SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4531

IZVEDBA ALGORITMA ZA PRECIZNO MJERENJE FAZNE RAZLIKE HARMONIČKOG ULTRAZVUČNOG SIGNALA

Denis Malić

Zagreb, lipanj 2016.

Sadržaj

1. Uvod 3
2. Više-harmonijski signali 4
3. Implementacija u Matlabu 6
3.1 Goerzelov algoritam, DFT, STFT 9
4. Implementacija u C-u 14
4.1 Frakcionalna aritmetika 14
4.1.1 Gubitak preciznosti i preljev 14
4.2 Opis pojedinih dijelova 15
4.2.1 Arkus tangens funkcija 18
4.3 Poboljšanja i optimizacije 20
5. Razvojni sustav eZdsp TMS320VC5505 25
5.1 Procesor TMS320VC5505 i periferni sklopovi 25
6. Zaključak 28
7. Sažetak 29
8. Literatura 27

1. Uvod

Anemometar je uređaj koji mjeri brzinu i smjer vjetra. Mjerenje značajki vjetra je vezano za industriju proizvodnje električne energije kroz korištenje vjetro-elektrana i njihovo planiranje. Također, mjerenje je važno i za praćenje prirodnih pojava u meteorologiji. Postoji nekoliko tipova anemometara kao što su mehanički, ultrazvučni i termički.

Anemometar je jedan od najčešćih uređaja korištenih u vremenskim postajama za mjerenje brzine i smjera vjetra. Najstarija njegova izvedba je u obliku zakretnih lopatica na temelju čijih se brzina zakretanja može odrediti brzina vjetra koja je proporcionalna rotacijskoj brzini osovine. Ali, takve su izvedbe neučinkovite i neprecizne jer ih ograničavaju grube mehaničke nesavršenosti kao trenje u osovini, što znači da ne mogu detektirati brzine vjetra malog iznosa. Također takav uređaj zahtijeva često održavanje pokretnih dijelova i neprikladan je za oštre vremenske uvijete. Ta značajna ograničenja rješava primjena ultrazvučnog anemometra.

Za razliku od toga, ultrazvučni anemometar radi u svim vremenskim uvjetima. On ima širi raspon detekcije brzine vjetra. Može se reći da je ultrazvučni anemometar napredniji uređaj za mjerenje brzine i smjera vjetra. Ultrazvučni anemometar je vrsta zvučnog anemometra. Razlog za odabir ultrazvuka prema zvuku iz čujnog područja je njegova mala valna duljina, odnosno velika frekvencija. Zvuk visoke frekvencije ima manje šumova u prirodi, suprotno tome zvuk čujnog područja bi vjerojatno imao puno šuma kad bismo razgovarali pored anemometra ili koristili ventilator. Isto tako manja valna duljina poboljšava razlučivost rezultata. Anemometar radi na principu mjerenja vremena propagacije signala između fiksnih točaka na temelju čega se određuje brzina vjetra. U dvodimenzionalnom, slučaju kojeg obrađuje ovaj rad, radi se o više putova u istoj ravnini koji služe za određivanje smjera vjetra. Postoje razne mogućnosti izvedbi, a dvodimenzionalni anemometri su najzastupljeniji u meteorološkim postajama.

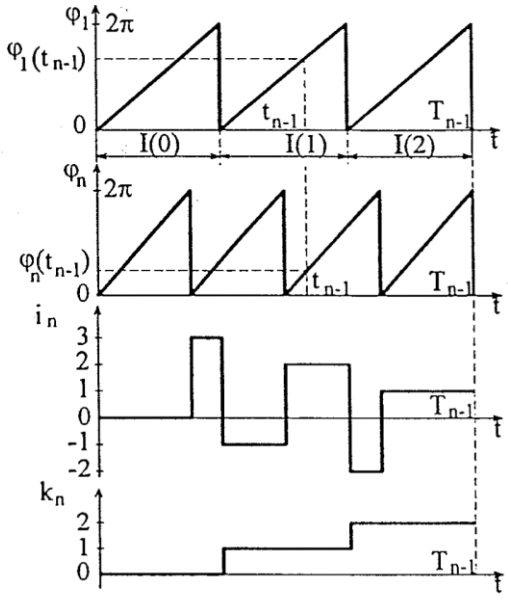
Sustav je temeljen na četiri ultrazvučna pretvarača koji su raspoređeni na uglovima romba pogodne dimenzije čije su dijagonale orijentirane prema glavnim geografskim osima (sjever, jug , istok i zapad). U okviru rada napisan je program za eZdsp TMS320VC5505 koji obrađuje dva skupa podataka sa dva prijemnika. Pojedini skup se sastoji od signala s predajnika koji je na nasuprotnom uglu romba (prijemnik i predajnik su vezani dijagonalom romba) i signala koji je odaslan sa prijemniku susjednog ugla romba.

2. Više-harmonijski signali

Na prethodno opisan način moguće je ostvariti osam konfiguracija mjerenja tako da se mijenja položaj dvaju odašiljača, na susjednim uglovima, prema prijemnicima (pretvarači, imaju dvostruku ulogu) i međusobna zamjena položaja dvaju odašiljača. Takav pristup omogućuje veću točnost određivanja smjera vjetra i ublažava moguću krivu interpretaciju brzine vjetra koja bi mogla nastati da se provodi samo jedna konfiguracija mjerenja uslijed njegovog vremenski kratkotrajnog odstupanja. Promjena konfiguracije mjerenja se sa strane programa za eZdsp očituje u različitim primljenim skupovima podataka.

Odaslani signali (dva različita signala) se sastoje od nekoliko sinusnih komponenti sa zajedničkim periodom i cjelobrojnim odnosom osnovnih frekvencija. Razlog tome je taj što se kod osnovne sinusoide ne može odrediti kašnjenje na temelju pomaka faze zbog prevelike frekvencije zbog koje faza, u našim vremenski predviđenim okvirima, nekoliko puta prekoračuje puni krug. Zato se pristupa slanju više-harmonijskih signala točno određenih višekratnika zajedničkog perioda za koje pretpostavljamo da se neće pomaknuti za iznos veći od zajedničkog perioda u ovdje opisanom mjerenju (s obzirom na odabranu udaljenost pretvarača).

U algoritmu su prijemnici označeni tako da se na jednom od njih prima signal S1 direktno, S2 dijagonalno, a na drugom S2 direktno i S1 dijagonalno. Takve oznake su u skladu sa svih osam konfiguracija mjerenja, a sa promjenom konfiguracije će iste oznake prijemnika odgovarati različitim prijemnicima u stvarnosti, što će se programski vidjeti samo kao različiti ulazni skupovi podataka.



Slika 1.

Na slici 1. je primjer određivanja vremena kašnjenja tn-1 signala koji se sastoji od dvije harmonijske komponente s tri i četiri perioda na zajedničkom periodu.

$$t_{n-1} = \frac{T_{n-1}}{p_n} k_n + \frac{T_{n-1}}{p_n} \cdot \frac{\varphi_1}{2\pi}, \qquad \text{where}$$

$$k_n = \begin{cases} -i_n \ldots\ldots i_n \le 0 \\ p_n - i_n \ldots i_n > 0 \end{cases} \qquad \text{and} \qquad i_n = \frac{1}{2\pi}((p_n + 1)\varphi_1 - p_n\varphi_n) .$$

Ova tvrdnja je ispravna dok se radi o idealnom mediju. U stvarnom mediju faze se mogu malo promijeniti zbog utjecaja medija. Zato se kao pobudni signali koriste signali sa tri harmonijske komponente. Njihove su frekvencije prema frekvenciji ukupnog signala točno određene i glase za prvi signal S1: 100, 101 i 104, a za S2: 98, 99, 105 perioda na zajedničkom periodu. Signali su uzorkovani s 242 točke na zajedničkom periodu, frekvencijom uzorkovanja 96000 uzoraka u sekundi.

