3 TVORBA A LADENIE PROGRAMOV V PROSTREDÍ VISUALDSP++

Ciele cvičenia:
vytvorenie kompletného projektu v prostredí VisualDSP++
využitie DSP projektov so zmiešanými C a ASM súbormi
linker a základné možnosti LDF súboru
ladiace možnosti v prostredí VisualDSP++
využitie knižničných funkcií

3.1 ÚVOD

Prostredie VisualDSP++ poskytuje bohaté možnosti na tvorbu a ladenie kompletných projektov pre DSP firmy Analog Devices. Prostredie je identické pre rodiny procesorov Blackfin, SHARC a TigerSHARC[1] a v súčasnosti (rok 2008) je aktuálna verzia VisualDSP++ v.5.0 a Update 4 zo septembra 2008. Pre procesory ADSP21xx je k dispozícii verzia ViusalDSP++ v.3.5 a Update z októbra 2006. Na stránkach firmy Analog Devices [1] je možné stiahnuť[2] inštalačné balíky najnovších verzií a Service Pack balíkov. Tieto verzie **sú identické** s verziami balíkov VisualDSP++, ktoré sú dostupné komerčne. Po inštalácii je pre správnu funkčnosť prostredia potrebné na stránkach firmy Analog Devices [3] získať **bezplatné** testovacie licencie, ktoré umožnia využívať **plne funkčné verzie** prostredia VisualDSP++ po dobu 90 dní.
Prostredie VisualDSP++ pre procesory Balckfin bude využívané vo všetkých nasledujúcich cvičeniach pri vytváraní, ladení a testovaní programov pre DSP. Cieľom cvičení je prezentácia **základných vlastností** prostredia VisualDSP++, ktoré umožnia postupne vytvoriť reálne DSP aplikácie. Súčasťou prostredia je bohatý on-line help a kompletné manuály v PDF formáte [4], ktoré je možné využiť v prípade potreby zvládnutia zložitejších možností, ktoré prostredie VisualDSP++ poskytuje. Základom pre dnešné cvičenie boli vybrané časti z úvodného manuálu k prostrediu VisualDSP++ [5].

3.2 DSP PROJEKT S KOMBINÁCIOU C A ASM SÚBOROV

Kombinácia zdrojových súborov v jazyku C a ASM je typickou technikou, ktorá umožňuje dosiahnuť pomerne vysokú efektivitu vytvoreného programu s relatívne

[1] Procesory SHARC a TigerSHARC sú výkonné 32-bitové procesory využívajúce aritmetiku s pohyblivou rádovou čiarkou.
[2] Z dôvodu ľahšej dostupnosti pre študentov KEMT FEI TU Košice sú aktuálne verzie prostredia VisualDSP++ a Update balíkov pre procesory Blackfin a ADSP21xx dostupné na www stránkach katedry [2]. Testovací licenčný kód je potrebné získať na www stránke Analog Devices [3].

malým úsilím programátora. V nasledujúcej časti bude vytvorený kompletný projekt realizujúci výpočet skalárneho súčinu, ktorý bol realizovaný v predchádzajúcom cvičení pomocou C jazyka. Telo funkcie na výpočet skalárneho súčinu bude realizované optimalizovanou funkciou v ASM. Projekt tak umožní porovnanie rýchlosti obidvoch techník. Kompletný projekt v prostredí ViusalDSP++ bude vytvorený len zo zdrojových C a ASM súborov.

3.2.1 VYTVORENIE PROJEKTU A ZAČLENENIE ZDROJOVÝCH SÚBOROV

Súčasťou projektu sú dva zdrojové súbory (dotprodasm.zip):

dotprod_main.c
dotprod_func.asm

ktorých funkčnosť je identická s projektom dotprod.zip, ktorý bol precvičovaný na minulom cvičení. Na rozdiel od projektu dotprod.zip je však výpočet realizovaný pomocou funkcie

```
int a_dot_c_asm( int *a, int *c );
```

analyzovanou na minulom cvičení. Hlavný program je modifikovaný nasledujúcim spôsobom

```
extern int a_dot_c_asm( int *a, int *c );


void main()
{
      int i;
      int result = 0;

      result = a_dot_c_asm( a, c );

      printf( "Dot product = %d\n", result );
}
```

