Tvorba knihovny funkcí pro digitální signálový procesor TMS320C50

Kamil Zobaník

Bakalářská práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky
akademický rok: 2005/2006

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Kamil ZOBANÍK**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Automatické řízení a informatika**
Téma práce:	**Tvorba knihovny funkcí pro digitální signálový procesor TMS320C50**

Zásady pro vypracování:

1. Seznámení se signálovým procesorem TMS320C50 firmy Texas Instruments po stránce programování (připojení k PC, vývojové prostředí, instrukční soubor).

2. Napsat v jazyce symbolických adres knihovnu algoritmů celočíselného dělení, dělení v pohyblivé řádové čárce, násobení v pohyblivé řádové čárce a funkce sinus. Metody řešení algoritmů jsou věcí studenta, avšak vedoucí práce dá doporučení.

3. Zdrojové texty algoritmů musí být součástí bakalářské práce stejně jako testovací data pro ověření správné funkce.

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ŠEBESTA, J.: Mikroprocesorová technika, VUT, Brno, 2002
2. SMITH, S.W.: The scientist and engineers guide to digital signal processing, second edition, California Technical Publishing,1999
3. KUBA, M.: Architektura počítačů, Masarykova universita, Brno, 1995
4. MARVEN, C., G. EWERS: A simple approach to digital signal processing, Texas Instruments, 1995

Vedoucí bakalářské práce: **Ing. Tomáš Družbík**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **14. února 2006**

Termín odevzdání bakalářské práce: **16. června 2006**

Ve Zlíně dne 14. února 2006

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Bakalářská práce je zaměřena na tvorbu algoritmů. Praktická část je věnována vývoji matematických knihoven pro řadu procesorů TMS320C5x DSP od firmy Texas Instruments. Podle zadání jsem naprogramoval matematické funkce sinus, dělení v pevné a pohyblivé řádové čárce a násobení v pohyblivé řádové čárce. Tyto knihovny se mohou volně vkládat jako podprogramy do libovolného zdrojového kódu jako plnohodnotné funkce, které vracejí výsledek. Práce obsahuje seznámení se s vývojovým prostředím a způsobem připojení digitálního signálového procesoru k osobnímu počítači. Součástí bakalářské práce jsou zdrojové kódy.

Klíčová slova: DSP, digitální signálový procesor, sinus, aproximace, násobení, dělení

ABSTRACT

This thesis deals with aspects of digital signal processors utilization and highlight of its advantages that are determined by processor architecture. The practical part deals with the development of mathematical libraries aimed to a family of TMS320C5x digital signal processors by Texas Instruments. I chose mathematical functions sinus, floating point division and multiplication and fixed point division. These libraries can be included as subroutines into any source code as fully qualified functions that return a result. The paper contains an introduction to development environment and a connection of the digital signal processor with a personal computer. This thesis also includes source codes of designed algorithms.

Keywords: DSP, digital signal processor, sinus, approximation, multiplication, division

Děkuji vedoucímu mé bakalářské práce Ing. Tomáši Družbíkovi za odborné vedení, za materiálovou podporu, spolupráci při testování aplikací vyvinutých pro digitální signálový procesor a také za rady a připomínky během zpracování bakalářské práce.

Souhlasím s tím, že s výsledky mé bakalářské práce může být naloženo podle uvážení vedoucího bakalářské práce a vedoucího katedry. V případě publikace budu uvedený jako spoluautor.

Prohlašuji, že jsem na celé bakalářské práci pracoval samostatně a použitou literaturu jsem citoval.

Ve Zlíně dne 15.6.2006 .....

podpis

OBSAH

ÚVOD

S rostoucí nabídkou digitálních technologií se stalo využívání programovatelných zařízení frekventovanějším. Digitální zařízení pracují s digitalizovanými signály. Na některé operace ze signály jsou kladeny nároky na zpracování signálu v reálném čase. Pro zvýšení účinnosti záznamu dat a práce s daty se v současnosti využívá komprese dat, kódování a dekódování hlasu, filtrace šumu a modulace signálů. Všechny tyto technologie se neustále rozvíjejí a pracuje se na jejich zdokonalování. Nároky na vysokou rychlost výpočtů umožnily digitálním signálovým procesorům (dále jen DSP) otevřít místo na světovém trhu, které si i nadále pravděpodobně udrží. Mezi největší korporace výrobců DSP patří firmy Texas Instruments, Analog Devices a Freescale Semiconductor. Tyto firmy neustále vyvíjí nové řady DSP dle poptávky trhu. Tato práce se věnuje, s ohledem na zadání práce, pouze produktu firmy Texas Instruments, a to *TMS320C50*. Praktické využití procesorů na výpočty žádaných matematických operací je hlavním cílem této práce.

I. TEORETICKÁ ČÁST

1 DIGITÁLNÍ SIGNÁLOVÉ PROCESORY

Kapitola představuje stručný úvod do problematiky DSP, popis produktových řad signálových procesorů firmy Texas Instruments. Dále je věnována pozornost vlastnostem procesoru vybraného pro praktickou část, včetně charakteristiky vývojového modulu *TMS320C5x DSP Starter Kit*.

1.1 Použití a vlastnosti DSP

Mezi celou řadou DSP existují i takové, které se vyvíjejí pro specifické aplikace. Tyto DSP mají často jedinou specificky zaměřenou funkci, pro kterou byly vyvíjeny, aby se dosáhlo efektivnějšího výkonu za nižší cenu. Signálové procesory jsou vhodné, již podle jejich názvu ke zpracování digitálních signálů. Abychom tyto procesory využili pro měření teploty, napětí, rychlosti otáček, tlaku nebo jiných veličin, je nutné tyto analogové signály nejprve převést na digitální data. Některé DSP mohou také obsahovat analogové vstupy a výstupy. Výhodou DSP je možnost zpracování signálů v reálném čase. Pojmem reálný čas uvažujeme, že data, které do DSP vstupují musí být přímo úměrná zpracovaným datům na výstupu z DSP. Také matematické zpracování digitálních signálů a jejich případné vizuální úpravy se mohou okamžitě zpracovávat. Data je možno dle potřeby dále odesílat do dalších přídavných modulů k uložení, vizualizaci nebo dalším úpravám (např. PC nebo DSP).

