4 IMPLEMENTÁCIA FIR FILTRA POMOCOU PROCESORA ADSP BLACKFIN

Ciele cvičenia:

- detailná analýza knižničnej funkcie *fir_fr16()* z knižnice dsplib,
- princíp korekcií anomálií rôznych verzií DSP jadra v knižničných funkciách,
- využitie štandardných IO funkcii z knižnice stdio pre ladenie programu,
- overenie správnosti výpočtu analyzovanej knižničnej funkcie,
- zrýchlenie simulácia s využitím predkompilovanej (compiled) simulácie,
- ďalšie dostupné knižničné funkcie pre FIR filtráciu.

4.1 ÚVOD

FIR filter patrí z pohľadu signálových procesorov medzi najjednoduchší algoritmus ČSS. Prakticky všetky dostupné DSP sú optimalizované práve pre tento typ algoritmu. Jadro FIR filtra je u klasických DSP s jednou MAC jednotkou tvorené jedinou MAC inštrukciou. V asembleri je tak program pre FIR filter väčšinou pomerne jednoduchý, čo bolo demonštrované v jednom z predchádzajúcich cvičení. Knižničné funkcie pre jazyk C, ktoré sú optimalizované (v asembleri) pre prostredie jazyka C však musia riešiť aj väzbu medzi volajúcou funkciou v jazyku C a knižničnou funkciou:

- odovzdávanie vstupných parametrov do volanej funkcie,
- uloženie používaných registrov pred začiatkom výkonnej časti knižničnej funkcie,
- spätné obnovenie používaných registrov pred návratom do volajúcej C funkcie,
- manažment zásobníka[1],
- odovzdanie návratovej hodnoty volajúcej C funkcie.

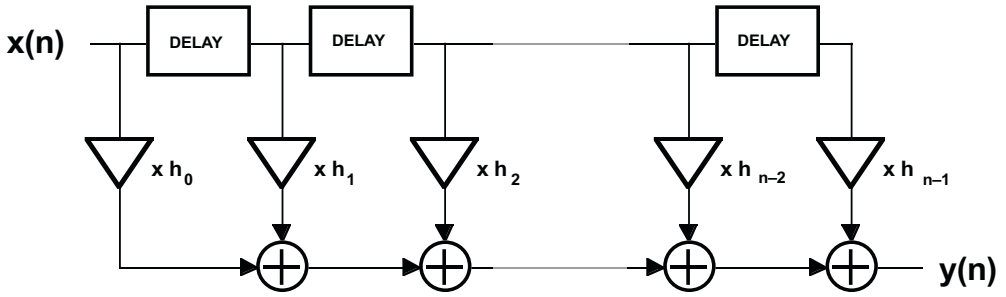
V prípade moderných DSP sa situácia ďalej komplikuje tým, že efektívne knižničné funkcie využívajú paralelný kód pracujúci s viacerými MAC jednotkami (v prípade jadra Blackfin s dvoma jednotkami MAC0 a MAC1). Zdrojové kódy knižničných funkcií sú tak, minimálne na prvý pohľad, pomerne zložité. Funkcia FIR filtra je najjednoduchšia funkcia ČSS a preto bude v rámci cvičenia detailne analyzovaná ako

[1] Zásobník je dátová štruktúra typu LIFO (Last In First Out), ktorá je vytváraná vo vyhradenej časti (dátovej) pamäte. Niektoré typy DSP používajú pre zásobník špeciálnu nezávislú pamäť (tzv. **hardvérový zásobník**). Ak je zásobník vytváraný v štandardnej pamäti, potom hovoríme o **softvérovom zásobníku**. Využitie zásobníka na odovzdávanie parametrov medzi volajúcou a volanou funkciou patrí medzi štandartné mechanizmy, ktoré sú používané v kompilátoroch jazyka C. Niektoré implementácie kompilátorov môžu **pri menšom počte** odovzdávaných parametrov použiť **namiesto zásobníka priamo vyhradené registre**. Týmto spôsobom je možné zjednodušiť manažment zásobníka a zrýchliť tak výsledný program.

typický príklad knižničnej funkcie pre ČSS. Knižničné funkcie pre IIR filter a FFT budú v ďalších cvičeniach analyzované hlavne z aplikačného pohľadu.

4.2 KNIŽNIČNÁ FUNKCIA FIR FILTRA FIR_FR16

Štruktúra implementovaného FIR filtra, ktorý je realizovaný funkciou *fir_fr16()* je zobrazená na obrázku 1. FIR filter je definovaný rádom filtra, jeho koeficientmi a obsahom (stavom)[2] oneskorovacej linky.



