1 PROCES TVORBY PROGRAMOV PRE SIGNÁLOVÉ PROCESORY

1.1 ÚVOD

Vysoký výpočtový výkon signálových procesorov (DSP) je len jedným z nutných predpokladov pre úspešnú realizáciu systémov ČSS na báze DSP. Na dosiahnutie vysokého výkonu v praktických aplikáciách má podstatný (a často dokonca dominantný) vplyv kvalita **vývojových prostriedkov**. Aj keď súčasné DSP sú z hľadiska typov vývojových prostriedkov porovnateľné s univerzálnymi mikroprocesormi[1], vzhľadom na požiadavku práce v reálnom čase je **nízko-úrovňové programovanie** v asembleri často jedinou reálnou alternatívou. V rámci tohto cvičenia opíšeme **základné prostriedky** na tvorbu programov, ktoré sa v súčasnosti typicky využívajú pri vývoji aplikácií na báze DSP, pričom príklady jednotlivých vývojových prostriedkov budeme demonštrovať na vývojových prostriedkoch pre signálový procesor ADSP218x od firmy Analog Devices. Tieto prostriedky budeme využívať aj v ďalších cvičeniach pri vysvetľovaní základných vlastností ADSP218x ako aj pri práci s reálnymi technickými prostriedkami (vývojové dosky EZ-KIT2181 Lite, EZ-KIT2191 Lite a EZ-KIT21535 Lite od firmy Analog Devices).

1.2 ZÁKLADNÉ VÝVOJOVÉ PROSTRIEDKY

Do tejto kategórie patria predovšetkým asemblery, linkovacie programy, knižničné funkcie, simulátory, vývojové dosky a emulátory.

1.2.1 ASEMBLERY

Asembler je program, ktorý prekladá špecifické inštrukcie procesora zapísané v zdrojovom ASCII súbore do binárneho **objektového kódu** (object code) pre cieľový DSP. Zvyčajne tento objektový kód (pokiaľ je v tzv. **relatívnom tvare**) vyžaduje dodatočnú transformáciu (**relokáciu** a **linkovanie**), ktorá sa realizuje linkovacím

[1] V oblasti univerzálnych (jednočipových) mikroporocesorov sú v súčasnosti široko využívané vyššie programovacie jazyky (typicky jazyk C, C++,…), operačné systémy a pod. Tieto vývojové prostriedky existujú aj pre DSP, ich využitie v reálnych systémoch na báze DSP je však do určitej miery limitované požiadavkou na prácu v reálnom čase resp. snahou o minimalizáciu veľkosti programových pamätí. V praktických aplikáciách sú preto tieto prostriedky zvyčajne kombinované s kódom v asembleri, ktorý je využitý napr. vo forme optimalizovaných knižníc resp. priamo vložených asemblerovských príkazov. Z hľadiska vývojových prostriedkov je možné tieto prostriedky zaradiť medzi **pokročilé prostriedky**, pričom im bude venovaná časť prednášok.

programom (**linkerom**). Linker generuje vykonateľný binárny kód pre cieľový DSP. Vo fáze linkovania je možné použiť dostupné **knižničné funkcie**.

Väčšina súčasných asemblerov sú tzv. **macro asemblery**, ktoré umožňujú definovať (alebo používať preddefinované) parametrizovateľné bloky kódu (**makrá**), ktoré sú do zdrojového kódu vkladané počas prekladu. Makrá umožňujú programátorovi zmenšiť množstvo zdrojového kódu, ktorý je treba udržiavať (čím je možné zvýšiť spoľahlivosť programov) ako aj eliminovať nadbytočné inštrukcie, ktoré je potrebné použiť pri volaní podprogramov a sú v programoch pre DSP veľmi často využívané. Nasledujúci kód dokumentuje použitie asemblerovského makra pri realizácii FIR filtra pomocou procesora Motorola DSP5600x.

```
; definícia makra „FIR", ktoré implementuje FIR filter s N koeficientmi
FIR     macro  N
        clr    a
        rep    #N-1
        mac    x0,y0,a x:(r0)+,x0      y:(r4)+,y0
        macr   x0,y0,a (r0)-
        endm

; inicializácia smerníkov na koeficienty a dáta
        move   #data,r0
        move   #koeficienty,r4

; volanie makra pre FIR filter s 512 koeficientmi
        FIR    512
```

Zaujímavou alternatívou ku klasickým asemblerom sú tzv. *algebraické asemblery*, u ktorých je snaha používať štýl písania asemblerovských programov, ktoré pripomínajú algebraický zápis algoritmu. Tento prístup využíva napr. firma Analog Devices, čo dokumentuje nasledujúci kód[2] FIR filtra pre procesor ADSP21xx:

```
.ENTRY          fir;
fir:    MR = 0, MX0 =  DM( I0, M1),    MY0 = PM(I4, M5);
        DO sop UNTIL CE;
sop:            MR = MR + MX0*MY0( SS ), MX0=DM(I0, M1), MY0=PM(I4, M5);
        MR = MR+MX0*MY0(RND);
        IF  MV  SAT  MR;
        RTS
```