Signálové procesory přispěly také k rozvoji mobilních komunikací. Přešlo se z klasických analogových modulačních metod bezdrátového přenosu signálu na sofistikovanější metody digitální modulace, která je efektivnější s mnohem lepší energetickou bilancí i lepším a úspornějším využíváním kmitočtového spektra. Další výhodou je vysoký výpočetní výkon, nízká spotřeba, univerzálnost aplikace, nízká cena v porovnání s některými technologiemi a předurčenost k matematickým aplikacím a zpracování digitálních signálů.[2]

Vysoký výkon zajišťuje specifická paralelní architektura těchto procesorů. Díky této architektuře je možné provést matematickou operaci násobení a sečtení (popřípadě i operaci bitového posunu, nepřímé adresování atd.) v jednom cyklu. Je také možné paralelně na nevyužitých jednotkách provádět i další operace souběžně, což přispívá ke zrychlení činnosti matematických výpočtů DSP. Je třeba dodat, že rychlost výpočtu určuje efektivní naprogramování zdrojového kódu aplikace. Některé DSP také nabízí hardwarové řešení operací s pohyblivou řádovou čárkou a můžou obsahovat aplikačně specializované koprocesory a výpočetní jednotky. Mimo výjimky mají DSP zvlášť umístěnou programovou a

zvlášť datovou paměťovou oblast, případně mohou být zvlášť umístěny i vstupy/výstupy (I/O). [5]

Vnitřní paměťové oblasti jsou rozděleny do několika samostatných bloků pro nezávislé přístupy. Urychluje to tak používání paměti plus možnost využití několika paměťových bloků současně v několika větvích programu. Mimo jiné výhody DSP nabízí bohaté adresovací možnosti, které mohou být při zpracování dat velkým přínosem. Také umožňují multiprocesorovou komunikaci, tedy shlukování procesorů do tzv. výpočetních sítí, které v souhrnu navyšují výpočetní výkon. Každý z procesorů ve výpočetní síti může mít svoji specifickou úlohu a výsledek předává dalšímu DSP v síti, který s ním dále pracuje. Nakonec nejeden programátor uvítá již hotové knihovny funkcí počínaje dělením, goniometrické funkce, funkce s pohyblivou řádovou čárkou (hlavně pro DSP, který není pohyblivou řádovou čárkou vybaven), a také konvolucí, rychlou Fourierovou transformací, algoritmy pro filtrování dat nebo algoritmy pro zpracování obrazu.

1.2 Dělení DSP firmy Texas Instruments

Nabídka firmy Texas Instruments pokrývá většinu uživatelských požadavků. Texas Instruments nabízí celou řadu více či méně univerzálních DSP. Každý vývojář má šanci vybrat si právě takový DSP, jaký pro svoji aplikaci potřebuje. Firma Texas Instruments rozděluje své procesory do několika rodin dle jejich vlastností. V rodinách jsou DSP rozdělovány na další podskupiny. Rozdíly v podskupinách nebývají zásadní.

Rozdělení DSP do základních skupin

Rodiny C1x, C2x a C2000 se vyznačují požadovanými periferiemi a bohatě dostačujícím výpočetním výkonem v jejich oboru aplikací, kterými jsou řízení hybných jednotek, frekvenční měniče, senzorické snímání a také jiné aplikace.[5]

Rodiny C5x a C5000 jsou optimalizované pro nízkou spotřebu energie a velmi vysoký výpočetní výkon. Procesory jsou předurčeny pro systémy napájené mimo elektrickou síť bateriemi. Tyto DSP jsou tedy vhodné pro osobní komunikátory, multimediální zařízení a také jsou vhodné pro zabezpečovací ústředny se záložním zdrojem energie.[5]

Rodiny C3x, C4x, C6x/C6000 a C8x se vyznačují opět vysokým výpočetním výkonem, ale navíc jsou opatřeny technologií, která umožňuje najednou vykonávat několik operací. Je-

jich struktura je vysoce paralelizovaná. Tato řada je předurčena pro své široké matematické možnosti pro výpočetně náročné a standardně zdlouhavé aplikace.[5]

Řady jsou uspořádány vzestupně podle jejich výkonnosti. Od C1x až po C8x, avšak opravdu vynikajícím výpočetním výkonem vyniká řada C6x nebo jinak zvaná C6000. Pak nejmenší rychlostí výpočtů je známa řada C1x, která však dostačuje k jí určeným aplikacím. K jejich ohodnocení se však musí připočítat využití integrovaných koprocesorů a specializovaných hardwarových výpočetních jednotek.[5]

1.3 Charakteristika rodiny C5x a C5000

C5x je rodina DSP C51, C52, C53, C53S, C56, C57, C57S a dvou skupin C54x a C55x. Firma Texas Instruments označuje skupiny C54x a C55x také jako rodinu C5000. C5000 má velmi obsáhlé zastoupení v nabídce DSP. Tato rodina procesorů podporuje cyklické instrukce, což je výhodou při vícenásobném opakování stejné instrukce nebo celého instrukčního bloku. Dále rodina C5x obsahuje instrukce s podporou dělení, instrukce pro softwarový zásobník a instrukce pro blokové přesuny dat.[5]

Základní vlastnosti skupiny C50

Jde o 16-b procesory. Základem procesorů této řady je jádro C25 z rodiny C2x. C50 nabízejí vyšší výkon oproti C1x a C2x, k čemuž se dospělo novou rozšířenou architekturou.

C50 mají instrukční dobu <instrukčního cyklu [ns] > {instrukční rychlostí [MIPS] }:

<20> {50}, <25> {40}, <30> {33}, <35> {28.6}, <50> {20}.

C50 jsou instrukčně a kódově kompatibilní s C1x a C2x/C2000, protože využívají stejné jádro. Zbývající modely C5x se pouze okrajově liší od C50.[5]

Multiprocesorová komunikace

Přes TDM.

Přes sdílenou externí paměť za použití řídících signálů.

Pomocí přímého přístupu do externí programové/datové paměti.

Periferie

TDM, standardní sériový port, časovač a JTAG port.

Paměť

1056 x 16b datové DARAM, 9k x 16b datové/programové SARAM, 2k x 16 ROM.

Externí adresovatelný prostor je 224K x 16b (64k x 16b program, 64k x 16b data,

64k x 16b I/O a 32k x 16b globální paměti). C50 nabízí programovatelný wait-state pro komunikaci s pomalejšími periferiemi.

Funkční jednotky

32-b ALU 32-b operand, 32-b výsledek

Jednotka pro posuny 16-b operandu o 0-16b vlevo, jednotka s 32-b operandem o

0-16b vpravo, jednotka s 32-b operandem o 0-7b vlevo a jednotka s 32-b

operandem a posunem o 0/1/4b vlevo nebo o 6b vpravo

16-b logická jednotka umožňující logické operace AND, OR a XOR

Násobička 16 x 16b s 32-b výsledkem

16-b ARAU

8 16-b AR

2 32-b akumulátory

Osmistupňový 16-b hardwarový zásobník

Jednostupňový hardwarový zásobník pro uchování 11 konfiguračních registrů při přerušení

Podpora 16-b softwarového zásobníku

Přerušení

6 externích přerušení z toho jedno nemaskovatelné a reset. Vnitřní přerušení náleží vstupu a výstupu standardní sériové linky, vstupu a výstupu TDM, časovači a 13 výhradně softwarovým přerušením. Softwarově lze ovládat všechny uvedené přerušení. Maskovatelné jsou všechny s výjimkou resetu, jednoho nemaskovatelného přerušení a všech softwarových. Tabulka přerušení je v paměti omezeně přesouvatelná.

Adresování

Přímé, nepřímé, bezprostřední, cyklické, bitově reverzní a registrové.

