VĚDECKÉ SPISY VYSOKÉHO UČENÍ TECHNICKÉHO V BRNĚ

Edice PhD Thesis, sv. 785

ISSN 1213-4198

Ing. Vojtěch Lamberský

Návrh metod a nástrojů pro zrychlení vývoje softwaru pro vestavěné procesory se zaměřením na aplikace v mechatronice

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

Ing. Vojtěch Lamberský

NÁVRH METOD A NÁSTROJŮ PRO ZRYCHLENÍ VÝVOJE SOFTWARU PRO VESTAVĚNÉ PROCESORY SE ZAMĚŘENÍM NA APLIKACE V MECHATRONICE

DESIGN OF METHODS AND TOOLS
ACCELERATING THE SOFTWARE DESIGN
FOR EMBEDDED PROCESSORS TARGETED
FOR MECHATRONICS APPLICATIONS

Zkrácená verze Ph.D. Thesis

Obor:	Aplikované vědy v inženýrství
Vedoucí práce:	doc. Ing. Robert Grepl, Ph.D.
Oponenti:	doc. Ing. Stanislav Věchet, PhD.
	doc. Ing. Gergely Takács, PhD.
Datum obhajoby:	23. června 2015

Klíčová slova:
automatické generování kódu, Cerebot MX7 cK, Simulink toolbox, benchmark vestavěných procesorů, predikce doby výpočtu

Keywords:
automatic code generation, Cerebot MX7 cK, Simulink toolbox, benchmarking of embedded processors, predicting execution time

Místo uložení práce:
Vysoké učení technické v Brně
Fakulta strojního inženýrství
Technická 2896/2
616 69 Brno (CZ)

ISBN 978-80-214-5234-0
ISSN 1213-4198

Obsah

1 ÚVOD

Zejména v oblasti vestavěných mikroprocesorů dochází v poslední době k velmi významnému nárůstu jejich výpočetního výkonu doprovázeného snižováním jejich ceny [1]. To přirozeně vede ke zvýšení poptávky po těchto řídících modulech tím, jak nalézají uplatnění ve stále větším počtu zařízení. Proto je přirozené vyvíjet nové nástroje umožňující zjednodušit a zefektivnit [2] vývoj produktů používajících vestavěné procesory [3]. Z této skupiny nástrojů je tato práce zaměřena na nástroje pro podporu vývoje programů pro vestavěné procesory.

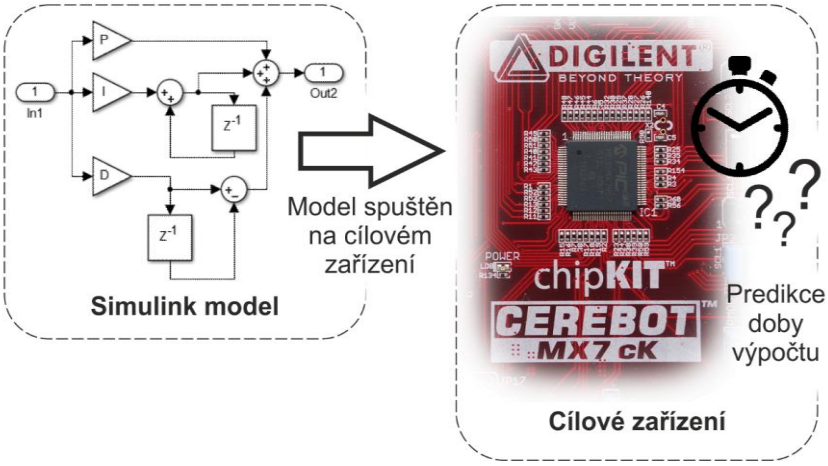
Právě díky vysoké efektivitě při vývoji algoritmů pro vestavěné procesory, se stávají stále oblíbenějšími a častěji používanými vyšší programovací jazyky [4]. Kromě toho, moderní vývojová prostředí jsou obvykle velmi dobře provázaná s ostatními nástroji, používanými při vývoji nového zařízení. Jedním z nástrojů, které poskytují prostředí postavené na vyšším programovacím jazyku umožňujícím *model based design* vývoj algoritmů je Matlab/Simulink [5]. Tento programový balík navíc poskytuje nástroje, které umožňují na základě Simulink modelu automaticky vygenerovat C kód, který je optimalizovaný pro vykonávání na méně výkonných - vestavěných procesorech.

Jedním z hlavních témat této práce je demonstrovat postup a možnosti, jak vytvořit nástroje, které umožní plně automaticky generovat kód pro vybraný vestavěný procesor z prostředí Matlab/Simulink. Plně automatickým generováním kódu se rozumí funkce, která po stisknutí jednoho tlačítka v Simulink modelu daný model přeloží do spustitelného kódu a ten je dále nahrán a spuštěn v cílovém zařízení. Tento proces se skládá z několika kroků jednak na úrovni Simulink a následně jsou vygenerované soubory dále zpracovány mimo toto prostředí. Jedním z problémů při generování kódu je obsluha komplexních periferií. Tato práce má za cíl najít způsob, kterým by bylo možné vytvořit program používající grafický displej přímo z prostředí Simulink.

Pokud navrhujeme algoritmus ve vyšším programovacím jazyce, bývá velmi obtížné odhadnout, jak rychle se vytvořený algoritmus bude vykonávat při spuštění na cílovém zařízení. Přitom právě dostatečně přesně odhadnout rychlost vykonávání navrženého algoritmu na konkrétním hardwaru, je klíčové při rozhodování, jestli je pro danou aplikaci toto zařízení vhodné.

Druhá hlavní část této práce rozebírá možnosti predikce doby výpočtu na vybraných hardwarových zařízeních. Jak rychle se bude konkrétní algoritmus vykonávat na zvoleném cílovém zařízení, ovlivňuje celá řada faktorů. V první řadě samotný výpočetní výkon daného zařízení, dále pak nastavení parametrů překladu z vyššího programovacího jazyka do spustitelného kódu. V odpovídajících kapitolách této práce je celý proces překladu Simulink modelu do spustitelného kódu popsán a na konkrétních příkladech je demonstrováno, jak se změní doba výpočtu algoritmu při změně těchto parametrů.

Pro ilustraci, obrázek 1. představuje Simulink model použitý k vytvoření programu a cílový hardware. Toto schéma ilustruje funkci nástrojů, které tato práce popisuje. První z nich je sada programů, která umožňuje vygenerovat ze Simulink kód a tímto kódem naprogramovat vestavěný procesor. Druhou skupinu představují metody, které umožní odhadnout, jak rychle se bude tento kód vykonávat na zvoleném procesoru.



Obrázek 1. Simulink model použitý k naprogramování vestavěného procesoru

2 DOSTUPNÉ NÁSTROJE V OBLASTI VESTAVĚNÝCH PROCESORŮ

Na trhu je celá řada mikro kontrolérů lišících se jednak svým výpočetním výkonem a dostupnými vývojovými nástroji (se kterým lze tyto zařízení naprogramovat). Cílem této kapitoly je představit jednak základní kategorie dostupných hardwarových zařízení a v druhé řadě programová prostředí (jazyky), které jsou vhodné pro tyto zařízení.

2.1 KATEGORIE HARDWARU

V dnešní době je k dispozici celá řada nástrojů určených pro rychlé vyvíjení prototypů. Oproti hardwaru určeného pro produkci se liší především tím, že je možné jej po vybalení začít ihned používat (pokud si koupíme samotný vestavěný procesor, musíme k němu nejprve přidat modul pro napájení, oscilátor, kondenzátory a další prvky, než je možné jej „spustit“ a komunikovat s ním). Dále je tento hardware vybaven konektory, takže k němu lze snadno připojit další komponenty (bez nutnosti pájení). Obvykle je součástí takového vývojového prostředí i softwarové vybavení umožňující velmi komfortní a rychlý vývoj (naprogramování) aplikací. Tyto vývojové nástroje lze je rozdělit podle několika parametrů, poměrně intuitivní je rozdělení podle ceny (jehož kategorie jsou v zásadě shodné jako kategorie rozdělení zařízení podle výkonu).

Výkonný hardware pro rychlý vývoj aplikací

Do kategorie hardware určeného pro vývoji zařízení s velmi vysokým výpočetním výkonem, lze zařadit například vybrané produkty firmy DSpace a National Instruments. Obě firmy nabízejí zásuvné moduly s periferiemi do základních platforem (DSpace je označuje jako procesorové desky, firma NI jako PXI platformy). Tento výpočetně výkonný hardware používá svůj vlastní dedikovaný procesor, na kterém běží simulace.

