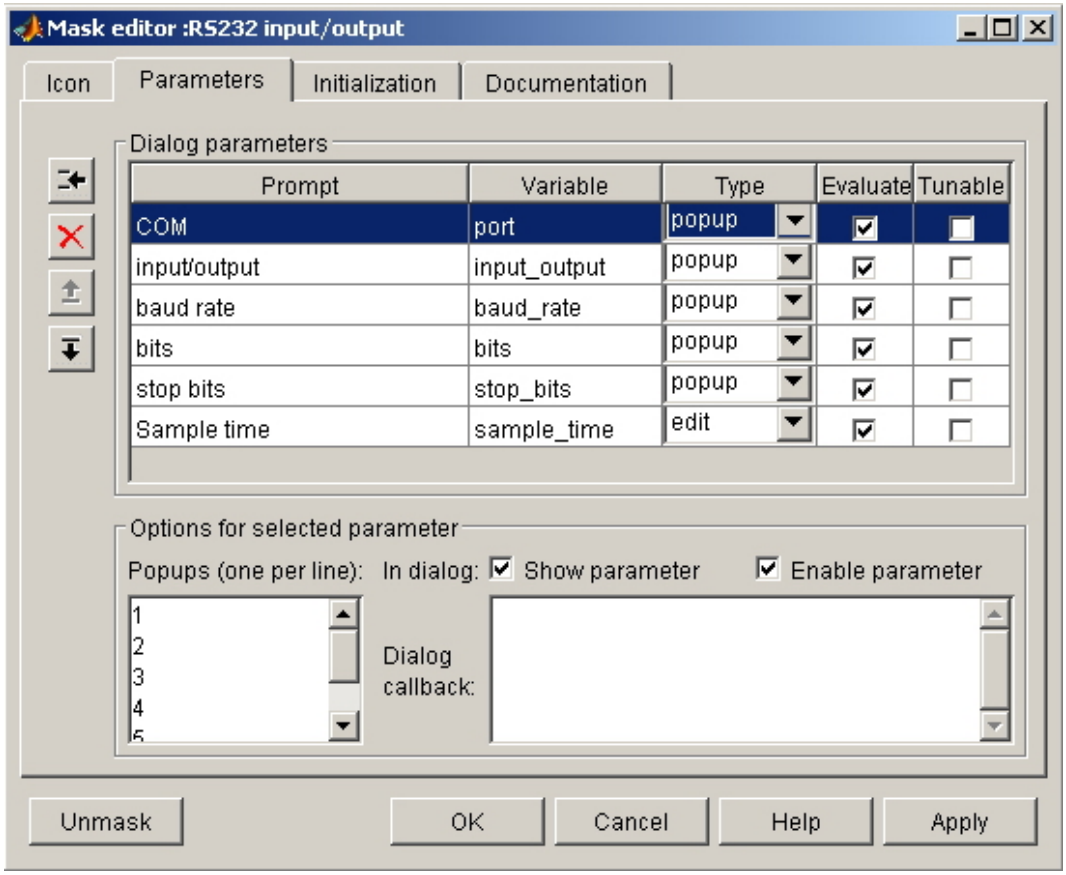
Následující postup shrnuje vytvoření vstupně výstupního bloku asynchronní sériové linky RS232 a jeho zpřístupnění z okna *Simulink Library Browser*. Výsledné soubory lze nalézt na přiloženém CD v adresáři */linux_grt_target/linux_grt_target/RS232*.

1. V adresářové struktuře vytvářeného targetu se vytvoří adresář RS232 a do něj se nakopíruje soubor *matlabroot/simulink/src/sfuntmpl_basic.c* reprezentující kostru obecné S-funkce. Soubor se přejmenuje na *sfungrtlinux_rs232.c*.

2. Ve stejném umístění se vytvoří soubor *sfungrtlinux_rs232_wrapper.c* obsahující funkce pro práci se sériovou linkou (otevření, zavření, zápis a čtění).

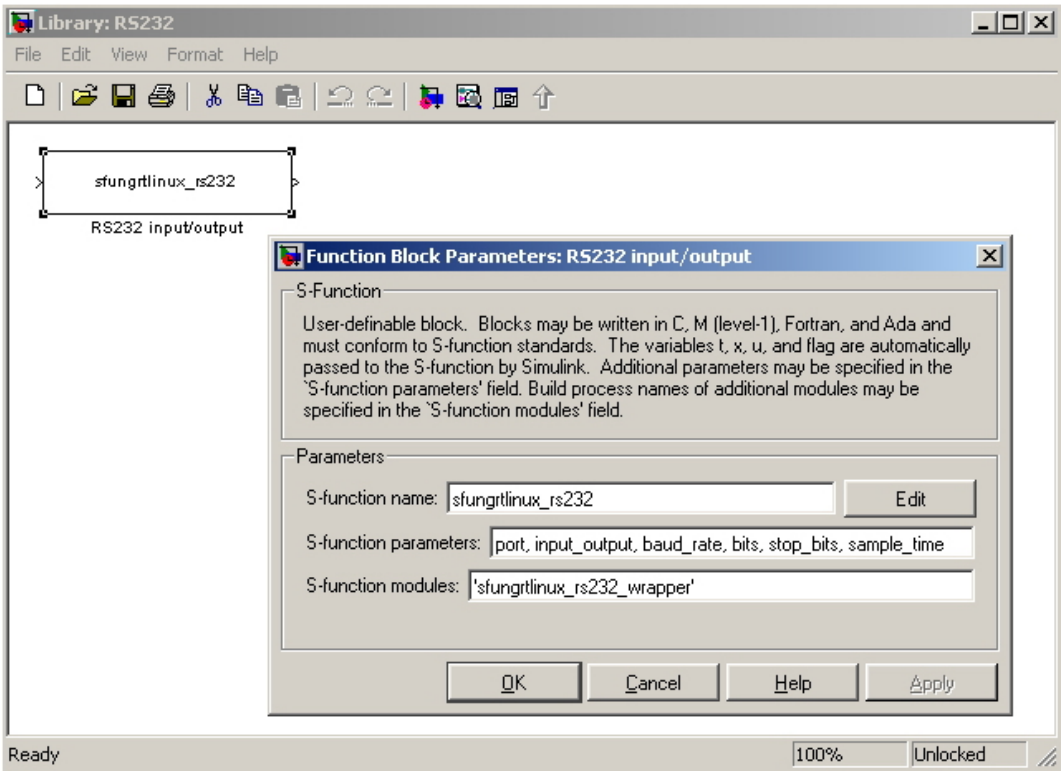
3. V okně *Simulink Library Browser* se vytvoří nová knihovna volbou *File:New:Library* z menu a uloží se k již vytvořeným souborům jako *RS232.mdl*.

4. Do vytvořené knihovny se vloží blok *Simulink-user-Defined Functions-S-function* z *Simulink Library Browser*. Kliknutím pravým tlačítkem myši na vložený blok a volbou *Edit mask* se ve spuštěném Mask editoru zadají proměnné popisující nastavení sériového portu, jak je vidět na obr. 6.3.



Obrázek 6.3: Editor masky S-funkce pro rozhraní RS232

5. Kliknutím pravým tlačítkem myši na vložený blok a volbou *Look under mask* se zapíše S-funkce reprezentující vytvořený blok, její parametry a využité moduly, jak je to vidět na obr. 6.4. Po ukončení práce s knihovnou je knihovna automaticky uzamčena a je nutné ji při následujících úpravách odemknout příkazem z hlavního menu ***Edit-Unlock Library***.



Obrázek 6.4: Maska S-funkce pro rozhraní RS232

6. V S-funkci (soubor *sfungrtlinux_rs232.c*) je nutné zadat vzorkovací periody vstupního a výstupního portu bloku reprezentovaného S-funkcí. To se provede následujícím kódem z funkce *mdlInitializeSampleTimes*. Nastavení musí předcházet inicializace vzorkování pro vstupně výstupní port ve funkci *mdlInitializeSizes*.

```
ssSetSampleTime(S, 0,  CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);

ssSetSampleTime(S, 1,  mxGetScalar(SAMPLE_TIME(S)));
    ssSetOffsetTime(S, 1, 0.0);
```

7. V S-funkci (soubor *sfungrtlinux_rs232.c*) se konfigurační parametry z vytvořeného bloku vyčítají pomocí následujících maker.

```
#define PORT(S) ssGetSFcnParam(S, 0)
#define INPUT_OUTPUT(S) ssGetSFcnParam(S, 1)
```

```
#define BAUD_RATE(S) ssGetSFcnParam(S, 2)
#define BITS(S) ssGetSFcnParam(S, 3)
#define STOP_BITS(S) ssGetSFcnParam(S, 4)
#define SAMPLE_TIME(S) ssGetSFcnParam(S, 5)
```

8. V S-funkci (soubor *sfungrtlinux_rs232.c*) se zapíší volání kódu pro práci se sériovou linkou.
   - Ve funkci *mdlStart* otevření sériového portu.
   - Ve funkci *mdlOutputs* čtění a zápis ze/na sériový port podle toho, zda je blok nastaven jako vstupní nebo výstupní.
   - Ve funkci *mdlTerminate* uzavření sériového portu.
9. Vytvoří se soubor *slblocks.m*, který jednoznačně definuje vytvořenou knihovnu a umožní tak její viditelnost z okna *Simulink Library Browser*.
10. Adresář RS232 se přidá do cest Matlabu.