Bootloader

Pokud je k dispozici, pak je uložený v interní ROM. Umožňuje zavádět program z externí paměti, externích I/O a ze standardního sériového portu.

Napájení

Standardně 5V nebo 3V

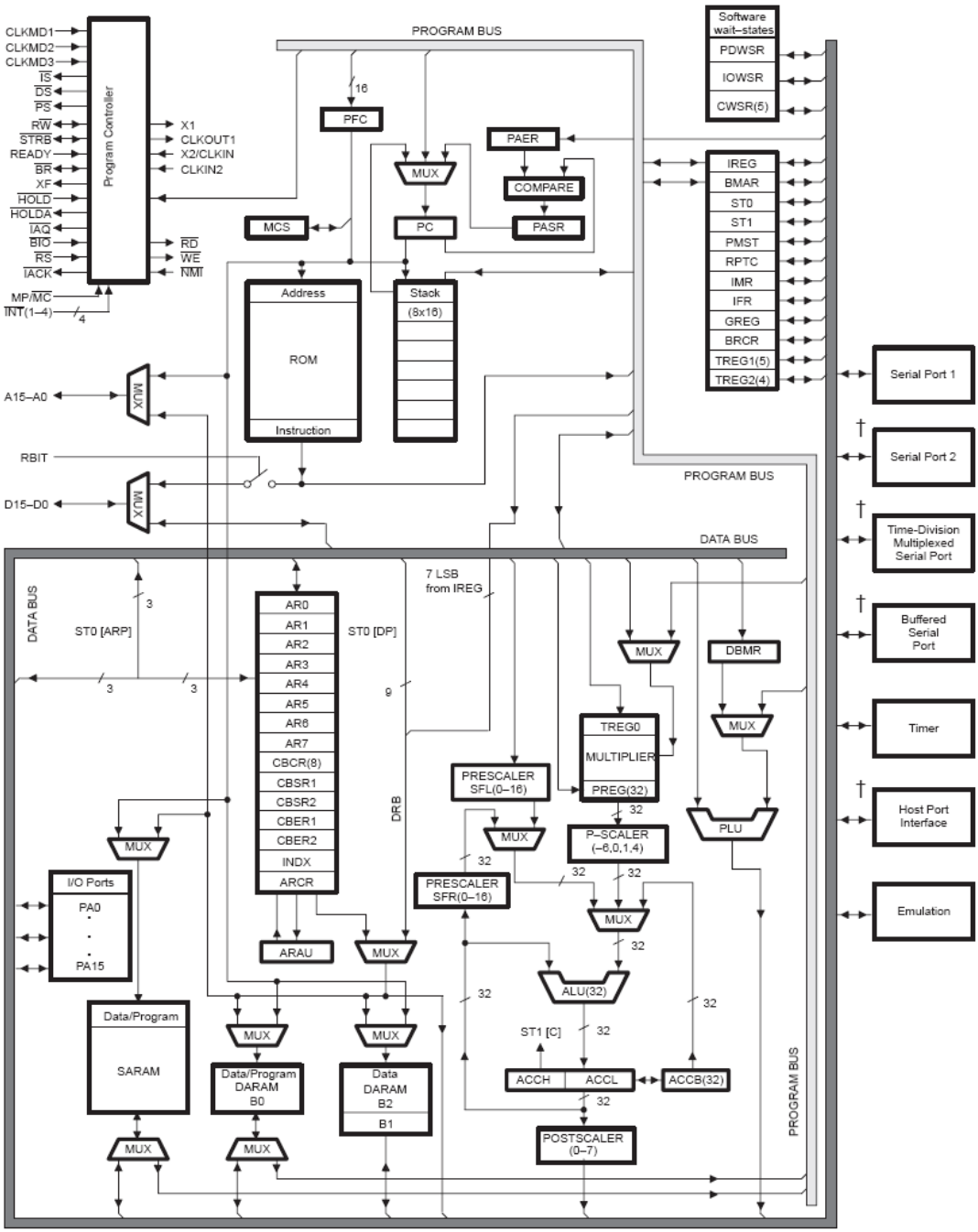
Aplikace

Zpracování audio a videosignálu, hlasová syntéza, grafické akcelerátory, řízení hybných prvků, inteligentní senzory, telefonie, modemy, opakovače a jiné.[5]

Oficiální dokumentaci ke všem typům procesorů od firmy Texas Instruments lze nalézt také na internetových stránkách Texas Instruments.[1]

Blokové schéma skupiny C50

Na (Obr.1) je blokové schéma DSP *TMS320C50*, které ukazuje propojení jednotlivých bloků procesoru. Dle návaznosti operačních bloků potom vytváříme program tak, aby se v jednom kroku provedlo více operací, pokud je potřeba. Například můžeme v jednom kroku vynásobit dvě čísla a následně je posunout. Pokud budeme tímto způsobem při vytváření aplikace uvažovat, zkrátíme její výpočetní čas.

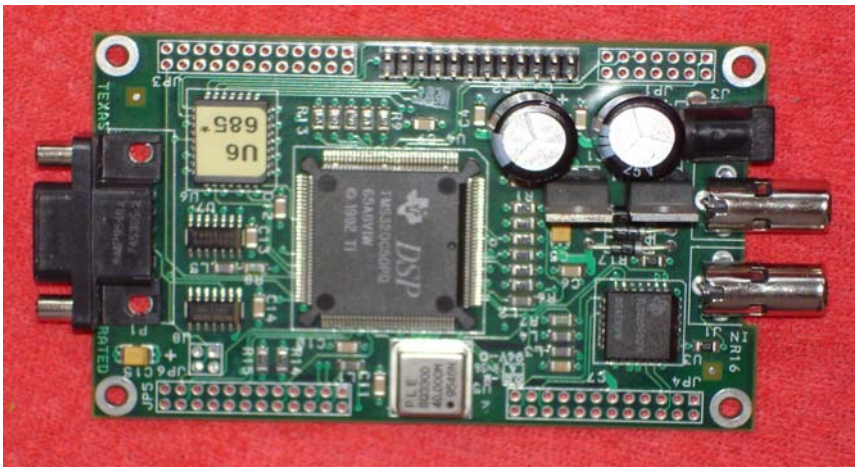


Obr. 1. *Blokové schéma procesoru TMS320C50*

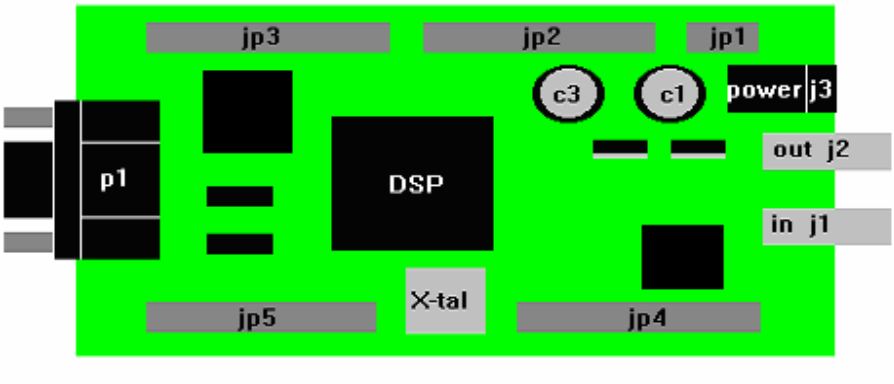
2 POPIS BALENÍ DSP TMS320C50

Hlavní součástí vývojového modulu je plošný spoj osazený procesorem, dále je v balení propojovací kabel pro sériovou komunikaci RS-232, napájecí adaptér, softwarové vybavení a manuál.

