SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4504

IZVEDBA ALGORITMA ZA VIŠE-HARMONIJSKO ODREĐIVANJE VREMENA KAŠNJENJA

LUKA MURN

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 16. ožujka 2016.

ZAVRŠNI ZADATAK br. 4504

Pristupnik: **Luka Murn (0036474720)**
Studij: Računarstvo
Modul: Obradba informacija

Zadatak: **Izvedba algoritma za više-harmonijsko određivanje vremena kašnjenja**

Opis zadatka:

U okviru završnog rada potrebno je realizirati i verificirati algoritam za određivanje vremenskog kašnjenja primjenom testnog signala koji se sastoji od nekoliko sinusnih komponenti sa zajedničkim periodom i cjelobrojnim odnosnom osnovnih frekvencija. Ulaz u algoritam su trenutačne faze svih harmonijskih komponenti od kojih je sačinjen prijemni signal, a rezultat treba biti estimirano trenutačno vremensko kašnjenje prijemnog signala u odnosu na vremensko ishodište odaslanog signala. Kašnjenja su definirana s estimiranim cijelim brojem perioda harmoničkih komponenti koje je potrebno odrediti iz međusobnih odnosa trenutačnih faza, kao i njihovim frakcionalnim dijelovima koji su definirani trenutačnim fazama (0 do 2*pi) komponenti. Radi povećanja točnosti estimacije potrebno je koristiti težinsko usrednjavanje između komponenti. Potrebno je realizirati i programsku kalibraciju mjernog sustava radi poništenja faznih pogrešaka mjernog lanca, kao i konačnu estimaciju prosječnog kašnjenja za cijeli testni signal uz robusne postupke usrednjavanja neosjetljive na iznimke. Prilagoditi, optimirati i verificirati algoritam za izvedbu korištenjem frakcionalnog zapisa uzoraka i koeficijenta, te vrednovati razliku i točnost u odnosu na zapis pomičnog zareza. Referentne modele algoritma realizirati u Matlabu. Stvarni algoritam je potrebno realizirati u programskom jeziku C i/ili assembleru za vremenski kritične odsječke korištenjem evaluacijskog sustava TMS320VC5505 eZdsp. Evaluirati ostvarenu točnost i vrijeme izvođenja algoritma za različite parametre sustava. Istražiti i diskutirati primjenu ovog algoritma u kontekstu ultrazvučnog mjerenja brzine i smjera vjetra.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 17. lipnja 2016.

Mentor:
Prof. dr. sc. Davor Petrinović

Djelovođa:
Doc. dr. sc. Marko Subašić

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Sven Lončarić

Zahvaljujem se obitelji i prijateljima na neizmjernoj podršci i mentoru prof.dr.sc. Davoru Petrinoviću na strpljenju, razumijevanju i ukazanoj pomoći.

„ Dajte mi šest sati da srušim drvo

i provest ću prvih četiri oštreći sjekiru.“

– Abraham Lincoln

Sadržaj

Uvod

Pri komunikaciji između dva nezavisna sustava koji prenose poruke blok-po-blok načinom, potrebna je vremenska korelacija između poslanih i primljenih poruka. Sustav za prihvat poruka mora odrediti početak svakog bloka za ispravno tumačenje ulaznih podataka. Postupak sinkronizacije na temelju faznih odnosa između nekoliko kosinusnih signala, pogodan za našu primjenu, opisan je u radu Petrinović D., Sudarević D., *A robust synchronization method based on harmonic signals* [1].

Budući da sustavi odašiljanja i prijama djeluju asinkrono, stjecanje bloka počinje nakon proizvoljnog kašnjenja. Dakle, blok ima djelomične podatke iz oba signala, tako da se podaci iz bloka ne mogu pravilno tumačiti. Zbog periodičnosti signala sinkronizacije, vrijeme kašnjenja se može odrediti iz faznih odnosa sinkronizacije harmonijske komponente prisutne u prijemnom signalu, kao što će i biti objašnjeno te prikazano u ostvarenom algoritmu.

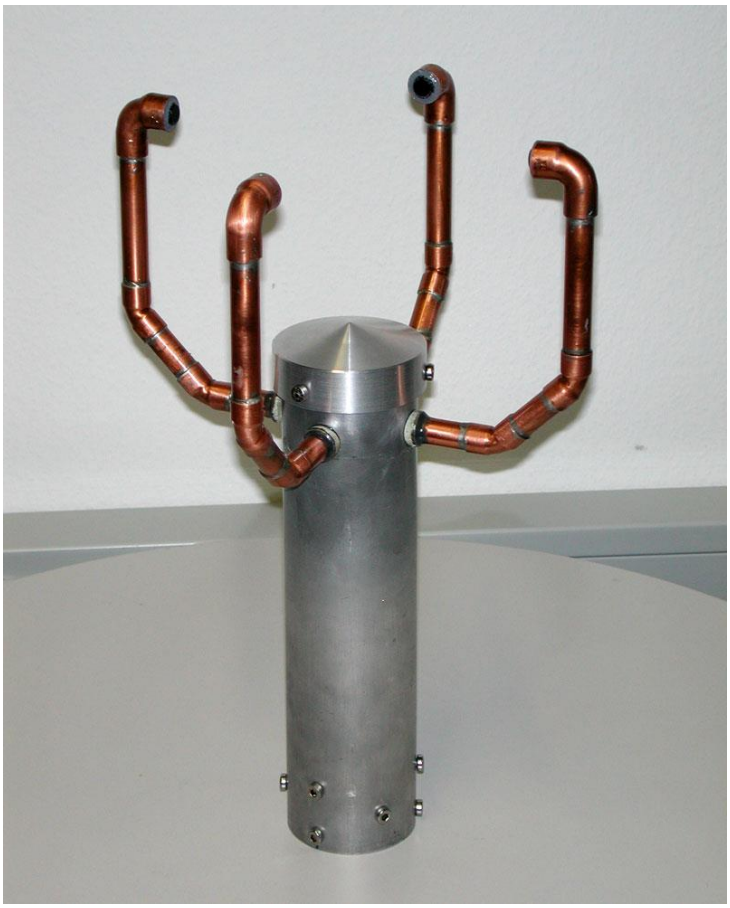
1. Ultrazvučni anemometar

Projektom četvorice studenata Fakulteta elektrotehnike i računarstva *Ultrazvučni anemometar* [2] udareni su temelji ovom radu. U spomenutom projektu, algoritam za više-harmonijsko određivanje vremena kašnjenja je primijenjen u MATLAB simulaciji ultrazvučnog anemometra.

Anemometar je uređaj koji mjeri brzinu i smjer vjetra. Mjerenje značajki vjetra je vrlo važno za industrijsku granu poput elektroenergetike, gdje se takva mjerenja koriste pri djelovanju vjetroelektrana. No mjerenje je važno i za praćenje prirodnih pojava u meteorološkom smislu. Postoji nekoliko tipova anemometara, među kojima razlikujemo mehanički, ultrazvučni, termički itd. Za razliku od često viđenog mehaničkog anemometra, ultrazvučni anemometar radi u svim vremenskim uvjetima te ima širi raspon detekcije brzine vjetra. Može se reći da je ultrazvučni anemometar napredniji uređaj za mjerenje brzine i smjera vjetra.

Razlog za odabir ultrazvuka umjesto zvuka iz čujnog područja jest njegova mala valna duljina, odnosno visoka frekvencija. Zvuk visoke frekvencije u prirodi nalazi manje šumova, dok 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 radi se o više puteva u istoj ravnini koji su važni za određivanje smjera vjetra. Vjetar paralelan zvuku na nekoj mjernoj osi povećava brzinu propagacije zvučnog vala. Mjerenje se može provesti odašiljanjem harmoničkog signala. Harmoničkom signalu se mjeri kašnjenje preko fazne razlike poslanog i primljenog signala.

