Ciele cvičenia

V priebehu cvičenia sa oboznámite s dvojicou moderných nástrojov pre digitálne spracovanie signálov na platforme C6000 firmy TI's. Jedná sa o integrované vývojové prostredie (IDE) Code Composer Studio (CCS) C6000 a vývojovú dosku „DSP Starter Kit" (TMS320C67113DSK). Na troch jednoduchých príkladoch sa naučíte základom práce v CCS a ich prostredníctvom otestujete činnosť DSK.

1. Úvod

Digitálne signálové procesory všeobecne, a teda aj rodina procesorov TMS320C6x, sú rýchle mikroprocesory so špeciálným typom architektúry a inštrukčným súborom vhodným pre spracovanie signálov. DSP využívame v širokom rozsahu aplikácii od komunikačných a riadiacich systémov až po spracovanie obrazu a rozpoznávanie či syntézu hlasu. Existujú implementácie aj v oblastiach robotiky, umelej inteligencie, neurónových štruktúr a v mnohých ďalších. Nájdeme ich v mobilných telefónoch, modemoch, faxoch, audio a video systémoch, atď. DSP sú určené predovšetkým pre spracovávanie signálov v reálnom čase, čo znamená, že procesor musí byť schopný plynule reagovať na rôzne externé udalosti v ideálnom prípade bez oneskorenia, v skutočnosti s konštantným časovým oneskorením daným rýchlosťou a matematickým výkonom procesora. Pod externou udalosťou obvykle rozumieme zachytenie vzorky analógového vstupného signálu.
Elektronické obvody s diskrétnymi analógovými súčiastkami sú obvykle pomerne citlivé na zmeny teploty. Na činnosť DSP systémov však teplota vplýva len nepatrne v dôsledku čoho dosahujú digitálne systémy vynikajúcu dlhodobú stabilitu.
Využitie DSP sa orientuje už od svojho vzniku predovšetkým na oblasť spracovania audio signálov, teda na signály vo frekvenčnom rozsahu 0Hz až 20kHz.
Pravdepodobne najčastejšie skloňovanými výrazmi v terminológii DSP je vzorkovanie a s ním súvisiaca vzorkovacia frekvencia a aliasing. Pri spracovávaní hlasu postačuje vzorkovacia frekvencia 8kHz, teda vzdialenosť medzi vzorkami (perióda vzorkovania), je 125µs. Spracovanie hudby je, samozrejme, náročnejšie a vyžaduje vzorkovaciu frekvenciu minimálne 40kHz, v praxi sa používa frekvencia 44,1kHz. V súčasnosti sú k dispozícii aj A/D moduly so vzorkovacou frekvenciou v ráde MHz.
Základný DSP systém pozostáva z AD prevodníka, ktorým vzorkujeme signál a získavame jeho digitálnu reprezentáciu. Jednotlivé vzorky potom spracúvame signálovým procesorom a takto získaný tok dát potom konvertujeme DA prevodníkom späť na analógový signál. Na vstupe DSP systému musí byť zaradený tzv. antialiasingový filter a výstupný rekonštrukčný filter, ktorý zabezpečuje vyhladenie výstupného signálu.

2. Moderné pracovisko pre vývoj DSP aplikácií

V tejto stati nájdete opis štandardného pracoviska pre vývoj moderných DSP aplikácii na báze platformy C6000, na ktorom budete pracovať v rámci tohto a nasledujúcich cvičení. Základ pracoviska tvorí vývojový balík DSK, ktorý obsahuje

a) integrované vývojové prostredie Code Composer Studio, poskytujúce potrebnú softvérovú podporu zahŕňajúcu kompilátor jazyka C, assembler, linker, debugger a pod,
b) samotnú vývojovú dosku,
c) USB kábel,
d) napájací zdroj.

Pri vývoji DSP aplikácií je nevyhnutné mať k dispozícii aspoň dvojkanálový osciloskop a zdroj signálu – generátor funkcií. Výhodný je aj analyzátor spektra.

2.1 Vývojová doska TMS320C6713 DSK

Starter kit TMS320C6713 DSK zahŕňa všetky potrebné hardverové a softverové prostriedky pre štúdium spracovávania signálov v reálnom čase. Jedná sa teda o kompletný DSP systém, ktorého jadro tvorí signálový procesor TMS320C6713 s plávajúcou rádovou čiarkou a 32-bitový sigma-delta kodek TLV320AIC23 zabezpečujúci AD a DA prevod a antialiasingovú a rekonštrukčnú filtráciu signálu. Vzorkovaciu frekvenciu kodeku možno meniť v rozsahu 8 - 96kHz a je odvodená od hodinového obvodu s frekvenciou taktu 12MHz.
Procesor C6713 je založený na architektúre VLIW (Very Long Instruction Word) špeciálne navrhnutej pre náročné matematické operácie. Interná programová pamäť je usporiadaná tak, že v každom strojovom cykle môže byť naraz rozpracovaných až osem 32-bitových inštrukcií, čo závisí od optimalizácie programu v zmysle zreťazenia (PIPELINE). Procesor disponuje 264kB internej pamäte, ôsmimi funkčnými a vykonávacími jednotkami: šesť aritmeticko-logických jednotiek a dve násobičky, 32-bitovou adresovou zbernicou, ktorá umožňuje adresovanie až 4GB priestoru a dvomi skupinami 32-bitových registrov pre všeobecné použitie. C6713 umožňuje operácie s plávajúcou rádovou čiarkou aj operácie s pevnou rádovou čiarkou.
DSK disponuje 16MB synchrónnou dynamickou RAM (SDRAM) a 256kB flash ROM. štyri konektory typu JACK na doske umožňujú vstup a výstup: MIC IN (mikrofónový vstup), LINE IN (signálový vstup), LINE OUT (signálový výstup), HEADPHONE (výstup pre slúchadlá). Pre jednoduchú verifikáciu činnosti dosky slúžia aj štyri DIP prepínače. Stav troch z nich môžeme detekovať vo vlastnom programe. Systémové hodiny dosky majú frekvenciu 225MHz. Hardvér dosky uzatvára napäťový regulátor poskytujúci napätie 1,26V pre jadro procesora a 3,3V pre pamäť a periférie.