Obr. 2. *Příslušenství k mikroprocesoru*



Obr. 3. *Deska mikroprocesoru*



Obr. 4. *Deska mikroprocesoru s označenými bloky*

DSP Starter-Kit

Je to kompletní balíček s hardwarovým vybavením a softwarem pro kompilaci, krokování programu a ladění programu. DSP Starter Kit je ideálním nástrojem, jak se dostat do zajímavého světa digitálních signálových procesorů. Tento balíček patří u firmy Texas Instruments k nejoblíbenějším, protože jej stačí pouze připojit k počítači přes sériové rozhraní RS232 tedy COM port. Standardně bývá nastaven port COM1, avšak po spuštění ladícího prostředí, které čeká na připojení DSP lze změnit číslo portu, rychlost komunikace a inverzi bitů.

Softwarové vybavení obsahuje v současné době CD se softwarem, avšak v mém případě se musím spokojit s disketou 3,5 palce, která byla k procesoru dodána. Na této disketě jsou obsaženy důležité ovladače procesoru. Jako největší komplikaci lze považovat, že verze tohoto softwarového vybavení funguje pouze pod operačním systémem MS-DOS, na který už běžně nenarazíte. Pokud si zrovna tento operační systém nehodláte nainstalovat je nutné ze startovací diskety smazat určité soubory, abyste uvolnily dostatek místa pro soubory na ovládání procesoru TMS320C50.

Soubor „dsk5a.exe“ je kompilátor. Program, který vytvoříte v textovém režimu zadáte k tomuto souboru jako spouštěcí parametr. Kompilátor projede váš program po řádcích. Pokud jste nikde neudělali chybu, tak se vám na disketě vytvoří nový zkompilovaný soubor s původním názvem vašeho programu, avšak jeho přípona bude „.DSK“.

Soubor „dsk5d.exe“ je debuger. Debuger je ladící prostředí, ve kterém lze program po krocích testovat a poté jej odladit. Je v něm umístěné okno s přehledem důležitých registrů. Lze tedy kontrolovat, jak se po jednotlivých krocích mění výsledky v registrech, které v programu právě využíváte. Po načtení programu tlačítkem <Load> DSK, můžeme ještě změnit obsahy registrů tlačítkem <Modify>. Do registrů uložíme hodnoty, se kterými chceme pracovat. A program můžeme otestovat zda vrací správné výsledky, které očekáváme.

Soubor „dsk5l.exe“ je loader. Tento program při zadání vašeho *dsk* souboru jako spouštěcího parametru nahraje váš program do procesoru, který jej bude provádět.

II. PRAKTICKÁ ČÁST

3 PŘÍPRAVA APLIKACÍ PRO TMS320C50

Pokud programujeme aplikace pro DSP, tak postupujeme nejlépe podle následujícího postupu. Nejprve se seznámíme s možnostmi procesoru a s jeho instrukční sadou. S touto instrukční sadou, která je podrobněji popsána v TMS320C5x User‘s Guide budeme vytvářet program. Dalším aspektem, který musíme sledovat je blokové schéma procesoru, které je vyobrazeno na (Obr.1). Pokud zvolíme optimální instrukci, která provede pouze požadovanou operaci, tak zvýšíme rychlost výpočtu algoritmu. Existují také instrukce, které nám nahradí několik operací v jediném cyklu, což je dáno architekturou procesoru.

Programování probíhá v jazyce assembler, což je programovací jazyk zpravidla odlišný pro každý typ procesoru. U tohoto programovacího jazyku se při volbě instrukcí musí hlídat jaké registry nám instrukce ovlivňuje a zda by nebylo vhodnější použít jinou instrukci.

Pro začátečníky je k procesoru *TMS320C50* dodávána příručka *Starter-Kit* a uživatelský manuál *User‘s Guide*. Starter-Kit obsahuje příklady pro začínající programátory DSP a ukazuje jakým stylem se program strukturuje a jak se zapisují instrukce. Dále se můžete dozvědět jaké se používají datové typy a formáty v jakých se dají data do programu zapisovat. Příručka User‘s Guide je obsáhlejší. V této příručce jsou podrobně popsány všechny instrukce včetně možnosti jejich použití a způsobu jejich zápisu do programu. Při zapisování programu bereme ohled na rozdělení paměti na programovou část a část datovou.

Program se dá napsat v textovém editoru a je rozumné potom tomuto souboru přiřadit koncovku *ASM*. Bude z toho zřejmé, že jde o zdrojový kód v assembleru.

Tento kód se nejprve musí zkompilovat. Poté co byl zkompilován získáme nový soubor s koncovkou *DSK*. Tento zkompilovaný zdrojový kód nakopírujeme do procesoru *TMS320C50* přes sériové rozhraní RS-232. Propojovací kabel je součástí balení od firmy Texas Instruments. Program můžeme testovat krokováním instrukcí pomocí ladícího prostředí, které je na disketě se softwarovým vybavením k DSP a opravíme případné chyby programu.

3.1 Násobení s pohyblivou řádovou čárkou

U tohoto algoritmu je důležité, aby se provedlo vynásobení mantisy a poté sečtení exponentů. Zápis čísla při pohyblivé řádové čárce bude vypadat v 32b vyjádření tak, že horních

16b bude mantisa a dolních 16b bude exponent. Pokud vynásobíme dvě mantisy v 16b vyjádření je nutné provádět zároveň korekci, která nám výsledné číslo upraví opět na 16b. Dalším aspektem pro postup při programování pohyblivé řádové čárky je práce s exponentem. Pokud násobíme dvě čísla, pak se jejich exponenty musí sečíst. Poté co je výpočet proveden ještě přičteme korekci chyby, která nám vnikne při normalizaci výsledku.

Jednoduchá rovnice násobení:

$$y = A \cdot B$$

Argumenty násobení:

$$A = mantisa_A \cdot 2^{\text{exponent}_A}$$

$$B = mantisa_B \cdot 2^{\text{exponent}_B}$$

S těmito argumenty budeme při programování počítat. Musíme také počítat s tím, že i výsledek bude ve stejném tvaru rozdělený na 16b mantisu a 16b exponent tedy 32b výsledek.

$$y = mantisa_y \cdot 2^{\text{exponent}_y}$$

Datová paměť je rozdělená po 16b paměťových blocích, což jsou dva bajty pro paměťovou buňku. Pokud počítáme s pohyblivou řádovou čárkou, tak je nejlepší zapisovat mantisu s exponentem do dvou po sobě následujících paměťových míst. Dále je při programování vhodné obsazovat postupně registry. Získáte tím přehled o tom, kolik je ještě volných registrů k použití.

