SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD 1554.

Dizajn i FPGA implementacija jednostavnih FIR filtara bez upotrebe množila

Martin Sertić

Zagreb, veljača 2018.

Sadržaj

1 Uvod

U današnje vrijeme filtri su sastavni dio svake aplikacije za digitalnu obradu signala. Sve se više traže učinkovite implentacije sustava za obradu signala od kod kojih se zahtijeva velika brzina rada. To vodi na filtre s malom složenosti i malog reda. Filtri malog reda i male složenosti postižu se strukturama bez množila gdje su koeficijenti izraženi kao sume potencija broja dva. Te strukture postižu se upotrebom posmaka i zbrajala. Međutim dodatna ušteda se postiže korištenjem samo posmaka. To vodi na koeficijente koji su izraženi kao predznačene potencije broja dva. Nadalje, za postizanje većih gušenja preferiraju se FIR filtri s neparnim brojem koeficijenata, tako da će isti biti razmatrani u ovom radu.

Biti će razmatrane dvije klase FIR filtara s linearnom fazom. To su niskopropusni FIR filtri te kompenzacijski filtri za CIC decimatore. Za potrebu FPGA implementacije i same provjere ispravnosti rada filtara koristit će se razvojna pločica Zedboard koja je dio Zynq serije 7. Na procesorskom dijelu Zedboarda bit će podignut linux operativni sustav prolagođen za rad s razvojnom pločicom. Filtri će biti implementirani u FPGA programabilnom dijelu pločice koja će komunicirati s procesorskim sustavom u linux okruženju. Izprocesorskog dijela ćemo slati ulaznu signale te zabilježiti izlaze u tekstualne datoteke kako bismo mogli provjeriti ispravnost rada.

2 FIR filtri

2.1 Osnovne značajke filtara

Filtri su linearni vremenski nepromijenjivi sustavi za propuštanje određenih frekvencija. Filtri se najčešće koriste za:

- **Odvajanje signala** koje se koristi u slučajevima kad je željeni signal pomiješan s interferencijama, šumovima i ostalim signalima. Tipičan primjer toga je uređaj koji mjeri signal otkucaja srca fetusa. Taj signal pomiješan je sa signalom disanja i otkucaja srca majke te je potrebno koristiti filtar kako bismo odvojili signale te ih mogli individualno analizirati.
- **Restauraciju signala**. Signal je potrebno restaurirati ukoliko je došlo do nekih izobličenja kod npr. snimanja zvuka mobitelom. Svi okolni signali kao što su zvukovi što stvaraju auti u prometu, zvukovi koji stvaraju ljudi u blizini te životinje, će utjecati na snimku koju je moguće filtrirati kako bismo dobili zapis koji što bolje predstavlja zvuk koji je bio na snimanju.

Idealni filtar propušta komponente signala određenih frekvencija bez ikakvog prigušivanja dok komponente na ostalim frekvencijama idealno prigušuje, odnosno blokira. Frekvencijska karakteristika prema tome ima vrijednost jedan ili nula. Područje frekvencije gdje frekvencijska karakteristika ima vrijednost jedan naziva se propusni pojas w_p, dok je područje gdje je frekvencijska karakteristika jednaka nula pojas gušenja w_s. S obzirom na to digitalne filtre možemo podijeliti na četiri osnovna tipa:

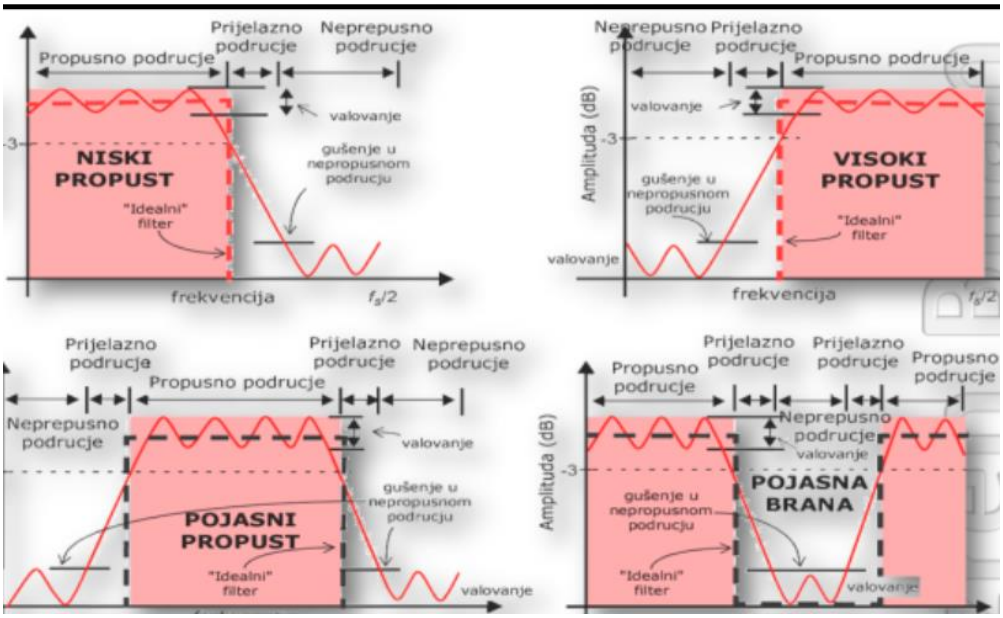
- Niskopropusni
- Visokopropusni
- Pojasno propusni
- Pojasno blokirni

Područje gušenja i propuštanja u realnom svijetu nikada neće odgovarati idealnoj karakteristici. Uglavnom signal oscilira oko jedinice u propusnom području odnosno nule u području gušenja. Te oscilacije (engl. ripple) opisane su sljedećim jednadžbama:

$$valovitost\ u\ području\ propuštanja = -20log_{10}\left(\frac{1+\delta_p}{1-\delta_p}\right) \quad (2.1)$$

$$atenuacija\ u\ području\ gušenja = -20log_{10}\delta_s \quad (2.2)$$

Vrlo bitan korak pri razvoju digitalnih filtra je određivanje ostvarive prijenosne karakteristike, koja će aproksimirati željenu frekvencijsku karakteristiku, koju možemo vidjeti na slici 2.1.



Slika 2.1. Amplitudno-frekvencijske karakteristike po tipovima filtra

Filtre možemo realizirati analogno i digitalno. Prednosti digitalnih filtra su brojne, od boljih frekvencijskih karakteristika, fleksibilnosti, neosjetljivosti na utjecaje okoline pa se danas razvojem tehnologije DSP-ova (engl. Digital Signal Processora) digitalni filtri široko upotrebljavaju. Analogni filtri su jeftini, brzi i imaju vrlo veliko dinamičko područje amplitude i frekvencije.

2.2 Prijenosna funkcija FIR filtra

Kod digitalnih filtara karakteristika je dana ovisnošću između ulaza i izlaza. Ulazni niz x[n] i izlazni niz y[n] povezani su jednadžbom diferencija sa konstantnim koeficijentima. Jednadžba u općem obliku poprima izgled:

$$y[n] = \sum_{k=0}^{M} a_k * x[n-k] - \sum_{k=1}^{N} b_k * y[n-k] \tag{2.3}$$

Nizovi a_k i b_k karakteriziraju dani filter i nazivaju se koeficijentima filtra. Ako primijenimo z-transformaciju na lijevu i desnu stranu relacije dobivamo prijenosnu funkciju H(z):

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} a_k\, z^{-k}}{1 + \sum_{k=0}^{M} b_k z^{-k}} \tag{2.4}$$

Ako za b_k uvrstimo nulu, diferencijalna jednadžba se svodi na:

$$y[n] = \sum_{k=0}^{M} a_k * x[n-k] \tag{2.5}$$

A prijenosna funkcija na:

$$H(z) = \sum_{k=0}^{M} a_k\, z^{-k} \tag{2.6}$$

Iz relacije slijedi da izlazni uzorak ovisi o ulaznom signalu koji su prethodili promatranom trenutku, a ne ovisi o uzorcima izlaznog signala. Očigledno na osnovi relacije ovakvi filtri imaju konačan impulsni odziv duljine *M* pa zbog toga takve filtre nazivamo filtrima s konačnim impulsnim odzivom, odnosno FIR (engl *Finite Impulse Response*) filtrima.

FIR filtri imaju nekoliko korisnih prednosti zbog kojih se smatraju boljima od IIR filtra, od kojih su neke:

- ne zahtijevaju povratnu vezu
- stabilni su
- mogu lagano biti dizajnirani da imaju linearnu fazu
- jednostavni su za implementaciju

2.3 CIC filtri

CIC (engl. *Cascaded-Integrator-comb*) decimacijski filtri su jednostavni filtri bez množila koji omogućuju velika gušenja. Međutim ta velika gušenja sa sobom povlače lošu karakteristiku u propusnom pojasu. CIC decimator prvog reda predstavlja usrednjivač uzoraka koji se može opisati sljedećom jednadžbom diferencija:

$$y(k) = \frac{x(k) + x(k-1) + \cdots + x(k-R+1)}{R} \tag{2.7}$$

Primijenimo li Z transformaciju na jednadžbu 2.7 dobivamo:

$$Y(Z) = \frac{1}{R}\sum_{k=0}^{R-1} z^{-k} X(z) \tag{2.8}$$

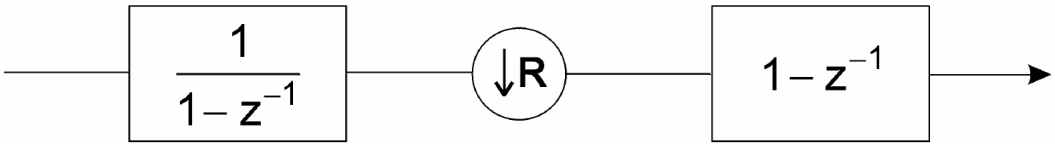
Već iz izraza 2.8 možemo vidjeti da je odziv usrednjivača konačnog trajanja što upućuje da je usrednjivač FIR filtar. Nadalje iz jednadžbe dobivamo prijenosnu funkciju oblika:

$$Y(Z) = \frac{1}{R}\sum_{k=0}^{R-1} z^{-k} \tag{2.9}$$

Ako primijenimo izraz za sumu, prijenosna funkcija CIC filtra prvog reda poprima oblik kao na sljedećoj jednadžbi :

$$Y(Z) = \frac{1}{R}\frac{1 - z^{-R}}{1 - z^{-1}} \tag{2.10}$$

Realizacija CIC decimatora prvog reda ima oblik kao na slici 2.2.

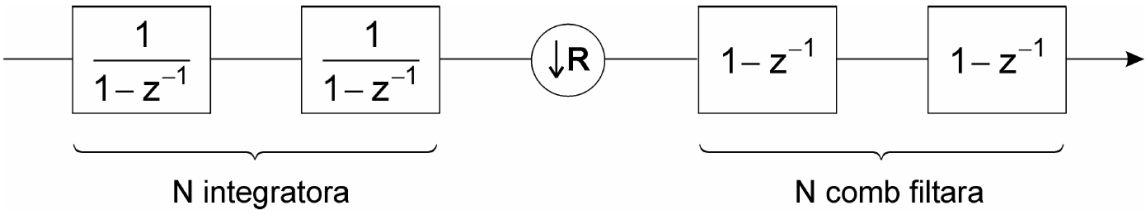


Slika 2.2. CIC decimator prvog reda

Ako više takvih CIC filtara prvog reda spojimo u kaskadu dobiva se CIC decimator N-tog reda koji je opisan prijenosnom funkcijom:

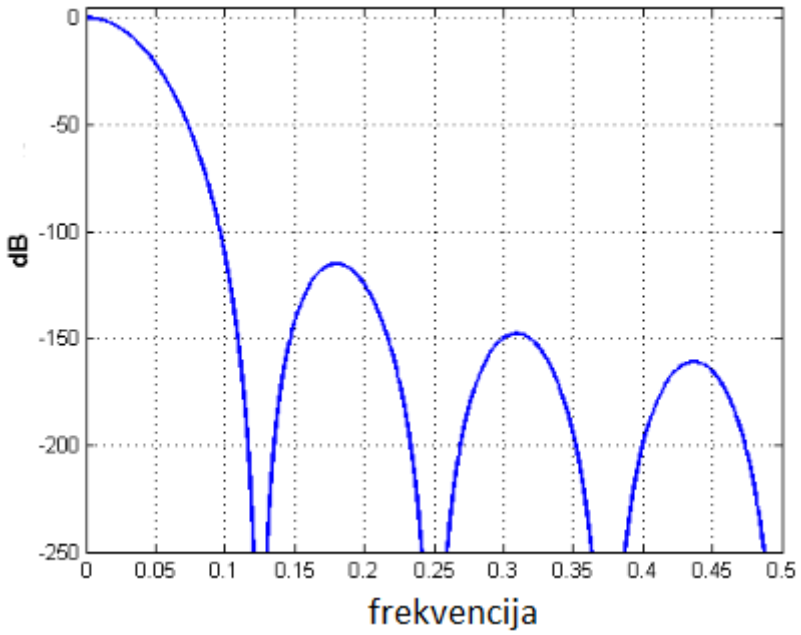
$$H(Z) = \left(\frac{1 - Z^{-R}}{1 - Z^{-1}}\right)^{N} \tag{2.11}$$

Iz čega proizlazi da CIC decimator N-tog reda poprima oblik koji prikazuje slika 2.3.



