# Implementacija FIR filtra zasnovanog na FPGA za digitalnu obradu zvuka u realnom vremenu

**Krajcar, Krunoslav**



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Diplomski sveučilišni studij**

# IMPLEMENTACIJA FIR FILTRA ZASNIVANOG NA FPGA ZA DIGITALNU OBRADU ZVUKA U REALNOM VREMENU

**Diplomski rad**

**Krunoslav Krajcar**

**Osijek, 2018.**

# Sadržaj

# 1. UVOD

Procesiranje signala bitan je proces koji se koristi u suvremenim tehnologijama. Jedan od oblika procesiranja signala, kojim se uklanjaju neželjene komponente signala je filtriranje. Filtri se koriste prilikom uklanjanja neželjenih frekventnih komponenta signala ili prilikom izdvajanja specifične frekventne komponente iz signala. Ovisno o obliku signala koji je potrebno filtrirati moguće je koristiti analogne ili digitalne filtre.

Analogni filtri sadrže analogne komponente i implementacija samog filtra je jednostavan proces. Analogne komponente nisu savršene, i u ovisnosti o vanjskim uvjetima karakteristike komponenta se mijenja, a samim time i karakteristika implementiranog filtra. Zbog navedenih nesavršenosti analognih komponenta i otežane implementacije visokog reda filtra, postoje ograničenja na performanse filtra. Ako se želi dobiti bolje i stabilnije performanse filtra, koriste se digitalni filtri.

Također, kod analognih filtara karakteristika filtra ovisi o komponentama filtra, i promjenom komponente mijenja se karakteristika filtra. Digitalni filtri imaju mogućnost lakše promjene karakteristika filtra, jer se vrši promjenom koeficijenata filtra spremljenih u memoriji.

Implementacija digitalnih filtra je kompleksnija u usporedbi s analognim, jer su za digitalne filtre potrebni diskretni signali. Zbog toga je potrebno koristiti analogno-digitalne pretvarače što također povećava cijenu filtra. No korištenjem diskretnih podataka i digitalne elektronike dobivamo sklop gdje vanjski utjecaji manje utječu na rad sklopa. Također implementacija filtra visokog reda olakšana je u usporedbi s analognim filtrima.

Zaključuje se ako je potrebno implementirati filtar, gdje zahtjevi na performanse filtra nisu veliki, idealno je koristiti analogni filtar. No ako su potrebni filtri gdje je bitna stabilnost filtriranja, potreban je filtar visokog reda i potrebna mogućnost lake promjene karakteristike filtra, koriste se digitalni filtri.

Zbog sve većih unaprjeđenja digitalnih tehnologija i FPGA (eng. Field programmable gate array), olakšan je proces implementacije digitalnog filtra, što je rezultiralo korištenjem digitalnih filtara u sve većem broju slučajeva. Karakteristike digitalnih filtara su: rekonfigurabilnost i laka izvedba uz pomoć procesora za obradu digitalnih signala (eng. Digital signal processor, DSP) implementiranih u FPGA. Tema diplomskog rada upravo se bavi rekonfigurabilnosti FIR filtra,

gdje implementirani filtar ima mogućnost promjene koeficijenta i reda filtra. Tema je proširena na implementaciju dvokanalnog evilizatora.

Prvi dio diplomskog rada bavi se karakteristikama filtra i principom rada FIR filtra, te objašnjava sve korištene tehnologije korištene tijekom procesa dizajna. Zatim je objašnjen dizajn implementiranog FIR filtra i ekvilizatora, prikazane blok sheme te opisane sve komponente. Posljednji dio bavi se mjerenjem prijenosnih karakteristika te usporedba s teorijskim rezultatima MatLab-a.