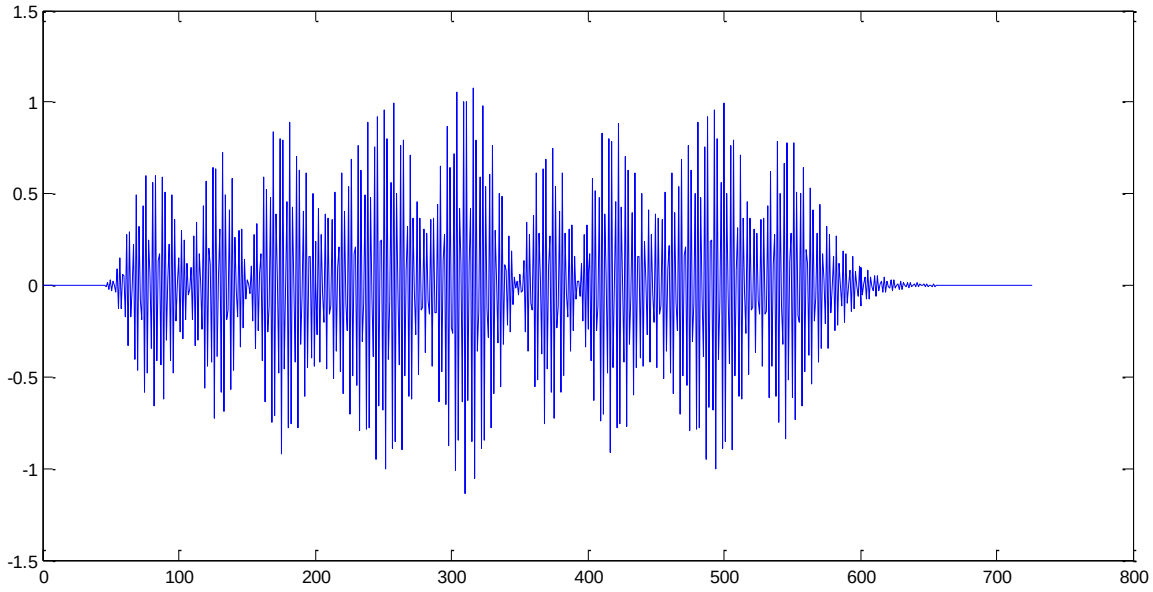
3. Implementacija u Matlabu

U Matlabu je uključena simulacija pobudnih signala za svih osam konfiguracija mjerenja. Svaka od konfiguracija je vezana za svoj slučajno odabrani skup faza od 0° do 360°. Pojedini skup faza je dodjeljen mjernom i kalibracijskom prijemnom signalu S1 i S2 u direktnom i dijagonalnom smjeru. Za svaku od konfiguracija su generirani i prijemni kalibracijski signali, koji su zakašnjeli različito na osima različite duljine i prijemni mjerni signali koji su dodatno različito, u usporedbi sa kalibracijskim, zakašnjeli zbog prisutnosti vjetra. Smjer vjetra je određen u dvije dimenzije i odabran na početku koda. Osnovni koordinatni sustav ima osi paralelne dijagonalama romba i 0° je na sjevernom vrhu romba, dok kut pozitivno raste prema istočnom vrhu. To je u skladu sa uobičajenim načinom izražavanja smjera vjetra. Određivanje brzine vjetra na x osi riješeno je tako da se amplituda brzine množi sa sinusom kuta kuta upada vjetra u koordinatni sustava i sve se pomnoži sa minus jedan zbog toga što je kut upada suprotan smjeru vektora vjetra. Slično se napravi i određivanjem brzine vjetra po y osi. Udaljenost pretvarača je 20 cm po x osi i 21 cm po y osi. Brzina zvuka u suhom zraku v=331.3*sqrt(T/273.15), gdje je T temperatura u kelvinima. Vjetar utječe na brzinu širenja zvuka. Zvuk je longitudinalan val. Zatim se za brzinu zvuka u zraku uzima u obzir utjecaj bočnog vjetra. Na x osi to je okomit vjetar na tu os, tj. y os. Pri traženju projekcije brzine vjetra u dijagonalnim smjerovima, moramo uzeti u obzir da su uređaji razmaknuti različito po okomitim osima, pa lokalne koordinatne sustave ne treba dobiti rotacijom za 45°, nego većim kutom pravokutnog trokuta φ kome su stranice razmaci 20 i 21. Za NW (SE) smjeru os x treba rotirati za φ u smjeru obratno kazaljke na satu pa lokalni sustav ima E´ i W´ u smjeru koji povezuje dva prijemnika (na N i W). Također uz pomak kuta za komponentnu brzine u lokalnom sustavu određujemo njenu okomitu komponentu. Isto radimo i za drugu dijagonalu, uz pomak koordinatnog sustava u drugom smjeru za φ, pa lokalni sustav ima x os koja gleda u WS (NE) smjeru. Zatim odredimo komponentu u tom smjeru i njenu okomitu komponentu. Nakon vektorskog zbroja brzine vjetra po svakoj osi jednostavno dobivamo vrijeme propagacije signala na svim osima uz poznate udaljenosti među prijemnicima. Ta vremena propagacije izražavamo u broju uzoraka.

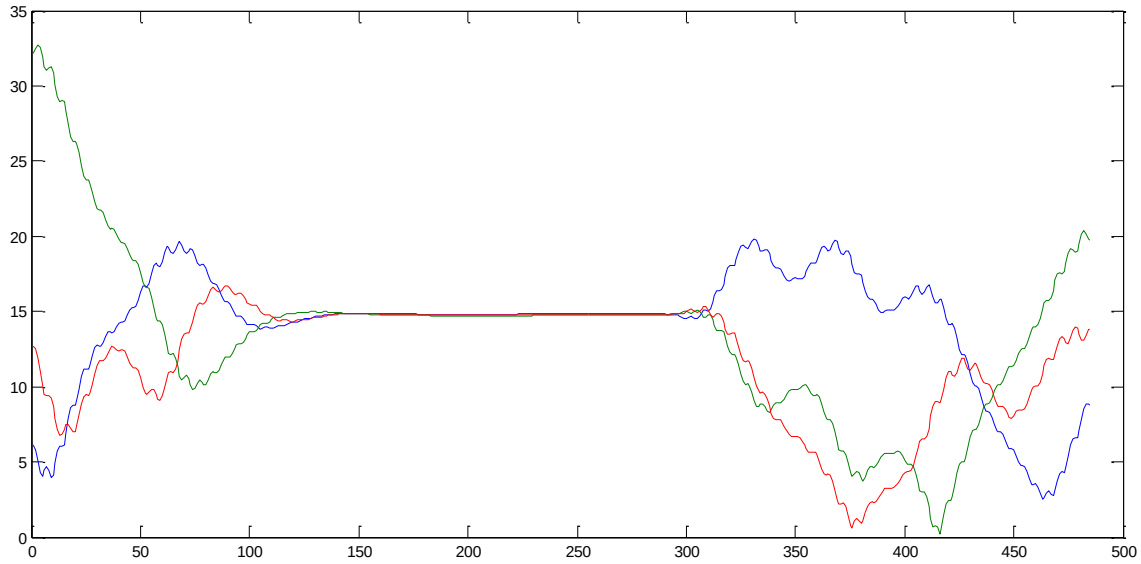
Nadalje, u Matlabu je filtar (vezan uz frekvencijsku karakteristiku pretvarača koji se koriste) koji simulira stvarno stanje prijemnog signala i izveden je u dva stupnja. Nakon prolaska kroz taj filtar komponente više frekvencije signala S1 i S2 registrirane su sa manjom amplitudom zbog većeg gušenja. Konačno, signal S1 ima komponente čije su pojedinačne amplitude ≈0.202, a komponente signala S2 su amplitude ≈0.122. Pri pojedinoj konfiguraciji mjerenja, na prvom pretvaraču se dohvaća signal S1 po dužoj, i S2 po kraćoj putanji sa dvostrukom amplitudom, a na drugom obratno, iz čega slijedi da se u najgorem slučaju na prvom pretvaraču dohvaća signal amplitude manje od 1.4, a na drugom pretvaraču amplitude manje od 1.6. S obzirom na to da će se rješenje realizirati frakcionalnom aritmetikom, ulazni signali će prije odrade na *DSP* modulu trebati biti skalirani sa 1.6. Osim

filtriranja signali su, ovisno o odabranoj konfiguraciji*, izjednačeni nulom na početnim* uzorcima čime je simulirano njihovo fizikalno kašnjenje i zbrojen im je šum kao svojstvo stvarnog sustava. Takav pristup znači da kod računanja vremenski kratkotrajne Fourieove transformacije ne možemo uzimati u obzir rješenja prozora koja uključuju nestabilni dio prijemnog signala. Također je signal na kraju niza od 726 uzoraka prigušen ponajprije zbog iščezavnja signala kraće putanje. Za sve slučajeve, u nizu pomaknutih prozora, prvi značajan smatramo onaj pomaknut za 108 uzoraka, a posljednji za 295 uzoraka.

Nakon generiranja prijemnih kalibracijskih signala na objema pretvaračima, gore navedene tvrdnje se mogu provjeriti promatrajući rezultate STFT (*short time Fourier transformation)*. Pojačanje komponenata u signalu nakon DFT (*discrete Fourier transformation*) je N/2, što vrijedi i za prozore STFT.

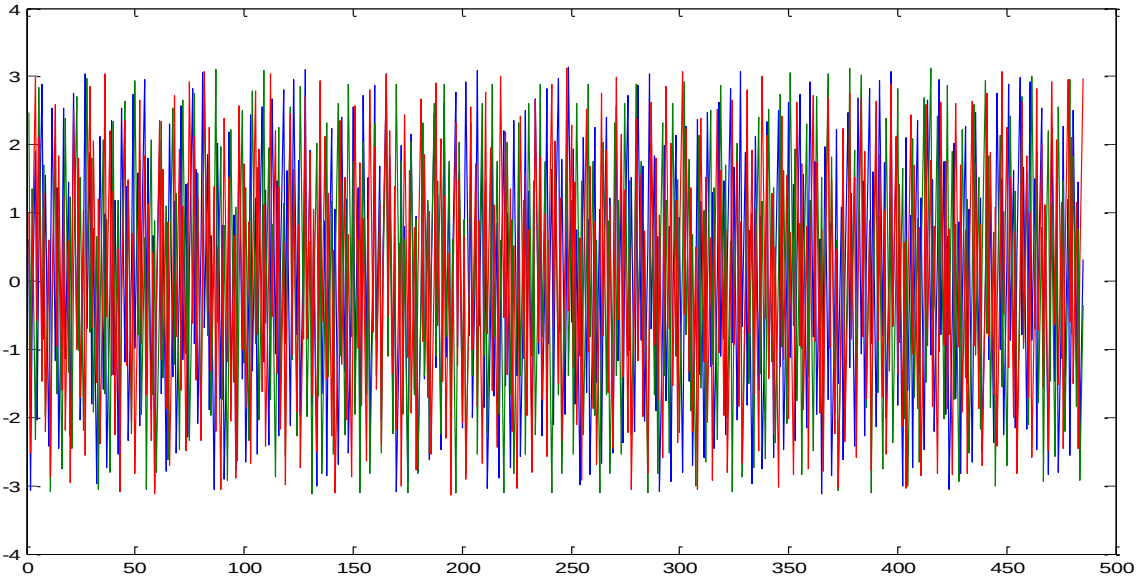


Slika 2. Prijemni diskretni neskalirani kalibracijski signal na drugom pretvaraču perioda N=242 uzorka i duljine 726 uzoraka



Slika 3. Amplitude komponenata kalibracijskog signala S2 na drugom pretvaraču (rješenje STFT za 485 prozora)

Bitno je primijetiti da se pri računanju STFT, tj. DFT, sa pomicanjem prozora, rezultati vide vremenski pomaknuti za pomak prozora. Pri svakom pomaku prozora duljine periodičnosti signala faza pojedine komponente se također povećava ovisno o frekvenciji te komponente.