Slika 2.3. CIC decimator N-tog reda

Budući da CIC decimatorski filtri imaju velike propade u passbandu kao što možemo vidjeti na slici 2.4, potrebno ih je smanjiti. Postoji nekoliko struktura za smanjivanje propada u propusnom dijelu frekvencije od kojih je najpoznatija FIR filtar pod nazivom kompenzator. Kompenzator se spaja s CIC filtrom u kaskadu i pošto je CIC filtar jednostavne strukture bez množila preporuča se da i kompenzator bude takvog oblika.



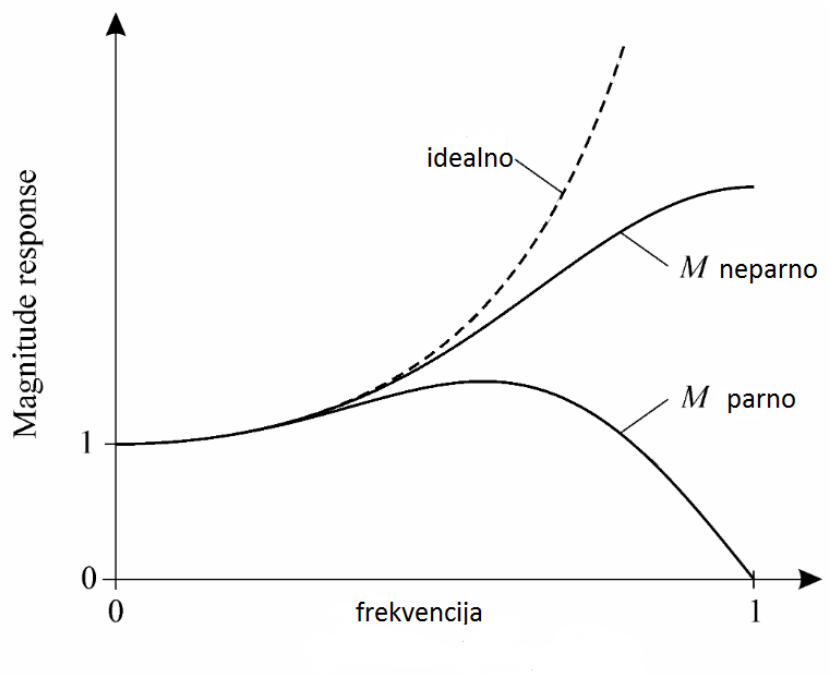
Slika 2.4. Karakteristika CIC filtra

Kompenzatori bez množila s tri koeficijenta su često u upotrebi jer dosta dobro poboljšavaju uski pojas propuštanja. No za širokopojasne aplikacije gdje se iziskuje značajno poboljšanje propada u passbandu koriste se kompenzatori s 5 koeficijenata.

Poznato je da kompenzatori s neparnim brojem koeficijenata postižu jednaku amplitudu ali u širem pojasu nego što je slučaj za kompenzatore s parnim brojem koeficijenata tako da su u ovom radu razmatrani samo kompenzatori s neparnim brojem koeficijenata. Amplitudni odziv kompenzatora s neparnim brojem koeficijenata dan je sljedećom jednadžbom:

$$H(\omega) = C_0 + 2\sum_{k=1}^{(L-1)/2} C_k \cos(k\omega) \tag{2.12}$$

Frekvencijsku karakteristiku kompenzatora s parnim i neparnim brojem koeficijenata, kako i idealnu kompenzacijsku karakteristiku možemo vidjeti na slici 2.5.



Slika 2.5. Usporedba karakteristika CIC kompenzatorskih filtara

2.4 Projektiranje FIR filtara

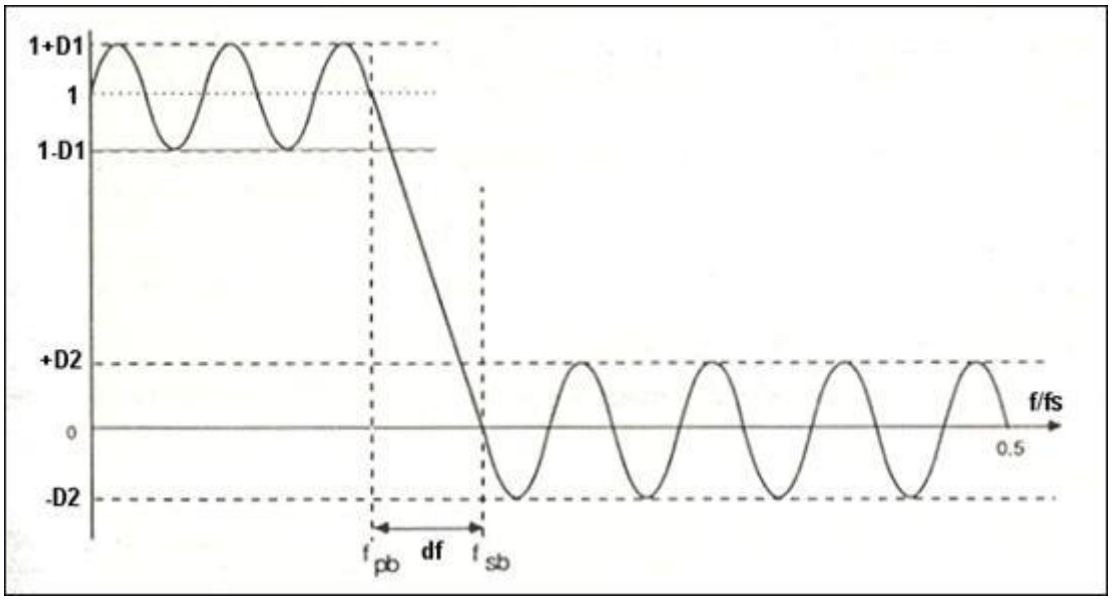
FIR filter se dizajnira tako što se traže koeficijenti i red filtra koji zadovoljava određene specifikacije, što se može raditi u vremenskoj ili u frekvencijskoj domeni.

Tri su dobro razvijena postupka:

- Postupak temeljen na Fourierovoj transformaciji i funkciji vremenskog otvora
- Postupak frekvencijskog uzorkovanja
- Postupak optimalnog projektiranja

Prvi postupak temeljen na Fourierovoj transformaciji i funkciji vremenskog otvora koji možemo zvati i vremenski prozor s obzirom da se tako i zove u engleskoj literaturi *(eng. Window Method)* je direktan i jednostavan. Primijenimo li diskretnu Fourierovu transformaciju na funkciju koja predstavlja frekvencijsku karakteristiku željenog filtra dobiti ćemo impulsni odziv željenog filtra. Kako vrijednosti impulsnog odziva FIR filtra odgovaraju slijedu koeficijenata filtra ujedno smo dobili i traženi oblik filtra. Međutim, nije sve tako idealno. Problem je što je ovako dobiveni filter ne-kauzalan i beskonačnog reda. Ovo drugo

je veći problem, jer je praktički neizvedivo korištenje beskonačnog broja ulaznih signala. Ako filtar radi u realnom vremenu, treba se ograničiti broj ulaznih signala. Tim odsijecanjem dolazi do degradacije karakteristike filtra, odnosno filter više nema ravnu karakteristiku u području propuštanja već valovitu (engl. *Pass-band ripplets*), prijelaz između propusnog i ne-propusnog dijela više nije strm već je postepen, a i u ne-propusnom dijelu se može javiti valovitost (engl. *Stop-band ripplets*). Slika 2 prikazuje izgled amplitudne frekvencijske karakteristike nisko – propusnog filtra s ucrtanom valovitošću propusnog i nepropusnog dijela.



Slika 2.6. Amplitudna frekvencijska karakteristika nisko propusnog filtra

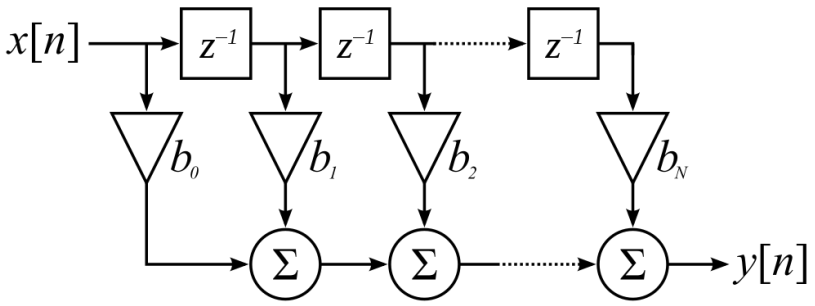
Drugi postupak projektiranja FIR filtra je postupak frekvencijskog uzorkovanja (engl. *Frequency Sampling Technique*). Kod ovog se postupka uzorkuje željena frekvencijska karakteristika filtra između frekvencije 0 i frekvencije uzorkovanja f_M u *N* točaka. Ove frekvencije u Z ravnini odgovaraju simetrično raspoređenim točkama na jediničnoj kružnici. Slijedeći je korak primjena diskretne Fourierove transformacije na svih *N* frekvencija i dobivanje *N* diskretnih uzoraka impulsnog odziva filtra koji ujedno odgovaraju i njegovim koeficijentima.

I na kraju treći postupak koji daje optimalnu strukturu filtra. Poznat je po nazivu Ramirezov algoritam izmjena (*engl. Ramez Exchange Algorithm*), dok ga neki autori zovu Parks-McClellan algoritam jednake valovitosti (*engl. Parks-McClellan equiripple FIR filter design algorithm*). Pod pojmom optimalni filtar mislimo da filtar za dani red filtra i grešku aproksimacije ima najoštriji prijelaz iz propusnog u nepropusni dio. Temelji se na Chebyshevljevoj aproksimaciji, a uz minimizaciju valovitosti u području propuštanja filtra. Primjena Ramirezovog algoritma daje FIR filtar koji ima najmanji broj koeficijenata za danu specifikaciju filtra. Algoritam je univerzalni i pomoću njega se može projektirati filtar koji ima više pojasa propuštanja, a u svakom od njih može se definirati drugačiji iznos minimalnog gušenja.

2.5 Realizacije FIR filtara

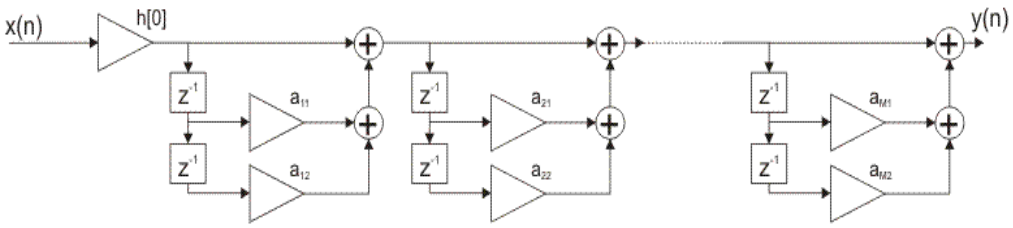
Pri realizaciji FIR filtara postoji nekoliko načina realizacije. Direktna realizacije je najjednostavnija. Broj kašnjenja jednak je redu filtra. Za realizaciju filtra reda *N* potrebno je *N+1* koeficijent i generealno zahtijevaju *N+1* množilo i *N* dvoulaznih zbrajala. Osim nje neke najbitnije realizacije su:

- Transponirana direktna
- Kaskadna
- Polifazne FIR strukture
- Linerane FIR strukture



Slika 2.7. Direktna realizacija FIR filtra

Kaskadna realizacija koristi se za prijenosne funkcije višeg reda, koje mogu biti realizirane kaskadom FIR sekcija drugog reda ili moguće i sekcijama prvog reda. Primjer kaskadne realizacije možemo vidjeti na slici 2.8.