Predajnici/prijemnici ultrazvučnog anemometra, prikazanog na Slici 1., se nalaze u vršcima ruka anemometra. Postavljen je NESW koordinatni sustav gdje 0° predstavlja sjever, 90° istok, 180° jug, a 270° zapad. Pri tlocrtnom pogledu na anemometar, jedna strana svijeta prikazuje jedan prijemnik. Pozitivna y-os je orijentirana od juga prema sjeveru, a pozitivna x-os je orijentirana od zapada prema istoku.



Slika 1. Ultrazvučni anemometar [3]

S obzirom na to da je geometrija mjerenja planarna kako bi se postigli što bolji rezultati mjerenja te što veća točnost čak i ako se uzme u obzir promjena brzine i smjera vjetra za vrijeme mjerenja, u MATLAB-u je simulirano mjerenje po svim putevima među prijemnicima, što uključuje osam kombinacija. Odašilju se dva signala odjednom sa dva susjedna predajnika. Svaki od tih signala se sastoji od tri superponirane frekvencijske komponente radi određivanja njihovog pomaka na prijemnicima.

Pri proračunu pomaka primljenog signala spektar se određuje diskretnom Fourierovom transformacijom (DFT) preko Goertzelova rekurzivnog algoritma. Takvom rješenju se pristupa jer direktna primjena DFT po matematičkoj definiciji zahtijeva programsku složenost $O(n^2)$, dok Goertzelov algoritam pojednostavljuje složenost na O(n*log(n)).

Nakon što se izračunaju faze primljenih signala, potrebno im je oduzeti inicijalne faze dobivene kalibriranjem. Potom, dobiveni rezultati pretvaraju se u pozitivnu frakcijsku fazu na intervalu od 0 do 1, gdje 1 predstavlja 2*π. Zatim se poziva funkcija koja računa vremena kašnjenja iz pomaka faza i popunjava tablicu kašnjenja. Potrebno je također pronaći standardnu devijaciju kašnjenja, pogrešku izraženu u uzorcima te na kraju odrediti ukupno vrijeme kašnjenja pomoću očekivanog kašnjenja i dobivenog izmjerenog kašnjenja.

Nakon izdvajanja vremena kašnjenja u individualne varijable, iz dobivenih se rezultata rekonstruira brzina i smjer vjetra po x, y i dijagonalnim osima. Konačna brzina i smjer vjetra izračuna se kombinacijom rezultata po pojedinim osima i kompenziranjem bočnog vjetra. Radi provjere rezultata određuje se i virtualna akustična temperatura zraka dobivena iz propagacijskih kašnjenja. Ukoliko je temperatura dovoljno blizu stvarnoj temperaturi zraka pri mjerenju, smatra se da je mjerenje valjano.

U ovom radu predstavit će se način kako da se u programskom jeziku C realizira algoritam za više-harmonijsko određivanje vremena kašnjenja, primijenjen upravo na ostvarenju ultrazvučnog anemometra. Iz ulaznih podataka, faza pomaka signala na prijemnicima, algoritam će izračunati vrijeme kašnjenja te odrediti totalno kašnjenje razlikom očekivanog i izmjerenog vremena kašnjenja. Napravit će se dva kôda, jedan će koristiti frakcionalni zapis uzoraka i koeficijenata, a drugi zapis pomičnog zareza te će se usporediti točnost njihovih izlaznih podataka u odnosu na rezultate MATLAB simulacije. U konačnici, navedeni kôdovi će se izvršiti na TMS320VC5505 procesoru za digitalnu obradu signala i ocijenit će se njihovo vrijeme izvođenja u odnosu na frekvenciju uzorkovanja od 96 kHz.

2. Algoritam za više-harmonijsko određivanje vremena kašnjenja

Algoritam pomoću kojeg će se odrediti vrijeme kašnjenja poslanih harmoničkih signala ostvaren je u programskom jeziku C.

Za potrebe usporedbe vremena izvođenja na procesoru za digitalnu obradu signala, načinjena su dva kôda, jedan je ostvaren uz *double* tip podataka, koji prikazuje realne brojeve, a drugi uz *integer* tip, odnosno cjelobrojne podatke. Pretvorba iz jednog u drugi tip prikaza podataka postignuta je frakcionalnim zapisom realnih brojeva. Frakcionalni zapis označava da realni broj ima točno određen broj znamenki prije i poslije decimalne točke. Jedan od mogućih frakcionalnih zapisa, i onaj koji će biti korišten, jest Q15 format, prema kojemu se realni brojevi pomnože s petnaestom potencijom broja 2 i zaokruže na najbliži cijeli broj [4].

Navedeni kôdovi su ostvareni za primjer mjerenja smjera i brzine vjetra između dva prijemnika i predajnika ultrazvučnog anemometra. Prvi signal se šalje putanjom sjever-jug, a drugi putanjom istok-zapad. U obzir se uzima i bočni vjetar, drugi signal s istočnog predajnika stiže i na južni prijemnik, dok prvi signal ima i komponentu koja sa sjevera dolazi na zapadni prijemnik.

Prikazat će se kôd sa zapisom pomičnog zareza, a u sljedećem poglavlju i njegova konverzija ka potpunom Q15 zapisu.

2.1. *Header* datoteka *constants.h*

Sve konstante i ulazni parametri koji su potrebni za pravilno funkcioniranje programa, preuzeti su iz *header* datoteke *constants.h*. *Header* datoteke su datoteke s ekstenzijom *.h* koje sadrže sve konstante, globalne varijable i slične podatke koji će se podijeliti među više izvornih datoteka realiziranih u programskom jeziku C.

Među tim konstantama jest i frekvencija uzorkovanja *fs* čija je vrijednost postavljena na 96000. Broj točaka diskretne Fourierove transformacije *N* postavljen je na brojku 242. Jedan blok podataka sastoji se od 3*N uzoraka, pri čemu 2*N označava broj uzoraka signala, dok je jedan N ostavljen za pauziranje između obrade podataka i akvizicije novog bloka. Zato konstanta *Na* predstavlja broj od 485 uzoraka signala kojima će se estimirati vrijeme kašnjenja. Konstante *N1* i *N2* određuju interval stabilnog dijela analiziranog signala. U MATLAB simulaciji ultrazvučnog anemometra za *N1* je dobivena brojka 109 i ona obilježava zadnju pojavu uzorka izravnog signala, a za *N2* brojka 296 obilježava prvi nestanak uzorka dijagonalnog signala.

U matricu *kas_oc*, koja prikazuje očekivano vrijeme kašnjenja, zapisane su vrijednosti nominalnog propagacijskog kašnjenja bez vjetra pri kalibracijskoj temperaturi te dijagonalnog propagacijskog kašnjenja bez vjetra pri kalibracijskoj temperaturi. Budući da se prvi signal šalje stazom sjever-jug, prvi član matrice jest *tauy*, odnosno propagacijsko kašnjenje po y-osi. Drugi član je *taud*, dijagonalno kašnjenje po stazi istok-jug. Treći član matrice jest *taux*, odnosno nominalno propagacijsko kašnjenje po x-osi, jer se drugi signal šalje putanjom istok-zapad. Posljednji član je *taud*, dijagonalno propagacijsko kašnjenje po putanji sjever-zapad.

Dvije ulazne matrice čijim će se podacima manipulirati su matrice estimiranog faznog kašnjenja *an1f* i *an2f*. Zapisane su u *double* formatu i sadrže po 485 redaka te 6 stupaca.

```
/*Prva konstelacija signala */
#define mi1_1 4
#define mi1_2 25

#define s1_1 mi1_1*mi1_2
#define s1_2 s1_1+1
#define s1_3 s1_1+mi1_1
#define p1_2 s1_1
#define p1_3 mi1_2

/* Druga konstelacija signala */
#define mi2_1 7
#define mi2_2 14

#define s2_1 mi2_1*mi2_2
#define s2_2 s2_1+1
#define s2_3 s2_1+mi2_1
#define p2_2 s2_1
#define p2_3 mi2_2
```

Slika 2. Mjerni signali s1 i s2

Na Slici 2., vidljivo je definiranje parametara dvaju mjernih signala *s1* i *s2*. Za prvu konstelaciju signala *s1_1*, *s1_2* i *s1_3* definira se polje *ss1* od tri člana, dok se za konstelaciju *s2_1*, *s2_2* i *s2_3* definira polje signala *ss2*.