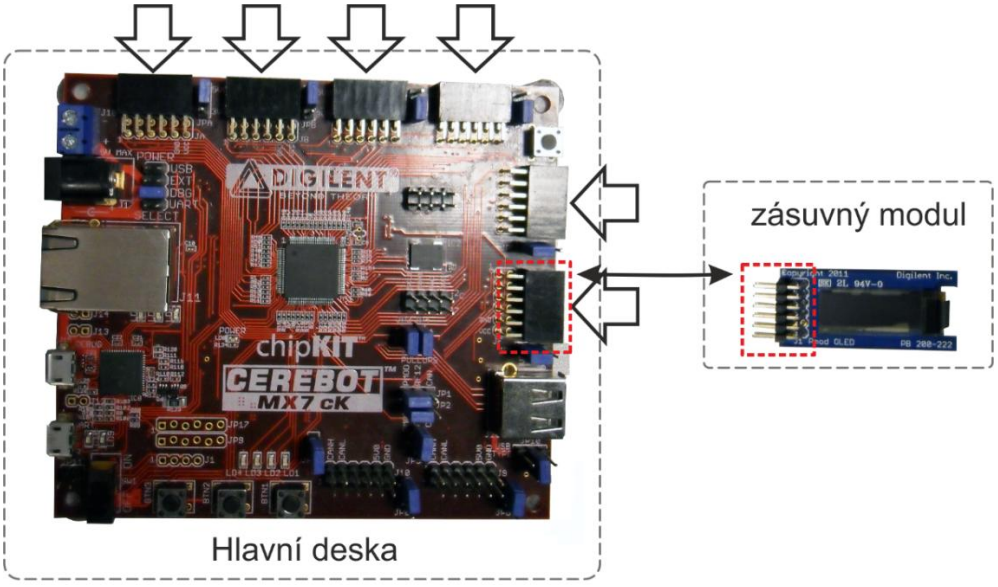
Kromě výše zmíněných zařízení by do této kategorie patřily i realtimové karty (např. karta MF624) firmy HUMUSOFT. Ty při běhu simulace používají hlavní procesor osobního počítače, ke kterému jsou připojené. To do jisté míry omezuje jejich výkon a použití v aplikacích, kde je důležité striktní časování, protože běžné operační systémy (například Windows), nedokáží garantovat přidělení hardwarových prostředků vybranému procesu v určitém čase.

Hardware s nižším výkonem pro rychlý vývoj aplikací

Do této kategorie patří všechny vývojové nástroje, které jsou osazené vestavěným procesorem s nízkým výpočetním výkonem. (obvykle jsou to 16 nebo 32 bitové procesory zpravidla nejsou vybavené koprocesorem urychlující matematické operace s čísly s plovoucí desetinnou čárkou).

Mezi nejznámější zástupy této kategorie se řadí hardware Arduino (používající čip ATmega), PIC vývojové desky (používají procesory Microchip), vývojové kity od firmy Texas Instruments, (používají procesory ARM).

Použitý hardware



Obrázek 2. Cerebot hardware schéma připojování zásuvných modulů s periferními zařízeními

Cerebot MX7 Ck je vývojový kit osazený jedním z nejvýkonnějších 32 bitových vestavěných procesorů firmy Microchip, konkrétně PIC32MX795. Hardware je vybaven „standardizovanými“ porty, Moduly s periferními zařízeními lze tedy jednoduše připojit do portu pomocí konektoru, bez nutnosti vytvářet „propojky“. U zařízení, které nemají specifikované rozhraní, je nutné propojit jednotlivé piny mezi procesorem a periferií ručně, což je nesrovnatelně pomalejší, než propojení dvou portů připraveným kabelem. Podle vyvíjené aplikace zvolíme moduly, které jednoduše připojíme k základní desce, Na obrázku 2 je tento koncept zásuvných modulů ilustrován.Jednotlivé volné porty na základní desce jsou označené šipkami a zásuvný modul je na obrázku vpravo. Další výhodou hardwaru Cerebot je velké množství hotových zásuvných modulů, které lze k základní desce připojit.

Jak už bylo řečeno, tato platforma je velmi levná, lze ji tedy použít i v produktech pro koncové zákazníky. Nicméně i přes svoji nízkou cenu je vybavena procesorem, který je dostatečně výkonný pro většinu aplikací (ať už řízení, zpracování dat nebo multimediální aplikace). Zejména cena tohoto zařízení, ale i relativně vysoký výpočetní výkon předurčuje tento hardware pro použití v mechatronických aplikací, od řízení a regulace přes zpracování signálu po práci s daty nebo jako výuková platforma.

2.2 VÝVOJ SOFTWARU PRO VESTAVĚNÉ PROCESORY

Ačkoliv principálně fungují vestavěné procesory stejně jako procesory v osobních počítačích, vzhledem k jejich použití mají určité odlišnosti, které do určité míry ovlivňují i způsob jakým se pro tyto zařízení vyvíjí programy.

Oproti stolním počítačům jsou vestavěné procesory daleko jednodušší a během jejich životnosti obvykle nedochází ke změně programu, který vykonávají. Navíc, program pro určitý vestavěný procesor, se obvykle nikdy nepoužije pro jiný typ hardwaru, než pro který byl vyvinutý. Což je jeden z důvodů, proč se ve většině případů aplikací používajících vestavěné procesory nepoužívá operační systém (ani základní BIOS). To má určité výhody, předně máme úplnou kontrolu nad přístupem k periferiím a víme přesně, co bude kód dělat. Určitou nevýhodou pak může být nutnost napsat kód pro obsluhu periferií a časování simulace, kterou by jinak zajišťoval operační systém a komplikovanou přenositelnost programu.

Vytvořit program pro mikrokontroléry je poměrně složitý úkol. Program v „běžných“ mikro kontrolérech může obsahovat přes půl milionu instrukcí. Psát program řetězením jednotlivých instrukcí ručně by v tomto případě bylo velmi obtížné a neefektivní (s případnou výjimkou nejjednodušších 4 bitových kontrolérů s jednoduchým řídícím algoritmem, které mohou být použity například v termostatech nebo časovačích).

Snaha zjednodušit vývoj programu vedla k vytvoření celé řady programovacích jazyků. Podkapitoly níže představí nejpoužívanější programovací jazyky pro vývoj programů pro vestavěné procesory.

HEX formát

Hex formát není programovací jazyk pouze standardizovaný formát, jakým se popisuje umístění jednotlivých instrukcí ve FLASH paměti. Původně jej zavedl Intel, ale dodnes se používá. Jeho specifikace je volně dostupná [6]. V podstatě lze tímto způsobem zapsat kód, o kterém budeme přesně vědět, co dělá (na úrovni jednotlivých registrů), nicméně kvůli nízké rychlosti kódování programu se příliš nepoužívá. Přesto ale většina debuggerů umožňuje prohlížet a případně i přepisovat hodnoty v paměti FLASH, obdobným způsobem jako bychom pracovali v HEX editoru.

Jazyk Assembler

Jazyk Assembler vznikl z potřeby zefektivnit psaní kódu. Pokud editujeme kód přímo, musíme0 jej v podstatě vždy napsat znova. Tento hlavní problém odstraňuje jazyk assembler, kdy umožňuje používat symbolické adresy, konkrétní místo v paměti (adresa) se přiřadí až při sestavování programu. Díky tomu může být určitá funkce znovu použitá v dalších projektech. Nicméně dnes se tento jazyk používá především v případech, kdy je potřeba mít kontrolu nad tím jaká instrukce se použije. Toho lze využít například tehdy, když potřebujeme napsat efektivní kód a pracovat přímo s jednotlivými registry procesoru.

Jazyk C, C++

Jazyk C++ umožňuje oproti jazyku C vytvářet objektové typy, nicméně zachovává většinu příkazů jazyka C, takže většina dnes používaných C++ překladačů je schopná přeložit i kód C. Poměrně jednoduše tak můžeme sestavit projekty, ve kterých jsou některé unity napsané v jazyce C a jiné v C++. Přesto, že se již před mnoha lety předpovídalo, že jazyky C a C++ budou nahrazeny efektivnějšími jazyky, zůstává i dnes C nejpoužívanějším jazykem pro vytváření programů pro vestavěné procesory [7]. Tento stav může mít několik příčin, nejčastěji se uvádí rychlost vykonávání zkompilovaného kódu a dále poměrně velké množství knihovních funkcí napsaných v jazyce C. Z praktických důvodů je to pak i dostupnost překladačů, protože téměř pro všechny druhy procesorů je k dispozici překladač z jazyka C, jiné jazyky tak rozsáhlou podporu napříč různými typy hardwarů nemají.

Modelovací jazyky

Tyto jazyky lze zařadit do skupiny „vysokých“ jazyků. Jejich výrazy a operátory se zásadně liší od těch, které provádí procesor (například maticové násobení je reprezentováno jedním symbolem v modelovacím jazyku, ale přeloženo jako několik desítek instrukcí, které provede procesor).

Hlavní předností grafických programovacích jazyků je efektivita vývoje programu, na jednom „řádku“ lze zapsat program, jehož popis by v jazyce C zabral desítky řádků. Díky tomu lze vyvíjet programy velmi rychle. Určitou nevýhodou může být rychlost vykonávání (efektivita) generovaného kódu, nicméně tento nedostatek se snaží odstranit různá vylepšení, která zlepšují výslednou rychlost vykonávání kódu generovaného pro vestavěné procesory z těchto prostředí.

Použité programovací jazyky

Jelikož je vývoj nástrojů automaticky generující kód poměrně komplexní úkol, při vývoji modulů implementující různé funkce je výhodné (případně nutné, pokud není k dispozici překladač pro daný hardware podporující jiný jazyk) kombinovat použití vyšších a nižších jazyků, tak jak je to vzhledem k jejich vlastnostem výhodné.

2.3 PREDIKCE DOBY VÝPOČTU

Existuje několik typů metod, které se používají k predikci výpočetního výkonu. Jednu skupinu představují metody porovnávající výkon hardwaru na základě jeho parametrů, což jsou různé typy predikcí na základě benchmark skóre nebo intuitivní odhad na základě známých parametrů procesoru. Druhým typem metod pro předpověď doby výpočtu jsou metody založené na modelování chování procesoru. Tedy takové metody, které simulují výpočet (jeho provádění) na konkrétním hardwaru a tím mohou určit, jak dlouho se bude daný algoritmus počítat.

