SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1239

DIGITALNI SUSTAV ZA FRAKCIONALNO KAŠNJENJE SIGNALA

Lovro Damiš

Zagreb, lipanj 2010.

Zahvaljujem mentoru prof.dr.sc. Davoru Petrinoviću na pomoći tijekom izrade ovoga rada.

Zahvaljujem dipl.ing. Ivanu Dokmaniću na svim savjetima i tehničkoj podršci bez koje ne bi bilo vidljivih rezultata.

Sadržaj

1. Uvod

U završnom radu implementiran je sustav za frakcionalno kašnjenje signala na Blackfin BF537 procesoru.

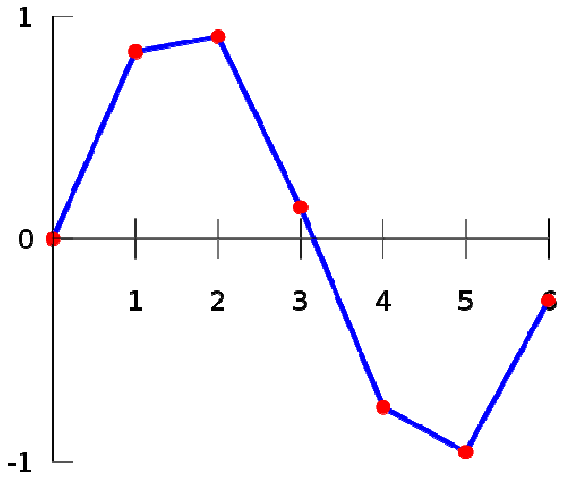
Sklopovi za kašnjenje signala imaju ključnu ulogu u sustavima gdje je bitna sinkronost signala. Primjerice, u digitalnoj televiziji slika kasni za zvukom zbog nejednakog vremena obrade signala, budući da se zvuk obrađuje brže od slike. Stoga je poželjno zakasniti zvuk za vrijeme koje je potrebno za obradu slika. Kašnjenje se može ostvariti RC linijom ili digitalnom obradom signala koja je predmet ovog završnog rada.

Da bi mogli obrađivati analogne signale DSP (*Digital Signal Processing*) procesorom, prvo ih je potrebno digitalizirati. U tu svrhu koristimo A/D pretvornik, kojim iz analognog signala uzimamo samo uzorke s razmacima definiranim frekvencijom uzorkovanja. Pri tom javlja se problem kada želimo zakasniti signal za neki realni broj nasuprot cjelobrojnim mogućnostima. Za razrješenje tog problema uvodimo pojam frakcija, a vrijednosti signala između dva uzorkovana podatka nalazimo upotrebom interpolacije.

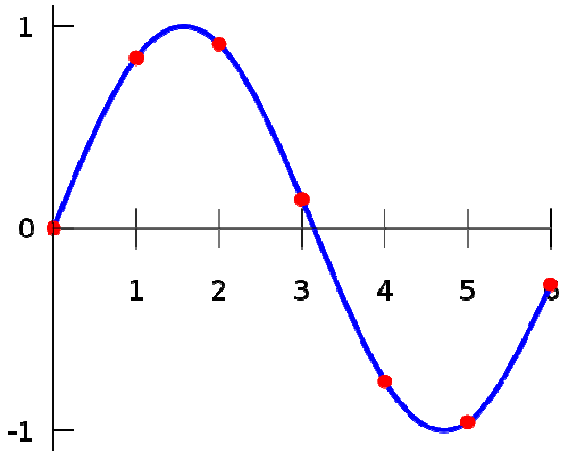
2. Interpolacija

Interpolacija je metoda koja nam omogućava da konstruiramo funkciju signala kroz uzorkovane točke. Postoji mnoštvo takvih metoda od kojih je najjednostavnija traženje aritmetičke sredine između dva uzorka na kojoj se bazira linearna interpolacija. Za neku kvalitetniju obradu signala, ista nije prihvatljiva zbog velikog izobličenja (Slika 2-1). S druge strane možemo koristiti polinomnu interpolaciju koja daje izvrsne rezultate, ali za svoj rad koristi mnogo točaka da bi se konstruirala funkcija. Nalazimo da je *spline* interpolacija daleko efikasnija za implementaciju u DSP procesore.

Spline interpolacija koristi polinome manjeg stupnja u svakom intervalu i odabire polinomne dijelove tako da skupa glatko pristaju. Tako dobivena funkcija i originalna funkcija poklapaju se u prvoj i drugoj derivaciji uzorkovanih točaka (Slika 2-2).



Slika 2-1 Funkcija dobivena linearnom interpolacijom



Slika 2-2 Funkcija dobivena spline interpolacijom

Na Blackfin procesoru BF537 slijedno će se uzimati uzorci sa A/D pretvornika na temelju kojih će se izračunavati koeficijenti za kubičnu B-spline interpolaciju upotrebom digitalnog filtra prema (2). Koeficijenti će se kasniti za željeni broj koraka, te će D/A pretvornik primati vrijednosti interpolirane funkcije prema izrazu (1) gdje je *n* trenutni uzorak, a *dx* frakcija između trenutnog i slijedećeg uzorka.

$$y_i[n] = \big((D[n] * dx + C[n]) * dx + B[n]\big) * dx + A[n] \quad (1)$$

Koeficijente *A*, *B*, *C* i *D* dobivamo prema izrazu (2)

$$[D(z) \quad C(z) \quad B(z) \quad A(z)]^T = T(z) * \begin{bmatrix} W(z) \\ Y(z) \end{bmatrix} \quad (2)$$

gdje je T(z) opisan matricom (3).

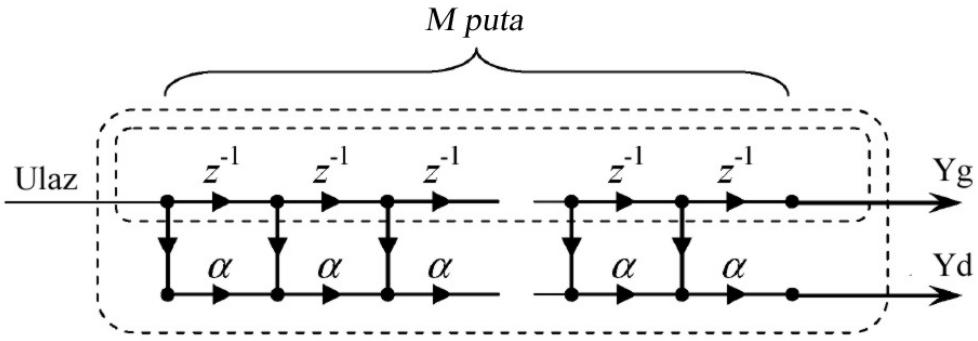
$$T(z) = \begin{bmatrix} 3 * (z^2 + z - 1 - z^{-1}) & -2(z-1) \\ -3 * (z^2 + 2z - 1 - 2z^{-1}) & 3(z-1) \\ 3(z - z^{-1}) & 0 \\ 0 & 1 \end{bmatrix} \quad (3)$$

Iz gore navedenog, uočavamo da bi za funkcioniranje ovakvog filtra bilo potrebno poznavati buduće uzorke (potencija z^2, z) što nije moguće u slijednom čitanju podataka. Upotrebljavamo matematički trik, cijelu jednadžbu dijelimo sa najvećom

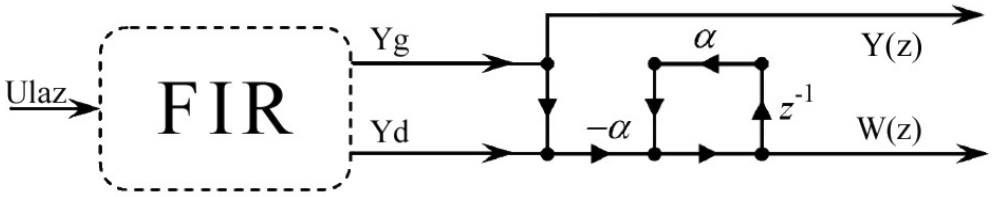
potencijom z^2 , te konačno dobivamo vrijednosti filtra prema izrazu (4) kojem su potrebni samo prethodni uzorci, a to jednostavno ostvarujemo primjenom memorijskog spremnika.

$$z^{-2} * T(z) = \begin{bmatrix} 3 * (1 + z^{-1} - z^{-2} - z^{-3}) & -2(z^{-1} - z^{-2}) \\ 3 * (1 + z^{-1} - z^{-2} - z^{-3}) & 3(z^{-1} - z^{-2}) \\ 3(z^{-1} - z^{-3}) & 0 \\ 0 & z^{-2} \end{bmatrix} \quad (4)$$

Parametre W(z) i Y(z) iz jednadžbe (2) dobivamo iz FIR filtra (Slika 2-3) i kauzalnog B – spline filtra kaskadne realizacije (Slika 2-4).



Slika 2-3 FIR filtar



Slika 2-4 Kauzalni kaskadni B – spline filtar

Klasični FIR filtar ima samo Yd izlaz, no za potrebe kauzalnog B – spline filtra izvodi se i gornja grana Yg koja je zapravo zakašnjeli ulazni signal. Filtri su podešeni prema parametrima (5) i (6) što daje zadovoljavajuće rezultate.

$$M = 2 \quad (5)$$

$$\alpha = \sqrt{3} - 2 \quad (6)$$

3. DSP procesor – Blackfin BF537

Današnje procesore možemo svrstati u dvije velike grupe. Prva grupa služi za manipulaciju podacima, njihovu usporedbu i pohranu u memoriju, a druga za izvođenje složenih matematičkih operacija. I prva i druga grupa procesora može raditi sve, ali ne s istom efikasnošću. Prvu skupinu predstavljaju procesori opće namjene, a drugu DSP (eng. *Digital Signal Processing*) procesori. DSP procesori su upravo oni koji obavljaju složene proračune i to čine u vrlo kratkom vremenu uz minimalnu potrošnju snage te služe kao nadopuna one prve skupine. Najčešće se DSP – ovi koriste kao koprocesorske jedinice procesora opće namjene.

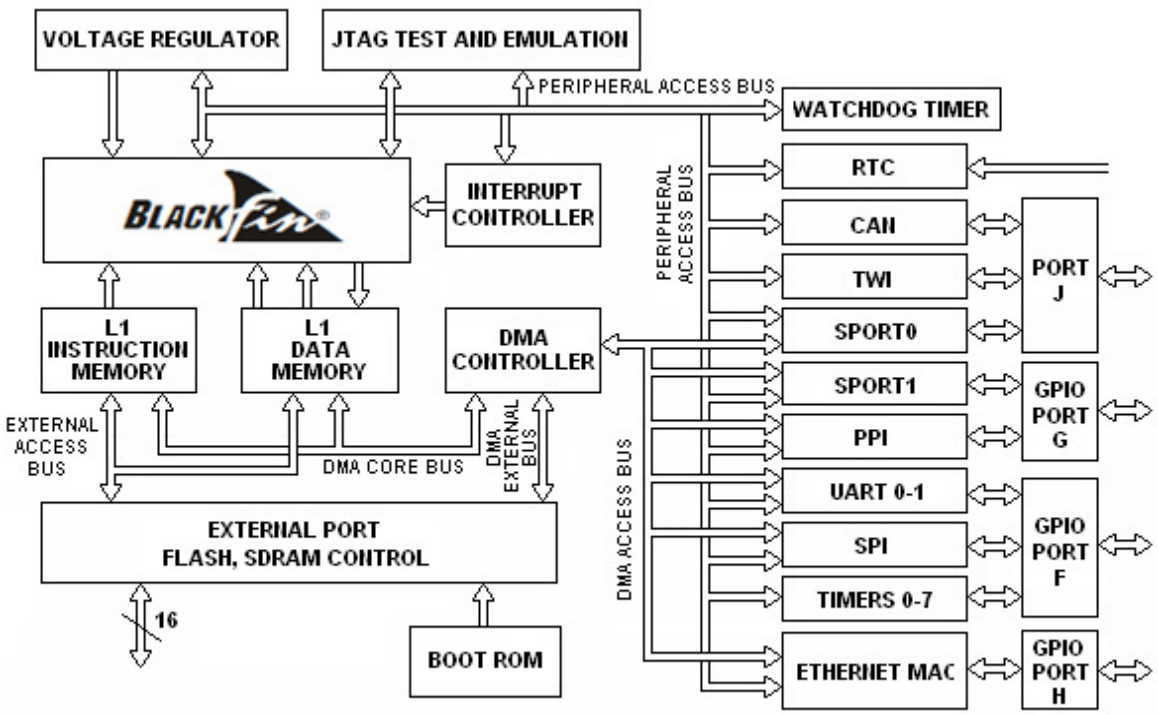
Osnovne značajke DSP procesora su:

- Brza jedinica za množenje i pribrajanje (eng. *multiply and accumulate*)
- Arhitektura sa višestrukim memorijskim pristupom
- Specijalizirani načini adresiranja argumenata
- Specijalizirano sklopovlje za kontrolu programskog toka
- Niz periferijskih međusklopova integriranih na čipu

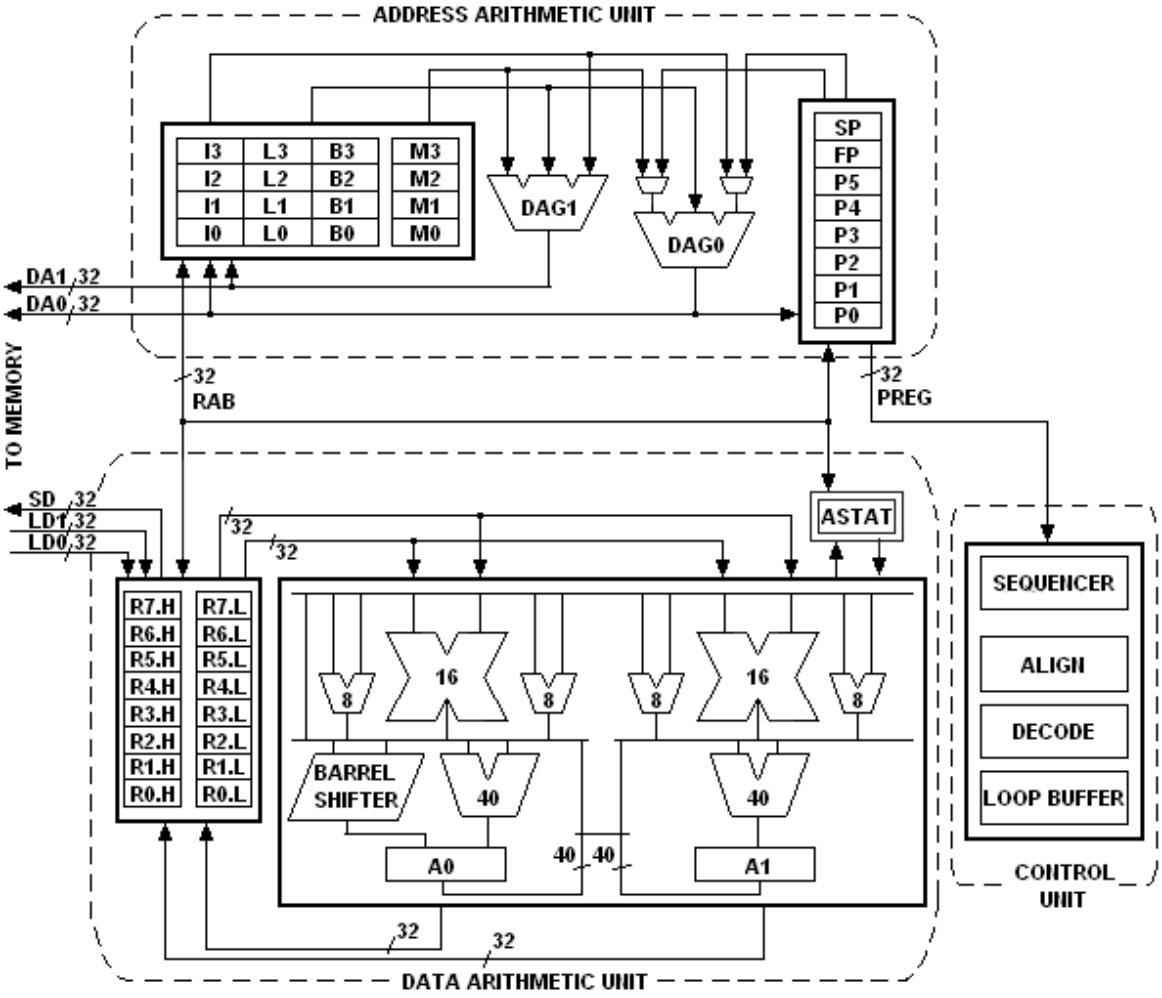
Tvrtka Analog Devices u suradnji sa Intel Corporation razvila je procesor Blackfin BF537 koji je sinteza procesora opće namjene i DSP procesora integriranih u jedinstvenu jezgru.

