Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroničke sustave i obradbu informacija

Napredne metode digitalne obrade signala
ak.godina 2008/2009

Potiskivanje šuma korištenjem wavelet transformacije, primjena empirijskog Wienerova filtra i realizacija na maketi sa DSP-om

Seminarski rad

Bilić Ivan
Jukić Ante
Vlahović Ivan

Zagreb, siječanj 2009.

Sadržaj:

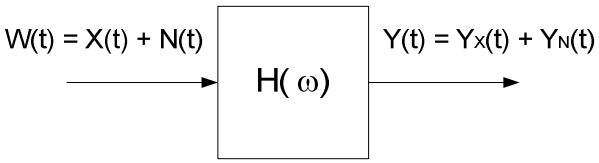
1. Uvod

Potiskivanje šuma je čest zahtjev u aplikacijama za obradu signala koji se redovito sastoje od originalnog signala uz dodane razne šumove i smetnje. Da bi se filtriranje efikasno izvršilo, često se zahtjeva poznavanje svojstava signala i šuma (npr. spektar, impulsni odziv…) Kada se potiskivanje šuma zahtjeva u stvarnom vremenu, javljaju se dodatni problemi, prvenstveno kašnjenje koje je posljedica složenosti algoritma za obradu. U ovom seminaru smo razmotrili jedan od poznatih algoritama za potiskivanje šuma u wavelet domeni, tzv. empirijski Wienerov filtar. Poznato je da je Wienerov filtar optimalan u smislu ekstrakcije signala iz šuma, uz poznavanje spektara snage istih. Kako se u praksi često obrađuju nestacionarni signali, čija se statistička svojstva mjenjaju s vremenom, potrebno je estimirati modele signala i šuma potrebne za formiranje uzoraka signala. Tu u priču ulazi wavelet transformacija, koja najčešće veliki dio signala transformira u nekoliko frekvencijskih pojaseva, dok ostali ostaju prazni, ili popunjeni šumom. *Hard tresholding* metodom (kao što je opisano u nastavku) možemo estimirati signal i šum, uz pomoć kojih, koristeći wavelet transformaciju s drugom bazom, možemo efikasno pročistiti signal od šuma. U ovom seminaru smo dani algoritam modelirali u MATLAB-u te implementirali na *Analog Devices*-ovoj *EZ-KIT Lite* maketi s ADSP-2191 DSP procesorom i AC97 kompatibilnim AD1885 *stereo codec*-om (ADC i DAC).

2. Metode uklanjanja šuma

2.2. Wienerov filtar u Fourierovoj domeni

Wienerov filtar je optimalni linearni sustav koji se koristi za ekstrakciju slučajnog signala iz aditivnog šuma. Optimalan je u smislu najmanje kvadratne pogreške.



Slika 1: Wienerov filtar u Fourierovoj domeni

Neka je korisnom slučajnom procesu *X(t)* pribrojen šum *N(t)* i pretpostavimo da su *X(t)* i *N(t)* nekorelirani i zajednički staconarni u širem smislu i da *N(t)* ima srednju vrijednost nula. Tada je optimalna prijenosna funkcija koja najbolje izdvaja signal iz šuma dana izrazom:

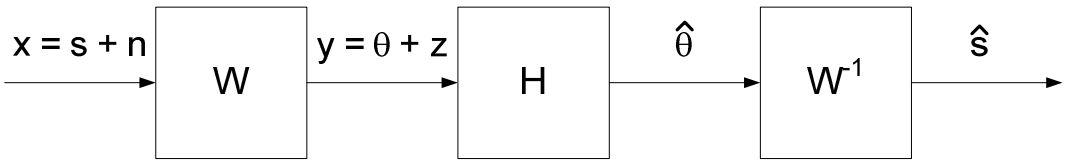
$$H_{opt}(\omega) = \frac{S_{XX}(\omega)}{S_{XX}(\omega) + S_{NN}(\omega)} \cdot e^{j\omega t_0}$$

gdje je $S_{XX}(\omega)$ spektar snage od *X(t)*, $S_{NN}(\omega)$ spektar snage šuma i t_0 kašnjenje. Da bi mogli odrediti prijenosnu funkciju Wienerovog filtra moramo poznavati spektralna svojstva ulaznog signala i šuma i samo onda je moguće optimalno uklanjanje šuma.

2.1. Metoda praga i Wienerov filtar u wavelet domeni

Metoda potiskivanja šuma pomoću praga (*threshold*) je metoda estimacije koja koristi dobra svojstva wavelet transformacije. Wavelet transformacija koncentrira energiju signala u relativno mali broj velikih koeficijenata i stoga je zanimljiva za primjenu u kompresiji i estimaciji dok se šum jednoliko raspoređuje po koeficijentima. Osnovna ideja metode potiskivanja šuma pomoću praga je da ulazni signal transformiramo u wavelet domenu, uzmemo samo najznačajnije koeficijente a ostale odbacimo i zatim napravimo inverznu transformaciju.

Blok shema sustava se nalazi na slici 2. Neka na ulazu sustava imamo nepoznati signal *s(t)* uz aditivni bijeli šum *n(t)* sa normalnom razdiobom, srednjom vrijednosti nula i varijancom σ^2. Problem potiskivanja šuma se svodi na estimaciju signala *s(t)* iz ulaznog signala $x(t) = s(t) + n(t)$.



Slika 2: Blok shema sustava za potiskivanje šuma metodom praga

Primjenom wavelet transformacije W na ulazni signal dobivamo sliku u domeni transformacije $(i) = W(x) = W(s) + W(n) = \theta(i) + z(i)$. Ova operacija preslikava tipične signale *s(t)* u mali broj koeficijenata velikog iznosa dok se *n(t)* preslikava u bijeli šum sa normalnom razdiobom i varijancom σ^2. Iz toga dolazimo do zaključka da je razuman pristup za estimaciju signala *s(t)* u wavelet domeni zanemarivanje malih koeficijenata (gdje zapravo nije signal nego samo šum) uz ostavljanje velikih koeficijenata (koji pripadaju signalu). Ova nelinearna operacija filtriranja je na slici 2 predstavljena blokom H čiji je izlaz određen jednadžbom $\hat{\theta}(i) = h(i) \cdot y(i)$. Nakon filtriranja radimo inverznu wavelet transformaciju i dobivamo estimaciju signala.

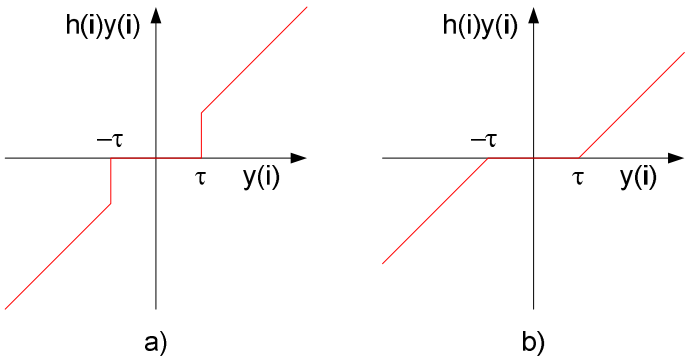
Dvije su osnovne ideje za realizaciju filtra H. Prva metoda je odbacivanje koeficijenata ispod nekog praga (*hard thresholding*) i u tom slučaju funkciju filtra možemo predstaviti izrazom

$$h(i) = h_H(i) = \begin{cases} 1, & |y(i)| > \tau \\ 0, inače. \end{cases}$$

Druga metoda je odbacivanje wavelet koeficijenata ispod nekog praga i umanjivanje ostalih koeficijenata za iznos praga (*soft thresholding*). Funkcija filtra je tada određena sljedećim izrazom

$$h(i) = h_S(i) = \begin{cases} \dfrac{sgn[y(i)] \cdot [y(i) - \tau]}{y(i)}, & |y(i)| > \tau \\ 0, inače. \end{cases}$$

Prag τ se odabire tako da se potisne većina šuma. Prva realizacija daje manju pogrešku pri rekonstrukciji signala u smislu najmanjih kvadrata. Druga je redovito bolja u primjenama, osobito kod potiskivanja šuma u slici jer nema naglih skokova u koeficijentima. Prijenosne funkcije su prikazane na slici 4.