Slika 3. Faza triju komponenata signala S2 drugog pretvarača za sve prozore

Signali su periodični sa N=242 uzorka, a diskretne kružne frekvencije za signale S1 i S2 su: $w_{1_i} = k_{1_i}\frac{2\pi}{N}$ i $w_{2_i} = k_{2_i}\frac{2\pi}{N}$, gdje su koeficijenti cijeli brojevi navedeni u poglavlju 1.1. Diskretna komponenta najmanje frekvencije u signalima je, također, periodična sa 242 uzorka. U kontinuiranom vremenu vrijedi da je $f = \frac{f_s}{N}$. Gdje je f

frekvencija kontinuiranog signala, a f_s frekvencija uzorkovanja koja iznosi 96000 uzoraka u sekundi. Iz toga slijedi da su pojedine kontinuirane kružne frekvencije komponenata signala S1 i S2 $\omega_{1_i} = k_{1_i}\frac{f_s}{N}$ i $\omega_{2_i} = k_{2_i}\frac{f_s}{N}$. Cjelobrojni koeficijenti znače da ne dolazi do preljeva pri određivanju DFT-a, nego da su komponente cjelobrojni višekratnici zajedničke frekvencije.

Implementacija u Matlabu je podijeljena na dvije cjeline. To su kalibracijski i mjerni postupak. Ta dva postupka vežu slučajne faze dodijeljene komponentama signala i oni se provode da bi se uklonio utjecaj tih slučajnih faza na mjerenje jer bez provedbe kalibracijskog postupka određivanje faza komponenata ne bi imalo smisla. U kalibracijskom postupku određuju se faze harmonika signala na pretvaračima za pojedinu konfiguraciju mjerenja prema signalu koji nije utjecan vjetrom i time dolaze do izražaja utjecaji slučajnih faza. U postupku mjerenja gdje su prijemni signali utjecani vjetrom oduzimaju se prije zapamćene kalibracijske faze od mjernih faza i time se dobivaju podaci koji se koriste pri određivanju brzine i smjera vjetra. U kalibracijskom postupku se nakon određivanja faza za svih šest komponenata po pretvaraču i svih 485 prozora linearno oduzima faza po prozorima prouzročena pomicanjem istih. Ono što se provodi samo kod kalibracijskog postupka je vektorsko zbrajanje jediničnih vektora rezultata po stabilnim prozorima. Od tog zbroja se uzima faza za svaku komponentu koja se kasnije oduzima od mjerne faze za istu komponentu i konfiguraciju na određenom pretvaraču. U mjernom postupku se ne poništavaju linearno rastuće faze, već se to radi kasnije pri pozivanju funkcije *find_delay,* gdje se provodi algoritam za više-harmonijsko određivanje vremena kašnjenja. Ono se određuje iz međuodnosa trenutačnih faza, kao i njihovim frakcionalnim dijelovima koji su definirani trenutačnim fazama. Radi veće točnosti koristi se težinsko usrednjavanje između komponenti. Neki izvadci iz koda su u poglavlju 3.1

3.1 Goerzelov algoritam, DFT, STFT

Diskretna Fourieova transformacija je definirana: $X_k \overset{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i k n/N}, \quad k \in \mathbb{Z}$,

$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{2\pi i k n/N}, \quad n \in \mathbb{Z}$.

Goerzelov algoritam služi za dobivanje informacija o pojedinoj frekvencijskoj komponeneti u funkciji duljine N uzoraka različitih od nule.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\mathrm{j}2\pi k \frac{n}{N}}, \qquad k = 0, \ldots, N-1. \quad (1)$$

Prilagodba jednadžbe (1) daje jednadžbu (2)

$$X[k] = e^{j2\pi k\frac{N}{N}} \sum_{n=0}^{N-1} x[n] e^{-j2\pi k\frac{n}{N}}, \quad (2)$$

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi k\frac{n-N}{N}}. \quad (3)$$

U jednadžbi (3) prepoznajemo konvoluciju funkcije $x[n]$ i $h_k[n] = e^{j\frac{2\pi}{N}k\cdot n} \cdot \mu[n]$, gdje je $x[n] = 0$ za $n < 0$ i $n \geq N$.

$$y_k[m] = \sum_{n=-\infty}^{\infty} x[n] h_k[m-n], \quad (4)$$

$$y_k[m] = \sum_{n=0}^{N-1} x[n] e^{j2\pi k\frac{m-n}{N}} u[m-n] \quad (5)$$

Iz (5) se vidi da je (3) konvolucija u točki $m = N$. K-ti višekratnik zajedničke frekvencije je konvolucija (5) u točki N.

$$X[k] = y_k[N] \quad (6)$$

Za pojedinu k-tu komponentu kompleksni uzorak je odziv sustava na pobudni signal $x[n]$ u točki N. Taj sustav ima impulsni odziv jednak $h_k[n]$. U slijedećim jednadžbama impulsni odziv će biti transformiran z-transformacijom u donje područje.

$$H_k(z) = \sum_{n=-\infty}^{\infty} h_k[n] z^{-n} \quad (7)$$

$$= \sum_{n=-\infty}^{\infty} e^{j2\pi k\frac{n}{N}} u[n] z^{-n} \quad (8)$$

$$= \sum_{n=0}^{\infty} e^{j2\pi k\frac{n}{N}} z^{-n} \quad (9)$$

$$= \sum_{n=0}^{\infty} \left(e^{j2\pi\frac{k}{N}} z^{-1}\right)^n, \quad (10)$$

Jednadžbu (10) možemo promatrati ako geometrijski red za koji vrijedi da je $q = e^{j\frac{2\pi}{N}k} z^{-1}$ i $|q| < 1$ za $|z| > 1$. Red konvergira i njegova je suma jednaka (11):

$$H_k(z) = \frac{1}{1 - e^{j\frac{2\pi k}{N}} z^{-1}}. \quad (11)$$

$$y_k[n] = x[n] + e^{j\frac{2\pi k}{N}} y_k[n-1], \quad \text{with } y_k[-1] = 0. \quad (12)$$

(12) je jednadžba diferencija tog sustava.

Međutim, ova jednadžba prvog reda sadržava kompleksni multiplikacijski faktor koji povećava računsku složenost što želimo izbjeći, a posebno kod implementacije na eZdsp TMS320VC5505 razvojnom sustavu koji sadrži procesor koji je namijenjen frakcionalnoj aritmetici. Zato množimo jednadžbu (11) kako slijedi:

$$H_k(z) = \frac{1 - e^{-j\frac{2\pi k}{N}} z^{-1}}{\left(1 - e^{-j\frac{2\pi k}{N}} z^{-1}\right)\left(1 - e^{j\frac{2\pi k}{N}} z^{-1}\right)} \qquad (13)$$

$$= \frac{1 - e^{-j\frac{2\pi k}{N}} z^{-1}}{1 - 2\cos\left(\frac{2\pi k}{N}\right) z^{-1} + z^{-2}}. \qquad (14)$$

$$y_k[n] = x[n] - x[n-1]e^{-j\frac{2\pi k}{N}} + 2\cos\left(\frac{2\pi k}{N}\right) y_k[n-1] - y_k[n-2] \qquad (15)$$

Sada možemo rastaviti prijenosnu funkciju na dvije i promatrati ih kao dva zasebna sustava.

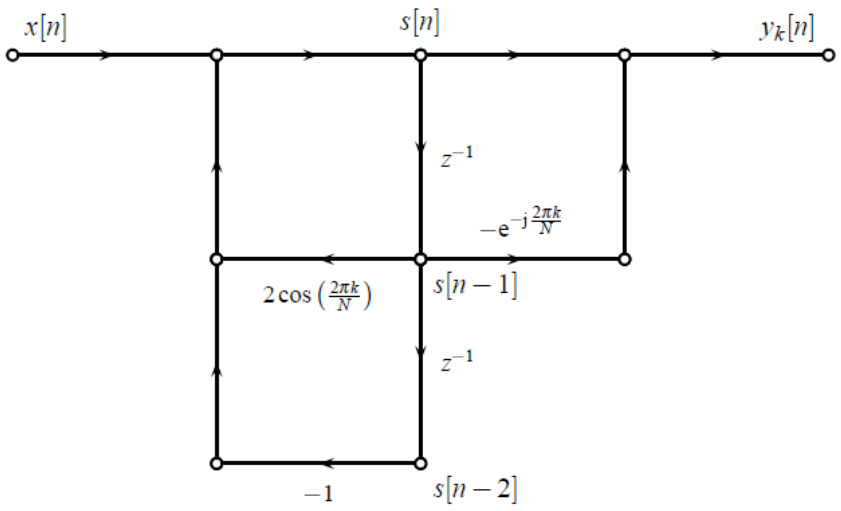
$$s[n] = x[n] + 2\cos\left(\frac{2\pi k}{N}\right) s[n-1] - s[n-2], \qquad (16)$$

Sa početnim uvjetima jednakim nuli.

$$y_k[n] = s[n] - e^{-j\frac{2\pi k}{N}} s[n-1] \qquad (17)$$

Sa početnim uvjetima jednakim nuli.