Obr.1 Štruktúra transverzálnej realizácie FIR filtra

Funkcia *fir_fr16()* je súčasťou knižnice LIBDSP prostredia VisualDSP++. Zdrojové kódy aktuálnych knižničných funkcií (v asembleri) pre procesory Blackfin sú v
..\VisualDSP 4.5\Blackfin\lib\src\libdsp

Kompletný zdrojový kód knižničnej funkcie *fir_16asm()* zo staršej knižnice LIBDSP verzie VisualDSP 4.0 je v prílohe. Význam jednotlivých inštrukcií a štruktúra programu **budú podrobne analyzované počas cvičenia**. Aj bez detailnej analýzy je však z uvedeného kódu vidieť niektoré základné vlastnosti:

a) Kód v knižničnej funkcii spracováva odlišne párny a nepárny počet vstupných vzoriek a tiež FIR filter s párnym a nepárnym počtom koeficientov. Knižničná funkcia je optimalizovaná tak, aby maximálne využila prítomnosť dvoch MAC jednotiek, ktoré môžu pracovať paralelne. Kód knižničnej funkcie je optimalizovaný pre všetky možnosti a je tak podstatne dlhší ako kód pre DSP s jednou MAC jednotkou. V prípade, že by bolo zaručené, že funkcia bude spracovávať napr. len párny počet vstupných vzoriek, prípadne, že FIR filtre budú mať vždy párny počet koeficientov (v prípade potreby užívateľom doplnený o nulový koeficient), je možné kód funkcie výrazne skrátiť.

b) Kód obsahuje časti kódu, ktoré sú realizované podmieneným prekladom časti kódu ako napr.

[2] **Stav oneskorovacej linky** FIR filtra v diskrétnom čase n obsahuje vstupné vzorky FIR filtra v predchádzajúcich časových okamihoch $x(n-1)$, $x(n-2)$, $\ldots x(n-N+1)$. Ich uložením v štruktúre **fir_state_fr16** je možné v programe jednoducho realizovať FIR filtráciu blokovým spôsobom.

```
#if defined(__WORKAROUND_CSYNC) || defined(__WORKAROUND_SPECULATIVE_LOADS)
```

alebo

```
#if defined(__WORKAROUND_BF532_ANOMALY38__)
```

Knižničná funkcia tak môže byť preložená v závislosti na definícii príslušných kľúčových slov. Prostredie VisualDSP++ obsahuje takto predkompilované knižnice. Programátor môže v menu projektu

Project-> Project Options->Project

okrem typu cieľového procesora Blackfin definovať aj tzv. **revíziu (Revision)** čipu [1]. V prípade potreby je možné presne definovať aj konkrétne **úpravy (Workarounds)** v menu

Project-> Project Options->Compile->Workarounds

Analog Devices (podobne ako iní výrobcovia čipov) priebežne vylepšuje návrh čipov s cieľom eliminovať predchádzajúce konštrukčné chyby. Konkrétny procesor má na svojom čipe aj číslo aktuálnej revízie. Aby bolo možné používať bez problémov aj staršie revízie čipu, je možné používať verzie knižníc, ktoré tieto chyby eliminujú (typicky na úkor zníženia efektivity výsledného kódu). Zdrojové kódy knižníc preto obsahujú časti kódu pre všetky revízie čipov, ktoré môžu mať vplyv na správnu činnosť kódu. Za výber správnej revízie čipu je zodpovedný programátor. Vývojové dosky používané na cvičeniach používajú procesor s označením

Analog Devices
ADSP-BF5333
SKBC600
64 12.99. 1 **0.4** <- číslo revízie čipu

t.j. revíziu 0.4.

c) Úvodná časť kódu

```
[--SP]=(R7:4,P5:4);      //PUSH R7-R4,P5-P4 ON STACK
P5=[SP+36];              // ADDRESS OF FILTER STRUCTURE
```

zabezpečí uloženie registrov **R7:4, P5:4** (tzv. **Call Preserved Registers**) na softvérový zásobník. V ďalšej časti kódu funkcia môže uvedené registre bez obmedzenia modifikovať, pretože tieto registre sú pred návratom z funkcie zo zásobníka obnovené

```
(R7:4,P5:4)=[SP++];    //POP R7-R4,P5-P4 FROM STACK
RTS;
```

Kód funkcie využíva aj ďalšie R registre

R0,R1,R2,R3

ktoré sú modifikované bez uloženia na zásobník. Tieto registre patria do skupiny tzv. **Scratch Registers**, ktoré je možné v tele C funkcie modifikovať bez obmedzenia a nie je potrebné ich obnovovať. Do tejto skupiny patria aj registre

P0,P1,P2
LB0,LB1
LC0,LC1
LT0,LT1
ASTAT
A0,A1
I0,I1,I2,I3
B0,B1,B2,B3
M0,M1,M2,M3

Do skupiny tzv. **Dedicated Registers** patria registre

SP (P6) - ukazovateľ zásobníka (Stack Pointer)
FP (P7) - ukazovateľ rámca (Frame Pointer)
L0,L1,L2,L3 - definujú dĺžku kruhových bufrov (Circular Buffers)

Podrobnejšie informácie o využití jednotlivých registrov v rámci tzv. C++ Run-Time Model and Environment je možné nájsť v [2].

d) Funkcia *fir_fr16()* má prototyp

```
void fir_fr16(const fract16 x[],fract16 y[],int n,fir_state_fr16 *s);
```

Prvé tri parametre sú z dôvodu efektívnosti presunuté v registroch R0, R1 a R2, štvrtý parameter – adresa štruktúry *fir_state_fr16 *s* je odovzdávaná cez zásobník [2]. Volajúca funkcia ukladá adresu na aktuálnu pozíciu zásobníka a volaná funkcia *fir_fr16()* ju vyčítava do registra P5 pomocou kódu

```
[--SP]=(R7:4,P5:4);     //PUSH R7-R4,P5-P4 ON STACK
P5=[SP+36];             // ADDRESS OF FILTER STRUCTURE
```

e) Kód funkcie *fir_fr16()* využíva kruhové bufre (modulo adresovanie) adresované pomocou DAG (Data Address Generator) jednotiek, ktoré umožňuje výrazné zefektívnenie kódu. Tento spôsob adresovanie je pre DSP typický. Nastavenie modulo adresovania je realizované zápisom dĺžky bufra do zodpovedajúcich L registrov jednotiek DAG, čo je realizované v úvodnej časti kódu.