Slika 3: Prijenosna fukcija filtra uz a) hard thresholding b) soft thresholding

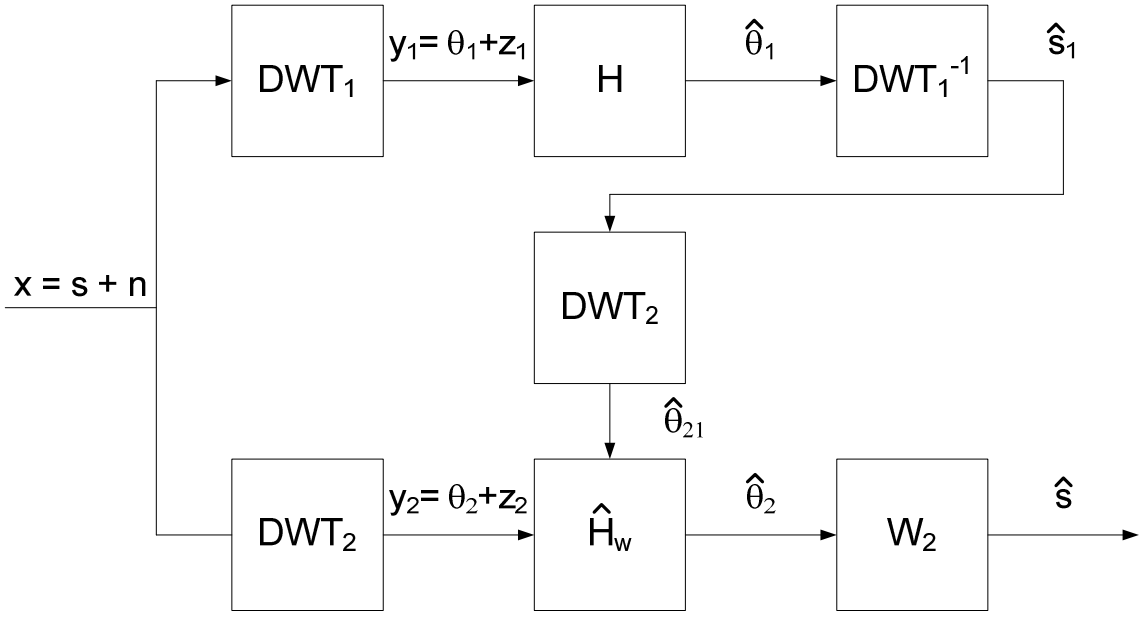
Za zadani signal konačne duljine optimalni filtar koji minimizira srednju kvadratnu pogrešku je Wienerov filtar koji je određen sa

$$h(i) = h_w(i) = \frac{\theta^2(i)}{\theta^2(i) + \sigma^2}$$

Wienerov filtar zahtijeva poznavanje statističkih svojstava šuma i slike signala u domeni transformacije i zbog toga daje optimalne rezultate.

3. Empirijski Wienerov filtar u wavelet domeni

Problem kod dizajniranja Wienerovog filtra u wavelet domeni je što moramo poznavati dekompoziciju signala *s* koji želimo estimirati i varijancu aditivnog šuma *n* koji je dodan signalu. Blokovska shema rješenja korištenog u ovom projektu je prikazana na slici 4.



Slika 4: Blok shema potiskivanja šuma korištenjem empirijskog Wienerovog filtra

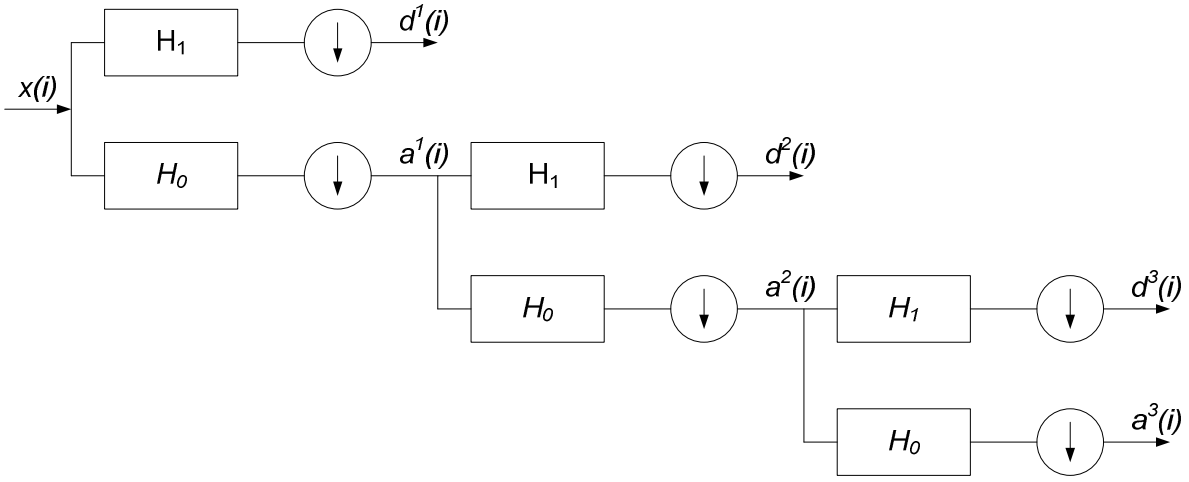
Gornja grana sustava koja se sastoji od blokova DWT_1, H i DWT_1^{-1} je zapravo sustav za potiskivanje šuma metodom praga uz *hard threshold*. Na taj način dobivamo estimaciju $\hat{s}_1$ signala koju koristimo za dizajniranje Wienerovog filtra u domeni wavelet transformacije DWT_2. Estimacija signala metodom praga daje mali broj značajnih koeficijenata $\hat{\theta}_1$ uz velik broj koeficijenata koji su jednaki nuli, tj. imamo oštre prijelaze između značajnih i neznačajnih koeficijenata. To odgovara oštrim bridovima prijenosne funkcije metode *hard threshold* sa slike 3. Zbog toga koristimo drugu wavelet transformaciju DWT_2 koja služi izglađivanju estimacije signala u domeni transformacije i rasteže koeficijente $\hat{\theta}_1$ u koeficijente $\hat{\theta}_{21}$ koji imaju manji broj nula.

Taj rezultat koristimo kao aproksimaciju wavelet koeficijenata u domeni DWT_2 i iz njega računamo koeficijente empirijskog Wienerovog filtra prema relaciji

$$h_w(i) = \frac{\hat{\theta}^{\,2}_{21}(i)}{\hat{\theta}^{\,2}_{21}(i) + \sigma^2}$$

gdje je σ^2 varijanca ulaznog šuma.

Još nam ostaje odrediti standardnu devijaciju ulaznog šuma iz koje lako dobivamo vrijednost varijance. U ovom projektu se koristi estimacija varijance šuma iz wavelet koeficijenata $y_1(i)$ i to na način da uzimamo u obzir samo koeficijente koji odgovaraju izlazu visokopropusnog filtra iz prve razine razlaganja. Pretpostavka je da je šum na ulazu bijeli sa Gaussovom (normalnom) raspodjelom amplituda uz srednju vrijednost jednaku nuli i varijancu σ^2. Takav šum se ortonormalnom wavelet transformacijom preslikava ponovno u bijeli Gaussov šum sa srednjom vrijednosti nula i istom varijancom σ^2. Analizom tipičnih signala dolazi se do zaključka da šum ima dominantan udio u koeficijentima $d^1(i)$ koji odgovaraju izlazu visokopropusnog filtra iz prve razine razlaganja i iz njih možemo estimirati standardnu devijaciju σ .