Ovo rješenje je dobro zato što samo u zadnjem koraku moramo množiti kompleksnim koeficijentom s obzirom na to da diferencijska jednadžba (17) nije rekurzivna. Često se to predočava grafički kako slijedi.



Slika 4. Grafički prikaz Gerzelovog algoritma

U literaturi se, međutim, često koristi poopćena varijanta Gerzelovog algoritma, koja je u našem slučaju nema značaja, a odnosi se na okolnosti pri kojima se Gerzelovim algoritmom želi dobiti podatak o komponenti koja nije cjelobrojni višekratnik zajedničke frekvencije. Za primjenu u okviru ovog rada bitno je uočiti da se N-ti uzorak odziva gore spomenutog sustava može ostvariti kao N-1 uzorak pomnožen kompleksnom eksponencijalom $e^{j\frac{2\pi}{N}k}$. To se može pokazati slijedećim jednadžbama:

$$y_k[N] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{j}2\pi k \frac{n-N}{N}} u[N-n] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{j}2\pi k \frac{n-N}{N}} \tag{18}$$

$$y_k[N-1] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{j}2\pi k \frac{n-(N-1)}{N}} u[(N-1)-n] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{j}2\pi k \frac{n-N}{N}} \mathrm{e}^{-\mathrm{j}2\pi k \frac{1}{N}}. \tag{19}$$

$$y_k[N] = y_k[N-1] \cdot \mathrm{e}^{+\mathrm{j}2\pi \frac{k}{N}}. \tag{20}$$

Navedena metoda je primjenjiva i za računanje STFT tako da se u jednom prolazu po signalu mogu dobiti rješenja za sve prozore. Tada se najprije signal mora prilagoditi diferencijacijom koja je u Matlabu napisana kako slijedi

```
% Extend input signals with N zeros from the left side
% (simulate zero states of the FIR filter)
s1tp=[zeros(1,N) rxw];
s2tp=[zeros(1,N) ryw];

% Input signal differentiation (1-z^-N)
s1td=s1tp(N+1:end)-s1tp(1:end-N);
s2td=s2tp(N+1:end)-s2tp(1:end-N);
```

Početna stanja diferencijskih jednadžbi su izjednačena nulom, taj dio je tu izostavljen.

```
% Gerzelov algoritam
for i=1:length(s1td),

    % Prvi prijemnik
    y=ones(1,6)*s1td(i)+2*cos([w1 w2]).*st1(1,:)-st1(2,:);
% OVO JE ANALOGNO JEDNADŽBI (16)
    if (i>=N),
       yc1(i-(N-1),:)=y-exp(-sqrt(-1)*[w1 w2]).*st1(1,:);
% OVO JE ANALOGNO JEDNADŽBI (17)
    end;
    st1(2,:)=st1(1,:);
    st1(1,:)=y;

    % Drugi prijemnik
    y=ones(1,6)*s2td(i)+2*cos([w2 w1]).*st2(1,:)-st2(2,:);
% OVO JE ANALOGNO JEDNADŽBI (16)
    if (i>=N),
       yc2(i-(N-1),:)=y-exp(-sqrt(-1)*[w2 w1]).*st2(1,:);
% OVO JE ANALOGNO JEDNADŽBI (17)
    end;
    st2(2,:)=st2(1,:);
    st2(1,:)=y;

end;
```

```
     yc1=ones(Na,1)*exp(-sqrt(-1)*[w1 w2]*(N-1)).*yc1;
     yc2=ones(Na,1)*exp(-sqrt(-1)*[w2 w1]*(N-1)).*yc2;
% OVO JE ANALOGNO JEDNADŽBI (20), k je cijeli broj
```

Pri implementaciji u C-u za eZdsp TMS320VC5505 javit će se neki problemi tipični frakcionalnoj aritmetici koji nisu riješeni Matlab implementacijom.

4. Implementacija u C-u

4.1 Frakcionalna aritmetika

Procesor razvojnog sustava je namijenjen cjelobrojanom radu što predstavlja ograničenja u točnosti i mijenja pristup programiraju. Podaci koji se koriste su izraženi su Q15 formatom. Q15 je format koji se sastoji od 15 podatkovnih bitova i jednog bita predznaka. To nas podsjeća na svima dobro poznat 2k način zapisa brojeva sa predznakom iz intervala $[-32768, 32767]$. To i je taj zapis, uz tu razliku, što niz bitova drugačije interpretiramo i tako elegantno rješavamo probleme na koje nailazimo zbog podatkovnih ograničenja zapisa određenog broja decimalnih mjesta, odnosno prije spomenutog 2k zapisa. Frakcionalna interpretacija je iz intervala $\left[-1, \frac{2^{15}-1}{2^{15}}\right]$. Iz 2k zapisa ovu interpretaciju dobivamo dijeljenjem sa 2^{15} (u slučaju Q15 formata). Na primjer ako se radi o Q4.11 formatu onda možemo prikazati brojeve iz intervala $[-2^4,\ 2^4 - 2^{-11}]$. Brojevi iz tog intervala se pomiču korakom od 2^{-11}. Jedan je bit najvećeg značaja rezerviran za predznak i ima težinu -1.

4.1.1 Gubitak preciznosti i preljev

Kod množenja dvaju 16-bitnih brojeva 2k rezultat je duljine 31 bita i prije spremanja u memorijsku lokaciju proširuje se jednim bitom s lijeva tako da rezultat ima dva bita predznaka. To ne utječe na interpretaciju 2k brojeva, ali utječe na interpretaciju frakcionalnog zapisa koji se pogrešno shvaćaju kao upola manji.

Rezultat množenja Q15 brojeva se često mora pohraniti na lokaciju istog tipa te je stoga potrebno pomaknuti riječ za 15 bita udesno i tako sačuvati gornji dio riječi koji se može frakcionalno interpretirati uz određeni gubitak preciznosti. Pomicanjem se brojevi pri zaokružuju uvijek prema manjoj vrijednosti. Da bi se uvela ravnoteža i smanjila vjerojatnost pogreške u algoritmima veće zahtjevnosti, prije posmicanja se dodaje 0x4000 i tako se u ovisnosti u prvom bitu donje riječi gornja riječ zaokružuje prema gore. Navedene operacije se mogu ostvariti programski ili korištenjem *intrinsic* funkcija karakterističnih sustavu od interesa.

Prema tome, kod množenja ne može doći do preljeva (izuzevši negaciju jedinice), već samo do podljeva koji nastaje kada se nakon množenja u gornjoj riječi registriraju sve nule uslijed računanja premalim frakcioanalnim vrijednostima. Kod zbrajanja, međutim, može doći do preljeva. Rješenje tog problema je ograničenje na prikaz brojeva iz intervala $\left[-1, \frac{2^{15}-1}{2^{15}}\right]$. No ako se oni događaju često, doći će do značajne akumulacije pogreške u algoritmu. Stoga sam se u izvedbi u C-u koristio skaliranjem podataka. Faktori skaliranja se nakon pretpostavljene vrijednosti moraju provjeriti simulacijom programa.

4.2 Opis pojedinih dijelova

Napisan je jedan i za kalibraciju i za mjerenje koji sadrži Goerzelov algoritam, a podatak u registru određuje o kojem se postupku radi. Kod kalibracije je korišten

```
i = 0; a = 32406; b = 0;

        for(j=0;j<6;j++){

                if (i == 0){

                        Re_lin_prvi[i][j]=a;
                        Im_lin_prvi[i][j]=b;

                        }

                }i++;

        for(j=0;j<6;j++){

                for(i=1; i<Na; i++){

                pom1=(long)a*cs_wprvi[j]+(long)b*sn_wprvi[j];
                pom2 = (long)b*cs_wprvi[j] - (long)a*sn_wprvi[j];
        pom1 = pom1 + 0x4000;
        pom2 = pom2 + 0x4000;



                Re_lin_prvi[i][j] = (short)(pom1 >> 15);
                Im_lin_prvi[i][j] = (short)(pom2 >> 15);

                a=Re_lin_prvi[i][j];
                b=Im_lin_prvi[i][j];

                }

        }
```

generator faze koji generira kompleksne brojeve kojima amplituda pada u svakom koraku kojih ima 485, koliko i prozora STFT za svih šest prijemnih komponenata. Na oba pretvarača u jednoj konfiguraciji mjerenja registriraju se komponente istih frekvencija, pa se matrice Re_lin_prvi[i][j] i Im_lin_prvi[i][j] koriste za oba pretvarača.