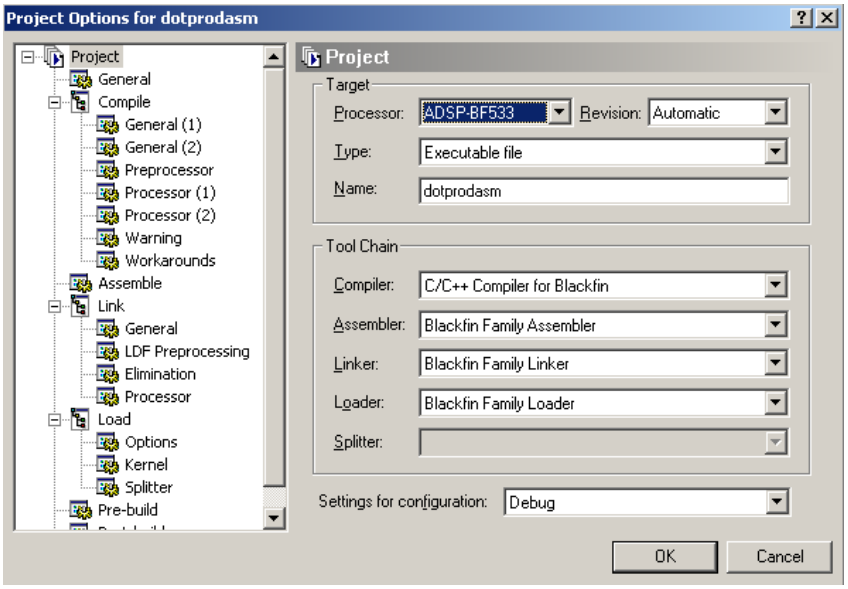
Súčasťou balíka **dotprodasm.zip** už však nie sú súbory VisualDSP++ projektu s príponami ***.dpj** a ***.mak**. Projekt v prostredí VisualDSP++ je možné vytvoriť nasledujúcim postupom[3]:

File->New->Project
v položke **General**
zadáme **meno** – napr. dotprodasm
zadáme **cestu** do pracovného adresára kde máme zdrojové súbory
zvolíme voľbu Standard application
v položke Output Type
vyberieme cieľový procesor
v položke Add Startup Code/LDF
add an LDF and startup code
a stlačíme **Finish**

[3] Uvedený postup umožňuje aj vytvorenie LDF súboru, modifikáciou ktorého je možné definovať napr. umiestnenie kódu prípadne dát do zvolených časí pamäti (napr. L1 CODE, L1 DATA, ...), čím je možné výrazne zvýšiť napr. rýchlosť programu. V prípade, že pri definovaní projektu nezvolíme pridanie LDF súboru, bude použitý preddefinovaný LDF súbor pre zvolený cieľový procesor.