Slika 2.8. Kaskadna realizacija FIR filtra

Polifazna dekompozicija prijenosne funkcije H(z) vodi ka paralelnoj strukturi. Za ilustraciju, zamislimo kauzalnu prijenosnu funkciju H(z), za filtar reda 8:

$$H(z) = (h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} + h(4)z^{-4} + h(5)z^{-5} + h(6)z^{-6} + h(7)z^{-7} + h(8)z^{-8} \quad (2.13)$$

H(z) možemo izraziti kao sumu 2 terma, od kojih jedan sadrži samo parne koeficijente, dok drugi sadrži neparne.

$$\begin{aligned} H(z) &= (h(0) + h(2)z^{-2} + h(4)z^{-4} + h(6)z^{-6} + h(8)z^{-8}) \\ &+(h(1)z^{-1} + h(3)z^{-3} + h(5)z^{-5} + h(7)z^{-7}) \\ &= (h(0) + h(2)z^{-2} + h(4)z^{-4} + h(6)z^{-6} + h(8)z^{-8}) + \\ &(z^{-1}(h(1) + h(3)z^{-2} + h(5)z^{-4} + h(7)z^{-6}) \end{aligned} \quad (2.14)$$

koristeći sljedeću notaciju:

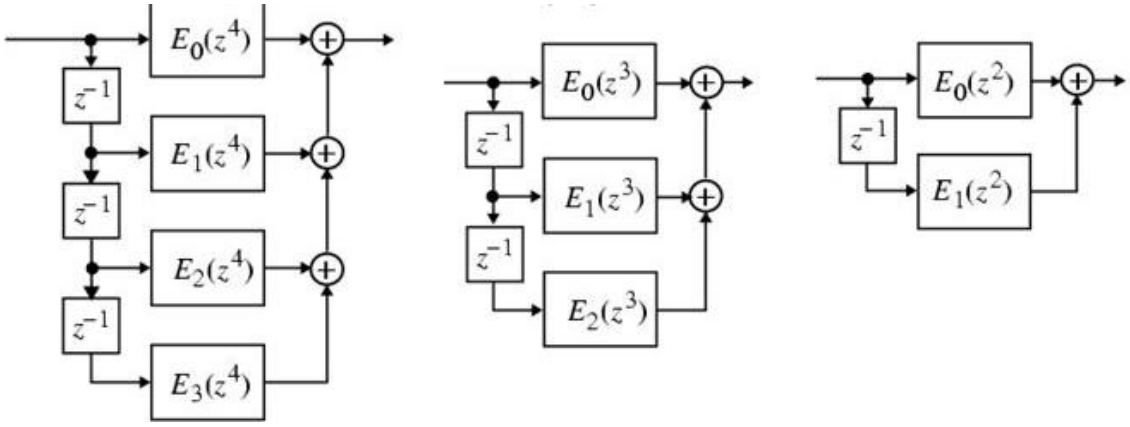
$$E_0(z) = h(0) + h(2)z^{-1} + h(4)z^{-2} + h(6)z^{-3} + h(8)z^{-4} \quad (2.15)$$

$$E_1(z) = h(1) + h(3)z^{-1} + h(5)z^{-2} + h(7)z^{-3}$$

možemo izraziti H(z) kao tzv. dvogranu polifaznu kompoziciju

$$H(z) = E_0(z^2) + z^{-1}E_1(z^2) \tag{2.16}$$

Realizacija H(z) bazirana na četverogranoj, trogranoj i dvogranoj polifaznoj dekompoziciji prikazana je na slici 2.9.



Slika 2.9. Polifazna dekompozicija

U ovom diplomskom radu radit ćemo s linearnim FIR filtrom pa je red spomenuti linearne FIR strukture. Simetrija ili antisimetrija je svojstvo FIR filtara koje se može iskoristiti kako bi smanjili broj množila skoro na pola u odnosu na direktnu ralizaciju. Simetrija impulsnog odziva je nužan preduvjet za linearnu fazu. Vrijedi sljedeći teorem, ako impulsni odziv nerekurzivnog filtra zadovoljava relaciju:

$$h(n) = h(M-1-n), \quad \begin{cases} za\ n = 0, 1, \ldots \dfrac{M}{2} - 1 \ \ ako\ je\ M\ parno, i \\ za\ n = 0, 1, \ldots \dfrac{M-1}{2} \ ako\ je\ M\ neparno \end{cases} \tag{2.17}$$

tada nerekurzivni digitalni filter ima linearnu karakteristiku. Sustav s linearnom fazom ima simetričan ili antisimetričan impulsni odziv, te se ovisno o tipu simetrije definiraju četiri tipa FIR filtra s realnim impulsnim odzivom duljine *M*:

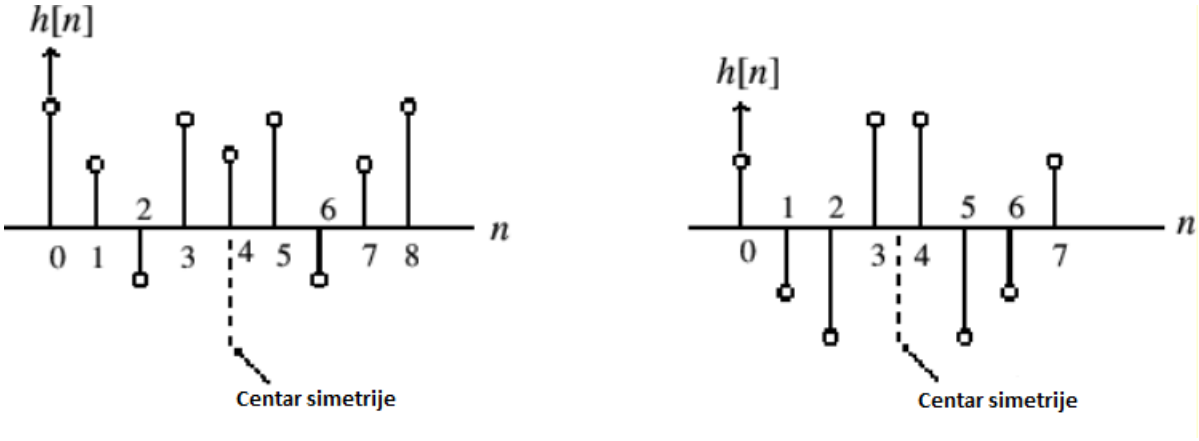
Nerekurzivni digitalni filter tipa I

- Simetričan impulsni odziv

- neparan broj uzoraka impulsnog odziva

Nerekurzivni digitalni filter tipa II

- Simetričan impulsni odziv
- paran broj uzoraka impulsnog odziva



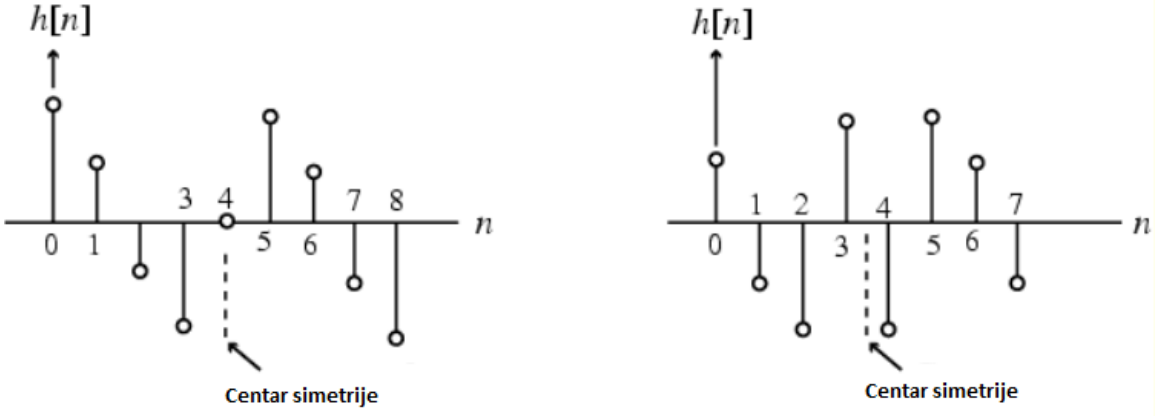
Slika 2.10. Impulsni odziv tip I i II

Nerekurzivni digitalni filter tipa III

- antisimetričan impulsni odziv
- neparan broj uzoraka impulsnog odziva

Nerekurzivni digitalni filter tipa IV

- antisimetričan impulsni odziv
- paran broj uzoraka impulsnog odziva



Slika 2.11. Impulsni odziv tip III i IV

Tablica 2.1: Tipovi FIR filtara s realnim impulsnim odzivima

Tip	h	M	A($e^{j\omega}$)	ϕ($e^{j\omega}$)
I	Simetričan	Paran	$\sum_{k=0}^{\frac{M}{2}} a_k \cos(\omega k)$	$-\frac{\omega M}{2}$
II	Simetričan	Neparan	$\sum_{k=1}^{\frac{M+1}{2}} b_k \cos(\omega(k-\frac{1}{2}))$	$-\frac{\omega M}{2}$
III	Simetričan	Paran	$\sum_{k=1}^{\frac{M}{2}} c_k \sin(\omega k)$	$\frac{\pi}{2}-\frac{\omega M}{2}$
IV	simetričan	Neparan	$\sum_{k=1}^{\frac{M+1}{2}} d_k \sin(\omega(k-\frac{1}{2}))$	$\frac{\pi}{2}-\frac{\omega M}{2}$

3 Dizajn jednostavnih FIR filtara bez množila

3.1 Niskopropusni filtri

Za izračun optimalnih koeficijenata brine se funkcija firmm.m koja prima uglavnom klasične parametre kad je u pitanju dizajniranje filtara. Izgled cijele funkcije vidljiv je u privitku A. Poziva se sljedećom matlab naredbom h=firmm(L ,B, wp, ws, K), gdje:

- L je broj koeficijenata
- B je širina riječi koeficijenata
- Wp je područje propuštanja

- Ws je područje gušenja
- K je težinski faktor

Osnovna ideja kod dizajniranja je određivanje željene amplitudno-frekvencijske karakteristike $A(e^{j\omega})$ koja aproksimira željenu frekvencijsku karakteristiku $D(e^{j\omega})$ prema nekom kriteriju težinske funkcije $W(\omega)$. Najprije je potrebno definirati izraz za optimizacijsku pogrešku, koji je preuzet iz [3] kako slijedi:

$$E(\omega) = W(\omega)[D(e^{j\omega}) - A(e^{j\omega})] \tag{3.1}$$

Za dizajn se koristi minimax kriterij koji kako mu i samo ime kaže minimizira najveću optimizacijsku pogrešku, ili zapisano izrazom:

$$e = \min(max|\mathrm{E}(\omega)|) \tag{3.2}$$

Karakteristika željenog niskopropusnog filtra definirana je izrazom:

$$D\left(e^{j\omega}\right) = \begin{cases} 1, 0 \le \omega \le \omega_p \\ 1, \ \omega_s \le \omega \le \pi \end{cases} \tag{3.3}$$

Metoda kreće od zadanih frekvencija ω_p i ω_s, broja koeficijenata, te omjera dozvoljenih odstupanja aproksimacije od idealne karakteristike u području propuštanja $\pm\delta_1$ i području gušenja $\pm\delta_2$, odnosno $K = \delta_1/\delta_2$. Zadaje se samo omjer dok iznosi odstupanja ostaju slobodne varijable. Pošto se iznosi odstupanja mogu razlikovati uvedena je težinska funkcija $W(\omega)$ koja omogućava različite težinske faktore pogreške za različita frekvencijska područja, a definirana je sljedećim izrazom:

$$W(\omega) = \begin{cases} \frac{1}{K}, 0 \le \omega \le \omega_p \\ 1, \ \omega_s \le \omega \le \pi \end{cases} \tag{3.4}$$

Matlab skripta je napisana da radi s neparnim brojem koeficijenata kako bi se postiglo što veće gušenje. Područje iz kojeg tražimo koeficijente su cjelobrojne potencije broja 2 što je

vrlo korisno u hardverskoj realizaciji filtra budući da se ne treba koristiti množenje, nego se sva množenja svode na posmake ulijevo ili udesno pa tako i definiramo vektor mogućih koeficijenata *coeffs.*

$$coeffs = [-2.\hat{}(0:B-1), 0, 2.\hat{}(0:B-1)]; \quad (3.5)$$

Poželjno je da broj koeficijenata *L* bude mali jer se za traženje optimalnog rješenja koristi metoda iscrpnog pretraživanja. Potrebno je ispitati svaku moguću kombinaciju iz skupa mogućih koeficijenata, što za veliki *L* znači i veliko vrijeme izvršavanje skripte. Zato su ovi niskopropusni filtri pogodni za manje zahtjevne primjene u obradi signala. Budući da tražimo filtre s linearnom fazom, kojoj su koeficijenti simetrični u odnosu na središnji možemo pretraživanje smanjiti tako da vektor koeficijenata *h* izrazimo kao samo jednu stranu, npr. za vektor koeficijenata *h*=2,4,8,4,2 uzimamo samo *h'*=8,4,2 jer znamo da se svi osim središnjeg preslikavaju i na drugu stranu.