Výsledek se při výpočtech řádově posouvá. Samotná normalizace výsledku spočívá v tom, že upravujeme výsledek posouváním do doby než narazíme na kombinaci 10. Pokud se dostaneme k tomuto vyjádření posunutého výsledku, tak se normalizace ukončí a výsledek bude ve správném tvaru uložen.

3.1.1 Algoritmus násobení s pohyblivou řádovou čárkou

```
1       .mmregs
```

```
;povolí práci s registry
  2     .ps 0a00h
;označuje místo v paměti, od kterého začíná program
  3     .entry
;označení začátku algoritmu
  4 START:    MAR *,AR0
;nastavení aktivního registru na AR0, kde je uložena adresa A
  5     LT *+,AR1
;načtení hodnoty do TREG0 z adresy uložené v AR0, kde je uloženo B
;nastavení aktivního registru na AR1
  6     MPY *+,AR0
;vynásobení mantisy A*B a nastavení aktivního registru na AR0
  7     PAC
;P-registr, kde je uložen produkt násobení se uloží do akumulátoru
  8     BCND NENULA,NEQ
;pokud výsledek v akumulátoru ACC ≠ 0 proveď skok na NENULA
  9     MAR *,AR3
;nastavení aktivního registru na AR3
  10    SACL *+
;uložení dolních 8 bitů z ACC tedy výsledku na adresu v AR3
  11    B KONEC
;nepodmíněný skok na KONEC
  12 NENULA:  LAR AR2,#0
;vynuluje paměť na adrese uložené v registru AR2
  13    MAR *,AR2
;nastavení aktivního registru na AR2
  14 SMYCKA:  NORM *-
;provede se normalizace výsledku
  15    BCND SMYCKA,NTC
;dokud bude v registru TC = 0 bude se provádět skok na SMYCKA
  16    MAR *,AR3
;nastavení aktivního registru na AR3
  17    SACH *+,AR0
;zkopíruj data z akumulátoru do paměti na adresa+1 v registru AR3
;nastavení aktivního registru na AR0
  18    LACL *,AR1
;prohodí horních 8b s dolními 8b a vynuluje horních 8b
  19    ADD *,AR3
;sečtou se exponenty exponent_A + exponent_B , aktivní bude registr AR3
  20    ADD AR2,AR3
;přičte se k výsledku chyba po normalizaci uložená na adrese v AR2
```

```
  21 KONEC:   SACL *
;zapíše výsledek z akumulátoru na adresu, na kterou ukazuje AR3
  22    .end
;označuje konec programu
  23
;prázdný řádek je informace pro kompilátor, kde má končit překlad
```

3.1.2 Doplňující informace k algoritmu

V programu se používají registry:

AR0 - ukazuje na adresu, kde je uloženo A

AR1 - ukazuje na adresu, kde je uloženo B

AR2 - ukazuje na adresu, kde se ukládá korekce výsledku

AR3 - ukazuje na adresu, kde je uložen výsledek násobení

Před spuštěním programu se do těchto registrů uloží adresy, na kterých jsou uložena data. Před spuštěním programů je potřeba nastavit registr SXM na hodnotu 1. Tento registr ovlivňuje stav akumulátoru.

Pro názorný příklad zde uvádím tabulku, ve které jde vidět jak se nastavení registrů oproti počátečním hodnotám na konci algoritmu změnilo.

Tab. 1. Tabulka obsahující změnu hodnot po proběhnutí algoritmu

Registr / paměťová buňka	**Před provedením algoritmu**	**Po provedení algoritmu**
AR0	0800	0801
AR1	0802	0803
AR2	0806	fff5
AR3	0804	0805
Obsah adresy 0800h (MANTISA A)	0010	0010
Obsah adresy 0801h (EXPONENT A)	0010	0010
Obsah adresy 0802h (MANTISA B)	0020	0020
Obsah adresy 0803h (EXPONENT B)	0020	0020
Obsah adresy 0804h (MANTISA Y)	0000	0200
Obsah adresy 0805h (EXPONENT Y)	0000	0030

3.2 Programování funkce sinus

Sinus je jednou z nejznámějších goniometrických funkcí. V pravoúhlém trojúhelníku ji definujeme jako poměr protilehlé odvěsny ku přeponě. Označujeme ji zkráceně sin a jejím grafem je sinusoida.

3.2.1 Základní vlastnosti funkce sinus

Funkce má následující vlastnosti:

Definiční obor: jsou to všechna reálná čísla R

Obor hodnot: $\langle -1;1 \rangle$

Rostoucí: v každém intervalu $\left(-\frac{\pi}{2} + 2 \cdot k \cdot \pi; \frac{\pi}{2} + 2 \cdot k \cdot \pi \right)$

Klesající: v každém intervalu $\left(\frac{\pi}{2} + 2 \cdot k \cdot \pi; -\frac{\pi}{2} + 2 \cdot k \cdot \pi \right)$

Maximum: $\frac{\pi}{2} + 2 \cdot k \cdot \pi$

Minimum: $-\frac{\pi}{2} + 2 \cdot k \cdot \pi$

Taylorův polynom: $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n+1}}{(2n+1)!}$

3.2.2 Aproximace funkcí

Digitální signálové procesory jsou svou architekturou přímo předurčeny pro řešení matematických úloh i složitých transformací. Nejčastějším problémem při programování aproximace funkce na klasických mikroprocesorech byla nutnost alokovat velkou část paměti pro zápis tabulky. Do tabulky se ukládaly funkční hodnoty *y*. S rostoucím požadavkem na přesnost výpočtů se velikost tabulky a alokovaného místa v paměti neustále zvětšovala. Tomuto však lze zabránit aproximací funkce. Nejjednodušší metodou je využití lineární aproximace úseků. Tato metoda nemusí dosahovat požadované přesnosti, i když značně uvolní místo v paměti.

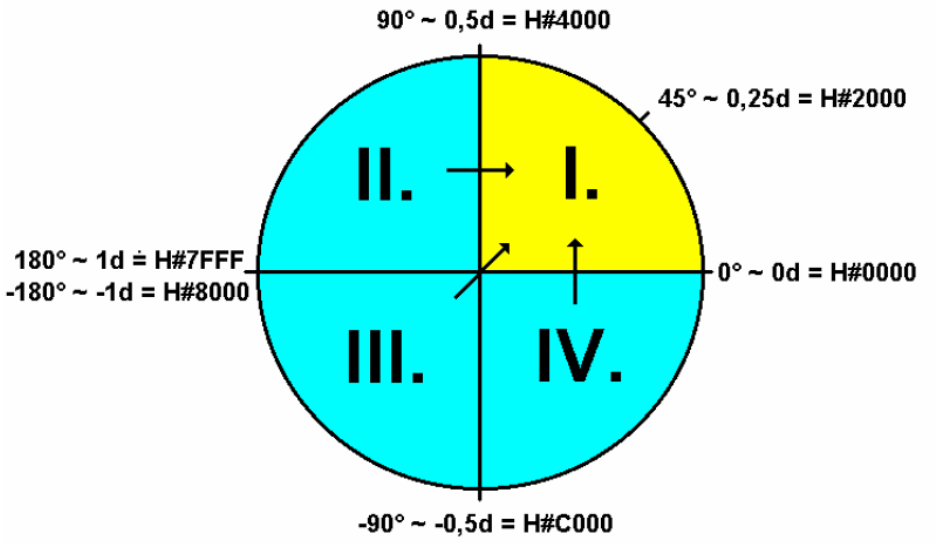
Vhodnější metodou je, díky výpočetnímu výkonu DSP, použití aproximace polynomem vyššího řádu.