1.2.2 KNIŽNIČNÉ FUNKCIE

Pre aplikácie na báze DSP majú optimalizované knižnice kľúčový význam. Aj keď typické jadrá algoritmov ČSS implementované pomocou DSP sú relatívne krátke, ich optimalizácia je z pohľadu programátora značne náročná úloha, ktorá vyžaduje dobrú znalosť architektúry cieľového DSP. Naviac je často potrebné zvážiť možnosti modifikovať samotný algoritmus ČSS do tvaru, ktorý je pre danú architektúru vhodnejší, čo zvyčajne vyžaduje špecifické znalosti z oblasti teórie ČSS. Je logické, že výrobcovia DSP poskytujú pre svoje procesory **optimalizované kódy**, ktoré umožňujú relatívne efektívnu implementáciu základných algoritmov ČSS (FIR, IIR, FFT, DCT…). Tieto kódy sú dostupné v rámci WWW stránok jednotlivých výrobcov (napr. [1], [2], [3], [4]), aplikačných príručiek a špecializovaných kníh (napr. [5], [6], [7]).

Aj keď je často možné tieto kódy ďalej vylepšiť, sú uvedené kódy dobrým štartovacím bodom pre vývoj aplikačných programov. Je dokonca výhodné pri

[2] Tento kód využíva staršiu formu zápisu asemblerovských príkazov vo forme napr. DM(I0, M1), asembler použitý v integrovanom vývojovom prostredí VisualDSP++ používa novší zápis tvare DM(I0+=M1).

zoznamovaní sa s architektúrou DSP využívať hotové programy resp. funkcie. Klasický postup, t.j. zvládnutie celej inštrukčnej sady a snaha o písanie vlastných programov je málo efektívnym prístupom k zvládnutiu DSP. V ďalších cvičeniach budeme preto analyzovať takéto (relatívne jednoduché) kódy pre FIR filter, IIR filter a FFT a na týchto príkladoch budeme vysvetľovať podstatné vlastnosti ADSP218x/9x.

Knižničné funkcie je možné pomocou asemblera preložiť do relatívneho objektového tvaru a spojiť pomocou programu – **knihovníka** do jedného súboru – **knižnice**. Knižnice je potom možné využívať napr. ako aplikačné balíky (napr. knižnica pre číslicovú filtráciu, FFT a pod).

Nasledujúci kód dokumentuje použitie asemblera v knižničnej funkcii[3] pre FIR filter v jayzku C s prototypom

```
int fir(int sample, int pm* coeffs, int* delay, int taps);
```

a telom funkcie optimalizovanom v asembleri

```
/************************************************************************
 *
 * fir.asm
 *
 * (c) Copyright 2001 Analog Devices, Inc.  All rights reserved.
 *
 ************************************************************************/

/*
  Finite Impulse Response filter (FIR)

  #include <filters.h>
  int fir(int sample, int __pm coeffs[], int __dm delay[], int taps);

  This function is an Analog Devices extension to the ANSI standard.

  The fir() function implements a finite impluse response (FIR)
  filter defined by the coefficients and delay line that are supplied
  in the call of fir. The function produces the filtered response of
  its input data. This FIR is structured as a sum of products. The
  characteristics of the filter (passband, stop band, etc) are
  dependent on the coefficient values and the number os taps
  supplied by the calling program.

  The integer input to the filter is sample. The integer taps indicates
  the length of the filter, which is also the length of the array
  coeffs. The coeffs array holds one FIR coefficient per element. The
  coefficients are stored in reverse order; for example, coeffs[0]
  holds the (taps-1) coefficient. The coeffs array is located in
  program memory data space to allow the single-cycle dual-memory
  fetch of the processor to be used.

  The state array contains a pointer to the delay line as its last
  element, preceded by the delay line values. The length of the state
  array is therefore one (1) greater than the number of taps. Each
  filter has its own state array, which should not be modified by the
  calling program, only by the fir function. The state array should be
  initialized to zeros (0) before the fir function is called for the
  first time. The state array must be on a circular boundary.


  The parameters sample, coeffs[], and state[], are all considered
  to be fractional numbers. The fir() function executes fractional
  multiplications that preserve the format of the fractional input.
  If your applications requires a true integer fir(), you should
  divide the output of the filter by two (2).
*/
```