Za punjenje matrice koeficijenata koristi se matlab funkcija permn, koja traži sve moguće permutacije vektora mogućih koeficijenata *coeffs*. Izgled matrice koeficijenata za *L=9 i B=8* možemo vidjeti u matrici *C*.

$$C = \begin{bmatrix} -128 & -128 & -128 & -128 & -128 \\ -128 & -128 & -128 & -128 & -64 \\ -128 & -128 & -128 & -128 & -32 \\ \dots & \dots & \dots & \dots & \dots \\ 128 & 128 & 128 & \dots & 128 \end{bmatrix}$$

Matrica 3.1. Matrica koeficijenata

Sljedeće množimo matricu koeficijenata s matricom *HH*, čiji izgled vidimo odmah ispod. Dobit ćemo matricu frekvencijskih odziva *H* za sve moguće kombinacije koeficijenata, gdje svaki redak predstavlja filtar čiji su koeficijenti određeni retkom iz matrice *C*.

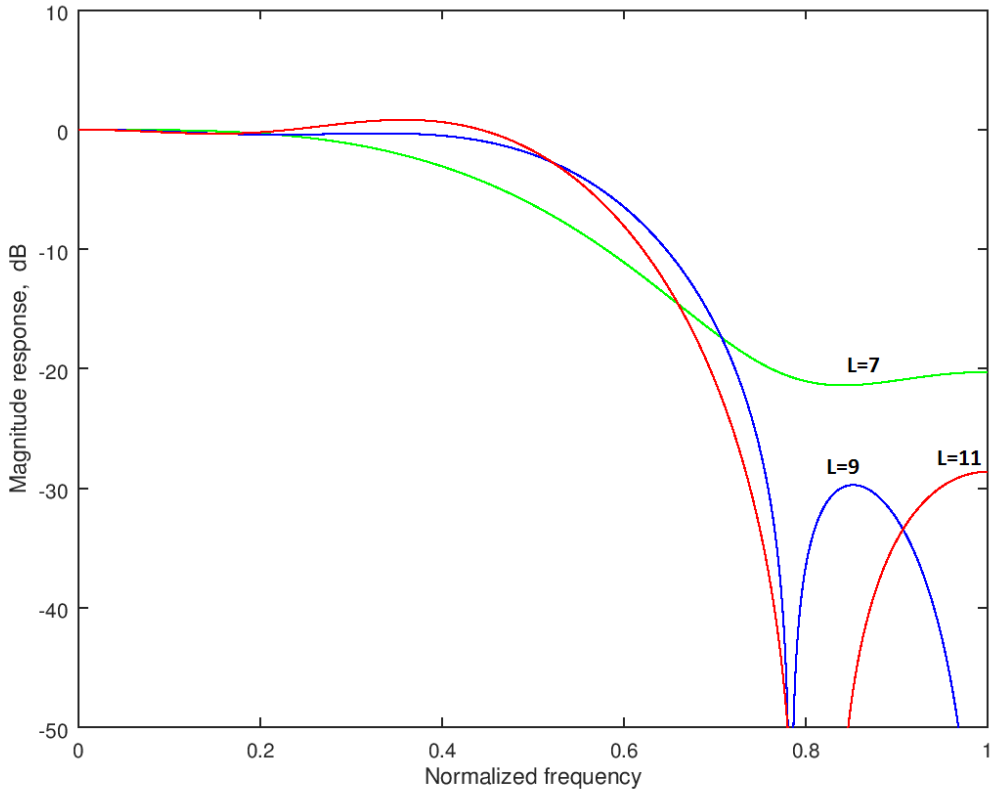
$$HH = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 2\cos(\omega_1) & 2\cos(\omega_2) & 2\cos(\omega_3) & \dots & 2\cos(\omega_n) \\ 2\cos(2\omega_1) & 2\cos(2\omega_2) & 2\cos(2\omega_3) & \dots & 2\cos(2\omega_n) \\ 2\cos(3\omega_1) & 2\cos(3\omega_2) & 2\cos(3\omega_3) & \dots & 2\cos(3\omega_n) \\ 2\cos(4\omega_1) & 2\cos(4\omega_2) & 2\cos(4\omega_3) & \dots & 2\cos(4\omega_n) \end{bmatrix}$$

Matrica 3.2. Matrica HH

$$H = \begin{bmatrix} \mathrm{H}(\omega_1) & \mathrm{H}(\omega_2) & \mathrm{H}(\omega_3) & \dots & \mathrm{H}(\omega_n) \\ \mathrm{H}(\omega_1) & \mathrm{H}(\omega_2) & \mathrm{H}(\omega_3) & \dots & \mathrm{H}(\omega_n) \\ \mathrm{H}(\omega_1) & \mathrm{H}(\omega_2) & \mathrm{H}(\omega_3) & \dots & \mathrm{H}(\omega_n) \\ \dots & \dots & \dots & \dots & \dots \\ \mathrm{H}(\omega_1) & \mathrm{H}(\omega_2) & \mathrm{H}(\omega_3) & \dots & \mathrm{H}(\omega_n) \end{bmatrix}$$

Matrica 3.3. Matrica frekvencijskih odziva za sve moguće kobinacije koeficijenata

Po izrazu 3.1 tražimo maksimalne pogreške u pojedinom redu matrice 3.3, i kao optimalnu kombinaciju koeficijenata uzimamo onu koja nam je daje minimalnu vrijednost od svih pogrešaka po izrazu iz formule 3.2. Također dodatno se može smanjiti prostor pretraživanja ako gledamo samo filtre gdje je središnji koeficijent pozitivna potencija broja dva, jer je to identičan filter onomu kojem su koeficijenti pomnoženi s minus jedan. Izgled dobivenih filtra za širinu riječi koeficijenata *B*=8, težinski faktor *K*=1 te frekvenciju propuštanja, odnosno gušenja od $\omega_p = \frac{\pi}{4}$ i $\omega_s = \frac{3\pi}{4}$ možemo vidjeti na slici 3.1. Crvenom bojom prikazana je karakteristika filtra s jedanaest koeficijenata, plavom bojom filtar s devet koeficijenata dok je zeleni filtar sa sedam koeficijenata.



Slika 3.1. Karakteristika dizajniranih niskopropusnih FIR filtra

3.2 Kompenzacijski filtri za CIC decimatore

Kod kompenzacijskih filtra za izračun optimalnih koeficijenata brine se funkcija fircompmm.m koja prima slične parametre kao i funkcija za dizajn niskopropusnih fir filtara te je također u potpunosti vidljiva u privitku B. Poziva se sljedećom matlab naredbom h=fircompmm(N, R, L, B, wp,), gdje:

- N je red filtra
- R je faktor decimacije
- L je broj koeficijenata
- B je širina riječi koeficijenata
- Wp je područje propuštanja

Ovdje krećemo od funkcije pogreške koja je definirana kao razlika između maksimalne i minimalne amplitude u području propuštanja po uzoru na [1], ili zapisano izrazom:

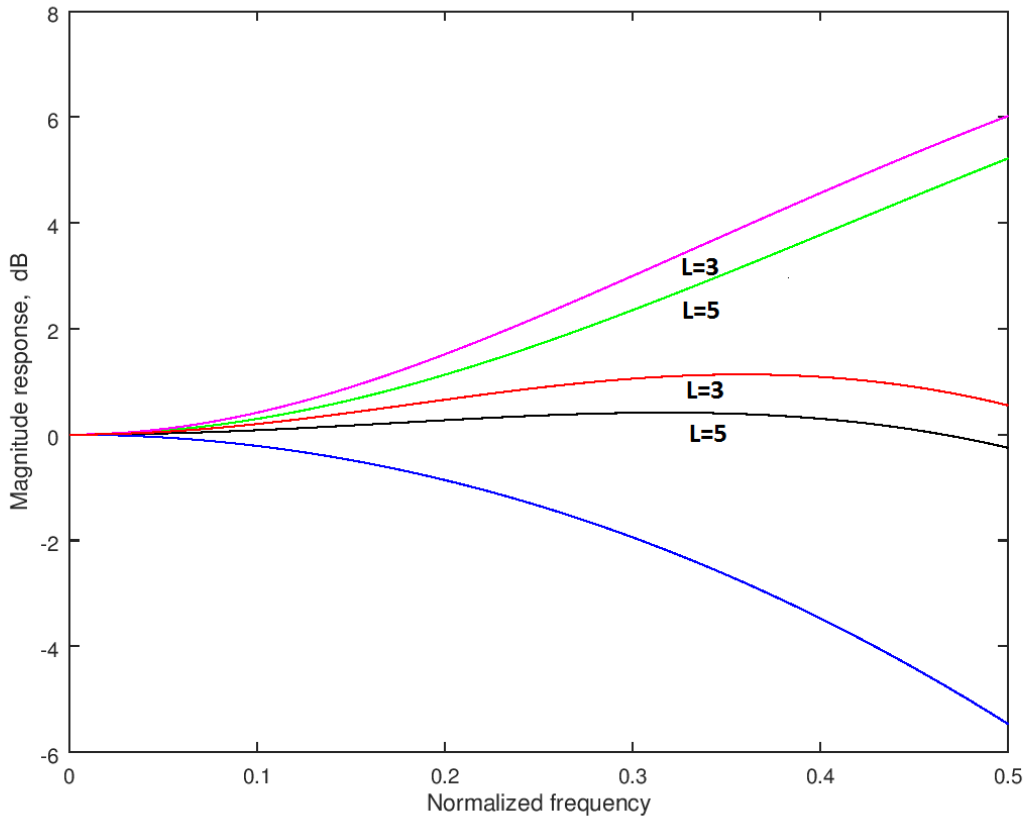
$$e(c) = \max H_c(\omega)H(\omega, c) - \min H_c(\omega)H(\omega, c) \tag{3.6}$$

Budući da se kompenzator spaja na izlaz iz CIC filtra te utječe na amplitudni odziv na niskoj frekvenciji, $H_c(\omega)$ je odziv CIC filtra na niskoj frekvenciji:

$$H_c(\omega) = \left[\frac{1}{R}\frac{sin\left(\frac{\omega}{2}\right)}{sin\left(\frac{\omega}{2R}\right)}\right]^N \tag{3.7}$$

Najprije uvrštavanjem proizvoljnih *N* i *R* dobijemo odziv CIC filtra na niskoj frekveciji. Ovdje također područje iz kojeg tražimo koeficijente čine cjelobrojne potencije broja 2. Postupak je dosta sličan dizajnu niskopropusnog filtra iz prošlog primjera. Najprije je potrebno izračunati odzive kompenzatora za sve moguće kombinacije koeficijenata, što dobijemo množenjem matrice koeficijenata sa matricom *HH* koje su napravljene kao u prošlom primjeru. Kad imamo odzive svih mogućih kompenzatora uvrstimo ih u formulu 3.6 kako bismo dobili funkciju pogreške i od njih izabiremo opet onu kombinaciju koeficijenata koja nam daje najmanju pogrešku. Rađena je kompenzacija CIC filtra reda *N*=6, faktora

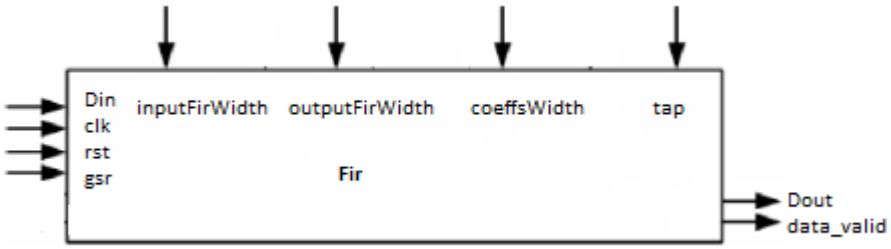
decimacije *R*=32, te pojasa propuštanja od $\omega_p = \frac{\pi}{2}$. Na slici 3.2 možemo vidjeti usporedbu karakteristika kompenzacijskih filtara za različite duljine koeficijenata *L*=3 koji je na slici predstavljen ljubičastom bojom te *L*=5 koji je prikazan zelenom bojom za područje propuštanja od $\omega_p = \frac{\pi}{2}$. Također na istoj slici su prikazane kompenzirane karakteristike sustava nakon spajanja kompenzacijskog filtra s tri koeficijenta crvenom bojom te nakon spajanja kompenzacijskog filtra s pet koeficijenata crnom bojom. Plavom bojom predstavljena je karakteristika CIC decimacijskog filtra.



Slika 3.2. Karakeristika kompenzacijskih filtra s tri i pet koeficijenta

4 Generički RTL model FIR filtra bez množila

Model filtra koji je implementiran u programskom jeziku VHDL (engl. Very High Speed Integrated Circuit Hardware Description Language) prikazan je na slici 4.1. U tablicama 4.1 i 4.2 dani su opisi potrebnih priključaka i generičkih konstanti implementiranog filtra.