2.2 Code Composer Studio C6000

Softvér Code Composer Studio C6000 (v ďalšom len CCS) patrí do kategórie IDE (Integrated Development Environment) a poskytuje užívateľovi komplexnú podporu pre tvorbu a ladenie vlastných aplikácií. CCS zahŕňa všetky potrebné nástroje, ktoré boli v minulosti samostatnými programami:

1. *textový editor*, v ktorom môžeme písať vlastný kód a prezerať existujúce zdrojové súbory, hlavičkové súbory a knižnice,
2. *kompilátor jazyka C*, ktorým po napísaní zdrojového kódu uskutočníme preklad z jazyka C do jazyka symbolických adries,
3. *assembler*, ktorým preložíme strojový kód z jazyka symbolických adries do strojového objektového kódu,
4. *linker*, ktorým zlúčime jednotlivé objektové súbory nášho projektu, objektové súbory knižníc a objektové hlavičkové súbory do vykonateľného výstupného súboru, ktorý môžeme uložiť do pamäte procesora, spustiť a ladiť ho.

Vývojové prostredie umožňuje jednoducho vkladať nové súbory do projektu a spravovať už vložené súbory prostredníctvom manažéra projektu, nastavovať vlastnosti kompilátora a linkera, nastavovať vlastnosti samotného prostredia, vkladať do zdrojového súboru breakpointy, sledovať stav premenných, pamäte a registrov. Môžeme zároveň prezerať zdrojový kód v jazyku C aj v JSA. Výhodnou vlastnosťou CCS je grafické zobrazovanie spracovávaného signálu formou kvázidigitálneho osciloskopu a monitorovanie času vykonávania programu, prípadne jeho sekvencií. Program môžeme krokovať, pričom môžeme využívať vnorenie do funkcií alebo ich úplne vykonávať, prípadne vykonávať len sekvenciu danú breakpointami. Aby sme mohli graficky zobrazovať spracovávaný signál

v reálnom čase, musíme využiť podporu RTDX (Real-Time Data Exchange). RTDX umožňuje dátovú výmenu medzi hostiteľským počítačom a vývojovou doskou v reálnom čase bez potreby zastavovania jej činnosti. Pri profesionálnom vývoji využívame rozhranie JTAG (Joint Team Action Group), ktoré zabezpečuje emuláciu priamo na čipe procesora a umožňuje riadenie a monitorovanie vykonávania programu v reálnom čase. Vývojová doska C6713 DSK toto rozhranie obsahuje, ale pre cvičenia nemá praktický význam.

CCS je podľa odporúčania nainštalovaný v adresári c:\ti. Po pripojení napájacieho napätia začne procesor v rámci rýchleho testu DSK počítať od nula do pätnásť, pričom aktuálna hodnota je zobrazovaná na štvorici LED. Pri inštalácii CCS je možné zvoliť inštaláciu úplnej dokumentácie k procesoru C6713, ktorá zahŕňa dokumentáciu všetkých spomínaných nástrojov CCS, interaktívnu výučbu prostredia CCS, kompilátora a vlastností nástrojov RTDX a DSP/BIOS. V dokumentácii tiež nájdete podrobný popis inštrukčného súboru a registrov procesora. Všetky materiály sú k dispozícii vo formáte pdf. CCS zahŕňa tiež niekoľko jednoduchých príkladov ako je napr. test DSK, spracovanie audiosignálu a príklady súvisiace s flash pamäťou.

V krátkosti spomeňme obsah podadresárov adresára c:\ti:

1. docs: dokumentácia a manuály,
2. myprojects: adresár pre ukladanie projektov užívateľa,
3. bin: obsahuje niekoľko utilít CCS,
4. c6000\bios: podporné súbory pre DSP/BIOS,
5. c6000\cgtools: nástroje pre tvorbu a správu kódu (Code Generation Tools),
6. c6000\examples: obsahuje príklady zahrnuté v CCS,
7. c6000\RTDX: podporné súbory pre dátovú výmenu v reálnom čase.

Pri práci v CCS sa budeme stretávať s niekoľkými typmi súborov s rozdielnymi príponami:

1. file.pjt: súbor projektu, v ktorom sú uložené nastavenia CCS pre daný projekt,
2. file.c: zdrojový program v jazyku C,
3. file.asm: zdrojový program v jazyku symbolických adries vytvorený užívateľom, kompilátorom alebo optimalizačnou rutinou,
4. file.sa: zdrojový program v lineárnom assembleri; slúži ako vstupný súbor pre lineárny optimizér produkujúci optimalizovaný zdrojový súbor v jazyku symbolických adries,
5. file.h: hlavičkový súbor,
6. file.lib: knižnica, napr. podpora pre „run-time“ rts6701.lib,
7. file.cmd: príkazový súbor linkera, ktorý fyzicky mapuje sekcie programu do pamäte,
8. file.obj: objektový súbor vytvorený assamblerom preložením zdrojového súboru *.asm,
9. file.out: vykonateľný súbor strojového kódu vytvorený linkerom; je možné ho priamo uložiť do pamäte procesora a spustiť ho.

