# 7 IMPLEMENTÁCIA IIR FILTRA POMOCOU PROCESOROV ADSP BLACKFIN

Ciele cvičenia:

- prehľad knižničných funkcií prostredia VisualDSP++ pre IIR filtráciu,
- praktické obmedzenia štandardných knižničných funkcií z knižnice dsp_run_time_lib prostredia VisualDSP++,
- príklad **externej optimalizovanej** knižničnej funkcie pre IIR filtráciu,
- projekt pre blokové spracovanie AD a DA vzoriek pomocou „**ping-pong**“ metódy.

## *7.1 ÚVOD*

IIR filter je, vzhľadom na prítomnosť spätnej väzby, **zložitejšia štruktúra** ako FIR filter. Z predchádzajúcich cvičení vieme, že je výhodné prenosovú funkciu IIR filtra **rozložiť** na jednoduchšie sekcie druhého rádu - **bikvady**. Všeobecný bikvad je charakterizovaný pomocou **5 koeficientov** $b_0$, $b_1$, $b_2$, $a_1$, $a_2$. Prakticky všetky dostupné signálové procesory sú optimalizované aj pre implementáciu IIR bikvadu a procesory Blackfin firmy Analog Devices nie sú v tomto smere výnimkou. Podobne ako kódy pre implementáciu FIR filtra aj kódy pre IIR filter efektívne využívajú základné vlastnosti duálnej harvardskej architektúry, modulo adresovanie a využitie MAC inštrukcie.

Na knižničných funkciách, ktoré sú súčasťou VisualDSP++ knižnice [1] bude demonštrované, že dostupné štandardné knižničné funkcie nemusia vyhovovať pre mnohé praktické aplikácie.

Bude tiež demonštrovaná vysoko optimalizovaná externá knižničná funkcia [2], ktorá umožňuje pri blokovom spracovaní dosiahnuť rýchlosť spracovania blížiacu sa 2,5 cyklom/bikvad, čo je teoretické minimum pre DSP s duálnymi MAC jednotkami. Uvedená knižničná funkcia naviac eliminuje obmedzenia IIR funkcií knižnice VisualDSP++.

## *7.2 KNIŽNIČNÉ FUNKCIE PROSTREDIA VISUALDSP++ PRE IIR FILTRÁCIU*

Existuje niekoľko verzií implementácie IIR filtrov, ktoré sa líšia štruktúrou zapojenia a počtom koeficientov. Z teórie ČSS je známe, že v praktických implementáciách IIR filtrov je výhodné využívať realizácie IIR filtrov pomocou kaskádného zapojenia sekcií 2 rádu – bikvadov. Prenosové funkcie bikvadov získame z prenosovej funkcie IIR filtra

$$H_{IIR}(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} \tag{0.1}$$

rozkladom do tvaru (pre jednoduchosť predpokladáme, že $N$ je párne)

$$H_{IIR}(z) = \prod_{k=1}^{N/2} \left( \frac{b_{0k} + b_{1k} z^{-1} + b_{2k} z^{-2}}{1 + a_{1k} z^{-1} + a_{2k} z^{-2}} \right) = \prod_{k=1}^{N/2} H_k(z) \tag{0.2}$$

pričom na implementáciu kompletnej prenosovej funkcie $H_{IIR}(z)$ je využité kaskádne zapojenie $N/2$ bikvadov. Takýto rozklad veľmi často realizujú automaticky aj programy pre praktický návrh filtrov. Rozklad prenosovej funkcie na bikvady je veĺmi výhodný práve pre DSP, pretože umožňuje jednoduchým spôsobom eliminovať problémy reprezentácie koeficientov filtra v zlomkovom formáte. Z predchádzajúcich cvičení vieme, že prakticky využívané koeficienty bikvadu $a_{jk}$ môžu byť z intervalu