[3] Táto knižničná funkcia je súčasťou integrovaného vývojového prostredia Visual DSP pre signálové procesory Analog Devices ADSP21xx (www.analog.com/dsp/tools), s ktorým budeme pracovať v rámci cvičení.

```
.section/code program;

.global _fir;

_fir:
   DIS M_MODE;                    // ###### Fractional mode #######
   M5=1;
   I6=I4;
   MODIFY(I6+=M5);
   MX0=DM(I6+=M5);                // sample
   MY0=DM(I6+=M5);                // pointer to coeffs
   MR1=DM(I6+=M5);                // pointer to delay
   AX1=DM(I6+=M6);                // taps

   M5=AX1;
   I6=MR1;                        // Set I6 to addr of delay_line_ptr
   modify(I6+=M5);                // Move to pointer to delay lines

   MR0 = DM(I6+=M6);              // Check if initialized
   NONE = PASS MR0;
   IF NE JUMP initialized;        // Yes, skip
   DM(I6+=M6) = MR1;              // No, Save ptr of delay line

initialized:
   M5=1; M3=-1;
   MR1=DM(I6+=M6);                // Fetch pointer to delay line
   L1=AX1;
   I1=MR1;
   DM(I1+=M1)=MX0;                // put sample in delay line

   MR0=I1;
   DM(I6+=M6)=MR0;                // save new delay_line_ptr

   I6=MY0;                        // The ptr to the coeff array
   I1=MR1;                        // The ptr to the delay line
   MR=0, MY0=PM(I6+=M5), MX0=DM(I1+=M3);
   CNTR=AX1;
   DO convolution UNTIL CE;
convolution: MR=MR+MX0*MY0 (SS), MY0=PM(I6+=M5), MX0=DM(I1+=M3);

   MR=MR (RND);
   IF MV SAT MR;
   AX1=MR1;
   L1=0;

   ENA M_MODE;                    // ###### End Fractional mode #######
   RTS;

  ._fir.end:
  .size (_fir,._fir.end-_fir);
```

1.2.3 SIMULÁTORY

Simulátory sú programy, ktoré simulujú prácu procesora na úrovni jednotlivých inštrukcií. Z pohľadu optimalizácie kódu pre DSP majú simulátory kľúčovú úlohu a umožňujú odhaliť problémy vznikajúce napr. vplyvom **zreťazenia**[4], prípadne zvýšiť efektivitu kódu identifikáciou inštrukcií, medzi ktorými sa vytvárajú tzv. „pipeline bubbles", t.j. okamihy, kedy zreťazené technické prostriedky nie sú plne využité. Vhodným preusporiadaním inštrukcií je často možné zvýšiť využitie prostriedkov procesora.

[4] Zreťazenie je jednou zo všeobecných metód zvýšenia výkonnosti mikroprocesorov a je v oblasti DSP široko využívaná. Metóda bude opísaná na prednáškach.

1.2.4 VÝVOJOVÉ DOSKY, EMULÁTORY

Pri vývoji zariadení na báze DSP je cieľom realizovať systém ČSS, ktorý pracuje s reálnymi vstupnými resp. výstupnými dátami. V určitej etape vývoja je dôležité mať k dispozícii reálne technické zariadenie (často sa technické a programové prostriedky vyvíjajú paralelne a cieľové zariadenie nie je v úvodnej fáze vývoja k dispozícii). Na overenie činnosti algoritmov ČSS v reálnom čase sú vhodné **vývojové dosky**, ktoré typicky obsahujú všetky základné bloky číslicového signálového procesora (obmedzovacie a rekonštrukčné filtre, AD a DA prevodníky, DSP, pamäte) . Všetci hlavní výrobcovia DSP poskytujú lacné vývojové moduly (v cenách 70-250 $), ktoré je možné dokonca v prípade menej náročných zariadení využiť aj ako konečné technické riešenie systému ČSS. Výhodné je, pokiaľ cieľový DSP podporuje **emuláciu na čipe**[5], čo zlepšuje možnosti ladenia priamo v reálnych podmienkach pri minimálnych nárokoch na dodatočné technické vybavenie.

Využitie klasických **emulátorov** je v praxi veľmi problematické predovšetkým vzhľadom na stále sa zvyšujúcu taktovaciu frekvenciu DSP a používanie prevažne SMD technológie, čo vylučuje nasadenie klasickej **emulačnej hlavice**.

1.3 VÝVOJOVÉ PROSTRIEDKY VYUŽÍVANÉ NA CVIČENIACH

V rámci cvičení budú využívané základné vývojové prostriedky pre signálové procesory Analog Devices ADSP218x dodávané firmou Analog Devices. Tieto programy sú voľne dostupné na WWW stránkach firmy Analog Devices (**www.analog.com/dsp/tools**) vo forme 30-dňovej testovacej verzie, ktorú je možné zdarma aktivovať po získaní aktivačného kódu (položka **VisualDSP++ Test Drive Registration**). Počas cvičení budú využívané predovšetkým tieto programy:

1) asembler **easm218x.exe**
2) C prekladač **cc218x.exe**
3) linker **linker.exe**
4) Simulátor (dostupný z integrovaného prostredia podobne ako aj asembler, prekladač a linker)