6.3.2.1 Rozlišení kódu pro simulaci v Simulinku a na Targetu

Aby se rozlišil C kód zdrojových souborů pro simulaci v Simulinku a simulaci na Targetu, je využit podmíněný překlad, jak je to ukázáno v následujícím kódu. Další možný způsob řešení (zde nevyužitý) spočívá ve vytvoření TLC souboru. V případě simulace v Simulinku je provedeno pouze přeposlání dat ze vstupního portu bloku na výstupní port bloku podle zvolené periody.

```
#if defined(MATLAB_MEX_FILE)
//Kód pro simulaci v Simulinku.
#else
  //Kód pro simulaci na Targetu.
#endif
```

6.3.2.2 Překlad pro simulaci v Simulinku

Pro využití vytvořeného RS232 Block Setu pro simulace v Simulinku pod OS Windows XP, je nutné zdrojové soubory přeložit do knihovny (dll) následujícími příkazi v příkazové řádce Matlabu (aktuální pracovní adresář musí být v umístění překládaných souborů):

```
mex sfungrtlinux_rs232_output.c sfungrtlinux_rs232_wrapper.c
mex sfungrtlinux_rs232_input.c sfungrtlinux_rs232_wrapper.c
```

6.3.2.3 Převod double-byte-double

Následující kód ukazuje, jak lze datový typ *double* převést na *pole bytů* a poté ho zpět složit na typ *double*. To umožňuje datový typ double přenést přes sériovou linku.

```
#include <stdio.h> /* Standard input/output definitions */
#include <string.h> /* String function definitions */

typedef union{
 double d;
 char buf[sizeof(double)];
}DblBuf;

int main(){
 DblBuf d;
 DblBuf d1;
 int i=0;

 d.d=12345.4789123456;

 for(i=0;i<sizeof(double);i++){
  d1.buf[i]=d.buf[i];
 }
 printf("Double ma %u bytu.\n",sizeof(double));
 printf("Double pred konverzi je: %f \n",d.d);
 printf("Double po konverzi je: %f \n",d1);
 return 0;
}
```

Kapitola 7

Závěr

Byl vytvořen *linux_grt_target* umožňující generovat C kód z modelu v Simulinku pro OS GNU Linux a obecný procesor. Tento generovaný kód byl upraven, aby jeho vykonávání probíhalo v reálném čase. Množina bloků z *Simulink Library Browser* pro modelování v Simulinku byla rozšířena o vstupně výstupní blok asynchronní sériové linky RS232 pro OS GNU Linux. Byla ověřena možnost komunikace External mode mezi Simulinkem a Targetem přes ethernetové rozhraní, která je významná kvůli vzdálenému řízení a manipulaci z prostředí Simulinku s běžícím kódem na Targetu.

Dále byl zprovozněn křížový překladač *cygwin32-x86-linux-gnu* a vytvořen křížový překladač *mingw32-powerpc-603e-linux-gnu* z OS Windows (procesor x86) na Targety (OS GNU Linux a procesory *x86* a *PowerPC s jádrem 603e*). Každý z těchto křížových překladačů využívá jiný emulátor Linuxu pod OS Windows XP a tím tedy vzrůstá možnost budoucího získání a využití dalších křížových překladačů na další procesory.

Byl vytvořen způsob, jak automaticky přeložit vygenerovaný kód z modelu v Simulinku pod emulátory Linuxu, přeložený binární kód nahrát na Targety a tam ho spustit. Tyto operace řídí automaticky generovaný skript *go* pro Linuxový interpreter příkazů.

OS GNU Linux pro Modul Boa5200 s procesorem PowerPC byl upraven tak, aby mohl být využit s vytvořeným *linux_grt_targetem*. Poté byl využit pro vytvoření demonstrační úlohy.

Tak bylo dosaženo toho, že s použitím výsledků této diplomové práce lze zvolit hardware s OS GNU Linux a procesorem x86 nebo PowerPC s jádrem 603e a zapojit ho do simulační smyčky pomocí rozhraní RS232. V prostředí Simulinku pak lze vytvořit model z kterého se automaticky získá odpovídající kód a přes ehternetové rozhraní se spustí na již zmíněném hardware a pomocí rozhraní External mode ho lze ze Simulinku vizualizovat a parametrizovat. Lze tvrdit, že každou změnu v modelu v Simulinku tak lze velmi

jednoduše promítnout do programu na hardware.

Další vývoj targetu *linux_grt_target* by měl především pokračovat v rozšíření vstupně výstupního *Block setu* pro komunikační rozhraní jako je CAN a Ethernet. Možné vylepšení targetu by přinesla implementace běhu kódu v reálném čase v módu multitasking.

Literatura

[1] Jelínek, Pavel. Simulace Processor in The Loop a Hardware In the Loop, Automa: *Časopis pro automatizační techniku*, 2007, roč. 7, č. 5., s. 34-35.

[2] The MathWorks, Inc. [online]. poslední revize 2007-11-04 [cit. 2007-11-04]. Dostupné na World Wide Web:
<http://www.mathworks.com>.

[3] RAPAVÝ, Martin. *Processor in The Loop simulace: Diplomová práce.* Praha: ČVUT, Fakulta elektrotechnická, Katedra řídící techniky, 2007. 125 l., 41 l. příl. Vedoucí diplomové práce Libor Waszniowsky

[4] Real-Time Workshop® User's Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf>.

[5] Boa5200 MPC5200 Module [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://www.ultsol.com/pdfs/Boa5200v1.1.pdf>.

[6] HOWTO [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://rtime.felk.cvut.cz/hw/index.php/HOWTO>.

[7] MPC5200_root [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<ftp://rtime.felk.cvut.cz/MPC5200/MPC5200_root_070321.tar.gz>.

[8] Dropbear [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://matt.ucc.asn.au/dropbear/dropbear.html>.

[9] RedBoot User´s Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://ecos.sourceware.org/docs-latest/redboot/redboot-guide.html>.

[10] Customizing the Target Build Process With the STF_make_rtw Hook File [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://www.kxcad.net/cae_MATLAB/toolbox/ecoder/ug/f10435.html>.

[11] Tutorial: Creating a Custom Target Configuration [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://www.kxcad.net/cae_MATLAB/toolbox/ecoder/det/f6613.html>.

[12] Cross-compiling on Windows for Linux [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://metamod-p.sourceforge.net/cross-compiling.on.windows.for.linux.html>.

[13] Tool chain [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://rtime.felk.cvut.cz/hw/index.php/HOWTO#Tool_chain>.

[14] Cross-Hosted MinGW Build Tool [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=12644>.

[15] MinGW [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://www.mingw.org>.

[16] Cygwin [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<http://www.cygwin.com>.

[17] Binutils [online]. v. 2.18, poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web:
<ftp://ftp.gnu.org/gnu/binutils/File:binutils-2.18.tar.gz>.

[18] Gcc [online]. v. 4.1.1, poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <ftp://ftp.gnu.org/gnu/gcc/gcc-4.1.1/gcc-4.1.1.tar.bz2>.

[19] Code Generation and the Build Process, Real-Time Workshop® User's Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf>.

[20] Downloading and Running the Executable Interactively, Real-Time Workshop® User's Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf>.

[21] Build a GCC-based cross compiler for Linux [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://sources.redhat.com/ml/crossgcc/2005-08/msg00114/l-cross-ltr.pdf>.

[22] Linux System Calls, Advanced Linux Programming [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://www.advancedlinuxprogramming.com/alp-folder>.

[23] Model Execution, Real-Time Workshop® User's Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf>.

[24] External Mode, Real-Time Workshop® User's Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf>.