Parametri su odabrani prema sinkronizacijskoj metodi predstavljenoj u radu [1]. Sinkronizacijski signal y(t) sastoji se od z kosinusnih signala s nultom fazom i istom amplitudom A_{sync}, definiran prema:

$$y(t) = \sum_{q=1}^{z} A_{sync} \cos(2\pi f_q t) \text{ , gdje je } f_q = \frac{1}{T} s_q \text{ , } s_q \in N \quad (1)$$

Frekvencije kosinusnih signala $sync_1$ do $sync_z$, su harmonički povezane, pri čemu je T duljina transmitiranih blokova. Prije prijenosa, signale s frekvencijama f_1 do f_Z treba eliminirati iz bloka podataka, i zamijeniti sinkronizacijskim signalom y(t). Cijeli brojevi s_q koji definiraju frekvencije f_q su odabrani kao:

$$s_1 = \prod_{i=1}^{z-1} \mu_i \text{ , } s_2 = s_1 + 1 \text{ , } s_q = s_1 + \prod_{i=1}^{q-2} \mu_i \text{ za } q = 3, \dots, z \quad (2)$$

gdje z-1 cijeli brojevi μ_i, { $\mu_i \in N \mid \mu_i > 1$} predstavljaju osnovu sinkronizacijske metode. Svaki par signala, $sync_1$ i $sync_n$, ima zajednički period duljine T_{n-1}, u kojem prvi signal ima p_n, a drugi p_{n+1} perioda. Brojevi p_n i zajednički periodi T_{n-1} zadaju se za z-1 parove signala:

$$p_n = \prod_{i=n-1}^{z-1} \mu_i \text{ , } T_{n-1} = T \frac{p_n}{s_1} \text{ , za } n = 2, \dots, z \quad (3)$$

2.2. Glavna funkcija programa

```
FILE *f = fopen("kas_tot.txt", "wb");

/* Izracun kasnjenja */
find_delay(&an1f[0][0], &kas_est[0][0], ss1, p1_2, p1_3);
find_delay(&an1f[0][3], &kas_est[0][1], ss2, p2_2, p2_3);
find_delay(&an2f[0][0], &kas_est[0][2], ss2, p2_2, p2_3);
find_delay(&an2f[0][3], &kas_est[0][3], ss1, p1_2, p1_3);
```

Slika 3. Poziv funkcije find_delay

Ulazni podaci u glavnu funkciju programa preuzimaju se iz datoteke *constants.h*, a izlazni podaci spremaju se u datoteku *kas_tot.txt* te su zapisani u binarnom obliku. Nakon što se inicijaliziraju varijable, poziva se funkcija *find_delay* (Slika 3.)., i to čak četiri puta, kako bi se procijenilo kašnjenje za sve kombinacije signala iz faznih odnosa oba primljena signala na svim frekvencijama komponenti. Funkciji se predaje pet argumenata – prvi argument jest matrica estimiranog faznog kašnjenja, drugi argument jest izlazna vrijednost funkcije, estimirano vrijeme kašnjenja *kas_est*, dok su posljednja tri argumenta oznake komponenata poslanog signala. Treba napomenuti kako funkcija *find_delay* ispuni jedan stupac od 485 vrijednosti vremena kašnjenja za ulazno fazno kašnjenje, a obrađuje matricu od 485 redaka i 3 stupaca faznog kašnjenja – u prvom pozivu računa se vrijeme kašnjenja za tri komponente direktnog signala *s1* koji putuje stazom sjever-jug, u drugom pozivu vrijeme kašnjenja se računa za dijagonalni signal istok-jug, u trećem za direktni signal *s2* istok-zapad, a u posljednjem pozivu za dijagonalni signal koji stiže na zapadni prijemnik sa sjevernog predajnika. Nakon završetka obrade podataka unutar četiri poziva funkcije, dobivena je matrica estimiranog vremena kašnjenja po svim stazama, *kas_est*, veličine 485 redaka i 4 stupaca.

No prije poziva funkcije *std_est*, potrebno je izuzeti moguće nestabilne uzorke od 485 uzoraka estimiranog kašnjenja po svakoj stazi, i zatim sortirati dobivenu matricu po stupcima. Zato se u matricu *helpm* zapisuju uzorci unutar intervala *N1* i *N2*, koji označavaju donju i gornju granicu stabilnih uzoraka analiziranog signala. Matrica *helpm* se zatim sortira po retcima, koristeći Shell sort postupak sortiranja [5]. Shellov sort je najstariji brzi algoritam za sortiranje, koji funkcionira kao modificirano sortiranje umetanjem. Izabire se inkrementalan slijed brojeva koji služe kao raskorak između elemenata polja koji se sortiraju. U ovom kôdu, koristi se Hibbardov slijed

koji za korak uzima potencije broja dva umanjene za jedan. U najgorem slučaju, njegova složenost iznosi $O(n^{3/2})$, dok prosječna složenost $O(n^{5/4})$ utvrđena simulacijom još uvijek nije teorijski dokazana.

Nakon sortiranja matrice *helpm* po retcima, ona se predaje funkciji *std_est* kao prvi argument, kao što je vidljivo i na Slici 4. Drugi argument jest multiplikator sigme, intervala unutar kojeg će se promatrati uzorci, a posljednji argument jest polje *ccmea* od četiri člana, u koje se zapisuju izlazne vrijednosti funkcije. Polje *ccmea* sadrži vrijednosti izmjerenog vremena kašnjenja za svaku od četiriju staza po kojima su poslani i primljeni signali.

```
for (k=0; k<4; k++){
    for (i=N1, j=0; i<=N2; i++, j++){
        helpm[k][j] = kas_est[i][k];
    }
    /* the following loops sort the array in ascending order */
    for (korak = (N2-N1+1) / 2; korak > 0; korak /= 2) {
        for (i = korak; i < (N2-N1+1); i++) {
            pom = helpm[k][i];
            for (j = i; j >= korak && helpm[k][j-korak] > pom; j -= korak) {
                helpm[k][j] = helpm[k][j - korak];
            }
            helpm[k][j] = pom;
        }
    }
}

/* Robusna ocjena kasnjenja i standardne devijacije */
std_est(&helpm[0][0], 2, &ccmea[0]);
```

Slika 4. Poziv funkcije std_est

Poslije poziva funkcija koje obavljaju većinu matematičkih operacija, još je potrebno izračunati vrijednost izlaznog polja *kas_tot*, čiji se elementi dobivaju razlikom očekivanog vremena kašnjenja stečenim simulacijom u MATLAB-u te izmjerenih vremena kašnjenja podijeljenima s frekvencijom uzorkovanja. Vrijednost cijelog polja sprema se u datoteku *kas_tot.txt*, čime se zaključuje ovaj program (Slika 5.).

```
for (i=0; i<4; i++){
    kas_tot[i]=kas_oc[1][i]-ccmea[i]/fs;
}

fwrite(kas_tot, sizeof(double), 4, f);
fclose(f);
```