3.1 Arhitektura Blackfin BF537 procesora

Arhitektura procesora sa jezgrom, memorijom i perifernim sklopovima prikazana je na blok shemi na slici (Slika 3-1), a sama jezgra procesora na slici (Slika 3-2).



Slika 3-1 Blok shema procesora BF537



Slika 3-2 Jezgra procesora BF537

Jezgra procesora sastoji se od dva 16 – bitna množila (MAC), četiri aritmetičko – logičke jedinice (ALU), dva 40 – bitna akumulatora i jedan 40 – bitni posmačni registar (barrel shifter). Podaci stižu u jezgru po dvije 32-bitne sabirnice što je velik napredak u DSP procesorima. Time se omogućava paralelno izvršavanje više operacija množenja i zbrajanja, ovisno o modu rada. Jezgra ne može direktno komunicirati sa memorijom, nego isključivo kroz podatkovni registarski stog. Stog se sastoji od osam 32-bitnih registara koji se opet mogu adresirati po 16-bita širine. Postoji i hijerarhija memorije prema kojoj jezgra prvo komunicira sa L1 cache memorijom (radi na istoj brzini kao jezgra), L1 se prijenosi u L2 cache (malo sporiji

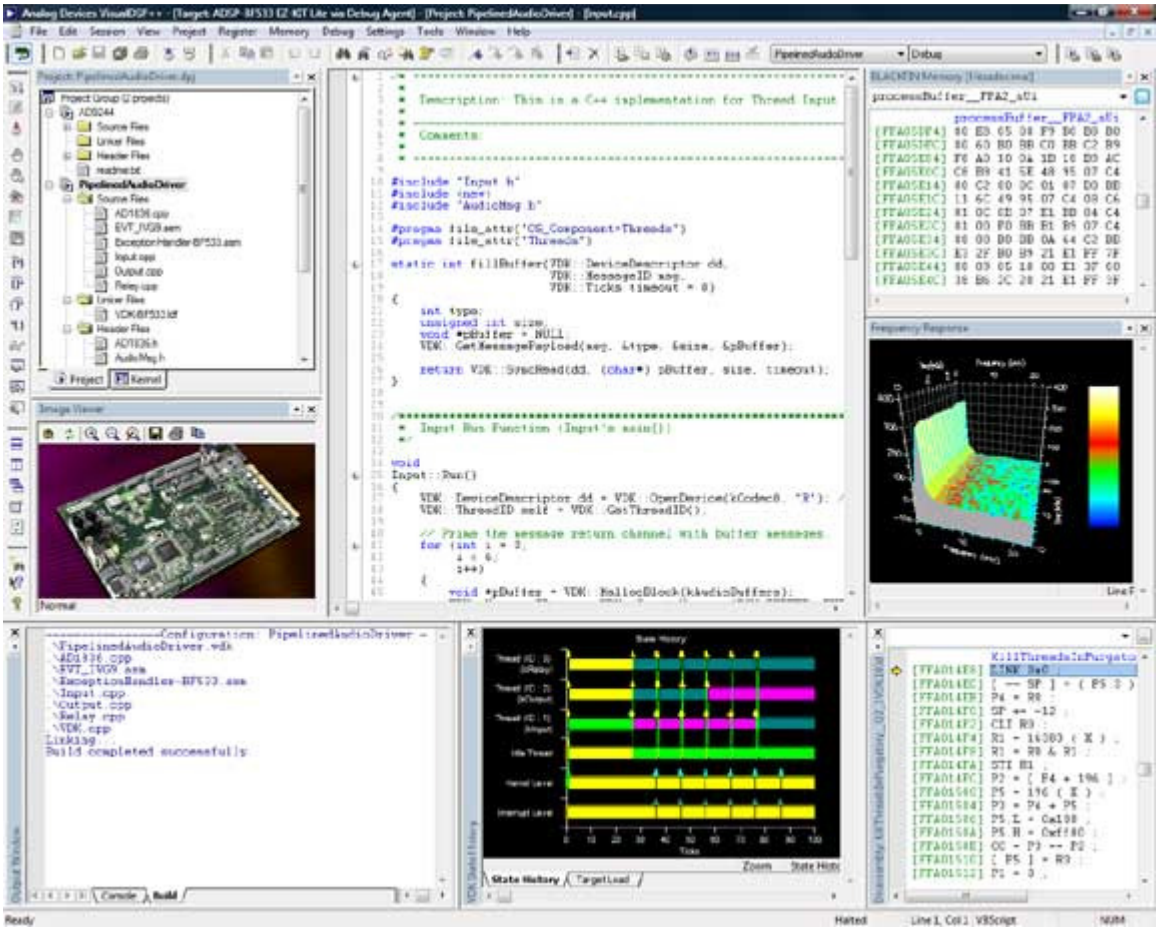
takt) i konačno L2 komunicira sa vanjskom memorijom koja je i najsporija. Koristeći hijerarhijsku memoriju i cjevovod (pipeline) dubine 10, BF537 postiže visoku efikasnost izvođenja instrukcija. Instrukcije k tomu mogu biti 32-bitne ili 16-bitne riječi čime se postiže velika gustoća kôda.

Uz sve navedeno, Blackfin procesor BF537 može raditi na maksimalnom taktu jezgre od 756Mhz čime dobiva titulu procesora visokih procesnih mogućnosti, a sve to moguće je pogoniti sa inteligentno upravljanim naponom napajanja. Najniži napon potreban za obavljanje osnovnih operacija iznosi 0.8V. Takav procesor idealan je za ugradnju u mobilne zvučne i video uređaje.

3.2 Razvojna podrška za Blackfin procesore

Analog Devices kao programska podrška za programiranje svojih procesora nudi programsko okruženje pod imenom VisualDSP++ 5.0 (Slika 3-3). Ono sadrži slijedeće alate:

- Integrirano okružje za razvoj i uklanjanje grešaka (eng. *Integrated Development and Debugging Enviroment*, IDDE)
- Optimizirani prevodilac C/C++ jezika
- Asembler i povezivač (eng. *linker*)
- Simulator
- EZ-KIT Lite sustav za evaluaciju
- Emulator

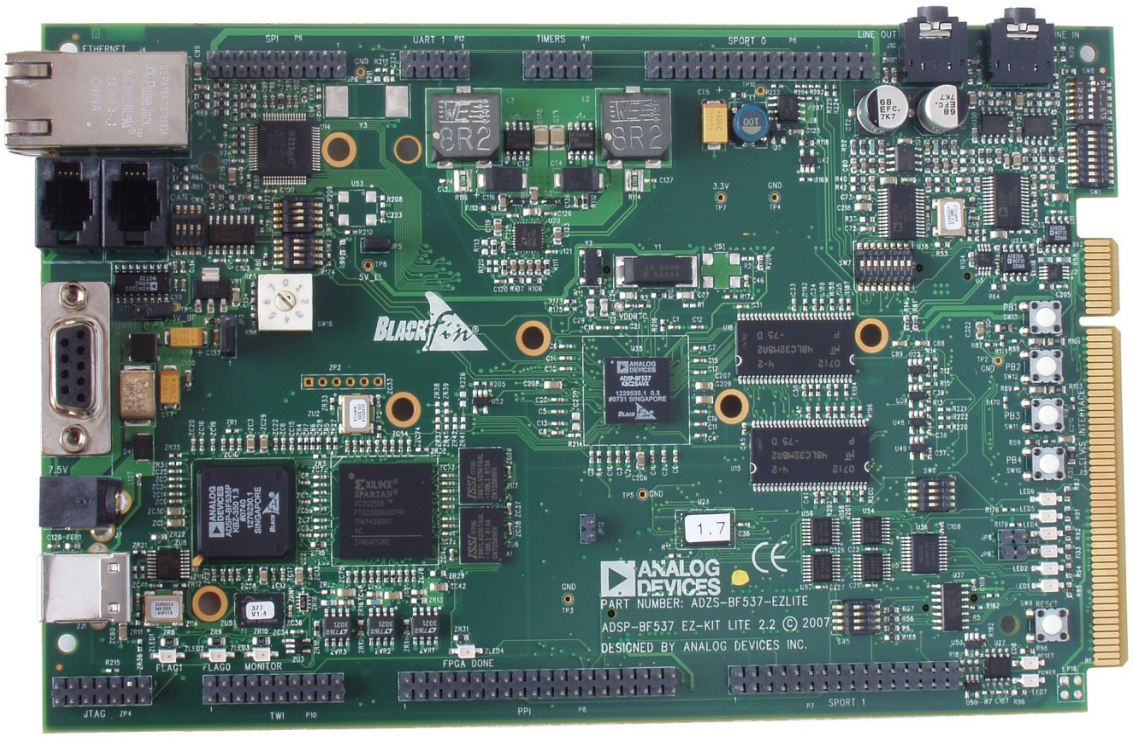


Slika 3-3 Programsko okruženje VisualDSP++

Sklopovska podrška također dolazi od proizvođača procesora pod imenom EZ-KIT Lite za procesor Blackfin BF537. Izgled razvojne ploče vidi se na slici (Slika 3-4). Na ploči se osim procesora nalaze i vanjski periferni sklopovi:

- Procesor BF537 maksimalne frekvencije takta do 600MHz
- 64 MB (32 M x 16) SDRAM-a i 4 MB (2 M x 16) FLASH memorije
- SMSC LAN83C185 10/100 PHY sa RJ45 konektorom
- Odašiljač-prijemnik sa 2 RJ10 konektora
- AD1871 96kHz/48kHz A/D pretvornik
- AD1854 96kHz/48kHz D/A pretvornik
- RS-232 UART

- Sučelje za virtualnu instrumentaciju *National Instruments Educational Laboratory Virtual Instrumentation Suite* (NI ELVIS)
- 10 LED dioda
- 5 tipkala (1 reset i 4 programibilna)



Slika 3-4 Razvojna ploča EZ-KIT Lite za BF537

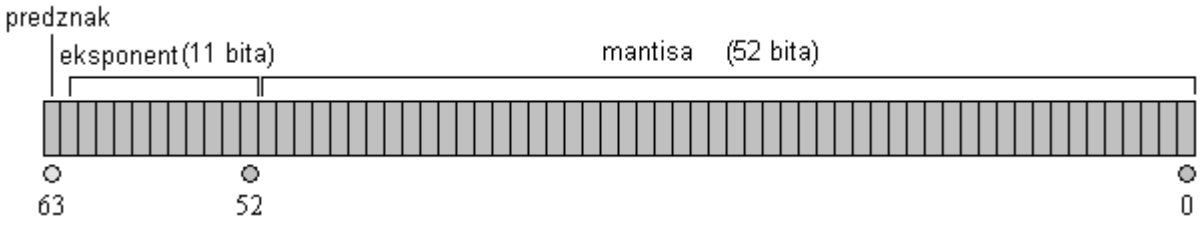
Od navedenih vanjskih periferija za potrebe završnog rada iskorišteni su A/D i D/A pretvornici, led diode te tipkala pomoću kojih se namješta željeni cjelobrojni/frakcionalni pomak signala.

Povezivanje razvojne ploče sa programom VisualDSP++ omogućeno je preko USB protokola.

4. Cjelobrojna aritmetika

Za potrebe obrade signala potrebni su precizni proračuni. Kako je ulazni analogni signal iz realne domene, takav bi trebao biti i izlazni signal. Problem je u tome što ne možemo pohraniti analognu realnu vrijednost već je moramo digitalizirati i pohraniti je kao neki cijeli broj. Postoji nekoliko standarda koji omogućuju zapis decimalnog broja u cjelobrojni registar. Koji god standard koristili uvijek postoji određeni gubitak informacije zbog ograničenosti registra.

Jedan od najraširenijih standarda je onaj sa pomičnom decimalnom točkom – IEEE 754 standard. Bitovi unutar tako određena registra interpretiraju se na slijedeći način (za 32-bitnu preciznost): Prvi bit je predznak (MSB), slijedećih 8-bitova je eksponent, dok je preostalih 23-bita mantisa. Moguće je koristiti dvostruku zapisom broja u 64-bitni registar, gdje se eksponent i mantisa proširuju (Slika 4-1). Ovakav oblik zapisa je intuitivan no zahtjeva skupo sklopovlje i troši mnogo resursa za obradu jedne operacije množenja ili zbrajanja.