Pred návratom z funkcie *fir_fr16()* je nevyhnutné nastaviť použité registre L späť na hodnotu 0, ktorá je vyžadovaná C++ Run-Time modelom. Hodnoty L=0 zabezpečia štandardné lineárne adresovanie DAG jednotiek.

4.3 TESTOVANIE FIR FILTRA S VYUŽITÍM IO FUNKCIÍ

Pri praktickom testovaní algoritmov je často potrebné overiť ich funkčnosť pre väčšiu množinu vstupných dát. Napr. testovanie funkcie *fir_fr16()* v predchádzajúcom cvičení je nedostatočné. Keďže bol spracovaný **len jeden** dátový blok, nebolo overené, či funkcia *fir_fr16()*, korektne uloží pred návratom všetky informácie o aktuálnom stave filtra do štruktúry *fir_state_fr16 state* a teda či následné ďalšie volania funkcie *fir_fr16()* **pre ďalšie** dátové bloky budú vracať korektné výsledky. Je teda potrebné simulovať spracovanie minimálne dvoch dátových blokov.

V praxi existujú aj aplikácie, kde je potrebné pri simulácií spracovávať veľké množstvo dát. Ako typický príklad je možné napr. uviesť analýzu zaokrúhľovacích chýb algoritmov ČSS. Takáto analýza typicky vyžaduje spracovanie veľkého množstva dát.

V ďalšej časti budú pri simulácii využité štandardné IO funkcie *read()* a *write()* dostupné v prostredí VisualDSP++.

4.3.1 IO FUNKCIE READ() A WRITE()

Prostredie VisualDSP++ umožňuje využitie niektorých IO funkcií z knižnice stdio. Tieto funkcie umožňujú realizovať čítanie a zápis do externých súborov uložených v PC

(s využitím tzv. Default Device Interface). Tento mechanizmus je možné využiť nielen v simulátore prostredia VisualDSP++, ale je pri ladení s vývojovými doskami EZ-Kit.

IO knižnice je možné tiež presmerovať do užívateľom definovaných periférii (napr. na rozhranie UARTu, s využitím mechanizmu rozšírenia o nové IO rozhrania). Podrobnejšie informácie je možné nájsť v [2].

V ďalšej simulácii budú využité tri štandardné funkcie, ktorých prototypy sú definované v hlavičkovom súbore <stdio.h>:

```
FILE * fopen(const char *, const char *);
size_t fread(void *, size_t, size_t, FILE *);
size_t fwrite(const void *, size_t, size_t, FILE *);
```

4.3.2 TESTOVANIE FIR FILTRA S VYUŽITÍM EXTERNÝCH DÁTOVÝCH SÚBOROV

Pri ladení algoritmov ČSS je výhodné ak výstup simulácie je možné porovnať napr. s **vysoko-úrovňovou simuláciou** napr. v Matlabe[3]. Pre tento účel je výhodné ak je možné pripojiť k simulátoru **vstupné a výstupné súbory**, ktoré budú simulátorom čítané resp. do ktorých simulátor bude zapisovať výstupné hodnoty. Je výhodné ak formát týchto súborov je **prenositeľný** aj do iných systémov (napr. Matlabu).

Súčasťou projektu **spvt_4cv.zip [3]** je **binárny súbor x.dat** so vstupnými vzorkami pre simuláciu. Každá vstupná vzorka je reprezentovaná 2 bajtmi a súbor x.dat bol generovaný v Matlabe pomocou m-súboru[4]:

```
% generuje data v binarnom formate (x.dat) pre overenie presnosti vypoctu FIR filtra
% v prostredi VisualDSP++

SCALE = 2^15;                % konverzia z 1.15 na 16.0
DATASIZE = 256*10            % pocet generovanych vzoriek

Fvz = 48000;                 % vzorkovacia frekvencia
A1 = 0.33;                   % amplituda 1. harmonickeho signalu
F1 = 2000;                   % frekvencia 1. harmonickeho signalu
A2 = 0.33;                   % amplituda 2. harmonickeho signalu
F2 = 8000;                   % frekvencia 2. harmonickeho signalu
A3 = 0.33;                   % amplituda 3. harmonickeho signalu
F3 = 10000;                  % frekvencia 3. harmonickeho signalu
n=(1:DATASIZE);              % diskretny cas

x = A1*sin(2*pi*F1*n/Fvz) + A2*sin(2*pi*F2*n/Fvz) + A3*sin(2*pi*F3*n/Fvz);
x = round( x*SCALE)';        % konverzia do formatu 16.0

outFile = fopen('x.dat','wb');
save x.txt x -ascii          % ulozenie vzoriek do suboru v ASCII formate (pre referencny vypocet)
fwrite(outFile,x,'short');   % ulozenie vzoriek v binarnom formate (2 bajty/vzorku) pre VisualDSp++
fclose(outFile);
```

Testovaný FIR filter bol navrhnutý v prostredí Filter Express na základe špecifikácie:

[3] Súčasťou projektu je aj testovací súbor firtest.m, pomocou ktorého je možné porovnať presnosť výpočtu FIR filtra v 16-bitovom procesore ADSP Blackfin a výpočtom FIR filtra v prostredí MATLAB s presnosťou 64 bitov.