Slika 5. Izlaz programa

2.3. Funkcija *find_delay*

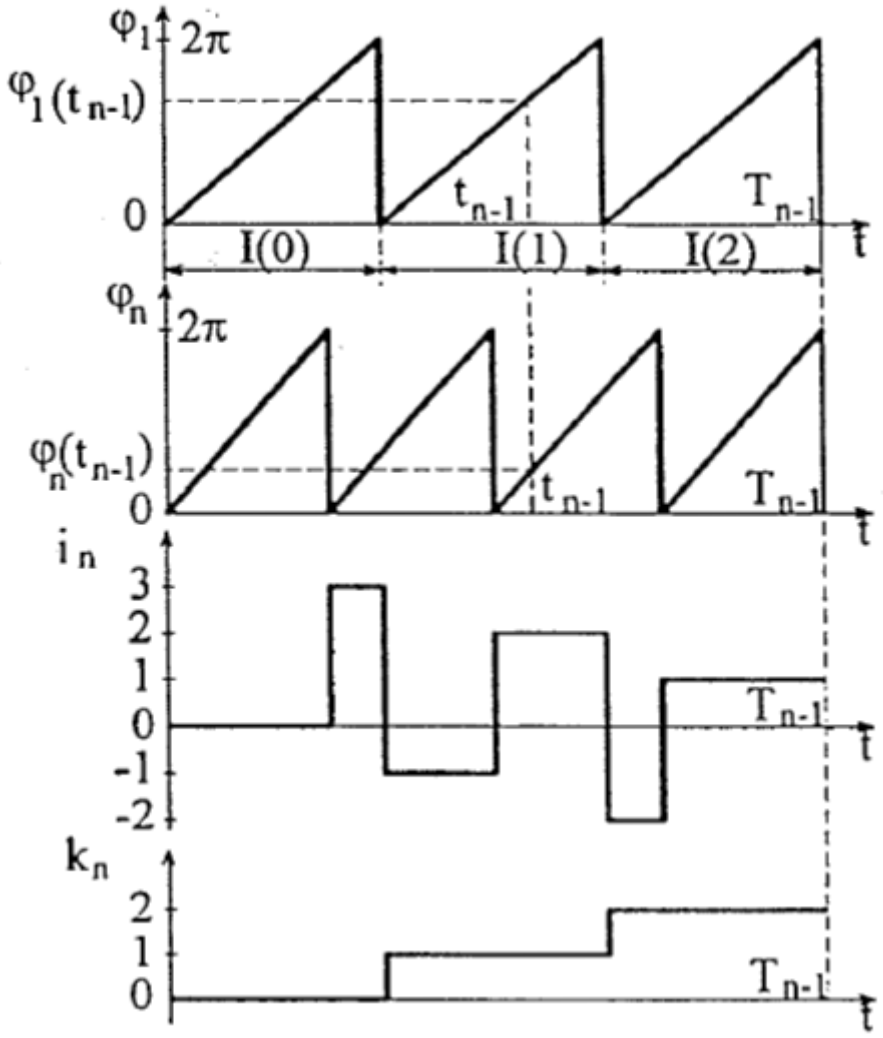
Funkcija *find_delay* poziva se iz glavnog programa četiri puta, a njezina služba jest da procijeni vrijeme kašnjenja svih kombinacija signala iz faznih pomaka oba primljena signala na svim frekvencijama komponenti.

Kao što je u uvodnom poglavlju napomenuto, postupak sinkronizacije na temelju faznih odnosa između dvaju kosinusnih signala, opisan je u radu Petrinović D., Sudarević D., *A robust synchronization method based on harmonic signals* [1]. U spomenutom radu navodi se da se izračun sinkronizacijskog kašnjenja temelji na fazama φ_1 do φ_z primljenih sinkronizacijskih signala $sync_1$ do $sync_z$. Za idealan prijenosni medij, svaka od faza je linearno ovisna o kašnjenju t, ali s različitim nagibom koji je definiran svojom frekvencijom, kao što je to što je prikazano na Slici 6. za faze φ_1 i φ_n. Faze φ_1 i φ_n su prikazane na njihovom zajedničkom periodu T_{n-1}, koji se zatim ponavlja s_1/p_n puta na cijelom bloku duljine T. Budući da sve faze dobivaju vrijednosti između 0 i 2π više puta na cijelom intervalu kašnjenja $t \in [0, T)$, nije moguće odrediti kašnjenje t iz samo jedne od faza. Međutim, kašnjenje t_{n-1} unutar podbloka T_{n-1} jedinstveno je definirano parom faza (φ_1, φ_n) u skladu sa sljedećim jednadžbama:

$$t_{n-1} = \frac{T_{n-1}}{p_n} k_n + \frac{T_{n-1}}{p_n} \frac{\varphi_1}{2\pi} \text{ , gdje je} \tag{4}$$

$$k_n = \begin{cases} -i_n \dots \dots \; i_n \leq 0 \\ p_n - i_n \dots \; i_n > 0 \end{cases} \quad \text{i} \quad i_n = \frac{1}{2\pi}((p_n + 1)\varphi_1 - p_n\varphi_n) \tag{5}$$

Broj k_n je indeks perioda $sync_1$, $I(k_n)$, koji sadrži kašnjenje t_{n-1}. Cijeli brojevi i_n i k_n prikazani su na trećem i četvrtom grafu na Slici 6., kao funkcija kašnjenja t. Budući da je $T_1 = T$, ukupno sinkronizacijsko kašnjenje koje će se odrediti, zapravo je kašnjenje t_1, koje se može izvesti iz samo dviju faza φ_1 i φ_2, prema (4) i (5). Ova tvrdnja vrijedi dok god je prijenosni medij idealan. Za realni prijenosni medij, faze φ_1 i φ_2 mijenjaju se zbog utjecaja medija, tako da se greška može pojaviti. U tom slučaju, može se koristiti metoda temeljena na više od dva sinkronizacijska signala, koja daje bolje rezultate.



Slika 6. Primjer sinkronizacije za $p_n = 3$

Budući da se procjene vremena kašnjenja triju konstelacija signala računaju analogno, prikazat će se metoda kako doći do estimacije kašnjenja *kas1_1*. Kombiniranjem prve i treće komponente ulaznog faznog pomaka *anf*, dolazi se do procjene kašnjenja *k1_3*, koja se zatim robusno estimira kako bi se dobila varijabla *k1_3r* (Slika 7.). Na isti način se kombiniraju prva i druga komponenta faznog pomaka te se dolazi do procjene kašnjenja *k1_2*, koja se dodatno uglađuje uz estimaciju *k1_r* i dobije se *k1_2r* procjena kašnjenja (Slika 8.). Ta se procjena koristi uz broj točaka diskretne Fourierove transformacije *N*, prvu komponentu faze *anf* i prvu komponentu signala *ss* kako bi se dobila vremenska funkcija procijenjenog kašnjenja *de1_1* (Slika 9.). Procjena konstantnog kašnjenja prve komponente signala *kas1_1* dobiva se oduzimanjem vremenske osi od *de1_1* i realnim ostatkom dijeljenja s *N*-om (Slika 10.).

```
i1_3[i]=(p_3+1)*anf[i*6]-p_3*anf[i*6+2];
k1_3[i]=-i1_3[i];
if (k1_3[i]<0)
    k1_3[i]=k1_3[i]+p_3;
k1_3r[i]=round(k1_3[i]);
```

Slika 7. Spajanje dviju komponenti

```
l1_2[i]= round((k1_2[i]-k1_3r[i])/p_3);
k1_2r[i]=k1_3r[i]+l1_2[i]*p_3;
```

Slika 8. Rafiniranje k1_2 estimacije korištenjem robusne estimacije k1_3r

```
de1_1[i]=(double)N/ss[0]*(k1_2r[i]+anf[i*6]);
```

Slika 9. Vremenska funkcija procjene kašnjenja izvedena iz prve komponente

```
kas1_1[i]=de1_1[i]-i;
kas1_1[i]=fmod(kas1_1[i],(double)N);
if (kas1_1[i]>=(N/2))
    kas1_1[i]=kas1_1[i]-N;
if (kas1_1[i]<=(-N/2))
    kas1_1[i]=kas1_1[i]+N;
```

Slika 10. Oduzimanje vremenske osi kako bi se procijenilo stalno kašnjenje

Opisani postupak ponavlja se dva puta, kako bi se procijenilo kašnjenje druge komponente signala *ss*, *kas1_2*, te kašnjenje treće komponente signala *ss*, *kas1_3*. Na samom kraju funkcije, ako su sve tri fazne greške jednake varijance, optimalna procjena kašnjenja *kas_est*, i izlazna vrijednost funkcije, može se pronaći kao ponderirana aritmetička sredina individualnih komponenti kašnjenja (Slika 11.).

```
kas1_est[i*4]=(kas1_1[i]*ss[0]*ss[0]+kas1_2[i]*ss[1]*ss[1]+
              kas1_3[i]*ss[2]*ss[2])/(ss[0]*ss[0]+ss[1]*ss[1]+ss[2]*ss[2]);
```