Intuitivní odhad

Nejčastěji se používá intuitivní odhad výpočetního výkonu procesoru na základě parametrů, které uvádí výrobci. Tedy frekvence procesoru, typ jeho architektury, hloubka *pipeline* a podobně. Pochopitelně porovnávat jednotlivé procesory jen podle rychlosti je velmi nepřesné (s výjimkou, kdy srovnáváme dva identické procesory, z nichž každý pracuje s jinou výpočetní frekvencí). Celkový výkon procesoru ovlivňuje celá řada dalších parametrů. Například procesor, který má větší frekvenci numerického jádra může provést výpočet stejně rychle jako procesor pomalejší, ale s kratší *pipeline*, pokud bude provádět hodně skoků při výpočtu. Do jaké míry ovlivní rychlost výpočtu určitého algoritmu konkrétní vlastnost procesoru, je velmi obtížné určit, prakticky se nepoužívají žádné formalizované vzorce, pouze intuitivní odhad vycházející ze zkušenosti daného člověka, který predikci výkonu provádí.

Odhad výkonu použitím benchmarkových skóre

O něco objektivnější představu o výkonu procesoru lze získat na základě skóre z benchmarkovacích databází. V současné době existují benchmarkovací algoritmy vytvořené tak, aby rovnoměrně testovaly všechny vlastnosti procesoru (rychlost přístupu do paměti, vykonávání sekvenčního kódu, skoků a další) [8]. Pokud tedy odhadujeme výpočetní výkon pro aplikaci, která používá procesor stejným

způsobem (obsahuje stejné typy funkcí), tak dostaneme na základě porovnávání skóre poměrně přesný odhad doby výpočtu takového algoritmu.

Aby bylo možné přesněji odhadnout dobu výpočtu algoritmů s určitými charakteristickými vlastnostmi, byly vytvořeny benchmarkovací algoritmy, které svým charakterem co nejvíce odpovídají testovanému algoritmu. Byly vytvořeny databáze benchmarkových skóre pro multimediální aplikace, síťové služby, řízení a další, které spravuje organizace EMBC [9]. Pokud tedy potřebujeme odhadnout výkon algoritmu zpracovávajícího signál, dostaneme daleko přesnější odhad doby výpočtu, pokud provádíme odhad na základě benchmarkových skóre algoritmů pro filtraci signálu.

Odhad výkonu hardware na základě simulace

Velmi přesný odhad doby výpočtu můžeme získat, pokud přesně modelujeme průběh výpočtu na cílovém hardwaru. Tyto metody obvykle používají matematický model procesoru a na tomto modelu spustí výpočet algoritmu. Pokud je model dostatečně přesný (modeluje výpočet až na úrovni operací s registry), lze získat až 100% přesný odhad doby výpočtu. Nicméně tyto metody predikce jsou poměrně pomalé, uvádí se, že dokáží simulovat výpočet rychlostí kolem 100K instrukcí za sekundu [10]. Proto byly představeny varianty této metody, která kombinuje simulace a predikce na základě databáze s dobou výpočtů určitých funkcí [11], [12]. To umožňuje významně zrychlit proces predikce, ale použití statistických metod při predikci (což je nezbytné při použití informace z databáze dob výpočtu určitých funkcí), dojde ke snížení přesnosti predikce doby výpočtu [13].

3 CÍLE PRÁCE

Cílem této práce je navrhnout metody a vytvořit nástroje, které by umožnily zrychlit vývoj softwaru pro vestavěné procesory. Konkrétně pak nástroje pro plně automatické generování kódu z prostředí Simulink pro zvolené cílové zařízení a metody umožňující predikovat dobu výpočtu algoritmu navrženého v Simulink prostředí. I když jsou tyto nástroje určené především pro zjednodušení a zrychlení vývoje mechatronických aplikací, lze je použít i v jiných oborech.

3.1 NÁSTROJ PRO AUTOMATICKÉ GENEROVÁNÍ KÓDU PRO CEREBOT HARDWARE

Dílčím cílem je vytvoření Cerebot blocksetu, který umožní plně automatické generování spustitelného kódu z prostředí Simulink pro platformu Cerebot MX7 cK hardware a použít jednoduché periferie. Kromě toho je tento blockset určený i jako základní modul, který lze rozšířit o další (i komplexní) periferie.

3.2 NÁSTROJ PRO AUTOMATICKÉ GENEROVÁNÍ KÓDU PRO KOMPLEXNÍ PERIFERII

Dalším dílčím cílem je vytvořit nástroj, který umožní plně automaticky generovat kód pro obsluhu (vytvoření programu používajícího) komplexní periferii grafického displeje. Tento nástroj je nutné plně integrovat s vyšším programovacím jazykem (Simulinkem). A vytvořit k němu komfortní a intuitivní uživatelské rozhraní. Tento nástroj představuje „nadstavbu“ základního Cerebot blocksetu.

3.3 METODY PRO PREDIKCI DOBY VÝPOČTU NA ZÁKLADĚ SIMULINK MODELU

Tímto dílčím cílem je vytvořit metody, které by umožnily odhadnout dobu výpočtu daného Simulink modelu na cílovém zařízení. Jednak porovnáním výpočetních výkonů (benchmarkových srovnání) umožňujících „intuitivní odhad“ doby výpočtu a metod pro plně automatickou predikci doby výpočtu (použitím automatického hledání v benchmarkových skóre a vyhodnocením).

4 CEREBOT BLOCKSET

Cerebot blockset představuje skupinu nástrojů a programů, které umožňují na základě Simulink modelu vytvořit spustitelný program, který je automaticky nahrán do procesoru PIC32MX795F512L (součástí Cerebot hardwaru) po stisku jediného tlačítka v Simulink modelu.

4.1 SOUBORY CEREBOT BLOCKSETU

Celý proces od generování C kódu ze Simulink, překlad těchto souborů až po nahrání spustitelného kódu do FLASH paměti na čipu procesoru probíhá v několika krocích. Nejvýznamnější nástroje použité během tohoto procesu jsou popsané dále.

Program pro nahrání kódu do paměti

Je program napsaný v Javě a slouží pro nakopírování přeložených souborů do FLASH paměti. Tento program lze spustit z příkazové řádky a jako parametr se mu předá jméno binárního souboru obsahující kód, který se má nahrát do paměti procesoru. Samotné nahrání do FLASH paměti proběhne automaticky bez nutnosti dalších akcí. Pochopitelně Cerebot kit musí být během nahrávání kódu do paměti vestavěného procesoru připojený k počítači.

Šablony pro vytvoření main.c souboru

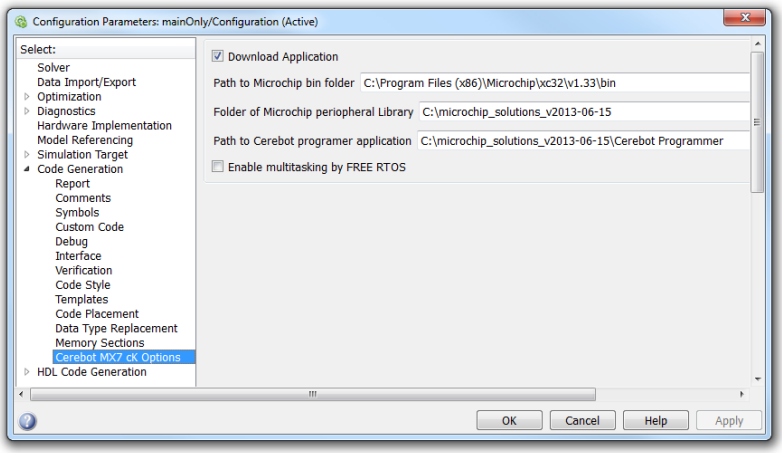
Jedná se o skupinu skriptů, která vygeneruje *main.c* soubory specifické pro určitý hardware. Mezi podporované módy simulace patří:

- Single tasking
- Kooperativní multitasking

- Preemptivní multitasking

Přičemž preemptivní multitasking je řízen pomocí FREE RTOS [16] operačního systému.

Konfigurační skript pro Simulink



Obrázek 3. Nastavení proměnných v modelu souvisejících s nastavením simulace

Tento soubor představuje vstupní bod celého blocksetu, jsou z něj připojené další vytvořené soubory a provedeno základní nastavení používaných proměnných Simulink. Kromě toho umožňuje vytvořit dialog pro nastavení uživatelských proměnných Cerebot blocksetu. Tento dialog je pro ilustraci na obrázku 3.

Make skript

Simulink svým tlc překladačem vygeneruje *.c* a *.h* soubory. Ty je dále potřeba přeložit. V závislosti na použitých funkcích v Simulink modelu je většinou nutné zkompilovat i unity v knihovnách (nebo přiložit jejich *.obj* soubory), tedy správně nastavit parametry překladače a linkeru. Přiložení všech potřebných knihoven v závislosti na generovaném kódu je jednou z hlavních funkcí vytvářeného make skriptu.

Řízení sekvence překladu

Během kompilace je potřeba ve správném pořadí použít nástroje provádějící překlad a transformace kódu. K tomu lze s výhodou použít *Callback funkce*, které umožňují během různých částí překladu volat externí funkce. Kromě toho je potřeba spustit make skript. Ten je vhodné spustit z funkce make, která se vykonává automaticky po dokončení generování *.c* a *.h* souborů.

4.2 BLOKY BLOCKSETU

Samotný Cerebot blockset obsahuje několik bloků, které umožňují použít základní periferie umístěné na desce Cerebot MX7 cK hardware. Každý takový blok se skládá z masky bloku, mex a tlc funkce.