[4] Súbor x.dat reprezentuje 16-bitové vzorky vstupného signálu zloženého z 3 sínusových signálov s rovnakými amplitúdami a frekvenciami zvolenými tak, že jedna zložka je z pásma priepustnosti testovaného FIR filtra.

FIR Filter	Požiadavky:
Equiripple	Pass Band 0.3 dB
BandPass	Stop Band 65 dB
Fvz = 48 kHz	F1 = 5 kHz F2 = 6 kHz F3 = 9 kHz F4 = 10 kHz

pričom po návrhu a kvantovaní do formátu 1.15 boli dosiahnuté hodnoty

Počet koeficientov	122
Pass Band	0.4507 dB
Stop Band	59.731 dB

Hlavný program *fir.c* je modifikáciou kódu z predchádzajúceho cvičenia, ktorý využíva knižničné funkciu *fir_fr16()* a *fir_init()*. Prístup k binárnym súborom **x.dat** a **y.dat** je realizovaný štandardným spôsobom pomocou funkcií *fopen()*, *read()* a *write()*:

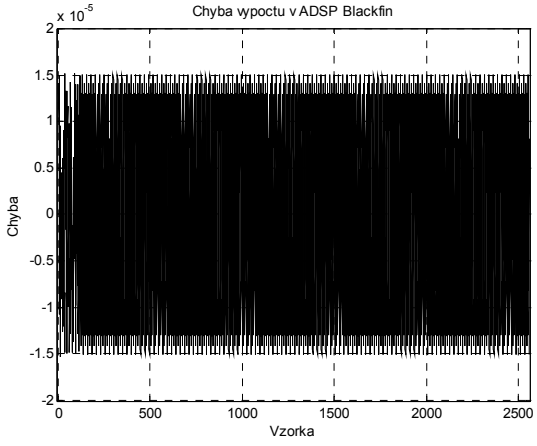
```
#include <stdio.h>
#include <fract.h>
#include <filter.h>

#define NUM_VEC     10      // pocet spracovavanych blokov
#define VEC_SIZE    256     // velkost spracovavaneho bloku
#define NUM_TAPS    122     // pocet koeficientov FIR filtra

fract16 coefs[NUM_TAPS] = {
        #include "coefs.h"      // subor s koeficientmi FIR filtra
};

fract16  delay[NUM_TAPS];
fir_state_fr16 state;           // declare filter state
fract16 in[VEC_SIZE];
fract16 out[VEC_SIZE];

FILE *inFile, *outFile;

int readInput(short *inBuf,int count) {
        int wordsRead=0;
        wordsRead = fread(inBuf,sizeof(short),count,inFile);
        return wordsRead;
}

int writeOutput(short *outBuf,int count) {
        int wordsRead=0;
        wordsRead = fwrite(outBuf,sizeof(short),count,outFile);
        return wordsRead;
}

void main( void ){
        int i;

        inFile = fopen("D:\\SPvT\\x.dat","rb");          // nastavena aktualna cesta k testovanym suborom
        outFile = fopen("D:\\SPvT\\y.dat","wb");         // (zmenit podla aktualnej struktury adresarov)

        fir_init(state, coefs, delay, NUM_TAPS, 1);      // inicializacia stavu filtra
        i = NUM_VEC;
        printf("Zaciatok testu\n");
        while( i-- ) {
                readInput(in, VEC_SIZE);                 // nacitanie vstupneho bloku
                fir_fr16(in, out, VEC_SIZE, &state);     // filtracia vstupneho bloku
                writeOutput(out,VEC_SIZE);               // zapis vystupneho bloku
        }
        printf("Koniec testu");
        fclose( inFile );
        fclose( outFile );
}
```

Počas cvičenia bude vykonaná kompletná simulácia programu fir.c so vstupnými vzorkami zo súboru **x.dat**, ktoré sú súčasťou projektu.

Nasledujúci jednoduchý program v Matlabe umožňuje porovnať výsledky zo simulácie ADSP Blackfin s referenčným výpočtom FIR filtra v Matlabe.

```
% Subor testuje vystup FIR filtra vypocitany v prostredi VisualDSP

SCALE = 2^15;
DATASIZE = 256*10              % pocet testovanych vzoriek

inFile = fopen('x.dat','rb');          % subor so vstupnymi vzorkami (format 16.0)
[x,cnt] = fread(inFile,DATASIZE,'short');
x = x/SCALE;                % prevod do formatu 1.15

outFile = fopen('y.dat','rb');         % subor s vystupnymi vzorkami z VisualDSP simulacie (format 16.0)
[y,cnt] = fread(outFile,DATASIZE,'short');
y = y/SCALE;                           % prevod do formatu 1.15

load coefs.txt;                        % koeficienty FIR filtra vo formate (1.15)
h = coefs;
N = length(h);                         % pocet koeficientov FIR filtra

y_ref = filter(h,1,x);                 % referencny vypocet FIR filtra v Matlabe
e = y - y_ref;                         % rozdiel medzi VisualDP a Matlabom

plot(e);                               % max chyba by mala byt radovo 2^(-15)
title('Chyba vypoctu v ADSP Blackfin');
xlabel('Vzorka');
ylabel('Chyba');
grid;
```

Na obrázku 2 je zobrazená chyba výpočtu FIR filtra v procesore ADSP Blackfin, pričom všetky údaje boli transformované do formátu 1.15. Z obrázku je zrejmé, že maximálna chyba pre daný vstupný signál je menšia ako 1 LSB $\doteq 2^{-15} = 3.1*10^{-5}$.