Slika 4.1. Generički model FIR filtra

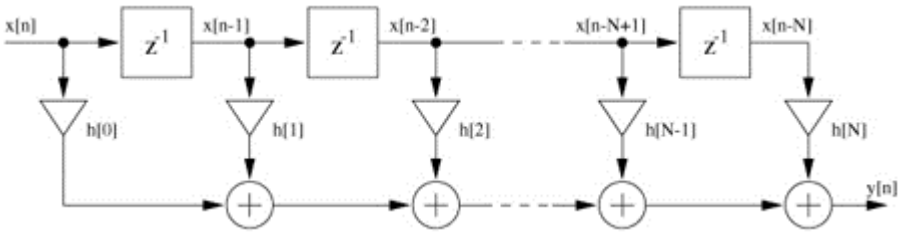
Tablica 4.1. Priključci implementiranog FIR filtra

Signal	**Opis**
clk	signal takta
rst	sinkroni reset
Din	ulazni podatak
gsr	služi za resetiranje svih komponenti FPGA dijela
data_valid	služi za dojavljivanje kad je izlazni podatak filtra spreman
Dout	izlazni podatak

Tablica 4.2. Generičke konstante implementiranog FIR filtra

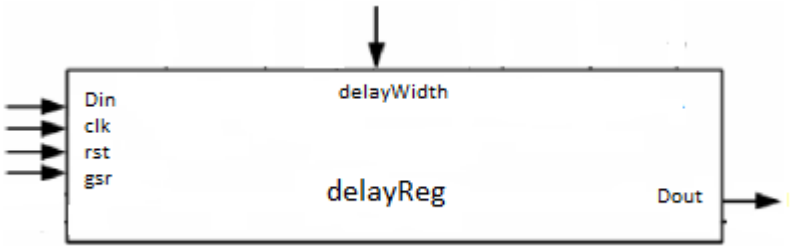
Generička konstanta	**Opis**
inputFirWidth	širina ulaznog signala
outputFirWidth	širina izlaznog signala
coeffsWidth	širina koeficijenata
tap	broj koeficijenata

Filtar zapravo obavlja konvoluciju ulaznog signala s koeficijentima filtra što se radi njihovim množenjem. Budući da su nam svi koeficijenti cjelobrojne potencije broja dva možemo napraviti filtar bez množila koristeći posmake ulijevo. Posmak broja za *N* mjesta ulijevo je zapravo množenje broja s brojem 2^N. Korištena je direktna realizacija FIR filtra. Kao što se vidi iz slike 4.2, za direktnu realizaciju potrebno je imati *N-1* blokova za kašnjenje *N* množila te *N-1* zbrajala.



Slika 4.2. Implementirana realizacija FIR filtra

Za kašnjenja se brine VHDL blok pod nazivom delayReg.vhd. Izgled mu možemo vidjeti na slici 4.3 , a zadaća mu je samo da signal na ulazu zakasni za jedan takt.



Slika 4.3. delayReg blok

Glavni je model FIR filtra spremljen pod nazivom fir.vhd. U njemu je opisana sva logika filtra. Sve operacije koje se rade unutar filtra rade se nad signalima duljine inputFirWidth + AccExt. AccExt je akumulatorska ekstenzija koja iznosi:

$$AccExt = ceil(log_2 \sum abs(h)) \tag{4.1}$$

Najprije je potrebno predznačno proširiti ulazni signal koji je unutar 16 bitova na potrebnu duljinu. Prvi ulazni signal možemo množiti s prvim koeficijentom direktno, dok ostale iznose množenja dobivamo kroz petlju koja ide od 1 do *N-2*. Ovdje pričamo o množenju, no to množenje je u VHDL-u FIR filtra implementirano posmacima ulijevo. U paketu package01.vhd definirana je funkcija how_many_places čiji izgled možemo vidjeti na slici 4.4. Funkcija za određeni koeficijent vraća broj pozicija za koliko moramo posmaknuti ulazni signal. Funkcije za posmak ulijevo i udesno su također definirane u paketu package01.vhd.

```
function how_many_places(d_in:std_logic_vector)
    return natural is
variable Shift_places:natural;
begin

case conv_integer(d_in) is
        when 2 =>   Shift_places := 1;
        when 16 =>   Shift_places := 4;
        when 32 =>   Shift_places := 5;
        when 128 =>   Shift_places := 7;
        when -2 =>   Shift_places := 1;
        when -4 =>   Shift_places := 2;
        when -32 =>   Shift_places := 5;
        when others => Shift_places := 0;
    end case;

return Shift_places;
end function how_many_places;
```

Slika 4.4. Funkcija how_many_places

Osim funkcija, u paketu su definirane generičke konstante nisko-propusnog FIR filtra i kompenzacijskog filtra te koeficijenti za iste. Ovisno o tome koji se implementira na sklopovlju potrebno je promijeniti fir.vhd datoteku. Početno je fir.vhd datoteka podešena da radi s niskopropusnim FIR filtrom čije karakteristike možemo vidjeti na slici 4.5.

```
-- ============================================================================
--                          Fir Filter characteristic
-- ============================================================================

constant inputFirWidth  :integer:=16;
constant outputFirWidth :integer:=17;
constant acc_extension  :integer:=7;
constant coeffsWidth    :integer:=8;
constant tap            :integer:=9;
constant N              :integer:=23;

type coeffs_type is array (0 to tap-1) of std_logic_vector(coeffsWidth-1 downto 0);
constant coeffs_cic:coeffs_type:=
           (
               X"02",
               X"FE",
               X"FC",
               X"10",
               X"20",
               X"10",
               X"FC",
               X"FE",
               X"02"
               );
```

Slika 4.5. Karakteristike niskopropusnog FIR filtra

Budući da moramo instancirati više jednakih komponenti, naredba generate ovdje je dosta korisna jer smanjuje pisanje koda. Koristi se za specificiranje grupe identičnih komponenata koristeći specifikacije samo jedne komponente i ponavljanje iste onoliko puta koliko nam je potrebno. Koristeći naredbu generate instanciramo *N-1* blok za kašnjenje, *N-1* množila i zbrajala. Na slici 4.6 možemo vidjeti isječak koda gdje je primjer korištenja naredbe generate za instanciranje zbrajala, množila i sklopova za kašnjenje.

```
delay_reg: for i in 1 to tap-1  generate
begin
   delay_i: component delayReg
   generic map
   (
   delayWidth => N
   )
     port map
   (
   clk => ce_fs_low,
   gsr => gsr,
   rst => rst,
   D => DELAY(i-1),
   Q => DELAY(i)
   );
end generate;

mull_add: for i in 1 to tap-1  generate
DELAY_neg(i)<=not(DELAY(i))+1;
MULL(i)<=    my_shift_left(DELAY(i),how_many_places(coeffs(i))) when coeffs(i)>0 else
             my_shift_left(DELAY_neg(i),how_many_places(coeffs(i)));

ADD(i)<=to_stdlogicvector(to_bitvector(ADD(i-1)))+to_stdlogicvector(to_bitvector(MULL(i)));

end generate;
```

Slika 4.6. Instanciranje komponenti naredbom generate

Kao što se vidi iz isječka koda u petlji se vrte ulazni signali i množe se s pripadajućim koeficijentima. Ako je koeficijent negativan broj, ulazni signal mijenja predznak te se isti množi s pripadajućim koeficijentom. Izlazni podatak bit će spremljen u izlazu posljednjeg zbrajala što je također jasno vidljivo iz slike 4.6. Nakon što se napravi konvolucija i imamo spreman izlazni uzorak postavljamo signal data_valid u '1' čime dojavljujemo FIFO memoriji da imamo spreman podatak za nju. O FIFO memorijama i komunikaciji CPU dijela s FPGA dijelom bit će govora u sljedećem poglavlju.

5 Implementacija na platformu ZedBoard

5.1 Opis sklopovlja

ZedBoard je dio Zynq-7000 porodice. Zynq-7000 porodica je bazirana na Xilinxovoj All Programmable SoC (engl. *System on Chip*) arhitekturi. Ti dizajni integriraju dvojezgreni ARM Cortex-A9 procesorski sustav (engl. Processing System) zajedno sa 28 nm programabilnom logikom (engl. *Programmable Logic*) na jednom uređaju. Zynq-7000 porodica nudi fleksibilnost i skalabilnost FPGA dok također pruža performancu, snagu i jednostavnost upotrebe koju tipično vežemo za ASIC (engl. *Application-Specific Integrated Circuit).* Velika pogodnost Zynq-7000 porodice je ta što omogućuje dizajnerima da rade aplikacije velikih performansi, koje su ujedno i osjetljive na cijenu, sve s jedne platforme. Zynq-7000 arhitektura ima podjelu na dva dijela, a to su procesorski te programabilni dio. Slika 5.1 nam prikazuje izgled Zynq-7000 arhitekture. Kao što se vidi i iz slike, procesorski dio sastoji se od 4 glavna bloka:

- APU
- Memory Interfaces
- I/O peripherals
- Interconect

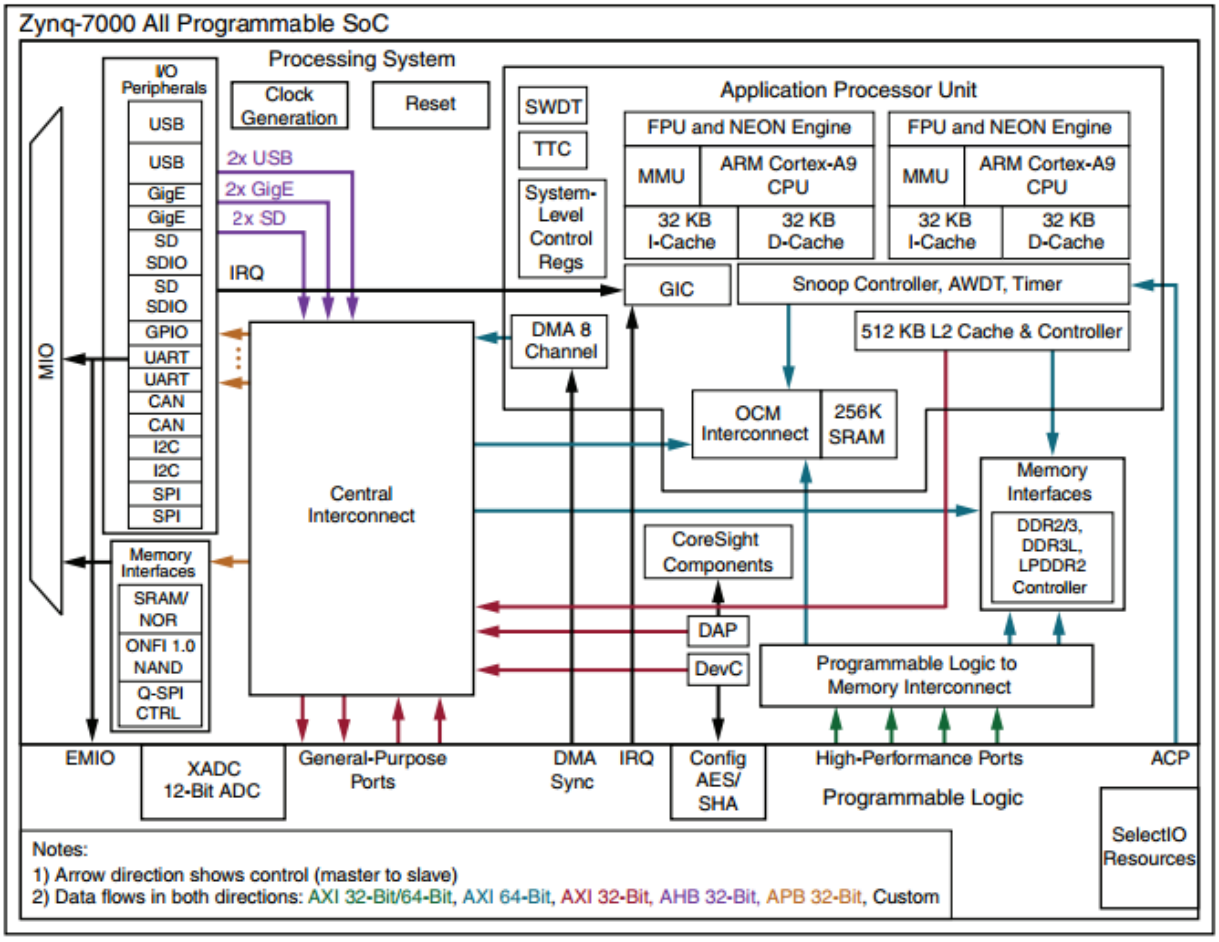
Glavni dijelovi APU-a su dvojezgreni ARM Cortex-A9 procesor, 512KB L2 cache, 256KB on-chip RAM memorije koji su dostupni kroz CPU ili programabilnu logiku, 8-kanalni DMA (engl. *Direct Memory Access*) koji podržava razne tipove prijenosa podataka (memory-to-memory, memory-to-peripheral, peripheral-to-memory). Sami procesor ima snagu od 2.5 DMIPS (engl. *Dhrystone Million Instructions Per Second*). Ovisno o modelu postoji više radnih frekvencija, no one se kreću od 667 MHz pa do 1GHz. Također procesor ima L1 cache memoriju (odvojeno za instrukcije i podatke, po 32KB svaka).