MEX funkce

MEX funkce je ve své podstatě zkompilovaná S-funkce, která definuje, jak se bude daný blok chovat během simulace modelu a jaké bude mít vlastnosti (například typ a počet vstupních signálů, jejich vzorkovací perioda). Pokud vytváříme blok pro práci s periferiemi, nemusíme vždy implementovat složitou funkcionalitu pro potřeby simulace. Například pokud vytváříme blok pro práci s výstupním portem, Simulink během simulace obsahující tento blok nemusí na jeho místě provádět žádný výpočet (se signálem, který do tohoto bloku přijde během simulace, nic nedělá). To může významně zrychlit a usnadnit vývoj velké skupiny bloků.

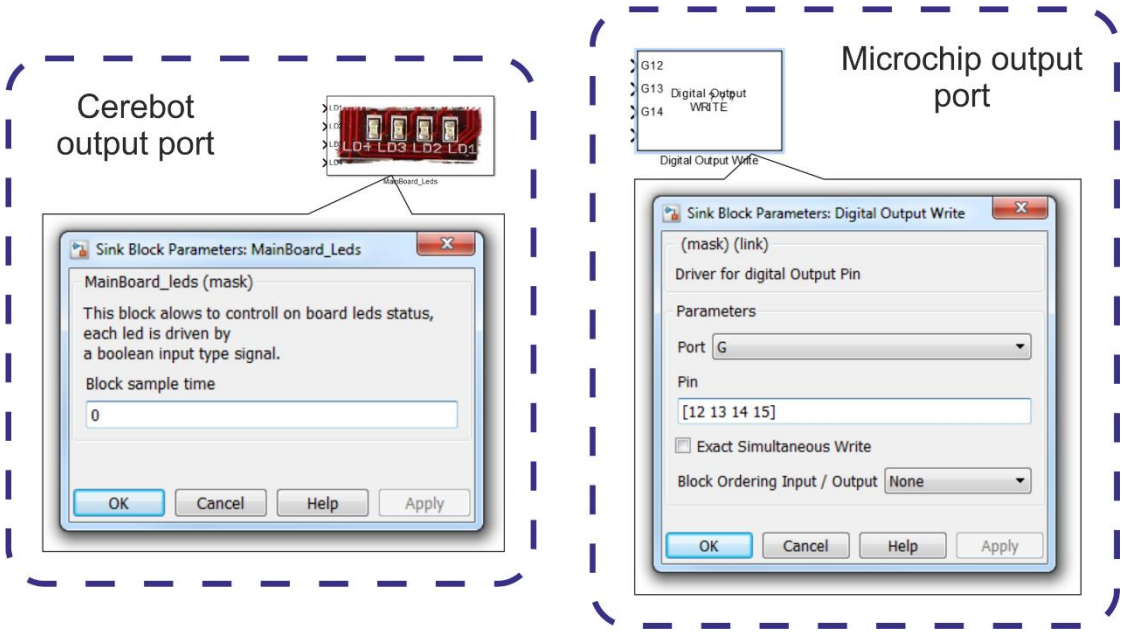
TLC funkce

TLC funkce obsahuje informaci o tom, jaký kód se má umístit na místo daného bloku při generování kódu. Jak lze z názvu těchto souborů tušit, používají tlc programovací jazyk, což je jazyk, který je optimalizován pro formátování textu (ale stejně pohodlně se v něm píší i funkce a výrazy). Jazyk TLC je popsán v dokumentaci, která je součástí distribuce Matlabu.

Simulink blok

Simulink blok je v zásadě blok S-funkce, kterému je přiřazena funkce popisující chování bloku (v *.mex* souboru) a jeho maska, která slouží pro změnu grafické podoby bloku a vytvoření uživatelského rozhraní pro zadávání parametrů bloku.

Periferie nevyžadující složitou konfiguraci



Obrázek 4. Blok pro ovládání led diod a jeho uživatelské rozhraní

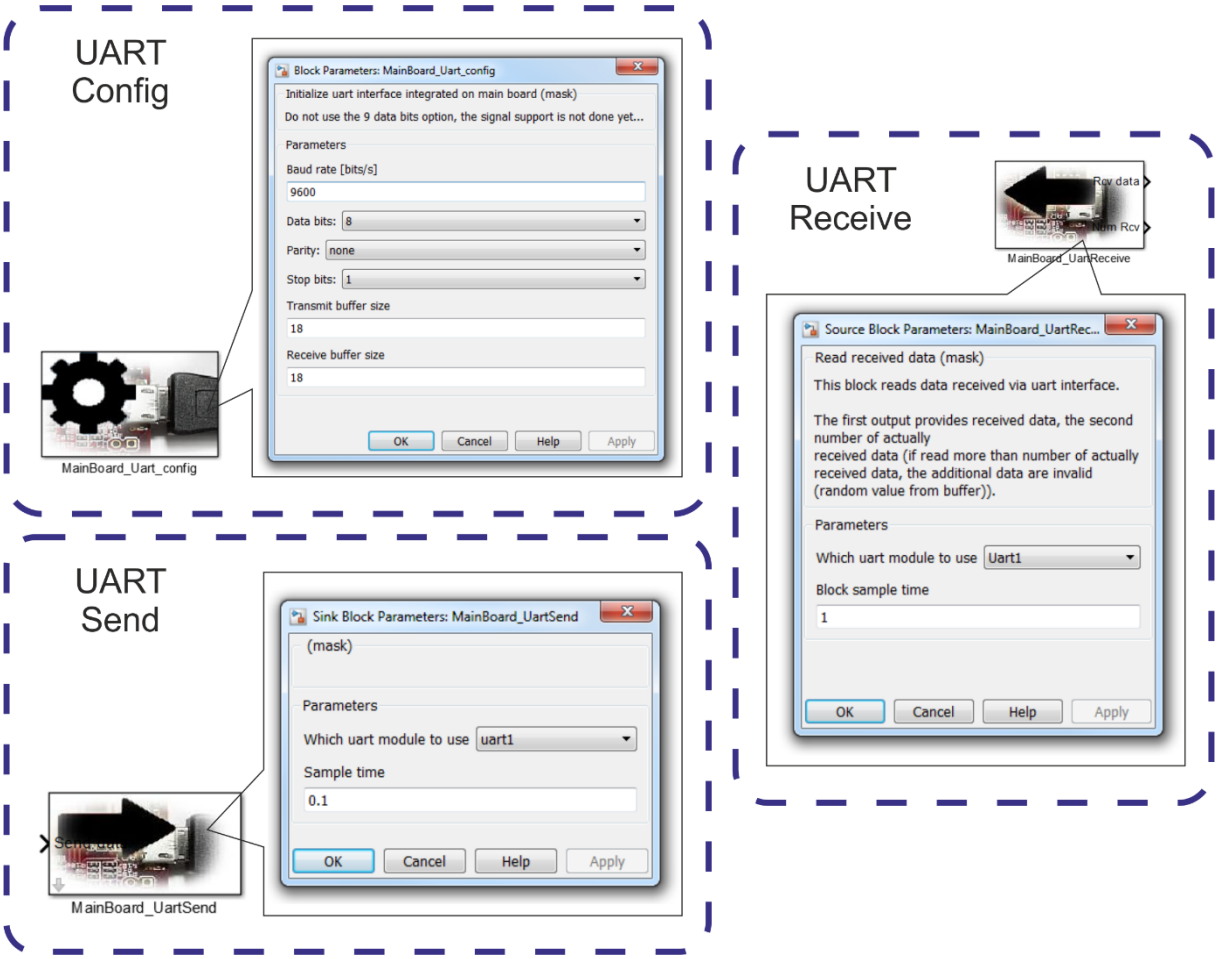
Pro implementaci funkcionality jednoduchých periferií lze vytvořit bloky, které nevyžadují téměř žádnou konfiguraci od uživatele, takovým příkladem jsou

například periferie používající digitální porty. Na obrázku 4 je pro srovnání uveden blok pro obsluhu led diod, který je součástí Cerebot blocksetu a blok, který je součástí jiného softwarového řešení (Microchip blocksetu [15], který je velmi podobný i Kerhuelově blocksetu [14]).

Jak je patrné blockset vytvořený pro platformu Cerebot je daleko názornější a jednodušší na použití, jelikož lze použít blok s mnohem menším množstvím nastavitelných parametrů, který je navíc graficky přehlednější.

Periferie vyžadující složitou konfiguraci

Periferie vyžadující složitou konfiguraci by bylo obtížné nakonfigurovat a obsluhovat pouze jedním blokem. Proto je výhodné rozdělit bloky pracující s touto periferií do několika skupin. Pokud by byl například pouze jeden blok provádějící čtení i zápis z UART periferie, muselo by čtení i zápis proběhnout vždy stejně často. Rozdělení do více bloků umožní provádět čtení i zápis z této periferie s různými frekvencemi. Bloky Cerebot blocksetu použité pro obsluhu UART periferie jsou pro ilustraci na obrázku 5.