## 1.1 Zadatak diplomskog rada

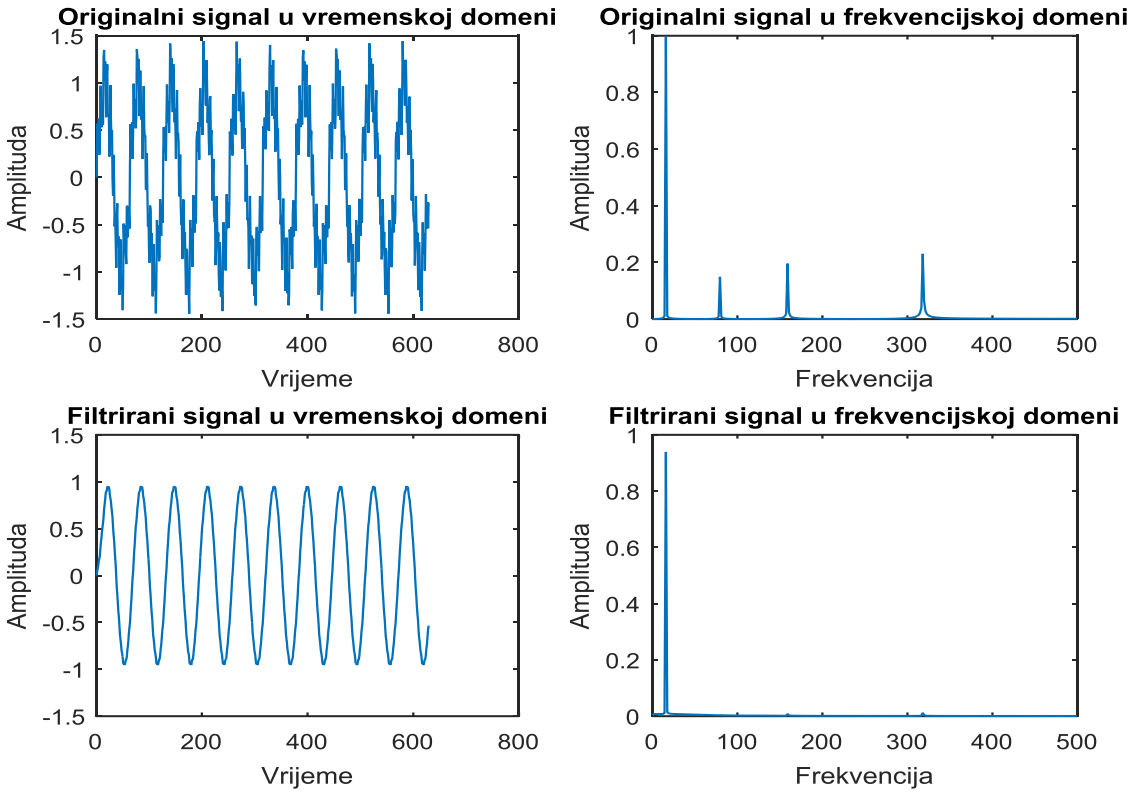
Digitalno filtriranje zvuka jedan je od važnijih postupaka digitalne obrade zvuka, s ciljem postizanja selektivnosti pojedinih frekvencijskih komponenti. Diplomskim radom potrebno je realizirati FIR (eng. Finite impulse response) filtar na Zynq-7000 SoC (engl. System on Chip) koji se nalazi u sklopu Zybo (ZYnq BOard) razvojne platforme. Filtar treba vršiti procesiranje zvuka u realnom vremenu s mogućnošću dinamičke promjene njegovih koeficijenata. Filter je potrebno realizirati programibilnom logikom korištenjem VHDL-a za opis fizičke arhitekture. Upravljanje filtrom u vidu unošenja koeficijenata filtra od strane korisnika treba realizirati na dvojezgrenom procesoru ARM Cortex-A9 u programskom jeziku C. Za analogno-digitalnu te digitalno-analognu pretvorbu koristi u razvojnu platformu ugrađen SSM2603 audio kodek kojeg treba konfigurirati prema potrebama navedenog sustava.