[25] Writing S-Functions for Real-Time Workshop, Real-Time Workshop® User's Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf>.

[26] Writing Wrapper S-Functions, Real-Time Workshop® User's Guide [online]. poslední revize 2008-1-03 [cit. 2008-1-03]. Dostupné na World Wide Web: <http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf>.

Příloha A

Seznam zkratek

Tabulka A.1: Tabulka zkratek

HIL	Hardware In the Loop
HW	HardWare
IO	Input Output
ISR	Interrupt Service Routine
JFFS2	Journale File System version 2
MinGW	Minimalist GNU for Windows
MSYS	Minimal System
OS	Operating System
PIL	Processor In the Loop
RetBoot	Red Hat Embedded Debug and Bootstrap
RT	Real Time
RTW	Real Time Workshop
SCP	SeCure Copy
SSH	Secure SHell
STF	System Target File (TLC)
TFTP	Trivial File Transfer Protocol
TLC	Target Language Compiler

Příloha B

Cygwin

Cygwin je emulátor Linuxu pro prostředí Windows. Skládá se z dll souboru, který emuluje Linux API vrstvu a kolekce Linuxových programů. Dovoluje tedy i využití GNU Toolchain pro překlad zdrojových kódů. Více v [16].

B.1 Instalace

Instalaci Cygwinu lze provést on-line přes odkaz [16]. To umožní nainstalování vybrané kolekce Linuxových programů z nabídnuté databáze prostředků. Pro potřebu této diplomové práce doporučuji nainstalovat všechny nabídnuté kategorie jako *Default*. Kategorii *Devel* jako *Install*, aby se nainstalovaly všechny prostředky v ní obsažené. Dále pak dohledat balíček *openssh* a označit požadavek na jeho instalaci.

B.2 Konfigurace prostředí

Kvůli překladu vygenerovaného C kódu z modelu v Simulinku je nutné zpřístupnit pracovní adresář Matlabu s modelem (work) a kořenový adresář s instalací Matlabu (matlabroot) do kořenového adresáře Cygwinu. To se provede příkazem *mount*. Jednou "namontované" adresáře zůstanou platné i po vypnutí a znovu zapnutí Cygwinu. Následuje sekvence příkazů provádějící popsanou akci.

```
cd /
mkdir /matlabroot
mount c:/matlabroot   /matlabroot

mkdir /work
mount c:/work   /work
```

´

B.3 Cross compiler pro procesory x86 a OS Linux

Křížový překladač pro OS GNU Linux s procesorem x86 z Cygwinu s procesorem x86 je již vytvořený a lze ho nalézt pod odkazem [12]. Jedná se o balíček *cygwin-gcc-3.3.6-glibc-2.3.2-linux.tar.bz2* obsahující již zkompilovaný křížový překladač pro Cygwin. Balíček je obsažen na přiloženém CD v adresáři */cross-compilers/cygwin32-x86-linux-gnu*. Tento balíček se nakopíruje do kořenové struktury Cygwinu a rozbalí se. Vytvoří se adresář */opt/crosstool*, který obsahuje potřebné nástroje pro přeložení C kódu pro Target OS GNU Linux s procesorem x86 (32bit. a 64 bit. verze). V adresáři */bin* pak lze nalézt dva linkové soubory ***gcc-linux*** a *gcc-linux-x86_64* odkazující se na překladače.

Příloha C

Mingw a Msys

MinGW (Minimalist GNU for Windows) a MSYS (Minimal SYStem) tvoří dohromady emulátor Linuxu, který umožňuje pod OS Windows pkřekládat zdrojové kódy s využitím GNU toolchain. Podrobné informace v [15].

C.1 Instalace

Zdrojové soubory lze dohledat v [15]. Pro potřeby této diplomové práce doporučji nainstalovat *MinGW-5.1.3.exe* a *MSYS-1.0.10.exe*. Dále pak pro využití *ssh* komunikce rozbalit do kořenového adresáře MSYS soubory *minires-1.01-1-MSYS-1.0.11.tar.bz2*, *openssh-4.6p1-MSYS-1.0.11.tar.bz2*, *openssl-0.9.8e-3-MSYS-1.0.11.tar.bz2*
a *zlib-1.2.3-MSYS-1.0.11.tar.bz2*.

C.2 Úprava prostředí

Po instalaci Msys bude pravděpodobně pracovní konzole pracovat v grafickém režimu. Doporučuji tento režim vypnout přepsáním souboru *rxvt.exe* z adresáře */bin* na soubor *rxvt-off.exe*. To způsobí práci v textovém režimu.

Kvůli překladu vygenerovaného C kódu z modelu v Simulinku je nutné zpřístupnit pracovní adresář Matlabu s modelem (work) a kořenový adresář s instalací Matlabu (matlabroot) do kořenového adresáře Msys. To se provede vytvořením zmíněných adresářů příkazy:

```
cd /
mkdir /matlabroot
mkdir /work
```

a přidáním řádek

```
c:/MATLAB /matlab
c:/work /work
```

do souboru */etc/fstab*. Po ukončení prostředí a jeho dalším spuštění jsou adresáře přístupné.

C.3 Cross compiler pro procesory PowerPC s jádrem 603e a OS Linux

Kvůli zjednodušení následujících zápisů se budou využívat termíny Host, Build, a Target podle následujícího významu. Před započetím práce doporučuji přečtení dokumentace [21].

Definice platformy build, host a target:
Build: OS GNU Linux, x86
Host: Mingw (OS Windows XP), x86
Target: OS GNU Linux, PowerPC s jádrem 603e

Pro potřebu využití vygenerovaného C kódu ze Simulinku na Targetu je nutné vytvořit překladač tohoto kódu. Překladač pro překlad z Build na Target byl již na Katedře řídící techniky vytvořen (viz. [13]), ale pro potřeby této práce je zapotřebí vytvořit překladač z Host na Target . To ale neznamená, že část již vytvořeného překladače nelze využít. Tento problém se nazývá *Canadian cross compilation*.

Následuje postup, kterým je možné vytvořit překladač z Host na Target. Kroky 1-6 se provádějí na platformě Build a kroky 7-9 na platformě Host:

1. Nejprve je nutné připravit nástroje, které budou překládat z *Build* na *Host*, aby bylo možné zkompilovat překladač (na *Build*), který poběží na *Host* a bude překládat pro

Target. K tomu se využijí skripty *Cross-Hosted MinGW Build Tool*, které lze nalézt pod odkazem [14] nebo na přiloženém CD v adresáři */cross-compilers/x86-mingw32-build-sh*. Tento balík skriptů obsahuje v textovém režimu kompletního průvodce pro vytvoření *Cross compiler* z Build na Host, který se spustí následujícím příkazem:

```
./x86-mingw32-build.sh target=mingw32
```

2. Již vytvořený překladač [13] z *Build* na *Target* nám umožní využít již obsažené přeložené soubory z balíku glibc. Následující příkaz rozbalí překladač na *Build*.

```
tar -xvf gcc-powerpc-603e-linux-gnu-4.1.1-bin.tar.gz
```

3. Dále je nutné pro platformu Host přeložit nejnovější verzi Binutils, kterou lze najít pod odkazem [17]. Aktuálně je dostupná verze *binutils-2.18*. Následující příkazy rozbalí Binutils do aktuálního adresáře, vytvoří adresář *binutils-build* do kterého se vstoupí, příslušná verze binutils se nakonfiguruje a přeloží. Konfigurační soubor lze nalézt na přiloženém CD v adresáři */cross-compilers/mingw32-powerpc-603e-linux-gnu/sources*. Poslední příkaz do adresáře DESTDIR nainstaluje všechny potřebné soubory pro překladač z Host na Target. Adresář *DESTDIR* se používá kvůli tomu, aby se v dalších krocích tyto soubory mohly využít pro sloučení s již vytvořeným překladačem z Build na Target (viz. [13]).

```
tar -xvf binutils-2.18.tar.gz
mkdir binutils-build
cd binutils-build
./binutils.cfg
make all
make install DESTDIR=/home/p/buffer
```

4. Podobně jako v předchozím bodě se bude postupovat s překladem Gcc. Již vytvořený překladač z Build na Target (viz. [13]) byl přeložen s překladačem gcc-4.1.1 a proto je nutné ze zdroje [18] stáhnout stejný překladač gcc, tedy stejné zdrojové soubory. Konfigurační soubor lze nalézt na přiloženém CD v adresáři */cross-compilers/mingw32-powerpc-603e-linux-gnu/sources*.

```
tar -xjvf gcc-4.1.1.tar.bz2
mkdir gcc-build
cd gcc-build
./gcc.cfg
make all
make install DESTDIR=/home/p/buffer
```