Slika 11. Izračun konačne procjene kašnjenja

2.4. Funkcija *std_est*

Funkcija *std_est* iz glavnog programa prima matricu kašnjenja *helpm* te radi robusnu *center-clipped* estimaciju za vrijednosti kašnjenja primjenom medijana i interkvartilnog raspona. *Center-clipped* estimacija znači da se pri procjeni u obzir uzimaju samo vrijednosti vremena kašnjenja unutar određenog intervala središnjih vrijednosti, koji je u ovom slučaju zadan umnoškom druge ulazne varijable *kap_est* i ocjene sigme kašnjenja. Za četiri staze mjerenja procjenjuje se vrijeme kašnjenja te se taj izračun sprema u polje *ccmea* od 4 člana.

```
for(i=0;i<4;i++){
    for (j=0;j<188;j++){
        red[j] = matrica[i*188+j];
    }
    median_val[i]=median(188,red);

    irq_val[i]=iqr(188,red)/irq_fact;

    mad_val[i]=1.5*mad(188,red);

    rob[i]=(irq_val[i]+mad_val[i])/2;
}
```

Slika 12. Određivanje medijana, IQR-a i MAD-a

Kao što je vidljivo iz Slike 12., matrica kašnjenja se obrađuje red po red, i svaki red se sprema u istoimeno polje *red* od 188 članova.

Varijabla *red* je sortirana još u glavnoj funkciji programa i, zajedno sa brojem njezinih elemenata, šalje se u funkciju *median* koja određuje sredinu distribucije [6]. Ako je broj elemenata reda paran, funkcija vraća aritmetičku sredinu dva središnja elementa. Ako je broj elemenata reda neparan, funkcija vraća središnji element. U varijablu *med_val* sprema se robusna ocjena prosječnog kašnjenja.

Zatim se isto polje *red* te broj članova polja šalje u funkciju *iqr* koja računa interkvartilni raspon spomenutog polja [7]. Postoji više načina određivanja interkvartilnog raspona, no u ovoj funkciji razmatrat će se najjednostavniji slučaj, gdje se raspon računa kao razlika trećeg i prvog kvartila elemenata polja. Treći kvartil označava element koji se nalazi na ¾ duljine reda, a prvi kvartil element na ¼ duljine reda. Varijabla *irq_val* predstavlja prvu robusnu ocjenu sigme kašnjenja,

a dobiva se kvocijentom izlazne vrijednosti funkcije *iqr* te faktora između interkvartilnog raspona i procjene sigme, *irq_fact*.

Kao posljednja statistička mjera disperzije, red od 188 uzoraka poslat će se u funkciju *mad* koja računa srednju apsolutnu devijaciju od aritmetičke sredine [8]. Varijabla *mad_val* prikazuje drugu robusnu ocjenu spomenute sigme kašnjenja.

Polje *rob*, aritmetička sredina dviju robusnih ocjena sigme kašnjenja, *irq_val* te *mad_val*, pomoći će korigirati ocjenu kašnjenja tako da se oko njega usrednje središnji uzorci distribucije pogreške unutar +/- kap_est*sigma.

Na Slici 13. je prikazano određivanje maske za vrijednosti kašnjenja oko medijana unutar intervala +/- kap_est*sigma. Svi uzorci koji se nalaze unutar tog intervala se pridodaju varijabli *suma*, koja se naposljetku podijeli s brojem prihvatljivih uzoraka i time se dobiva *center-clipped* procjena vremena kašnjenja, *ccmea*, neosjetljiva na iznimke.

```
for(i=0;i<4;i++){
    br=0;
    suma=0;
    for(j=0;j<188;j++){
        if(matrica[i*188+j]>=(median_val[i]-kap_est*rob[i])){
            if(matrica[i*188+j]<=(median_val[i]+kap_est*rob[i])){
                suma+=matrica[i*188+j];
                br++;
            }
        }
    }
    ccmea[i]=suma/br;
}
```

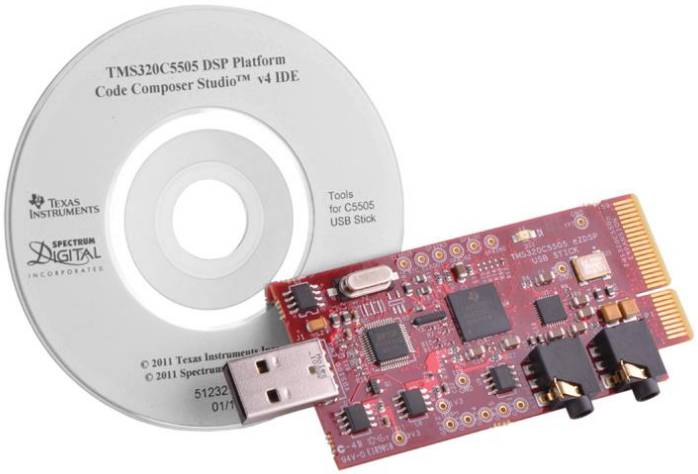
Slika 13. Određivanje maske za vrijednosti kašnjenja

3. Simulacija algoritma na procesoru za digitalnu obradu signala

Prethodno opisani program primijenit će se u sklopu ostvarenja ultrazvučnog anemometra, koji će se razviti na uređaju Raspberry Pi, a na kojeg će se spojiti procesor za digitalnu obradu signala (u daljnjem tekstu DSP – eng. *Digital Signal Processor*) tvrtke Texas Instruments. Algoritam za određivanje vremena kašnjenja pokrenut će se upravo na DSP-u pa ga zato treba evaluirati, utvrditi njegovo trajanje i odrediti je li moguće prikupljati te obrađivati podatke za mjerenje smjera i brzine vjetra u stvarnom vremenu.

3.1. Tehničke značajke

3.1.1. C5505 eZdsp USB Stick Development Tool



Slika 14. C5505 eZdsp™ USB Stick Development Tool [9]

TMDX5505eZDSP (Slika 14.) je razvojni DSP alat na USB pogon koji sadrži sav hardver i softver potreban za evaluaciju 16-bitnih DSP-ova najniže energije: TMS320C5504 i TMS320C5505. USB priključak pruža dovoljno snage za upravljanje C5505 procesorom tako da nije potreban nikakav vanjski izvor energije.

Ovaj alat pruža brzu i laku ocjenu naprednih mogućnosti C5505 i C5504 procesora. Ugrađen je XDS510 emulator za potpunu sposobnost otklanjanja neispravnosti (eng. *debug*) i podržava Code Composer Studio™ v4 integrirano razvojno okruženje (IDE) i eXpressDSP™ softver koji uključuje DSP / BIOSvkernel.

TMS320C5504 i TMS320C5505 su 16-bitni procesori najniže energije, što pomaže očuvati energiju na visokim razinama i omogućuje dulje trajanje baterije. Sa do 320KB memorije na čipu, višom razinom integracije (uključujući hardverski akcelerator za FFT računanja) od usporedivih uređaja po sličnim cijenama, C5504 i C5505 pružaju temelj za niz primjena pri obradi signala, poput diktafona, glazbenih instrumenata, prijenosnih medicinskih rješenja, ali i primjena za raznu drugu potrošačku elektroniku u industriji sigurnosnih aplikacija.

Programeri lako mogu razvijati TMS320C5504/5 DSP putem robusne i sveobuhvatne Code Composer Development Studio platforme Texas Instrumentsa, koja obuhvaća potpuno integrirano razvojno okruženje (IDE), efikasan optimirajući

C/C++ *compiler/assembler*, *linker*, *debugger*, integrirani CodeWright editor s CodeSense tehnologijom za brže stvaranje koda te vizualizaciju podataka.

Značajke TMDX5505eZDSP-a uključuju: razvojni DSP alat za C5505 procesor, TMS320C5505 DSP male snage i TLV320AIC3204 32-bitni programabilni stereo kodek.