Slika 5: Izračunavanje koeficijenata DWT wavelet filtarskim stablom

Iz statistike je poznato da se standardna devijacija može estimirati prema izrazu

$$\hat{\sigma} = K \cdot MAD$$

gdje je *MAD* medijan apsolutne devijacije a *K* konstanta ovisna o pripadnoj distribuciji i za normalnu razdiobu iznosi približno 1.4826. Medijan apsolutne devijacije za niz uzoraka se definira kao medijan apsolutnog odstupanja od medijana tog istog niza, tj. kao

$$M = median(x)$$
$$MAD = median(|x - M|)$$

Kako $d^1(i)$ odgovara uzorcima bijelog šuma sa srednjom vrijednosti nula slijedi da je *M=0* i prema tome $MAD = median(|x|)$. Medijan lako možemo pronaći sortiranjem koeficijenata po apsolutnoj vrijednosti i uzimanjem apsolutne vrijednosti srednjeg člana pa varijancu procjenjujemo kao

$$\hat{\sigma}^2 = (1.4826 \cdot median(|d^1(i)|))^2$$

3.1. Simulacija u MATLAB-u

U MATLAB-u je napisan model sustava za uklanjanje šuma sa empirijskim Wienerovim filterom. Rezultate pojedinih algoritama uspoređujemo po kriteriju srednje kvadratne pogreške (*MSE, Mean Squared Error*) koja je za estimator $\hat{\theta}$ definirana izrazom

$$MSE(\hat{\theta}) = E\left(\left(\hat{\theta} - \theta\right)^2\right)$$

gdje je θ parametar koji estimiramo. Srednja kvadratna pogreška se također može izraziti i kao zbroj varijance odstupanja i kvadrata odstupanja, tj.

$$MSE\left(\hat{\theta}\right) = Var\left(\hat{\theta} - \theta\right) + \left(E\left(\hat{\theta} - \theta\right)\right)^2$$

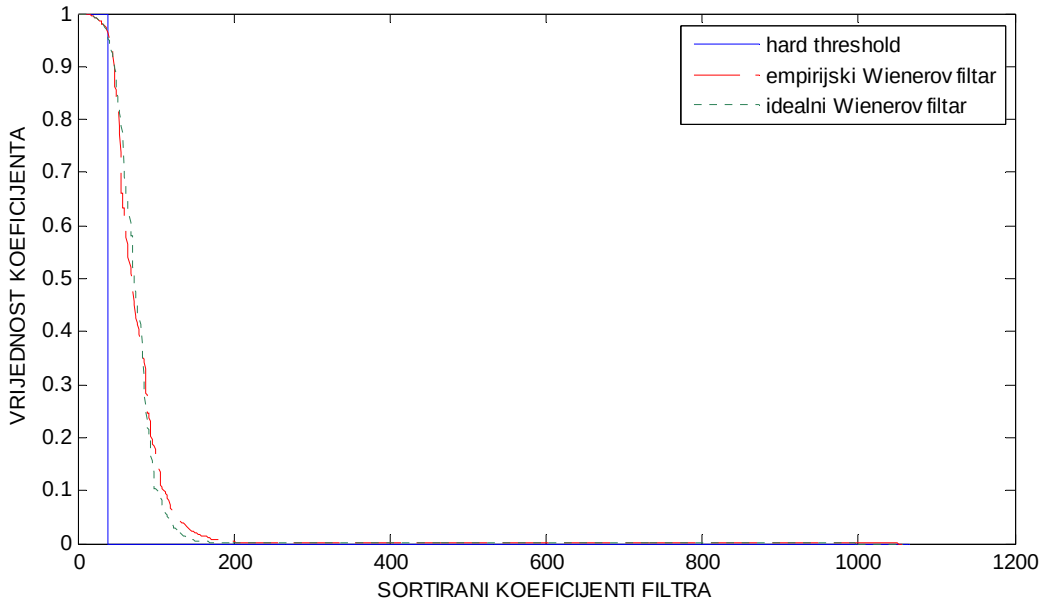
U tablici 1 se nalazi usporedba rezultata za pojedine algoritme za Dopplerov testni signal.

Tablica 1: Usporedba rezultata za Dopplerov testni signal uz $\sigma = 1$

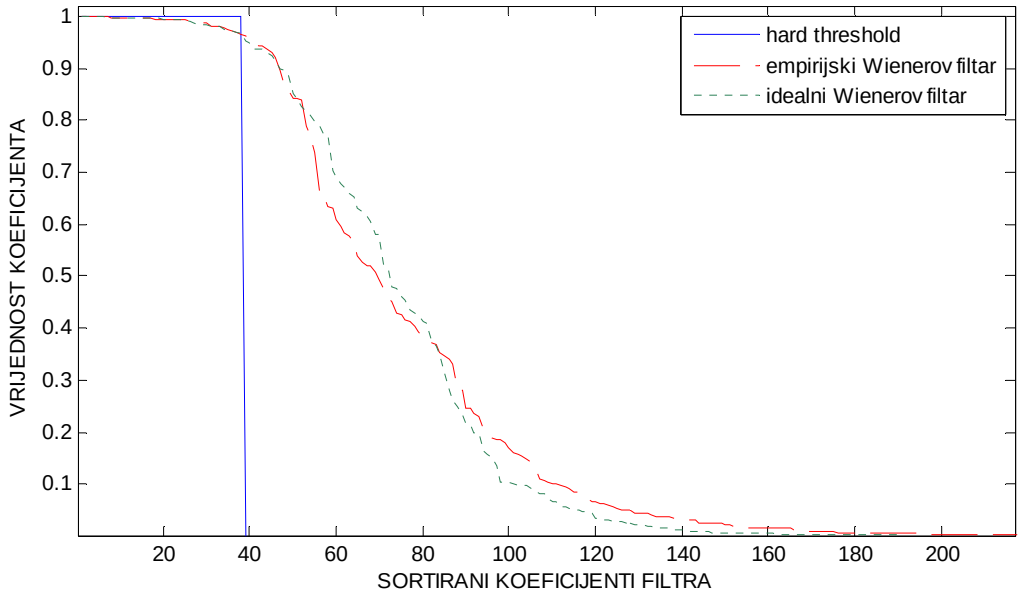
Estimator	MSE	Kvadrat odstupanja	Varijanca odstupanja
Hard threshold	0.3183	0.0016	0.3170
Empirijski Wienerov filtar	0.1678	0.0018	0.1660
Idealni Wienerov filtar	0.0974	0.0006	0.0968

Na slici 6 prikazani su iznosi sortiranih koeficijenata filtara u domeni wavelet transformacije. Vidi se da su za slučaj hard threshold vrijednosti koeficijenata filtra samo

nula ili jedan, tj. ako smatramo da je određeni koeficijent u transformaciji signala (po iznosu) značajan propuštamo ga, a inače ga zanemarujemo. Razlika između empirijskog i idealnog Wienerovog filtra se može bolje vidjeti na slici 7.