Memorijsko sučelje sastoji se od dinamičkog i statičkog memorijskog kontrolera. Dinamički memorijski kontroler podržava DDR3, DDR3L, DDR2 i LPDDR2 memorije, dok statički memorijski kontroler podržava NAND flash sučelje, Quad-SPI flash sučelje te paralelno NOR flash sučelje.

I/O periferija je zapravo periferija za prijenos podataka. Podržava sljedeće protokole za razmjenu podataka:

- Dvije 10/100/1000 tromodne Ethernet periferije
- Dvije USB 2.0 periferije
- Dva CAN 2.0 kontrolera
- Dva SPI porta
- Dva UART-a
- Dva I^2C sučelja
- Dva SD/SDIO kontrolera

APU, memory interface unit te I/OP su svi međusobno te sa PL povezani višeslojnim ARM AMBA-AXI (engl. *Advanced Micro-controller Bus Architecture-Advanced Extensible Interface*) sabirničkim sustavom.



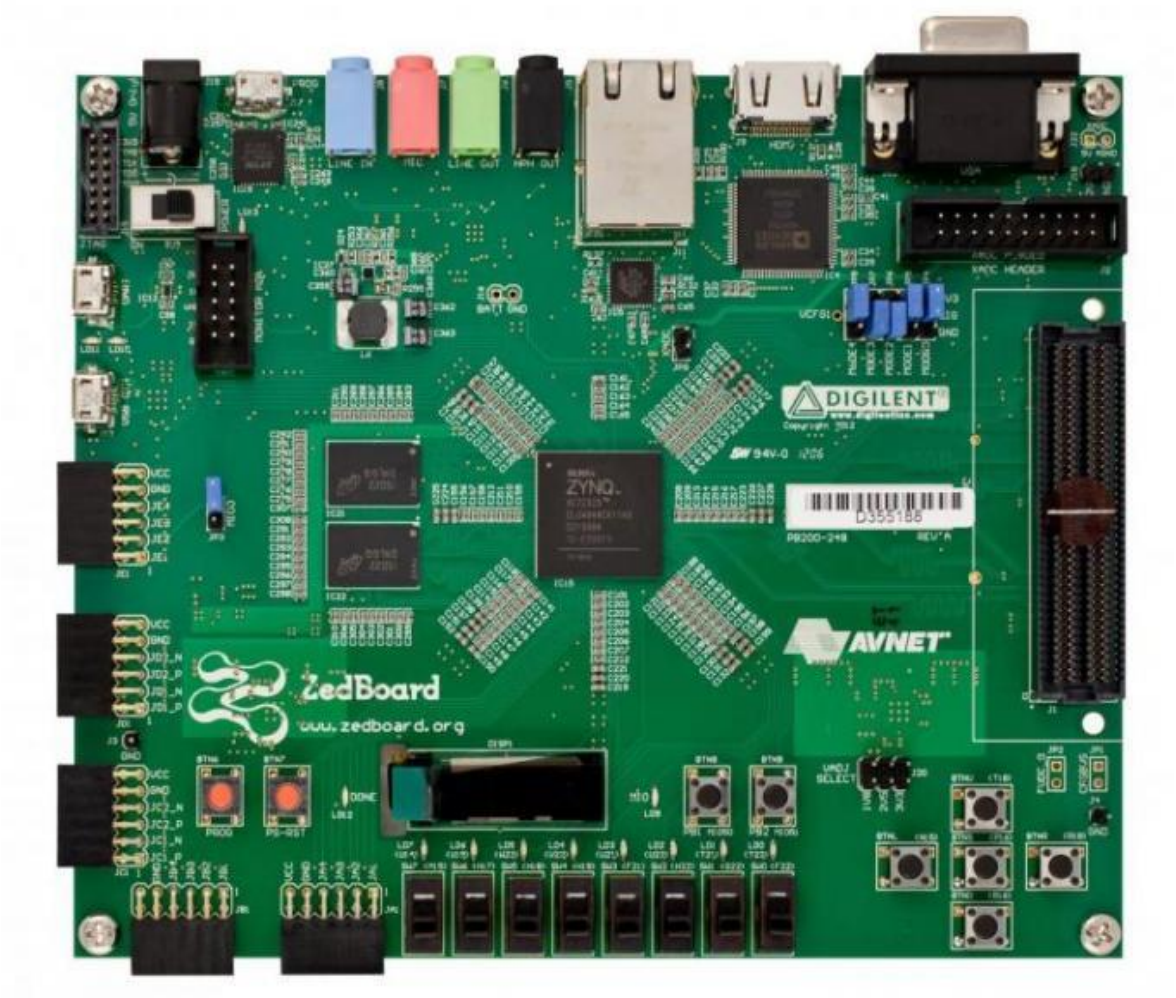
Slika 5.1. Zynq-7000 arhitektura

Glavni dijelovi programabilne logike su:

- CLB-ovi (engl. *Configural Logic Block*)
- 36Kb blok RAM
- DSP odresci
- Programabilni I/O blokovi
- Dva 12-bitna A/D (engl. *Analog-to-digital*) konvertera
- PL konfiguracijski modul

ZedBoard je jedan od najmanjih uređaja iz Zynq-7000 porodice i baziran je na Artix-7 logičkom dijelu s kapacitetom od 13 300 logičkih odrezaka, 220 DSP48E1S i 140 BlockRAM-a. Izgled Zedboard razvojne pločice vidljiv je na slici 5.2. Na ZedBoardu je puno perifernih sučelja:

- GPIO (engl. General Purpose Input/Output) – 9 ledica, 8 sklopki te 7 gumba
- OLED (engl. Organic Light Emitting Diode) display
- Ethernet
- USB-OTG (periferija)
- USB-JTAG (programiranje)
- USB-UART (komunikacija)
- SD card utor (na donjoj strani pločice)
- FMC sučelje
- Video HDMI i VGA



Slika 5.2. Zedboard

Dodatno postoji sučelje prema 256Mbit flash memoriji te 512MB DDR3 memoriji. Također postoje dva oscilatora za signal takta, jedan na frekvenciji od 100 MHz, a drugi na 33.333 MHz. ZedBoard se može programirati na 4 različita načina:

- USB-JTAG – to je najjednostavniji i najuobičajeniji način programiranja ZedBoarda putem USB-micro-USB kabela koji se dobije uz ZedBoard
- Tradicionalni JTAG – ukoliko korisnik želi, može umjesto USB-JTAG koristiti Xilinx JTAG konektor koji je dostupan na pločici, no kabel nije uključen u ZedBoard paketu
- Quad-SPI flash memory – flash memorija na pločici nije volatile tako da može biti iskorištena da spremi podatke o konfiguraciji te tako omogući bežično programiranje

- SD kartica – postoji SD utor s donje strane pločice gdje se umetne SD kartica i programira se ZedBoard podacima spremljenim na kartici. Ovo je takoĎer bežični način programiranja ZedBoarda

Korisnik sam odabire metodu programiranja pomoću seta kratkospojnika (engl. jumper) koji su locirani malo iznad Digilent loga. Od pet kratkospojnika, srednja tri koriste se za definiranje načina programiranja (JTAG, flash memorija ili SD kartica). Lijevi kratkospojnik govori da li se koristi unutarnji PLL, dok skroz desni kontrolira JTAG mod.

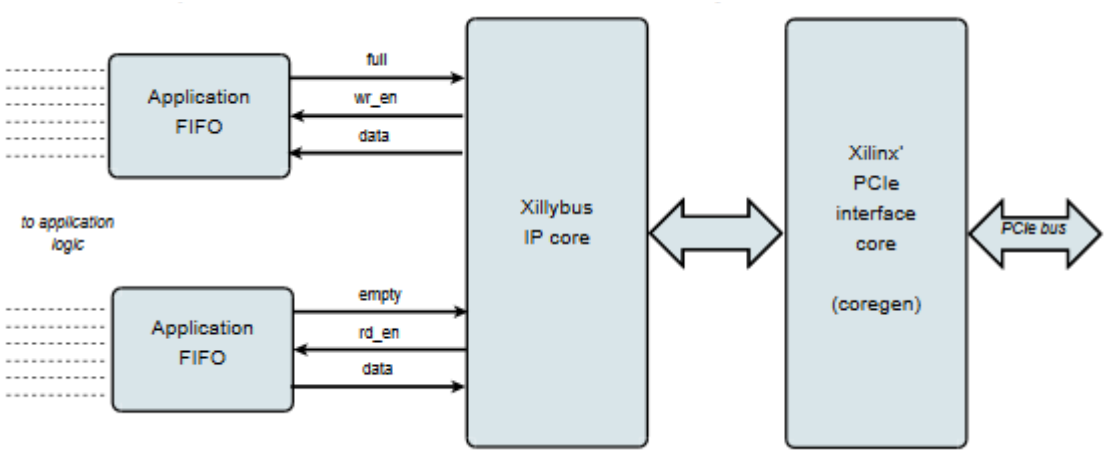
5.2 Razvojno okruženje za ZedBoard

Xilinx za sklopove porodice Zynq-7000 nudi dva razvojna alata, a to su ISE Design Suite s paketom XPS (engl. *Xilinx Platform Studio*) te Vivado Design Suite. Oba alata imaju u sebi uključen SDK (engl. *Software Development Kit*) alat za razvoj programske podrške. Vivado je noviji alat, čija upotreba počinje od Xilinx sklopova serije 7 dok je ISE namijenjen za starije Xilinxove serije sklopovlja. Za potrebe ovog rada koristio se Vivado Design Suite. Vivado Design Suite u sebi sadrži niz alata među kojima su:

- IP Integrator koji je svojevrsna zamjena za dosad korišteni EDK (engl. *Embedded Development Kit*) za novije FPGA uređaje iz Xilinxove porodice 7
- SDK (engl. *Software Development Kit*) za razvoj programske podrške
- HLS (engl. *High-Level Synthesis*) podrška za projektiranje s visoke razine specifikacija, odnosno automatsko prevođenje specifikacije iz c programskog jezika na RTL razinu
- DSP Design (engl. *Digital Signal Processing*) podrška za implementaciju DSP specifičnih funkcionalnosti
- IP Packaging podrška za integraciju novorazvijenih korisničkih jezgri

5.3 FIFO memorija i Xillybus sučelje za ZedBoard

Xillybus je rješenje za prijenos podataka između FPGA dijela i domačina u Linux ili Windows okruženju. Kao što se vidi na slici 5.5, dizajniran je da radi s FIFO ili sinkronom memorijom kao posrednikom.



Slika 5.4. Tok podataka realiziran Xillybusom

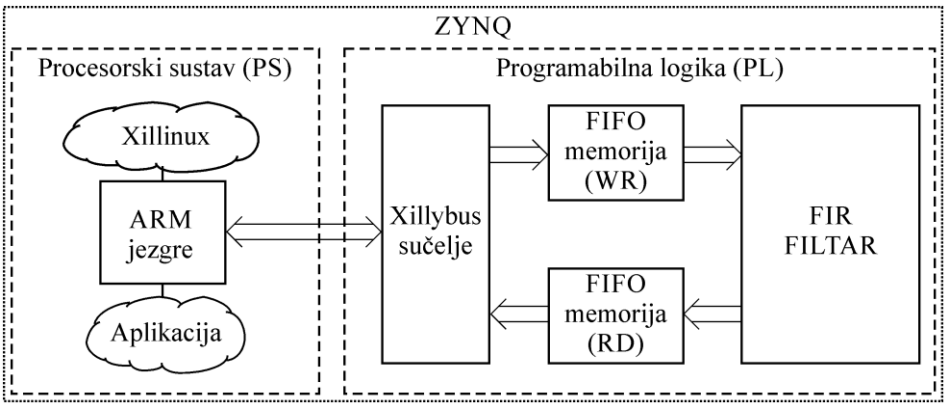
Upisivanjem podatka u donji FIFO sa slike, IP jezgra osjeti da je podatak spreman za čitanje. Ubrzo nakon toga ga pročita te ga šalje domaćinu na daljnju obradu. Domaćin nakon obrade vraća podatak koji će se upisati u gornju FIFO memoriju na slici. FIFO memorija služi za kratkotrajnu pohranu podataka te kao poveznica između dva procesa koja rade na različitim taktovima. Svaka FIFO memorija radi s podacima širine 8, 16 ili 32 bita.