Teoretski u početnom koraku (za prozor koji nije pomaknut) bi realni dio trebao biti jednak jedan i svi slijedeći rezultati bili bi na jediničnoj kružnici. U frakcionalnoj aritmetici, uslijed dodavanja 0x4000 i posmicanja za 15 bitova broj može izaći iz dinamike što uzrokuje akumulaciju greške i većina redaka matrice će imati

besmislena rješenja. Zato je u ulaznom koraku odabran broj manji od $2^{15} - 1$. A za potrebe algoritma je bitno jedino da se očuvaju odnosi faza, a ne amplituda.

```
for (i=0; i < 3*N; i++){

                                        for (j=0; j<=5; j++){

                                        temp3 = (long)st1[1][j];
                                        temp3=(long)temp3<<15;

                                        temp3_d = (long)st2[1][j];
                                        temp3_d=(long)temp3_d << 15;

                                        temp1 = (long)f_rxw[i] * pom;
                                        temp1_d = (long)f_ryw[i] * pom;
                    //pom je faktor skaliranja

                                        temp2 = (long)cs_wprvi[j] * st1[0][j];
                                        temp2_d = (long)cs_wprvi[j] * st2[0][j];

                                        temp2=(long)(temp2<<1);
                                        temp2_d=(long)(temp2_d<<1);

                                        y[j] = (long)temp1 + temp2- temp3;
                                        d_y[j] = (long)temp1_d + temp2_d- temp3_d;

                                        z_y[j] = (long)y[j] + 0x4000;
                                        z_d_y[j] = (long)d_y[j] + 0x4000;

                                        f_y[j] = (short)(z_y[j] >> 15);
                                        d_f_y[j] = (short)(z_d_y[j] >> 15);

                                        }
```

Zapisan je prvi dio Goerzelovog algoritma. Matrice st1[i][j] i st2[i][j] su stanja iz prethodnog koraka diferencijske jednadžbe. Njima se ostvaruje rekurzivna veza. Razlikuju se u imenu prema prvom i drugom pretvaraču. U retcima su pohranjena stanja (prethodno i prethodno prethodnom). U ovom odsječku su zapamćeni iz prethodnog prolaza petljom. Varijable pomoćne se razlikuju u imenu prema nastavku _d koji označava da se radi o drugom pretvaraču. Obavljene su matematičke operacije pokazane u poglavlju 3.1 samo prilagođene za frakcionalnu aritmetiku. Može se primijetiti da su međurezultati u ovoj petlji spremani u *(long)* tip podatka duljine 32 bita. Time je izbjegnut nepotreban gubitak preciznosti koji bi nastao kada bi svakog puta broj zapisivali u kraći zapis kako je objašnjeno. Tako se radi u cijelom kodu. Računanje Gerzelovog algoritma za neku komponentu je određeno koeficijentom k te komponente, a budući da se na pretvaračima registriraju komponente istih koeficijenata, matrica cs_wprvi[j] koja slijedi iz

jednadžbe (16) je određena samo za prvi pretvarač. To znači da će se za drugi pretvarač u matricu rezultati pohraniti u matricu po koeficijentima istim redoslijedom kao za prvi pretvarač, prvo komponente signala S1 onda S2.

```
if (i>=N-1){
                                        for (j=0; j<6; j++){

                                        temp4 = (long)cs_wprvi[j] * st1[0][j];
                                        temp4_d = (long)cs_wprvi[j] * st2[0][j];

                                        temp5 = (long)y[j] - temp4;
                                        temp5_d = (long)d_y[j] - temp4_d;

                                        temp5 =(long)temp5+  0x4000;
                                        temp5_d =(long)temp5_d+  0x4000;

                                        temp6 = (long)sn_wprvi[j] * st1[0][j];
                                        temp6_d = (long)sn_wprvi[j] * st2[0][j];

                                        temp6 = (long)temp6 + 0x4000;
                                        temp6_d = (long)temp6_d + 0x4000;

                                        Re_yc1[i-N+1][j] = (short)(temp5 >> 15);
                                        Re_yc2[i-N+1][j] = (short)(temp5_d >> 15);

                                        Im_yc1[i-N+1][j]=(short)(temp6>>15);
                                        Im_yc2[i-N+1][j]=(short)(temp6_d>>15);

                                        }

                                }

                                for (j=0; j<=5; j++){

                                        st1[1][j] = st1[0][j];
                                        st1[0][j] = f_y[j];

                                        st2[1][j] = st2[0][j];
                                        st2[0][j] = d_f_y[j];

                                }

                }
```

Nakon ovoga slijedi dio algoritma analogan jednadžbi (20) koji tu nije napisan.

Svi dijelovi algoritma koji uključuju rad s kompleksnim brojevima, zapisani su s razdvojenim realnim i imaginarnim dijelovima u Kartezijevom sustavu.

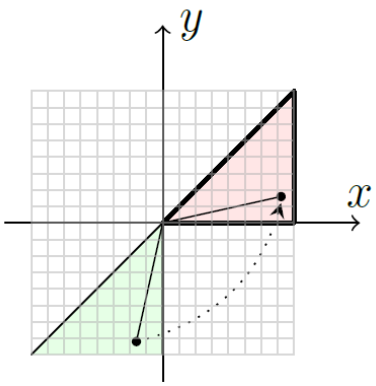
4.2.1 Arkus tangens funkcija

U implementaciji za korišteni eZdsp,u cjelobrojnoj aritmetici, nije moguće koristiti *ansi-C* funkciju uključenu u *math.h* biblioteku. Potrebno je naći način računanja faze kompleksnog broja koristeći se osnovnim operacijama koje procesor podržava i maksimalno skratiti vrijeme izvođenja i računsku složenost.

To je ostvareno aproksimacijom funkcije arkus tangens polinomom Taylorovom transformacijom. Odabran je onaj dio te funkcije za koje funkcija daje vrijednosti iz intervala koji je pogodan za frakcionalnu aritmetiku i koji se može daljnjim manipulacijama iskoristiti i za ostale vrijednosti te funkcije.

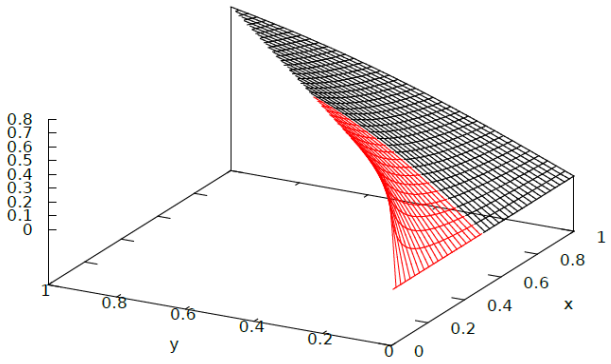
S obzirom na grešku koja je akumulirana u kodu koji prethodi rješenju arkus tangens funkcije, odabrano je da je referentni dio funkcije onaj koji daje vrijednosti prvog oktanta.

$$\begin{cases} x < 0 \\ y < 0 \\ y < x \end{cases} \Rightarrow \operatorname{atan2}(y,x) = -\frac{\pi}{2} - \operatorname{atan2}(|x|,|y|)$$



Slika 5.

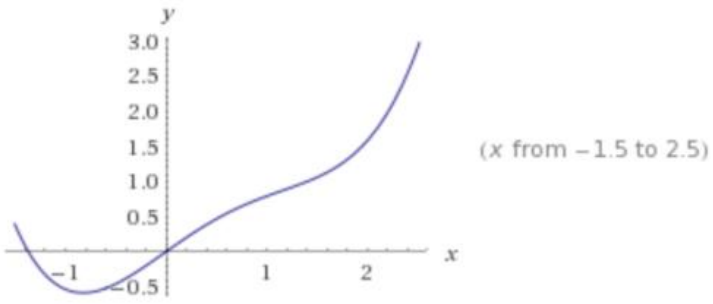
Na slici 5. je prikazana simetrija na primjeru računanja vrijednosti funkcije u šestom oktantu. Simetrija se može grafički uočiti i za preostalih šest slučaja koji ne daju rezultate iz prvog oktanta.



Slika 6. Arkus tanges funkcija za prvi oktant

Implementacija je napravljena tako da je argument funkcije kvocijent imaginarnog i realnog dijela. Označen je sa z i uvijek je manji od jedan, budući da računamo

vrijednosti iz prvog oktanta. Z – je apsolutni kvocijent koji je neovisno o vrijednostima kojima je pozvana funkcija manji od jedan.



Slika 7. Polinomijalna aproksimacija $g(z)$ za prvi oktant

Polinomijalna aproksimacija je četvrtog reda oko točke z=0.5 koja je približno na sredini prvog oktanta. Polinom je zapisan Hornerovom shemom da se može slijedno izvršiti samo operacijama množenja. Koeficijenti polinoma su skalirani na vrijednosti manje od jedan, a rješenje polinoma je pomnoženo sa $\frac{S}{2\pi}$, gdje je S faktor skaliranja. Vrijednosti iz $[0, 2\pi)$ su prikazane intervalom $[0, 1)$, zato je dijeljeno sa 2π.

$$g(z) = v * \big(v * \big((0.1536 * v - 0.349867) * v - 0.0255995\big) + 1.0112\big) - 0.00141863$$

Prilagođeno frakcionalnoj aritmetici to izgleda ovako:

```
shift=(long)x1;
p1 = (long)4576 * z - (long)(shift << 15);
p1 = p1 + 0x4000;
s_p1 = (short)(p1 >> 15);

shift=(long)x2;
p2 = (long)s_p1*z - (long)(shift << 15);
p2 = p2 + 0x4000;
s_p2 = (short)(p2 >> 15);

shift=(long)x3;
p3 = (long)s_p2*z + (long)(shift << 15);
p3 = p3 + 0x4000;

s_p3 = (short)(p3 >> 15);
shift=(long)x4;
p4 = (long)s_p3*z - (long)(shift << 15);
p4 = p4 + 0x4000;                    s_p4 = (short)(p4 >> 15);
```

Pristupanje ostalim oktantima se može razumijeti iz slijedećeg programskog koda.

```
If      (absI > absR && R > 0 && Im > 0){shift=(long)a;
        pom_rez = (long)(shift << 15) - pom_rez; } // drugi oktant

       else if (absI > absR && R<0 && Im>0){shift=(long)a;
        pom_rez = (long)(shift << 15) + pom_rez; } // treci

       else if (absI < absR && R<0 && Im>0){shift=(long)b;
       pom_rez = (long)(shift << 15) - pom_rez; } // itd

       else if (absI < absR && R<0 && Im<0){shift=(long)b;
        pom_rez = (long)(shift << 15) + pom_rez; }

       else if (absI > absR && R<0 && Im<0){shift=(long)c;
       pom_rez = (long)(shift << 15) - pom_rez; }

       else if (absI > absR && R>0 && Im<0){shift=(long)c;
       pom_rez = (long)(shift << 15) + pom_rez; }

       else if (absI < absR && R>0 && Im<0){shift=(long)b;
       pom_rez = (long)(shift << 15) - pom_rez;

       pom_rez = pom_rez + (long)(shift << 15);

       }; // osmi

       pom_rez = pom_rez + 0x4000;
       rez = (short)(pom_rez >> 15);
```

Greška akumulirana ovom funkcijom je manja od one akumulirane prethodnim algoritmom pa stoga ne treba više od ovoga optimizirati funkciju povećanjem polinomijalnog stupnja.