Obr.2 Chyba výpočtu FIR filtra v procesore ADSP Blackfin

Príklad

Realizujte kompletnú simuláciu FIR filtra pomocou funkcie fir_fr16()_a overte správnosť údajov na Obr. 2. Sledujte výstup funkcie printf(), ktorý je presmerovaný do výstupného okna prostredia VisualDSP++.

Príklad
Určite rýchlosť výpočtu funkcie fir_fr16() pre párny a nepárny počet vstupných vzoriek a koeficientov FIR filtra. Asymptoticky by mal FIR filter v jadre DSP Blackfin na spracovanie jedného bloku dosahovať rýchlosť $\sim N_{FIR}N_{block}/2 = \sim 122*256/2 = 15616$ *cyklov/blok. Namerané výsledky sú výrazne odlišné.* ***Prečo?***

Príklad
Zopakujte predchádzajúci test po pridaní nasledujúcich riadkov (zvýraznený tučným písmom):

```
static segment("L1_data_b")
fract16 coefs[NUM_TAPS] = {
#include "coefs.h"              // subor s koeficientmi FIR filtra
};

static segment("L1_data_a")
fract16 delay[NUM_TAPS];
```

Ako a prečo sa zmenia výsledky?

4.3.3 PREDKOMPILOVANÁ SIMULÁCIA

Predchádzajúce simulácie v prostredí **klasického simulátora** VisualDSP++ boli pomerne pomalé. VisualDSP++ umožňuje zrýchliť simulácie s využitím tzv. predkompilovanej simulácie (Compiled Simulation). Klasický simulátor využíva v počítači PC **opakované** dekódovanie a interpretovanie simulovaných inštrukcií. Predkompilovaná simulácia zrýchľuje simuláciu predkompilovaním celého simulovaného kódu, ktorý už počas simulácie nie je potrebné opakovane dekódovať, čo umožňuje výrazné zrýchlenie simulácie[5]. Nevýhodou predkompilovanej simulácie je obmedzená podpora simulačných vlastností prostredia VisualDPS++[6].

Predkompilovanú simuláciu je možné zvoliť v menu
Session\Select Sesion\ADSP-BF533 Blackfin Family Compiled Simulation

Ak ešte v prostredí VisualDSP++ predkompilovaná simulácia nebola využívaná, je potrebné zvoliť ju pre konkrétny cieľový procesor v menu
Session\New Session

Príklad
Porovnajte rýchlosť predkompilovanej simulácie a klasickej simulácie pri testovaní FIR filtra s využitím externých dátových súborov.

[5] Aj keď moderné počítače PC sú pomerne výkonné, rýchlosť DSP sa tiež výrazne zvyšuje. Kompletná simulácia napr. DSP zo 600 MHz taktovaním tak môže byť pri zložitejších simuláciách pomerne zdĺhavá.

[6] Nie je napr. podporovaný zápis resp. čítanie do/z rozhrania UART pomocou tzv. „streams“, ktoré budú využívané v nasledujúcich cvičeniach.

4.4 ĎALŠIE KNIŽNIČNÉ FUNKCIE PRE FIR FILTRÁCIU

Súčasťou knižnice LIBDSP sú knižničné funkcie pre decimáciu a interpoláciu[7] pomocou FIR filtrov:

```
void fir_decima_fr16(const fract16 x[], fract16 y[], int  N,  fir_state_fr16  *s);
void fir_interp_fr16(const fract16 x[], fract16 y[], int  N,  fir_state_fr16  *s);
```

Prototypy uvedených funkcií sú v hlavičkovom súbore <filter.h> a zdrojové kódy knižničných funkcií (v asembleri) pre procesory Blackfin sú v

..\VisualDSP 4.5\Blackfin\lib\src\libdsp

podobne ako pre funkciu *fir_fr16()*. Všetky funkcie pre FIR filtráciu využívajú štruktúru *fir_state_fr16* opísanú v predchádzajúcej časti.

4.5 ZÁVER

Implementovaný FIR filter vyžaduje na realizáciu FIR filtra s N koeficientmi približne[8] $N/2$ inštrukčných cyklov. Signálové procesory rodiny ADSP BF533-600, ktoré budú využívané na cvičeniach, môžu pracovať na frekvencii 600 MHz a realizovať 1200 MAC/s. Tieto hodnoty určujú maximálne frekvencie vzorkovania F_{vz} resp. pri danej frekvencii vzorkovania maximálny rád FIR filtra N, ktorý je možné implementovať. Napr. pre FIR filter s $N = 1000$ koeficientmi (čo už je pomerne zložitý FIR filter) je možné s procesorom, ktorý realizuje 1200 MAC/s pracovať až do

$$F_{vz} \approx \frac{600 \times 10^6}{N/2} = \frac{1200 \times 10^6}{1000} = 1,2\,[MHz] \tag{4.1}$$

čo jasne dokumentuje vysokú výkonnosť architektúry ADSP Blackfin pri realizácii tohto základného algoritmu ČSS.