2.3 Rýchly test činnosti DSK

Po spustení CCS vykonajte: GEL → Check DSK → Quick Test. Tento test používame pri overovaní inštalácie CCS a správnej činnosti DSK. Pri vykonávaní testu trikrát bliknú LED a zobrazí sa nasledujúca správa:

Switches: 16
Revision: 2
Target is OK

To znamená, že prvé štyri prepínače, USER_SW1,USER_SW2 a USER_SW3, USER_SW4 sú v pozícii ON. Prepínačmi môžete nastaviť hodnotu 0 až 15 a overiť si zopakovaním rýchleho testu načítanie správnej hodnoty. Vo vlastnom programe potom môžete podmieniť vykonávanie jeho sekvencií nastavením týchto prepínačov. Rýchly test umožňuje verifikovať

správnu činnosť hlavných komponentov DSK: INternú, externú, flash pamäť; dva viacnásobné seriálové porty so zásobníkom (McBSP); DMA; codec na doske; LED.

2.4 Podporné súbory pre činnosť DSK

Pri vývoji vlastných aplikácii a ich testovaní na DSK budeme musieť k nášmu projektu pripojiť niekoľko dôležitých súborov, ktorých obsah si v krátkosti objasníme:

1. *C6713dsk.cmd*: príklad príkazového súboru linkera upraveného pre použitie v súvislosti s možnosťami DSK,
2. *C6713dskinit.h*: hlavičkový súbor s prototypmi funkcií,
3. *C6713dskinit.c*: obsahuje niekoľko funkcií použitých v príklade codec_poll, ktorý je zahrnutý v CCS,
4. *Vectors_intr.asm*: modifikovaná verzia súboru vectors.asm zahrnutého v CCS, ktorý obsahuje inicializáciu vektorov prerušení. Dvanásť prerušení INT4 - INT15 je dostupných, z ktorých INT11 je použitý pre program riadený prerušením.
5. *Vectors_poll.asm*: modifikovaný súbor vektora používajúci výzvy.
6. *rts6700.lib, dsk6713bsl.lib, csl6713.lib*: knižnica reálneho času, knižnica pre podporu dosky a pre podporu čipu. Tieto knižnice sú súčasťou CCS a nachádzajú sa v adresároch *C6000\cgtools\lib*, *C6000\dsk6713\lib*, a *C6000\bios\lib*.

Úloha č.1: Generovanie sínusového signálu z tabuľky ôsmych vzoriek

V tejto úlohe sa budeme zaoberať generovaním sínusového signálu s frekvenciou 1kHz tabuľkovou metódou. Metóda je založená na preddefinovanej tabuľke funkčných hodnôt, z ktorej jednotlivé hodnoty v pravidelných intervaloch posielame na DA prevodník. Čím viac hodnôt má tabuľka, tým kvalitnejší (z hľadiska odstupu signálu od šumu a harmonického skreslenia) je generovaný signál pri danom rekonštrukčnom filtri. V rámci úlohy sa oboznámime so základnými možnosťami CCS, ako sú správa projektu, editovanie a kompilovanie zdrojových súborov, ukladanie programu do pamäte procesora a jeho testovanie.

Vytvorenie projektu

V tejto stati na konkrétnom príklade ukážeme, ako vytvoriť projekt a pripojiť k nemu všetky potrebné zdrojové a pomocné súbory.

1. Najprv vytvoríme súbor projektu s názvom sine8_intr.pjt: Project→ New. Zobrazí sa dialógové okno, v ktorom vypíšete meno projektu a zvolíte jeho umiestnenie v preddefinovanom adresári C:\ti\myprojects\sine8_intr\. Zvoľte typ projektu: executable (.out) a cieľový procesor TMS320C67xx,
2. Pripojíme potrebné súbory k projektu: Project→ Add Files to Project. V dialógu prejdeme do adresára sine8_intr a zvolíme zobrazovanie zdrojových súborov v jazyku C (*.c). Označíme súbory C6xdskinit.c a sine8_intr.c a otvoríme ich (vložíme do projektu). V okne manažéra projektu si môžeme overiť, že sa príslušné súbory pričítali do zdrojovej časti projektu. Zopakujeme: Project Add Files to Project zvolíme zobrazovanie zdrojových súborov v JSA (*.asm) a otvoríme (vložíme) súbor vectors_11.asm. Tieto kroky zopakujeme a vložíme do projektu ešte súbor C6xdsk.cmd,
3. Zopakujeme bod 2), ale v dialógu zvolíme zobrazovanie objektových súborov a knižníc z adresára "**support**", alebo prejdeme do adresára
   C:\ti\c6000\dsk6713\lib a otvoríme (vložíme) knižnicu dsk6713bsl.lib,
   C:\ti\c6000\cgtools\lib a otvoríme (vložíme) knižnicu rts6701.lib,

C:\ti\c6000\bios\lib a otvoríme (vložíme) knižnicu csl6713.lib,

4. V okne Project view môžeme skontrolovať, či boli k projektu pripojené všetky zvolené súbory. V tomto okne tiež môžeme vidieť súbor dsk6416_6713.gel, ktorý sa k projektu pripája hneď po jeho vytvorení a zabezpečuje inicializáciu DSK,
5. K projektu potrebujeme ešte pripojiť niekoľko hlavičkových súborov spomínaných v predchádzajúcom. Môžme postupovať štandardným spôsobom ako v bode 2) alebo využiť príkaz: Project→ Scan All Dependencies, ktorý pripojí k projektu požadované súbory C6xdsk.h, C6xinterrupts.h, C6xdskinit.h a C6x.h automaticky. Tento príkaz prezrie pripojené zdrojové súbory a nájde v nich odkazy na hlavičkové alebo iné vkladané súbory a pokiaľ ich nájde v adresárovej štruktúre c:\ti, automaticky ich pripojí k projektu.

Súbory je možné do projektu vkladať aj metódou „drag and dropp“ z dialógu Open do okna Project view. Hlavičkové a zdrojové súbory je možné editovať dvojitým kliknutím na ne.