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n$$

Do paměti se budou místo tabulky ukládat pouze koeficienty a_n u příslušných mocnin polynomu a ostatní výpočty mnohočlenu obstará výkonné jádro řešící tento algoritmus.

Můžeme dokonce využít stejný algoritmus i pro různé typy funkčních závislostí pouhou změnou koeficientů polynomu.

Pro klasické matematické funkce lze využít rozvoje do Taylorových řad. Pro žádanou přesnost je nutné si zvolit, jak přesné budou koeficienty mnohočlenu, jaký bude řád mnohočlenu a rozsah výstupních operandů.



Obr. 5. *Jednotková kružnice*

Na obrázku, kde je vyobrazena jednotková kružnice (Obr.5) můžeme vidět, že funkce sinus je rozdělena do čtyř kvadrantů. Protože je však jednoduché převádět výstupní hodnoty funkce sinus mezi kvadranty, tak jsem se při tvorbě algoritmu rozhodl, že budu počítat pouze se vstupními hodnotami zadaného úhlu v I. kvadrantu. Rozsah zadaných hodnot musí tedy ležet v intervalu $\langle 0;90 \rangle$ stupňů. V algoritmu se počítá s tím, že se bude hodnota x zadávat v Radiánech.

$$radiány = stupně \cdot \frac{\pi}{180}$$

Aproximace funkce sinus:

$$y = 0{,}0056560964 \cdot x^5 + 0{,}0063339479 \cdot x^4 - 0{,}1725383830x^3 + \ldots$$

$$\ldots + 0{,}0023465585 \cdot x^2 + 0{,}9996666298 \cdot x + 0{,}0000058512$$

3.2.3 Algoritmus funkce sinus

U předchozího algoritmu násobení jsem detailně popisoval co se ve kterém kroku v programu přesně stane. Z popisu předchozího algoritmu lze vyčíst, jakým způsobem procesor pracuje. Zde u algoritmu funkce sinus popíšu, jaké operace se provedou v bloku instrukcí.

Do paměti jsem zapsal konstanty polynomu v hexadecimálním vyjádření. Před spuštěním programu nastavte do registru AR2 adresu hodnoty x, pro kterou chcete sinus spočítat v 16b vyjádření.

Zdrojový kód algoritmu

```
   1        .mmregs
   2        .ps 0800h
;první instrukce aktivuje práci s registry a druhá místo v paměti,
 kam budeme zapisovat konstanty
   3  X1    .word 0000h
   4  X2    .word 0ffeah
   5  X3    .word 009ah
   6  X4    .word 2c2bh
   7  X5    .word 019fh
   8  X6    .word 0173h
   9  S1    .word 0000h
;v tomto bloku definujeme vstupní proměnné v hexadecimálním vyjádření
   10       .ps 0a00h
   11       .entry
;v této části jsem definoval začátek programu
   12 START:      LAR AR0,#X1
   13             LAR AR2,#S1
   14             MAR *,AR0
   15             LACC *+,14,AR1
; zde se provádí operace a = 1·A
   16             LT *,AR0
   17             MPY *+,AR1
   18             APAC
```

```
; zde se provádí operace b = a + X2 · x
    19            SQRA *,AR2
    20            EXAR
    21            PAC
    22            BSAR 13
    23            SACL *,AR0
    24            EXAR
    25            LT *+,AR2
    26            MPY *,AR2
    27            APAC
;zde se provádí operace c = a + b + X3 · x
    28            SPLK #2, BRCR
    29            RPTB KON_CYK - 1
    30            LT *,AR1
    31            MPY *,AR2
    32            EXAR
    33            PAC
    34            BSAR 13
    35            SACL *,AR2
    36            EXAR
    37            LT *,AR0
    38            MPY *+,AR2
    39            APAC
    40 KON_CYK
;zde se provádí cyklus, ve kterém se dořeší d = a + b + c + X4 · x ,
e = a + b + c + d + X5 · x a nakonec sin x = a + b + c + d + e + X6 · x
    41            SACH *,AR2
;výsledek zkopíruj z akumulátoru na adresu, která je zapsána v AR2
    42
    43      .end
    44
```

Na konci programu za posledním zápisem .END označujícím konec je nutné ještě nechat následující řádek prázdný, aby kompilátor rozeznal konec programu.

Výsledek se zapíše na adresu, na kterou právě ukazuje registr AR2.

Při použití této goniometrické funkce jako podprogramu stačí zapsat na začátek programu název návěští, pod kterým budete podprogram volat. Na konec algoritmu by jste napsali RET pro návrat z podprogramu.

3.3 Dělení s pevnou řádovou čárkou

U tohoto algoritmu se provede vydělení mantisy argumentů. Zápis čísla při pevné řádové čárce bude 16b mantisa. Exponenty nás nebudou zajímat, budeme předpokládat že se jedná o celočíselné vyjádření argumentů. Dalším předpokladem bude, že ani jeden argument dělení nebude záporný. Udělat ze záporného čísla kladné není obtížné, takže jsem algoritmus neošetřoval, ale považuji za důležité na to upozornit.

Jednoduchá rovnice dělení:

$$y = A / B$$

Argumenty násobení:

$$A = mantisa_A$$

$$B = mantisa_B$$

S těmito argumenty budeme při programování počítat. Musíme také počítat s tím, že i výsledek bude ve stejném tvaru takže 16b mantisa.