LITERATÚRA

[1] ADSP-BF533 Blackfin Processor Anomaly List. Revision R-October 21, 2005, pp.1-54, www.analog.com.

[2] VisualDSP++ 4.5 C/C++ Compiler and Library Manual for Blackfin Processors. Analog Devices, Inc., 2006, pp.1-281.

[7] Decimácia je proces, pri ktorom je znížená vzorkovacia frekvencia spracovávaného signálu. Decimačný FIR filter zabezpečí, že nedochádza k prekrytiu spektra. Interpolačný filter naopak zvyšuje vzorkovaciu frekvenciu spracovávaného signálu pričom interpolačný FIR filter zabezpečí zachovanie pôvodného spektra vstupného signálu. Knižničné funkcie podporujú len celočíselné decimačné a interpolačne faktory.

[8] Vzťah platí dostatočne presne pre veľké hodnoty N a optimálne umiestnenie bufrov (pre stavy filtra a jeho koeficienty) v oddelených interných datových pamätiach.

[3] http://www.kemt.fei.tuke.sk/Predmety/KEMT412_SPvT/_materialy/Cvicenia/spvt_4cv.zip

PRÍLOHA

```
/*****************************************************************************
C/opyright(c) 2000-2004 Analog Devices Inc.
*****************************************************************************
File Name         : fir_fr16.asm
Function Name  : __fir
Module Name    : FILTER Library
Description       : This function performs FIR filter operation on given input.
Operands          : R0 - Address of input vector,
                    R1 - Address of output vector,
                    R2 - Number of input elements
                    Filter structure is on stack.

Prototype:      void fir(const fract16 x[],fract16 y[],int n,fir_state_fr16 *s);

Structure of fir_state_fr16:
          {
               fract16 *h,              // filter coefficients
               fract16 *d,              // start of delay line
               fract16 *p,              // read/write pointer
               int k,                   // no. of coefficients
               int l                    // interpolation/decimation index
          } fir_state_fr16;

Registers used :

     R0, R1, R2, R3, R3, R4, R5, R6, R7

     I0 -> Address of delay line (used for updating the delay line)
     I1 -> Address of delay line (used for fetching the delay line data)
     I2 -> Address of filter coeff. h0, h1 , ... , hn-1
     I3 -> Address of output array y[]

     P0 -> Address of input array x[]
     P1, P2, P4
     P5 -> Address of structure s


     Code size: 354 bytes
_____________________________________________________________________________
     Computation time:

          a) Even Taps, Even no. inputs   : 64 + Ni/2*(3 + Nc)
            For Ni=256, L=8                    : 1472

          b) Odd Taps, Even no. inputs    : 70 + Ni/2*(3 + Nc)
            For Ni=256, L=9                    : 1606

          c) Even Taps, Odd no. inputs    : 64 + (Ni-1)/2*(3 + Nc) + Nc
            For Ni=257, L=8                    : 1480

          d) Odd Taps, Odd no. inputs          : 73 + (Ni-1)/2*(3 + Nc) + Nc
            For Ni=257, L=9                    : 1618
_____________________________________________________________________________

*****************************************************************************/

#if defined(__ADSPLPBLACKFIN__) && defined(__WORKAROUND_AVOID_DAG1)
#define __WORKAROUND_BF532_ANOMALY38__
#endif

.section program;
.global    __fir_fr16;
.align 2;
__fir_fr16:
     [--SP]=(R7:4,P5:4);          //PUSH R7-R4,P5-P4 ON STACK
     P5=[SP+36];                  // ADDRESS OF FILTER STRUCTURE
```

```
        P0=R0;                  // ADDRESS OF INPUT ARRAY
        I3=R1;                  // ADDRESS OF OUTPUT ARRAY
        CC=R2<=0;               // CHECK IF NUMBER OF INPUT ELEMENTS<=0
        P1=[P5++];              // POINTER TO FILTER COEFFICIENTS
        P2=[P5++];              // POINTER TO DELAY LINE
        P4=[P5++];              // ADDRESS TO READ/WRITE POINTER
        R1=[P5--];              // NUMBER OF COEFFICIENTS

        IF CC JUMP _fir_fr16_RET_END;
        CC=R1<=0;               // CHECK IF NUMBER OF COEFF. <=0
        IF CC JUMP _fir_fr16_RET_END;
        R5=R1;                  // STORE NUMBER OF FILTER COEFF. IN R5
        R1=R1+R1;               // DOUBLE R1 TO INCREMENT AS HALF WORD
        I2=P1;                  // INITIALISE I2 TO FILTER COEFF. ARRAY
        B2=P1;                  // MAKE FILTER COEFFS. AS CIRCULAR BUFFER