Slika 4-1 Format dvostruke preciznosti IEEE 754 standarda

DSP procesori projektirani su kako bi bili što brži u izračunavanju matematičkih operacija. Stoga koriste isključivo cjelobrojnu aritmetiku tj. standard s *nepomičnom decimalnom točkom – frakcionalni brojevi*. Brojevi se zapisuju u dvojnom komplementu i prvi bit (MSB) predstavlja predznak. Potrebno je dogovoriti dinamiku broja - Ako se radi o 16-bitnom broju, potrebno je odrediti koliko će se prvih (MSB) bitova koristiti za cijeli broj, a preostali (LSB) bitovi interpretirat će se kao decimalni dio. Tako primjerice zapis 0.15 definira da će se svih 15 bitova koristiti za zapis decimalne vrijednosti dok je 16. bit predznak. Kada bi imali format 3.12 moguće bi

bilo prikazati cijeli broj u tri (MSB) bita dvojnog komplementa, ali se sada smanjilo mjesto za upis decimalnog djela na 12 bita. Potrebno je uvijek odabrati najbolju moguću dinamiku kako se ne bi bespotrebno gubila preciznost vrijednosti. Općenito, da bi se mogla točno izračunati neka matematička operacija nad takvim zapisom brojeva, potrebno je brojeve potpisati. Moraju se poravnati decimalne točke. Poravnanje se vrši posmakom brojeva različitih dinamika. Ukoliko broj posmičemo udesno dolazi do gubitka informacije na (LSB) bitovima. Takvu pojavu možemo kompenzirati zaokruživanjem najnižeg značajnog bita koji se neće izgubiti nakon posmaka.

Interpretacija brojeva zapisanih s nepomičnom decimalnom točkom u formatu M.N (MSB bit je predznak!) je slijedeća:

- Najveći pozitivni broj: $2^{m} - 2^{-n}$: 0,1,1,1,…,1,1
- Najmanji pozitivni broj: 2^{-n}: 0,0,0,…,0,1
- Najmanji negativni broj: -2^{m}: 1,0,0,0,…,0,0
- Najveći negativni broj: -2^{-n}: 1,1,1,…,1,1

Primjer za format 2.3 u 6-bitnom registru:

- Najveći pozitivni broj: $2^{2} - 2^{-3} = 4 - 0.125 = 3.875$
- Najmanji pozitivni broj: $2^{-n} = 0.125$
- Najmanji negativni broj: $-2^{2} = -4$
- Najveći negativni broj: $-2^{-n} = -0.125$

Primjer pretvaranja decimalnog broja *x* u frakcionalni *F* (6-bitni registar, format 2.3) i obrnuto uz izračun pogreške:

1. $x = 3.14$
2. $F = round(x * 2^m) = round(3.14 * 2^3) = round(25.12) = 25$
3. $F = 25_{10} = 011.001_2$
4. $x' = \frac{F}{2^3} = \frac{25}{8} = 3.125$
5. $p = \frac{x'-x}{x} * 100\% = \frac{0.015}{3.14} * 100\% = 0.48\%$ Relativna ukupna pogreška
6. $p = \frac{x'-x}{x} * 100\% = \frac{0.015}{0.14} * 100\% = 10.71\%$ Relativna pogreška frakcije

5. Referentni kôd

Prije nego započnemo razvijati kôd za direktnu implementaciju u Blackfin BF537 procesor potrebno je krenuti od referntnog kôda te kroz par faza uhodavanja razviti i konačnu implementaciju u programu VisualDSP++.

Referentni kôd razvijen je u programu Matlab koristeći IEEE 754 standard dvostruke preciznosti. Dakle radi se o varijablama tipa *double.* Kôd obuhvaća generiranje ulaznog signala, definiciju konstanti, filtriranje signala B-spline filtrom, interpolaciju izlaznog signala i usporedbu sa ulaznim. Pri tom je korišten samo frakcionalni pomak dok cjelobrojni nije (ugrađuje se kasnije).

5.1 Generiranje referentnog ulaznog signala

Referentni ulazni signal generira se jednom i binarno pohranjuje u datoteku koja se kasnije koristi kao izvor ulaznog signala. Izveden je kao niz od 100 uzoraka kosinusnog signala amplitude od -1 do 1.

Ključni dijelovi kôda:

- `x=cos(2*pi*f*t);` `%Generiranje funkcije u 100 uzoraka`
- `fwrite(fp,x,'double');` `%Binarni upis u datoteku`
- `[X,n]=fread(fp,inf,'double');` `%Binarno čitanje iz datoteke`

Cijeli kôd nalazi se u *Prilogu A: GenCosData.m.*

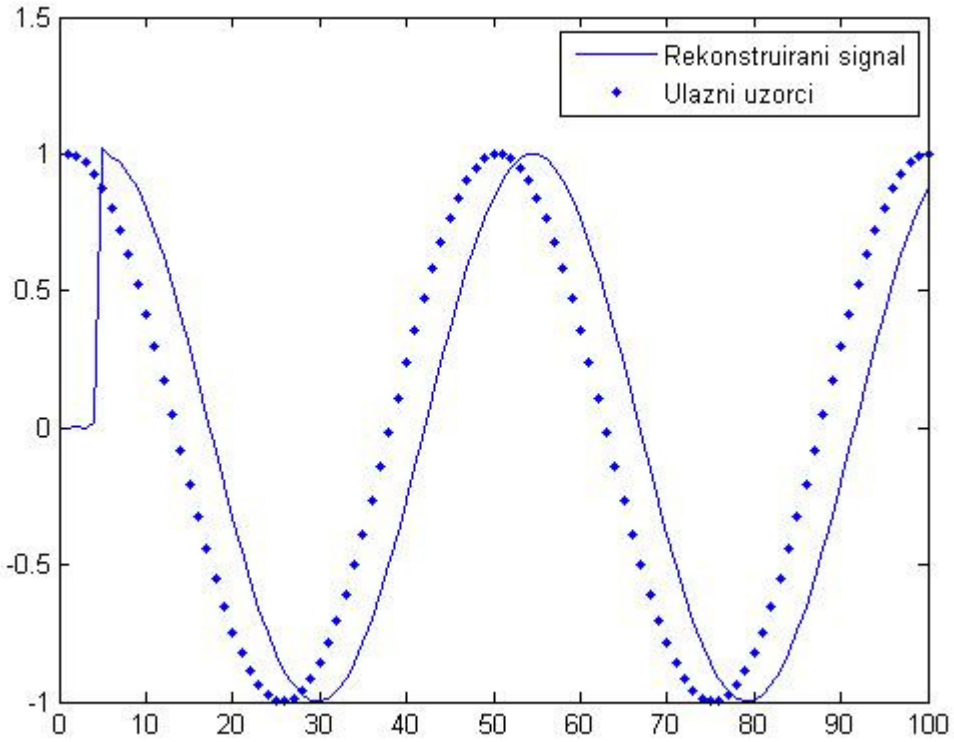
5.2 Izračun referentnih parametara

U datoteci Primjer.m nalazi se referentni kôd koji učitava ulazni signal iz generirane datoteke (CosData.bin) te ga provlači kroz filtar shematski prikazan na slikama (Slika 2-3) i (Slika 2-4). U konačnici dobivamo parametre A, B, C i D potrebne za interpolaciju te crtamo ulazni i interpolirani signal (Slika 5-1).

Ključni dijelovi kôda:

```
• al=sqrt(3)-2;                          %Konstanta α
• M=2;                                   %Red FIR filtra
• co=al.^[M:-1:1];                       %Generiranje koeficijenata FIR filtra
• yg=[zeros(1,M) x(1:end-M)];            %Yg je zakašnjeli signal
• yd=filter(co,1,x);                     %Izlaz iz FIR filtra
• ul=(yg+yd)*(-al);                      %Ulazni signal u B-spline filtar
• wcas=filter(1,[1 -al],ul);             %Izlaz iz B-spline filtra
• dcw=3*[1 1 -1 -1];                     %Komponenta Dw
• dcy=-2*[0 1 -1 0];                     %Komponenta Dy
• d=filter(dcw,1,w)+filter(dcy,1,yg);    %Izračun parametra D
• dx=0.0314;                             %Frakcionalno kašnjenje
• Yout=((d(i)*dx+c(i))*dx+b(i))*dx+a(i); %Rekonstrukcija signala
• plot(i,Yout,i,x,'.');                  %Crtanje signala
• fwrite(fp_D,d,'double');               %Upis parametra D u datoteku
• fwrite(fp_Y,Yout,'double');            %Upis rekonstruiranog signala u datoteku
```

U gore navedenom kôdu navedeni su samo ključni dijelovi referentnog programa. Tako je primjerice naveden samo izračun parametra D dok se preostala tri analogno računaju. Cijeli kôd nalazi se u *Prilogu B: Primjer.m.*



Slika 5-1 Interpolacija ulaznih uzoraka

Sa slike možemo primijetiti da rekonstruirani izlazni signal kasni za ulaznim uzorcima četiri takta. To je očekivana pojava zbog utitravanja B-spline interpolatora koji koristi memorijski spremnik od četiri uzorka koji su inicijalno postavljeni na nulu.

6. Implementacija kôda u ANSI C

Slijedeća faza razvoja digitalnog sustava za frakcionalno kašnjenje jest implementacija referentnog koda iz Matlaba u standardizirani (ANSI) C jezik. Kao programsko okruženje korišten je DevC++. Cilj je prenijeti kôd iz datoteke Primjer.m u C uz razvoj potrebnih funkcija. Također su generirani parametri A, B, C i D te se

sistematizirano uspoređuju s koeficijentima dobivenim referentnim kodom. Usporedba se vrši čitanjem referentnih parametara iz datoteka.

Simuliran je dohvat ulaznih podataka tako da se slijedno čitaju iz datoteke CosData.bin u kojoj je pohranjen referentni ulazni signal. U konačnoj implementaciji na procesoru ulazni podaci će slijedno stizati sa A/D pretvornika frekvencijom uzorkovanja. U ovom slučaju nakon 100 pročitanih uzoraka program staje te vrši usporedbu podataka sa referentnim podacima.

6.1 Implementirane funkcije

*double *GenCo(int M, const double al);*	Generira koeficijente FIR filtra
*void Push(double *p,int M,double x);*	Cirkularno stavlja podatak na stog
*int BrojPodataka(FILE *f);*	Vraća broj podataka u datoteci
*double MaxErr(FILE *f, double P[], int N);*	Vraća maksimalnu razliku dobivenog i referentnog podatka

Detaljniji opis funkcija nalazi se u *Prilogu C: ANSI_C_funkcije.*

6.2 Glavni program

U glavnom programu (*main*) inicijalizirani su koeficijenti filtara, otvorena je datoteka sa ulaznim podacima, alocirana je i inicijalizirana memorija koja simulira cirkularne memorijske spremnike te je simulirano slijedno dohvaćanje ulaznih podataka i obrada nad istim. Rezultati se vide na slici (Slika 6-1).

Ključni dijelovi kôda:

- fread(&x,sizeof(double),1,Cos); //Čitaj podatak
- yg=buff[M-1]; //Yg signal
- Push(buff,M,x); //Stavi podatak u spremnik
- for(i=0,yd=0;i<M;i++) yd+=co[i]*buff[i]; //Izračunaj Yd signal
- ul=(yg+yd)*(-al); //Ulaz u B-spline filtar
- Wcas=ul+al*Wbuf[0]; //Izlaz iz B-spline filtra
- Push(Wbuf,4,Wcas); //Spremi W(z) u cirk. Spremnik
- Push(Ybuf,4,yg); //Spremi Y(z) u cirk. Spremnik
- bw=3*Wbuf[1]-3*Wbuf[3]; //Komponenta Bw
- by=0; //Komponenta By
- b=bw+by; B[br]=b; //Izračunaj parametar B i pohrani ga u spremnik
- MaxErr(fB,B,N); //Maksimalna pogreška dobivenih i referntnih za parametara B

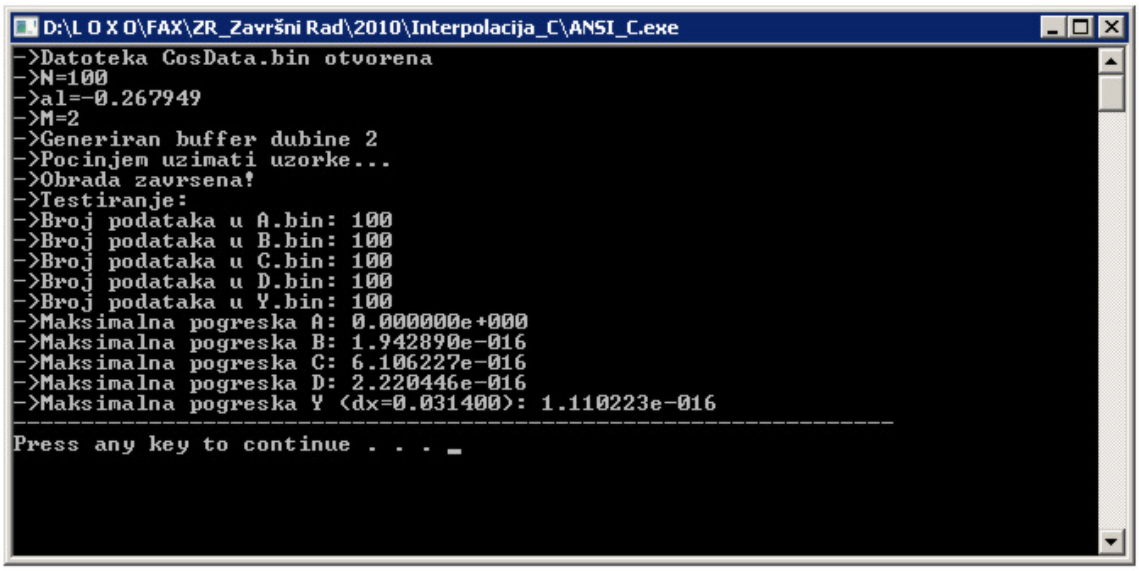
- Xf=0.0314; //Frakcionalno kašnjenje
- for(X=0;X<N;X++){ //Rekonstrukcija signala

  yi=((D[X]*Xf+C[X])*Xf+B[X])*Xf+A[X];

  Y[X]=yi;}

- MaxErr(fY,Y,N); //Maksimalna pogreška dobivenih i referentnih rekonstruiranih signala

U gore navedenom kôdu prikazan je samo način izračuna parametra B. Ostali parametri dobivaju se analogno prikazanom. Cijeli kôd nalazi se u *Prilogu D: ANSI_C_main.*



Slika 6-1 Rezultati ANSI C implementacije

Na slici gore (Slika 6-1) primjećujemo da su pogreške proračuna parametara zanemarive (reda 10^{-6}) te zaključujemo da je implementacija ispravna i spremna za završnu fazu koja slijedi – Implementacija u DSP procesor.

7. Implementacija kôda u VisaIDSP++

U prvoj fazi razvoja implementirani su referentni kod i generirani referentni parametri u Matlabu. U drugoj fazi simulirani su slijedni dohvat i obrada signala u ANSI C-u. U ovoj fazi potrebno je implementirati efikasni kôd koji će se vrtjeti na DSP procesoru Blackfin BF537. Prethodne dvije faze radile su sa *double* tipom podataka. Kako Blackfin BF537 nema ugrađene sklopove za manipulaciju podacima s pomičnim zarezom, potrebno je koristiti frakcionalni tip podataka.