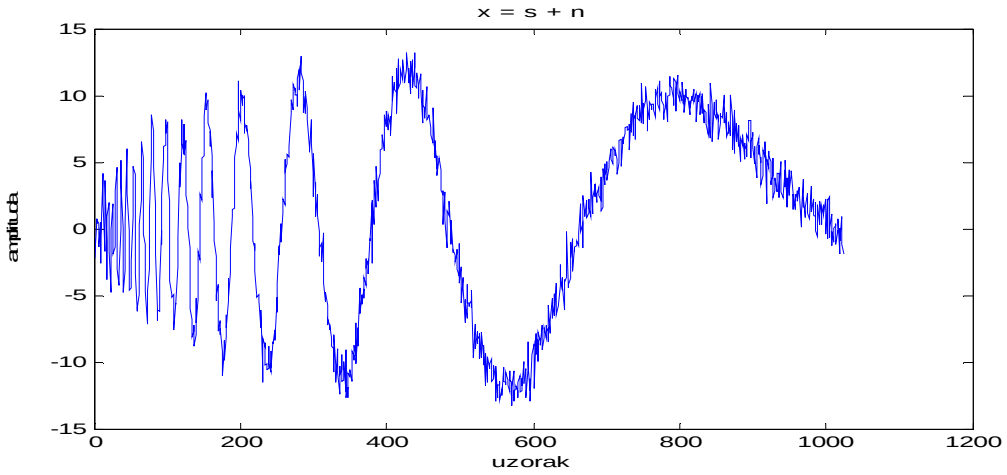


Slika 6: Prikaz filtara u domeni wavelet transformacije

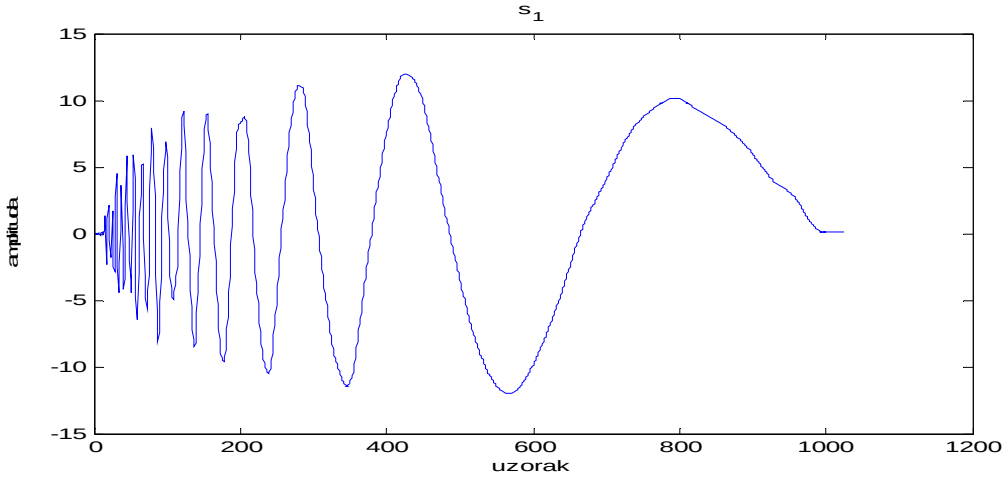


Slika 7: Detalj prikaza koeficijenata filtara

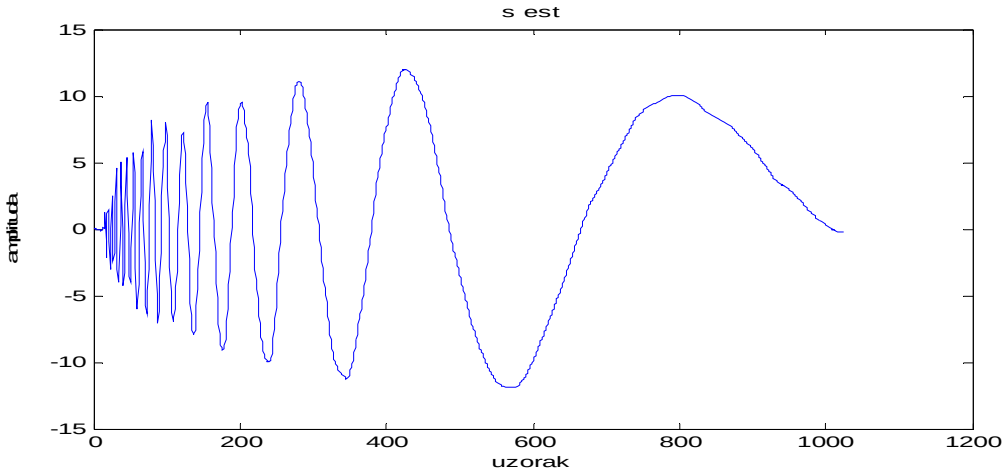
Kao baza razlaganja za transformaciju DWT1 je korišten *sym5* wavelet a za transformaciju DWT2 *sym4* wavelet. U oba slučaja je signal razložen u pet razina uz decimaciju jer će i na maketi biti realizirano filtarsko stablo sa decimacijom. Prag za hard threshold u domeni transformacije DWT1 se računa kao $K_2 \cdot \hat{\sigma}$ uz odabir $K_2 = 4$.



Slika 8: Ulazni Dopplerov testni signal sa dodanim šumom uz $\sigma = 1$



Slika 9: Estimacija uz hard threshold



Slika 10: Estimacija uz empirijski Wienerov filtar

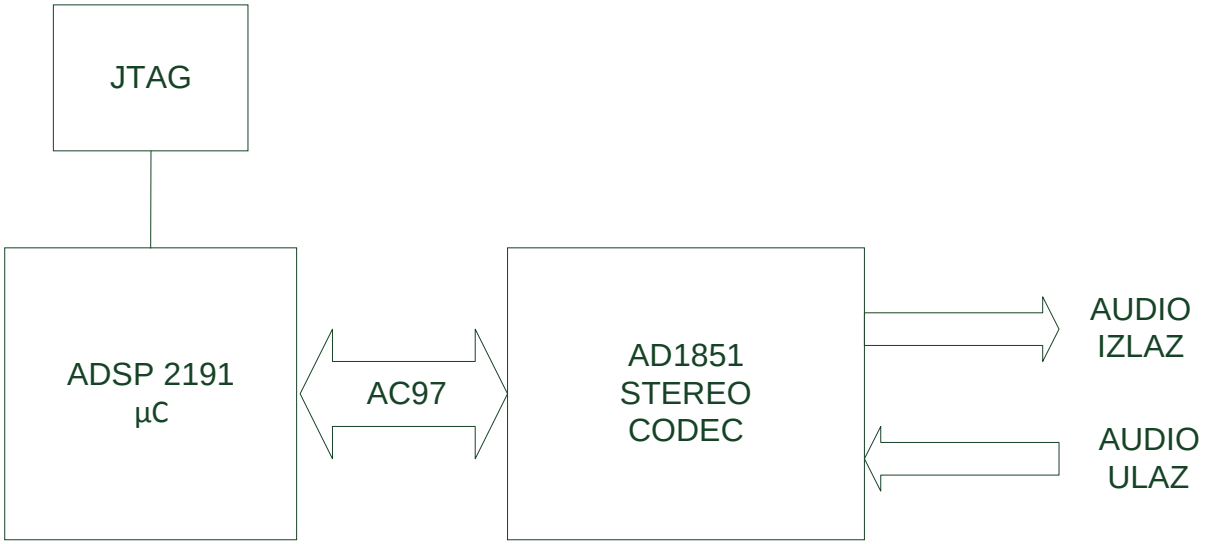
4. Implementacija na maketi sa DSP-om

Algoritam za potiskivanje šuma modeliran u MATLAB-u implementirali smo na *EZ-KIT Lite* maketi koja je testno razvojno okruženje za *Analog Devices* **ADSP-2191** DSP mikrokontroler. Obrada se vrši nad audio signalom, tim da se analogno-digitalna i digitalno analogna konverzija vrše uz pomoć **AD1885** *codec*-a koji je serijskom sabirnicom spojen s mikrokontrolerom. Testirali smo algoritam uz korištenje više baza (waveleta) i razina razlaganja, te pronašli optimum, obzirom na ograničenja koja će biti navedena u nastavku.