5. Adresář *DESTDIR* je nutné zabalit programem *tar*, pro další postup na Host. Důležité je použití přepínače *-h*, který způsobí, že veškeré symbolické linky jsou nahrazeny skutečnými soubory.

```
cd /home/p/buffer
tar -h -cvf gcc-powerpc-603e-linux-gnu-addwin.tar usr
```

6. Stejné zabalení programem *tar* s přepínačem -h je nutné aplikovat i na původní překladač z Build na Target. Tento překladač je nutné rozbalit do nějakého dočasného a prázdného adresáře a znovu zabalit, právě s volbou *-h*. Soubor lze nalézt na přiloženém CD v adresáři */cross-compilers/x86-linux-powerpc-603e-linux-gnu*

```
tar -h -cvf gcc-powerpc-603e-linux-gnu.tar usr
```

7. Do instalačního adresáře MinGW rozbalit gcc-powerpc-603e-linux-gnu.tar.

```
tar -xvf gcc-powerpc-603e-linux-gnu.tar
```

8. Do instalačního adresáře MinGW rozbalit gcc-powerpc-603e-linux-gnu-addwin.tar.

```
tar -xvf gcc-powerpc-603e-linux-gnu-addwin.tar
```

9. Nastavit cestu pro překladač. Toto nastavení se provede přidáním následující řádky do souboru /etc/profile

```
export PATH=$PATH:/mingw/usr/bin
```

Příloha D

Modul Boa5200

Tato příloha popisuje zprovoznění OS GNU Linux se ssh serverem na modulu Boa5200 (viz. [5]) s procesorem PowerPC MPC5200 pro využití spolu s linux_grt_targetem, vytvořeným v této diplomové práci.

V následujících částech D.1 a D.2 jsou popsány doplňující informace, které spolu s dokumentací a zdrojovými soubory [6] vytvořenými na Katedře řídící techniky umožňují vytvoření OS GNU Linux (GNU Linoxové jádro 2.6 a file systém JFFS2) pro modul Boa5200. Následuje část D.3 o doimplementování ssh serveru Dropbear do OS GNU Linux, dále pak část D.4 o nahrání jádra OS GNU Linux 2.6 a file systému JFFS2 na modul Boa5200 a na závěr je uvedená část D.5 o konečné konfiguraci OS GNU Linux přímo na modlulu Boa5200, tedy zapsání skriptu pro automatické spuštění OS GNU Linux, finální konfiguraci ssh serveru a vytvoření uživatelského účtu pro bezpečné ssh přihlašování. Závěrečná část této přílohy shrnuje stav modulu Boa5200 s OS GNU Linux po aplikování výše popsaných operací.

D.1 Linuxové jádro 2.6

Překlad Linuxového jádra 2.6 byl vytvořen na Katedře řídící techniky a je popsán v dokumentaci [6]. Žádné další úpravy nejsou nutné. Přeložené linuxové jádro i se zdrojovými soubory je obsahem přiloženého CD v souboru */DP/CD/boa5200/powerpc-linux-2.6.tar*. Image přeloženého jádra je reprezentován souborem *zImage.elf*.

D.2 File system

Pracovní adresář *MPC5200_root* je možné stáhnout ze zdroje [7]. Jedná se o adresářovou strukturu obsahující programy přeložené pro procesory PowerPC s jádrem 603e. Pracovní adresář je nutné přeložit do formátu file systému JFFS2 (Journaling Flash File System version 2) pro budoucí upload do paměti flash na modul Boa5200 následujícím příkazem:

```
mkfs.jffs2 -r MPC5200_root -o myfs.jffs2 -b -e 0x10000
```

Tím vznikne soubor *myfs.jffs2* (file systém JFFS2) reprezentující pracovní adresář *MPC5200_root*. Finální verze upraveného pracovního adresáře *MPC5200_root*, který byl vytvořen úpravou původního pracovního adresáře podle následující části D.3 o implementaci ssh serveru je na přiloženém CD v souboru */boa5200/MPC5200_root.tar* a z něj vytvořený file system JFFS2 v souboru */boa5200/myfs.jffs2*.

D.3 Dropbear

Dropbear je open source ssh2 server a klient, viz. [8]. Je vhodný především pro embedded aplikace díky malým nárokům na paměťový prostor. Následuje postup pro jeho překlad pro platformu PowerPC a jeho zakomponování do pracovního adresáře *MPC5200_root*.

Použitá verze Dropbear

Dropbear 0.5, viz. [8]

Definice platformy build a host

Build: GNU Linux, x86
Host: GNU Linux, PowerPC(core 603e)

Prerekvizity

Na platformě Build nainstalovaný cross compiler powerpc-603e-linux-gnu, který lze nalézt na přiloženém CD v adresáři */cross compilers/x86-linux-powerpc-603e-linux-gnu* nebo pod odkazem [6].

1. Vytvoření pracovního adresáře Dropbear. Rozbalení zdrojových souborů z balíčku dropbear-0.50.tar.gz. Vytvoření adresáře dropbear-install, kam se nainstalují přeložené programy a odkud se budou kopírovat do pracovního adresáře *MPC5200_root*.

```
mkdir Dropbear
cd Dropbear

# nakopírovat do adresáře Dropbear zdrojové soubory,
# nyní aktuální verze dropbear-0.50.tar.gz

tar -xvf dropbear-0.50.tar.gz
mkdir dropbear-install
```

2. Konfigurace pro platformu host.

```
cd dropbear-0.50
./configure --host=powerpc-603e-linux-gnu --disable-zlib
```

3. Překlad pro platformu host. Překládají se jenom programy nutné pro práci sshd serveru.

```
make PROGRAMS="dropbear dropbearkey dropbearconvert scp"
```

4. Instalace přeložených programů do adresáře *dropbear-install*.

```
make PROGRAMS="dropbear dropbearkey scp" \
install DESTDIR=/cesta k adresáři/dropbear-install
```

Přeložené binární spustitelné programy (stejné, které vzniknou v adresáři *dropbear-install*) pro platformu Host jsou na přiloženém CD v souboru */boa5200/dropbear-0.5-powerpc-603e-linux-gnu.tar*. Další část popisuje zakomponování obsahu adresáře *dropbear-install*, tedy ssh serveru, do pracovního adresáře *MPC5200_root*.

1. Obsah adresáře *dropbear-install* se nakopíruje do adresáře *MPC5200_root*, tedy do kořenové struktury budoucího JFFS2 file systému.

2. Vytvoření automatického startu ssh serveru během zavádění OS GNU Linux zahrnuje editaci souboru */etc/inittab* v pracovním adresáři *MPC5200_root*. Přidá se řádka *::once:/etc/init.d/start-sshd*, která se odkazuje na spouštěcí skript ssh serveru. SSH komunikace vyžaduje také nakonfigurování ethernetového rozhraní, skript spustí řádka *::once:/etc/init.d/config-eth0*. Důležitou vlastností je možnost nejenom čtení na JFFS2, ale i zápisu. Konfigurační skript spustí řádka *::once:/etc/init.d/mount-root*. Tyto skripty se spustí jednou po "namontování" systému.

```
#Start userland network services
...
::once:/etc/init.d/mount-root
::once:/etc/init.d/config-eth0
::once:/etc/init.d/start-sshd
```

Následující skripty je nutné vytvořit v adresáři */etc/init.d/*.

Skript *mount-root*, který povolí zápis na file system JFFS2:

```
#!/bin/sh
...
echo -n "... mount /dev/root"
mount -o remount,rw /dev/root
echo ": done"
sleep 1
```

Skript *config-eth0*, který nakonfiguruje ethernetové rozhraní eth0 na adresu 192.168.0.100:

```
#!/bin/sh
...
echo -n "... config eth0"
ifconfig eth0 192.168.0.100 netmask 255.255.255.0 up
echo ": done"
sleep 1
```

Skript *start-sshd*, který spouští ssh server (dropbear)":

```
#!/bin/sh
...
echo -n "... start sshd (dropbear)"
export PATH=$PATH:/usr/local/bin:/usr/local/sbin
dropbear -d /etc/ssh/dropbear_dss_host_key -r \
/etc/ssh/dropbear_rsa_host_key -p 22
echo ": done"
sleep 1
```

Popsané skripty lze nalézt v přiloženém CD v adresáři */boa5200/start-scripts*.