Nastavenie kompilátora

Pred kompiláciou projektu je nutné nastaviť vlastnosti kompilátora v dialógovom okne vyvolanom príkazom: Project→ Build Options. Najskôr je potrebné zadefinovať základnú charakteristiku správania sa kompilátora v menu „Category“. Keďže pre začiatok nepotrebujeme špeciálne funkcie, nastavíme „Basic“. Je zrejmé, že z hľadiska rozsahu nie je možné venovať sa popisu nastavenia kompilátora komplexne, ale kategóriu Basic budeme používať priamo na cvičení a z tohto dôvodu sa jej budeme venovať podrobnejšie.
V ľavej časti sa objaví submenu, kde v položke „Target Version“ nastavíme „Default“, v položke „Generate Debug Info“ nastavíme „Full Symbolic Gebug“, v položke „Opt Speed vs Size“ nastavíme Speed Most Critical“, v položke „Opt Level“ nastavíme „None“ a v položke „Program Level Opt“ nastavíme tiež „None“. Výsledný súhrn našej voľby je zobrazený aj v „textovej“ forme v príkazovom okne:
-g –k –s –fr”C:\ti\myprojects\sine8_led”
Voľba –k umožňuje zachovať zdrojový súbor po preklade, voľba –g je vysvetlená nižšie a používa sa obvykle v súčinnosti s voľbou –s, ktorá súvisí s previazaním komentárov optimalizátora a zdrojového kódu v C so zdrojovým kódom v JSA. Voľba –fr nesúvisí priamo s kompilátorom, ale umožňuje v príkazovom riadku zadať cestu k adresáru, v ktorom budú uložené objektové súbory.
V položke „Target Version“ máme možnosť zvoliť procesor z rodiny C6x, pre ktorý bude generovaný kód. Ak si nevyberieme sami, bude kompilátor predpokladať procesor C62x. Pri voľbe „Default“ bude kompilátor generovať kód s pevnou rádovou čiarkou.
V položke „Generate Debug Info“ môžme zvoliť jednu z nasledujúcich položiek:

- Full Symbolic Debug (-g) – kompilátor generuje symbolické direktívy pre ladenie zdrojového programu v jazyku C alebo v JSA,
- Function Profile Debug (-gp) – povoľuje profiláciu kódu pri jeho optimalizácii,
- Dwarf Debug Info (-gw) – kompilátor generuje symbolické direktívy pre ladenie zdrojového programu v jazyku C/C++ alebo v JSA. Navyše generuje informáciu o ladení „DWARF“,
- No Debug – zakáže ladenie kódu.

V položke „Opt Speed vs Size“ môžme vybrať jednu z piatich úrovní: „Speed Most Critical (no ms)”, „Speed More Critical (-ms0)”, „Speed Critical (-ms1)”, „Size Critical (-ms2)” a „Size Most Critical (-ms3)”, ktorých voľba zapríčiní, že kompilátor znižuje výkon v prospech redukcie dĺžky kódu.
V položke „Opt Level“ nastavujeme spôsob optimalizácie kompilátora:

- Register (-o0) – povolenie optimalizácie na úrovni registrov,
- Local (-o1) – povolenie lokálnej optimalizácie,
- Function (-o2) – povolenie optimalizácie na úrovni funkcií,
- File (-o3) – povolenie optimalizácie na úrovni súborov (maximálna optimalizácia),
- None – zakázanie optimalizácie.

V položke „Program Level Opt“ bližšie špecifikujeme optimalizáciu z hľadiska programu využitím voľby –pm v súčinnosti s voľbou –o3. S výnimkou položky None, tieto položky dovoľujú voľbu –pm, vďaka čomu je možné bližšie určiť podmienky, ktoré umožnia kompilátoru zlepšiť optimalizáciu:

- No External Var Refs – špecifikuje, že modul obsahuje funkcie volané mimo zdrojového kódu poskytnutého kompilátoru, ale nepoužíva premenné modifikované mimo zdrojového kódu (-op3),
- No External Func Refs – špecifikuje, že modul obsahuje premenné modifikované mimo zdrojového kódu poskytnutého kompilátoru, ale nepoužíva funkcie volané mimo zdrojového kódu (-op2),
- No External Func/Var Refs – špecifikuje, že modul neobsahuje premenné ani funkcie modifikované alebo volané mimo zdrojového kódu poskytnutého kompilátoru (-op1).
- External Func/Var Refs – špecifikuje, že modul obsahuje premenné a funkcie modifikované alebo volané mimo zdrojového kódu poskytnutého kompilátoru (-op0).

Uvedený opis sa týka len nastavení, ktoré použijeme na cvičení, v prípade potreby je možné preštudovať ostatné nastavenia kompilátora v nápovede ku CCS C6000.

Nastavenie linkera

V okne Build Options prejdeme na záložku Linker a zvolíme kategóriu Basic. Zaškrtneme položky „Suppress Banner“ a „Exhaustively Read Libraries“. V Output Module zvolíme „Absolute Executable“, v „Output Filename“ vypíšeme sine8_intr.out, v prípade, že ho nešpecifikujeme, použije linker názov projektu a v „Autoinit Model“ zvolíme „Run-time Autoinitialization“. Výsledný súhrn našej voľby je zobrazený opäť v „textovej“ forme v príkazovom okne, kde ho môžeme editovať a prípadne dopĺňať:
-q –c –o“sine8_led.out“ –x
Výsledný vykonateľný súbor je možné ukladať do špeciálneho podadresára „Debug“. Podrobnosti o nastavení linkera nájdeme opäť v nápovede ku CCS.