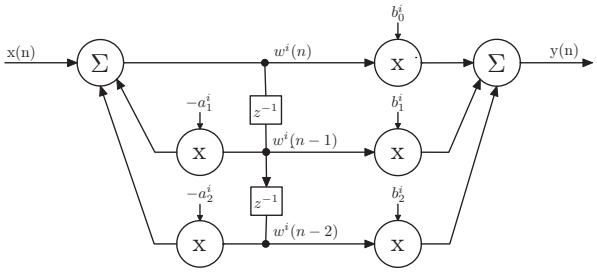
$$\left| a_{jk} \right| < 2.0 \qquad j = 1,2 \quad k = 1,2,\ldots,N/2 \tag{0.3}$$

a tak tieto koeficienty musia byť v prípade DSP s pevnou rádovou čiarkou (pretože interval čísel je u typických DSP obmedzený na interval $\langle -1,1 )$ ) uložené so zmenenou mierkou. V prípade bikvadov stačí všetky koeficienty $a_{jk}$ vydeliť mierkovou konštantou $2$. Po zmene mierky koeficientov je samozrejme potrebné zmeniť zodpovedajúcim spôsobom aj zdrojový kód bikvadu.

V DSP knižnici prostredia VisualDSP++ existujú dve funkcie na implementáciu IIR filtra:

```
void iir_fr16( const fract16 x[], fract16 y[], int n, iir_state_fr16 *s)
```

([1], str. 4-127), ktorá realizuje implementáciu pomocou bikvadov (0.2)[1]. Táto realizácia využíva priamu formu II (DF II) realizácie bikvadu[2], ktorá je znázornená na Obr.1.




*Obr.1 Priama forma II realizácie bikvadu využívaná vo funkcii iir_fr16()*

[1] Podľa manuálu [1] funkcia implementuje bikvady so zápornými znamienkami koeficientov v menovateli prenosovej funkcie bikvadov. V komentároch zdrojového kódu sú znamienka kladné ...

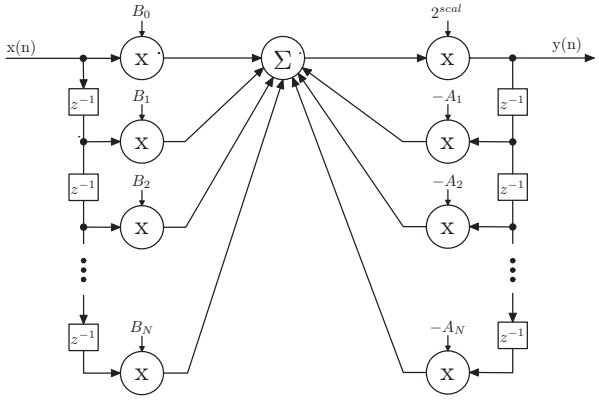
[2] Táto realizácie je tzv. kanonickou realizáciou, pretože obsahuje minimálny počet stavebných blokov (predovšetkým oneskorovacích liniek).

Funkcia je optimalizovaná z pohľadu rýchlosti tak, že spracováva paralelne pomocou duálnej MAC architektúry 2 vstupné vzorky a dosahuje rýchlosť $\sim 3$ cykly/vzoru, čo demonštruje pomerne efektívnu implementáciu[3] tejto funkcie. Významným praktickým obmedzením tejto knižničnej funkcie je skutočnosť, že **koeficienty prenosovej funkcie sú neškálované a teda musia byť z intervalu** $\langle -1,1)$ ! Množina realizovateľných IIR filtrov je tak výrazne obmedzená a nezahrňuje mnoho praktických IIR filtrov.

Druhou knižničnou funkciou je

```
void iirdf1_fr16(const fract16 x[],  frac16 y[],  int n, iirdf1_fr16_state *s)
```

([1], str. 4-129), ktorá realizuje IIR na základe prenosovej funkcie (0.1). Táto realizácia využíva priamu formu I (DF I) realizácie IIR filtra, ktorá je znázornená na Obr.2.