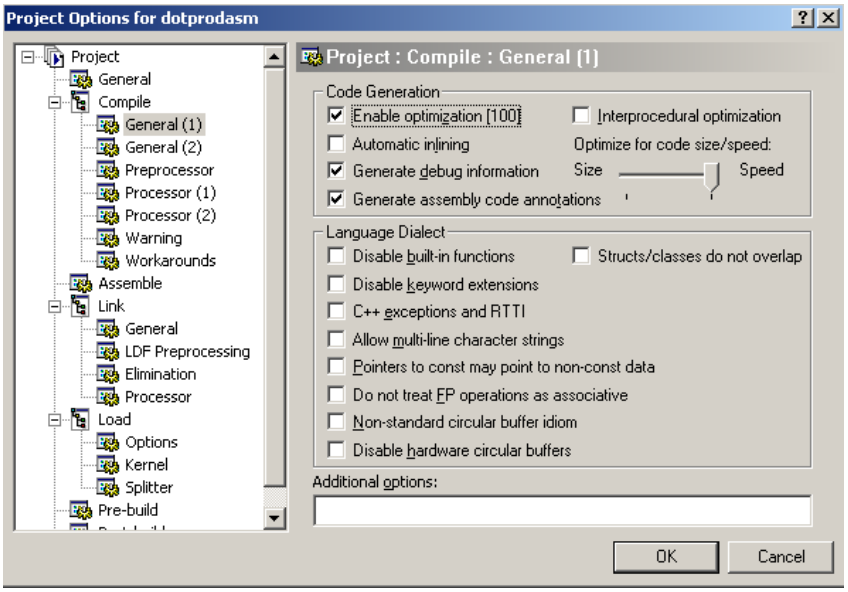
Project->Project Options

zvolíme cieľový procesor **ADSP-BF533,** prípadne ďalšie voľby podľa Obr.1.



Obr.1 Voľby Project Options

V položke **Compile** zvolíme ďalšie položky podľa Obr.2.



Obr.2 Voľby Project Compile

Potvrdíme stlačením **OK**.

V **okne Project** dodáme do položky **Source Files** (kliknutie myšou a voľba položky **Add File(s) to Folder ...**) zdrojové súbory *dotprod_main.c* a *dotprod_func.asm*.

3.2.2 VYTVORENIE LDF (LINKER DESCRIPTION FILE) SÚBORU

V predchádzajúcich krokoch boli definované zdrojové súbory, ktoré budú súčasťou projektu. Následne je potrebné zvoliť ich umiestnenie v pamäti procesora. Často je táto činnosť realizovaná prostredím VisualDSP++ automaticky. Prostredie VisualDSP++ umiestni kód preložený zo zdrojového C kódu do preddefinovanej časti pamäti procesora Blackfin, ktorá je preddefinovaná[4] pre segment kódovej pamäti. Keďže ASM funkcia

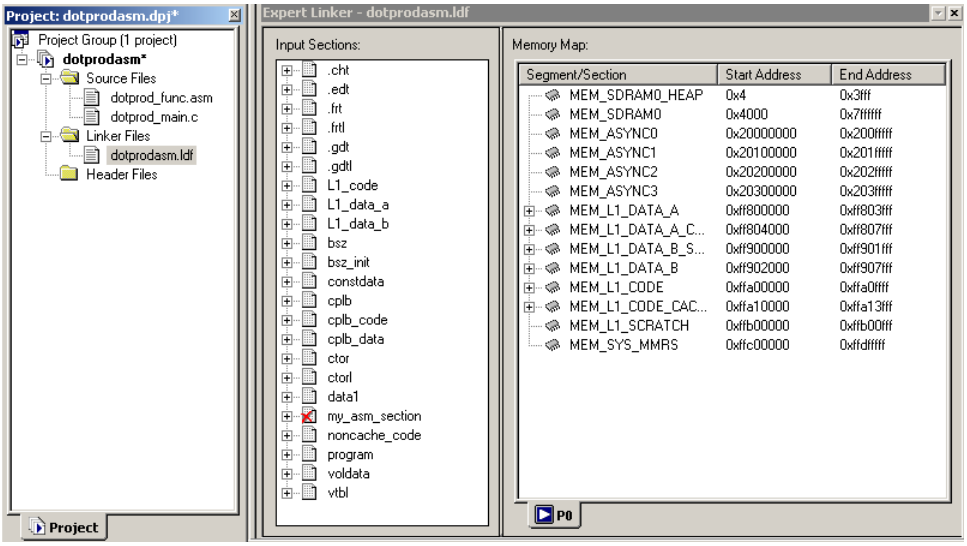
```
int a_dot_c_asm( int *a, int *c );
```

je umiestnená do **užívateľom definovaného** segmentu **my_asm_section** je potrebné definovať[5] do ktorej časti pamäte procesora Blackfin má byť uvedený segment definovaný. Umiestnenie je v prostredí ViusalDSP++ definované v LDF súbore. Prácu s LDF súborom výrazne zjednodušuje grafické rozhranie – **Expert Linker**, ktorý využijeme[6].
Potvrdením preddefinovaných volieb sa v pracovnom adresári vytvoril[7] súbor *dotprodasm.ldf*. Pomocou ľubovoľného editora je možné prezrieť si tento automaticky generovaný súbor. Grafická reprezentácia je dostupná po kliknutí na súbor *dotprodasm.ldf* v okne project. Kliknutím na pravú časť (Memory Map) a voľbou