# 2. FIR FILTAR

## 2.1 Filtri

Filtri se koriste prilikom obrade signala za uklanjanje neželjenih frekvencija. Koriste se u svrhe odvajanja signala i uklanjanje izobličenja iz signala. Odvajanje signala izvodi se kada jedan signal sadrži više informacija na raznim frekvencijama. Kako bi se izdvojila željena informacija provodi se filtriranje signala na željenoj frekvenciji. Filtar će ukloniti sve neželjene frekvencije i na izlazu dobiva se signal koji sadrži samo željenu informaciju. [1]

Zbog nelinearnosti komponenti sustava, signali često sadrže promjene u frekvencijskom spektru, koje nisu postojale u izvornom signalu. Moguće je smanjiti utjecaj navedenih izobličenja upotrebnom filtra. Primjer navedenog slučaja promatramo prema slici 2.1. gdje originalni signal sadrži signal informacije i neželjene frekventne komponente. Korištenjem filtra uklanjaju se više frekvencije i dobiva se jedno harmonijski signal.




***Sl. 2.1.*** *Filtriranje signala gdje su uklonjene visokofrekvente komponente*

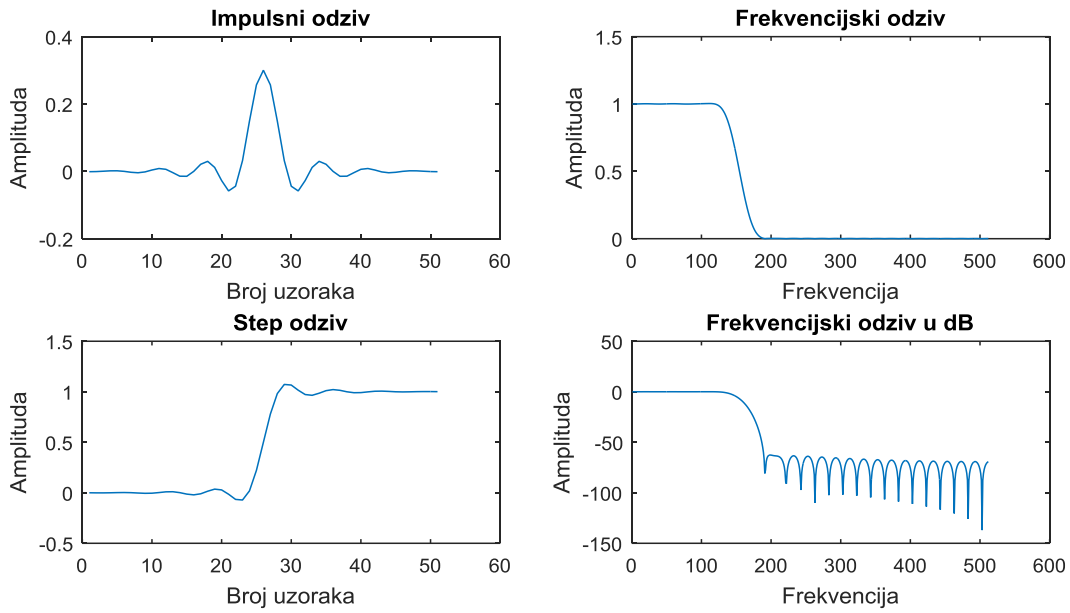
Zbog navedenih razloga filtri nalaze primjenu u brojnim područjima kao što su: telekomunikacije, radio prijenos, procesiranje audio signala itd. Primjer filtriranja signala prikazano je prema slici 2.1. [1]

Ovisno o postavljenim zahtjevima prilikom filtriranja, mogu se koristiti analogni ili digitalni filtri. Karakteristike digitalnih i analognih filtra mogu se vidjeti prema tablici 2.1.

***Tab. 2.1*** *Usporedba karakteristika digitalnih i analognih filtra*

| Digitalni filtri | Analogni filtri |
|---|---|
| • Visoka preciznost<br>• Linearna faza<br>• Mogućnost fleksibilnog i adaptivnog filtriranja<br>• Laka simulacija i dizajn<br>• Frekvencija ulaznog signala limitirana frekvencijom uzorkovanja<br>• Potrebni analogno digitalni pretvornik (eng. Analog digital converter, ADC), digitalno analogni pretvornik (eng. Digital analog converter, DAC) i procesor za obradu digitalnih podataka (Digital singla processor, DSP). | • Niska preciznost (tolerancija komponenta)<br>• Nelinearna faza<br>• Otežano adaptivno filtriranje<br>• Otežana simulacija i dizajn<br>• Nisu ograničeni frekvencijom uzorkovanja već frekvencijskim ograničenjima komponenta<br>• Nema potrebe za ADC, DAC i DSP |