4.1 Opis arhitekture

Kao što je spomenuto, kao hardver za implementaciju algoritma potiskivanja šuma je iskorištena *EZ-KIT Lite* maketa, čije je srce mikrokontroler **ADSP-2191** koji je dizajniran za aplikacije digitalne obrade signala (DSP). Ključne značajke ADSP-2191 mikrokontrolera su:

- 160 milijuna instrukcija u sekundi (MIPS-a) pri frekvenciji takta 160MHz
- Hardverska 16-bitna aritmetika s fiksnim zarezom (uključujući i monoženje, te množenje i dodavanje u akumulator, osnovne operacije konvolucija)
- Programabilni DMA (*direct memory access*) kontroler s transparentnim pristupom cijelom memorijskom prostoru
- JTAG sučelje za emulaciju i debugiranje



Slika 11: Blok shema sklopa za obradu

Veza mikrokontrolera sa vanjskim svijetom je **AD1885** AC97 *stereo codec* (CoderDecoder – integrirani analogno-digitalni i digitalno-analogni pretvarač). To je čip izrađen po *Intelovom* **AC97** – *Audio Codec 1997* standardu koji je dugo vremena bio jedan od najpopularnijih sučelja za zvučne kartice, pogotovo one integrirane na matičnim pločama osobnih računala. AC97 standard propisuje fizičko sučelje između codeca i uređaja koji s njim komunicira (u našem slučaju mikrokontroler) a koje se sastoji od signala takta frekvencije 12.288 Mhz, koju generira codec, sinkronizacijski i reset signal, te dva serijska signala (ulazni i izlazni) koji osiguravaju dvosmjernu komunikaciju. Osim u slučajevima kad vanjski uređaj čita ili zapisuje

u konfiguracijske registere codec-a, ulazni i izlazni tokovi podataka su vremenskim multipleks 13 20-bitnih kanala maksimalne frekvencije 48kHz, te jednog (prvog, 16-bitnog) kontrolnog kanala. Tako jedan AC97 codec može kontrolirati 13 ulaznih i 13 izlaznih tokova (audio signala), a pomoću kontrolnog signala moguće je i njihovo kaskadiranje.

U našoj aplikaciji koristimo kanale linijskog ulaza i izlaza, za koje na maketi postoje konektori s adekvatnom analognom predobradom.

U razvoju i testiranju sustava, kao i učitavanju programa u programsku memoriju mikrokontrolera intenzivno je korišteno **JTAG** sučelje, koje se preko USB kontrolera spaja na osobno računalo.

Blok shema sustava dana je na slici 11.

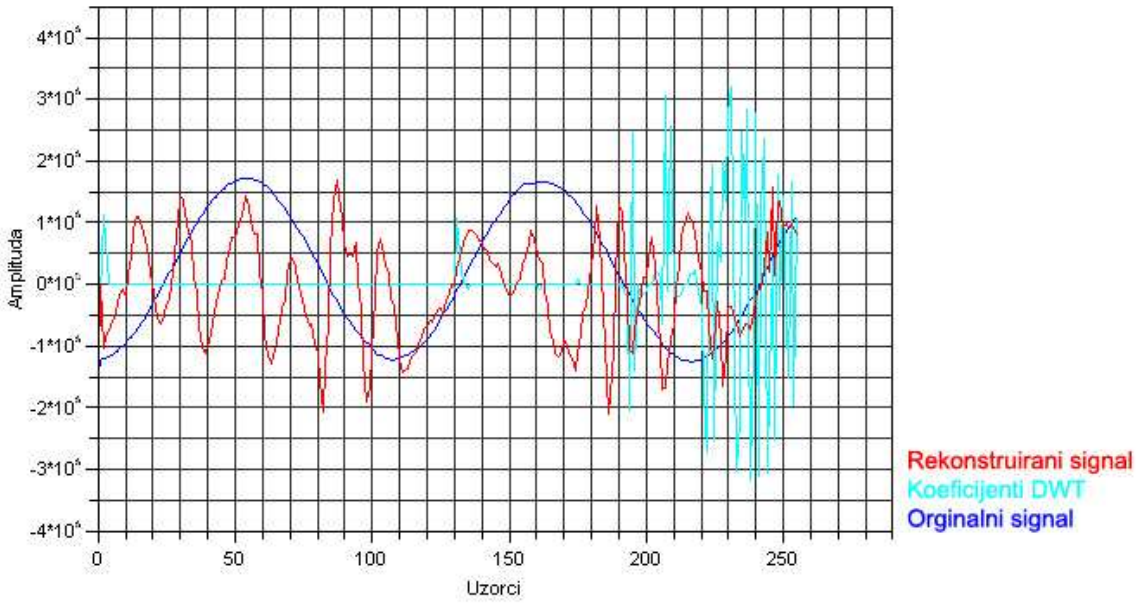
4.2 Programsko rješenje

Programska implementacija izvedena je u C jeziku, korisreći **Analog Devices VisualDSP++ 3.5** razvojno okruženje.

Implementacija u C-u, iako znatno jednostavnija od one u asembleru korištenog procesora, može stvoriti poprilične poteškoće koje je potrebno izbjeći da bi se dobilo upotrebljivo rješenje. Prvenstveno se to odnosi na brzinu izvođenja programa, tj. optimizaciju brzine izvođenja, koja je često u konfliktu s ograničenim memorijskim kapacitetom mikrokontrolera. Zato je potrebno pažljivo razraditi implementaciju algoritma, da uz ograničene resurse daje tražene rezultate. Ovo je posebno bitno kod realizacije diskretne (i inverzne) wavelet transformacije (DWT), koja je najveći potrošač memorije i vremena procesiranja. Da bi se postigla željena brzina, DWT je potrebno izvesti nekom od efikasnih realizacija. Mi smo se odlučili za polifaznu realizaciju filtarskih slogova transformacije. Stvar izbora je i računanje transformacije nad svakim primljenim uzorkom (radi se o audio signal frekvencije 48kHz) ili obrada u blokovima. Pokazalo se da obrada blokova pojednostavljuje algoritam i korištenje memorije. Nadalje, duljina bloka se mora odabrati tako da trajanje njene obrade ne bude veće od vremena uzorkovanja slijedećeg bloka, što bi izazvalo gubitke dijelova signala i neželjene nelinearne distorzije. Također, veći blokovi podataka koji se obrađuju nose i povećanu potrošnju memorije zbog većih spremnika u kojima su spremljeni uzorci i popratni međurezultati računanja.

Još jedan faktor koji je presudan za brzinu izvođenja je korištenje aritmetike koja je sklopovski implementirana u procesoru. Naime, izračuni koji barataju decimalnim brojevima mogu se vršiti s podatcima formatiranim kao brojevi s pomičnim zarezom (*floating point*) ili u frakcijskom formatu (*fixed point*). Iako je u C-u daleko elegantnije koristiti aritmetiku s pomičnim zarezom, ovaj procesor (za razliku od mnogih novijih DSP procesora) posjeduje samo instrukcije za baratanje decimalnim brojevima u frakcijskom formatu. Tako da svako baratanje *float* tipom podataka kompajlira u relativno dugi kod koji softverski (korištenjem dostupnih aritmetičko-logičkih operacija procesora) dolazi do rezultata operacije. Bitno je dakle u svim vremenski kritičnim dijelovima koda, kao što su duge iteracije množenja i zbrajanja (npr. računanje konvolucije signala s impulsnim odzivom filtara) koristiti frakcijsku aritmetiku, što se kompajleru mora eksplicitno naglasiti (npr. množenje dva broja u *fract16* formatu se ne radi korištenjem "*" operatora, već pozivom funkcije *mult()*). Problemima tu nije kraj, jer aritmetika s fiksnim zarezom ima svoja ograničenja. Naime, ugrađene funkcije koje kompajler koristi barataju brojevima u 15.1 frakcijskom formatu, što znači da se 16 bitni

podatak interpretira tako da najznačajniji bit (MSB) predstavlja predznak, a svaki slijedeći težinu (1 ili 0) s kojom frakcija broja dva na toj poziciji ulazi u iznos broja. Tako na primjer broj 0100 1000 0000 $0001_{(2)}$ predstavlja (+) $2^{-1} + 2^{-4} + 2^{-15}$. Očito je da je ovom reprezentacijom moguće prikazati samo brojeve u rasponu [-1, 1). To nam predstavlja problem prilikom dijeljenja ili množenja s brojevima većim od 1, koji su potrebni npr. prilikom računanja koeficijenata Wienerovog filtra. U takvim situacijama moramo konvertirati podatke u format s pomičnim zarezom, obaviti račun i ponovno pretvoriti u frakcijski format. Poželjno je da se ove konverzije minimiziraju na mjestima gdje je brzina kritična. Još je jedna nezgodna situacija moguća prilikom korištenja rekurzivnih ili kaskadnih realizacija poput stabla wavelet transformacije: naime, iako možda imamo realizaciju koja jamči potpunu rekonstrukciju signala na izlazu rekonstrukcijske grane, bilo gdje unutar stabla kao posljedicu zbrajanja možemo imati signale koji amplitudom višestruko nadmašuju originalni signal. Naravno, ako se dogodi da njihove amplitude premašuju interval koji frackijska reprezentacija dopušta, doći će ili do zasićenja na najveću pozitivnu odnosno negatvnu vrijednost, ili do promjene predznaka, što se očituje kao šiljak. Jasno je da distorzije poput ovih bilo gdje u stablu jamče da od potpune rekonstrukcije neće biti ništa. Primjer je dan na slici 12 na kojoj je vidljivo kako rekonstruirani signal nema veze s originalom.