View Mode ->Memory Map Tree
je možné získať grafické zobrazenie sekcií kódu a pamäte procesora zobrazené na Obr.3. V okne Input Sections je červeným krížikom zvýraznené, že sekcia *my_asm_section* nemá zvolené umiestnenie v pamäti procesora.

Umiestnenie v pamäti je možné definovať presunutím (po kliknutí na +) položky **dotprod_func.doj**[8] do sekcie MEM_L1_CODE[9]. Do tejto sekcie bol automaticky mapovaný aj programový kód main( ) funkcie **dotprod_main.doj**.

[4] Samozrejme preddefinované umiestnenie je možné zmeniť.
[5] Voľbou niektorého z preddefinovaných segmentov by bolo možné vyhnúť sa nutnosti definovať konkrétne umiestnenie. Pre typické optimalizované ASM funkcie je však výhodné mať priamu kontrolu na procesom umiestňovania **programových** prípadne **dátových** sekcií do pamäte procesora. Je tým možné výrazne **zvýšiť rýchlosť kritických častí kódu**, čo je dané rozdielnymi rýchlosťami **hierarchického** pamäťového priestoru procesorov Blackfin (pamäť L1, L2, externá SDRAM, ...).
[6] LDF súbor je textový súbor, ktorý využíva preddefinované príkazy a syntax. Je ho teda možné vytvárať a modifikovať aj ručne pomocou štandardného editora. Tento prístup však vyžaduje hlbšie znalosti. Alternatívou je modifikácia LDF súboru pomocou grafického rozhrania Expert Linkera.
[7] Súbor **sa nevytvorí** pokiaľ sú na počítači **nevhodne nastavené prístupové práva** pre program VisualDSP. V prípade problémov je možné použiť napr. účet s administratórskymi právami.
[8] Relatívny súbor (object code) funkcie definovanej v ASM súbore.
[9] Najrýchlejšia časť internej pamäte procesora Blackfin. Hierarchický pamäťový systém procesorov Blackfin bude podrobne analyzovaný v rámci prednášok.



Obr.3 Grafická reprezentácia súboru LDF (časť okna Memory Map je zvyčajne viditeľná až po rozšírení celého okna Expert Linker)

3.2.3 LADENIE PROJEKTU

Po definovaní LDF súboru je možné projekt preložiť voľbou **Project->Build (F7)** a následne ladiť podobne ako v predchádzajúcom cvičení (**Menu Debug**, výpis funkcie printf() v okne Output Window, …).

Príklad
Porovnajte rýchlosť výpočtu ASM funkcie a C funkcie z predchádzajúceho cvičenia. Na určenie počtu cyklov využite register ***CYCLES***[10] *dostupný v menu* ***Register->Core->Cycles***[11].

Pokiaľ bol pri meraní rýchlosti C-projektu z predchádzajúceho cvičenia použitý pôvodný nemodifikovaný projekt, je rozdiel rýchlosti medzi C funkciou (*a_dot_c( )*) a ASM funkciou (*a_dot_c_asm*( )) na výpočet skalárneho výpočtu pomerne veľký. Na základe tohto príkladu však nie je možné tvrdiť, že ASM realizácia funkcie je výrazne rýchlejšia. Pri pohľade na kód ASM funkcie (predovšetkým na jej najvnútornejší cyklus) je zrejmé, že nie je využitý paralelizmus dvoch MAC jednotiek v jadre Blackfin. Primárnym dôvodom veľkého rozdielu je to, že v projekte z minulého cvičenia nebola povolená optimalizácia prekladača. Na druhej strane nie je možné očakávať, že zapnutie optimalizácie prekladača zabezpečí generovanie kódu s rýchlosťou porovnateľnou s **kvalitne** ručne optimaliovaným kódom.