Tablica 5.1. Priključci FIFO memorije

Signal	**Opis**
clk	signal takta
srst	sinkroni reset
din	ulazni podatak
wr_en	postavljanjem ovog signala upisuje se podatak s din porta u memoriju
rd_en	postavljanjem ovog signala prosljeđuje se podatak iz memorije na izlaz
dout	izlazni podatak
full	signal koji označava da je memorija puna, ne može se više pisati u nju
empty	signal koji označava da je memorija prazna, ne može se čitati iz nje

5.4 Implementacija na platformu ZedBoard

FPGA implementacija je izvedena korištenjem Xillybus sučelja. S Xillybusove stranice preuzet je paket s potrebnim datotekama gdje je implementirana povratna petlja za jednu FIFO memoriju. Sva funkcionalnost isprogramirana je u paketu pod nazivom xillydemo.vhd. Podaci se šalju s računala domaćina, prolaze kroz FIFO memoriju te se vraćaju natrag domaćinu. Taj dizajn modificiran je da radi filtriranje tako što je dodana je još jedna FIFO memorija te FIR filtar. Izgled dizajna koji je implementiran na razvojnoj pločici Zedboard prikazuje slika 5.6.



Slika 5.5. Implementacija FIR filtra u integriranom krugu Zynq

Kao što je vidljivo na samoj slici jedna od FIFO memorija će služiti za prijenos podataka od procesorskog sustava do FIR strukture koja je implementirana u FPGA dijelu, dok će druga izlazni podatak vraćati natrag procesorskom sustavu. Obje memorije su standardne veličine od 512 podataka po 32 bita. U gornju FIFO memoriju se upisuju podaci iz ulazne tekstualne datoteke ulaz.txt. Podaci su prilagođeni da rade s FIR filtrom koji prihvaća 16-bitne podatke na ulazu. Budući da FIFO memorije korištene u ovom primjeru rade s 32-bitnim podacima, ulazne podatke potrebno je pomaknuti za 16 bitova u lijevo čime omogućujemo korištenje gornjih 16 bitova za filtar. Xilllydemo datoteka je također modificirana na način da je dodana komponenta fir.vhd kojoj je ulaz spojen na izlaz gornje FIFO memorije, dok joj je izlaz spojen na ulaz donje FIFO memorije. Način spajanja FIFO memorija s FIR filtrom prikazan je isječkom VHDL koda na slici 5.7.

```
--  32-bit fifo + fir loop
FIR_int : fir
          generic map(
                      inputFirWidth     =>    inputFirWidth,
                      outputFirWidth    =>    outputFirWidth,
                      coeffsWidth       =>    coeffsWidth,
                      tap               =>    tap
                        )
          port map(
                   Din        => Din(31 downto 32-inputFirWidth),
                   clk        => bus_clk,
                   rst        => reset_32,
                   gsr        => '0',
                   Dout       => Dout(31 downto 32-outputFirWidth),
                   data_valid => outfifo_wr_en
                  );

fifo_in : fifo_32x512
   PORT MAP (
      clk => bus_clk,
      srst => reset_32,
      din => user_w_write_32_data,
      wr_en => user_w_write_32_wren,
      rd_en => ce_fs_low,
      dout => Din,
      full => user_w_write_32_full,
      empty => infifo_empty
      );


  fifo_out : fifo_32x512
    PORT MAP (
      clk => bus_clk,
      srst => reset_32,
      din => Dout,
      wr_en => ce_fs_low_delayed,
      rd_en => user_r_read_32_rden,
      dout => user_r_read_32_data,
      full => outfifo_full,
      empty => user_r_read_32_empty
      );
```

Slika 5.6. Isječak koda za spajanje FIFO memorija i FIR filtra

Cijeli dizajn radi na frekvenciji od 25Mhz, što je pola od maksimalne frekvencije koju može postići sama pločica. Za dijeljenje signala takta uveden je sklop mtu01.vhd koji ima zadaću propustiti svaki drugi signal takta. Razlog uvođenja mtu01.vhd sklopa je kako bismo spriječili čitanje prazne izlazne FIFO memorije. To se radi tako što zakasnimo signal ce_fs_low, koji odgovara taktu na frekvenciji od 25MHz, za jedan ciklus glavnog takta. Nakon što se završi filtriranje i dobijemo izlazni podatak, aktivira se zastavica data_valid koji signalizira drugoj FIFO memoriji da ima podatak za nju. Izlazni podatak će biti spremljen u

gornjih 17 bitova donje FIFO memorije, nakon čega ga možemo pročitati iz procesorskog sustava. Također potrebno je dobiveni rezultat posmaknuti u desno za 15 bitova jer FIFO memorije u dizajniranom sustavu rade s 32-bitnim podacima.

Podaci o iskorištenosti pojedinih dijelova Zedboard sklopa pri implementaciji niskopropusnog FIR filtra u sklopu xillydema navedeni su u tablici X, dok u tablici Y vidimo zauzeće samog FIR filtra.

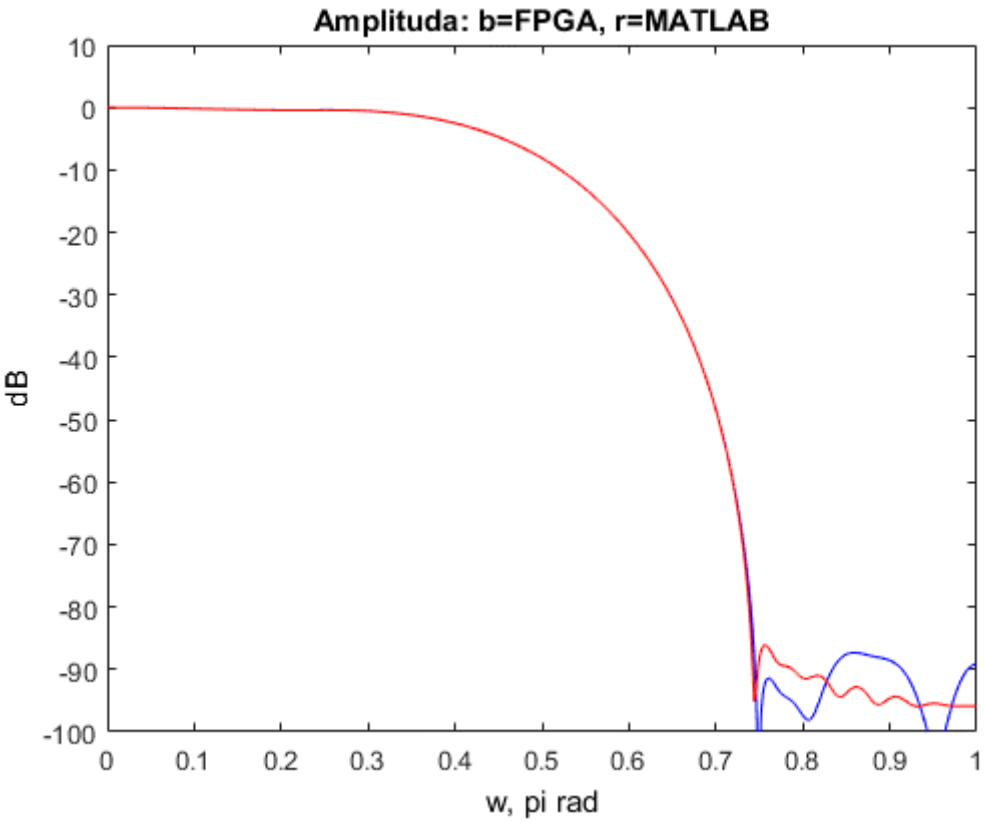
Tablica 5.2. Iskorištenost Zynq sklopa za kompletni dizajn

komponenta	**iskorišteno**	**dostupno**	**postotak iskorištenosti %**
FF	4870	106400	4.56
LUT	4829	53200	8.88
LUTRAM	335	17400	1.93
BRAM	5	140	3.57
IO	85	200	42.50
BUFG	4	32	12.50
PLL	1	4	25.00

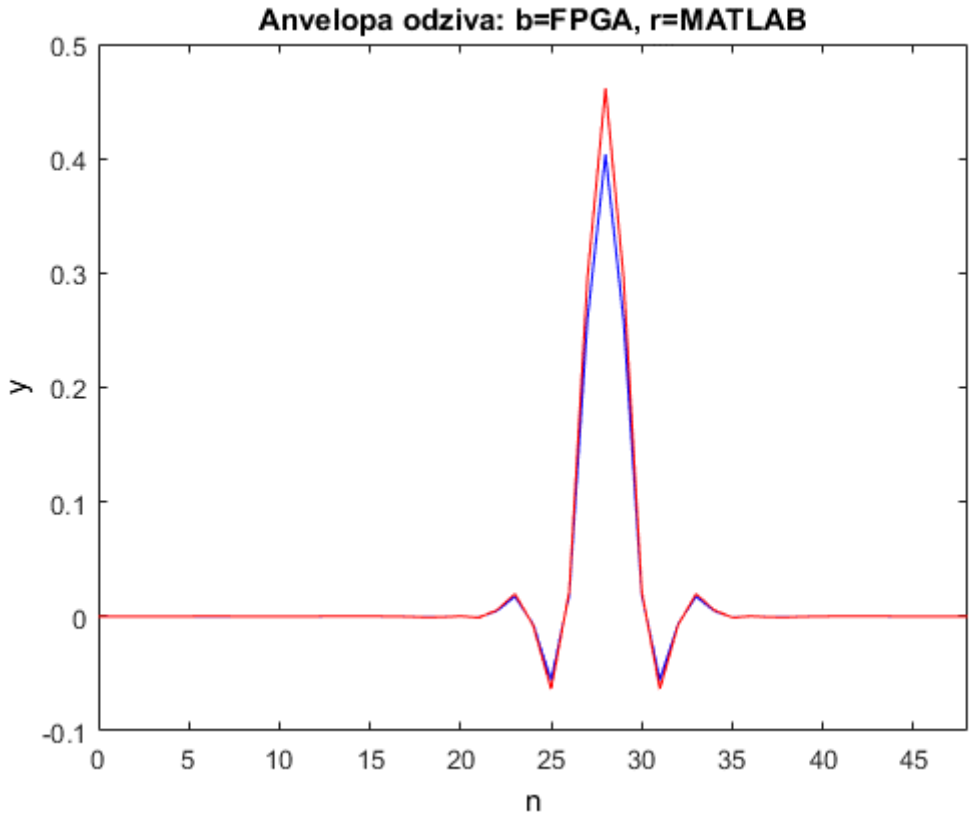
Tablica 5.3. Iskorištenost Zynq sklopa za FIR filtar

komponenta	**iskorišteno**	**dostupno**	**postotak iskorištenosti %**
FF	164	106400	0.15
LUT	220	53200	0.41

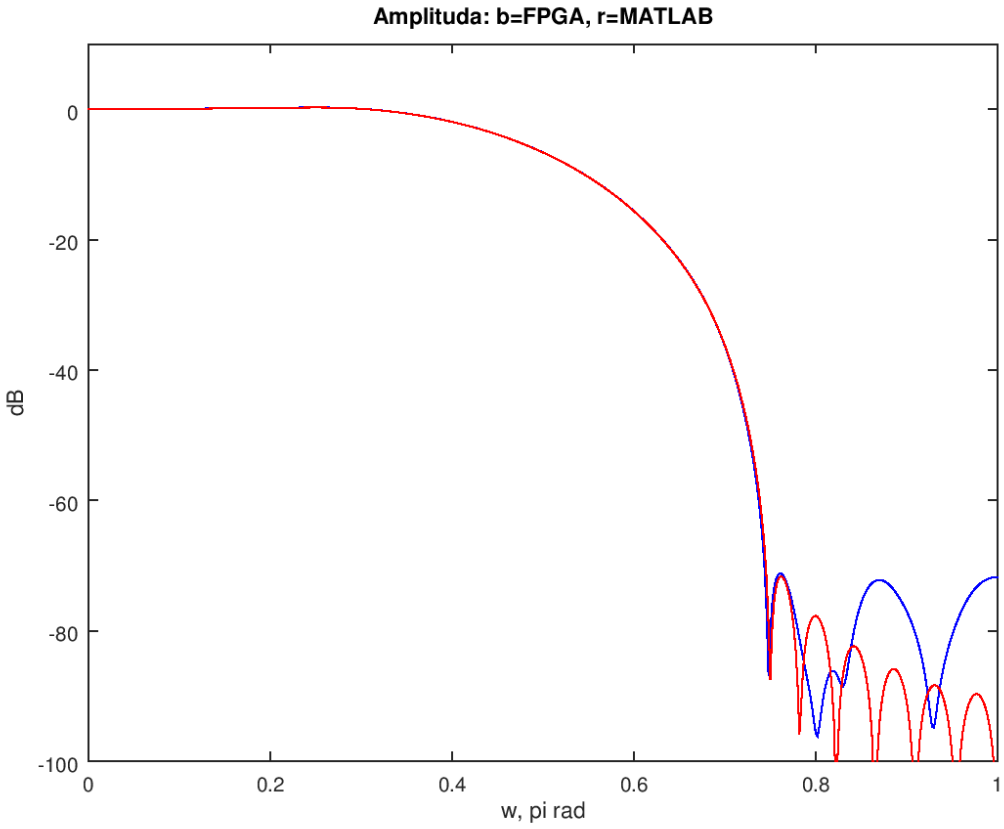
Za provjeru ispravnosti rada kao ulazna pobuda uzimaju se uzorci niskopropusnog raised cosine signala. Dobivena amplitudna karakteristika odziva niskopropusnog FIR filtra zajedno s referentnom amplitudom prikazana je na slici 5.8, dok na slici 5.9 možemo vidjeti Anvelopu odziva u usporedbi s referentnom iz Matlaba. Navedeni filtar ima 9 koeficijenata širine 8 bitova, te frekvenciju propuštanja, odnosno gušenja od $\omega_p = \frac{\pi}{4}$ i $\omega_s = \frac{3\pi}{4}$.
Usporedba amplitudnih karakteristika kompenzacijskog filtra za CIC decimator zajedno s referentnom amplitudom iz Matlaba prikazana je slikom 5.10, a Anvelopa odziva u usporedbi s referentnom iz Matlaba vidljiva je na slici 5.11. Kompenziramo CIC filtar reda *N*=6, faktora decimacije *R*=32, te pojasa propuštanja od $\omega_p = \frac{\pi}{2}$.