Svaki filtar se može opisati s jednim od tri parametra: impulsnim odzivom, skokovitim odzivom te frekvencijskim odzivom. Navedeni podaci daju informaciju o ponašanju filtra u vremenskoj i frekvencijskoj domeni. Impulsni odziv filtra dobiva se tako što se na ulaz filtra dovodi impulsni signal te mjerenjem izlaza. Kao što prikazuje slika 2.2. dobiveni impulsni odziv se sastoji od sinusoida čija amplituda eksponencijalno pada. U teoriji to bi značilo da je krivulja beskonačno duga, no u praksi uzorke čija je amplituda manja od amplitude šuma sistema se zanemaruju. [1]

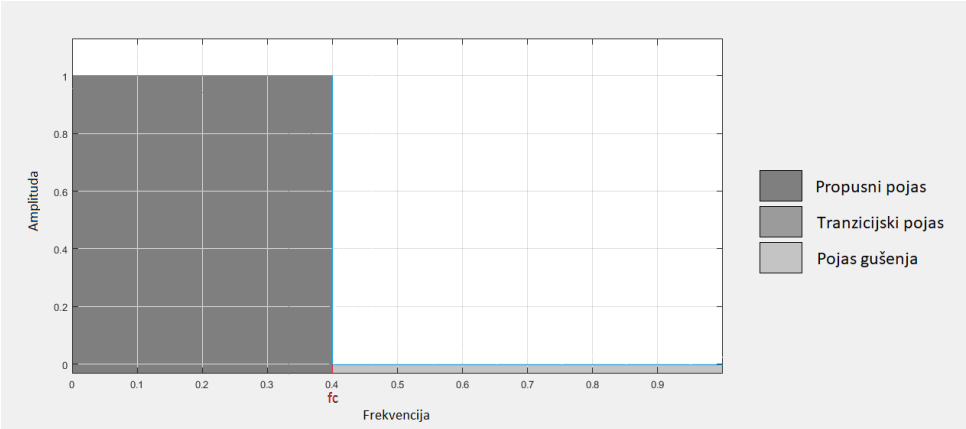



***Sl 2.2.*** *Impulsni, stepenasti i frekvencijski odziv filtra*

Ukoliko je definiran samo jedan od tri parametra, ostala dva se mogu lako izračunati. Primjenom Fourierove transformacije na impulsni odziv dobiva se Frekvencijski odziv, dok se integriranjem impulsnog dobiva stepenasti odziv. [1]

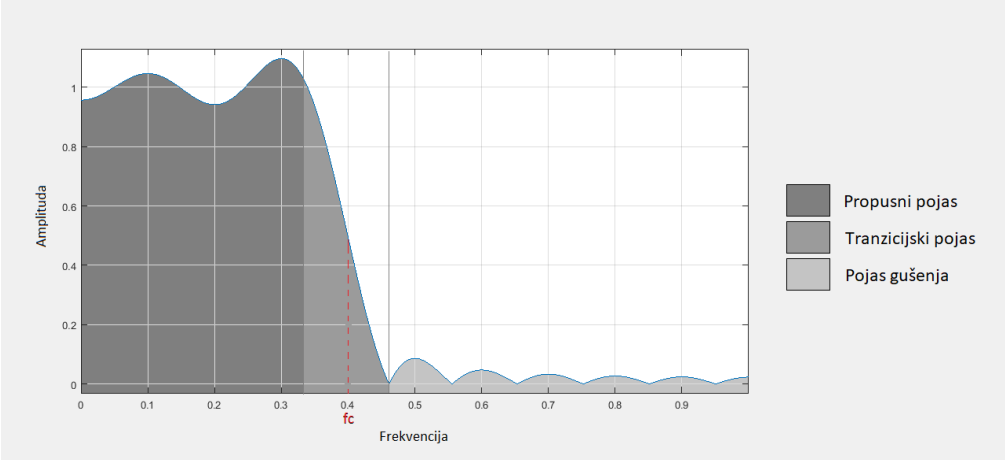
## 2.2 Karakteristike filtra u frekvencijskoj domeni