[10] Registre CYCLES a CYCLES2 sú **reálne** 32-bitové registre v jadre procesorov Balckfin. Umožňujú **presné meranie** počtu cyklov nie len počas simulácie ale aj pri ladení pomocou reálnych technických prostriedkov.
[11] Podobne ako aj u iných registrov je možné **zvoliť tvar zobrazovania** (celočíselný, hexadecimálny, binárny, ...) pre každú skupinu registrov kliknutím myšou na príslušné okno. Hodnotu registrov je tiež možné **editovať**, čo je napr. výhodné pri meraní počtu cyklov, keď je možne pred vstupom do meranej časti kódu register CYCLES vynulovať.

Príklad
Porovnajte rýchlosť výpočtu ASM funkcie a ***optimalizovanej*** *C funkcie z predchádzajúceho cvičenia. Optimalizácia prekladača je definovaná v menu* ***Project->Project Options->Compile*** *v položke* ***Enable optimization****.*

Príklad
Pri ladení funkcie a_dot_c_asm( ) zobrazte vnutorné rgistre procesora Blackfin (R0-R7, P a I registre, ...) a sledujte ich modifikácie počas krokovania programu .

3.3 DSP PROJEKT S VYUŽITÍM KNIŽNIČNÝCH C FUNKCIÍ

Tvorba efektívneho kódu pre moderné DSP je pomerne náročná úloha vyžadujúca veľmi dobré zvládnutie architektúry cieľového DSP, asemblera príslušného DSP a tiež podrobnú znalosť implementovaného algoritmu. Súčasťou prostredia VisualDSP++ je aj množina optimalizovaných knižničných funkcií (optimalizovaných v asembleri) [6], ktoré je možné využívať v aplikačných C programoch a tak výrazne zjednodušiť vývoj reálnych aplikácií. Projekt FIR_LIB je príkladom, ako je možné s minimálnym úsilím realizovať efektívnu filtráciu bloku dát pomocou FIR filtra.

3.4 ZDROJOVÉ KÓDY PROJEKTU FIR_LIB

Projekt **fir_lib.zip** obsahuje 3 zdrojové súbory. Súbory **coefs.dat** a **in.dat** obsahujú[12] koeficienty použitého FIR filtra a vstupné vzorky ktoré spracováva hlavný program *main( )* v súbore **fir.c**:

```
#include <fract.h>                          // definicie novych datovych typov
#include <filter.h>                         // prototypy pouzitych kniznicnej funkcii a makier

#define VEC_SIZE 256                        // dlzka spracovavaneho vstupneho vektora
#define NUM_TAPS 8                          // pocet koeficientov FIR filtra

fract16 in[VEC_SIZE] = {                    // definovanie vstupneho vektora
        #include "in.dat"
};
fract16 coefs[NUM_TAPS] = {                 // definovanie koeficientov pouziteho FIR filtra
        #include "coefs.dat"
};
fract16 delay[NUM_TAPS];                    // oneskorovacia linka pre FIR filter
fract16 out[VEC_SIZE + NUM_TAPS - 1];       // vector pre vystupne vzorky

fir_state_fr16 state;                       // struktura pre stavovu premennu FIR filtra

int main() {
        fir_init(state, coefs, delay, NUM_TAPS, 1);    // inicializacia stavovej premennej
        fir_fr16(in, out, VEC_SIZE, &state);           // filtracia vektora in FIR filtrom
}
```