Popis zdrojového kódu a prilinkovaných súborov

Predtým, ako budeme pokračovať, je dôležité aspoň čiastočne pochopiť štruktúru a obsah súborov, s ktorými budeme pracovať. Najskôr sa zameriame na výpis zdrojového súboru sine8_intr.c, ktorý obsahuje v podstate kompletný program pre generovanie sinusového signálu z tabuľky:

```
//Sine8_LED.c  Sine generation with DIP switch control

#include "dsk6713_aic23.h"                //support file for codec,DSK
Uint32 fs = DSK6713_AIC23_FREQ_8KHZ;      //set sampling rate
short loop = 0;                           //table index
short gain = 10;                          //gain factor
short sine_table[8]={0,707,1000,707,0,-707,-1000,-707};
//sine values
```

```
void main()
{
 comm_poll();                              //init DSK,codec,McBSP
 DSK6713_LED_init();                       //init LED from BSL
 DSK6713_DIP_init();                       //init DIP from BSL
 while(1)                                  //infinite loop
 {
  if(DSK6713_DIP_get(0)==0)                //=0 if DIP switch #0 pressed
  {
      DSK6713_LED_on(0);                   //turn LED #0 ON
    output_sample(sine_table[loop]*gain);//output for on-time sec
    if (loop < 7) ++loop;                  //check for end of table
    else loop = 0;                         //reinit loop index
  }
  else DSK6713_LED_off(0);                 //turn LED off if not pressed
 }                                         //end of while (1) infinite loop
}
                                           //end of main
```

Stručne o činnosti: tabuľku hodnôt reprezentuje pole `sin_table` typu `short`, ktoré je inicializované hodnotami funkcie sin(t), kde t = 0, 45, 90, 135, 180, 225, 270 a 315 stupňov, pričom sú normované na hodnotu 1000. Jadro programu tvorí funkcia `main()`, v ktorej sa nachádzajú len dve ďalšie funkcie. Funkcia `comm_poll()` zabezpečuje inicializáciu vývojovej dosky, kodeku TLV320AIC23 a dvoch multikanálových sériových portov s bufframi (McBSP). Príkaz `while(1)` spôsobí vznik nekonečnej slučky, počas ktorej čakáme na prerušenie. V prípade, že prerušenie nastane, vykoná sa obslužná rutina `c_int11()`, ktorá vyšle na AD prevodník aktuálnu vzorku z poľa násobenú ziskom uloženým v premennej `amplitude`. Všimnime si, že v zdrojovom súbore nemáme prototyp ani definíciu funkcií `output_sample()` a `comm_poll()`. Tieto nájdeme v súbore c6713dskinit.c:

```
//C6713dskinit.c Includes functions from TI in the C6713 CSL and C6713DSK
BSL



#include "C6713dskinit.h"
#define using_bios                 //if BIOS don't use top of vector table
extern Uint32 fs;                  //for sampling frequency



void c6713_dsk_init()              //dsp-peripheral initialization
{
DSK6713_init();                    //call BSL to init DSK-EMIF,PLL)

//handle(pointer) to codec
hAIC23_handle=DSK6713_AIC23_openCodec(0, &config);
DSK6713_AIC23_setFreq(hAIC23_handle, fs);  //set sample rate

//interface 32 bits toAIC23
MCBSP_config(DSK6713_AIC23_DATAHANDLE,&AIC23CfgData);

//start data channel again
MCBSP_start(DSK6713_AIC23_DATAHANDLE, MCBSP_XMIT_START | MCBSP_RCV_START |
      MCBSP_SRGR_START | MCBSP_SRGR_FRAMESYNC, 220); }
```

```
void comm_poll()              //added for communication/init using polling
{
      poll=1;                                    //1 if using polling
   c6713_dsk_init();                             //init DSP and codec
}

void comm_intr()              //for communication/init using interrupt
{
      poll=0;                        //0 since not polling
   IRQ_globalDisable();              //disable interrupts
      c6713_dsk_init();              //init DSP and codec
      //McBSP1 Xmit
      CODECEventId=MCBSP_getXmtEventId(DSK6713_AIC23_codecdatahandle);

#ifndef using_bios                   //do not need to point to vector table
      IRQ_setVecs(vectors);          //point to the IRQ vector table
#endif                               //since interrupt vector handles this

      IRQ_map(CODECEventId, 11);     //map McBSP1 Xmit to INT11
      IRQ_reset(CODECEventId);       //reset codec INT 11
   IRQ_globalEnable();               //globally enable interrupts
      IRQ_nmiEnable();               //enable NMI interrupt
   IRQ_enable(CODECEventId);         //enable CODEC eventXmit INT11

      output_sample(0);         //start McBSP interrupt outputting a sample
}

void output_sample(int out_data)    //for out to Left and Right channels
{
      short CHANNEL_data;

      AIC_data.uint=0;                  //clear data structure
      AIC_data.uint=out_data;           //32-bit data -->data structure

/*The existing interface defaults to right channel. To default instead to
the left channel and use output_sample(short), left and right channels are
swapped In main source program use LEFT 0 and RIGHT 1 (opposite of what is
used here)*/

      //swap left and right channels
      CHANNEL_data=AIC_data.channel[RIGHT];
      AIC_data.channel[RIGHT]=AIC_data.channel[LEFT];
      AIC_data.channel[LEFT]=CHANNEL_data;

      //if ready to transmit
      if (poll) while(!MCBSP_xrdy(DSK6713_AIC23_DATAHANDLE));

      //write/output data
      MCBSP_write(DSK6713_AIC23_DATAHANDLE,AIC_data.uint);
}

void output_left_sample(short out_data)   //for output from left channel
{
      AIC_data.uint=0;                    //clear data structure
      //data from Left channel -->data structure
      AIC_data.channel[LEFT]=out_data;

      //if ready to transmit
      if (poll) while(!MCBSP_xrdy(DSK6713_AIC23_DATAHANDLE));
```

```
      //output left channel
      MCBSP_write(DSK6713_AIC23_DATAHANDLE,AIC_data.uint);
}

void output_right_sample(short out_data)  //for output from right channel
{
      AIC_data.uint=0;                    //clear data structure
      //data from Right channel -->data structure
      AIC_data.channel[RIGHT]=out_data;

      //if ready to transmit
      if (poll) while(!MCBSP_xrdy(DSK6713_AIC23_DATAHANDLE));
      //output right channel
      MCBSP_write(DSK6713_AIC23_DATAHANDLE,AIC_data.uint);
}

Uint32 input_sample()                                 //for 32-bit input
{
      short CHANNEL_data;

      //if ready to receive
      if (poll) while(!MCBSP_rrdy(DSK6713_AIC23_DATAHANDLE));
      AIC_data.uint=MCBSP_read(DSK6713_AIC23_DATAHANDLE);//read data

      //Swapping left and right channels (see comments in output_sample())
      CHANNEL_data=AIC_data.channel[RIGHT]; //swap left and right channel
      AIC_data.channel[RIGHT]=AIC_data.channel[LEFT];
      AIC_data.channel[LEFT]=CHANNEL_data;

      return(AIC_data.uint);
}

short input_left_sample()                       //input to left channel
{
      //if ready to receive
      if (poll) while(!MCBSP_rrdy(DSK6713_AIC23_DATAHANDLE));
      //read into left channel
      AIC_data.uint=MCBSP_read(DSK6713_AIC23_DATAHANDLE);
      return(AIC_data.channel[LEFT]);           //return left channel data
}

short input_right_sample()                      //input to right channel
{
      //if ready to receive
      if (poll) while(!MCBSP_rrdy(DSK6713_AIC23_DATAHANDLE));
      //read into right channel
      AIC_data.uint=MCBSP_read(DSK6713_AIC23_DATAHANDLE);
      return(AIC_data.channel[RIGHT]);          //return right channel data
}
```