3. Pro budoucí umístění klíčů pro ssh server vytvořit adresář */etc/ssh* v pracovním adresáři *MPC5200_root*.

4. Zbývá vytvořit z pracovního adresáře *MPC5200_root* file system JFFS2 podle příkazu z části D.2.

D.4 Nahrání jádra Linux 2.6 a File systému na modul Boa5200

Podle dokumentace [6] je možné pripojit modul Boa5200 asynchronní sériovou linkou RS232 s parametry Baud Rate:38400, Bits: 8, Parity: No, StopBits: 1. Po zapnutí napájecího napětí proběhne RedBoot (Red Hat Embedded Debug and Bootstrap, viz. [9]), který skončí příkazovou řádkou. RedBoot během svého spuštění akceptuje přidělení IP adresy pro modul Boa5200 od DHCP serveru a umožňuje nahrávání souborů přes TFTP (Trivial File Transfer Protocol). Jednoduchý DHCP server a TFTP server pro Windows XP obsahuje freeware software Tftpd32. Instalační soubor Tftpd32 je obsažen v přiloženém CD v adrešáři *software*. Po správném nastavení DHCP serveru a TFTP serveru je RedBootem akceptována IP adresa a v případě, že FTFP má nastaven zdrojový adresář do míst, kde je obsaženo jádro GNU Linuxu 2.6 (zImage.elf, viz. D.1) a vytvořený file systém JFFS2 (myfs.jffs2, viz. D.2 a D.3), můžeme GNU Linux (zImage.elf a myfs.jffs2) z příkazové řádky Redbootu nahrát do flash paměti modulu Boa5200 následujícími příkazy:

```
fis init
mfill -b 0x100000 -l 0xFF0000 -p 0xFFFFFFFF
load -r -v -b 0x100000 /myfs.jffs2
fi cr JFFS2 -l 0xFF0000
load -v -b 0xFF0000 zImage.elf
fi cr Linux
reset
```

D.5 Konfigurace

D.5.1 Spouštěcí skript pro OS GNU Linux

Po nahrání file systému JFFS2 a jádra Linux 2.6 na modul Boa5200 v části D.4 a resetu modulu Boa5200 lze příkazem *fconfig* z příkazové řádky RedBootu zadat následující příkaz pro automatické spuštění OS GNU Linux.

```
exec 0xF1050000
```

Parametrem příkazu *exec* je místo ve flash paměti modulu Boa5200, kam se nahrálo jádro Linuxu 2.6 pod jménem Linux. To lze zjistit příkazem *fis list*.

D.5.2 Konfigurace ssh serveru

Po dalším resetu modulu Boa5200 se automaticky spustí OS GNU Linux, povolí právo zápisu do file systému JFFS2, nakonfiguruje ethernetové rozhraní na IP adresu 192.168.0.100. Při prvním spuštění je však ssh server spuštěn s chybou, protože nemá vygenerované klíče. Ty lze vygenerovat následujícím způsobem.

```
cd /etc/ssh
/usr/local/bin/./dropbearkey -t rsa -f dropbear_rsa_host_key
/usr/local/bin/./dropbearkey -t dss -f dropbear_dss_host_key
```

Zbývá nastavit heslo pro superivzora.

```
passwd root
```

Po dalším resetu modulu Boa5200 se ssh server spustí správně a od této chvíle je možné se k modulu připojit ssh klientem jako root.

D.5.3 Uživatelský účet pro ssh

Připojení ssh klientem k modulu Boa5200 jako root není z bezpečnostního hlediska vhodné, protože neopatrnou prací, nebo špatně napsanými programy je možné zničit celý OS. Následuje kód, který vytvoří uživatelský účet na jméno *rtw* a heslo *rtw*.

```
cd /
mkdir home
cd home
mkdir rtw
adduser -h /home/rtw rtw
chown rtw:rtw rtw
```

D.6 Shrnutí

Výsledný stav modulu Boa5200 je po provedení výše popsaných postupů následující:

- Po zapnutí napájecího napětí se automaticky spustí OS Linux GNU, IP adresa se nastaví na hodnotu 192.168.0.100.
- Na OS GNU Linux jsou vytvořeny dva účty se jmény *root* a *rtw*. Hesla k účtům jsou pro modul Boa5200 použitý v této diplomové práci nastavena na ekvivalent přihlašovacího jména.
- K modulu je možné připojení přes asynchronní sériovou linku RS232 s parametry Baud Rate:38400, Bits: 8, Parity: No, StopBits: 1 a program terminál.
- K modulu je možné připojení přes ethernet a program telnet.
- K modulu je možné připojení přes ethernetové rozhraní využívající program ssh a scp. To umožňuje připojení jako *root* a správu modulu, dále pak připojení jako uživatel *rtw*. V obou případech je možné kopírování souborů směrem na modul programem *scp*.

Příloha E

Demonstrační úloha

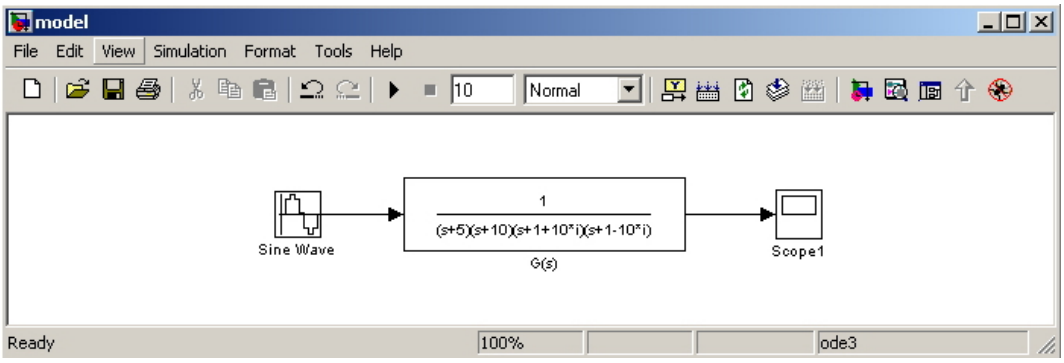
Tato příloha má za úkol ukázat výsledky diplomové práce *Podpora simulace s hardware ve smyčce* na praktickém příkladu. Dále poskytnout návod vedoucí k vytvoření příkladu, který je sepsán proto, aby mohl být odrazovým můstkem pro mnohé, kteří budou diplomovou práci využívat. Neposledním záměrem je vyjasnění mnohých nesrovnalostí, které můžou vzniknout nedostatečným prostudováním problematiky. Použitý a nakonfigurovaný model využitý v demonstrační úloze je obsažen na přiloženém CD v souboru */demonstracni-uloha/model.mdl*.

Následující postup předpokládá osobní PC s procesorem x86 s instalací OS Windows XP, Matlab 7.0.1 (R14) Service Pack 1 a Simulink 6.1 (R14) Service Pack 1 s nástrojem RTW (Real Time Workshop). Dále instalaci a nastavení emulátoru Linuxu MinGW s Msys podle přílohy C. Nakonec zprovozněný modul Boa5200 podle přílohy D.