Dinamika frakcionalnih varijabli se mijenja od parametra do parametra no konačni rezultati su poravnati sa formatom 1.14 koji je eksperimentalno određen. Korištene su optimizirane ugrađene funkcije za matematičke operacije množenja, zbrajanja i zaokruživanja nad frakcionalnim brojevima. Nalaze se u biblioteci fract.h, a prototipovi su slijedeći:

- fract32 mult_fr1x32(fract16 f1, fract16 f2); //Množenje: f1 * f2
- fract32 add_fr1x32(fract32 f1, fract32 f2); //Zbrajanje: f1 + f2
- fract16 round_fr1x32(fract32 f1) //Zaokruživanje na 16-bita

Sve se operacije vrše nad frakcionalnim tipovima veličine 16-bita. Nakon množenja dvije takve varijable rezultat se sprema u 32-bitni registar i potrebno ga je zaokružiti na 16-bitnu vrijednost.

Primjer ispravnog množenja dva 16-bitna broja:

```
fract16 x, a = 1000, b = 3000;
x = round_fr1x32(mult_fr1x32(a, b));
```

Ukupna implementacija programa u VisualDSP++ okruženju podijeljena je, radi preglednosti, na više dijelova (datoteka). Glavni program, inicijalizacija, posluživanje prekida, obrada signala i razvijene vlastite funkcije.

7.1 Glavni program – main.c

Stvara globalne i lokalne varijable,alocira cirkularne spremnike, inicijalizira frakcionalne vrijednosti, poziva inicijalizacijske funkcije i završava u beskonačnoj petlji očekujući prekide vanjskih jedinica. U nastavku slijede ključni dijelovi kôda, a cijeli se nalazi u *prilogu E: main.c.*

```
void main(void){
…
al_fr=-8780;            //Frakcionalna vrijednost al=Sqrt(3)-2
…
A=(fract16*)malloc(CIRC_SIZE*sizeof(fract16));          //Cirkularni spremnici
B=(fract16*)malloc(CIRC_SIZE*sizeof(fract16));
C=(fract16*)malloc(CIRC_SIZE*sizeof(fract16));
D=(fract16*)malloc(CIRC_SIZE*sizeof(fract16));
for(i=0;i<CIRC_SIZE;i++){D[i]=0; C[i]=0; B[i]=0; A[i]=0;}          //Inicijalizacija
…
Init_Flags();
Audio_Reset();
Init_Sport0();
Init_DMA();
Init_Interrupts();
Enable_DMA_Sport0();

while(1);       //Beskonačna petlja
}
```

7.2 Inicijalizacija

Sadrži implementirane funkcije za inicijalizaciju periferija, DMA kontrolera i prekidnih rutina.

Popis funkcija:

- void Init_Flags(void); //Inicijalizacija tipkala i LED dioda
- void Audio_Reset(void); //Reset A/D i D/A pretvornika
- void Init_Sport0(void); //Inicijalizacija serijske komunikacije
- void Init_DMA(void); //Inicijalizacija DMA kontrolera
- void Enable_DMA_Sport0(void) //Omogućavanje DMA podrške za A/D i D/A pretvornike
- void Init_Interrupts(void); //Inicijalizacija prekida

Detaljniji opis funkcija i njihov potpuni kôd nalazi se u *prilogu F: Initialize.c*.

7.3 Posluživanje prekida

Implementirana su dva prekidna vektora. Jedan poslužuje DMA jedinicu sa serijske komunikacije svaki puta kada A/D pretvornik pošalje jedan podatak. Sprema podatak u odgovarajuću varijablu te ga šalje na obradu preko funkcije Process_Data.

Drugi prekidni vektor poslužuje tipkala pomoću kojih namještamo cjelobrojni odnosno frakcionalni pomak signala.

Prototip prve i druge prekidne funkcije:

- EX_INTERRUPT_HANDLER(Sport0_RX_ISR);
- EX_INTERRUPT_HANDLER(PORTF_IntA_ISR);

Detaljniji opis funkcija i njihov potpuni kôd nalazi se u *prilogu G: ISR.c*.

7.4 Obrada signala

Obrada signala obuhvaća provlačenje novog uzorka kroz FIR filtar drugog reda i B-spline filtar. Nakon toga sprema parametre u veliki cirkularni spremnik kako bi se omogućio pomak signala te izbacuje vrijednost interpoliranog uzorka na traženom mjestu – u ovisnosti o cjelobrojnom i frakcionalnom podešenom pomaku. Sve je to implementirano u jedinstvenu funkciju prototipa:

```
void Process_Data(void);
```

Sadržaj kôda sličan je onom pisanom u ANSI C-u s razlikom da se ovdje svaka matematička operacija realizira upotrebom ugrađenih funkcija, a ne pomoću operatora plus i puta. Također je dodan dio za kontrolu pomaka signala preko tipkala.

Pomak signala izveden je tako da prividno omogućava pomicanje u plus i minus, odnosno signal možemo ili kasniti ili ubrzati. To je postignuto namještanjem dva pokazivača cirkularnog spremnika. Jedan je referentan i neprestano se vrti po spremniku, dok je drugi pomaknut lijevo, desno ili je poravnat s referentnim. U usporedbi s ulaznim signalom u A/D pretvornik, izlazni signali uvijek kasne s minimalnim kašnjenjem od četiri uzorka.

Detaljniji opis funkcije i njezin potpuni kôd nalazi se u *prilogu H: Process_data.c*

7.5 Razvijene vlastite funkcije

Kako bi se olakšalo programiranje razvijene su manje funkcije koje se često koriste:

```
fract16 *GenCo(int _M, const fract16 _al);          //Generira koeficijente FIR filtra

double Zaokruzi(double x);              //Služi kao pomoć prethodnoj funkciji kod
                                        zaokruživanja double broja

fract16 Interpoliraj(unsigned int X, fract16 dx);   //Nalazi vrijednost
                                                    frakcionalnog pomaka od
                                                    traženog uzorka

void GenerirajPomak(int C, int F);              //Generira pomak za cjelobrojni C i
                                                frakcionalni F
```

Detaljniji opis funkcija i njihov potpuni kôd nalaze se u *prilogu I: MojeFunkcije.c.*

8. Uhodavanje i rezultati

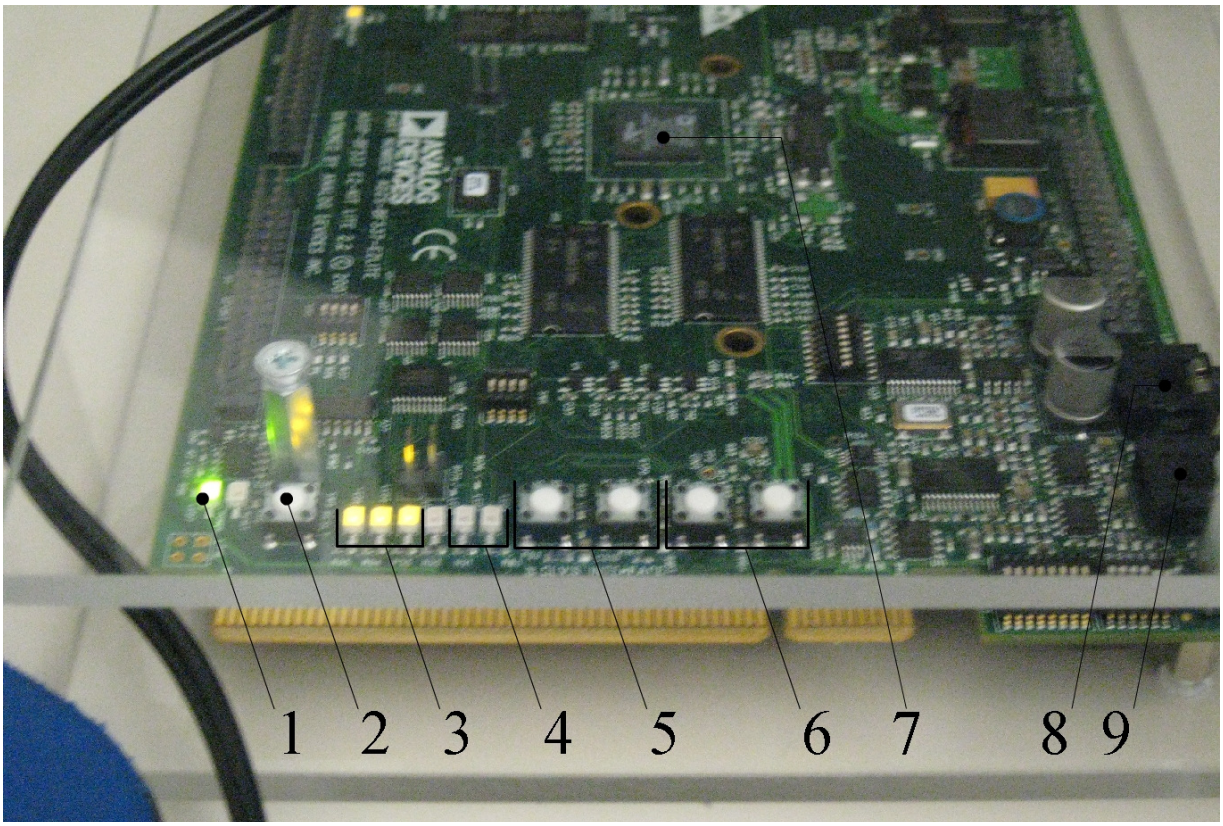
Nakon razvijenog programa unutar programskog okruženja VisualDSP++ provodi se prevođenje istoga, njegovo kopiranje u memoriju Blackfin procesora te pokretanje i ispitivanje ispravnosti rada.

Ukoliko je prevođenje uspješno javlja se poruka: „Build completed successfully.“, što znači da nema greške u sintaksi programa i da je spremno za slijedeći korak, prijenos programa u Blackfin memoriju. Program se tada jednostavno pokreće pritiskom na tipku *F5* ili naredbom *run.*

Za potrebe ispitivanja sustava, na ulaz A/D pretvornika dovodimo sinusni signal frekvencije 12kHz što je četiri puta manje od frekvencije uzorkovanja (48 kHz). Ulazni signal generiran je pomoću generatora funkcije. Izlazni, obrađeni signal pratimo na analognom osciloskopu. Na prvi kanal osciloskopa dovodimo nepomaknuti, ali obrađeni signal (referentni), dok na drugi kanal dovodimo pomaknuti obrađeni signal (promatrani). Bitno je napomenuti da originalni ulazni signal ne pratimo jer nije interesantan za efekt diferencijalnog pomaka signala kojeg pokušavamo dobiti. Pošto moramo imati mogućnost pomicanja signala u realnoj domeni, dodana su četiri tipkala, po dva u paru. Jedan par služi za podešavanje cjelobrojnog pomaka signala (+/-) dok drugi analogno prvom omogućava namještanje frakcije. Kako bi se pojednostavila demonstracija cjelokupnog uređaja, namještanje frakcije je moguće u osam koraka (od 0 do 7) gdje jedan korak predstavlja pomak od 12.5% cjelobrojnog pomaka. Naravno, upotrebom digitalnog potenciometra lako bi se izvelo finije podešavanje. Trenutni položaj pomaka može se pratiti pomoću signalizacije izvedene LED diodama. Imamo dva skupa LED dioda, za praćenje cjelobrojnog pomaka (dvije LED diode) i za frakcije (tri LED diode). Ukoliko je promatrani signal u fazi s referentnim, početni položaj bez cjelobrojnog pomaka, signalizacija je ugašena. Kada promatrani signal kasni svijetli desna LED dioda, a kada rani lijeva. Frakcionalni pomak očitava se binarnom reprezentacijom skupine od tri LED dioda. Kada su sve tri

ugašene nema pomaka frakcijama, a kada su sve tri upaljene (krajnji položaj) predstavljaju pomak u 7 koraka odnosno 87.5% cjelobrojnog pomaka (7 x 12.5%). Frakcionalni pomak od 100% cjelobrojnog pomaka nije potreban jer se može namjestiti upotrebom cjelobrojnog pomaka za jedan korak.

Gore navedeno može se predočiti slijedećom slikom (Slika 8-1) i pripadnom legendom (Tablica 8-1).



Slika 8-1 Pregled korištenog sučelja

Tablica 8-1 Legenda za sliku 8-1

Broj	Objašnjenje
1	Signalizacija uključenosti makete
2	Tipka za reset
3	Signalizacija frakcionalnog pomaka
4	Signalizacija cjelobrojnog pomaka
5	Tipkala za namještanje cjelobrojnog pomaka
6	Tipkala za namještanje frakcionalnog pomaka
7	Blackfin BF537 procesor
8	Priključak za izlazne signale
9	Priključak za ulazni signal

Osnovni uvjet ispravnosti rada ovakvog sustava jest vrijeme obrade signala. Obrada signala mora se sprovesti unutar vremena jedne periode frekvencije uzorkovanja. Da bi provjerili koliko vremena potroši razvijena funkcija *Process_Data(),* koristimo posebne funkcije VisualDSP++ okruženja koje nam omogućavaju takvo mjerenje:

```
//Poseban tip varijabli za brojanje ciklusa
cycle_t start_count;
cycle_t final_count;

START_CYCLE_COUNT(start_count); //Počni brojati cikluse

Process_Data(); //Obradi podatak

STOP_CYCLE_COUNT(final_count,start_count); //Zaustavi brojanje ciklusa
printf("Broj ciklusa: %d\n",final_count); //Ispiši izmjereni broj ciklusa
```

Korištenje funkcija START_CYCLE_COUNT i STOP_CYCLE_COUNT uspješno je samo kada se program pokreće u *release* modu, a potrebno je i podesiti opciju u *Project Options – Compile – General* upisom naredbe -DDO_CYCLE_COUNTS u polje *Addition options*.