V zdrojovom kóde c6713dskinit.c sú vložené používané hlavičkové súbory. Nájdeme tu inicializáciu DSK, konfiguráciu a povolenie prerušenia INT11 pri vysielaní dát sériovým portom, funkcie zabezpečujúce činnosť sériového portu a pomocné funkcie pre vstup a výstup vzoriek do a z kodeku AIC23. Zo zdrojového kódu sine8_intr.c je zrejmé, že na časovú synchronizáciu používame prerušenie INT11, ktoré v pravidelných intervaloch ($T = 1/F_s = 1/8000 = 0{,}125$ms) generuje kodek. Inicializačná funkcia pre tento prípad je `comm_poll()`. V tele funkcie voláme niekoľko ďalších podporných funkcií, ktoré nájdeme v hlavičkovom súbore c6xdskinterrupts.h. Okrem riadenia programu pomocou prerušenia je možný princíp

vyvolávania (polling). Takto orientovaný program kontinuálne vyvoláva test, či sú alebo nie sú dáta pripravené na prijatie, resp. vysielanie. Takýto program je však menej efektívny a preto sa vo väčšine prípadov využívajú na riadenie toku programu prerušenia. Vo funkcii pre čítanie zo sériového portu `mcbsp0_read()`, je obsah riadiaceho registra sériového portu (SPCR) logicky vynásobený (AND) hodnotou 0x2 z dôvodu testovania prvého bitu tohto registra. Podobne je vo funkcii pre zápis do sériového portu `mcbsp0_write()`, obsah SPCR logicky vynásobený hodnotou 0x20000 z dôvodu testovania sedemnásteho bitu tohto registra. Vstupné dáta sú ukladané do registra prijatých dát (DRR) a výstupné dáta sú vysielané prostredníctvom registra vysielaných dát (DXR). Oba registre sú súčasťou viackanálového sériového portu s bufframi (McBSP).
Ďalším súborom, ktorým sa budeme v rámci cvičenia zaoberať, je vectors_poll.asm:

```
*Vectors_poll.asm  Vector file for polling
   .global _vectors
   .global _c_int00
   .global _vector1
   .global _vector2
   .global _vector3
   .global _vector4
   .global _vector5
   .global _vector6
   .global _vector7
   .global _vector8
   .global _vector9
   .global _vector10
   .global _vector11
   .global _vector12
   .global _vector13
   .global _vector14
   .global _vector15

   .ref _c_int00                    ;entry address

VEC_ENTRY .macro addr
    STW   B0,*--B15
    MVKL  addr,B0
    MVKH  addr,B0
    B     B0
    LDW   *B15++,B0
    NOP   2
    NOP
    NOP
   .endm

_vec_dummy:
  B    B3
  NOP  5

 .sect ".vecs"
 .align 1024

_vectors:
_vector0:   VEC_ENTRY _c_int00      ;RESET
_vector1:   VEC_ENTRY _vec_dummy    ;NMI
_vector2:   VEC_ENTRY _vec_dummy    ;RSVD
_vector3:   VEC_ENTRY _vec_dummy
_vector4:   VEC_ENTRY _vec_dummy
_vector5:   VEC_ENTRY _vec_dummy
_vector6:   VEC_ENTRY _vec_dummy
```

```
_vector7:   VEC_ENTRY _vec_dummy
_vector8:   VEC_ENTRY _vec_dummy
_vector9:   VEC_ENTRY _vec_dummy
_vector10:  VEC_ENTRY _vec_dummy
_vector11:  VEC_ENTRY _vec_dummy
_vector12:  VEC_ENTRY _vec_dummy
_vector13:  VEC_ENTRY _vec_dummy
_vector14:  VEC_ENTRY _vec_dummy
_vector15:  VEC_ENTRY _vec_dummy
```