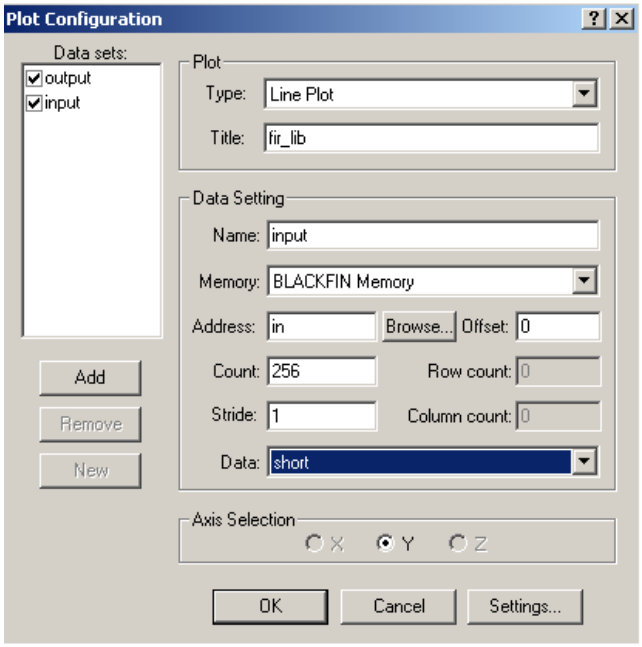
Použité knižničné funkcie resp. makrá sú definované hlavičkovým súborom **<filter.h>**. Zátvorky < > určujú, že sa jedná o systémový hlavičkový súbor. Systémové hlavičkové súbory sú uložené v adresári **..\VisualDSP 4.5\Blackfin\Include**

[12] Zahrnutie vstupných vzoriek pomocou direktívy **#include "in.dat"** je pomerne netypické, z hľadiska syntaxe jazyka C však úplne korektné riešenie. Použité riešenie je výhodné predovšetkým v počiatočnej fáze ladenia aplikácií v simulátore. V reálnych aplikáciách sú zvyčajne vstupné resp. výstupné dáta čítané resp. zapisované cez vstupno-výstupné kanály (AD, DA prevodníky a pod.).

Podrobnejšie informácie o použitých knižničných funkciách je možné nájsť v súbore filter.h, on-line manuále a v [6].

3.5 LADENIE PROJEKTU FIR_LIB

Projekt FIR_LIB je možné preložiť a ladiť štandardným postupom opísaným v predchádzajúcej podkapitole. Prostredie VisualDSP++ však poskytuje ďalšie ladiace nástroje pre **vizualizáciu väčšieho množstva dát**, ktoré je možné využiť pri ladení projektov typických DSP projektov pre ČSS.
Pomocou **okna Plot** (menu **View->Debug Windows -> Plot**) je možné graficky zobraziť obsah pamätí v ktorých sú uložené vektory dát. V našom prípade to je vektor vstupných (pole *in[ ]*) a výstupných (pole *out[ ]*) dát. Tieto údaje sú v našom prípade uložené ako 16-bitové premenné, čomu v danej implementácii jazyka C zodpovedá premenná typu *short*. Konfigurácia okna Plot je zobrazená na Obr.4.



Obr.4 Konfigurácia Plot zobrazenia pre projekt FIR_LIB

Po simulácii programu (Run alebo F5) je možné zobraziť graficky obsah pamäti. V okne plot je možné meniť rozlíšenie zobrazenia (zoom), zapnúť graficky kurzor, exportovať zobrazenie do JPEG súboru, meniť parametre zobrazenia (napr. farbu zobrazenia, vzorkovaciu frekvenciu pre zobrazenie na osi x, ...), typ zobrazenia (napr. zvoliť zobrazenie spektra, ...) a pod.

Príklad
S použitím zobrazenia spektra v okne Plot zistite aký je tvar vstupného signálu a ako sú modifikované FIR filtrom jeho jednotlivé zložky.

LITERATÚRA

[1] http://www.analog.com/processors/blackfin/evaluationDevelopment/blackfinProcessorTestDrive.html

[2] http://www.kemt.fei.tuke.sk/adsp/visualdsp/index.html

[3] http://forms.analog.com/Form_Pages/processors/visualDSPTestDrive.asp

[4] http://www.analog.com/processors/technicalSupport/technicalLibrary/index.html

[5] VisualDSP++ 4.5 Getting Started Guide. Analog Devices, Inc., 2006.

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