Nakon pokretanja programa dobivamo ispis:

Broj ciklusa: 529

Dakle izmjeren je broj od 529 ciklusa. Slobodno ga zaokružujemo na 530 ciklusa jer je to rigorozniji uvjet. Poznajući frekvenciju takta procesora (*fo*), frekvenciju uzorkovanja (*fs*) i broj ciklusa za obradu jednog uzorka signala (*N*) računamo prema (9) maksimalnu moguću frekvenciju uzorkovanja (fs_{max}):

$$fo = 300MHz$$
$$fs = 48kHz$$
$$N = 530$$

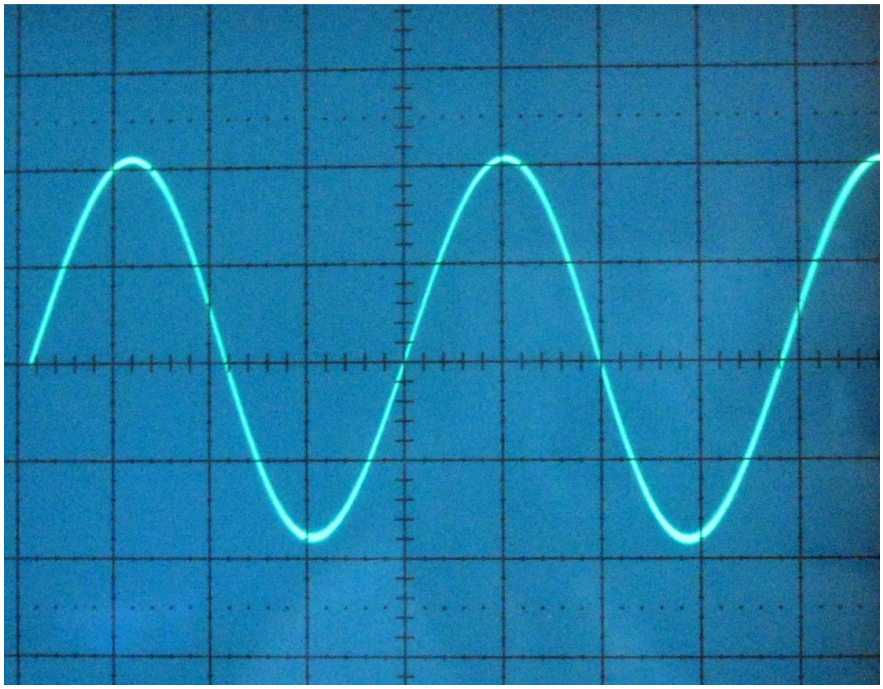
$$Ts_{max} = N * To \quad (7)$$
$$f = \frac{1}{T} \quad (8)$$
$$fs_{max} = \frac{fo}{N} \quad (9)$$

$$fs_{max} = \frac{300 * 10^6}{530} \approx 566\ kHz$$

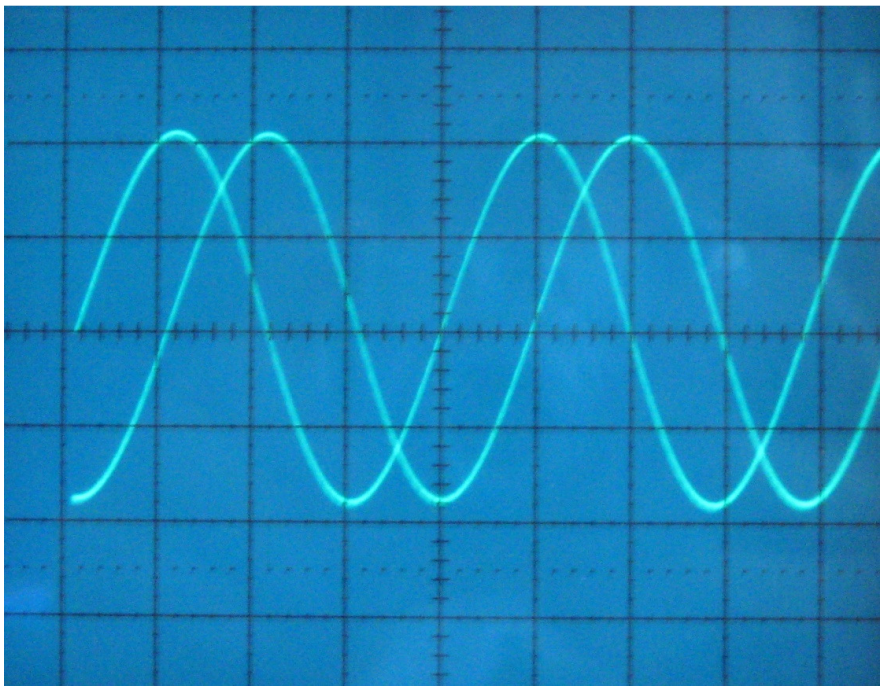
Dobivamo maksimalnu frekvenciju uzorkovanja 566 kHz, dok je korištena 48 kHz. Time smo provjerili da je obrada moguća i zaključili da potrebno vrijeme obrade jednog uzorka signala nadilazi kritični uvjet 12 puta.

U nastavku su prikazane fotografije signala prikazanih analognim osciloskopom za neke tipične ispitne pomake.

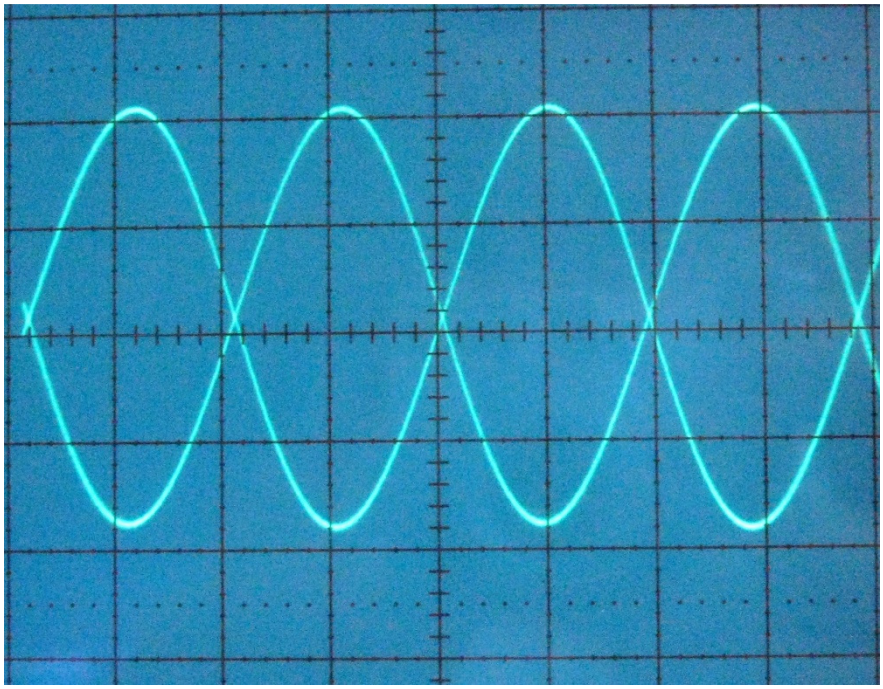
Na fotografiji (Slika 8-2) prikazano je inicijalno stanje. Promatrani i referentni signali su u fazi. Slijede prikazi cjelobrojnog pomaka signala za 90° (Slika 8-3), 180° (Slika 8-4) i 270° (Slika 8-5).



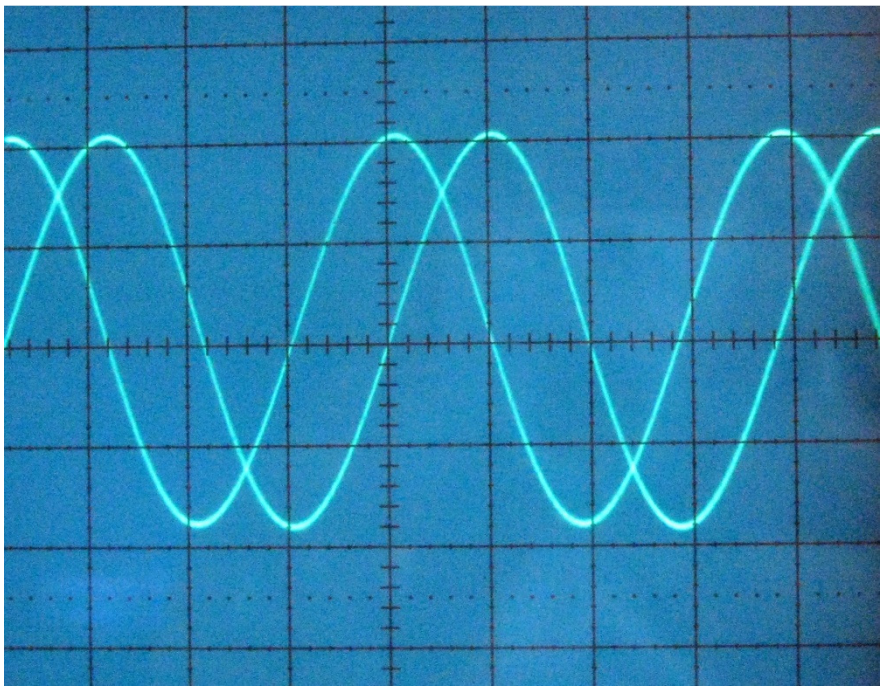
Slika 8-2 Inicijalno stanje bez pomaka signala



Slika 8-3 Cjelobrojni pomak signala za 90°

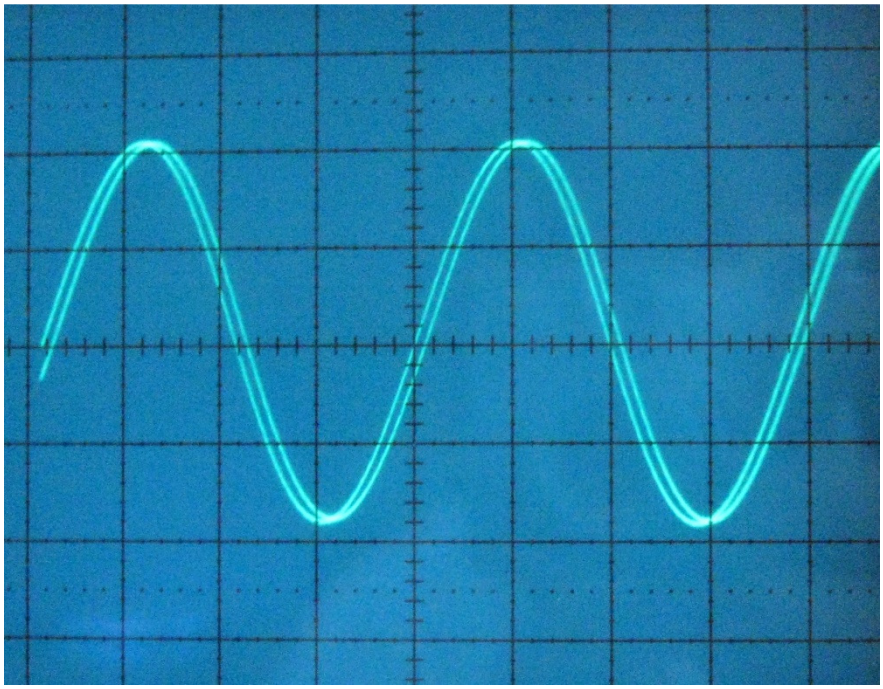


Slika 8-4 Cjelobrojni pomak signala za 180°



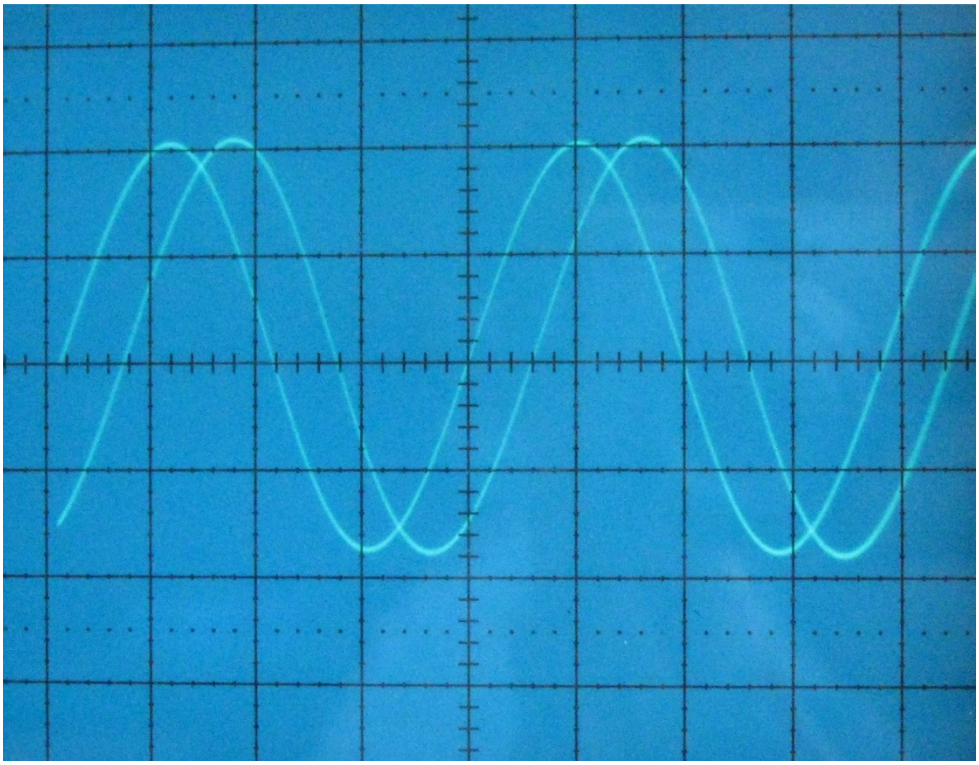
Slika 8-5 Cjelobrojni pomak signala za 270°

Na slikama gore prikazani su cjelobrojni pomaci signala frekvencije 12 kHz. Primjećujemo da za jedan cjelobrojni korak signal pomičemo za jednu dekadu. Frakcionalni pomak ostvaren B-spline interpolacijom omogućava nam pomak signala i unutar jedne dekade. Slijedeće fotografije prikazuju ispitivanje frakcionalnog pomaka signala pri cjelobrojnom pomaku jednakom nuli.