4.3 Poboljšanja i optimizacije

Sustav bi trebalo optimirati da radi u stvarnom vremenu i to tako da blokovski obrađuje mjerne podatke s dva pretvarača za svaku konfiguraciju mjerenja. Važnu ulogu u ostvarenju te zadaće ima ugradbeni stereo Codec TLV320AIC3204. Njega treba konfigurirati za rad s frekvencijom otipkavanja od 96000 uzoraka u sekundi koja utječe na rad A/D i D/A pretvornika. U tu svrhu treba konfigurirati DMA sklop za automatski prijenos uzoraka pobudnog, i uzoraka mjernog signala prema Codec-u. Takav sustav znači dvostruke memorijske spremnike u vremenski preklapajućem radu tako da jedan par prikuplja pobudni signal, a drugi par obrađuje prethodno prikupljene podatke. Također treba ostvariti proslijeđivanje obrađenih podataka serijskom sabirnicom preko USB sučelja nadređenom računalu u stvarnom vremenu.

S obzirom na te zahtjeve, algoritam mora raditi što brže, te je potrebno da svi podaci za koje je to moguće budu unaprijed pohranjeni u memoriju kao što su konstante i koeficijenti kojih ima poveći broj. Dalje, potrebno je maksimalno iskoristiti arhitekturu razvojnog sustava. Podatke o sustavu nalazimo u službenim priručnicima firme *Texas Instruments.* Treba iskoristiti mogućnost dvostruke MAC (*Multiply and accumulate*) operacije u jednom ciklusu izvođenja procesora. To se programski u C-jeziku rješava tako da se unutar iste petlje pozivaju operacije množenja s istim operandom koje ne zapisuju vrijednost na istu lokaciju. Dodatno za to treba *kompajleru* dati uputu da može izvršiti operaciju *dual MAC* u istom ciklusu, odnosno da se rješenje pojedine operacije ne koristi za računanje njoj dualne operacije. To se obavlja prefiksom *onchip* nekoj varijabli. U tu svrhu je i pojednostavljena implementacija u C-u tako da se u istoj petlji obavlja Gerzelov algoritam za oba pretvarača jer su njihova rješenja vezana koeficijentima komponenata i međusobno neovisna.

Informaciju o broju ciklusa u nekom dijelu kodu ili za neku petlju dobivamo koristeći funkciju clock() koju je potrebno inkorporirati u kod na ključnim mjestima. Pored toga, CC4 (Code composer studio 4) ima ugrađen brojač ciklusa kojim se još lakše dobivaju informacije. Prethodno opisani postupci su nešto od čega treba početi čak i bez analize asemblerskog koda, čija se struktura može pratiti pri izvođenju, i koja se posredno mijenja. Najkritičniji odsječak koda je Gerzelov algoritam zbog jako velikog broja iteracija koje zahtijevaju međurezultat iz prethodnog koraka, akumulira se greška koja se prenosi u slijedeći korak i tako dalje. Još jedna zanimljiva informacija je ta da je Gerzelov algoritam kao IIR filtar na granici stabilnosti, tj. polovi IIR sustava su na jediničnoj kružnici, što u frakcionalnoj implementaciji kod koje se akumuliraju greške uslijed zaokruživanja, znači da za velik broj uzorka odziv može početi rasti i davati nama neželjene rezultate. Ali u našem slučaju broj od 726 uzoraka nije dovoljno velik da bi se to dogodilo. Primjenom optimizacija u Gerzelovoj petlji se smanjio broj ciklusa izvođenja za oko 120000. Slijedi izmjereni broj ciklusa po pojedinim dijelovima koda (sve pomnožiti sa 10^6): $f_{procesor} = 100MHz, T = 10^{-8}s$

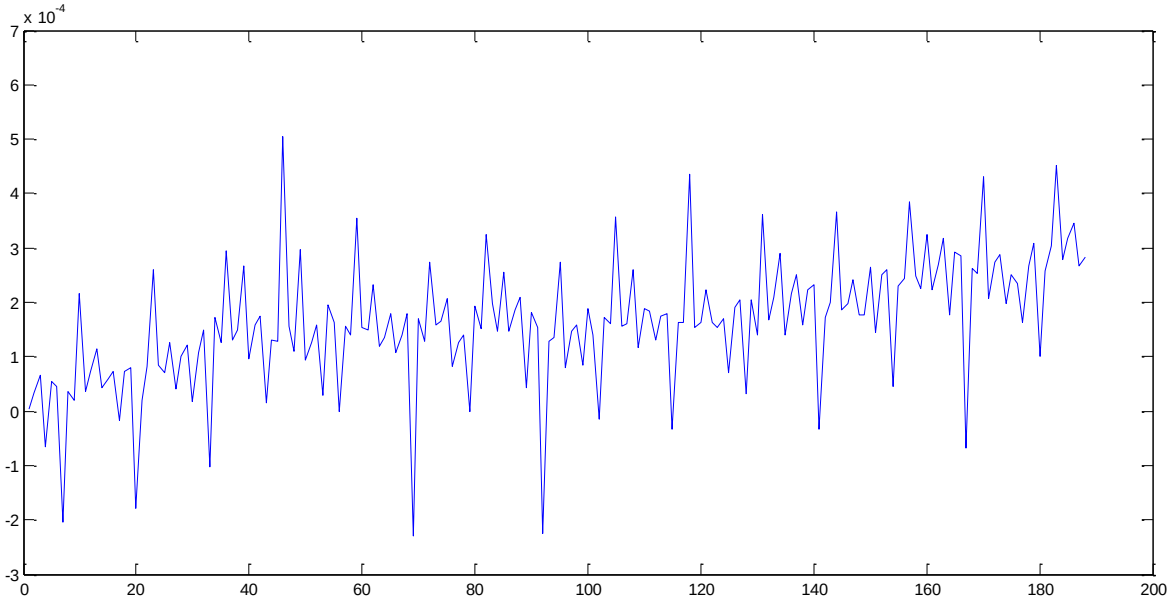
Tablica 1.

generator negativno rastuće faze	0.238
Gerzelov algoritam	1.236 (prije optimizacije 1.4)
oduzimanje linearne faze (i ostala množenja dva kompleksna broja sličnog su broja ciklusa)	0.5
akumulator koji se koristi kod usrednjavanja pri kalibraciji	0.107

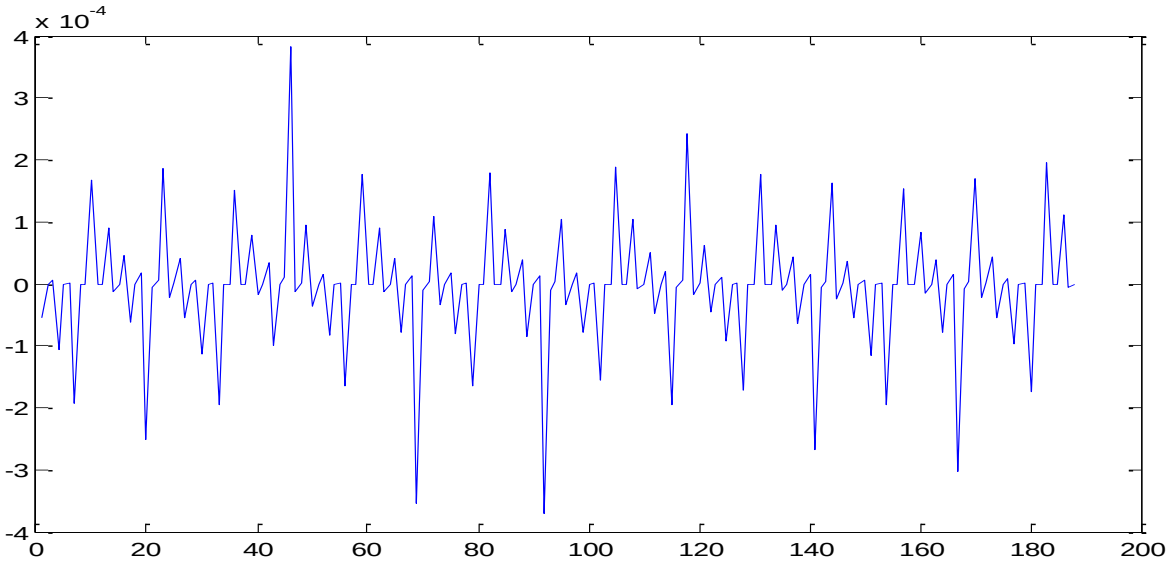
Ostali dijelovi su procesorski manje zahtijevni prema ovdje navedenima.

Ispravnost programa je testirana tako da je napisan novi kod iste funkcionalnosti onome u Matlabu s tipovima podataka *double* u ansi C-u. Cjelobrojna implementacija je zatim pisana u C-u korištenjem tipova podataka od 16 i 32 bita kada se provjeravala ispravnost cjelobrojne aritmetike, testiralo faktore skaliranja i uspoređivalo rezultate s verzijom napisanom u *doublu* za koju smatramo da je beskonačno precizna i čiji su rezultati identični rezultatima iz Matlaba. Podaci koji će se, kada se sustav realizira, dovoditi serijskom sabirnicom, čitani su iz tekstualnih datoteka. Ulazni podaci za cjelobrojnu implementaciju su skalirani sa 1.6 i pretvoreni u željeni interval množenjem sa 2^{15} i nakon toga zaokruženi na cjelobrojnu vrijednost. Na kraju prilagođen je kod minimalnim izmjenama za izvođenje u CCS4 i nakon toga je optimiran za specifičnu arhitekturu. Rezultati dobiveni CCS4 okruženjem na DSP pločici su identični onima u C-u za testiranje ispravnosti aritmetike.