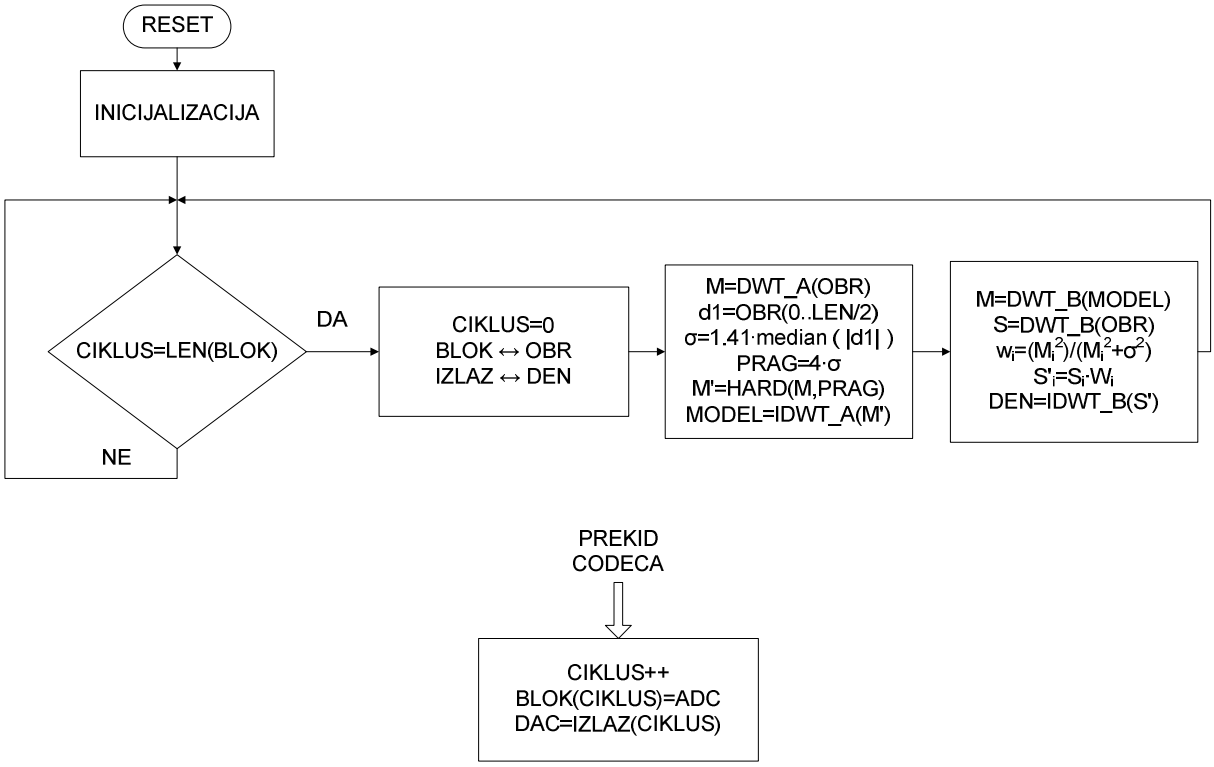
Slika 12: Distorzija usljed ograničenja dinamike

Veći broj razina razlaganja povlači veće amplitude koeficijenata na višim razinama. I ovaj problem je moguće riješiti, skaliranjem signala prije transformacije, no svako dijeljenje s 2 (pomicanje ili *shift* u desno) smanjuje rezoluciju uzoraka za 2, što dodatno kvari odnos signal-šum kojeg želimo popraviti. Stoga nije moguće očekivati da će povećanje razina dekompozicije poboljšati rezultate potiskivanja šuma kao što simulacija u MATLAB-u pokazuje.

Nakon nešto eksperimentiranja, došli smo do slijedećih parametara za implementaciju: duljina bloka podataka nad kojim se vrši obrada je 256 uzoraka, a maksimalni broj razina razlaganja 3 ili 4, uz korištenje sym3 i sym4 waveleta.

Dijagram toka programa je dan na slici 13. Prvi korak je inicijalizacija sustava koja se sastoji od slijedećega: sinkronome serijskome portu SPORT0 na kojeg je vezan codec AD1885 dajemo pravo korištenja DMA kontrolera, i podesimo se tako da prekid generira nakon što je kompletna poruka s codec-a učitana. Podatci koje primamo i šaljemo na codec se nalaze u memoriji kojoj DMA kontroler pristupa autonomno i automatizirano, pa je u prekidnoj rutini samo potrebno zapisati i pročitati podtake na tim lokacijama. Slijedi standardna inicijalizacija prekidnog sustava, gdje u prekidni vektor zapisujemo adresu prekidne rutine. Slijedi reset i inicijalizacija codeca zapisivanjem parametara u njegove kontrolne registre (uključivanje linijskog ulaza i izlaza, pojačanja, frekvencija uzorkovanja itd.)

Nakon postavljanja statičkih varijabli i inicijalizacije parametara korisničkog djela koda (potiskivanje šuma) kreće se u obradu.



Slika 13: Dijagram toka programa

U glavnoj petlji se čeka da se u ulazni spremnik ne učita broj uzoraka koji je potreban za obradu. Ovaj proces se vrši u prekidnoj rutini DMA kontrolera (codeca), gdje se ujedno inkrementira indikator učitanih uzoraka (CIKLUS na slici 13). Kad se dosegne željeni broj uzoraka, resetira se brojač ciklusa, te se zamjenjuju vrijednosti pokazivača ulaznog spremnika i spremnika za obradu. Postoje dva niza u memoriji na koje ovi pokazivači izmjenično pokazuju; time se postiže da se jedan spremnik puni dok se podatci iz drugoga obrađuju, pa nema straha od prepisivanja neobrađenih podataka novima. Isti postupak se primjenjuje nad izlaznim spremnicima.

Nad pristiglim podacima se zatim vrši prva wavelet transformacija, DWT_A. DWT je realizirana polifazno, dakle paralelnom obradom parnih i neparnih uzoraka ulaznog signala. Računa se rekurzivno, tako da se koeficijenti dobiveni u svakoj grani dekompozicije spremaju u jedan niz duljine ulaznog niza na slijedeći način: izlaz visokopropusne grane d^1 (vidi sliku 5) se pohranjuje u prvu polovicu izlaznog niza, a izlaz niskopropusne grane u

privremeni spremnik koji se koristi kao ulaz u slijedeću razinu razlaganja. Koeficijenti iz d^2 se spremaju u četvrtinu niza iza polovine, itd. do zadnje razine razlaganja koja niz popunjava koeficijentima zadnje razine koje slijedi aproksimacija na toj razini. Primjer slaganja rezultata u niz s 256 uzoraka i 3 razine razlaganja dan je na 14:

d^1	d^2	d^3	a^3
0	128	192	224

Slika 14: Format niza kojeg daje DWT (l=256 n=3)

Izlazi niskopropusnih grana se spremaju u pomočni spremnika koji služi kao ulazni niz slijedeće razine. Prilikom računanja konvolucije signala i impulsnih odziva filtara, potrebno je ovisno o redu filtra poznavati i određeni broj uzoraka prije onog kojeg trenutno obrađujemo. Da ne bi došlo do diskontinuiteta između blokova, za svaki filtar na svakoj razini postoji kružni spremnik zakašnjelih uzoraka u kojeg se trenutni uzorak sprema na način da prebriše najstariji, koji više nije potreban. To je moguće korištenjem modulo operatora na način da i-tom zakašnjelom uzorku u nizu duljine N u kojem se na poziciji i0 nalazi najnoviji uzorak pristupamo ovako: x_i = X[(i0 + i) MOD N]. Sadržaj ovih spremnika ostaje isti i nakon obrade bloka, čime se osigurava kontinuitet obrađenih signala u slijedećoj iteraciji (bloku).