Obrázek 5. Bloky pro obsluhu periferie UART a jejich konfigurační dialogy

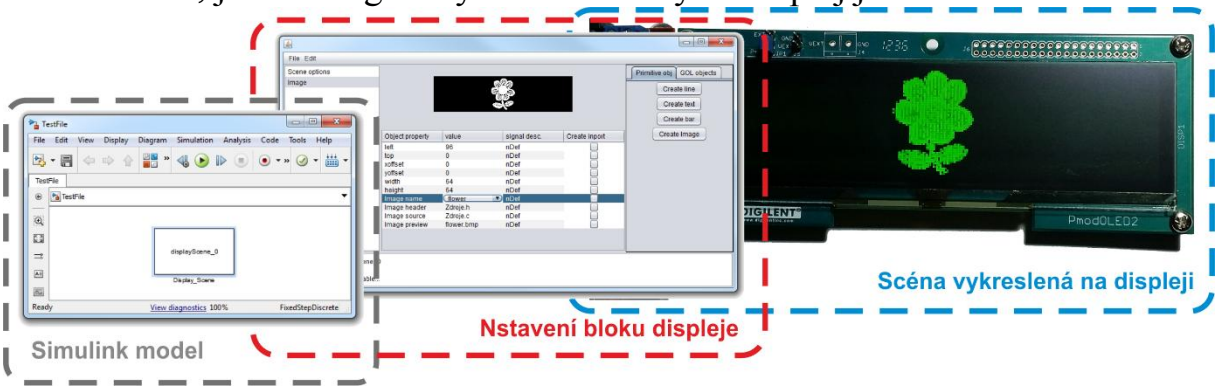
Knihovna bloků

Vytvořený soubor *target.tlc* stačí umístit do cesty, kterou prohledává Matlab, Matlab jej pak najde a automaticky přidá do nabídky dostupných target souborů pro generování kódu. Oproti tomu model, ve kterém jsou umístěné bloky, musí být zaregistrován. K tomu lze použít připravenou funkci *slblocks.m*, kterou je ale nutné upravit tak, aby se správně zobrazily vytvořené bloky v Simulink knihově.

5 BLOK PRO OBSLUHU KOMPLEXNÍ PERIFERIE

Pokud je sestavován blok pro obsluhu velmi komplexní periferie, jakou je například grafický displej. Je potřeba konfigurovat jednotlivé zobrazovací prvky a zadávat velké množství parametrů, které se navíc mění v závislosti na aktuálním rozložení scény displeje. Právě limitované možnosti nástroje pro vytváření uživatelských dialogů, který je součástí prostředí Simulink, představují problém při vývoji komplexního uživatelského rozhraní Simulink bloků. Tento problém byl vyřešen vytvořením samostatného programu, který mimo jiné vytváří uživatelské rozhraní pro zadávání parametrů a konfiguraci scény pro zobrazení na displeji.

Vytvořený nástroj funguje tak, že jedné scéně (rozložení grafických prvků na displeji) odpovídá jeden Simulink blok. Pokud tento blok umístíme do Simulink modelu, scéna se v daném simulačním kroku vykreslí na displej zařízení. Schéma Simulink bloku, jeho dialogu a vykreslené scény na displej je na obrázku 6.



Obrázek 6. Schéma vytvořeného bloku a rozhraní a vykreslené scény

Celý program pro implementaci bloku pro grafický displej zajišťuje poměrně komplexní funkcionalitu, která mimo jiné provádí:

- Propojení funkcí generovaných z Matlab/Simulink s knihovnou Microchipu (zejména funkce pro práci s grafickými objekty)
- Konfigurace a aktualizace proměnných (respektive datových struktur) pro předávání dat mezi funkcemi grafické knihovny a funkcemi generovanými Simulinkem.
- Vygenerování unit, které obsahují funkce potřebné při sestavování kódu a případně modifikace nastavení linker skriptu.

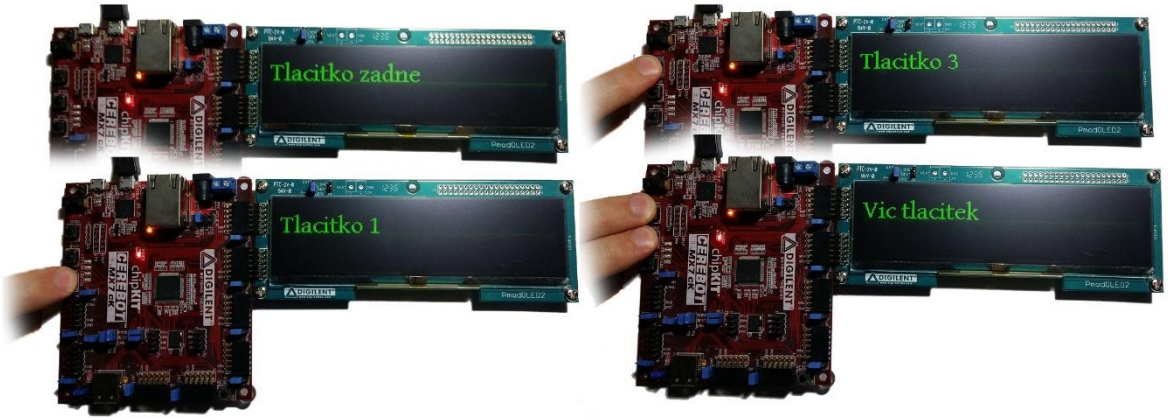
- Vytvoření uživatelského rozhraní pro konfiguraci bloku.
- Volby uživatelského rozhraní pro spuštění dalších programů (součástí distribuce Microchip grafických zdrojů) umožňujících editovat zobrazitelné objekty (například umožňují převod bitmapy do *hex* formátu, editaci fontů).
- Vytvoření odpovídajících souborů pro Simulink (MEX a TLC soubory) a jejich přeložení, je-li potřeba aktualizovat blok – například pokud se změní počet jeho vstupů.
- Uložení parametrů scény tak, aby ji bylo možné případně znovu otevřít a editovat.

5.1 DEMONSTRACE POUŽITÍ

Blok pro obsluhu displeje je plně integrován s Cerebot blokcksetem. Samotné vytvoření programu s použitím Cerebot blocksetu a bloku pro obsluhu grafického displeje je velmi jednoduché, stačí provést tyto kroky:

- Správně zapojit hardware (připojení Cerebot modulu k spočítači (usb kabel) a displeje k základnímu Cerebot modulu).
- Nakonfigurovat Simulink model (zvolení správného target tlc souboru a nakonfigurování simulace pro generování kódu)
- Umístit blok pro vykreslení scény na displej do Simulink modelu a nakonfigurovat jej, případně přidat kód měnící vlastnosti zobrazených grafických prvků

Příklad programu informujícím o stisknutém tlačítku je na obrázku 7.



Obrázek 7. Jednoduchý program kompletně vytvořený v Simulink

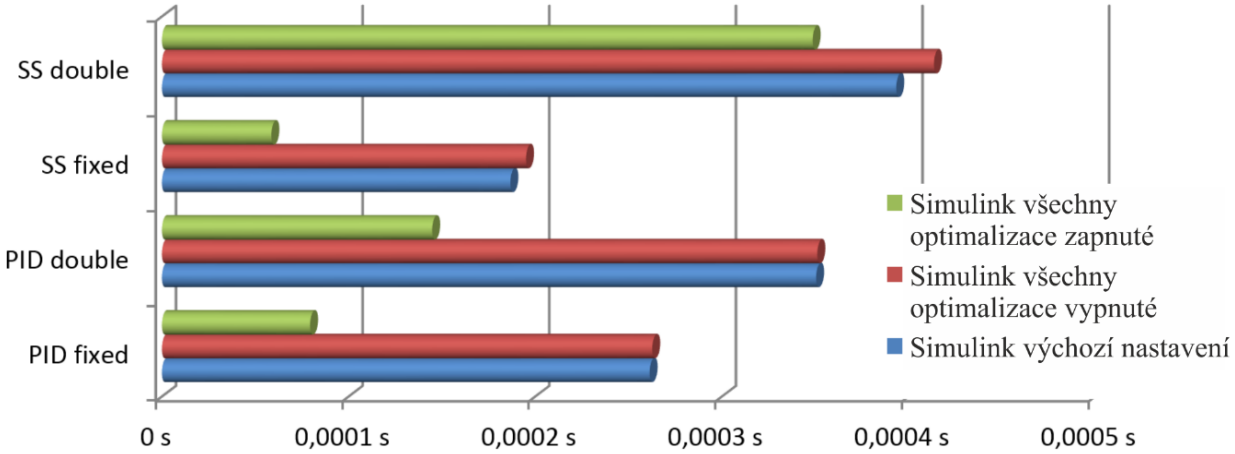
6 PREDIKCE VÝPOČETNÍHO VÝKONU

Metody prezentované v této části práce lze rozdělit na metody, které poskytují informace k lepšímu intuitivnímu odhadu doby výpočtu a plně automatické metody pro odhad doby výpočtu.