Razlika u rezultatima frakcionalnog i pristupa pomičnog zareza se očituje razlikom najčešće na trećoj ili drugoj decimali. Na slici 8. je razlika rezultata dobivenih pristupom pomičnim zarezom i frakcionalnom aritmetikom. Rezultati se odnose na prvi pretvarač i to komponentu najmanje frekvencije signala S1 (koji do prvog pretvarača prelazi duži put nego S2). Slično izgledaju razlike grešaka i za ostale slučajeve. Razlika je prikazana na duljini od 188 uzoraka. Ti uzorci odgovaraju stabilnom intervalu pomaknutih prozora (prvi značajan smatramo onaj pomaknut za 108 uzoraka, a posljednji za 295 uzoraka) za koji ima fizikalnog smisla promatrati rezultate.



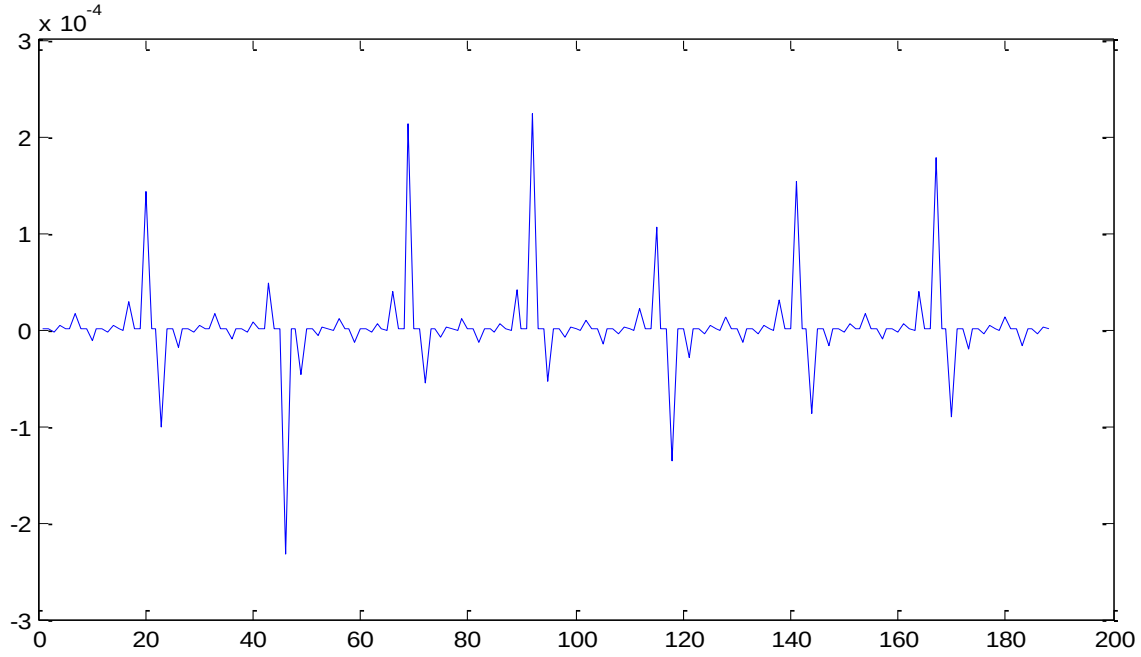
Slika 8.



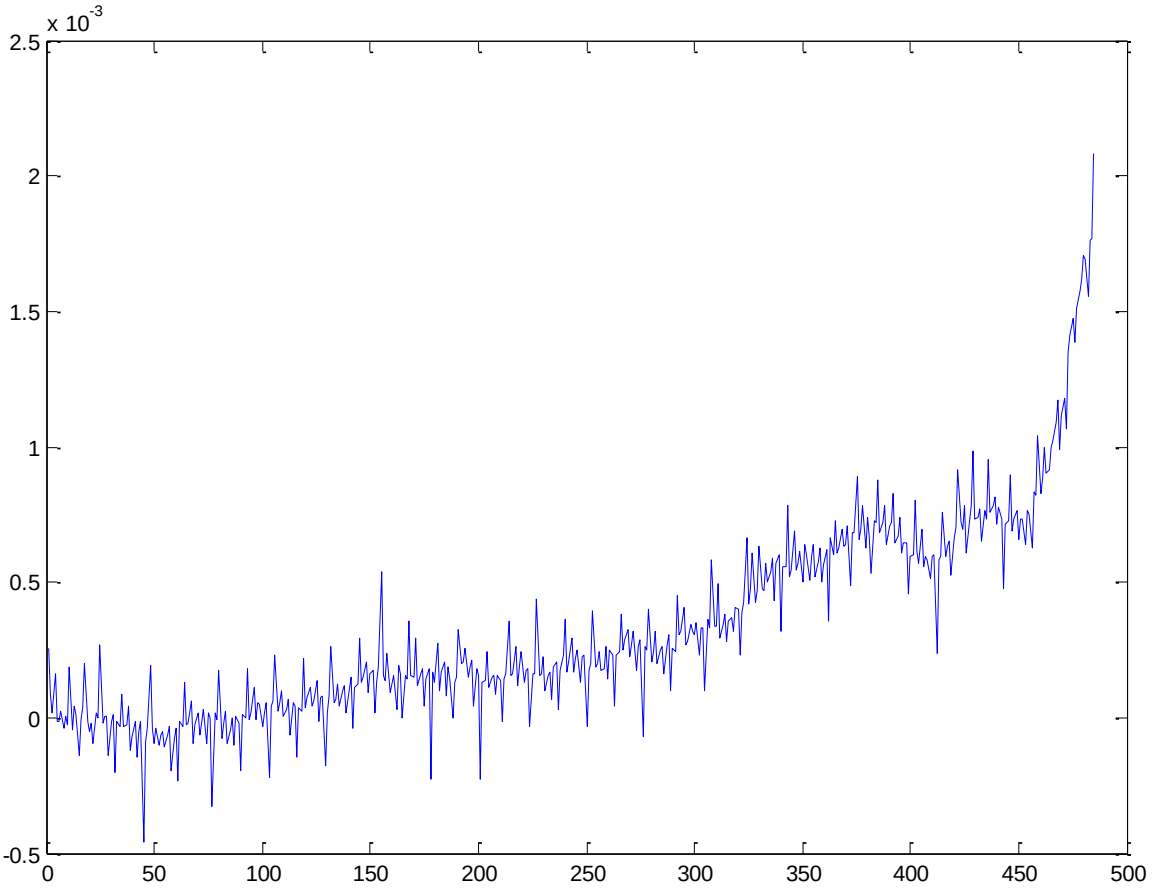
Slika 9.

Najprije se uočava da je ta razlika sve veća s pomakom prozora, najviše zbog akumulacije greške u Gerzelovom algoritmu. Zatim vide se gotovo periodičke oscilacije u razlici koje su povezane sa arkus tangens funkcijom. Na slici 9. je uspoređena razlika korištenja arkus tangens funkcije iz biblioteke math.h i arkus tangens funkcije napisane polinomijalnom aproksimacijom (obje napisane varijablama tipa *double*). Radi se o istoj konfiguraciji mjernja koja je vezana za prethodnu sliku. Uslijed periodičkog povećavanja faze s pomicanjem prozora, ulazi arkus tangens funkcija se ponavljaju nakon određenog broja uzoraka (koji ovisi o frekvenciji komponente). Tada dolazi do izražaja nesavršenost arkus tangens funkcije polinomijalno aproksimirane, koja za određene ponavljajuće argumente daje manje točne rezultate.Grafički je to uočljivo uspoređujući slike.

Na slici 10. je, za usporedbu, razlika rezultata arkus tangens funkcije iz *math.h* i one polinomijalne šestog reda.

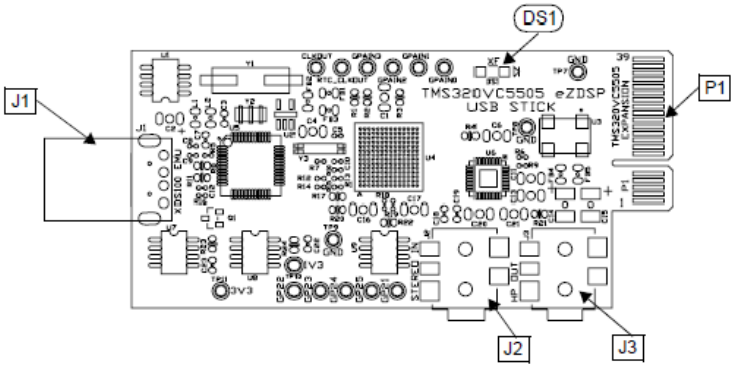


Slika 10.



Slika 11. Razlika rješenja *fixed-point* i *floating-point* za sve prozore

5. Razvojni sustav eZdsp TMS320VC5505



Slika 8.

Tablica 1.

Konektor	Funkcija	Strana pločice	Br. pinova
J1	USB	gornja	2
J2	Stereo ulaz	gornja	2
J3	Stereo izlaz	gornja	2
P1	Ekspanzija	gornja/donja	20x2

Sustav ima četiri konektora navedena u tablici 1. Nije potrebno dovoditi napajanje, već je sustav napajan preko USB priključka. Preko J1 konektora sustav se spaja na osbno računalo. Preko J2 priključka signal se dovodi na Codec. Priključak J3 signal odvodi sa Codeca. J2 i J3 imaju po dva pina od značaja, tako da se može slati dva niza podataka na J2 u našem slučaju.