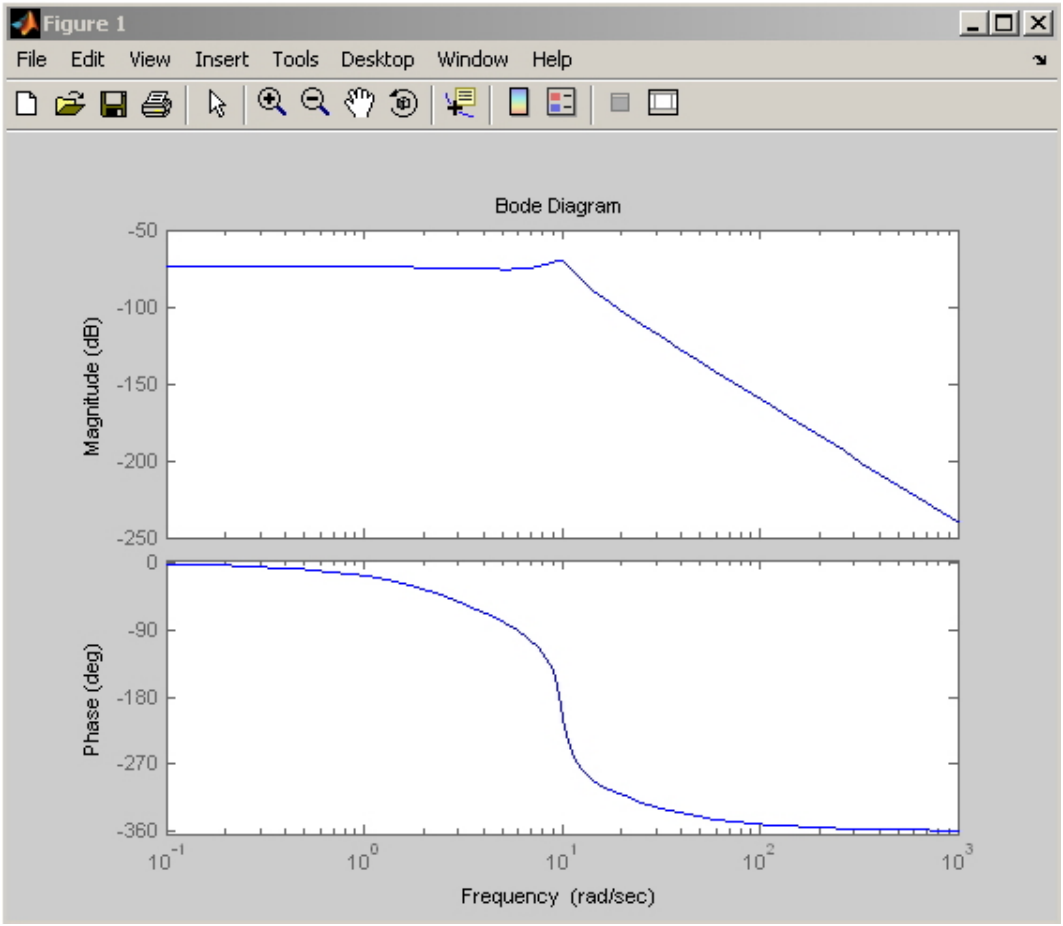
1. V Simulinku je vytvořen model s následující přenosovou funkcí.

$$G(s) = \frac{1}{(s+5)(s+10)(s+1+10i)(s+1-10i)}$$

   Frekvenční charakteristika přenosové funkce je vidět na obr. E.2. Do této přenosové funkce vstupuje sinusový signál a na výstup je umístěn osciloskop. Vytvořený model je vidět na obr. E.1.

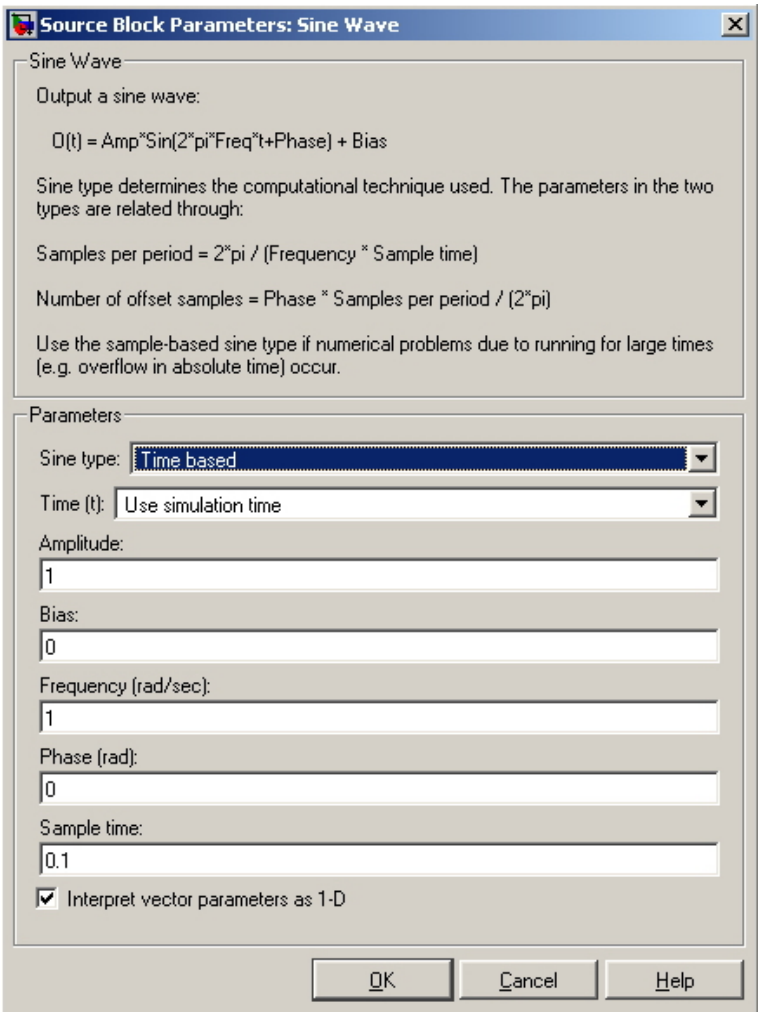


Obrázek E.1: Model v Simulinku



Obrázek E.2: Frekvenční charakteristika přenosové funkce

2. Provede se editace bloku *Sine Wave* a nastaví se vzorkovací perioda *Sample Time* na hodnotu *0.1s* (viz. obr. E.3).



Obrázek E.3: Konfigurace bloku *Sine Wave*

3. Pracovní adresář Matlabu se nastaví do umístění souboru modelu.

4. Zdrojové soubory targetu (viz. přiložené CD, adr. */linux_grt_target*) se umístí na disk *c:* (viz. Kapitola 2.3) a cesty

   ```
   c:\...\linux_grt_target\linux_grt_target
   c:\...\linux_grt_target\linux_grt_target\RS232
   ```

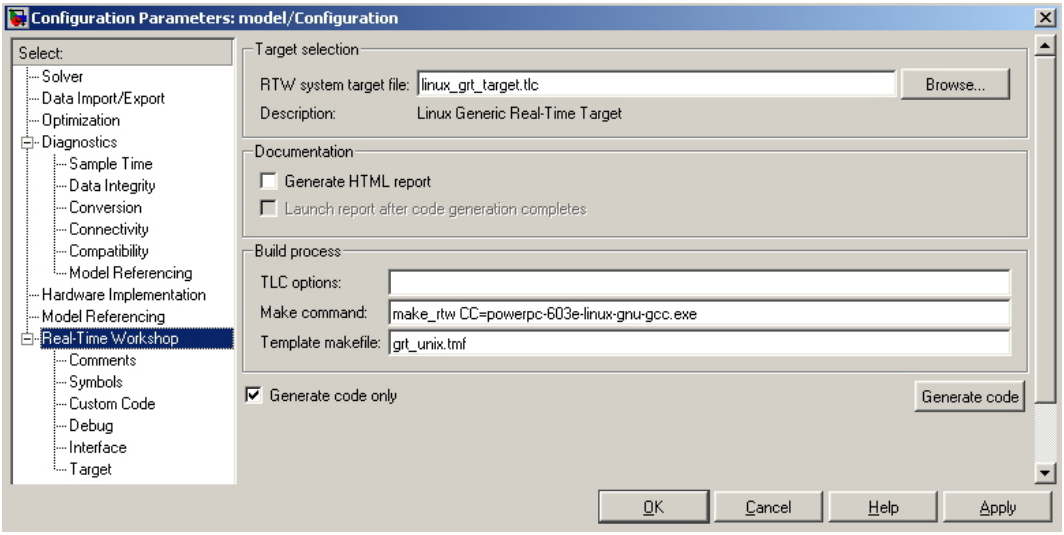
   se přidají do cest Matlabu.