*Obr.2 Priama forma I realizácie IIR filtra využívaná vo funkcii iirdf1_fr16()*

Funkcia využíva koeficienty so zmenenou mierkou v tvare $A_1 = a_1 / S$ , $\ldots B_N = b_N / S$, pričom vhodne zvolený škálovací faktor $S$ zabezpečuje, že žiadny koeficient nemá hodnotu mimo zlomkového intervalu $\langle -1,1)$. Konverziu koeficientov do vhodnej formy realizuje funkcia

```
void coeff_iirdf1_fr16( const float acoeff[], const float bcoef[], fract16 coeff[], int nstages)
```

([1], str.4-79).

Funkcia dosahuje rýchlosť $\sim 2$ cykly/vzorku. Aj keď uvedená hodnota naznačuje veľkú efektivitu, je všeobecne známe, že praktické realizácie IIR filtrov pomocou priamej formy bez rozkladu **sú v aritmetike s konečnou presnosťou náchylné k nestabilite**. V praktických aplikáciách je tak ich využitie značne problematické[4].

## *7.3 OPTIMALIZOVANÁ KNIŽNIČNÁ FUNKCIA PRE IIR FILTRÁCIU*

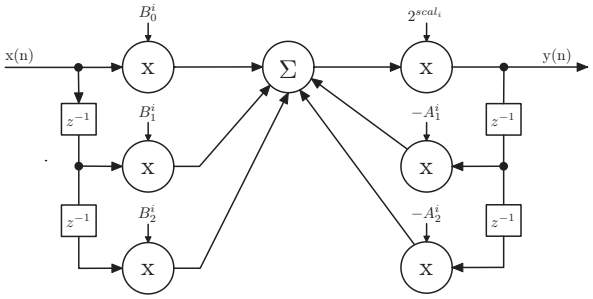
Uvedené nedostatky odstraňuje knižničná funkcia pre IIR filtráciu pomocou procesorov Blackfin, ktorá bola implementovaná na KEMT FEI TU v Košiciach

[3] Zdrojové kódy všetkých knižničných funkcií sú v **...\Blackfin\lib\src\libdsp**

[4] Simulácia **stabdemo.m**, ktorá je súčasťou podkladov k cvičeniu obsahuje demonštráciu nestabilitu priamej realizácie IIR filtra 20-teho rádu aj napriek 64-bitovej presnosti prostredia Matlab. Zároveň ukazuje, že realizácia uvedeného filtra pomocou 10-tich sekcií bikvadov realizuje filtráciu bez problémov.

```
void iir2_fr16( const fract16 x[], fract16 y[], int n, iir_state_fr16 *s)
```

([2]), ktorá realizuje implementáciu pomocou bikvadov (0.2). Táto realizácia využíva nekanonickú formu[5] realizácie bikvadu, ktorá je znázornená na Obr.3.



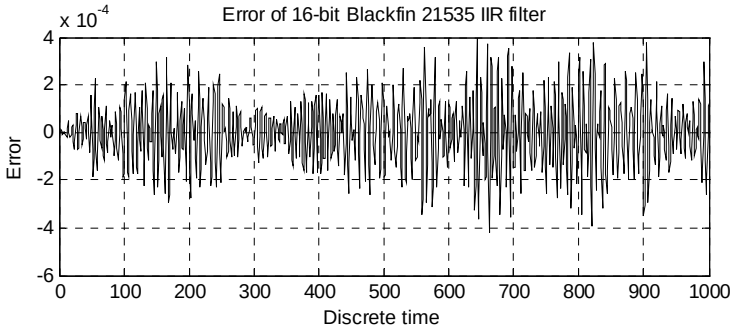

*Obr.3 Nekanonická realizácia bikvadu využívaná vo funkcii iir2_fr16()*

Funkcia využíva koeficienty so zmenenou mierkou v tvare $A_1 = a_1 / S$ , $\ldots B_N = b_N / S$ , pričom pevne zvolený škálovací faktor $S = 2$ zabezpečuje, že žiadny koeficient nemá hodnotu mimo zlomkového intervalu $\langle -1,1)$ .