Tento súbor je napísaný v JSA a obsahuje definíciu vektorov prerušení, čím máme na mysli zadefinovanie nepodmienených skokov na príslušné obslužné rutiny daných prerušení (ISR-Interrupt Service Routine). Môžeme sa presvedčiť, že pri vzniku prerušenia INT11 dôjde k presmerovaniu toku programu na ISR návestie `_c_int00` zadefinované v súbore sine8_led.c. Posledným súborom je príkazový súbor linkera c6713dsk.cmd:

```
/*C6713dsk.cmd  Linker command file*/

MEMORY
{
  IVECS:    org=0h,           len=0x220
  IRAM:     org=0x00000220,   len=0x0002FDE0 /*internal memory*/
  SDRAM:    org=0x80000000, len=0x00100000 /*external memory*/
  FLASH:    org=0x90000000, len=0x00020000 /*flash memory*/
}

SECTIONS
{
  .EXT_RAM :> SDRAM
  .vectors :> IVECS     /*in vector file*/
  .text    :> IRAM      /*Created by C Compiler*/
  .bss     :> IRAM
  .cinit   :> IRAM
  .stack   :> IRAM
  .sysmem  :> IRAM
  .const   :> IRAM
  .switch  :> IRAM
  .far     :> IRAM
  .cio     :> IRAM
  .csldata :> IRAM
}
```

Tento súbor umožňuje užívateľovi namapovať pamäť procesora a umiestniť sekcie programu na zvolených adresách. Formát zápisu objektových súborov (zdrojové súbory po preklade assemblerom, knižnice) a riadenie ich linkovania prostredníctvom *.cmd súborov pochádza z dielne vývojárov OS UNIX a nazýva sa COFF (Common Object File Format). Platformu COFF následne prebrali mnohí výrobcovia DSP a mikroradičov.

Kompilácia a testovanie programu

Všetky potrebné súbory máme pripojené k projektu a urobili sme všetky potrebné nastavenia, takže môžeme pristúpiť ku kompilácii a následnému zlinkovaniu súborov. Zvolíme Project→ Build All, resp. Project → Rebuild All, v prípade, že sme už predtým projekt kompilovali a vykonali sme nejaké úpravy v zdrojovom kóde, pričom využívame to, že budú znovu kompilované len modifikované súbory. Oba príkazy nájdeme aj na nástrojovej lište. Výsledkom procesu je výstupný vykonateľný strojový kód v súbore sine8_intr.out, ktorý

môžeme uložiť do pamäte procesora a spustiť. Činnosť kompilátora, assemblera a linkera je riadená pravidlami, ktoré sme nastavili v dialógu Build Option. Vo vytvorenom súbore cc_build_Debug.log nájdeme výpis kompilovaných a prekladaných súborov a výpis používaných podporných funkcií.
Ďalším krokom je uloženie programu do pamäte procesora, čo vykonáme príkazom File→ Load Program. V dialógu vyberieme príslušný súbor *.out a potvrdíme. V CCS je možné nastaviť, aby bol program ukladaný do pamäte procesora hneď po zlinkovaní. Program spustíme príkazom Debug→ Run, alebo použijeme ikonu „bežiaceho muža“ na zvislej nástrojovej lište úplne vľavo. Výstupný konektor (J6) na DSK prepojíme so vstupom osciloskopu, na ktorom môžeme pozorovať generovaný sínusový signál. Vzorkovacia frekvencia F_s kodeku je fixne 8kHz. Generovaná frekvencia je

$f=F_s/(\text{počet bodov tabuľky}) = 8\text{kHz}/8=1\text{kHz}$

Amplitúda signálu je približne $0{,}85V_{pp}$.

Sledovanie premenných

To, či procesor vykonáva program, si môžeme jednoducho overiť v ľavom okne stavového riadku. Ak potrebujeme sledovať hodnotu premenných, prípadne meniť hodnoty parametrov, zvolíme View→ Quick Watch window. V dolnej časti CCS sa objaví okno Watch, ktoré môžeme zadokovať vedľa okna správ. Do editačného riadku vpíšeme názov premennej „amplitude“ a klikneme na „Add to Watch“. Hodnota tejto premennej je 10 (viď zdrojový kód). V okne Watch môžeme teraz túto hodnotu zmeniť napr. na 30, čo má za následok zväčšenie amplitúdy na hodnotu približne $2{,}6V_{pp}$. Vidíme, že je možné meniť parametre programu priamo počas jeho vykonávania. Vyskúšajme zmeniť amplitúdu na hodnotu 33. Uvidíme, že amplitúda aj výška tónu sa výrazne zmenila. Dôvodom je pretečenie kapacity kodeku AIC23. Pretože funkčné hodnoty v tabuľke sú násobené hodnotou premennej „amplitúda“, je rozsah hodnôt privádzaných na DA prevod ±33000. Keďže kodek je 16 bitový a pracuje s číslami v dvojkovom doplnku, môže spracovávať čísla v rozsahu od $<-2^{15}; 2^{15}-1>$.

Vytvorenie súboru typu GEL

CCS ponúka zaujímavý nástroj vo forme interpretačného jazyka podobného jazyku C pod názvom „General Extension Language (GEL)“. Umožňuje napr. vizualizáciu premenných a ich zmenu počas vykonávania programu pomocou štandardných prvkov OS Windows. Princíp a použitie jazyka GEL si ukážeme na jednoduchom príklade. V predchádzajúcom sme vytvorili sinusový generátor, ktorého amplitúdu môžeme nastavovať zmenou príslušnej premennej v okne Watch window. Teraz vytvoríme (napr. v textovom editore CCS príkazom File → New) nový súbor a zapíšeme doň krátky kód v jazyku GEL:

```
menuitem “Sine Amplitude”

slider Amplitude(10,35,5,1,amplitudeparameter) /*start na 10, stop na 35*/
{
        gain = amplitudeparameter;        /*vracia nastavenu hodnotu*/
}
```

Súbor uložíme pod názvom amplitude.gel. Ak už súbor bol vytvorený, je možné ho k projektu pripojiť príkazom Select File → Load GEL. Uložíme program do pamäte procesora a spustíme ho, potom zvolíme GEL→ Sine Amplitude → Amplitude. CCS vytvorí okno s jazdcom, pomocou ktorého môžeme nastavovať hodnotu premennej „amplitude“. Kód gel súboru, ktorý sme vytvorili, je natoľko jednoduchý, že nepotrebuje komentár. Spomeňme len parametre pre vytvorenie jazdca: za kľúčovým slovom „slider“ nasleduje identifikátor

a v zátvorke prvé číslo reprezentuje počiatočnú a druhé konečnú hodnotu rozsahu jazdca. Tretie číslo definuje krok pri pohybe jazdca šipkami a posledné číslo reprezentuje minimálny krok pri pohybe myšou. „Amplitudeparameter“ je názov lokálnej premennej.