6.1 INTUITIVNÍ ODHAD DOBY VÝPOČTU

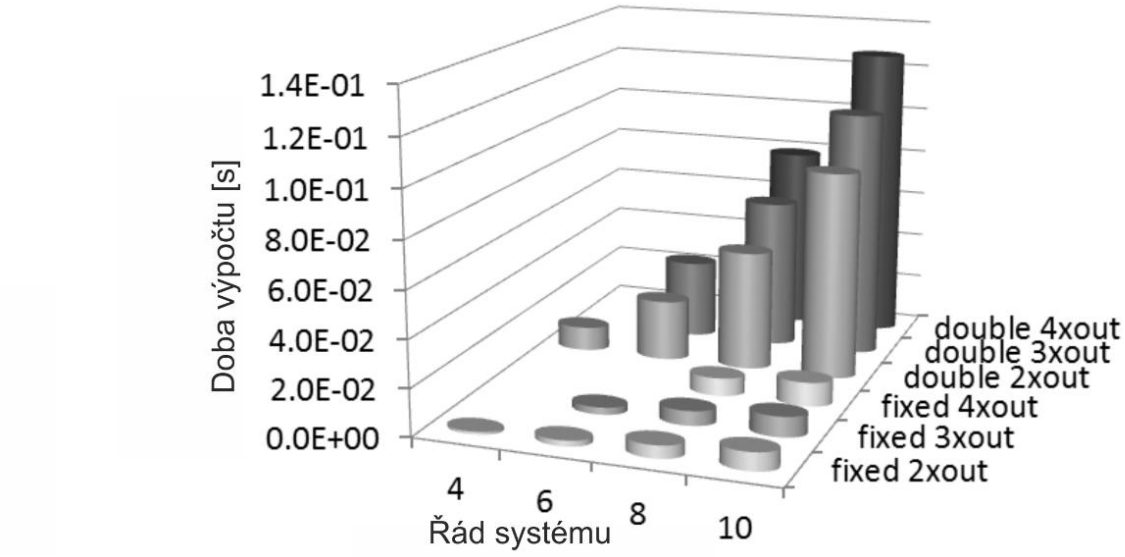
Jak je patrné z předchozího textu, překlad Simulink modelu do spustitelného kódu je poměrně komplexní proces. Výsledný kód tedy ovlivňuje celá řada parametrů a nastavení použitých nástrojů.

Následující graf na obrázku 8 demonstruje, jak se změní doby výpočtu výsledného algoritmu vygenerovaného ze Simulink při použití různé implementační aritmetiky a optimalizačních parametrů. V tomto grafu jsou znázorněny doby výpočtu různých regulátorů implementovaných v Simulink, konkrétně SS reprezentuje stavový regulátor a PID regulátor typu PID. Double označuje použití aritmetiky s plovoucí desetinnou čárkou a fixed použití celočíselné. Doba výpočtu regulačních algoritmů byla provedena na procesoru dsPIC33FJ128MC804 s taktem 20MHz.



Obrázek 8. Doba výpočtu v závislosti na použitých optimalizačních parametrech

Poměrně dobrý odhad doby výpočtu je možné získat na základě analogie. Známe-li dobu výpočtu určitého algoritmu, můžeme dobu podobného algoritmu odhadnout na základě tédo doby. Příkladem grafu, který zobrazuje dobu výpočtu stavového regulátoru (běžícího na procesoru 33fJ256GP710), v závislosti na typu regulované soustavy je prezentována na obrázku 9. Označení *out* znamená množství výstupních signálů z dané soustavy, fixed a double pak použité datové typy v modelu.



Obrázek 9. Doba výpočtu v závislosti na typu regulované soustavy

6.2 AUTOMATICKÝ ODHAD DOBY VÝPOČTU

Předešlé metody vyžadovaly od uživatele provedení odhadu doby výpočtu na základě grafů. Abychom odstranili nutnost ruční analýzy dat, byla vytvořena metoda provádějící odhad doby výpočtu plně automaticky.

Princip metody

Na základě Simulink modelu je sestaven model doby výpočtu algoritmu, který je přímo odvozen od Simulink modelu použitého pro generování kódu. Podobný princip používá celá řada metod například metody založené na identifikaci fragmentů (sekvencí) kódů [17], [18], [19] nebo výkonové modely mající charakter síťové struktury [20] případně jejich modifikace[21], [22] .

Samotná doba výpočtu je určena jako doba výpočtu jednotlivých bloků a doba výpočtu časovače simulace (1), kde *t_bl* reprezentuje dobu výpočtu jednoho Simulink bloku a *t_sch* dobu výpočtu časovače simulace. V tomto vztahu je doba časování simulace modelována konstantou a lineární závislostí na počtu bloků.

$$t_{jeden\_krok} = \sum_{i=1}^{bloky} t\_bl_i + t\_sch \qquad (1)$$

Jednotlivé individuální bloky *t_bl* jsou Simulink bloky, které mají unikátní vlastnosti, konkrétně mají stejný:

- Typ bloku
- Velikost vstupních a výstupních portů, jejich počet a rozměr
- Datový typ vstupních a výstupních signálů
- Případné parametry bloku

Pokud rozepíšeme dobu výpočtu časovače, můžeme celou rovnici (1) převést do maticového tvaru (2).

$$\begin{bmatrix} n\_blck_{1,1} & n\_blck_{1,2} & \dots & n\_blck_{1,n} & 1 & sum\_blck_1 \\ n\_blck_{2,1} & n\_blck_{2,2} & \dots & n\_blck_{2,n} & 1 & sum\_blck_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ n\_blck_{m,1} & n\_blck_{m,2} & \dots & n\_blck_{m,n} & 1 & sum\_blck_m \end{bmatrix} \cdot \begin{bmatrix} t\_blck_1 \\ t\_blck_2 \\ \dots \\ t\_blck_n \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} t\_sim_1 \\ t\_sim_2 \\ \dots \\ t\_sim_m \end{bmatrix} \quad (2)$$

V tomto vyjádření, proměnná *n_blck* reprezentuje počet bloků daného typu v modelu. Poslední dva sloupce v matici na levé straně (tvořené jedničkami a *sum_blck* proměnnou, reprezentující počet bloků v daném modelu) jsou použité k odhadu doby potřebné pro časování simulace. V matici na pravé straně jsou změřené doby výpočtů jednotlivých Simulink modelů. Takže prostřední matice po vyřešení soustavy (sestávající z *t_blck* proměnných) bude obsahovat doby výpočtů jednotlivých Simulink bloků a c_1 respektive c_2 konstanty budou neznámé konstanty pro určení doby výpočtu časovače simulace.

Matematický problém pro určení dob výpočtu jednotlivých bloků tak jak jej popisuje soustava rovnic (2) lze vyjádřit následujícím způsobem (3).

$$\arg\min_{t\_blck_j, c_1, c_2 \in \langle 0, +\infty)} \sum_i \left(\sum_j \left(n\_blck_{i,j} \cdot t\_blck_j \right) + c_1 + c_2 \cdot sum\_blck_i - t\_sim_i \right) \quad \textbf{Zo}$$

Abychom tuto úlohu mohli efektivně řešit, použijeme algoritmy z knihovny Matlabu. Ty ve skutečnosti minimalizují druhou mocninu tohoto problému. Nicméně, umocnění výrazu příliš neovlivní nalezené řešení, ale zásadním způsobem zjednoduší výpočet. Vybrané funkce, které jsou součástí standardní knihovny Matlabu, vhodné pro řešení lineárních rovnic s omezením na nalezené řešení jsou funkce: *lsqlin()*, *lsqnonneg()*, případně obecný řešič *fminsearch()*.

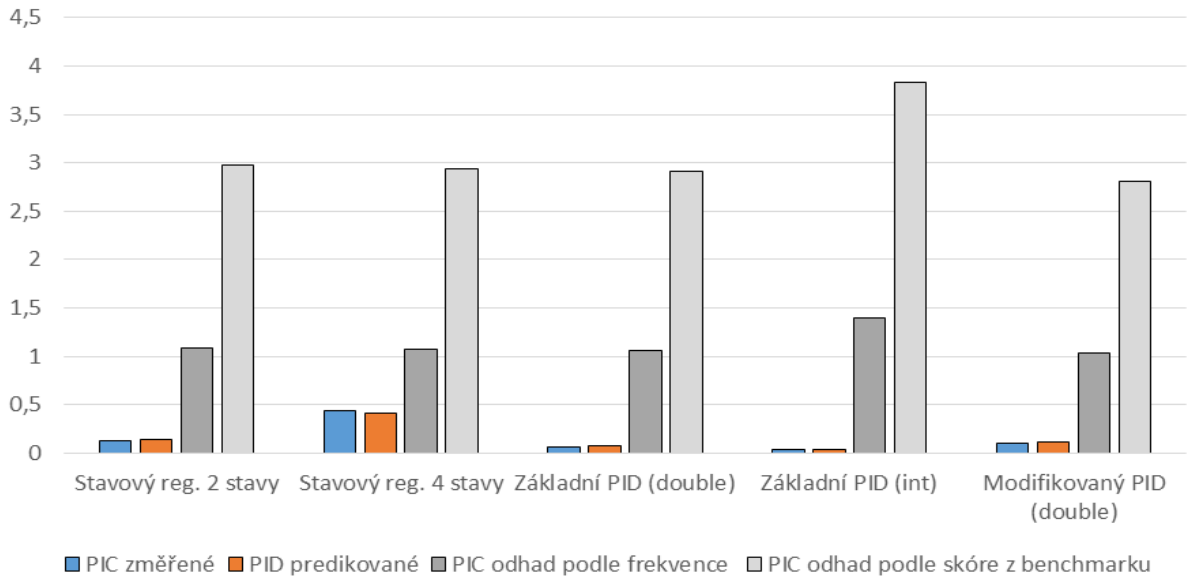
Demonstrační úlohy

Pro demonstrační účely byla vytvořena databáze 2000 modelů, které obsahují 400 bloků. Tyto demonstrační modely byly sestaveny použitím základních bloků Simulik knihovny a nakonfigurovány vybranými typy náhodně generovaných parametrů, přičemž se dodržují následující pravidla při sestavování modelu:

- Každý blok musí být úplně připojený (všechny jeho vstupy a výstupy).
- V každém kroku při sestavování nového modelu je náhodně vybrán blok z databáze bloků a nakonfigurován (náhodně) vybráním z předdefinovaných možností.
- Celý model se vytváří v sloupcích, přičemž bloky z každého dalšího sloupce preferují připojení k předchozímu.
- Propojení jednotlivých bloků by mělo být vyrovnané (signál vystupující z jednoho bloku by neměl vézt do příliš mnoha dalších bloků).