Funkcia je optimalizovaná z pohľadu rýchlosti tak, že spracováva paralelne pomocou duálnej MAC architektúry 2 vstupné vzorky a dosahuje rýchlosť[6] $\sim 2,5$ cyklu/vzorku, čo demonštruje mimoriadnu efektivitu zdrojového kódu. Knižničná funkcia bola testovaná na vývojových doskách procesorov Blackfin BF535 ako aj BF533 (ktoré využívajú odlišné zreťazenie v riadiacej a dátových jednotkách) a v oboch prípadoch boli dosiahnuté identické výsledky. Na Obr.4 sú znázornené výsledky testovania presnosti eliptického IIR filtra 8-rádu pri spracovaní 16-bitových vstupných údajov pomocou procesora ADSP Blackfin BF535 [2]. Ako referenčná implementácia bola použitá implementácia kaskády bikvadov pomocou Matlabu.

[5] Z pohľadu reálnych implementácií väčší počet oneskorovacích liniek oproti kanonickým realizáciám nie je praktickým obmedzením. Oneskorovacie linky na výstupe bikvadu môžu byť zdieľané ako vstupné oneskorovacie linky nasledujúceho bikvadu, čím sa rozdiel v počte stavebných prvkov pri komplikovanejších IIR filtroch výrazne zmenšuje.

[6] Funkcia dosahuje tieto hodnoty asymptoticky, t.j. len pre IIR filtre veľkého rádu a veľké bloky spracovávaných vzoriek. Je to priamy dôsledok blokového charakteru všetkých analyzovaných knižničných funkcií pre IIR filtráciu. Keďže uvedená IIR funkcia využíva extrémne optimalizované jadro, ktoré v jednom prechode spracováva 2 vstupné vzorky vo 2 za sebou nasledujúcich bikvadoch [2], musí byť rád IIR filtra párny. Pri zložitejších IIR filtroch nie je táto podmienka žiadnym obmedzením, pretože nepárny rád môže byť vždy doplnený posledným bikvadom s jednotkovým prenosom.




*Obr.4 Chyba výpočtu funkcie iir2_fr16() v porovnaní s referenčným výpočtom v Matlabe)*

## *7.4 BLOKOVÉ SPRACOVANIE VZORIEK Z AD A DA PREVODNÍKOV*

V predchádzajúcom cvičení bol analyzovaný projekt spracovania vzoriek z AD a DA prevodníkov vývojovej dosky ADSP BF533 EZ-KIT Lite, ktoré bolo realizované tzv. **rekurzívnym spôsobom** (t.j. spracovanie vstupnej vzorky muselo byť ukončené do príchodu nasledujúcej vzorky toho istého kanálu). Spracovanie bolo realizované v prerušení pomocou funkcie Proces_Data():

```
#include "Talkthrough.h"

//-------------------------------------------------------------------------------------------------------------------------------//
// Function:      Process_Data()                                                                                                 //
//                                                                                                                               //
// Description: This function is called from inside the SPORT0 ISR every                                                        //
//                                    time a complete audio frame has been received. The new                                     //
//                                    input samples can be found in the variables iChannel0LeftIn,                               //
//                                    iChannel0RightIn, iChannel1LeftIn and iChannel1RightIn                                     //
//                                    respectively. The processed          data should be stored in                              //
//                                    iChannel0LeftOut, iChannel0RightOut, iChannel1LeftOut,                                     //
//                                    iChannel1RightOut, iChannel2LeftOut and      iChannel2RightOut                             //
//                                    respectively.                                                                              //
//-------------------------------------------------------------------------------------------------------------------------------//
void Process_Data(void)
{
        iChannel0LeftOut = iChannel0LeftIn;
        iChannel0RightOut = iChannel0RightIn;
        iChannel1LeftOut = iChannel1LeftIn;
        iChannel1RightOut = iChannel1RightIn;
}
```

Nevýhoda rekurzívneho spracovania sa plne prejaví pri pokuse využiť dostupné knižničné funkcie pre IIR filtráciu, ktoré sú optimalizované pre blokové spracovanie (t.j. dĺžka bloku by mala byť značne väčšia ako 1). Naviac pre dosiahnutie efektívnosti je výhodné, aby blok spracovávaných vzoriek obsahoval párny počet vzoriek.