Úloha č.2: Grafické zobrazenie generovaného signálu

V predchádzajúcich statiach sme sa naučili základnému ovládaniu CCS. V tejto časti bude našou úlohou modifikovať zdrojový kód sine8_buf.c tak, aby sme nemuseli na kontrolu výsledku činnosti procesora používať osciloskop, ale priamo nástroje CCS, ktoré nám to umožňujú. Vytvoríme si projekt s názvom sine8_buf.pjt podľa úlohy 1 s rozdielom súboru "vectors_intr.asm" namiestom "vectors_poll.asm".

Zobrazovanie priebehu signálu v CCS

Aby sme mohli zobraziť generovaný signál priamo v CCS, je potrebné v zdrojovom kóde definovať pole, do ktorého budeme ukladať výstupné vzorky signálu a toto pole musíme pri behu programu korektne napĺňať. Ak už máme program upravený, odladený a spustený, zvolíme View→ Graph → Time/Frequency. Zobrazí sa dialóg, v ktorom musíme nastaviť niekoľko parametrov. Prvým je „Display Type“, ktorým nastavujeme typ grafu: „Single Time“ je graf zobrazujúci vzorky v časovej doméne, „FFT Magnitude“ je graf zobrazujúci spektrálnu charakteristiku signálu získanú pomocou rýchlej Fourieovej transformácie (FFT). V položke „Graph Title“ uvedieme názov grafu. Nasleduje dôležitý parameter, „Start Address“, v ktorom definujeme štartovaciu adresu prírastkového buffra, ktorý obsahuje dáta pre zobrazenie. Namiesto fyzickej adresy môžeme zadávať názov buffra, pretože ten je v pamäti reprezentovaný ako pole začínajúce na danom návestí. Nasleduje parameter „Acquisition Buffer Size“, v ktorom zadávame veľkosť buffra. Parameter „Index Increment“ umožňuje nastaviť krok, s ktorým budú zobrazované vzorky z prírastkového buffra. Parametrom „Display Data Size“ určujeme veľkosť zobrazovacieho buffra, ktorého obsah bude zobrazovaný na monitore. Obsah zobrazovacieho buffra je pravidelne obnovovaný z prírastkového buffra. Parameter „DSP Data Type“ umožňuje nastaviť formát zobrazovaných dát. Parametrom „Q-Value“ definujeme počet miest za rádovou čiarou. Parameter „Sampling Rate (Hz)“ umožňuje nastaviť vzorkovaciu frekvenciu pre prírastkový buffer podobne ako pre AD prevodník. Význam ostatných parametrov je zrejmý z ich názvu. V prípade nejasností nájdete podrobnejšie informácie v nápovede k CCS.
Vytvorte grafické zobrazenie časového priebehu generovaného signálu a zobrazte jeho spektrálnu charakteristiku. Modifikovaný zdrojový súbor a obidva grafy priložte k elaborátu.

Úloha č.3: Násobenie prvkov dvoch polí a akumulácia výsledkov

Proces násobenia prvkov dvoch polí spojený s akumuláciou výsledkov patrí k častým úlohám riešeným pri digitálnom spracovaní signálov ako sú digitálna filtrácia, korelácia signálov, spektrálna analýza atď. Z tohto dôvodu vyvinuli výrobcovia DSP procesorov hardvérové násobičky, ktoré umožňujú násobenie čísel v jednom strojovom cykle. Procesory rodiny C6x navyše umožňujú násobiť a akumulovať dve dvojice čísel v jednom cykle. V rámci tejto úlohy napíšeme program, ktorý bude násobiť hodnoty dvoch polí podľa zadania a parciálne výsledky akumulovať. Pri jeho testovaní sa naučíme využívať krokovanie programu.

Návod:

Najprv si musíme vytvoriť v C:\ti\myprojects adresár (napr. Multiplication) a v CCS nový projekt. V textovom editore napíšeme definíciu polí v tvare #define field_name x,y,z,...
Súbor uložíme ako *.h. V CCS vytvoríme nový zdrojový súbor *.c. Nesmieme zabudnúť vložiť doň náš hlavičkový súbor s poľami známou direktívou #include "*.h" a tiež štandardný hlavičkový súbor pre IO operácie #include <stdio.h>, aby sme mohli použiť funkciu printf().
Na začiatku súboru musíme uviesť prototypy používaných funkcií, napr.:
int function_name(short *a, int *b, ...);
Konštanty definujeme direktívou #define constant_name value.
Deklarácia premenných a polí je štandardná, napr.: short data[4] = {data_array};

Keďže v tejto úlohe nepoužívame spracovanie v reálnom čase nebudeme musieť do projektu pripájať niektoré súbory, ktoré boli v predchádzajúcich úlohách nevyhnutné. Okrem nášho zdrojového súboru a hlavičkového súboru s definíciou polí musíme k projektu pripojiť príkazový súbor linkera c6xdsk.cmd, knižnicu rts6701.lib a definíciu vektorov prerušení vectors.asm. Súbor vectors.asm vytvoríme modifikáciou súboru vectors_poll.asm.
Zdrojový súbor k tejto úlohe a modifikovaný súbor vectors.asm priložte k elaborátu.