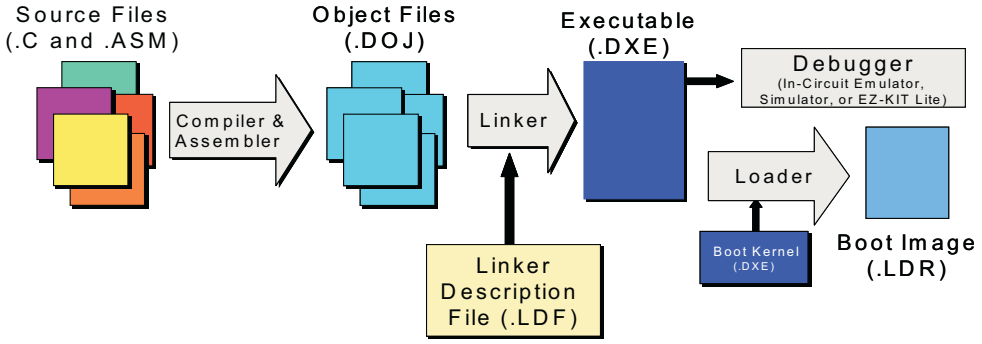
Ďalšie programy ako knihovník, rôzne konverzné a pomocné programy sú tiež dostupné, počas cvičení však nebudú využívané[6]. Počas cvičení budú využívané optimalizované kódy pre ADSP218x, ktoré sú súčasťou prostredia VisualDSP++ resp. sú dostupné na WWW stránke firmy Analog Devices (**http://www.analog.com/techSupport/designTools/codeExamples/index.html**) Tieto kódy patria do kategórie voľne šíriteľných programov (freeware) a reprezentujú kódy pre algoritmy z rôznych oblastí ČSS.

[5] Špeciálna metóda ladenia, ktorá využíva ladiace obvody implementované priamo v čipe procesora.

[6] Cieľom cvičení je vysvetliť základné vlastnosti DSP, čo budeme dokumentovať na relatívne jednoduchých príkladoch pri ktorých vystačíme s asemblerom, prekladačom, linkerom a simulátorom. Programy ako napr. knihovník je výhodné využívať pri väčších projektoch (ako napr. diplomové úlohy), pri ktorých môžu výrazne zrýchliť a sprehľadniť generovanie výsledného kódu.

Podstatná časť informácií je platná aj pre novšiu a výkonnejšiu triedu signálových procesorov Analog Devices ADSP219x[7]. Pre tieto procesory je dokonca k dispozícii prekladač jazyka C++.

Integrované vývojové prostredie VisualDSP od firmy Analog Devices využíva špecifické prípony pre generované výstupné súbory. Informácie o projekte sú uložené v súbore s príponou **.prj**. Zdrojové kódy majú zvyčajne prípony **.asm**, **.h**, **.c**. Kód preložený do relatívneho objektového tvaru má príponu **.doj** a kód preložený do absolútneho objektového tvaru má príponu **.dxe**. Technické prostriedky pre ktoré je vytváraný výsledný program sú opísané v tzv. Linker Description File s príponou **.ldf**. Knihovník vytvára zlúčením viacerých súborov s príponami **.doj** knižnicu s príponou **.dlb**. Proces tvorby programového kódu pre signálové procesory Analog Devices ADSP21xx a použité programové prostriedky sú znázornené na Obr.1.