Navedeni stereo kodek je za potrebe ovog rada vrlo važan jer ima digitalno-analogni i analogno-digitalni konverter koji može raditi na potrebnoj frekvenciji uzorkovanja od 96 kHz.

3.1.2. TLV320AIC3204 Stereo Audio CODEC

Slika 15. TLV320AIC3204 Stereo Audio CODEC [10]

TLV320AIC3204 (Slika 15.) je fleksibilan, stereo audio kodek niske snage i niskog napona s programabilnim ulazima i izlazima, fiksnim unaprijed definiranim blokovima obrade signala, integriranim PLL-om i fleksibilnim digitalnim sučeljem.

Značajke kodeka: stereo audio digitalno-analogni konverter (DAC) sa 100 dB odnosom signal-šum (SNR), 4.1mW stereo 48kbps reprodukcija, stereo audio analogno-digitalni konverter (ADC) sa 93 dB SNR, 6.1mW stereo 48 kbps ADC snimanje, opsežne mogućnosti obrade signala, programabilna fazno kontrolirana petlja (PLL).

Uređaj TLV320AIC3204 obuhvaća operacije od 8 kHz mono do 192kHz stereo snimanja, a sadrži programabilne konfiguracije ulaznog kanala koje pokrivaju asimetrične i diferencijalne postavke, kao i miješanje ulaznih signala. Put reprodukcije nudi blokove za obradu signala za filtriranje i efekte, a podržava fleksibilno miješanje DAC-a i analognih ulaznih signala, kao i programabilno upravljanje glasnoće.

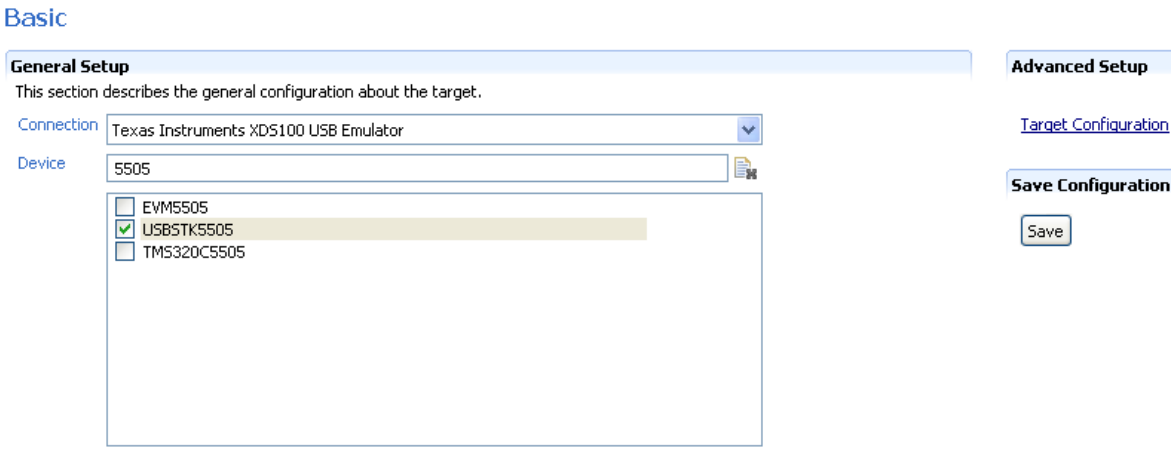
3.2. Pokretanje C programa u razvojnom okruženju Code Composer Studio

Code Composer Studio je integrirano razvojno okruženje (IDE) koje podržava mikrokontrolere i ugradbene procesore iz portfelja Texas Instrumentsa [11]. Code Composer Studio sadrži skup alata koji se koriste za razvoj i ispravljanje neispravnosti ugradbenih aplikacija. To uključuje optimirajući C/C ++ kompajler, editor izvornog koda, okruženje za izgradnju projekta, program za pronalaženje pogrešaka, *profiler*, i mnoge druge značajke. Intuitivan IDE pruža jedinstveno korisničko sučelje koje vodi kroz svaki korak toka razvoja aplikacija. Code Composer Studio kombinira prednosti Eclipse softvera s ugrađenim naprednim sposobnostima ispravljanja pogreški Texas Instrumentsa rezultirajući u uvjerljivoj razvojnoj okolini bogatoj značajkama za programere.

Za potrebe ovog rada, instalirat će se četvrta verzija aplikacije Code Composer Studio (CCS), dostupna za operacijske sustave Windows XP i Vista. Pri prvom pokretanju CCS-a, potrebno je odrediti lokaciju radnog prostora te aktivirati besplatnu 30-dnevnu licencu. Nakon što su obavljene spomenute tehnikalije, CCS se treba spojiti na VC5505 eZDSP USB Stick koji se priključuje u jedan od USB pogona osobnog računala ili laptopa.

U CCS v4 prozoru, treba kliknuti na izbornik „Target“ i izabrati opciju „New Target Configuration File“. Potom će se prikazati prozor „New Target Configuration“ gdje će se upisati ime datoteke.

Otvorit će se „Basic" prozor za postavljanje konfiguracije u kojemu je potrebno odabrati „Texas Instruments XDS100 USB emulator“ iz „Connection“ izbornika, zatim izabrati „USBSTK5505" s popisa i klikom na gumb „Save“ spremiti konfiguraciju (Slika 16.).



Slika 16. Odabir Target konfiguracije

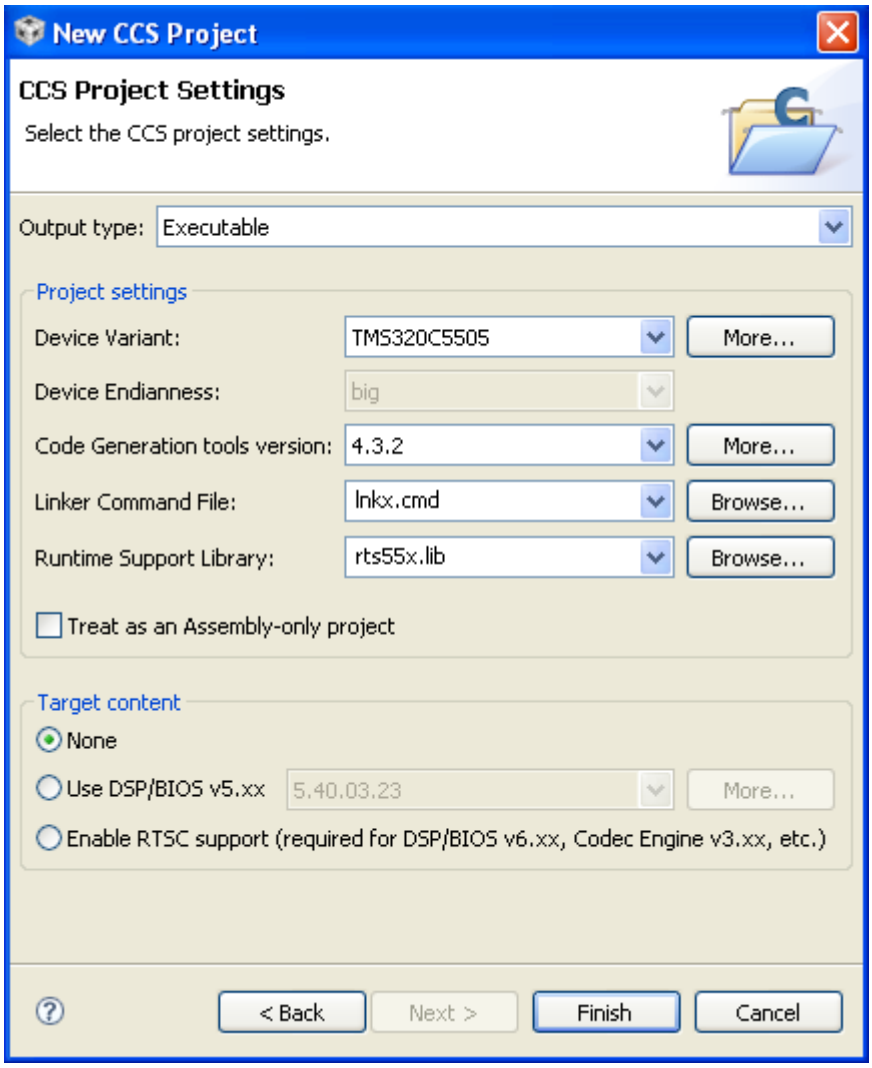
Klikom na izbornik „View“ i odabiranjem „Target Configurations“ izložit će se konfiguracije koje su izgrađene ili uvezene. Nova kartica s oznakom „Target Configurations“ postat će dostupna u prozoru CCS, unutar koje treba proširiti „User Defined“ mapu, kliknuti desnom tipkom miša na novu konfiguraciju koja se stvorila i kliknuti na „Launch Selected Configuration“.