        L2=R1;
        I1=P4;                  // INDEX TO READ/WRITE POINTER
        B1=P2;                  // INITIALISE B1 AND L1
        L1=R1;                  // FOR  DELAY LINE CIRCULAR BUFFER
        I0=P4;                  // INDEX TO READ/WRITE POINTER
        B0=P2;                  // INITIALISE B0 AND L0
        L0=R1;                  // FOR  DELAY LINE CIRCULAR BUFFER

        P1=R2;                  // SET OUTER LOOP COUNTER
        P2=R5;                  // SET INNER LOOP COUNTER
        L3 = 0;

        R6=PACK(R2.H,R2.L) || I1+=4;
        CC=R6==1;

        IF CC JUMP _fir_fr16_SING_SAMP;

        CC=BITTST(R5,0);        // Check if the number of filter taps are odd
        P2+=-2;                 // Nc-2

        IF CC JUMP _fir_fr16_FIR_ODD;   //Odd tap FIR

#if defined(__WORKAROUND_CSYNC) || defined(__WORKAROUND_SPECULATIVE_LOADS)
        NOP;
        NOP;
        NOP;
#endif

/*****************************************************************************/
        /** Even number of input samples **/
        /** Even number of filter taps   **/

#if defined(__WORKAROUND_BF532_ANOMALY38__)

        /* ---------- Start of BF532 Anomaly#38 Safe Code ----------          **
        **                                                                    **
        ** which implements the IIR filter for an even number of samples     **
        ** and an even number of filter coefficients                          **
        **                                                                    */

        I0+=2 || R2=[I2++];
        R0=[P0++];
        I1+=2;

        LSETUP(_fir_fr16_E_FIR_START,_fir_fr16_E_FIR_END) LC0=P1>>1;
        // Loop 1 to Ni/2
_fir_fr16_E_FIR_START:
            A1=R0.H*R2.L,A0=R0.L*R2.L ||I1-=2 || W[I0--]=R0.L;
            R4=PACK(R0.H,R4.L) || R0.H=W[I1++];

            LSETUP(_fir_fr16_E_MAC_ST,_fir_fr16_E_MAC_END) LC1=P2>>1;
            //Loop 1 to Nc-1/2
_fir_fr16_E_MAC_ST:   R2.L=W[I2++];
                      A1+=R0.L*R2.H,A0+=R0.H*R2.H || R0.L=W[I1++];
                      R2.H=W[I2++];
_fir_fr16_E_MAC_END: A1+=R0.H*R2.L,A0+=R0.L*R2.L || R0.H=W[I1++];

            R3.H=(A1+=R0.L*R2.H),R3.L=(A0+=R0.H*R2.H) || R0=[P0++] || W[I0--]=R4.H;
_fir_fr16_E_FIR_END:
            R2=[I2++] || [I3++]=R3;
```

```
#else  /* End of BF532 Anomaly#38 Safe Code */

	/* ---------- Start of NON BF532 Anomaly#38 Safe Code ---------	**
	**                                                              **
	** which implements the IIR filter for an even number of samples	**
	** and an even number of filter coefficients                    **
	**                                                              */

	I0+=2 || R0.L=W[I1++];
	R0=[P0++] || R2=[I2++];

	LSETUP(_fir_fr16_E_FIR_START,_fir_fr16_E_FIR_END) LC0=P1>>1;
	// Loop 1 to Ni/2
_fir_fr16_E_FIR_START:
		A1=R0.H*R2.L,A0=R0.L*R2.L ||I1-=2 || W[I0--]=R0.L;
		R4=PACK(R0.H,R4.L) || R0.H=W[I1++] || NOP;

		LSETUP(_fir_fr16_E_MAC_ST,_fir_fr16_E_MAC_END) LC1=P2>>1;
		//Loop 1 to Nc-1/2

_fir_fr16_E_MAC_ST:    A1+=R0.L*R2.H,A0+=R0.H*R2.H || R2.L=W[I2++] || R0.L=W[I1++];
_fir_fr16_E_MAC_END: A1+=R0.H*R2.L,A0+=R0.L*R2.L || R0.H=W[I1++] || R2.H=W[I2++];
		R3.H=(A1+=R0.L*R2.H),R3.L=(A0+=R0.H*R2.H) || R0=[P0++] || W[I0--]=R4.H;
_fir_fr16_E_FIR_END:
		R2=[I2++] || [I3++]=R3;

#endif		/* End of Alternative to BF532 Anomaly#38 Safe Code */

	R0.L=W[I0--] || I2-=4;
	R0=I0;
	R2.L=W[I1--];
	P0+=-4;

	JUMP _fir_fr16_DATA;

/*****************************************************************************/
		/** Even number of input samples **/
		/** Odd number of filter taps    **/

_fir_fr16_FIR_ODD:

	B3=I3;
	R2>>=1;
	R2=R2 << 2 || R6.H=W[I1++];
	L3=R2;
	R5=[P0];

#if defined(__WORKAROUND_BF532_ANOMALY38__)

	/* ---------- Start of BF532 Anomaly#38 Safe Code ----------	**
	**                                                              **
	** which implements the IIR filter for an even number of samples	**
	** and an odd number of filter coefficients                     **
	**                                                              */

	R2=[I2++] || I3-=4;
	R3=[I3];

	LSETUP(_fir_fr16_O_FIR_START,_fir_fr16_O_FIR_END) LC0=P1>>1;
	//Loop 1 to Ni/2
_fir_fr16_O_FIR_START:
		R4.H=W[I0] || W[I0++]=R5.H;
		R0=PACK(R6.H,R5.L) || R7=[P0++];
		A1=R5.H*R2.L,A0=R5.L*R2.L || R5=[P0] || W[I0--]=R5.L;