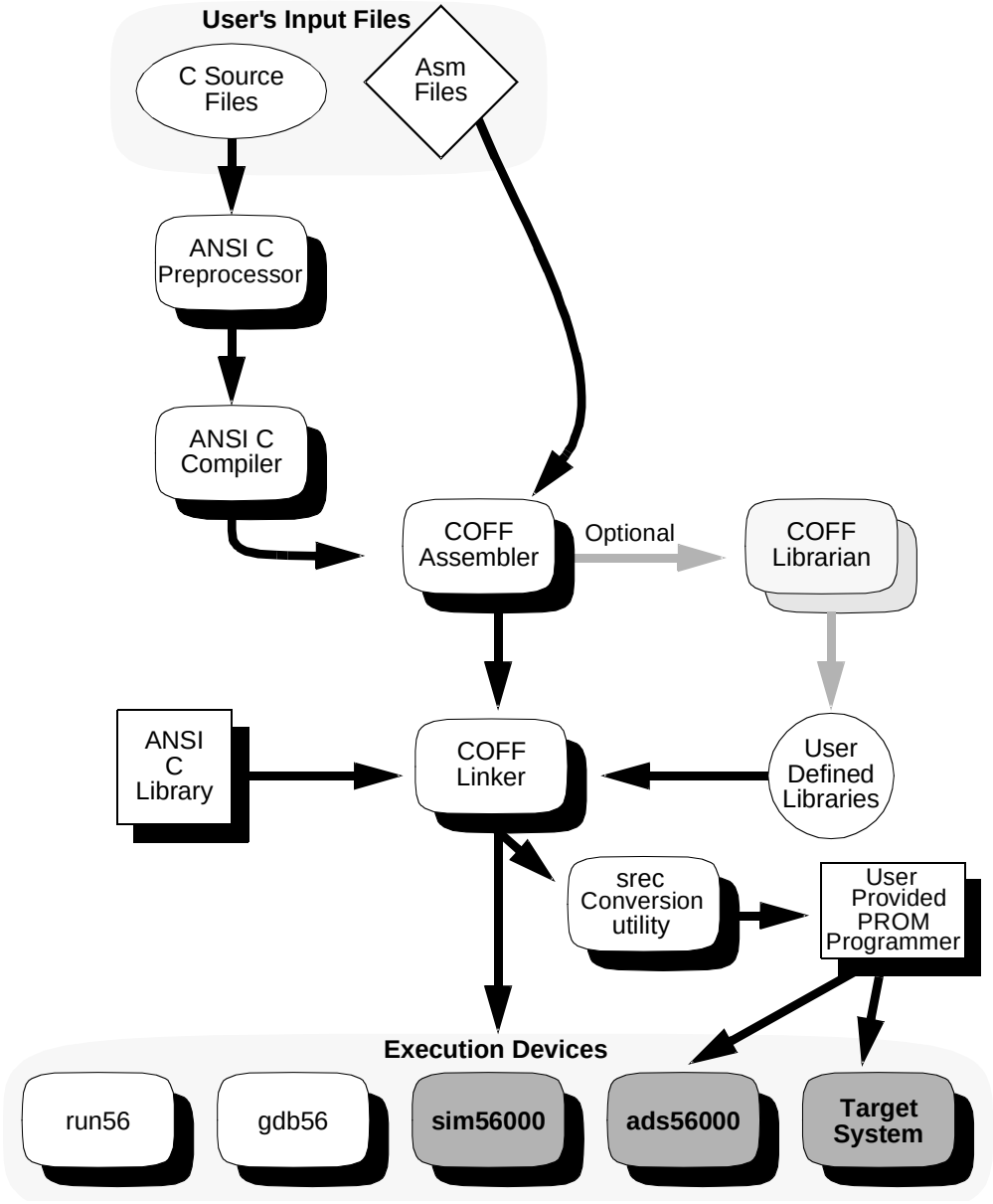


Obr.1 Štruktúra programových prostriedkov pre DSP firmy Analog Devices

Tento obrázok reprezentuje možnosti využitia vývojových prostriedkov. Vykonateľný kód bude v počiatočných cvičeniach simulovaný pomocou simulátora a neskôr vykonávaný v cieľovom systéme – EZ-KIT2181 Lite.

Na Obr.2 je znázornený proces tvorby kódu pre signálové procesory Motorola, pričom obrázok podrobnejšie naznačuje aj možné využitie **prekladača** (kompilátora) z vyššieho programovacieho jazyka C a **knihovníka** (Librarian) pre tvorbu užívateľských knižníc. Samozrejme tieto princípy sú využívané aj v prostredí VisualDSP prípadne aj napr. vo vývojových prostrediach pre jednočípové mikroprocesory.

[7] Inštrukčná sada rodiny ADSP219x tvorí **nadmnožinu** inštrukčnej sady ADSP218x a je typickým príkladom dedičnosti medzi procesormi. Základné informácie o tejto rade procesorov budú preberané v rámci prednášky.



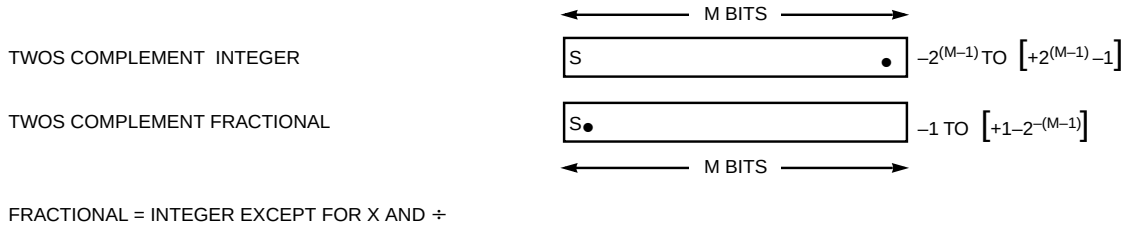
Obr.2 Štruktúra programových prostriedkov pre DSP firmy Motorola

2 ZLOMKOVÝ FORMÁT ČÍSEL V DSP

Základnou úlohou (pre ktorú sú DSP optimalizované) je matematické spracovanie číslicových údajov. Medzi typické algoritmy pre ktoré sú komerčne dostupné DSP optimalizované patria číslicová filtrácia a algoritmus FFT. Základným dátovým typom DSP **s pevnou rádovou čiarkou** (medzi ktoré patria aj procesory Motorola DSP5600x [8], DSP563xx a Analog Devices ADSP21xx [9]) patrí **zlomkový formát** (fractional format, firma Analog Devices používa pre ich 16-bitové DSP tiež označenie fractional representation 1.15), pomocou ktorého sú čísla v intervale $\langle -1,1)$ reprezentované v tvare M bitov (**dĺžka slova**) $b_m \in \{0,1\}$, $m = 0,1,2,\ldots,M-1$, ktoré vyjadrujú číslo v tvare

$$x_{pevna} = (-1)b_0 + \sum_{m=1}^{M-1} b_m 2^{-m} \tag{1.1}$$

ktorý sa od klasického celočíselného formátu (ktorý reprezentuje záporné čísla pomocou dvojkového doplnku) odlišuje polohou desatinnej čiarky, čo je dokumentované na nasledujúcom obrázku.