Klikom na „View -> Debug“ pa „Target -> Connect Target“. CCS će se pokušati spojiti na VC5505 eZDSP USB Stick i pokrenuti GEL datoteku. Nakon što se uspostavi veza, a GEL datoteka izvrši, prozor konzole će ispisati poruku „Target Connection Complete“ (Slika 17.). Bitno je napomenuti da se ispisuje i poruka o frekvenciji na kojoj djeluje DSP, a ona iznosi 100 MHz.

```
Configuring PLL (100.00 MHz).

PLL Init Done.
Target Connection Complete.
```

Slika 17. Uspostavljanje veze s DSP-om

Odabirom izbornika „File“ pa opcije „New -> CCS Project“ otvorit će se prozor unutar kojeg se mogu izabrati detaljne postavke (Slika 18.). Klikom na „Finish“ kreira se novi projekt vidljiv pod karticom „C/C++ Projects“. Desnim klikom miša na mapu aktivnog projekta i odabirom „Add Files to Project…“ dodaju se izvorne i *header* datoteke napisane u programskom jeziku C, u čijim kôdovima je sadržan prethodno opisan algoritam za više-harmonijsko određivanje vremena kašnjenja.



Slika 18. Postavke CCS projekta

Klikom na „Project -> Build Active Project“, izgradit će se trenutni projekt i ispisati poruka o potvrdi u prozoru konzole. Pri ponovnom pogledu na karticu „C/C++ Projects“ te proširenjem mape „binaries“, primijetit će se novonastala *.out* datoteka. Desnim klikom na navedenu datoteku i odabirom „Load Program“, unijet će se projekt koji je spreman za pokretanje na procesoru za digitalnu obradu signala. Klikom na „Target -> Run“ program se izvršava.

3.3. Izvršavanje tri programa s različitim formatom zapisa

Prvi program koji će se izvršiti unutar platforme CCS v4 je zapisan u formatu s pomičnim zarezom. Iako je spomenuti program zbog zapisa u realnim brojevima izrazito precizan, njegovo vrijeme izvođenja je neprihvatljivo za obradu podataka u stvarnom vremenu.

Zbog toga je načinjen program napisan dijelom u Q15 frakcionalnom zapisu, a dijelom je ostavljen u zapisu pomičnog zareza. Napravljene promjene u kôdu neće značajno utjecati na preciznost, dok će vrijeme izvođenja biti znatno kraće. No tijekom izvršavanja programa korak po korak, i dalje je vidljivo znatno trošenje ciklusa procesora od strane preostalih naredbi zapisanih u *double* formatu.

Prva sporna naredba ponavlja se tri puta pri određivanju vremenske funkcije procjene kašnjenja. Potrebno je izmijeniti redoslijed računskih operacija kako bi se izbjeglo pretvaranje iz cijelih u realnih brojeva pa natrag u cijele brojeve (Slika 19.).

```
/*Prvotna verzija*/
de1_1[i]=(int)((float)N/ss[0]*(k1_2r[i]+anf[i*6]));

/*Promjena*/
de1_1[i]=N*(k1_2r[i]+anf[i*6])/ss[0];
```

Slika 19. Dijeljenje unutar određivanja vremenske funkcije

Druga sporna naredba pojavljuje se pri spremanju izlaznih vrijednosti funkcije *find_delay* u varijablu *kas1_est*. Budući da umnožak tri broja prikazanih u varijablama *kas_1*, *kas_2* te *kas_3* može poprimiti vrijednosti veće od maksimalne dopuštene vrijednosti koju može prikazati *int* zapis, te se varijable pretvaraju u realne brojeve pa se poslije dijeljenja unutar *kas1_est* vraćaju u cjelobrojni zapis. Navedeno manevriranje se može zaobići time da se u *header* datoteku unaprijed zapišu varijable *ss10, ss11, ss12, ss20, ss21* i *ss22* koje zastupaju udio pojedine komponente signala u ponderiranoj aritmetičkoj sredini. Odnosno, riječ je o kvocijentu jedne komponente signala te zbroja kvadrata triju komponenata. Navedene varijable se također množe sa osmom potencijom broja dva kako bi se dobiveni brojevi manji od jedan mogli zapisati u cjelobrojnom formatu. Na samom kraju određivanja varijable *kas1_est*, ona se podijeli s brojem 256 kako bi se poništila ta unesena promjena (Slika 20.).

```
/*Prvotna verzija*/
kas_1 = ((float)kas1_1[i])*ss[0]*ss[0];
kas_2 = ((float)kas1_2[i])*ss[1]*ss[1];
kas_3 = ((float)kas1_3[i])*ss[2]*ss[2];

kas1_est[i*4]=(int)((kas_1+kas_2+kas_3)/
                    (ss[0]*ss[0]+ss[1]*ss[1]+ss[2]*ss[2]));

/*Promjena*/
if(ss[0] == 100){
    kas_estl = kas1_1[i]*ss10+kas1_2[i]*ss11+kas1_3[i]*ss12;
} else {
    kas_estl = kas1_1[i]*ss20+kas1_2[i]*ss21+kas1_3[i]*ss22;
}
kas1_est[i*4] = kas_estl/256;
```

Slika 20. Spremanje vrijednosti u kas1_est

Posljednja naredba zapisana u formatu realnih brojeva jest ona koja računa *kas_tot*. Vrijednosti varijable *kas_oc* manje su od jedan pa je i njih moguće pomnožiti s potencijom broja dva kako bi se zapisali u cjelobrojnom formatu, umjesto u onome realnih brojeva. Zbog što veće točnosti, odlučeno je pomnožiti ih sa tridesetiprvom potencijom broja dva pa će izlaznu varijablu *kas_tot* u nekoj daljnjoj obradi biti potrebno podijeliti s navedenom potencijom broja dva (Slika 21.).

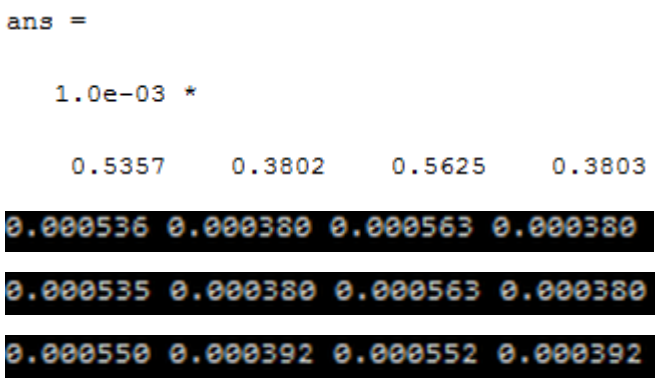
```
/*Prvotna verzija*/
for (i=0; i<4; i++){
    kas_tot[i]= kas_oc[1][i]-ccmea[i]/32768.0/fs;
}

/*Promjena*/
for (i=0; i<4; i++){
    kas_tot[i]=kas_oci[1][i]-rounddiv(ccmea[i],fs)*65536;
}
```

Slika 21. Izračun ukupnog kašnjenja

Izvršavanjem kôda u potpunom frakcionalnom zapisu, primjetno je njegovo kratko vrijeme izvršavanja, no i unos određene nepreciznosti u konačnom rješenju. Vrijeme izvođenja i preciznost programa bit će raspravljeni u sljedećem potpoglavlju.

