УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Departman za računarstvo i automatiku

Smer računarstvo i automatika

PROJEKTNI ZADATAK

Student: **Milica Vojnović RA 59-2020**

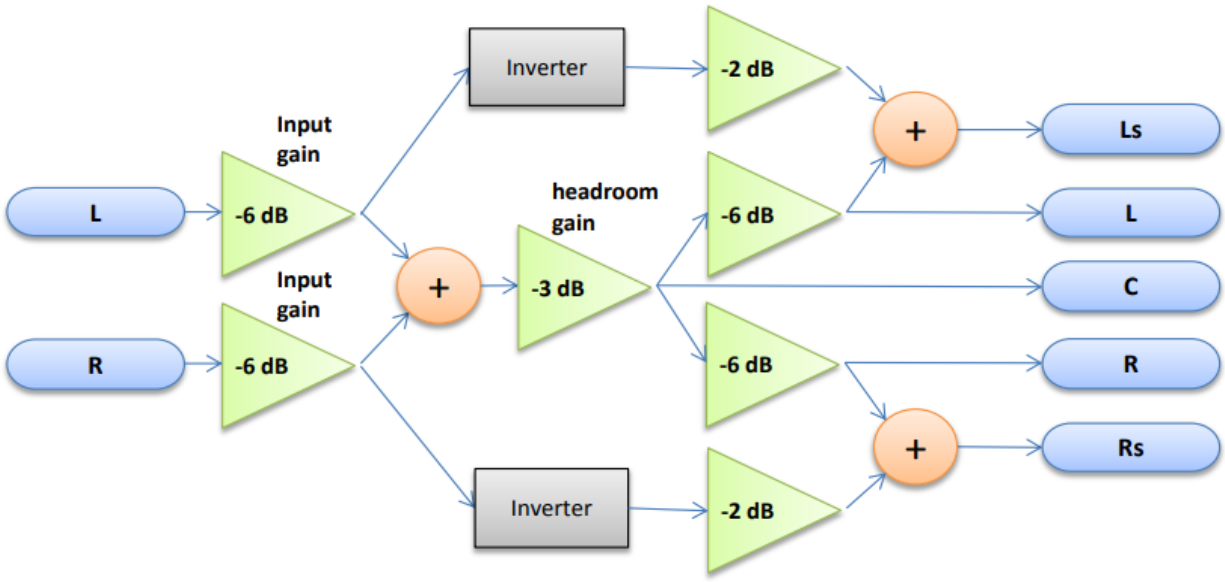
Predmet: **Algoritmi i arhitekture DSP**

Tema rada: **Realizacija algoritma kombinovanja kanala na Cirrus Logic DSP platformi**

Novi Sad, novembar 2023.

Projektni zadatak

Pred nama je projektni zadatak koji implementira sva znanja stečena tokom predmeta Algoritmi i arhitekture DSP-a. Cilj je da realizujemo kombinovanje kanala na osnovu **date šeme i tebele.** Za implementaciju koristimo razvojna okruženja Visual Studio i CLIDE. Koristimo C kod, koji potom prilagođavamo aritmetici Crystal DSP procesora, i na kraju se služimo asemblerom. Rešenje realizujemo postepeno, kroz **modele**. Počinjemo sa referentnim C kodom u aritmetici pokretnog zareza.



control	Enable	Input gain	Headroom gain	Output Mode
values	On/Off	From 0 to -∞ dB	From 0 to -∞ dB	2_0_0, 0_2_0, 3_2_0
default value	On	-6 dB	-3 dB	2_0_0

Tabela 1 – Korisničke kontrole

Model 0

Model 0 je zapravo **referentni kod** za dalji rad. Razvija se isključivo kako bi procesing algoritam proradio (univerzalno rešenje bez obzira na konkretnu DSP platformu). Realizuje se u Visual Studio-u, sa x86 kompajlerom, u C/C++ kodu.

Fokus u ovom koraku je na dve stvari, prva od njih je implementacija unosa željenih parametara i izrada njihove funkcionalnosti (tabela korisničkih kontrola). *Enable* određuje da li će ulaz biti obrađen, *Input gain* i *Headroom gain* služe za menjanje vrednosti signala. *Output mode* nam govori koje kanale treba da formiramo, prosleđuje se u formatu X_Y_Z (X – broj L, R i C kanala; Y – broj Ls i Rs kanala; Z broj LFE/SUB kanala).