Obr.3 Porovnanie celočíselnej a zlomkovej reprezentácie

Dĺžka slova v komerčne využívaných DSP s pevnou rádovou čiarkou je 16 a 24 bitov[8]. Základným dôvodom, prečo je využívaná zlomková reprezentácia je jej tesnejšia väzba s formátom **pohyblivej rádovej čiarky**, ktorý sa bežne využíva pri vedecko-technických výpočtoch (medzi ktoré patria aj algoritmy ČSS). Napríklad všetky bežne dostupné programy pre návrh číslicových filtrov poskytujú koeficienty v pohyblivej rádovej čiarke, čo zvyčajne možné pretransformovať do zlomkového formátu jednoduchou **zmenou mierky** (t.j. predelením hodnôt mierkovou konštantou).

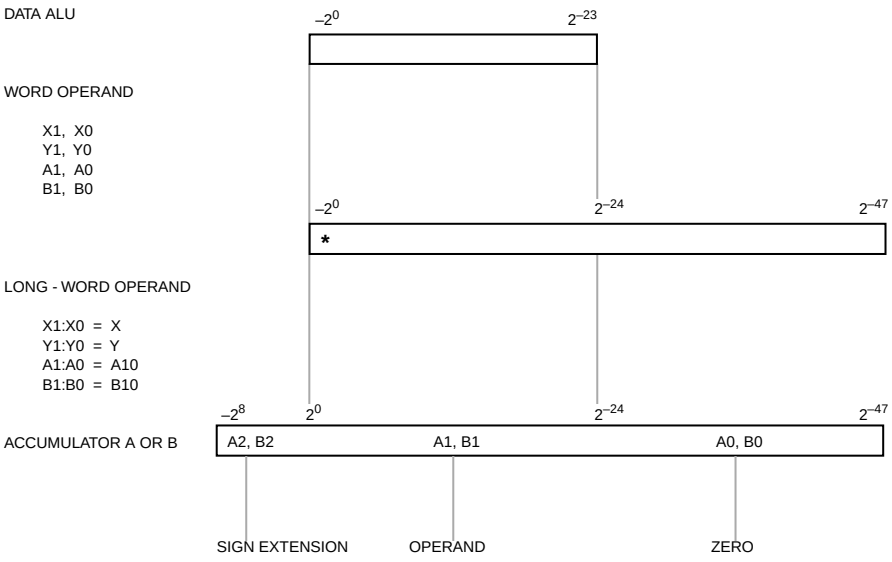
[8] 16-bitové DSP sa typicky využívajú v telekomunikačnej technike a 24-bitové DSP sú dominantné predovšetkým pre audio aplikácie. Procesory Motorola DSP5600x a DSP563xx sú 24-bitové DSP. Všetky DSP firmy Analog Devices, ktoré využívajú aritmetiku s pevnou rádovou čiarkou sú 16-bitové.

Ďalšou výhodnou vlastnosťou je že ak $x, y \in \langle -1,1)$ potom platí

$$z = x * y \in \langle -1,1) \tag{1.2}$$

a teda pri operácií násobenia nedochádza k pretečeniu výsledkov mimo interval $\langle -1,1)$. K pretečeniu mimo zlomkový interval $\langle -1,1)$ dochádza len[9] pri operácií sčítania resp. odčítania.

V rámci štruktúry DSP (presnejšie **dátových registrov**) obyčajne existuje niekoľko typov dátových registrov s rôznou presnosťou. V signálových procesoroch DSP5600x napr. existujú 24-bitové registre (X0,X1,Y0,Y1) a 56-bitové registre (**akumulátory**) A, B, ktorých štruktúra a váhy jednotlivých bitov sú znázornené na nasledujúcom obrázku.



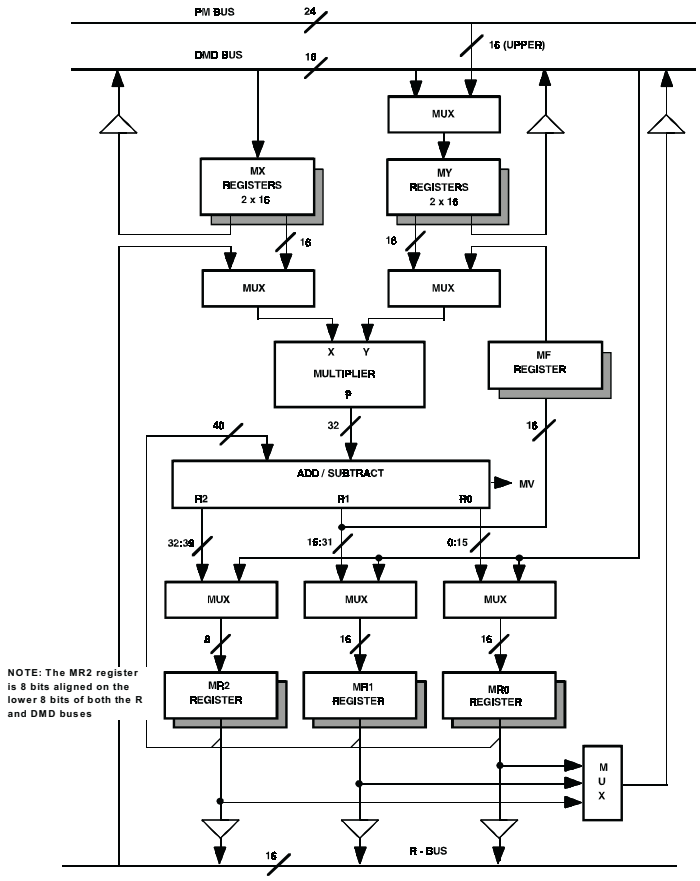
Obr.4 Štruktúra registrov v procesoroch Motorola DSP5600x