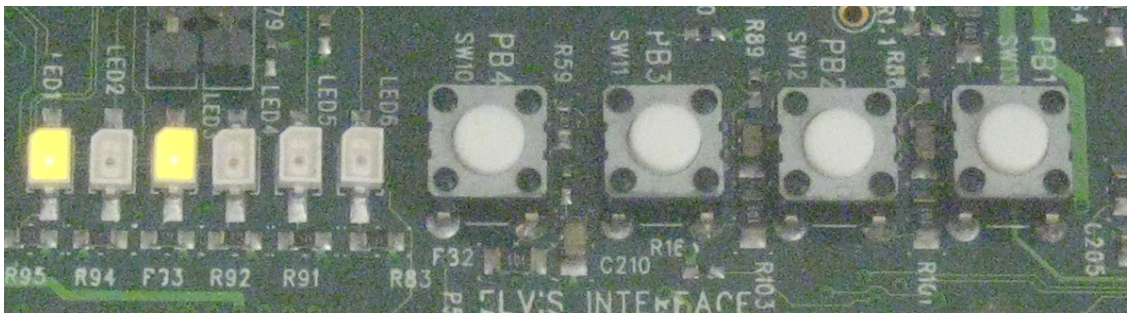


Slika 8-6 Frakcionalni pomak signala za 1 frakciju

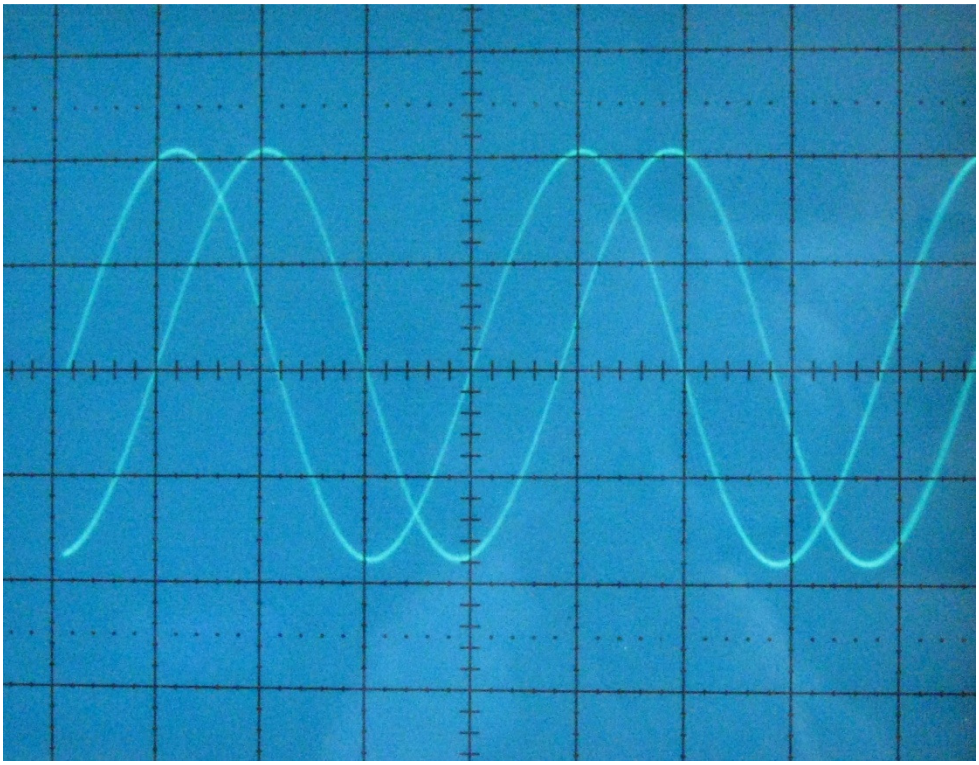
U nastavku su prikazana dva para fotografija. Prikaz frakcionalnih pomaka od 5 (101_2) (Slika 8-7) i od 7 (111_2) (Slika 8-9) frakcija uz prikaz signalizacije LED diodama na fotografijama (Slika 8-8) i (Slika 8-10). LSB je bit lijevo.



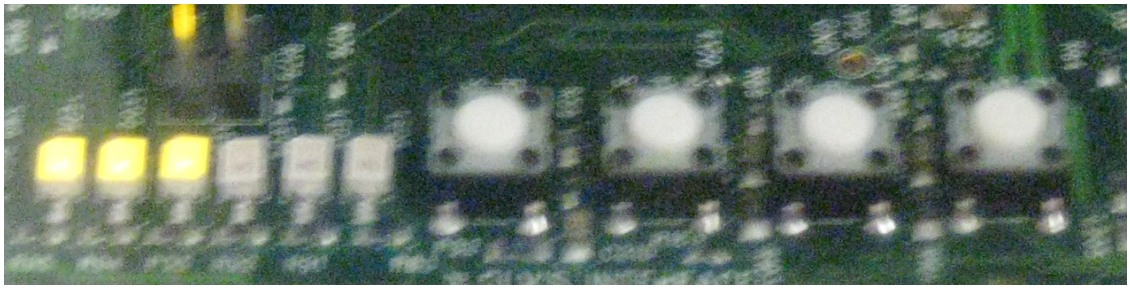
Slika 8-7 Frakcionalni pomak signala za 5 frakcija



Slika 8-8 Signalizacija za pomak od 5 frakcija



Slika 8-9 Frakcionalni pomak signala za 7 frakcija



Slika 8-10 Signalizacija za pomak od 7 frakcija

9. Zaključak

Kroz tri velike faze uhodavanja sustava, u završnom radu, ostvarena je implementacija digitalnog sustava za frakcionalni pomak signala na Blackfin BF537 procesoru. Konačni rezultati dali su zadovoljavajuće pogreške u usporedbi s referentnim kôdom. Vrijeme obrade jednog uzorka daleko je nadmašilo kritično vrijeme uzorkovanja. Time se dokazuje prednost DSP procesora pred procesorima opće namjene na kojima ova realizacija ne bi bila moguća. Način interpretacije brojeva unutar DSP procesora ključan je faktor. Koristeći cjelobrojnu aritmetiku značajno se ubrzalo vrijeme izvođenja obrade jednog uzorka ali nauštrb ukupne preciznosti rezultata.

Ovaj sustav mogao bi se primijeniti u raznim sklopovima za sinkronizaciju zvučnog ili video signala. U daljnjem razvoju mogla bi se unaprijediti preciznost brojeva te optimizirati assembleski kôd.

10. Literatura

1) Wikipedia, Interpolacija, http://hr.wikipedia.org/wiki/Interpolacija, 19. 4. 2010.
2) Fractional Value Built-in Functions in C, *Blackfin GCC Built-in Functions,* http://docs.blackfin.uclinux.org/doku.php?id=toolchain:built-in_functions, 19. 4. 2010.
3) Petrinović, D. Causal Cubic Splines: Formulations, Interpolation Properties and Implementations. IEEE Transactions on Signal Processing. Vol. 56, No. 11, studeni 2008., str. 5442-5453.
4) Petrinović, D., Uvod u procesore za digitalnu obradu signala (DSP), 26. 5. 2009., *Projektiranje ugradbenih računalnih sustava,* http://www.fer.hr/_download/repository/PURS01_prezentacija.pdf, 19.4.2010.
5) Petrinović, D., DSP procesori porodice Blackfin - Analog Devices, 2. 6. 2009., *Projektiranje ugradbenih računalnih sustava,* http://www.fer.hr/_download/repository/PURS02_prezentacija.pdf, 19.4.2010.

11. Sažetak

Digitalni sustav za frakcionalno kašnjenje signala

U programu Matlab razvijen je B-spline interpolator 2. reda koji daje izvrsne razultate. Rezultati i međurezultati iz Matlaba pohranjeni su u datoteke i prozvani referentnim rezultatima.

Koristeći standardiziranost ANSI C jezika razvijena je aplikacija u programskom okruženju Dev-C++. Aplikacija simulira uzimanje uzoraka kosinusnog signala, njegovu obradu te upis rezultata i međurezultata u odgovarajuće datoteke. Također uspoređuje svoje rezultate sa referentnim rezultatima gdje je pogreška minimalna.

Program razvijen u ANSI C-u prenosi se u programsko okruženje VisualDSP++ uz dodavanje prekidnih rutina, rutina za sučelje, inicijalizacija DSP procesora i prelazak na cjelobrojnu aritmetiku. Takav se prevodi i implementira u DSP procesor Blackfin BF537.

Osnovne komponente sustava su memorijski međuspremnik pomoću kojeg se ostvaruje kašnjenje i B-spline interpolator koji omogućava generiranje frakcije između dva cjelobrojna uzorka.

Sustav se ispituje promatranjem signala na osciloskopu koje dobivamo preko priključaka razvojne ploče za Blackfin procesore EZ-Kit Lite. Razvojna ploča sadrži sučelje za korisnika te A/D i D/A pretvornike koji omogućuju ispitivanje sustava nad realnim signalima.

Prilog A: GenCosData.m

```
clear                              %Brisanje svih varijabli
N=100;                             %Definiranje 100 uzoraka
Fs=48e3;                           %Frekvencija otipkavanja 48kHz
Ts=1/Fs;
t=[0:Ts:Ts*N-Ts];
f=(1/(Ts*(N-1)))*2;
x=cos(2*pi*f*t);                   %Generiranje funkcije u 100 uzoraka
xqi=round(x*2^15);
xq=xqi/2^15;

fp=fopen('CosData.bin','wb');      %Otvori datoteku za binarni upis
fwrite(fp,x,'double');             %Binarni upis u datoteku
fclose(fp);                        %Zatvori datoteku


fp=fopen('CosData.bin','rb');      %Otvaranje datoteke za provjeru
[X,n]=fread(fp,inf,'double');      %Binarno čitanje iz datoteke
fclose(fp);                        %Zatvori datoteku
```

Prilog B: Primjer.m

```
clear

fp=fopen('CosData.bin','rb');    %Otvaranje datoteke
[x,N]=fread(fp,inf,'double');    %Čitanje ulaznog signala
fclose(fp);                      %Zatvaranje datoteke
x=x';

fp_A=fopen('A.bin','wb');        %Otvaranje datoteka...
fp_B=fopen('B.bin','wb');
fp_C=fopen('C.bin','wb');
fp_D=fopen('D.bin','wb');
fp_Y=fopen('Y.bin','wb');

al=sqrt(3)-2;                    %Konstanta

M=2;                             %Red FIR filtra
co=al.^[M:-1:1];                 %Generiranje koeficijenata FIR
filtra

yg=[zeros(1,M) x(1:end-M)];      %Yg je zakašnjeli signal
yd=filter(co,1,x);               %Izlaz iz FIR filtra

%kaskadni
ul=(yg+yd)*(-al);                %Ulazni signal u B-spline filtar
wcas=filter(1,[1 -al],ul);       %Izlaz iz B-spline filtra

w=wcas;                          %Odabir kaskadne realizacije filtra

dcw=3*[1 1 -1 -1];               %Matrica Dw komponente
ccw=-3*[1 2 -1 -2];              %Matrica Cw komponente
bcw=3*[0 1 0 -1];                %Matrica Bw komponente
acw=[0 0 0 0];                   %Matrica Aw komponente

dcy=-2*[0 1 -1 0];               %Matrica Dy komponente
ccy=3*[0 1 -1 0];                %Matrica Dy komponente
bcy=[0 0 0 0];                   %Matrica Dy komponente
acy=[0 0 1 0];                   %Matrica Dy komponente


d=filter(dcw,1,w)+filter(dcy,1,yg);     %Parametar D
c=filter(ccw,1,w)+filter(ccy,1,yg);     %Parametar C
b=filter(bcw,1,w)+filter(bcy,1,yg);     %Parametar B
a=filter(acw,1,w)+filter(acy,1,yg);     %Parametar A

i=[1:100];                              %Vektor X-osi
dx=0.0314;                              %Frakcionalno kašnjenje
Yout=((d(i)*dx+c(i))*dx+b(i))*dx+a(i);  %Rekonstrukcija signala

p1=plot(i,Yout); hold on;
p2=plot(i,x,'.'); hold off;
```

```
legend([p1,p2],'Rekonstruirani signal','Ulazni uzorci',1);
%plot(i,Yout,n,x(5:end),'.');            %Crtanje signala



fwrite(fp_A,a,'double');                 %Upis parametra A u datoteku
fwrite(fp_B,b,'double');                 %Upis parametra B u datoteku
fwrite(fp_C,c,'double');                 %Upis parametra C u datoteku
fwrite(fp_D,d,'double');                 %Upis parametra D u datoteku
fwrite(fp_Y,Yout,'double');              %Upis rekonstruiranog
                                          signala u datoteku

fclose(fp_A);                            %Zatvaranje datoteka...
fclose(fp_B);
fclose(fp_C);
fclose(fp_D);
fclose(fp_Y);
```

Prilog C: ANSI_C_funkcije

```
// --- Generira koeficijente FIR filtra ---
double *GenCo(int M,  const double al){
        double *tmp;
        int i,j;
        tmp=(double*)malloc(M*sizeof(double));
        for(i=0,j=M;i<M;i++,j--) tmp[i]=pow(al,j);
return tmp;
}

// --- Sprema podatak na stog ---
void Push(double *p,int M,double x){
        int i;
        for(i=M-1;i>0;i--)p[i]=p[i-1];
        p[i]=x;
return;
}

// --- Broji ulazne podatke iz datoteke ---
int BrojPodataka(FILE *f){
        long lSize;
        int n;
        fseek (f , 0 , SEEK_END);
        lSize = ftell (f);
        rewind (f);
        n=(int)lSize/sizeof(double);
return n;
}

// --- Traži maksimalnu pogrešku sa referentnim parametrima ---
double MaxErr(FILE *f,  double P[],  int N){
        double max=0,tmp;
        double err;
        int i;
        rewind(f);
        for(i=0;i<N;i++){
                fread(&tmp,sizeof(double),1,f);
                err=P[i]-tmp;
                if(err<0)err=-err;
                if(err>max)max=err;
        }
        rewind(f);
return max;
}
```