Codec i procesor su vezani signalima I2S0_FS, I2S0_DX, I2S0_CLK, I2S0_RX, GPIO_CODEC_RESET, I2C_SCL,I2C_SDA. Ti moduli sudjeluju u serijskom prijenosu podataka.

5.1 Procesor TMS320VC5505 i periferni sklopovi

TMS320VC5505 je *fixed point Digital Signal Processor* (DSP) iz porodice TMS320C5000 proizvođača Texas Instruments. Procesor je namijenjen aplikacijama niske potrošnje. Navedeni sustav je temeljen na TMS320VC55xx DSP generaciji procesorskih jezgara. Pozornost je posvećena paralelizmu o kojem ima više informacija u službenim dokumentima proizvođača. Vezano za sabirnice, procesor ima jednu programsku, tri podatkovne sabirnice za čitanje, od kojih je jedna 32-bitna i dvije 16-bitne. Uz to postoje i dvije sabirnice za pisanje duljine 16 bita (i ostale ovdje nespomenute). Takva arhitektura omogućava istodobno čitanje četiri i pisanje dvaju 16-bitnih podataka u jednom ciklusu.

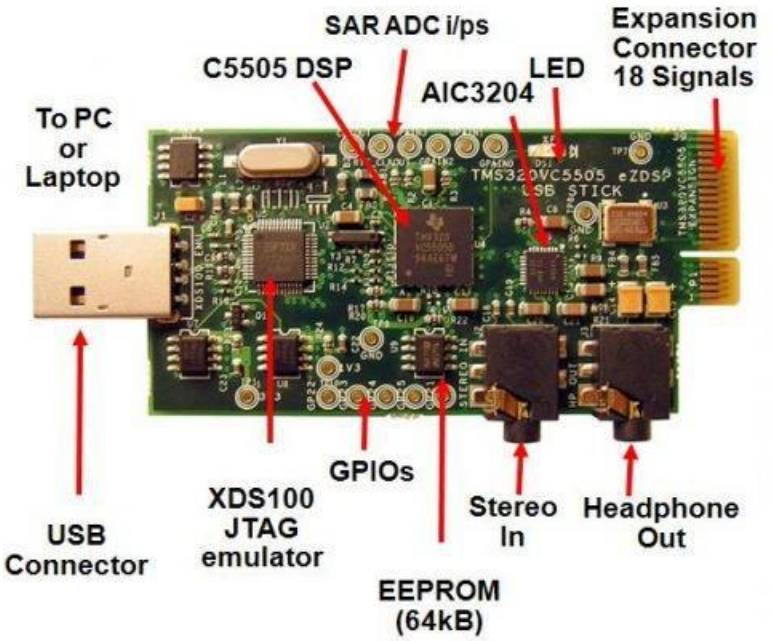
CPU ima mogućnost izvođenja jednog 17*17 bitnog množenja i jednog 32-bitnog zbrajanja po ciklusu. Ako se koristi *dual MAC* način rada, moguće je obavljati te operacije odjednom u jednom ciklusu. Središnja aritmetičko-logička jednica (40-bita) i dodatna ALU jedinica (16-bita) doprinose optimizaciji paralelizma i smanjuju potrošnju enegije. Neke značajne periferne jedinice su: A/D i D/A pretvornik, GPIO (*general purpose input output*) sabirnica, USB (*universal serial bus*), memorijsko sučelje, memorija EPROM(*electrical programmable read only memeory*) i ostale.

Već spomenuti I2S 0 moduli zajedno sa I2C modulom, koristeći GPIO sabirnicu, čine vezu među procesorom i Codecom. I2S 0 služi kao sučelje kojim se dovode signali sa A/D pretvornika prema procesoru i odvode sa procesora na D/A pretvornik, odnosno prema Codec-u.

I2C modul služi procesoru za razmjenu podataka sa vanjskim jedinicama. Preko tog modula se prenose podaci potrebni za konfiguraciju kodeka.

GPIO se također može konfigurirati ukoliko se žele čitati podaci sa vanjskog uređaja ili ih slati na neki vanjski uređaj.

Ugradbeni stereo Codec TLV320AIC3204 je codec niske potrošnje energije. Njime se mogu prilagođavati ulazni i izlazni kanali s obzirom na aplikaciju za koju se koristi. Može se kontrolirati pojačanje ulaznih signala prije A/D pretvorbe (što je bitno za našu aplikaciju) i ovisno o aplikaciji, izlazni analogni signali se mogu pojačavati do 29dB. Također bitno je podesiti signal takta za pojedinu aplikaciju (u našem slučaju 96000 uzoraka u sekundi). Pretvornici mogu raditi na frekvencijama od 8kHz do 192kHz.



Slika 9.

Implementacija algoritma ostvarena je korištenjem frakcionale aritmetike, za razliku od implementacije u Matlabu. Frakcionalna aritmetika je pogodna za procesor razvojnog sustava. Drugi način implementacije bio bi jako neučinkovit. Procesori koji su namijenjeni radu sa cjelobrojnim tipovima podataka, pristupačni su cijenom i sposobni su izvesti složene algoritme velikom brzinom uz pogrešku koja je mnogim aplikacijama prihvatljiva. Programer mora pri tom uložiti više vremena pri realizaciji sustava nego što bi trebao da se radi o implementaciji na *floating point* procesoru. U toku programiranja mora se voditi brigu o dinamičkim rasponima pojednih varijabli i programski spriječiti preljeve skaliranjem, dinamičkim skaliranjem ili ograničavanjem vrijednosti varijabli s obzirom na aplikaciju o kojoj se radi.

6. Zaključak

Na temelju teorijske podloge, matematičkih proračuna, te upoznavanjem sa razvojnim sustavom koji je temeljen na procesoru koji je namijenjen radu u frakcionalnoj aritmetici, koristeći podatke tipa *int* i *long*, realiziran je algoritam koji određuje faznu razliku ultrazvučnog signala. Rezultati pokazuju da sustavi slični ovdje korištenom imaju primjenu u velikom rasponu aplikacija koje toleriraju određenu pogrešku. Na primjeru anemometra napravljen je algoritam koji zajedno sa preostalim dijelom kojega radi kolega, čini cjelinu i može se praktično upotrijebiti. U ovom radu odabran je način određivanja faze komponenata na pretvaračima pomoću više-harmonijskih signala kojima je riješen problem određivanja pomaka periodičnog signala velike frekvencije. U tu svrhu korišten je razvojni sustav eZdsp TMS320VC5505 proizvođača Texas Instruments.

7. Sažetak

U ovom radu se proučava primjena sustava temeljenog na procesoru koji je namijenjen radu s podacima nepomičnog zareza, implementacijom algoritma određivanja fazne razlike ultrazvučnog signala. Najprije se rješava dvosmislenost registriranja faze harmoničnog signala koji na prijemnik dolazi zakašnjen za iznos višestukog perioda primjenjujući više-harmonijski signal s većim zajedničkim periodom od pojedinačnih. Zatim se matematički proučavaju diskretne Fourieove transformacije. Odabran je Gertzelov algoritam jer nema potrebe za računanjem svih kompleksnih uzoraka spektra. U daljnjem razvoju algoritma proučavaju se ograničenja i specifičnosti frakcionalne aritmetike, te se s obzirom na to isti ostvaruje u C-u. Jedna od posljedica tog pristupa je potreba za arkus tangens funkcijom dobivenom polinomijalnom aproksimacijom. Korišteni razvojni sustav je eZdsp TMS320VC5505.

Ključne riječi: fixed-point procesor, eZdsp TMS320VC5505, Gerzelov algoritam, više-harmonijski signal, anemometar

Summary

In this paper, fixed-point based processor development kit is studied trough implementation of phase-difference ultrasonic signal algorithm. First the phase-differece ambiguity of harmonic signal is managed using multy-harmonic signal. The discrete Fourier transforms and their use are discussed. Goerzel algorithm was chosen for that matter becouse of its optimal computational complexity for this application. In algorithm development, limitations and specific features of fractional arithmetics are analyzed. Coding was done using C-language in Code Composer Studio v4. One of the consequences of this approach is a need for arcus tangens polinomial approximation function.The development kit of interest is eZds TMS320VC5505.

Key words: fixed-point processor, eZdsp TMS320VC5505, Goerzel algorithm, multy-harmonic signal, anemometer

8. Literatura

[1] Prikaz brojeva u obliku frakcija, http://www.fer.hr/_download/repository/DOS0304_vj3.pdf, 25.5.2016

[2] Dr. sc. Petrinović Davor, A roubust synchronization method based on harmonic signals, Circuit theory and design, Davos Switzerland 30.8.1993

[3] TMS320VC5505 eZdsp USB Stick, *Technical Reference,* http://support.spectrumdigital.com/boards/usbstk5505/revb/files/usbstk5505_TechRef_revb.pdf, 1.6.2016

[4] TMS320C55x Optimizing C/C++ Compiler v 4.4, *User's Guide,* http://www.ti.com/lit/ug/spru281g/spru281g.pdf, 25.5.2016

[5] TMS320C55x, *Technical Overview,* http://www.ti.com/lit/ug/spru393/spru393.pdf, 24.5.2016

[6] TMS320C55x DSP, *Programmer's Guide*, http://www.ti.com/lit/ug/spru376a/spru376a.pdf, 20.5.2016

[7] Sysel and Rajmic, *Goertzel algorithm generalized to non-integer multiples of fundamental frequency*. EURASIP Journal on Advances in Signal Processing 2012