Druga stvar je funkcija koja vrši obradu ulaznih signala:

```
void gainProcessing(double pIn[][BLOCK_SIZE], double pOut[][BLOCK_SIZE], const double inGain, const double headGain, const int nSamples, const char* OutputMode)
```

Funkcija kao parametre prima blok ulaznih signala, prostor u koji smeštamo izlazne signale, Input gain, Headroom gain, broj uzoraka i Output mode. Kada podelimo šemu po koracima, obrada se sastoji od aritmetičkih operacija, sabirača dva signala i invertera. Prilikom rada sa vrednostima moramo da uočimo da li može doći do prekoračenja. Zbog toga tokom npr. sabiranja dve vrednosti moramo da odradimo i saturaciju signala.

1. Input gain na L i R signale (vrednosti od 0 do -∞)
2. Sabiranje L i R i inicijalizacija C kanala
3. Headroom gain na konačni C signal (vrednosti od 0 do -∞)
4. Inverter koji uzima L i R i inicijalizacija Ls i Rs kanala
5. Konstantna vrednost (-6 dB) na konačne L i R signale
6. Konstantna vrednost (-2 dB) na Ls i Rs signale
7. Sabiranje L sa Ls i R sa Rs kako bi dobili konačne Ls i Rs signale

Model 1

Model 1 je korak u kojem **funkcionalno optimizujemo** referentni C kod iz modela 0. To uključuje organizaciju podataka, poput smanjenja broja argumenata kod funkcija i uklanjanja lokalnih struktura i promenjivih, i izbacivanje nezavisnih delova koda iz petlji. Tim pristupom smo izbacili prosleđivanje parametara, kojima možemo da pristupamo kao globalnim promenljivima, i sveli poziv funkcije procesiranja na:

```
void gainProcessing(double pIn[][BLOCK_SIZE], double pOut[][BLOCK_SIZE])
```

Najosetljiviji deo ovog koraka je menjanje pristupa podacima. CS497xx ne podržava indeksiranje u C-ovskom kontekstu, koje zauzima puno instrukcija, zbog čega moramo da koristimo pokazivače i duple pokazivače. Primer ovoga bi bio:

```
// 1. Model 0
pOut[LEFT_CH][j] = pIn[LEFT_CH][j] * LocalInputGain;
// 1. Model 1
double* InLSamplePtr = *(pIn + LEFT_CH);
double* OutLSamplePtr = *(pOut + LEFT_CH);
(*OutLSamplePtr) = (*InLSamplePtr) * GlobalInputGain;
```

Za validaciju izlaza iz svih modela koristimo alat **PCMCompare**, a kao validan i željeni izlaz posmatramo dobijene izlaze iz modela 0. Očekivana razlika između modela 0 i modela 1 bi trebala da bude 0 bita, odnosno izlazi bi trebali da su identični.

```
PS C:\Users\Lenovo\Desktop\AADSP\tools> ./PCMCompare -b16
project/Model0/speech_out_02.wav C:/Users/Lenovo/Desktop/
_02.wav
      No differences encountered!
      No differences encountered!
PS C:\Users\Lenovo\Desktop\AADSP\tools>
```

Model 2

Cilj modela 2 je da se algoritmi i C kod **modifikuju za tipove podataka sa nepokretnim zarezom**, dakle, prelazimo iz floating-point u fixed-point aritmetiku. Uvodimo fixed-point emulacione klase, koje su prilagođene DSP aritmetici, kao i prilagođene tipove promenljivih. Fract predstavlja data registre i memoriju sa s.31 veličinom, a Long_accum predstavlja akumulacione registre sa s8.63 veličinom. Primer izmena:

```
// Model 1
double* InLSamplePtr = *(pIn + LEFT_CH);
(*InLSamplePtr) *= GlobalInputGain;

// Model 2
DSPfract* InLSamplePtr = *(pIn + LEFT_CH);
(*InLSamplePtr) = (DSPfract)(*InLSamplePtr) * GlobalInputGain;
```

Razlika između modela 2 i modela 0 ili 1 bi trebala da bude u opsegu od 0 do 2 bita. Nakon formiranja ova tri modela, pokreće se skripta koja proverava njihove izlaze. Kao ulazne vrednosti uzima sve vrednosti koje neki flag može da ima (enable, mute, output mode) ili granične i ubičajne vrednosti za neki parametar koji ima širok opseg vrednosti (gain).

```
PS C:\Users\Lenovo\Desktop\AADSP\tools> ./PCMCompare -b16 C:/Use
ers/Lenovo/Desktop/project/Model2/speech_out_02.wav
    Max difference is 1 (1 bits, -96.33dB)
    Max difference is 1 (1 bits, -96.33dB)
    2965536 samples compared
   ------------------------------------------------------------
    Dif(bits) |    Samples    |   PERCENT   |   First dif
   ------------------------------------------------------------
         1    |         36    |    0.00%    |  0x00032674
   ------------------------------------------------------------
      Error   |         36    |    0.00%
PS C:\Users\Lenovo\Desktop\AADSP\tools>
```

Model 3 – C kod

Iz Visual Studio okruženja prelazimo u CLIDE. Kod treba da se prilagodi iz C++ u C jezik, zbog čega unosimo sitne izmene. Nakon pokretanja, razlika između izlaza modela 2 i modela 3 dobijenim C kodom bi trebala da bude 0. Ovo je osnova iz koje dalje vršimo **izmene za željenu arhitekturu**.