Jedným zo spôsobov, ako je možné uvedený problém pomerne jednoducho vyriešiť[7] je využitie tzv. **„ping-pong"** metódy, ktorá pre každý kanál[8] využíva 2 bufre. Pre smer AD -> bufre sú to nasledujúce polia short (int16) čísel:

[7] Pri uvedenej metóde je možné využiť HW konfiguráciu kodeku AD1836 a procesora z rekurzívneho spracovania AD a DA vzoriek, ktorý bol analyzovaný v predchádzajúcom cvičení. Blokové spracovanie je dosiahnuté len pridaním vhodných bufrov, a ich „softvérovo" riadeným

```
#pragma align 4
short Left_In0_A[BLOCK_SIZE];
#pragma align 4
short Left_In1_A[BLOCK_SIZE];
#pragma align 4
short Left_In0_B[BLOCK_SIZE];
#pragma align 4
short Left_In1_B[BLOCK_SIZE];
#pragma align 4
short Right_In0_A[BLOCK_SIZE];
#pragma align 4
short Right_In1_A[BLOCK_SIZE];
#pragma align 4
short Right_In0_B[BLOCK_SIZE];
#pragma align 4
short Right_In1_B[BLOCK_SIZE];
```

Pre smer bufre->DA to sú polia[9]:

```
#pragma align 4
short Left_Out0_A[BLOCK_SIZE];
#pragma align 4
short Left_Out1_A[BLOCK_SIZE];
#pragma align 4
short Left_Out0_B[BLOCK_SIZE];
#pragma align 4
short Left_Out1_B[BLOCK_SIZE];
#pragma align 4
short Right_Out0_A[BLOCK_SIZE];
#pragma align 4
short Right_Out1_A[BLOCK_SIZE];
#pragma align 4
short Right_Out0_B[BLOCK_SIZE];
#pragma align 4
short Right_Out1_B[BLOCK_SIZE];
```

Interné riadenie počítadla pre zápis resp vyčítavanie bufrov v prerušení je odvodené od premenných (kedže sú typu static, nie sú viditeľné mimo zdrojového kódu codec_buffers.c):

```
static int cnt=0;                   // pocitadlo prijatych vzoriek
static int Ping_Pong = 0;           // 0  => from AD to A, from B to DA
                                    // !0 => from AD to B, from A to DA
```

Signalizácia naplnenia AD bufrov resp. vyprázdnenie DA bufrov je signalizovaná hlavnému programu pomocou premennej (jedinej premennej, ktorá musí byť typu volatile):

```
volatile int New_Blocks_Received;   // 1 => AD to A_Blocks,
                                    // 2 => AD to B_Blocks
```

Spracovanie AD vzoriek z ľavého a pravého kanálu vstupu 0 je realizované v nekonečnej slučke hlavného programu v závislosti na tom, ktorý vstupný bufer je signalizovaný ako plný:

---

ovládaním. Efektívnejším, avšak podstatne komplikovanejším riešením, by bolo priame preprogramovanie AD1836, prerušovacieho systému BF533 a DMA kanálov.

[8] Dvojica bufrov je použitá pre každý AD kanál, ako aj pre každý DA kanál. Vhodnejším kódovaním by bolo možné vysielacie a prijímacie bufre zlúčiť, z dôvodu zachovania väčšej prehľadnosti kódu zlúčenie nebolo využité.