Akumulátory A=(A2:A1:A0) a B=(B2:B1:B0) sa skladajú z 24-bitových subregistrov A0, A1, B0, B1 a 8-bitových registrov A2, B2. Akumulátory A a B sa využívajú predovšetkým pomocou inštrukcie MAC, ktorá realizuje výpočet

$$Akum = Akum \pm \mathrm{Re}\, g\_X * \mathrm{Re}\, g\_Y \tag{1.3}$$

Štruktúra registrov v procesoroch ADSP21xx je podstatne bohatšia, čo je dané predovšetkým využitím 3 výkonných jednotiek – **ALU**, **MAC** a **posúvača** (barrel shifter). Vzhľadom na to, že procesory ADSP21xx využívajú menší dynamický rozsah (16 resp. 40 bitov oproti 24 a 56 bitom u DSP5600x), obsahujú procesory ADSP podstatne lepšiu podporu pre aritmetiku v blokovej pohyblivej rádovej čiarke a v aritmetike s dvojnásobnou presnosťou. Na Obr.5 je znázornená štruktúra MAC jednotky v ADSP 218x. Architektúra ADSP bude podrobnejšie vysvetlená v prednáškach.

[9] Operácia delenia je z pohľadu DSP špeciálna operácia, ktorá je podstatne pomalšia ako operácie +,-,*. DSP typicky poskytujú špeciálne inštrukcie, pomocou ktorých je možné realizovať delenie iteračným spôsobom, t.j. na dosiahnutie výsledku delenia je potrebných niekoľko inštrukcií. V procesoroch ADSP21xx je napr. možné využiť inštrukcie DIVS (divide sign) a DIVQ (divide quotient).



Obr.5 Štruktúra MAC jednotky v procesoroch ADSP 218x

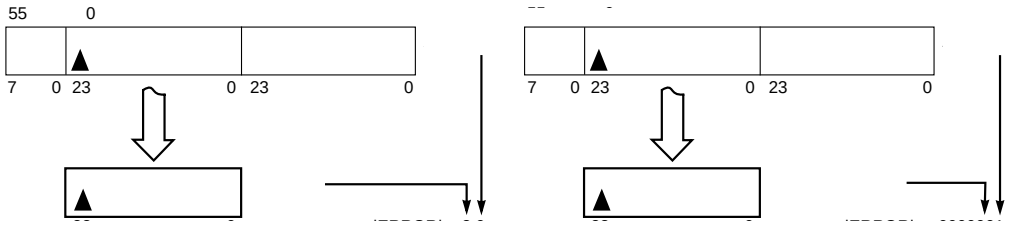
Podobne ako v procesoroch Motorola DSP5600x majú aj ADSP21xx v jednotke MAC akumulátor MR=(MR2:MR1:MR), ktorý sa skladá zo 16-bitových subregistrov MR0, MR1a 8-bitového registra MR2. Akumulátor MR sa využívajú predovšetkým pomocou inštrukcie, ktorá realizuje výpočet