5. Z menu v okně modelu se zvolí *Simulation:Configuration parameters*. Tak se spustí konfigurační okno.

6. Ze záložky *Real-Time Workshop* se v oblasti *Target selection* vybere target *linux_grt_target.tlc*.

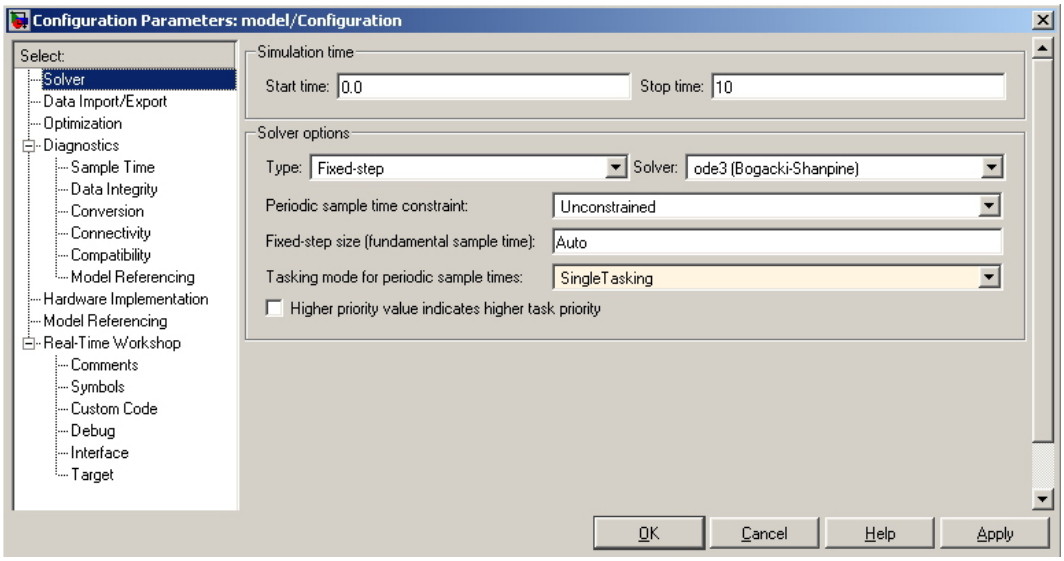
   V oblasti *Build process* se do editační řádky *Make command* přidá argument ***CC=powerpc-603e-linux-gnu-gcc.exe***, který říká, jaký překladač bude pro překlad vygenerovaného kódu v emulátoru Linuxu MinGW použit.

   Nastavení by mělo vypadat podle obr. E.4.



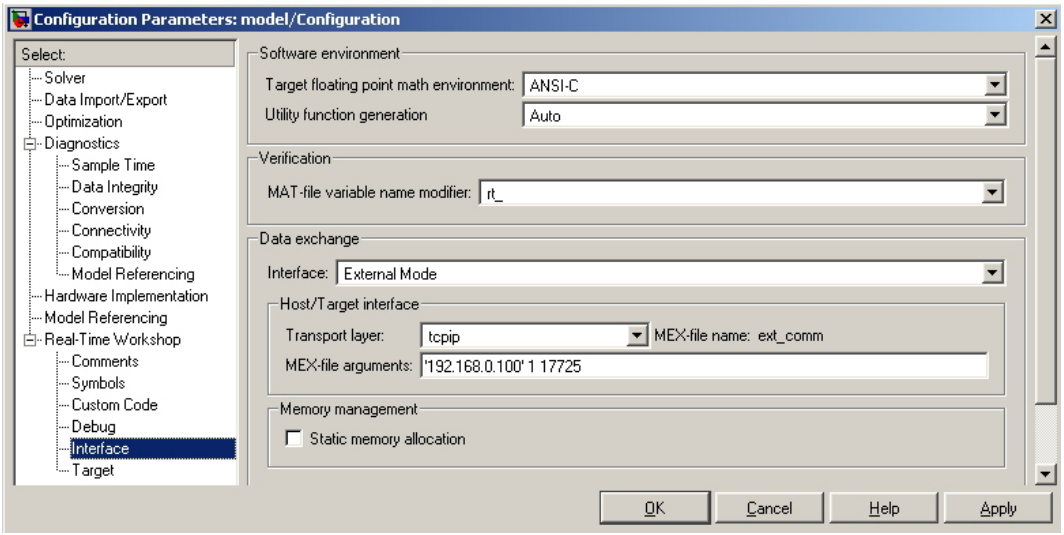
Obrázek E.4: Volba targetu

7. V záložce *Solver* a oblasti *Solver options* se nastaví položka *Type* na hodnotu *Fixed-step* a *Tasking mode for periodic sample times* na hodnotu *Single tasking* , jak je to ukázáno na obr. E.5.



Obrázek E.5: Záložka Solver

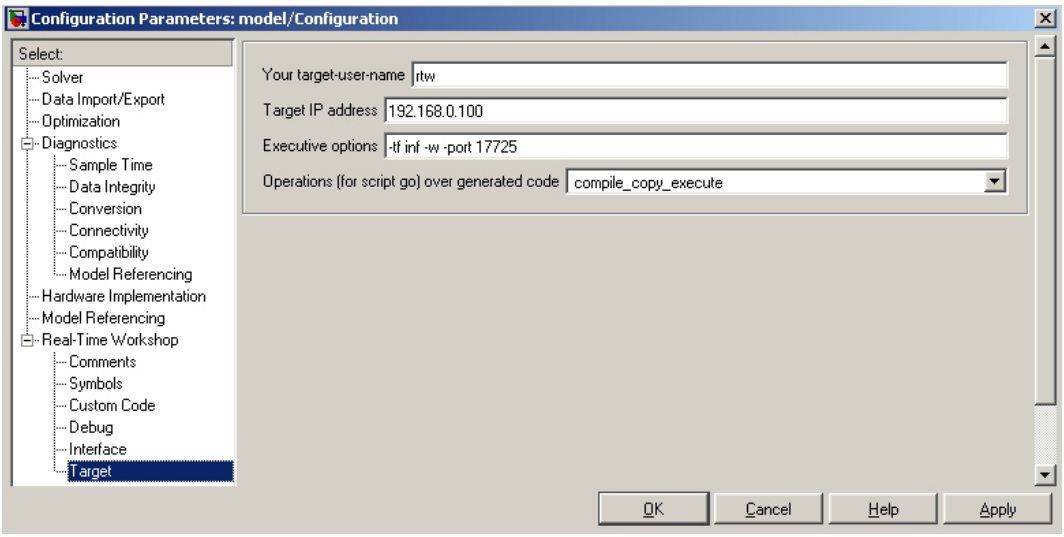
8. V záložce *Real-Time Workshop:Interface* se nastaví Externí rozhraní podle obr. E.6. Argument *192.168.0.100* udává IP adresu modulu Boa5200, argument *1* zapíná hlášení o probíhající komunikaci a argument *17725* označuje komunikační port pro TCP/IP protokol.



Obrázek E.6: Záložka Interface

9. V záložce *Real-Time Workshop:Target* se provede nastavení pro vygenerování skriptu

go podle obr. E.7. *Rtw* je název uživatelského účtu na OS GNU Linux na modulu Boa5200 a IP adresa *192.168.0.100* je adresou modulu Boa5200. Argumenty *-tf inf -w -port 17725* pro spuštění bin. kódu modelu pod OS GNU Linux na modulu Boa5200 říkají, že doba běhu kódu na modulu Boa5200 není omezena, spuštění běhu kódu bude provedeno přes Externí rozhraní a bude využit komunikační port 17725. *Compile_copy_execute* znamená, že skript *go* bude vygenerován pro operaci překladu vygenerovaného kódu z modelu v Simulinku, nahrání přeloženého kódu na modul Boa5200 a následné spuštění.



Obrázek E.7: Záložka Target

10. Vygenerování C kódu z modelu do pracovního adresáře Matlabu se provede stiskem tlačítka ***Generate code*** ze záložky *Real-Time Workshop*. Průběh generování kódu se vypisuje v hlavním okně Matlabu.

11. Na osobním PC s modelem v Simulinku se nastaví ethernetové rozhraní na statickou IP adresu, např. 192.168.0.10.

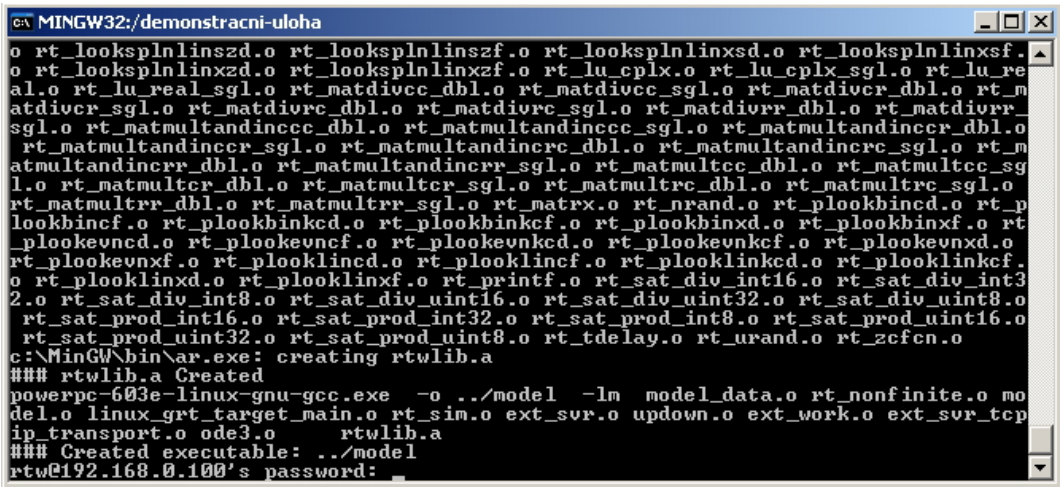
12. Modul Boa5200 se propojí s osobním PC ethernetovým rozhraním (křížený kabel).