Souhrnný graf na obrázku 10 ukazuje srovnání přesnosti odhadu doby výpočtu pomocí automatické predikce na základě velkého množství benchmark modelů a intuitivního odhadu na základě jednoho benchmark skóre [23], případně charakteristiky procesoru (jeho frekvence). Kde použitými zařízeními jsou:

- ✓ **PC**: Intel Core 2 Quard Q6600 @ 2.39Ghz – (±2330 MIPS/core): CoreMark/Core 9138.00
- ✓ **PIC**: PIC32MX795F512L @ 80Mhz – (105DMIPS): CoreMark/Core 111.74



Obrázek 10. Srovnání přesnosti predikce doby výpočtu použitím různých metod

7 SHRNUTÍ VÝSLEDKŮ A ZÁVĚR PRÁCE

Cílem této práce bylo splnit především dva hlavní cíle. Prvním je vytvoření nástroje, který by umožnil automaticky generovat kód pro zařízení používající displej. Druhým cílem bylo vytvořit metody pro odhad doby výpočtu Simulink modelu na cílovém zařízení. Oba tyto cíle byly splněny. Vytvořené nástroje a metody svojí funkcí dosahují vlastností požadovaných v zadání a jejich parametry jsou výrazně lepší než vlastnosti dnes dostupných nástrojů (jejich kombinaci), které by bylo možné použít jako alternativu k námi vytvořeným.

Konkrétněji jsou dosažené výsledky v jednotlivých kategoriích popsány dále.

7.1 VYTVOŘENÍ BLOCKSETU PRO PLATFORMU CEREBOT MX7 CK

Vytvořený „Cerebot blockset“ je skupina nástrojů, která umožňuje na základě Simulink modelu plně automaticky vygenerovat spustitelný kód a ten nahrát do vestavěného procesoru zařízení Cerebot MX7 cK.

Kapitoly popisující vývoj blocksetu poskytují detailní popis konceptů použitých při vývoji jednotlivých komponent tohoto nástroje a vazeb mezi nimi. Použité metody pro vytvoření kompletního řetězce nástrojů pro automatické generování kódu z prostředí Simulink nejsou srovnatelným způsobem popsány v dostupných materiálech na jednom místě. To může velmi usnadnit práci lidem, kteří potřebují získat přehled o problematice automatického generování kódu z prostředí Simulink, jelikož přináší ucelený a detailní popis mechanizmů a konceptů používanými při generování kódu.

Navíc jsou prezentované metody vždy doplněné příkladem s jejich implementací, čímž je demonstrována funkčnost jejich konceptu a ilustrován způsob použití. Díky detailnímu popisu implementace jednotlivých konceptů lze použít představený postup jako návod k vytvoření obdobné funkcionality pro další nepodporovaný cílový hardware (vytvoření blocksetu pro jiný, nepodporovaný hardware).

Výsledky v oblasti generování kódu pro Cerebot MX7 cK hardware byly publikované v [6a], kromě toho vyšlo několik článků, které obecněji popisují možnosti generování kódu pro specifický hardware [3a], zohledňující specifické požadavky na vzhled generovaného kódu [5a].

7.2 VYTVOŘENÍ BLOKU PRO OBSLUHU KOMPLEXNÍ PERIFERIE

Jedním z hlavních cílů této práce bylo vytvořit blok pro obsluhu komplexní periferie. Konkrétně blok, který by umožnil automaticky generovat kód pro ovládání displeje připojeného jako periferní zařízení k vestavěnému procesoru.

Vytvořit nástroj, který by splnil požadavky zadání, není možné s použitím standardních nástrojů programu Matlab/Simulink. Byla proto vyvinuta nová metoda pro implementaci požadované funkcionality. Tato metoda spočívá ve vytvoření samostatného programu, který poskytuje uživatelské rozhraní a generuje soubory implementující funkcionalitu pro automatické generování kódu pro Simulink prostředí. Vytvořený nástroj je dobře popsán z uživatelského i programátorského hlediska. Díky tomu je možné nově představenou metodu modifikovat pro implementaci automatického generování kódu pro další složité periferie, vyžadující komplexní uživatelské rozhraní.

Představené řešení doposud nebylo použité v žádném dostupném blocksetu. Poprvé je tak možné použít Simulink pro generování kódu i pro složité periferie, což ani s použitím komerčně nabízených produktů nebylo takto komfortním způsobem doposud možné.

Postupy pro vytvoření bloku pro implementaci rozhraní komplexní periferie v Simulink byly publikovány v [7a].

7.3 VÝVOJ METOD PRO PREDIKCI DOBY VÝPOČTU

Druhým hlavním bodem této práce bylo vytvořit metody pro predikci doby výpočtu Simulink modelu na cílovém zařízení.

V rámci tohoto tématu bylo vytvořeno několik postupů, které umožňují tuto dobu odhadnout. Buď poskytují nástroje, které umožní dobu výpočtu odhadnout na

základě analogie (srovnání s benchmark skóre) nebo dobu výpočtu určí plně automaticky na základě Simulink modelu.

Dílčím výsledkem jsou sady benchmarkových modelů a jejich skóre (doby výpočtu), na základě nichž lze s požitím analogie získat odhad doby výpočtu určitého algoritmu. Jednotlivé benchmarky jsou zvolené tak, aby demonstrovaly výpočetní výkon vybraných cílových zařízení s nejběžnějšími typy algoritmů používanými v mechatronice. Tyto výsledky byly publikovány v [1a] a[4a].

Výsledky jednotlivých benchmarků pak umožňují získat mimo představy o výpočetním výkonu daného hardwaru, také povědomí o vlivu jednotlivých optimalizačních a implementačních parametrů na výslednou rychlost algoritmu. Tato znalost může být užitečná při vývoji a kompilaci nového algoritmu, tím, že identifikuje parametry, které mají největší vliv na výkon algoritmu. Shrnutí těchto výsledků bylo zveřejněno v [2a].

Druhý typ vytvořených metod pro predikci doby výpočtu představuje metoda, která predikuje dobu výpočtu plně automaticky na základě Simulink modelu. Predikce doby výpočtu probíhá na základě databáze dob výpočtů jednotlivých Simulink bloků, která je získána identifikací na základě dat z velkého počtu benchmarkových modelů. Tato nová metoda automatické predikce poskytuje v porovnání s intuitivním odhadem doby výpočtu na základě benchmark skóre daleko přesnější odhad. Kromě toho je i samotný odhad doby výpočtu velmi rychlý. Tato metoda je plně automatická a uživatel nemusí zadávat žádné parametry.

7.4 PŘÍNOSY PRÁCE

Nejvýznamnější praktické přínosy práce jsou:

- *Cerebot blockset* – nástroj umožňující automaticky vygenerovat spustitelný kód pro Cerebot MX7 cK hardware na základě Simulink modelu. Výsledky jsou publikované v [7a].
- *Simulink blok pro obsluhu komplexní periferie* – Program těsně integrovaný v programu Simulink slouží pro konfiguraci a vytvoření šablon pro generování kódu pro grafický displej. Tento nástroj byl prezentován v [6a].

Nejvýznamnější teoretické přínosy práce jsou:

- *Specializované benchmarky* – Vytvořené sady tabulek a grafů umožňují získat představu o výpočetním výkonu hardware a efektivitě generovaného kódu v závilosti na nastavení parametrů překladače. Výsledky byly publikované v [1a], [2a] a [4a].
- *Automatická predikce doby výpočtu* – Vytvořený nástroj umožňuje použitím „modelu výpočetního výkonu“ sestaveného na základě Simulink modelu automaticky predikovat dobu výpočtu na cílovém zařízení. Výsledky jsou připravovány na publikaci v časopise.

8 VYBRANÉ PUBLIKACE AUTORA

[1a] LAMBERSKÝ, V.; VEJLUPEK, J. Performance of dsPIC controller programmed with code generated from Simulink. In *Mechatronics, Recent Technological and Scientific Advances.* 2011. s. 105-113. ISBN: 978-3-642-23243- 5.

[2a] LAMBERSKÝ, V.; VEJLUPEK, J. BENCHMARKING THE PERFORMANCE OF A DSPIC CONTROLLER PROGRAMED WITH AUTOMATICALLY GENERATED CODE. In *Technical Computing Prague 2011*. Praha: ICT Prague Press, 2011. s. 75-75. ISBN: 978-80-7080-794- 1.

[3a] LAMBERSKÝ, V. Model based design and automated code generation from Simulink targeted for TMS570 MCU. In *Education and Research Conference (EDERC), 2012 5th European DSP*. 2012. s. 225-228. ISBN: 978-1-4673-4595- 8.

[4a] LAMBERSKÝ, V.; GREPL, R. Benchmarking various rapid control prototyping targets supported in Matlab/ Simulink development environment. In *Mechatronics 2013: Recent Technological and Scientific Advances.* Mechatronics. Cham, Heidelberg, New York, Dordrecht, London: Springer International Publishing, 2013. s. 669-675. ISBN: 978-3-319-02293- 2.

[5a] LAMBERSKÝ, V.; KRIŽAN, J.; ANDREEV, A. Generating Code Consistent with Simulink Simulation for Aperiodic Execution on a Target Hardware Powered by a Free RTOS. In *Mechatronics 2013: Recent Technological and Scientific Advances.* Mechatronics. Cham, Heidelberg, New York, Dordrecht, London: Springer, 2013. s. 95-101. ISBN: 978-3-319-02293- 2.