Prilog D: ANSI_C_main

```
#include <stdio.h>
#include <stdlib.h>

double *GenCo(int M,  const double al);
void Push(double *p,int M,double x);
int BrojPodataka(FILE *f);
double MaxErr(FILE *f,  double P[],  int N);

int main(){
	FILE *Data,*Cos,*fA,*fB,*fC,*fD,*fY;
	int N=0,i,br,X;
	double Xf,yi;
	double *dta,*co,*buff;
	double x,yg,yd,ul,Wcas,Wbuf[4],Ybuf[4];
	double d,c,b,a,*D,*C,*B,*A,*Y;
	double dw,cw,bw,aw,dy,cy,by,ay;
	const double al=sqrt(3)-2;
	const int M=2;
	char pom[20];
	long lSize;

	for(i=0;i<4;i++){Wbuf[i]=0; Ybuf[i]=0;}

	Cos=fopen("..\\CosData.bin","rb"); //Otvaranje datoteke
	printf("->Datoteka CosData.bin otvorena\n");
	N=BrojPodataka(Cos);
	printf("->N=%d\n",N);
	printf("->al=%f\n",al);
	printf("->M=%d\n",M);

	co=GenCo(M,al); //Generiraj koeficijente FIR filtra

	A=(double*)malloc(N*sizeof(double)); //Alokacija spremnika...
	B=(double*)malloc(N*sizeof(double));
	C=(double*)malloc(N*sizeof(double));
	D=(double*)malloc(N*sizeof(double));
	Y=(double*)malloc(N*sizeof(double));

	for(i=0;i<N;i++) //Inicijalizacija spremnika
		{D[i]=0; C[i]=0; B[i]=0; A[i]=0;}

	buff=(double*)malloc(M*sizeof(double));
	for(i=0;i<M;i++)buff[i]=0;
	printf("->Generiran buffer dubine %d\n",M);

	//Obrada podataka!!
	printf("->Pocinjem uzimati uzorke...\n");
	for(br=0;br<N;br++){
```

```
fread(&x,sizeof(double),1,Cos); //Čitaj podatak
yg=buff[M-1]; //Yg signal
Push(buff,M,x); //Stavi podatak u spremnik
for(i=0,yd=0;i<M;i++) yd+=co[i]*buff[i]; //Izračunaj Yd signal
ul=(yg+yd)*(-al); //Ulaz u B-spline filtar
Wcas=ul+al*Wbuf[0]; //Izlaz iz B-spline filtra
Push(Wbuf,4,Wcas); //Spremi W(z) u cirk. spremnik
Push(Ybuf,4,yg); //Spremi Y(z) u cirk. spremnik
dw=3*Wbuf[0]+3*Wbuf[1]-3*Wbuf[2]-3*Wbuf[3]; //Komponenta Dw
cw=-3*Wbuf[0]-6*Wbuf[1]+3*Wbuf[2]+6*Wbuf[3]; //Komponenta Cw
bw=3*Wbuf[1]-3*Wbuf[3]; //Komponenta Bw
aw=0; //Komponenta Aw
//-------------------------------------------
dy=-2*Ybuf[1]+2*Ybuf[2]; //Komponenta Dy
cy=3*Ybuf[1]-3*Ybuf[2]; //Komponenta Cy
by=0; //Komponenta By
ay=Ybuf[2]; //Komponenta Ay
//-------------------------------------------
d=dw+dy; D[br]=d; //Izračunaj parametar D i pohrani ga u spremnik
c=cw+cy; C[br]=c; //Izračunaj parametar C i pohrani ga u spremnik
b=bw+by; B[br]=b; //Izračunaj parametar B i pohrani ga u spremnik
a=aw+ay; A[br]=a; //Izračunaj parametar A i pohrani ga u spremnik
//-------------------------------------------
}
fclose(Cos); //Zatvori datoteku CosData.bin
printf("->Obrada zavrsena!\n");
printf("->Testiranje:\n");

fA=fopen("..\\A.bin","rb"); //Otvaranje referentnih datoteka
fB=fopen("..\\B.bin","rb");
fC=fopen("..\\C.bin","rb");
fD=fopen("..\\D.bin","rb");
fY=fopen("..\\Y.bin","rb");

printf("->Broj podataka u A.bin: %d\n",BrojPodataka(fA));
printf("->Broj podataka u B.bin: %d\n",BrojPodataka(fB));
printf("->Broj podataka u C.bin: %d\n",BrojPodataka(fC));
printf("->Broj podataka u D.bin: %d\n",BrojPodataka(fD));
printf("->Broj podataka u Y.bin: %d\n",BrojPodataka(fY));

//Ispis max pogreški...
printf("->Maksimalna pogreska A: %e\n",MaxErr(fA,A,N));
printf("->Maksimalna pogreska B: %e\n",MaxErr(fB,B,N));
printf("->Maksimalna pogreska C: %e\n",MaxErr(fC,C,N));
printf("->Maksimalna pogreska D: %e\n",MaxErr(fD,D,N));

Xf=0.0314; //Frakcionalno kašnjenje
//Xf=0.1;
for(X=0;X<N;X++){
      yi=((D[X]*Xf+C[X])*Xf+B[X])*Xf+A[X]; //Rekonstrukcija signala
      Y[X]=yi;
}
```

```
        printf("->Maksimalna pogreska Y (dx=%f): %e\n",Xf,MaxErr(fY,Y,N));

        fclose(fA); //Zatvaranje datoteka
        fclose(fB);
        fclose(fC);
        fclose(fD);
        fclose(fY);

        printf("-------------------------------------------------------\n");
        system("PAUSE");
        return 0;
}
```

Prilog E: Main.c

```
#include "Talkthrough.h"
#include <sysreg.h>
#include <ccblkfn.h>
#include <math.h>

//--- Globalne varijable ---
// Ulazni podaci za lijevi kanal iz AD1871
int iChannel0LeftIn, iChannel1LeftIn;
// Ulazni podaci za desni kanal iz AD1871
int iChannel0RightIn, iChannel1RightIn;
// Izlazni podaci lijevog kanala za AD1854
int iChannel0LeftOut, iChannel1LeftOut;
// Izlazni podaci desnog kanala za AD1854
int iChannel0RightOut, iChannel1RightOut;
// SPORT0 DMA transmit buffer
int iTxBuffer1[2];
// SPORT0 DMA receive buffer
int iRxBuffer1[2];
fract16 *co,*buff;
int N;

fract16 Wbuf[4];
fract16 Ybuf[4];
fract16 a,b,c,d;
fract16 buff_idx=0;
fract16 Wbuf_idx=0,Ybuf_idx=0;
fract16 al_fr, Xf;
int idx=0;
fract16 *D,*C,*B,*A,*Y;
bool T;
int idx_pass, idx_offset;
int P;
int Fr;
//----------------------------

void main(void){
	fract16 x;
	double ErrA,ErrB,ErrC,ErrD;
	fract32 tmp;
	int i, br, br2, X;

	al_fr=-8780; //Frakcionalna vrijednost al=Sqrt(3)-2
	T=false; //Onemogućeno slanje podataka na izlaz
	P=0; //Cjelobrojna inicijalizacija pomaka signala
	Fr=0; //Frakcionalna inicijalizacija pomaka signala
	GenerirajPomak(P,Fr);

	for(i=0;i<4;i++){Wbuf[i]=0; Ybuf[i]=0;} //Reset pričuvnih spremnika
```

```
        A=(fract16*)malloc(CIRC_SIZE*sizeof(fract16)); //Cirkularni spremnici…
        B=(fract16*)malloc(CIRC_SIZE*sizeof(fract16));
        C=(fract16*)malloc(CIRC_SIZE*sizeof(fract16));
        D=(fract16*)malloc(CIRC_SIZE*sizeof(fract16));

        for(i=0;i<CIRC_SIZE;i++) //Inicijalizacija cirkularnih spremnika
              {D[i]=0; C[i]=0; B[i]=0; A[i]=0;}

        co=GenCo(M,al_fr); //Generiraj koeficijente FIR filtra

        buff=(fract16*)malloc(M*sizeof(fract16)); //Spremnik FIR filtra
        for(i=0;i<M;i++)buff[i]=0;

        Init_Flags(); //Inicijalizacija porta PORTF
        Audio_Reset(); //Reset ADC i DAC
        Init_Sport0(); //Inicijalizacija serijske komunikacije
        Init_DMA(); //Inicijalizacija DMA
        Init_Interrupts(); //Inicijalizacija prekida
        Enable_DMA_Sport0(); //Omogućavanje DMA, ADC i DAC

        while(1); //Beskonačna petlja,
                  //čekanje i obrada podataka
}
```

Prilog F: Initialize.c

```
#include "Talkthrough.h"

// --- Konfiguracija porta PORTF za upravljanje ADC, DAC i LED ---
void Init_Flags(void){
	int temp;
	temp = *pPORTF_FER;
	temp++;
	//Isključi Function Enable Register
	*pPORTF_FER = 0x0000;
	*pPORTF_FER = 0x0000;
	//Odredi bitove za ulaz/izlaz
	*pPORTFIO_DIR = 0x1FC0;
	//Omogući ulaz za tipkala
	*pPORTFIO_INEN = 0x003C;
	//Ugasi sve LED diode
	*pPORTFIO_CLEAR = 0x0FC0;
	//Generiraj prekid na rastući brid
	*pPORTFIO_EDGE = 0x003C;
	//Maska (prekida) za tipkala
	*pPORTFIO_MASKA = 0x003C;
	//Pali LED diode
	*pPORTFIO_SET = 0x1FC0;
	//Gasi LED diode
	*pPORTFIO_CLEAR = 0x1FC0;}

// --- Reset ADC i DAC ---
void Audio_Reset(void){
	int i;
	for(i = 0; i< delay;i++){}; //Pričekaj neko vrijeme
	*pPORTFIO_SET = PF12; } //Postavi PORTF.12 u "1"

// --- Inicijalizacija serijske komunikacije ---
void Init_Sport0(void){
	// Sport0 receive configuration
	// External CLK, External Frame sync, MSB first, Active Low
	// 24-bit data, Secondary side enable, Stereo frame sync enable
	// Users of ADSP-BF537 EZ-KIT Board Rev 1.0 must enable the
	// internal clock and frame sync
	// *pSPORT0_RCR1 = RFSR | LRFS | RCKFE | IRFS | IRCLK;
	*pSPORT0_RCR1 = RFSR | RCKFE;
	*pSPORT0_RCR2 = SLEN_24 | RSFSE;
	// *pSPORT0_RCLKDIV = 0x0013;
	// *pSPORT0_RFSDIV = 0x001F;
	// Sport0 transmit configuration
	// External CLK, External Frame sync, MSB first, Active Low
	// 24-bit data, Secondary side enable, Stereo frame sync enable
	// Users of ADSP-BF537 EZ-KIT Board Rev 1.0 must enable the
	// internal clock and frame sync
	// *pSPORT0_TCR1 = TFSR | LTFS | TCKFE | ITFS | ITCLK;
```

```
        *pSPORT0_TCR1 = TFSR | TCKFE;
        *pSPORT0_TCR2 = SLEN_24 | TSFSE;
        // *pSPORT0_TCLKDIV = 0x0013;
        // *pSPORT0_TFSDIV = 0x001F;
}

// --- Inicijalizacija DMA ---
void Init_DMA(void){
        //Konfiguracija DMA3
        //32-bitni transfer, generira prekid,
        //koristi automatski spremnik
        *pDMA3_CONFIG = WNR | WDSIZE_32 | DI_EN | FLOW_1;
        // Početna adresa spremnika
        *pDMA3_START_ADDR = iRxBuffer1;
        // DMA3 brojač
        *pDMA3_X_COUNT = 2;
        // DMA3 adresni inkrement
        *pDMA3_X_MODIFY = 4;
        //Konfiguracija DMA4
        //32-bitni transfer, koristi automatski spremnik
        *pDMA4_CONFIG = WDSIZE_32 | FLOW_1;
        // Početna adresa spremnika
        *pDMA4_START_ADDR = iTxBuffer1;
        // DMA4 brojač
        *pDMA4_X_COUNT = 2;
        // DMA4 adresni inkrement
        *pDMA4_X_MODIFY = 4;}

// --- Omogućavanje DMA, ADC i DAC ---
void Enable_DMA_Sport0(void){
        // Omogući DMA-ove
        *pDMA4_CONFIG = (*pDMA4_CONFIG | DMAEN);
        *pDMA3_CONFIG = (*pDMA3_CONFIG | DMAEN);
        // Omogući Sport0 TX i RX
        *pSPORT0_TCR1 = (*pSPORT0_TCR1 | TSPEN);
        *pSPORT0_RCR1 = (*pSPORT0_RCR1 | RSPEN);}

// --- Inicijalizacija prekida za Sport0 RX ---
void Init_Interrupts(void){
        //Postavi Sport0 RX (DMA3) prekidni prioritet na 2 = IVG9
        *pSIC_IAR0 = 0xff2fffff;
        *pSIC_IAR1 = 0xffffffff;
        *pSIC_IAR2 = 0xffffffff;
        //*pSIC_IAR3 = 0xffffffff;
        *pSIC_IAR3 = 0xffff5fff;
        //Pridruživanje prekidnih vektora
        register_handler(ik_ivg9, Sport0_RX_ISR); //ADC
        register_handler(ik_ivg12, PORTF_IntA_ISR); //LED
        //Omogući prekde za: Sport0 RX i tipkala
        *pSIC_IMASK = 0x00000020 | 0x08080000;
}
```

Prilog G: ISR.c

```
#include "Talkthrough.h"

// --- Obrada prekida ADC-a ---
EX_INTERRUPT_HANDLER(Sport0_RX_ISR)
{
	//Potvrdi prekid
	*pDMA3_IRQ_STATUS = 0x0001;

	//Kopiranje podataka sa DMA u varijable
	iChannel0LeftIn = iRxBuffer1[INTERNAL_ADC_L0];
	iChannel0RightIn = iRxBuffer1[INTERNAL_ADC_R0];

	Process_Data(); //Obradi podatke

	//Kopiraj obrađene podatke u izlazni spremnik za DAC
	iTxBuffer1[INTERNAL_DAC_L0] = iChannel0LeftOut;
	iTxBuffer1[INTERNAL_DAC_R0] = iChannel0RightOut;
}

// --- Obrada prekida tipkala ---
EX_INTERRUPT_HANDLER(PORTF_IntA_ISR)
{
	unsigned int i;
	int p,f;
	p=0;
	f=0;
	for(i=0;i<10000000;i++); //Pričekaj istitravanje tipkala

	//Provjeri izvor prekida
	if((*pPORTFIO & 4)) f++;
	if((*pPORTFIO & 8)) f--;
	if((*pPORTFIO & 16)) p++;
	if((*pPORTFIO & 32)) p--;

	//Potvrdi prekid
	*pPORTFIO_CLEAR = 4|8|16|32;
	GenerirajPomak(p,f); //Generiraj promjenu pomaka signala
}
```