Nakon što smo dobili sliku signala u wavelet domeni slijedi estimacija varijance, odnosno standardne devijacije šuma. Kako je pokazano, standardna devijacija šuma σ se vrlo dobro aproksimira medianom apsolutne devijacije, odnosno medianom apsolutne vrijednosti, obzitrom da smo pretpostavili bijeli aditivni šum sa srednjom vrijednosti nula. Da bi se median relativno brzo izračunao, potrebno je koristiti neki od algoritama koji djelomično sortiraju niz. Zato se dio niza koeficijenata (prva polovica) kopira u odvojeni niz kojem računamo apsolutnu vrijednost i pozivamo funkciju za traženje mediana temeljenu na *quickselect* algoritmu, vrlo sličnom *quicksort*-u. Nakon što je median pronađen, prag za otkidanje uzoraka *hard tresholding* metodom se određuje kao vrijednost mediana pomnožena s faktorom 3-5 (fiksiran na 4 u kodu, daje vrlo dobre rezultate). Svi uzorci wavelet koeficijenata čija je apsolutna vrijednost manja od praga se postavljaju na nulu, s izuzetkom uzoraka aproksimacije u posljednjoj razini. Naime, ovi uzorci predstavljaju dio spektra signala koji uključuje i istosmjernu vrijednost, koja može rezultirati micanjem valnog oblika iznad ili ispod nule na način da samo vrhovi signala budu s druge strane nule. Ako su te amplitude manje od praga, dolazi do rezanja signala koje uzrokuje nelinearnu distorziju signala. Istosmjernu komponentu ne smijemo ukloniti, jer imamo samo 256 uzoraka u memoriji. Kako radimo s 48 kilohertznim otipkavanjem, istosmjerna komponenta ovih 256 uzoraka može samo biti dio valnog oblika frekvencije sporije od 48000/256 = 187.5 Hz, što ulazi u čujni dio spektra, kojeg nije poželjno eliminirati. Ovako “odrezani” signal inverznom DWT vraćamo u vremensku domenu i time formiramo model signala korišten u Wienerovom filtru. Nizu dodajemo i broj uzoraka niza iz prethodne iteracije koji odgovara kašnjenju obrade u prvoj grani (DWT-IDWT) kako bi smo isto kompenzirali.

Nad nizom modela signala i originalnim signalom vršimo DWT koristeći drugu bazu i dobivene uzorke pretvaramo u format s pomičnim zarezom, kako bismo mogli izračunati koeficijente Wienerovog filtra. Nakon filtracije (množenjem transformata signala s uzorcima filtra), vraćamo uzorke u format s fiksnim zarezom i inverznom wavelet transformacijom.

Za vrijeme obrade prekidna rutina konstantno popunjava drugi blok uzoraka koji će se obrađivati u slijedećem ciklusu, te na codecu prosljeđuje uzorke iz prethodnog ciklusa obrade. Da bi se očuvao kontinuitet, bitno je da trajanje obrade bude kraće od vremena

potrebnoga za punjenje i pražnjenje svih uzoraka drugog para spremnika. Odabirom duljine bloka od 256 uzoraka, i ograničenjem razina dekompozicije na 3-4 se ovaj uvjet zadovoljava.

5. Zaključak

Simulacijom u MATLABU pokazano je da primjenom empirijskog Wienerovog filtra u odnosu na *hard treshold* metodu postižemo do 3 dB bolje potiskivanje šuma, ali na račun znatno povećane računske složenosti. Dakle, do dva puta bolje potiskivanje šuma se plaća više no dvostruko složenijim i dužim računanjem. U aplikacijama u stvarnom vremenu koje zahtijevaju potiskivanje šuma odabir metode će uvelike ovisiti o procesorskoj i memorijskoj moći i opterećenju. Dok brzom DSP procesoru kao što je ADSP-2191, na kojem se algoritam potiskivanja šuma vrti kao samostalni proces, ovakvo opterećenje ne predstavlja preveliki problem, na 8 bitnom mikrokontroleru koji osim potiskivanja šuma podatke mora spremiti, obraditi, poslati dalje u lancu obrade i sl. će se vjerojatno razmotriti neka manje procesorski zahtjevna metoda na štetu kvalitete potiskivanja. Naravno, procesorska moć čak i jeftinih mikrokontrolera opće namjene je svakim danom sve veća i veća, a njihove arhitekture sve manje nepogodne za obradu signala, što ovaj algoritam čini primamljivim za implementaciju i na takvim platformama.

6. Literatura

[1] Sandeep P. Ghael, Akbar M. Sayeed, Richard G. Baraniuk, “Improved Wavelet Denoising via Empirical Wiener Filtering”, Proceedings of SPIE, Mathematical Imaging, San Diego, July 1997

[2] Analog Devices, “DSP Microcomputer, ADSP-2191M datasheet”, Analog Devices Inc.

[3] Analog Devices, “ADSP-2191 EZ-KIT Lite Evaluation System Manual”, Analog Devices Inc., October 2003.

[4] Intel Corporation, “Audio Codec ‘97”, Intel Corporation, September 2000.

[5] Analog Devices Inc., “AC’97 SoundMAX Codec AD1885”, Analog Devices Inc., 2000.

[6] Damir Seršić, “Napredne metode digitalne obrade signala”, materijali za predavanja, 2008.