		LSETUP(_fir_fr16_O_MAC_ST,_fir_fr16_O_MAC_END) LC1=P2>>1;
		//Loop 2 to Nc/2-1
_fir_fr16_O_MAC_ST:   A1+=R0.L*R2.H,A0+=R0.H*R2.H || R2=[I2++];
			R0.L=W[I1++];
_fir_fr16_O_MAC_END: A1+=R0.H*R2.L,A0+=R0.L*R2.L || R0.H=W[I1++];

		A1+=R0.L*R2.H,A0+=R0.H*R2.H || R2.H=W[I2++] || I0-=4;
		//Done only for Odd number of coeffs
		R3.L=(A0+=R4.H*R2.H) || R6.H=W[I1++] ||  [I3++]=R3 ;
```

```
_fir_fr16_O_FIR_END:
            R3.H=(A1+=R0.H*R2.H) || R2=[I2++];

#else               /* End of BF532 Anomaly#38 Safe Code */

      /* ---------- Start of NON BF532 Anomaly#38 Safe Code ----------      **
      **                                                                    **
      ** which implements the IIR filter for an even number of samples     **
      ** and an odd number of filter coefficients                          **
      **                                                                    */

      R4.H=W[I0] || I3-=4;
      R3=[I3] || R2=[I2++];

      LSETUP(_fir_fr16_O_FIR_START,_fir_fr16_O_FIR_END) LC0=P1>>1;
      //Loop 1 to Ni/2
_fir_fr16_O_FIR_START:
            R0=PACK(R6.H,R5.L) || R7=[P0++] || W[I0++]=R5.H ;
            A1=R5.H*R2.L,A0=R5.L*R2.L || R5=[P0] || W[I0--]=R5.L;

            LSETUP(_fir_fr16_O_MAC_ST,_fir_fr16_O_MAC_END) LC1=P2>>1;
            //Loop 2 to Nc/2-1
_fir_fr16_O_MAC_ST:    A1+=R0.L*R2.H,A0+=R0.H*R2.H || R2=[I2++] || R0.L=W[I1++];
_fir_fr16_O_MAC_END:  A1+=R0.H*R2.L,A0+=R0.L*R2.L || R0.H=W[I1++];

            A1+=R0.L*R2.H,A0+=R0.H*R2.H || R2.H=W[I2++] || I0-=4;
            //Done only for Odd number of coeffs
            R3.L=(A0+=R4.H*R2.H) || R6.H=W[I1++] ||  [I3++]=R3 ;
_fir_fr16_O_FIR_END:
            R3.H=(A1+=R0.H*R2.H) || R2=[I2++] || R4.H=W[I0];

#endif              /* End of Alternative to BF532 Anomaly#38 Safe Code */

      L3 = 0;
      R0 = I0;
      I2-=4;
      [I3++] = R3 || I1-=2;

/*****************************************************************************/

_fir_fr16_DATA:
      CC=BITTST(R6,0);
      IF !CC JUMP _fir_fr16_RET_END1;
      //If even number of input samples, jump to ret_end

/*****************************************************************************/
            /** Process the last sample separately **/
            /** for an odd number of input samples **/

      P2+=2;

_fir_fr16_SING_SAMP:

#if defined(__WORKAROUND_BF532_ANOMALY38__)

      /* ---------- Start of BF532 Anomaly#38 Safe Code ----------          **
      **                                                                    **
      ** which processes the last sample separately for an odd number of   **
      ** number of input samples                                           **
      **                                                                    */

#if defined(__WORKAROUND_CSYNC) || defined(__WORKAROUND_SPECULATIVE_LOADS)
      NOP;
      NOP;
      NOP;
#endif

            A1=A0=0 || R0=W[P0] (Z);
            R4 = PACK(R0.H,R0.L) || R2.H=W[I0++];

            LSETUP(_fir_fr16_L_MAC_ST,_fir_fr16_L_MAC_END)LC1=P2;//Loop 1 to Nc
_fir_fr16_L_MAC_ST:     R2.L=W[I2++];
_fir_fr16_L_MAC_END:   R3.L=(A0+=R0.L*R2.L) || R0.L=W[I1++];

#else               /* End of BF532 Anomaly#38 Safe Code */
```

```
        /* ---------- Start of NON BF532 Anomaly#38 Safe Code ----------          **
        **                                                                  **
        ** which processes the last sample separately for an odd number of    **
        ** number of input samples                                          **
        **                                                                  */

        A1=A0=0 || R0=W[P0] (Z) || R2.L=W[I2++];
        R4 = PACK(R0.H,R0.L) || R2.H=W[I0++];

        LSETUP(_fir_fr16_L_MAC_ST,_fir_fr16_L_MAC_ST)LC1=P2;//Loop 1 to Nc
_fir_fr16_L_MAC_ST:   R3.L=(A0+=R0.L*R2.L) || R2.L=W[I2++] || R0.L=W[I1++];

#endif              /* End of Alternative to BF532 Anomaly#38 Safe Code */

        W[I0--]=R4.L;
        W[I3++]=R3.L || I0-=2;
        R0=I0;

_fir_fr16_RET_END1:
        [P5]=R0;

/*****************************************************************************/

_fir_fr16_RET_END:
        L0=0;
        L1=0;
        L2=0;
        (R7:4,P5:4)=[SP++];     //POP R7-R4,P5-P4 FROM STACK
        RTS;

.__fir_fr16.end:
```