Prilog H: Process_data.c

```
#include "Talkthrough.h"

// --- Obrađuje ulazni uzorak signala ---
void Process_Data(void)
{
	fract16 yg, yd, ul, Wcas;
	fract16 dw, cw, bw, aw;
	fract16 dy, cy, by, ay;
	fract32 acc=0,pom,pom1,pom2,pom3,pom4;
	fract16 yi;
	int i;

	yg=buff[buff_idx]; //Zakšnjeli ulazni signal Yg

	buff[buff_idx]=iChannel0LeftIn>>8; //Učitaj novi podatak u 16-bita

	buff_idx=circindex(buff_idx,1,M); //Spremi ga u cirk. Spremnik

	for(i=0;i<M;i++){ //FIR filtar
		pom=mult_fr1x32(co[(M-1)-i],buff[buff_idx]);
		pom=round_fr1x32(pom);
		acc=add_fr1x32(acc,pom);
		buff_idx=circindex(buff_idx,1,M);
	}
	yd=acc; //Rezultat spremi u Yd

	//ul=(yg+yd)*(-al);
	pom=add_fr1x32(yg,yd);
	pom=mult_fr1x32(pom,-al_fr);
	ul=round_fr1x32(pom); //Signal Ul

	//B-spline filtar
	Wbuf_idx=circindex(Wbuf_idx,-1,4);
	pom=mult_fr1x32(al_fr,Wbuf[Wbuf_idx]);
	pom=round_fr1x32(pom);
	pom=add_fr1x32(ul,pom);
	Wcas=pom; //Izlaz iz B-spline filtra

	//Spremanje signala Wcas i Yg u spremnike
	Wbuf_idx=circindex(Wbuf_idx,1,4);
	Wbuf[Wbuf_idx]=Wcas;
	Wbuf_idx=circindex(Wbuf_idx,1,4);
	Ybuf[Ybuf_idx]=yg;
	Ybuf_idx=circindex(Ybuf_idx,1,4);
//-----------------------------------------------------------------------------------------
	//Dw komponenta
	//dw=3*z(0)+3*z(-1)-3*z(-2)-3*z(-3)
	Wbuf_idx=circindex(Wbuf_idx,-1,4);
	acc=3<<13; //Format 2.13
```

```
if(Wbuf[Wbuf_idx]&2)
	pom=(Wbuf[Wbuf_idx]>>2)+1;
else
	pom=(Wbuf[Wbuf_idx]>>2);
pom1=(mult_fr1x32(pom,acc))>>13;

Wbuf_idx=circindex(Wbuf_idx,-1,4);
acc=3<<13; //Format 2.13
if(Wbuf[Wbuf_idx]&2)
	pom=(Wbuf[Wbuf_idx]>>2)+1;
else
	pom=(Wbuf[Wbuf_idx]>>2);
pom2=(mult_fr1x32(pom,acc))>>13;

Wbuf_idx=circindex(Wbuf_idx,-1,4);
acc=-3<<13; //Format 2.13
if(Wbuf[Wbuf_idx]&2)
	pom=(Wbuf[Wbuf_idx]>>2)+1;
else
	pom=(Wbuf[Wbuf_idx]>>2);
pom3=(mult_fr1x32(pom,acc))>>13;

Wbuf_idx=circindex(Wbuf_idx,-1,4);
acc=-3<<13; //Format 2.13
if(Wbuf[Wbuf_idx]&2)
	pom=(Wbuf[Wbuf_idx]>>2)+1;
else
	pom=(Wbuf[Wbuf_idx]>>2);
pom4=(mult_fr1x32(pom,acc))>>13;

//Množenje ima 2 predznaka, jedan odbacujemo!
dw=add_fr1x32(add_fr1x32(pom1,pom2),add_fr1x32(pom3,pom4))>>1;


//Cw komponenta
//cw=-3*z(0)-6*z(-1)+3*z(-2)+6*z(-3);
Wbuf_idx=circindex(Wbuf_idx,-1,4);
acc=-3<<12; //Format 3.12
if(Wbuf[Wbuf_idx]&4)
	pom=(Wbuf[Wbuf_idx]>>3)+1;
else
	pom=(Wbuf[Wbuf_idx]>>3);
pom1=(mult_fr1x32(pom,acc))>>12;

Wbuf_idx=circindex(Wbuf_idx,-1,4);
acc=-6<<12; //Format 3.12
if(Wbuf[Wbuf_idx]&4)
	pom=(Wbuf[Wbuf_idx]>>3)+1;
else
	pom=(Wbuf[Wbuf_idx]>>3);
pom2=(mult_fr1x32(pom,acc))>>12;

Wbuf_idx=circindex(Wbuf_idx,-1,4);
acc=3<<12; //Format 3.12
```

```
        if(Wbuf[Wbuf_idx]&4)
              pom=(Wbuf[Wbuf_idx]>>3)+1;
        else
              pom=(Wbuf[Wbuf_idx]>>3);
        pom3=(mult_fr1x32(pom,acc))>>12;

        Wbuf_idx=circindex(Wbuf_idx,-1,4);
        acc=6<<12;  //Format 3.12
        if(Wbuf[Wbuf_idx]&4)
              pom=(Wbuf[Wbuf_idx]>>3)+1;
        else
              pom=(Wbuf[Wbuf_idx]>>3);
        pom4=(mult_fr1x32(pom,acc))>>12;

        cw=add_fr1x32(add_fr1x32(pom1,pom2),add_fr1x32(pom3,pom4))>>1;


        //bw=3*z(-1)-3*z(-3);
        Wbuf_idx=circindex(Wbuf_idx,-1,4);
        pom1=0;
        Wbuf_idx=circindex(Wbuf_idx,-1,4);
        acc=3<<13;  //Format 2.13
        if(Wbuf[Wbuf_idx]&2)
              pom=(Wbuf[Wbuf_idx]>>2)+1;
        else
              pom=(Wbuf[Wbuf_idx]>>2);
        pom2=(mult_fr1x32(pom,acc))>>13;

        Wbuf_idx=circindex(Wbuf_idx,-1,4);
        pom3=0;
        Wbuf_idx=circindex(Wbuf_idx,-1,4);
        acc=-3<<13;  //Format 2.13
        if(Wbuf[Wbuf_idx]&2)
              pom=(Wbuf[Wbuf_idx]>>2)+1;
        else
              pom=(Wbuf[Wbuf_idx]>>2);
        pom4=(mult_fr1x32(pom,acc))>>13;

        //Množenje ima 2 predznaka, jedan odbacujemo!
        bw=add_fr1x32(add_fr1x32(pom1,pom2),add_fr1x32(pom3,pom4))>>1;


        //Aw komponenta
        //aw=0;
        aw=0;
//-------------------------------------------------------------------------------------
        //Dy komponenta
        //dy=-2*z(-1)+2*z(-2);
        Ybuf_idx=circindex(Ybuf_idx,-1,4);
        pom1=0;
        Ybuf_idx=circindex(Ybuf_idx,-1,4);
        acc=-2<<13;  //Format 2.13
        if(Ybuf[Ybuf_idx]&2)
              pom=(Ybuf[Ybuf_idx]>>2)+1;
```

```
else
      pom=(Ybuf[Ybuf_idx]>>2);
pom2=(mult_fr1x32(pom,acc))>>13;

Ybuf_idx=circindex(Ybuf_idx,-1,4);
acc=2<<13;  //Format 2.13
if(Ybuf[Ybuf_idx]&2)
      pom=(Ybuf[Ybuf_idx]>>2)+1;
else
      pom=(Ybuf[Ybuf_idx]>>2);
pom3=(mult_fr1x32(pom,acc))>>13;
Ybuf_idx=circindex(Ybuf_idx,-1,4);
pom4=0;

//Množenje ima 2 predznaka, jedan odbacujemo!
dy=add_fr1x32(add_fr1x32(pom1,pom2),add_fr1x32(pom3,pom4))>>1;


//Cy komponenta
//cy=3*z(-1)-3*z(-2);
Ybuf_idx=circindex(Ybuf_idx,-1,4);
pom1=0;
Ybuf_idx=circindex(Ybuf_idx,-1,4);
acc=3<<13;  //Format 2.13
if(Ybuf[Ybuf_idx]&2)
      pom=(Ybuf[Ybuf_idx]>>2)+1;
else
      pom=(Ybuf[Ybuf_idx]>>2);
pom2=(mult_fr1x32(pom,acc))>>13;

Ybuf_idx=circindex(Ybuf_idx,-1,4);
acc=-3<<13;  //Format 2.13
if(Ybuf[Ybuf_idx]&2)
      pom=(Ybuf[Ybuf_idx]>>2)+1;
else
      pom=(Ybuf[Ybuf_idx]>>2);
pom3=(mult_fr1x32(pom,acc))>>13;
Ybuf_idx=circindex(Ybuf_idx,-1,4);
pom4=0;

//Množenje ima 2 predznaka, jedan odbacujemo!
cy=add_fr1x32(add_fr1x32(pom1,pom2),add_fr1x32(pom3,pom4))>>1;


//By komponenta
//by=0;
by=0;


//Ay komponenta
//ay=z(-2);
Ybuf_idx=circindex(Ybuf_idx,-1,4);
Ybuf_idx=circindex(Ybuf_idx,-1,4);
Ybuf_idx=circindex(Ybuf_idx,-1,4);
```

```
        if(Ybuf[Ybuf_idx]&2)
              pom=(Ybuf[Ybuf_idx]>>2)+1;
        else
              pom=(Ybuf[Ybuf_idx]>>2); //Format 2.13
        Ybuf_idx=circindex(Ybuf_idx,-1,4);
        ay=pom;
        //-------------------------------------------------------------

              //Svi su formata 2.13 osim Cw koji je 3.12 !!!
              if(cy&1)  //Pretvori u 2.13 i zaokruži
                    cy=(cy>>1)+1;
              else
                    cy=(cy>>1);

              //Signali A, B, C, D
              d=add_fr1x32(dw,dy);
              c=add_fr1x32(cw,cy);
              b=add_fr1x32(bw,by);
              a=add_fr1x32(aw,ay);
              //Pretvaranje u 2.14
              d=d<<1;
              c=c<<2;
              b=b<<1;
              a=a<<1;
              //Spremanje A, B, C, D u cirkularne spremnike
              D[idx]=d;
              C[idx]=c;
              B[idx]=b;
              A[idx]=a;
              idx=circindex(idx,1,CIRC_SIZE);
              // --- Pomicanje signala ---
              //Pričekaj da se cirkularni spremnik napuni, inicijalno postavi pokazivač na pola
              if(!T){
                    if( idx == (CIRC_SIZE - 1)){
                          T=true;
                          idx_pass=CIRC_SIZE_HALF;
                          idx_offset= CIRC_SIZE_HALF + P;
                    }
              //Izračunaj pomaknuti signal i pošalji ga na izlaz,
              //pošalji na izlaz referentni signal;
              //Izlazni signali su veličine 32, a ne 16 bita!
              }else{
              yi=Interpoliraj(idx_pass,0);
              iChannel0LeftOut=(int)((int)yi<<8);
              yi=Interpoliraj(idx_offset,Xf);
              iChannel0RightOut=(int)((int)yi<<8);
              idx_pass=circindex(idx_pass,1,CIRC_SIZE);
              idx_offset=circindex(idx_offset,1,CIRC_SIZE);
              }
//-------------------------------------------------------------------------------------------------
}
```

Prilog I: MojeFunkcije.c

```
#include "Talkthrough.h"

// --- Generira koeficeijente za FIR filtar ---
fract16 *GenCo(int _M, const fract16 _al){
	fract16 *tmp;
	double pom;
	int i,j;
	tmp=(fract16*)malloc(M*sizeof(fract16));
	for(i=0,j=_M;i<_M;i++,j--){
		pom=pow(al,j);
		tmp[i]=(fract16)Zaokruzi(pom*pow(2,15));
	}
return tmp;
}

// --- Zaokružuje realan broj na najbliži cijeli ---
double Zaokruzi(double x){
	int pom,pom2;
	double tmp;
	tmp=x;
	pom=(int)x;
	tmp=(x-pom)*10;
	pom2=(int)tmp;
	if(pom2>=5) pom++;
	tmp=(double)pom;
return tmp;
}

// --- Računa vrijednost frakcionalnog uzorka ---
fract16 Interpoliraj(unsigned int X, fract16 dx){
	fract32 tmp;
	tmp = round_fr1x32(mult_fr1x32(D[X],dx));
	tmp = add_fr1x32(tmp,C[X]);
	tmp = round_fr1x32(mult_fr1x32(tmp,dx));
	tmp = add_fr1x32(tmp,B[X]);
	tmp = round_fr1x32(mult_fr1x32(tmp,dx));
	tmp = add_fr1x32(tmp,A[X]);
	dx=tmp;
return dx;
}

// --- Podešava željeni pomak signala i kontrolira signalizaciju ---
void GenerirajPomak(int C, int F){
	//Obradi cjelobrojni pomak
	if(C==1)idx_offset=circindex(idx_offset,1,CIRC_SIZE);
	else if(C==-1)idx_offset=circindex(idx_offset,-1,CIRC_SIZE);
```

```
        //Signalizacija cjelobrojnog pomaka
        if(idx_offset==idx_pass) *pPORTFIO_CLEAR=(3<<10);
        else if(idx_offset<idx_pass)
                {*pPORTFIO_CLEAR=(3<<10); *pPORTFIO_SET=(1<<10);}
        else if(idx_offset>idx_pass)
                {*pPORTFIO_CLEAR=(3<<10); *pPORTFIO_SET=(1<<11);}

        //Omogućava 8 frakcija uz zasićenje (korak = 0.125)
        if(F==1){
                Fr++;
                if(Fr>=8) Fr=7;
        }
        else if(F==-1){
                Fr--;
                if(Fr<=-1) Fr=0;
        }

        *pPORTFIO_CLEAR=(7<<6); //Signalizacija frakcionalnog pomaka
        *pPORTFIO_SET=(Fr<<6);
        Xf=(fract16)(Fr* 4096); //0.125 * Fr
}
```

Prilog J: Talkthrough.h

```
#ifndef __Talkthrough_DEFINED
      #define __Talkthrough_DEFINED

//-------------------------------------------------------------------------//
// Header files
//-------------------------------------------------------------------------//
#include <sys\exception.h>
#include <cdefBF537.h>

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cycle_count.h>
#include <fract.h>

//-------------------------------------------------------------------------//
// Symbolic constants
//-------------------------------------------------------------------------//
// names for registers in AD1854/AD187 converters
#define INTERNAL_ADC_L0 0
#define INTERNAL_ADC_R0 1
#define INTERNAL_DAC_L0 0
#define INTERNAL_DAC_R0 1

#define delay 0xf00

// SPORT0 word length
#define SLEN_24 0x0017

// DMA flow mode
#define FLOW_1 0x1000

//Moje konstante
#define al (sqrt(3)-2)
#define M (1+1) //Red filtra + 1
#define CIRC_SIZE 1440 //+- 15ms kasnjenje
#define CIRC_SIZE_HALF 720 //CIRC_SIZE/2

//-------------------------------------------------------------------------//
// Global variables
//-------------------------------------------------------------------------//
extern int iChannel0LeftIn;
extern int iChannel0RightIn;
extern int iChannel0LeftOut;
extern int iChannel0RightOut;
extern fract16 frChannel0LeftIn;
extern fract16 frChannel0RightIn;
extern fract16 frChannel0LeftOut;
extern fract16 frChannel0RightOut;
```

```
extern int iRxBuffer1[];
extern int iTxBuffer1[];
extern fract16 frRxBuffer1[];
extern fract16 frTxBuffer1[];
extern fract16 *D,*C,*B,*A,*Y;
extern fract16 *co,*buff;
extern int N;
extern fract16 Wbuf[];
extern fract16 Ybuf[];
extern fract16 a,b,c,d;
extern fract16 al_fr, Xf;
extern fract16 buff_idx;
extern fract16 Wbuf_idx,Ybuf_idx;
extern int idx;
extern bool T;
extern int idx_pass, idx_offset;
extern int P,Fr;

//-------------------------------------------------------------------------//
// Prototypes
//-------------------------------------------------------------------------//
// in file Initialize.c
void Init_Flags(void);
void Audio_Reset(void);
void Init_Sport0(void);
void Init_DMA(void);
void Init_Interrupts(void);
void Enable_DMA_Sport0(void);

// in file Process_data.c
void Process_Data(void);

// in file ISRs.c
EX_INTERRUPT_HANDLER(Sport0_RX_ISR);
EX_INTERRUPT_HANDLER(PORTF_IntA_ISR);

//in file MojeFunkcije.c
fract16 *GenCo(int _M, const fract16 _al);
void Simulate_interrupt(void);
double Zaokruzi(double x);
fract16 Interpoliraj(unsigned int X, fract16 dx);
void GenerirajPomak(int C, int F);
#endif //__Talkthrough_DEFINED
```