$$y = mantisa_y$$

Algoritmus dělení s pevnou řádovou čárkou

Program je postaven tak, že do registru ARCR zapisujeme argument dělení B a do registru AR0 zapíšeme argument dělení A. mezivýsledek se bude ukládat do datové paměti na adresu S1.

```
   1             .mmregs
   2             .ps 0800h
   3 S1          .word 0h
;provede se aktivace registrů, nastavení aktuálního místa v paměti
 a vynulovaného S1 do paměti
   4             .ps 0a00h
   5             .entry
;označuje místo v paměti, kde začíná program
   6 START:      MAR AR0
   7             SPLK #15,BRCR
;nastavíme kolikrát má proběhnout smyčka, protože jde o úpravu 16b
 čísla proběhne cyklus od 15 do 0 přesně 16-krát
   8             RPTB KON - 1
   9 POROVNEJ:   CMPR 1
```

```
   10            BCND JEDNA,TC
   11            BCND NULA,TNC
;pokud je číslo na adrese, na kterou ukazuje ARCR větší než číslo
 na adrese, na kterou ukazuje AR0 skoč na návěští JEDNA, pokud ne
 proveď skok na návěští NULA
   12 JEDNA:     LACL S1
   13            CLRC C
;vynuluje se registr C (carry)
   14            ROL
;načte se hodnota z S1 do akumulátoru a provede se rotace doleva
   15            SACL S1
   16            B KONEC
;výsledný tvar se uloží do akumulátoru a provede se
 nepodmíněný skok na návěští KONEC
   17 NULA:      LACL S1
   18            SETC C
;registr carry bude nastaven na 1
   19            ROL
   20            SACL S1
   21            LACL AR0
   22            SUB ARCR
   23            SACL AR0
;provede se rozdíl argumentů a výsledek z akumulátoru se uloží
 na adresu uloženou v AR0
   24 KONEC:     LACL ARCR
   25            BSAR 1
   26            SACL ARCR
;výsledek se uloží na adresu na kterou ukazuje registr ARCR
   27 KON
;označení konce smyčky
   28      .end
   29
;konec programu
```

Poznámky k algoritmu dělení s pevnou řádovou čárkou

Výsledkem bude mantisa uložená na adrese, která je zapsána v registru ARCR. Výsledkem bude 16b číslo. Program není ošetřen na záporné argumenty, proto bude výsledek vždy kladné číslo.

3.4 Dělení s pohyblivou řádovou čárkou

Jde o propojení algoritmů dělení s pevnou řádovou čárkou a části algoritmu pro násobení s pohyblivou řádovou čárkou, který zajišťuje normalizaci výsledku. Ovšem s rozdílem, že při dělení se od sebe exponenty musí odečítat.

Rovnice pro dělení:

$$y = A / B$$

Argumenty násobení:

$$A = mantisa_A \cdot 2^{\text{exponent}_\text{A}}$$

$$B = mantisa_B \cdot 2^{\text{exponent}_\text{B}}$$

S těmito argumenty budeme při programování počítat. Musíme také počítat s tím, že i výsledek bude ve stejném tvaru rozdělený na 16b mantisu a 16b exponent.

$$y = \frac{mantisa_A}{mantisa_B} \cdot 2^{\text{exponent}_\text{A} - \text{exponent}_\text{B}} = mantisa_y \cdot 2^{\text{exponent}_\text{y}}$$

Algoritmus dělení s pohyblivou řádovou čárkou

```
1              .mmregs
2              .ps 0800h
3 S1           .word 0h
;provede se aktivace registrů, nastavení aktuálního místa v paměti
 a vynulovaného S1 do paměti
4              .ps 0a00h
5              .entry
;označuje místo v paměti, kde začíná program
6 START:       MAR AR0
7              SPLK #15,BRCR
;nastavíme kolikrát má proběhnout smyčka, protože jde o úpravu 16b
 čísla proběhne cyklus od 15 do 0 přesně 16-krát
8              RPTB KON - 1
9 POROVNEJ:    CMPR 1
10             BCND JEDNA,TC




11             BCND NULA,TNC
```

```
;pokud je číslo na adrese, na kterou ukazuje ARCR větší než číslo
 na adrese, na kterou ukazuje AR0 skoč na návěští JEDNA, pokud ne
 proveď skok na návěští NULA
12 JEDNA:     LACL S1
13            CLRC C
;vynuluje se registr C (carry)
14            ROL
;načte se hodnota z S1 do akumulátoru a provede se rotace doleva
15            SACL S1
16            B KONEC
;výsledný tvar se uloží do akumulátoru a provede se
 nepodmíněný skok na návěští KONEC
17 NULA:      LACL S1
18            SETC C
;registr carry bude nastaven na 1
19            ROL
20            SACL S1
21            LACL AR0
22            SUB ARCR
23            SACL AR0
;provede se rozdíl argumentů a výsledek z akumulátoru se uloží
 na adresu uloženou v AR0
24 KONEC:     LACL ARCR
25            BSAR 1
26            SACL ARCR
;výsledek se uloží na adresu na kterou ukazuje registr ARCR
27 KON
;označení konce smyčky
28            LAR AR1,#0
;vynuluje paměť na adrese uložené v registru AR1
29            MAR *,AR1
;nastavení aktivního registru na AR1
30 SMYCKA:    NORM *-
;provede se normalizace výsledku
31            BCND SMYCKA,NTC
;dokud bude v registru TC = 0 bude se provádět skok na SMYCKA
32            MAR *,AR2
;nastavení aktivního registru na AR2


33            SACH *+,AR0
```

```
;zkopíruj data z akumulátoru do paměti na adresa+1 v registru AR2
;nastavení aktivního registru na AR0
34            LACL *,AR1
;prohodí horních 8b s dolními 8b a vynuluje horních 8b
35            SUB *,AR2
;odečtou se exponenty exponent_A-exponent_B, aktivní bude registr AR2
36            ADD AR1,AR2
;přičte se k výsledku chyba po normalizaci uložená na adrese v AR1
37 FINAL:     SACL *
;zapíše výsledek z akumulátoru na adresu, na kterou ukazuje AR3
38      .end
;označuje konec programu
39
;prázdný řádek je informace pro kompilátor, kde má končit překlad
```

Poznámky k algoritmu dělení s pevnou řádovou čárkou

Výsledkem bude mantisa v 16b vyjádření uložená na adrese, která je zapsána v registru ARCR a exponent po dělení bude uložen tam kam ukazuje adresa registru AR2. Program není ošetřen na záporné argumenty, proto bude výsledek vždy kladné číslo. Dále se provádí normalizace tedy korekce výsledků, která se ukládá na adresu uloženou v registru AR1.

ZÁVĚR

Cílem práce bylo vytvořit matematické knihovny pro DSP *TMS320C5x*. Jako nejdůležitější kapitoly své práce tedy považuji tvorbu aplikací pro DSP, které lze volat jako podprogram nějaké jiné aplikace. Všechny algoritmy po zadání vstupních parametrů provedou výpočet a po ukončení algoritmu se uloží jeho výsledek.

Asi nejsložitější při přechodu na digitální signálové procesory je rozdílný zápis vstupních argumentů a dat, se kterými DSP pracují. Hlavní rozdíl mezi běžnými mikroprocesory a DSP sem pocítil při práci se dvojkovým doplňkem, se kterým DSP pracují. Tento aspekt mění uvažování programátory při vyvíjení aplikací pro DSP. Jako u každého procesoru se i u DSP hlídá zásobník, aby nedošlo k jeho přetečení. Pokud by došlo k přetečení vznikl by v programu chaos a nedalo by se již očekávat, že nám aplikace vrátí správný výsledek. Rozdělení argumentu při výpočtech s pohyblivou řádovou čárkou na mantisu a exponent opět stěžuje přemýšlení programátora. Programátor musí zajistit, aby byl výsledek ve správném vyjádření. Výsledek algoritmu musí mít mantisu i exponent ve správném tvaru. Proto se během výpočtů provádí normalizace výsledku.