$$MR = MR + X * Y \tag{1.4}$$

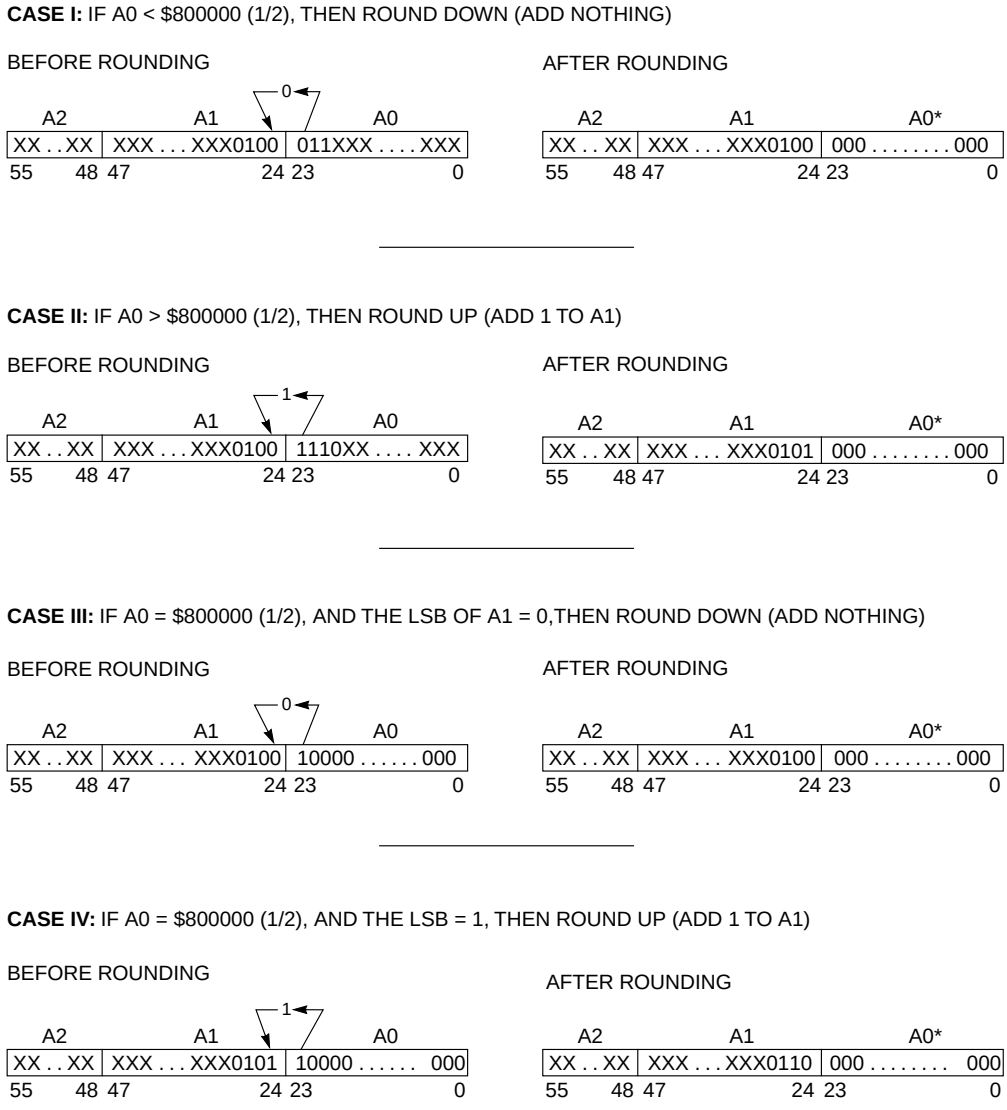
pričom X (register MX0 alebo MX1) a Y (register MY0 alebo MY1) sú vstupné 16-bitové registre MAC jednotky. Je teda možné konštatovať, že štruktúra registrov v ALU procesorov Motorola a MAC jednotke procesorov ADSP je prakticky totožná a odlišuje sa len dĺžkou resp. počtom registrov.

S aritmetikou v procesoroch DSP5600x (ADSP21xx) sú spojené dva pojmy – **saturačná aritmetika (saturačný mód v ALU a MAC)** a **konvergentné zaokrúhľovanie (zaokrúhľovací mód v MAC[10]).** Tieto vlastnosti sú demonštrované na Obr.6 a Obr.7 a podrobne vysvetlené počas cvičenia.

[10] Novšie procesory (napr. ADSP217x, 218x, 21msp58) umožňujú aj využitie klasického zaokrúhľovania (tzv. biased rounding), ktoré sa využíva napr. pri implementácií kompresných rečových kodekov, ktoré musia poskytovať bitovo zhodný výstup s referenčným kodekom, ktorý je zvyčajne implementovaný v jazyku ANSI C.



Obr.6 Princíp saturačnej aritmetiky v procesoroch Motorola (56-bitový akumulátor), zhodný princíp sa uplatňuje aj v prípade ADSP21xx (40-bitový akumulátor)



Obr.7 Princíp konvergentného zaokrúhľovania v procesoroch Motorola (56-bitový akumulátor), zhodný princíp sa uplatňuje aj v prípade ADSP21xx (40-bitový akumulátor)

LITERATÚRA

[1] www.analog.com

[2] www.ti.com

[3] www.mot.com

[4] www.kemt.fei.tuke.sk/adsp

[5] El-Sharkawy, M.: *Real Time Digital Signal Processing Applications With Motorola's DSP56000 Family*. Prentice Hall, Englewood Cliffs, New Jersey, 1990

[6] Mar, E. (editor): Digital Signal Processing Applications using the ADSP-2100 Family – Volume I. Prentice Hall, Englewood Cliffs, 1992 (dostupné aj v elektronickej forme).

[7] Babst, J. (editor): Digital Signal Processing Applications using the ADSP-2100 Family – Volume I. Prentice Hall, Englewood Cliffs, 1995 (dostupné aj v elektronickej forme).

[8] DSP56000 Digital Signal Processor Family Manual. DSP56KFAMUM/AD, Motorola, Inc., 1992

[9] ADSP-2100 Family User's Manual, Analog Devices, Inc., 1995