[9] Vzorky z AD prichádzajú ako 24-bitové a ako 24-bitové sú vysielané do DA prevodníkov. Keďže použitá IIR funkcia je optimalizovaná pre spracovanie 16-bitových vzoriek, sú vzorky v bufroch uložené ako 16-bitové (short integer).

```
while(1) {
    if(New_Blocks_Received==1) {    // FINISHED ->from AD to A, from A to DA
        iir2_fr16(Left_In0_A, Left_Out0_A, BLOCK_SIZE, &state_left);
        iir2_fr16(Right_In0_A, Right_Out0_A, BLOCK_SIZE, &state_right);
        memcpy(Left_Out1_A, Left_In1_A, 2*BLOCK_SIZE);
        memcpy(Right_Out1_A, Right_In1_A, 2*BLOCK_SIZE);
        New_Blocks_Received = 0;    // signalizacia spracovania bloku
    }
    if(New_Blocks_Received==2) {    // FINISHED ->from AD to B, from A to DA
        iir2_fr16(Left_In0_B, Left_Out0_B,BLOCK_SIZE, &state_left);
        iir2_fr16(Right_In0_B,Right_Out0_B, BLOCK_SIZE, &state_right);
        memcpy(Left_Out1_B,Left_In1_B,2*BLOCK_SIZE);
        memcpy(Right_Out1_B,Right_In1_B,2*BLOCK_SIZE);
        New_Blocks_Received = 0;    // signalizacia spracovania bloku
    }
}
```

**Príklad**
*Analyzujte kompletné zdrojové kódy priloženého projektu blokovej IIR filtrácie a identifikujte*

- *umiestnenie začiatku bufrov na adresy s násobkom 4,*
- *kompresiu 24-bitových vzoriek na 16-bitové pri čítaní dát z AD prevodníkov*
- *dekompresiu 16-bitových vzoriek na 24-bitové pri zápise dát do DA prevodníkov,*
- *umiestnenie (+ inicializáciu) koeficientov a stavových premenných do dátových pamätí L1 z dôvodu dosiahnutia maximálnej rýchlosti,*
- *filtráciu v main.c,*
- *ďalšie zmeny oproti projektu z predchádzajúceho cvičenia úpravou ktorého vznikol projekt blokového spracovania,*
- *spôsob využitia[10] externej knižnice* ***iir2lib.dlb****.*

**Príklad**
*Pomocou osciloskopu a preladiteľného generátora odmerajte pásmo priepustnosti IIR filtra v priloženom projekte.*

**Príklad**
*Navrhnite vlastný IIR filter a upravte zdrojový kód priloženého projektu tak, aby realizoval navrhnutý filter. Overte správnosť pomocou osciloskopu a preladiteľného generátora.*

## *7.5 ZÁVER*

Dostupné knižničné funkcie pre IIR filtráciu pre procesor Blackfin, ktoré sú súčasťou prostredia VisualDSP++ sú jasným príkladom, že tvorbu knižničných funkcií je potrebné realizovať aj so zohľadnením teórie ČSS a čistá „programátorska" implementácia môže mať vážne praktické nedostatky.

[10] Názov funkcie v externej knižnici **bol zámerne zmenený na iir2_fr16()**, ktorý je odlišný od názvu originálnej funkcie iir_fr16(). Názov by však mohol byť aj zhodný s názvom pôvodnej funkcie iir_fr16() v štandardnej DSP knižnici VisualDSP++. Zahrnutím zdrojového kódu (resp. knižnice) s rovnakým menom ako má funkcia v štandardnej knižnici priamo do projektu je možné nahradiť štandardné funkcie novými s identickým menom.

# LITERATÚRA

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

[2] Drutarovský, M.: An Implementation of High Performance IIR Filtration on 2-MAC Blackfin DSP Architecture. Proceedings of the 6th International Scientific Conference on Digital Signal Processing and Multimedia Communications - DSP-MCOM 2005, September 13-14, 2005, Kosice, Slovakia, pp.110-113. (http://www.kemt.fei.tuke.sk/personal/drutarovsky/publications/dspmcom2005.pdf).