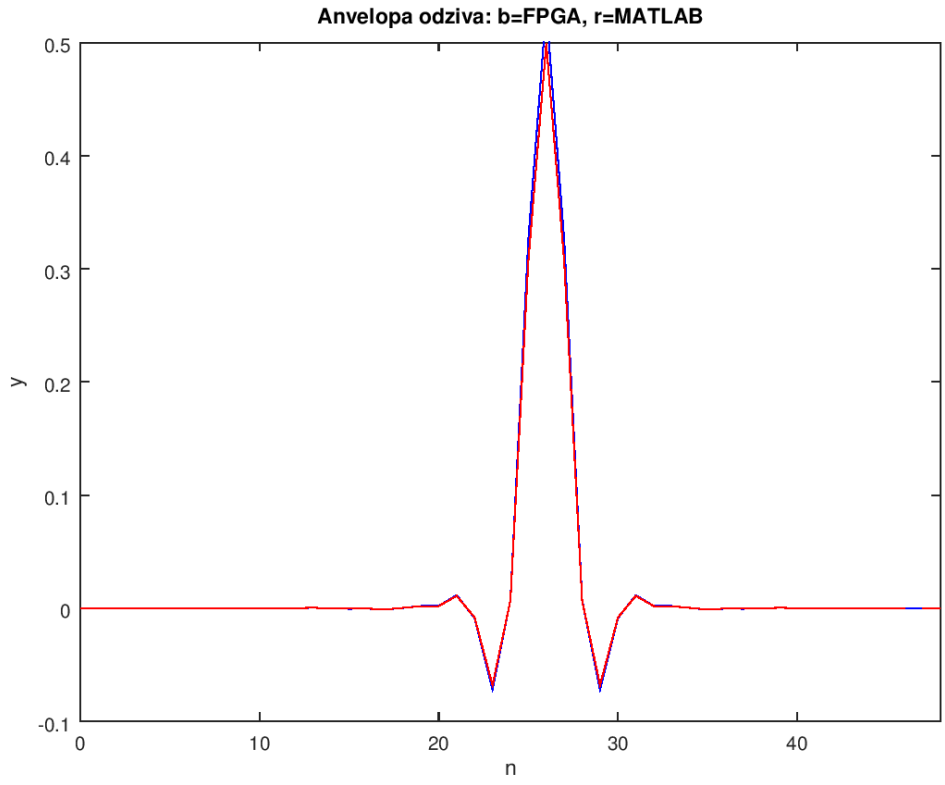
Slika 5.7. . Usporedba amplitudnih karakteristika za niskopropusni FIR



Slika 5.8. Usporedba anvelopi odziva za niskopropusni FIR



Slika 5.9. Usporedba amplitudnih karakteristika za kompenzacijski filtar



Slika 5.10. Usporedba anvelopi odziva za kompenzacijski filtar

6 Zaključak

U radu je opisan dizajn i implementacija jednostavnih FIR filtara bez množila. Demonstrirana su dva različita dizajna FIR filtara, prvi od kojih je niskopropusni FIR filtar, dok je drugi kompenzacijski filtar za CIC decimatore. Dizajn filtara, tj. propračuni njihovih parametara napravljani su u Matlab razvojnom okruženju.

Filtri bez množila su jako dobri za FPGA implementaciju budući da se sva množenja, koja su inače nužna kako bi se izvršilo filtriranje ulaznog signala, pretvaraju u posmake. Kako bi to bilo moguće prostor traženja koeficijenata za filtre definiran je kao predznačene cjelobrojne potencije broja dva.

U sklopu rada razvijeni su fizički ostvarivi modeli niskopropusnog FIR filtra te kompenzacijskog filtra za CIC decimatore u jeziku za opis sklopovlja VHDL. Implementacija filtara bez množila je vrlo učinkovita s aspekta iskorištenja dostupnih resursa na integriranom krugu Zynq. Dizajn je rađen po uzoru na xillydemo.vhd koji je dostupan na stranicama xillybusa za razvojnu pločicu Zedboard. Procesorski sustav radi na linux operacijskom sustavu xillinux i komunicira s FPGA dijelom pomoću FIFO memorija.

Za potrebe verifikacije rada razvijena su odgovarajući ispitni slučajevi u razvojnom okruženju Matlab gdje je izvršena simulacija rada na odgovarajućem pobudnom signalu. Dobiveni rezultati dokazuju da se korištenjem razvijenih Matlab funkcija za dizajn filtara dobivaju dobre karakteristike filtara.

7 Literatura

[1] Aljosa Dudarin; Goran Molnar; Mladen Vucic, "Simple multiplierless CIC compensators providing minimum passband deviation", Proceedings of the 10th International Symposium on Image and Signal Processing and Analysis, Sept. 2017, pp. 70-73

[2] Sanjit K. Mitra; "Digital Signal Processing: A computer-based approach"; McGrawHill Higher Education, 2006

[3] D. Petrinović, D. Petrinović, R. Bregović, H. Babić, B. Jeren; "Digitalna obradba signala, upute za laboratorijske vježbe"; Fakultet elektrotehnike i računarstva, Zavod za elektroničke sustave i obradbu informacija, 2003.

[4] Mladen Vučić i Goran Molnar; "Alati za razvoj digitalnih sustava, materijali za predavanja"; Fakultet elektrotehnike i računarstva, Zavod za elektroničke sustave i obradbu informacija, 2009.

[5] E. B. Hogenauer, “An economical class of digital filters for decimation and interpolation,” IEEE Trans. Acoust., Speech, Signal Process., vol. 29, no. 2, pp. 155–162, Apr. 1981.

[5] Xilinx. URL https://www.xilinx.com.

[6] Xillybus. URL https://www.xillybus.com.

[7] FIR. URL https://en.wikipedia.org/wiki/Finite_impulse_response

8 Sažetak

Dizajn i FPGA implementacija jednostavnih FIR filtara bez upotrebe množila

U radu su opisani dizajn i implementacija jednostavnih FIR filtara bez množila. Kod takvih struktura filtara koeficijenti su izraženi kao predznačene potencije broja dva, što omogućuju uporabu posmaka umjesto množila. Takva implementacija filtara omogućuje obradu signala u sustavima gdje se iziskuje velika brzina rada. Za dizajn filtara korišteno je razvojno okruženje Matlab. Modeli filtra razvijeni su u jeziku za opis sklopovlja VHDL-u te su implementirani na programabilnim logičkim poljima (engl. FPGA) Zedboard razvojne pločice. Procesorski sustav Zedboard pločice radi pod operacijskim sustavom Xillinux, distribucije linuxa prilagođene radu s pločicom. Komunikacija procesorskog sustava i programabilnog dijela ide Xillybus sabirničkim sustavom. Za prijenos podataka Xillybusom u oba smjera koriste se FIFO memorije. Također u sklopu rada razvijeno je odgovarajuće ispitno okruženje u Matlabu koje služi za provjeru funkcionalnosti rada filtara.

Ključne riječi: FIR filtar, CIC kompenzator, globalna optimizacija, minimax aproksimacija, predznačena potencija broja dva, implementacija bez množila, VHDL, Zedboard, Xillinux, Xillybus, FIFO memorije

9 Summary

Design and FPGA implementation of simple multiplierless FIR filters

In this thesis the design and implementation of simple multiplierless FIR filters has been described. When dealing with those structures, coefficients are expressed as signed powers of number two, which allows usage of shift instead of multiplication. That kind of implementation of filters allows usage of filters in systems that are expected to work on high frequencies. For design of filters Matlab environment was used. Filter models have been developed in hardware description language VHDL and implemented on Zedboard programable System-on-chip. Processing part of Zedboard works on Xillynux, linux version for ARM. It creates a simple interface for us to transfer information to and from the FPGA called Xillybus. FIFO memories are used for data transfer between FPGA and processing system. For verification of results using implemented filters, test environment has been designed in Matlab.

Keywords: FIR filter, CIC compensator, global optimisation, minimax aproximation, signed power of two, multiplierless implementation, VHDL, Zedboard, Xillinux, Xillybus, FIFO memories

Privitak A: funkcija firmm.m

```
% FIRMM - minimax design of linear phase fir filters
%   using 5 input variables L - number of coefficients(implemented only for odd
%   number of coefficients), B - wordlength, wp - passband ((must be assigned with
%   a*pi, where a is element [0,1]) , ws - stopband (same as wp), K weight factor
%   function returns coefficients of linear phase low pass fir filter
%
%   Example:
%   h=firmm(9,8,0.25*pi,0.75*pi,1);
%

function h=firmm(L,B,wp,ws,K)
tic
debug = 1;
n=100;
limit=5000;

CoeffNum=ceil(L/2);
PermNum=(2*B+1)^CoeffNum;

wpb=linspace(0,wp,n);
wsb=linspace(ws,pi,n);
w=[wpb, wsb]';
k=0;
opt=1;

%coefficients space
coeff=[-2.^(1:B), 0, 2.^(1:B)];

%desired response
Hd= [ones(n,1);zeros(n,1)];

%weight function
W=[ones(n,1)./K; ones(n,1)];

% optimisation
epsi = realmax;
while (opt)
  if (k <fix(PermNum/limit))
    C = permn(coeff, CoeffNum, (1:limit)+limit*k);
    k=k+1;
    else
    C = permn(coeff, CoeffNum, limit*k+1:PermNum);
```

```
   opt=0;
  end
  C=C((C(:,1)+2*sum(C(:,2:end),2))~=0,:);

  % matrix of cos
  HH=cos(w*(0:CoeffNum-1));
  HH(:,2:end)=2*HH(:,2:end);
  %matrix of FIR filters
  H=HH*C';
  [e,ind] = min(( max( abs((H./H(1,:) - Hd) .* W )))) ;
  if e < epsi
    epsi=e;
    c = C(ind,:);
    h=[fliplr(c(2:end)),c];
   %only positive summ of coeffs
    if sum(h)<0
      h=-h;
    end
  end
end
```

Privitak B: funkcija fircompmm.m

```
% CICCOMPMM - minimax design of CIC compensators
%   using 5 input variables N - order of CIC filter, R - decimation ratio,
%   L - number of coefficients(implemented only for odd number of coefficients),
%   B - wordlength, wp - passband (must be assigned with a*pi, where a is element [0,1])
%   function returns coefficients of CIC compensator
%
%   Example:
%   h=ciccompmm(6,32,5,8,0.5*pi);
%


function h=ciccompmm(N,R,L,B,wp)
tic
n=100;
debug=1;
w=linspace(0,wp,n);
CoeffNum=ceil(L/2);
%coefficients space
coeffs=[-2.^(1:B), 0, 2.^(1:B)];

 %low rate CIC response
Hcic=((1/R).*(sin(w/2)./(sin(w/(2*R))))).^N;
Hcic(1,1)=1;
```

```
%matrix of coeffs
C = permn(coeffs, CoeffNum);

%matrix of cos
HH=cos(w(:)*(0:CoeffNum-1));
HH(:,2:end)=2*HH(:,2:end);

%matrix of comp
Hcomp=HH*C';

%error for minimisation
epsi = realmax;
[e,ind] = min( max((Hcic'.*Hcomp)./Hcomp(1,:))- min((Hcic'.*Hcomp)./Hcomp(1,:)));
if e < epsi
  epsi=e;
  c = C(ind,:);
  h=[fliplr(c(2:end)),c];
%only positive summ of coeffs
  if sum(h)<0
     h=-h;
  end
end
```