Model 3 – assembler

Sledeći korak je da se funkcija `gainProcessing` prebaci iz C koda u ručno pisani asemblerski kod date platforme. Svrha toga je da obrada signala bude optimizovana i da vreme izvršavanja bude kraće. Razlika ovog modela sa drugima je prihvatljiva u opsegu od 0 do 2 bita. Primer koraka u C jeziku i u asemblerskom jeziku:

```
// Model 3 – C kod
(*OutLsSamplePtr) = (DSPfract)(*OutLsSamplePtr) * MINUS_2DB;
```

```
// Model 3 – asemblerski kod
x0 = ymem[i2]
y0 = xmem[_Minus_2DB]
a0 = x0 * y0
ymem[i2] = a2
```

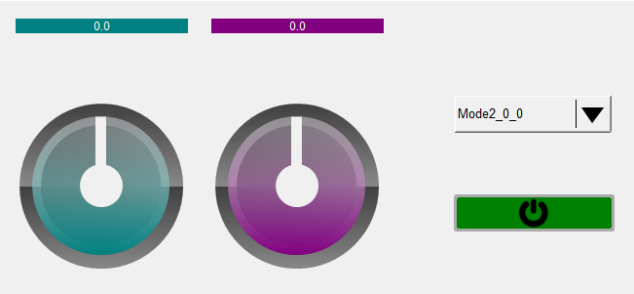
Model 3 – optimizacija

Iako asemblerski kod već troši mnogo manji broj ciklusa od originalnog, postoje metode kako da se platforma i njena struktura maksimalno iskoriste, kako bi se uštedelo još ciklusa. Osnovna ideja je paralelizovanje delova koda koji jedan sa drugim nemaju mimoilaženja. Pored toga, možemo optimizovati MAC instrukcije, premestiti invarijantne delove koda van petlje itd. Ovako optimizovan kod se koristi u finalnom modelu, gde se obrada vrši in real time. U nastavku sledi tabela sa brojevima ciklusa `gainProcessing` funkcije za sve modele 3.

Model 3 – C kod	4014
Model 3 – assembler	1787
Model 3 – optimizacija	1149

Finalni model

Finalni model se svodi na **integraciju u okreženje, dalju optimizaciju i verifikaciju**. Formira se projekat u CLIDE okruženju, zajedno sa *example_app, example_ovly, example_module* fajlovima, kao i našim asemblerskim rešenjem funkcije za obradu signala. Umesto unošenja argumenata preko komandne linije, formiramo elemente koji u realnom vremenu primaju vrednosti i koriste se pri obradi signala.



control	Enable	Input gain	Headroom gain	Output Mode
values	On/Off	From 0 to -∞ dB	From 0 to -∞ dB	2_0_0, 0_2_0, 3_2_0
default value	On	-6 dB	-3 dB	2_0_0

Testiranje

Veliki deo projekta se svodi na validaciju izlaza u odnosu na prethodne modele. Za model 0, pomoću debaga možemo korak po korak da ispratimo menjanje vrednosti i računanjem da utvrdimo da li se izlaz poklapa sa šemom. Takođe, pomoću alata **Audacity** možemo da pogledamo izgled signala i njegove karakteristike. Verifikacija modela 1 i 2 je automatizovana pomoću **run_test.py** skripte, koja u svojoj osnovi prosleđuje sve kombinacije ulaza i testira izlaze pomoću **PCMCompare** datoteke. Modeli 3 su dosta spoji, i njihovo testiranje je ipak malo zahtevnije, kao i za finalni model. Glavne metode testiranja su korišćenje Audacity-ja, ručno pokretanje PCMCompare, kao i praćenje vrednosti u debug modu. Korišćeni test vektori: Tone_L1k_R3k.wav, funky_sample.wav, speech_2ch.wav.

Zaključak

Uspešno smo implementirali obradu signala koja je zadata na početku. Od modela 0 do finalnog modela, prešli smo iz C++ u asemblerski jezik, iz aritmetike sa pokretnim zarezom u aritmetiku sa nepokretnim zarezom, iz Visual Studio u CLIDE okruženje, iz fiksne obrade u obradu u realnom vremenu. Svi modeli zadovoljavaju uslov da imaju razliku u opsegu od 0 do 2 bita od početnog modela.