[7] Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997

7. Prilog: Skripta za simulaciju u MATLAB-u

```
% brisanje svih varijabli u workspaceu
clear
clc

% propuštanje zašumljenog signala kroz sustav

% generiranje testnog signala
% s ... originalni signal
% x ... zašumljeni signal
% n ... šum
s = wnoise(4, 10, 7);


% ucitavanje wav datoteke sa zvukom gitare
% [s, fs, nbits] = wavread('gitara.wav');
% uzimamo samo jedan kanal
% s = (s(1:end,1))';

% generiranje bijelog šuma koji dodajemo signalu
mu = 0;
sigma = 2;
duljina = length(s);
n = normrnd(mu, sigma, 1, duljina);

% zašumljeni signal na ulazu sustava
x = s + n;

% prikaz generiranih signala
figure('Name','Prikaz signala na ulazu')
subplot(3,1,1), plot(s);
title('s'); xlabel('uzorak'); ylabel('amplituda');
subplot(3,1,2), plot(n);
title('n'); xlabel('uzorak'); ylabel('amplituda');
subplot(3,1,3), plot(x);
title('x = s + n'); xlabel('uzorak'); ylabel('amplituda');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prva wavelet transformacija ulaznog signala i obrada sa hard thresholdom
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% zadajemo wavelet za prvu transformaciju
W_1 = 'sym5';

% dekompozicija u 5 razina
[y_1_C, y_1_L] = wavedec(x, 5, W_1);
y_1_a5 = appcoef(y_1_C, y_1_L, W_1, 5);
y_1_d5 = detcoef(y_1_C, y_1_L, 5);
y_1_d4 = detcoef(y_1_C, y_1_L, 4);
y_1_d3 = detcoef(y_1_C, y_1_L, 3);
y_1_d2 = detcoef(y_1_C, y_1_L, 2);
y_1_d1 = detcoef(y_1_C, y_1_L, 1);

% prikaz dekompozicije y_1
figure('Name','Prikaz dekompozicije y_1')
subplot(6,1,1); plot(y_1_a5); title('y_1 a5');
subplot(6,1,2); plot(y_1_d5); title('y_1 d5');
subplot(6,1,3); plot(y_1_d4); title('y_1 d4');
```

```
subplot(6,1,4); plot(y_1_d3); title('y_1 d3');
subplot(6,1,5); plot(y_1_d2); title('y_1 d2');
subplot(6,1,6); plot(y_1_d1); title('y_1 d1');

% procjena sigme iz medijana
% procjenjujemo je kao: 1.4826*medijan_apsolutne_vrijednosti
% ( vidi funkciju wnoisest )

K1 = 1.4826;
sigma_est = K1*median(abs(y_1_d1));

% idealna procjena sigme za ovaj signal je iz koeficijenata d1, tj. detalja
% u prvoj razini razlaganja jer u tom pojasu imamo samo šum

% granica za threshold
% K2 odabiremo između 2 i 5
K2 = 4;
G = K2*sigma_est;

% obrada u domeni transformacije
% hard threshold sa granicom G
theta_1_a5 = wthresh(y_1_a5,'h',G);
theta_1_d5 = wthresh(y_1_d5,'h',G);
theta_1_d4 = wthresh(y_1_d4,'h',G);
theta_1_d3 = wthresh(y_1_d3,'h',G);
theta_1_d2 = wthresh(y_1_d2,'h',G);
theta_1_d1 = wthresh(y_1_d1,'h',G);


% prikaz koeficijenata nakon hard threshold-a
figure('Name','Prikaz koeficijenata theta_1')
subplot(6,1,1); plot(theta_1_a5); title('theta_1 a5');
subplot(6,1,2); plot(theta_1_d5); title('theta_1 d5');
subplot(6,1,3); plot(theta_1_d4); title('theta_1 d4');
subplot(6,1,4); plot(theta_1_d3); title('theta_1 d3');
subplot(6,1,5); plot(theta_1_d2); title('theta_1 d2');
subplot(6,1,6); plot(theta_1_d1); title('theta_1 d1');

theta_1_C = [theta_1_a5   theta_1_d5   theta_1_d4   theta_1_d3   theta_1_d2
theta_1_d1];
theta_1_L = y_1_L;

% inverzna transformacija
s_1 = waverec(theta_1_C, theta_1_L, W_1);

% prikaz rekonstruiranog signala nakon hard thresholda
figure('Name','Prikaz signala s_1')
subplot(4,1,1), plot(s);
title('s'); xlabel('uzorak'); ylabel('amplituda');
subplot(4,1,2), plot(x);
title('x = s + n'); xlabel('uzorak'); ylabel('amplituda');
subplot(4,1,3), plot(s_1);
title('s_1'); xlabel('uzorak'); ylabel('amplituda');
subplot(4,1,4), plot(s_1-s);
title('s - s1'); xlabel('uzorak'); ylabel('amplituda');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% druga wavelet transformacija ulaznog signala i obrada sa empirijski
% dobivenim Wienerovim filtrom
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% druga wavelet transformacija ulaznog signala i izlaza iz prve wavelet
% transformacije

% zadajemo wavelet za drugu wavelet transformaciju W2
W_2 = 'sym4';

% dekompozicija u 5 razina wavelet transformacijom W2

% ulazni signal
[y_2_C, y_2_L] = wavedec(x, 5, W_2);
y_2_a5 = appcoef(y_2_C, y_2_L, W_2, 5);
y_2_d5 = detcoef(y_2_C, y_2_L, 5);
y_2_d4 = detcoef(y_2_C, y_2_L, 4);
y_2_d3 = detcoef(y_2_C, y_2_L, 3);
y_2_d2 = detcoef(y_2_C, y_2_L, 2);
y_2_d1 = detcoef(y_2_C, y_2_L, 1);

% aproksimacija ulaznog signala dobivena hard threshold-om
[theta_21_C, theta_21_L] = wavedec(s_1, 5, W_2);
theta_21_a5 = appcoef(theta_21_C, theta_21_L, W_2, 5);
theta_21_d5 = detcoef(theta_21_C, theta_21_L, 5);
theta_21_d4 = detcoef(theta_21_C, theta_21_L, 4);
theta_21_d3 = detcoef(theta_21_C, theta_21_L, 3);
theta_21_d2 = detcoef(theta_21_C, theta_21_L, 2);
theta_21_d1 = detcoef(theta_21_C, theta_21_L, 1);

% prikaz dekompozicije
figure('Name','Prikaz dekompozicije y_2 i theta_21')
subplot(6,2,1); plot(y_2_a5); title('y_2 a5');
subplot(6,2,3); plot(y_2_d5); title('y_2 d5');
subplot(6,2,5); plot(y_2_d4); title('y_2 d4');
subplot(6,2,7); plot(y_2_d3); title('y_2 d3');
subplot(6,2,9); plot(y_2_d2); title('y_2 d2');
subplot(6,2,11); plot(y_2_d1); title('y_2 d1');

subplot(6,2,2); plot(theta_21_a5); title('\theta_{21} a5');
subplot(6,2,4); plot(theta_21_d5); title('\theta_{21} d5');
subplot(6,2,6); plot(theta_21_d4); title('\theta_{21} d4');
subplot(6,2,8); plot(theta_21_d3); title('\theta_{21} d3');
subplot(6,2,10); plot(theta_21_d2); title('\theta_{21} d2');
subplot(6,2,12); plot(theta_21_d1); title('\theta_{21} d1');

% konstruiranje Wienerovog filtra
h_w_a5 = theta_21_a5.^2 ./ ( theta_21_a5.^2 + sigma_est^2 );
h_w_d5 = theta_21_d5.^2 ./ ( theta_21_d5.^2 + sigma_est^2 );
h_w_d4 = theta_21_d4.^2 ./ ( theta_21_d4.^2 + sigma_est^2 );
h_w_d3 = theta_21_d3.^2 ./ ( theta_21_d3.^2 + sigma_est^2 );
h_w_d2 = theta_21_d2.^2 ./ ( theta_21_d2.^2 + sigma_est^2 );
h_w_d1 = theta_21_d1.^2 ./ ( theta_21_d1.^2 + sigma_est^2 );

% prikaz Wienerovog filtra u domeni transformacije W2
figure('Name','Prikaz empirijskog Wienerovog filtra')
subplot(6,1,1); plot(h_w_a5); title('h_w a5');
subplot(6,1,2); plot(h_w_d5); title('h_w d5');
subplot(6,1,3); plot(h_w_d4); title('h_w d4');
subplot(6,1,4); plot(h_w_d3); title('h_w d3');
subplot(6,1,5); plot(h_w_d2); title('h_w d2');
subplot(6,1,6); plot(h_w_d1); title('h_w d1');
```

```
% filtracija Wienerovim filtrom u domeni transformacije W2
theta_2_a5 = y_2_a5.*h_w_a5;
theta_2_d5 = y_2_d5.*h_w_d5;
theta_2_d4 = y_2_d4.*h_w_d4;
theta_2_d3 = y_2_d3.*h_w_d3;
theta_2_d2 = y_2_d2.*h_w_d2;
theta_2_d1 = y_2_d1.*h_w_d1;

% inverzna transformacija W2

theta_2_C = [theta_2_a5   theta_2_d5   theta_2_d4   theta_2_d3   theta_2_d2
theta_2_d1];
theta_2_L = y_2_L;

% inverzna transformacija
s_est = waverec(theta_2_C, theta_2_L, W_2);

% prikaz rekonstruiranog signala nakon obrade Wienerovim filtrom
figure('Name','Prikaz signala s_est')
subplot(4,1,1), plot(s);
title('s'); xlabel('uzorak'); ylabel('amplituda');
subplot(4,1,2), plot(x);
title('x = s + n'); xlabel('uzorak'); ylabel('amplituda');
subplot(4,1,3), plot(s_est);
title('s est'); xlabel('uzorak'); ylabel('amplituda');
subplot(4,1,4), plot(s_est-s);
title('s est - s'); xlabel('uzorak'); ylabel('amplituda');
```