U DSP *TMS320C5x* je možnost zvolit si v jakém vyjádření chceme zapsat vstupní argument jestli má být ve vyjádření Q15, hexadecimálním vyjádření nebo v binárním k´du záleží na požadavcích uživatele. Ladící prostředí k DSP umožňuje měnit v jakém formátu budou pro uživatele data při ladění programu zobrazena. částečně se zjednoduší kontrola programu zda vrací správné výsledky.

Odladěné programy, které jsou součástí práce jsou v nepopsaném tvaru k dispozici v příloze. Stačí provést jejich kompilaci a lze je využít. Nebo můžete tento kód přidat jako podprogram ke svému programu.

Zbývající část práce, která se nezabývá tvorbou algoritmů je méně důležitá a slouží spíše pro seznámení s digitálními signálovými procesory. Pokud chce někdo začít programovat DSP, tak je dobré, aby se seznámil se způsobem jejich programování a s matematickými základy týkající práce se dvojkovým doplňkem.

SEZNAM POUŽITÉ LITERATURY

[1] Procesory firmy Texas Instruments [online].[cit 2006-05-16], elektronická verze dokumentace k procesorům. Dostupný z WWW: http://www.ti.com

[2]Mikroprocesorová technika [online].[cit 2006-04-25], Západočeská univerzita v Plzni. Internetové stránky zabývající se mikroprocesory. Dostupný z WWW: http://home.zcu.cz/~mhanus

[3]Texas Instruments: User's Guide TMS320C5xS. USA: Texas Instruments, 1996. ISBN 2547301-9761.

[4]Matematika [online].[cit 2006-04-25], Wikipedie - otevřená encyklopedie.

Internetové stránky elektronické encyklopedie, která se neustále rozvíjí.

Dostupný z WWW: http://cs.wikipedia.org

[5]Družbík, T.: Příprava Aplikace digitálního signálového procesoru TMS320C5x. Diplomová práce, UTB ve Zlíně, 2003.

[6]Mikroprocesorová technika [online].[cit 2006-02-7], Vysoké učení technické v Praze. Studentské stránky Pískoviště. Dostupný z WWW: http://bbs.cvut.cz/

[7]Matematika [online].[cit 2005-11-22], Přírodovědecká fakulta Masarykovy

univerzity v Brně. Internetové stránky zabývající se matematickými a fyzikálními výpočty. Dostupný z WWW: http://www.math.muni.cz/studijni/dsp.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DSP	Digitální signálový procesor.
PC	Osobní počítač (Personal Computer).
p1	Konektor pro sériovou komunikaci přes RS-232.
X-tal	Krystal řídící časování procesoru.
c1, c3	Kondenzátory pro stabilizaci napájení.
power j3	Konektor pro připojení napájení.
in j1	Analogový vstupní konektor.
out j2	Analogový výstupní konektor.
jp1 - jp5	Konektory pro připojení vnějších periferií pro digitální komunikaci.
ACC	Akumulátor digitálního signálového procesoru.
assembler	Jazyk symbolických adres (JSA).
b	Toto písmeno za číslem znamená hodnotu v bitech, což je nejmenší jednotka informace, která nabývá hodnoty 1 nebo 0.
h	Toto písmeno za číslem znamená hodnotu v hexadecimálním vyjádření, což je šestnáctková soustava. Nabývá hodnot 1 až 9 a A až F.
*	V programu znamená, že se provádí operace s aktuálně nastaveným registrem.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: NÁSOBENÍ S POHYBLIVOU ŘÁDOVOU ČÁRKOU

```
          .mmregs
          .ps 0a00h
          .entry
START:    MAR *,AR0
          LT *+,AR1
          MPY *+,AR0
          PAC
          BCND NENULA,NEQ
          MAR *,AR3
          SACL *+
          B KONEC
NENULA:   LAR AR2,#0
          MAR *,AR2
SMYCKA:   NORM *-
          BCND SMYCKA,NTC
          MAR *,AR3
          SACH *+,AR0
          LACL *,AR1
          ADD *,AR3
          ADD AR2,AR3
KONEC:    SACL *
          .end
```

PŘÍLOHA P II: ALGORITMUS FUNKCE SIN X

```
          .mmregs
          .ps 0800h
X1        .word 0000h
X2        .word 0ffeah
X3        .word 009ah
X4        .word 2c2bh
X5        .word 019fh
X6        .word 0173h
S1        .word 0000h
          .ps 0a00h
          .entry
START:    LAR AR0,#X1
          LAR AR2,#S1
          MAR *,AR0
          LACC *+,14,AR1
          LT *,AR0
          MPY *+,AR1
          APAC
          SQRA *,AR2
          EXAR
          PAC
          BSAR 13
          SACL *,AR0
          EXAR
          LT *+,AR2
          MPY *,AR2
          APAC
          SPLK #2, BRCR
          RPTB KON_CYK - 1
          LT *,AR1
          MPY *,AR2
          EXAR
          PAC
          BSAR 13
          SACL *,AR2
          EXAR
          LT *,AR0
          MPY *+,AR2
          APAC
          KON_CYK
          SACH *,AR2
```

```
.end
```

PŘÍLOHA P III: DĚLENÍ S PEVNOU ŘÁDOVOU ČÁRKOU

```
        .mmregs
        .ps 0800h
S1      .word 0h
        .ps 0a00h
        .entry
START:          MAR AR0
                SPLK #15,BRCR
                RPTB KON - 1
POROVNEJ:       CMPR 1
                BCND JEDNA,TC
                BCND NULA,TNC
JEDNA:          LACL S1
                CLRC C
                ROL
                SACL S1
                B KONEC
NULA:           LACL S1
                SETC C
                ROL
                SACL S1
                LACL AR0
                SUB ARCR
                SACL AR0
KONEC:          LACL ARCR
                BSAR 1
                SACL ARCR
KON
        .end
```

PŘÍLOHA P IV: DĚLENÍ S POHYBLIVOU ŘÁDOVOU ČÁRKOU

```
        .mmregs
        .ps 0800h
S1      .word 0h
        .ps 0a00h
        .entry
START:      MAR AR0
            SPLK #15,BRCR
            RPTB KON - 1
POROVNEJ:   CMPR 1
            BCND JEDNA,TC
            BCND NULA,TNC
JEDNA:      LACL S1
            CLRC C
            ROL
            SACL S1
            B KONEC
NULA:       LACL S1
            SETC C
            ROL
            SACL S1
            LACL AR0
            SUB ARCR
            SACL AR0
KONEC:      LACL ARCR
            BSAR 1
            SACL ARCR
KON
            LAR AR1,#0
            MAR *,AR1
SMYCKA:     NORM *-
            BCND SMYCKA,NTC
            MAR *,AR2
            SACH *+,AR0
            LACL *,AR1
            SUB *,AR2
            ADD AR1,AR2
FINAL:      SACL *
        .end
```