3.4. Usporedba preciznosti i vremena izvođenja programa

Izlazni podaci iz C programa spremaju se u varijablu *kas_tot*. Njezine ciljane vrijednosti dobivene su MATLAB simulacijom. Iz Slike 22., vidljivo je određeno odstupanje izlaznih vrijednosti dobivenih izvršavanjem triju programa od traženog rješenja. U slučaju programa zapisanih s pomičnim zarezom te djelomičnom frakcionalnom zapisu, spomenuta su odstupanja zanemariva, dok u slučaju potpunog frakcionalnog zapisa rješenja nisu sasvim prihvatljiva.



Slika 22. Od vrha prema dnu: rezultati MATLAB simulacije, prvog, drugog te trećeg algoritma

Pri izvršavanju programa na procesoru za obradu digitalnih signala, prvotno je nužno uključiti sat koji će izmjeriti broj potrošenih ciklusa između početne i završne linije kôda odabirom „Target -> Clock -> Enable“ [12]. Broj ciklusa potrebnih za izvršavanje svakog od triju programa prikazan je na sljedećoj slici:

Slika 23. Broj ciklusa za prvi (podijeljen u dva dijela), drugi i treći algoritam

Pretvorba iz ciklusa u stvarno vrijeme, moguća je po sljedećoj formuli:

$$t = DCLK * 1/CLK \quad (6)$$

DCLK označava potreban broj ciklusa za izvršavanje programa, a CLK frekvenciju na kojoj radi procesor. Po Slici 17., VC5505 eZDSP USB Stick radi na 100 MHz.

Tablica 1. Maksimalno odstupanje od rezultata MATLAB simulacije i vrijeme izvođenja programa

	Maksimalno odstupanje	Vrijeme izvođenja
Prvi program	0.19%	831.51 ms
Drugi program	0.37%	284.69 ms
Treći program	3.16%	20.46 ms

Iz Tablice 1. mogu se izvesti nekoliko opažanja. Kôdovi napisani u zapisu pomičnog zareza i djelomičnom frakcionalnom formatu zapisa su dovoljno precizni da bi ih se moglo koristiti u stvarnim mjerenjima. S druge strane, njihovo vrijeme izvođenja je daleko dulje od 7.56 ms, koliko je potrebno za akviziciju bloka podataka. Znači da oba algoritma ne mogu obrađivati podatke u stvarnom vremenu.

Za razliku od njih, vrijeme izvođenja trećeg programa, u potpunom frakcionalnom zapisu, je podosta kraće, no i dalje nije manje od vremena akvizicije bloka podataka te se ni on ne može izvoditi u stvarnom vremenu. Njegova preciznost je nezadovoljavajuća, maksimalno odstupanje vrijednosti izlaznih podataka od vrijednosti dobivenih MATLAB simulacijom iznosi 3.16%. Ali ako se pri određenoj obradi podataka više cijeni brzina u odnosu na određenu nepreciznost, ovaj program jest najbolji izbor od prikazanih. No algoritam opisan u ovome radu vrši mjerenje samo za jedan par signala poslanih s dva predajnika na dva prijemnika. U ostvarenju ultrazvučnog anemometra, vrši se osam mjerenja, za sve moguće parove signala, kako bi se u potpunosti umanjio utjecaj iznimki. Ukoliko bi se potpuni frakcionalni algoritam pozvao osam puta, njegovo bi vrijeme izvođenja iznosilo 163.68 ms.

Zaključak

Na samome kraju, algoritam za više-harmonijsko određivanje vremena kašnjenja ostvaren u potpunom frakcionalnom zapisu pokazao se najbržim za obradu bloka podataka. Ipak, njegova određena nepreciznost izlaznih rješenja ostavlja prostora optimizaciji kôda u budućnosti. Jedna od mogućnosti poboljšanja kôda jest i iskorištavanje potpunog opsega cjelobrojnog formata pri zapisu svake varijable, kako bi se maksimalno povećala njihova točnost. Nadalje, potrebno je i ubrzati izvođenje kôda kako bi se algoritam mogao izvesti u stvarnom vremenu. Funkcija *find_delay* troši daleko najviše ciklusa procesora od ostatka programa i nju bi se moralo pojednostavniti. No taj posao optimizacije ostavlja se za neki budući poduhvat.

Ovaj algoritam ključan je dio u primjeni ultrazvučnog mjerenja brzine i smjera vjetra. Riječ je o posljednjem dijelu kôda koji obrađuje cjelokupni primljeni blok podataka. Budući da blok ima djelomične podatke iz oba signala, podaci iz bloka se ne mogu pravilno tumačiti bez implementacije ovog algoritma koji će konačno vrijeme kašnjenja odrediti iz faznih odnosa sinkronizacije harmonijske komponente prisutne u prijemnom signalu. Algoritam iz četveroznamenkastog broja uzoraka određuje konačnu estimaciju prosječnog kašnjenja za cijeli signal uz robusne postupke usrednjavanja neosjetljive na iznimke te time osigurava puno jednostavnije daljnje rukovanje podacima.

Literatura

[1] Petrinović D., Sudarević D., *A robust synchronization method based on harmonic signals*, Davos 1993.

[2] Radman B., Borić L., Murn L., Malić D., *Ultrazvučni anemometar*, Zagreb 2016.

[3] http://www.dl1glh.de/ultrasonic-anemometer.html

[4] https://en.wikipedia.org/wiki/Q_(number_format)

[5] https://en.wikipedia.org/wiki/Shellsort

[6] https://hr.wikipedia.org/wiki/Medijan

[7] https://en.wikipedia.org/wiki/Interquartile_range

[8] https://en.wikipedia.org/wiki/Average_absolute_deviation

[9] http://www.ti.com/tool/TMDX5505EZDSP

[10] http://www.ti.com/product/TLV320AIC3204

[11] http://www.ti.com/tool/ccstudio

[12] http://processors.wiki.ti.com/index.php/Profile_clock_in_CCS

Sažetak

Izvedba algoritma za više-harmonijsko određivanje vremena kašnjenja

Algoritam za više-harmonijsko određivanje vremena kašnjenja određuje vrijeme kašnjenja iz faznih odnosa sinkronizacije harmonijske komponente prisutne u prijemnom signalu. Budući da sustavi odašiljanja i prijama ultrazvučnog anemometra djeluju asinkrono, svaki blok podataka ima djelomične podatke iz oba signala, tako da je izvedba algoritma za više-harmonijsko određivanje vremena kašnjenja ključna u kontekstu ultrazvučnog mjerenja brzine i smjera vjetra. Primjer algoritma ostvaren u programskom jeziku C prikazan je korištenjem frakcionalnog zapisa i zapisa pomičnog zareza. Algoritam realiziran u frakcionalnom zapisu pokrenut na evaluacijskom sustavu TMS320VC5505 eZDSP, ima kraće vrijeme izvođenja, ali manju točnost, u odnosu na zapis pomičnog zareza.

Ključne riječi

ultrazvučni anemometar, fazni pomak, estimirano vremensko kašnjenje, frakcionalni zapis, procesor za digitalnu obradu signala

Abstract

Implementation of a Multi-Harmonic Time Delay Estimation Algorithm

Multi-harmonic time delay estimation algorithm determines the time delay from the phase relations of the synchronization harmonic components present in the received signal. Since the transmitting and the receiving systems of the ultrasonic anemometer are operating asynchronously, each block of data has partial data from both signals, so the implementation of multi-harmonic timing delay algorithm is key in the context of ultrasonic measurements of speed and wind direction. An example of the algorithm realized in the C programming language is presented using the fractional format and floating point format. The algorithm implemented in fractional format running on TMS320VC5505 eZDSP evaluation system has a shorter runtime, but is less accurate, compared to the floating-point notation.

Keywords

ultrasonic anemometer, phase shift, time delay estimation, fractional format, digital signal processor