13. Na modul Boa5200 se připojí napájecí napětí a vyčká se cca. 1 minutu, než se spustí OS GNU Linux. Otestovat spuštění lze z příkazové řádky osobního PC příkazem *ping 192.168.0.100*.

14. Na osobním PC se spustí emulátor Linuxu Mingw s Msys a vstoupí se do pracovního adresáře Matlabu, tedy do míst, kam byl vygenerován kód z modelu.

15. Příkazem *./go* se provede překlad kódu, nahrání přeloženého kódu na modul Boa5200 a jeho spuštění. Na konci překladu kódu je důležité si všimnout řádky *### Created executable: ../model* (viz. obr. E.8), která informuje o úspěšném přeložení kódu do binárního souboru *model*. Následují dvě výzvy k zadání hesla (*rtw*). Poprvé to vyžaduje program *scp* kvůli nakopírování souboru *model* na modul Boa5200 a podruhé program *ssh* kvůli spuštění soboru *model* pod OS GNU Linux na modulu Boa5200.

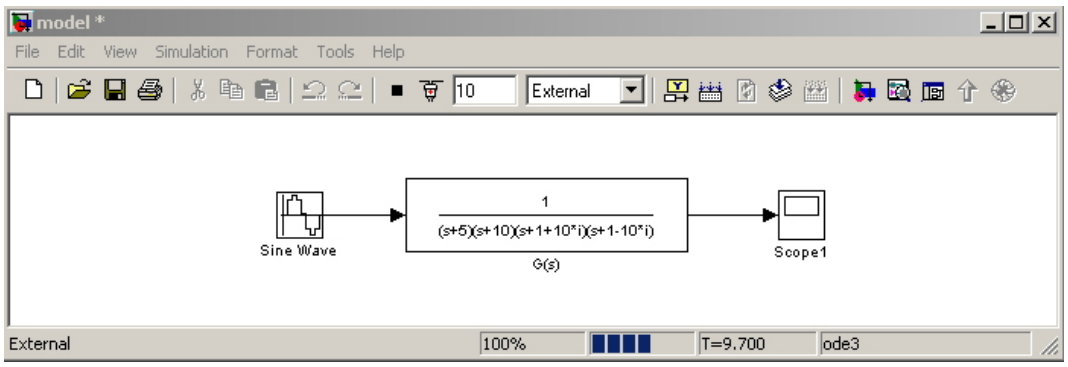
    Pozn.: Může se stát, že získaný binární soubor (*model*) nebude mít práva pro spuštění. Změna práv pod emulátorem Linuxu MinGW příkazem *chmod* není možná kvůli jeho chybnému chování. Potom je nutné přihlášení programem ssh (příkaz *ssh rtw@192.168.0.100*) na účet *rtw* pod OS GNU Linux na modulu Boa5200 a změna práv k binárnímu souboru příkazem *chmod 700 model* přímo na modulu.



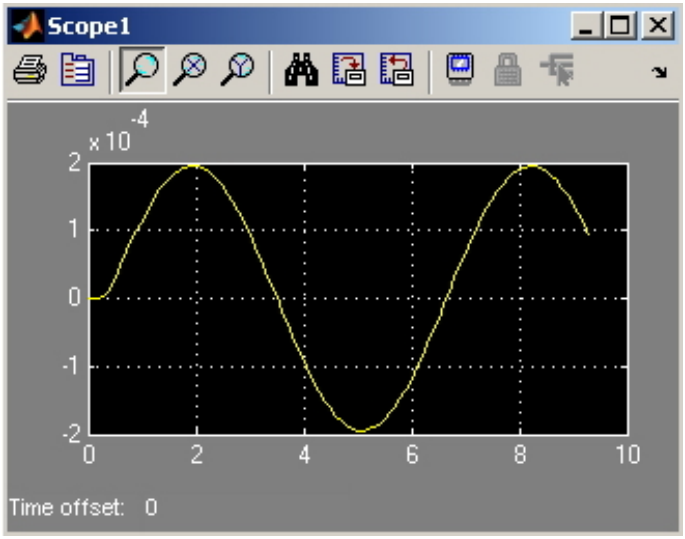
Obrázek E.8: Překlad vygenerovaného kódu

16. Pripojení k již spuštěnému kódu na modulu Boa5200 avšak čekajícímu na spuštění běhu kódu přes External mode se provede z okna modelu v Simulinku volbou *Simulation: Connect To Target*. Spuštění vykonávání kódu modelu provede volba *Simulation: Start Real-Time Code*.

17. Spuštěný model na modulu Boa5200 v Simulinku pomocí External mode je vidět na obr. E.9. Získaný průběh ze Scope1 z Targetu pomocí External Mode je na obr. E.10



Obrázek E.9: Běžící model v režimu External mode



Obrázek E.10: Signál zaznamenaný pomocí External mode

18. Ukončení běhu kódu na modulu Boa5200 a ukončení spojení External mode se provede volbou *Simulation: Stop Real-Time Code* z okna modelu v Simulinku.

Příloha F

Seznam použitého software

- OS Windows XP Professional
- Matlab 7.0.1 (R14) Service Pack 1
- Simulink 6.1 (R14) Service Pack 1 a RTW (Real Time Workshop)
- Cygwin 1.5.24
- MinGW 5.1.3 a Msys 1.0
- Tftpd32 3.22
- OS GNU Linux (jádro 2.6)
- Mandriva 2007 (OS GNU Linux)
- Dropbear 0.5
- amltd_tools-jffs2_utils-1.45-2.i386

Křížové překladače obsažené na přiloženém CD

- cygwin-gcc-3.3.6-glibc-2.3.2-linux
- x86-linux-powerpc-603e-linux-gnu
- x86-mingw32-build-sh (skripty pro vytvoření křížového překladače)

Vytvořený křížový překladač obsažený na přiloženém CD

- mingw32-powerpc-603e-linux-gnu

Příloha G

Obsah přiloženého CD

Tabulka G.1: Obsah přiloženého CD

/Boa5200		
	/start-scripts	*startovací skripty pro OS GNU Linux*
	/zImage.elf	*image Linuxového jádra 2.6 přeložené pro procesory PowerPC(core 603e)*
	/powerpc-linux-2.6.tar	*zdrojové soubory Linuxového jádra 2.6 a přeložené jádro pro procesory PowerPC(core 603e)*
	/myfs.jffs2	*jffs2 file systém vytvořený z pracovního adresáře MPC5200_root*
	/MPC5200_root.tar	*upravený pracovní adresář*
	/dropbear-0.5-powerpc-603e-linux-gnu.tar	*přeložený ssh server pro OS GNU Linux s procesorem PowerPC(core 603e)*
	/amltd_tools-jffs2_utils-1.45-2.i386.rpm	*balíček pro vytváření JFFS2 file systému pod OS GNU Linux*

/cross compilers		
	/cygwin32-x86-linux-gnu	*adresář s křížovým překladačem z Cygwin na OS GNU Linux, vše na procesoru x86*
	/mingw32-powerpc-603e-linux-gnu	*adresář s křížovým překladačem z MinGW (x86) na OS GNU Linux (PowerPC s jádrem 603e) a konfigurační soubory použité při vytváření*
	/x86-linux-powerpc-603e-linux-gnu	*adresář s křížovým překladačem z OS GNU Linux,x86 na OS GNU Linux (PowerPC s jádrem 603e)*
	/x86-mingw32-build-sh	*adresář se skripty pro automatické vytvoření překladače pro mingw (x86)*
/demonstracni-uloha		
	/modellinux_grt_rtw	*adresář s vygenerovaným kódem z modelu*
	/slprj	*adresář s vygenerovaným kódem z modelu*
	/model.mdl	*model v Simulinku*
	/model	*výsledný spustitelný binární kód z modelu*
	/go	*skript vygenerovaný z modelu*
/linux_grt_target		
	/linux_grt_target	*soubory linux-grt-targetu*
/software		
	/Tftpd32-3.22-setup.exe	*jednoduchý TFTP a DHCP server pro Windows XP*
/dp_Jelinek_Pavel.pdf		*diplomová práce*