[6a] LAMBERSKÝ, V.; VEJLUPEK, J.; GREPL, R.; SOVA, V. Development of Simulink blockset for embedded system with complex peripherals. In *COMMUNICATIONS, CIRCUITS and EDUCATIONAL TECHNOLOGIES Proceedings of the 2014 International Conference on Electronics and Communication Systems II (ECS '14).* Prague, Czech Republic: europment, 2014. s. 112-117. ISBN: 978-1-61804-231- 6.

[7a] LAMBERSKÝ, V.; VEJLUPEK, J.; SOVA, V.; GREPL, R. Creating support for fully automatic code generation for Cerebot MX7cK hardware from Simulink environment. *International Journal of Circuits Systems and Signal Processing,* 2014, roč. 2014, č. 8, s. 536-544. ISSN: 1998- 4464.

9 SEZNAM POUŽITÉ LITERATURY

[1] P.G. Paulin, C. Liem, M. Cornero, F. Nacabal, G. Goossens, "Embedded software in real-time signal processing systems: application and architecture trends," Proceedings of the IEEE , vol.85, no.3, pp.419,435, Mar 1997, doi: 10.1109/5.558716

[2] Woon-Seng Gan, Yong-Kim Chong, W. Gong, and Wei-Tong Tan. 2000. Rapid prototyping system for teaching real-time digital signal processing. IEEE Trans. on Educ. 43, 1 (February 2000), 19-24.

[3] R. Grepl, V. Lamberský, J. Vejlupek, M. Jasanský, F. Vadlejch, P. Čoupek, Development of 4WS/4WD Experimental Vehicle: platform for research and education in mechatronics. In ICM 2011, IEEE International Conference on Mechatronics. 2011. s. 893-898. ISBN: 978-1-61284-982- 9.

[4] M. Kuhl, C. Reichmann, I. Protel, K.D. Muller-Glaser, "From object-oriented modeling to code generation for rapid prototyping of embedded electronic systems," Rapid System Prototyping, 2002. Proceedings. 13th IEEE International Workshop on , vol., no., pp.108,114, 2002.

[5] Ã. Netland and A. Skavhaug, "Software Module Real-Time Target: Improving Development of Embedded Control System by Including Simulink Generated Code into Existing Code, " in Software Engineering and Advanced Applications, 39th Euromicro Conference on, 2013.

[6] Intel Hexadecimal Object File Forma Specification Revision A, 1/6/88 [ONLINE] Available at: http://www.interlog.com/~speff/usefulinfo/Hexfrmt.pdf. [Accessed 27 May 2013].

[7] P.G. Paulin, C. Liem, M. Cornero, F. Nacabal, G. Goossens, "Embedded software in real-time signal processing systems: application and architecture trends," Proceedings of the IEEE , vol.85, no.3, pp.419,435, Mar 1997, doi: 10.1109/5.558716

[8] M. Guthaus, et al., "MiBench: A free, commercially representative embedded benchmark suite,"In IEEE International Workshop on Workload Characterization, pp. 3-14, 2001.

[9] EEMBC -- The Embedded Microprocessor Benchmark Consortium. [ONLINE] Available at: https://www.eembc.org/. [Accessed 10 Juli 2014].

[10] N. Topham and D. Jones. High speed CPU simulation using JIT binary translation. In Proceedings of MoBS -- Workshop on Modeling, Benchmarking and Simulation, 2007.

[11] Björn Franke. Statistical Performance Modeling in Functional Instruction Set Simulators. ACM Trans. Embed. Comput. Syst. 11S, 1, Article 22 (June 2012), 22 pages. 2012. DOI=10.1145/2180887.2180899

[12] Björn Franke. Fast cycle-approximate instruction set simulation. In Proceedings of the 11th international workshop on Software & compilers for embedded systems (SCOPES '08). ACM, New York, NY, USA, 69-78. 2008.

[13] G. Bontempi and W. Kruijtzer. A Data Analysis Method for Software Performance Prediction. In Proceedings of the conference on Design, automation and test in Europe (DATE '02). IEEE Computer Society, Washington, DC, USA. 2002

[14] Lubin KERHUEL's personnal website contributors, 'Simulink - Embedded Target for PIC', Lubin KERHUEL's personnal website, http://www.kerhuel.eu/wiki/Simulink_-_Embedded_Target_for_PIC [accessed 19 August 2014]

[15] MPLAB 16-Bit Device Blocks for Simulink, [online] Available at: http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=SW007023 [accessed 19 August 2014]

[16] R. BARRY, Using the FreeRTOS™ Real Time Kernel: A Practical Guide. 1. vyd. 2010. ISBN 9781446169148.

[17] G. Bontempi, W. Kruijtzer, A data analysis method for software performance prediction, in Design, Automation and Test in Europe Conference and Exhibition, Proceedings, pp. 971-976, 2002.

[18] H. Gomaa and D.A. Menasce´, "Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures," ACM Proc. Int'l Workshop Software and Performance, pp. 117-126, 2000.

[19] H. Gomaa and D. Menasce´, "Performance Engineering of Component-Based Distributed Software Systems," Performance Eng., R. Dumke et al., eds. pp. 40-55, 2001.

[20] C.U. Smith, Performance Engineering of Software Systems. Addison Wesley, 1990.

[21] L. Kleinrock, Queueing Systems, Volume 1: Theory. Wiley, 1975.

[22] K.S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications. John Wiley and Sons, 2001.

[23] Shay Gal-On and Markus Levy, Creating portable, repeatable, realistic benchmarks for embedded systems and the challenges thereof. SIGPLAN Not. 47, pp. 149-152. 2012.

10 AUTOROVO CV

Osobní údaje

Jméno a příjmení: Vojtěch Lamberský
Datum narození: 27. 5. 1985
Stav: Svobodný
Adresa: Českých bratří 978,
563 01 Lanškroun (CZ)
Telefon: +420 776166296
Email: vojtech@lambersky.cz

Dosažené vzdělání

VUT Brno, 2010 – dosud
(Fakulta strojního inženýrství, Inženýrská mechanika, Doktorské studium)

VUT Brno, 2008 – 2010
(Fakulta strojního inženýrství, Mechatronika, Magisterské studium)

VUT Brno, 2005 – 2008
(Fakulta strojního inženýrství, Mechatronika, Bakalářské studium)

Gymnázium Lanškroun, 1997 – 2002
(Všeobecné čtyřleté gymnázium, střední škola)

Kurzy a pracovní stáže

AIESEC internship jako učitel – Abua (Nigeria) 2013, 1,5 měsíce
Chalmers University of Technology, Göteborg, (Švédsko), 2008, 1 semestr
Vedanta Aluminium limited, Jharsuguda (Indie) 2010, 2,5 měsíce
TU MAMI – Formula student (Rusko) 2010, 1,5 měsíce

Účast na mezinárodních konferencích

Europment 2014 (2.4.2014 – 4.4.2014), Praha (CZ)
Mechatronics 2013 (7. 10. 2013 – 9. 10. 2013), Brno (CZ)
EDERC2012 (11. 9. 2012 – 15. 9. 2012), Amsterodam (NL)
Technical Computing Prague 2011 (8. 11. 2011) Praha (CZ)
Mechatronics 2011 (21. 9. 2011 – 24. 9. 2011) Varšava (PL)

11 ABSTRAKT

Tato dizertační práce je zaměřena na vývoj nástrojů a metod umožňujících zrychlit vývoj softwaru pro vestavěné procesory používané v mechatronických aplikacích.

V úvodní části práce jsou představeny dnes používané softwarové a hardwarové nástroje pro rychlý vývoj nových aplikací. V této oblasti se práce zabývá dvěma hlavními tématy. První je vývoj nástroje pro automatické generování kódu z prostředí Simulink pro vestavěný procesor. Druhým tématem je pak vývoj nástrojů pro predikci doby výpočtu Simulink modelu na vestavěném procesoru.

Další část práce popisuje vývoj a vlastnosti Cerebot blocksetu, což je skupina nástrojů umožňující automaticky generovat kód z prostředí Simulink pro vestavěný procesor. Následující sekce popisuje metody pro predikci doby výpočtu na vestavěném procesoru na základě Simulink modelu.

Hlavní přínos práce spočívá ve vytvoření podpory pro automatické generování kódu pro platformu Cerebot MX7 cK a navíc umožňuje použít i komplexní periferii (grafický displej), což dnes dostupné řešení neumožňují. Dalším významným výsledkem je vytvořená metoda pro automatickou predikci doby výpočtu na základě Simulink modelu.

The main focus of this dissertation thesis is on methods and tools which can increase the speed of software development process for embedded processors used in mechatronics applications.

The first part of this work introduces software and hardware tools suitable for a rapid development and prototyping of new applications used today. This work focuses on two main topics from the mentioned application field. The first topic is a development of tools for an automatic code generation from the Simulink environment for an embedded processor. The second topic is a development of tools enabling execution time prediction based on a Simulink model.

Next chapter of this work describes various aspects and properties of the Cerebot blockset, which is a toolset for a fully automatic code generation from a Simulink environment for an embedded processor. Following chapter describes various methods that are suitable for predicting the execution time on an embedded processor based on a Simulink model.

Main contribution of this work presents the created support for a fully automatic code generation from a Simulink software for the MX7 cK hardware, which enables a code generation supporting also a complex peripheral (a graphic display unit). The next important contribution of this work presents the developed method for an automatic prediction of the software execution time based on a Simulink model.