Uloga filtra je da određene frekvencije propušta nepromijenjene i istovremeno guši ostale frekvencijske komponente. Pri promatranju i analiziranju prijenosne funkcije grafa u frekvencijskoj domeni, potrebno je definirati područja interesa grafa. Ta područja su pojas propusta, pojas gušenja, tranzicijski pojas te granična frekvencija. Pojas propusta se odnosi na frekvencije koji trebaju ostati nepromijenjene kroz filtar, dok se pojas gušenja odnosi na frekvencije koje filtar treba ukloniti. Tranzicijski pojas je pojas između propusnog pojasa i pojasa. Kod idealnog filtra tranzicijski pojas je minimalan, kao što se može vidjeti prema slici 2.3, sastoji se od propusnog pojasa i pojasa gušenja. Granična frekvencija se definirana kao frekvencija gdje amplituda prijenosne karakteristike opadne za -3dB. [1]



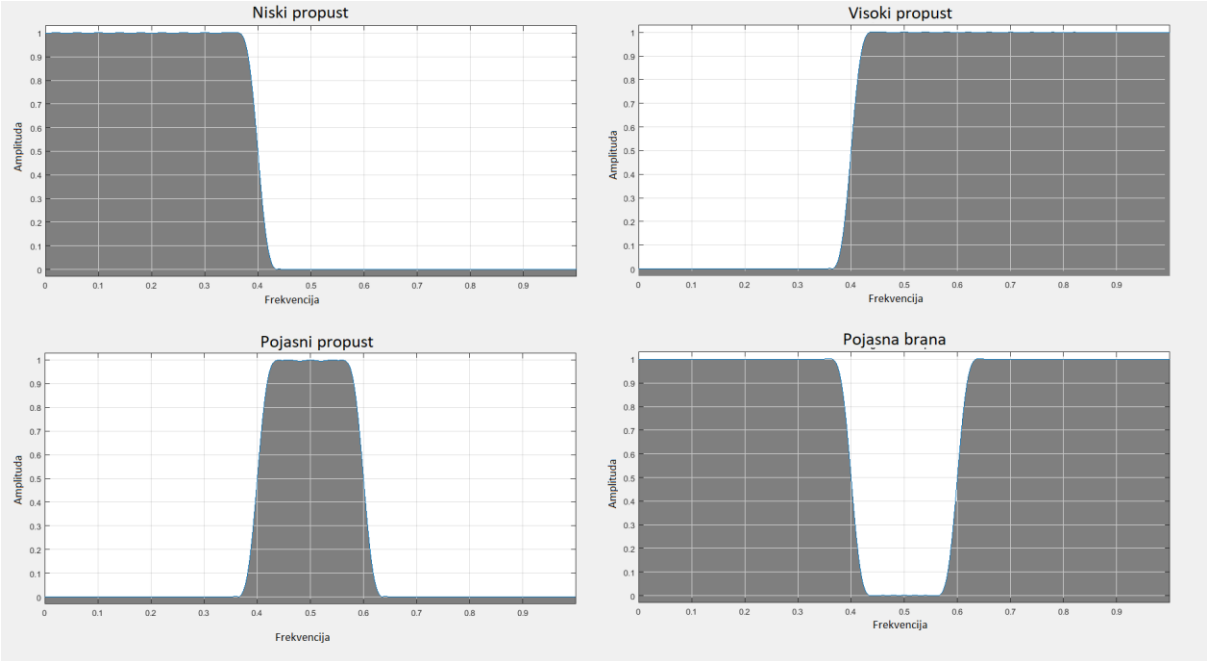

***Sl 2.3.*** *Prijenosna karakteristika idealnog filtra u frekvencijskoj domeni*

Frekvencijsku karakteristiku stvarnog filtra prikazana je slikom 2.4. Filtar propušta sve frekvencije manje od granične frekvencije fc, dok ostale guši.




**Sl 2.4**. *Prijenosna karakteristika stvarnog filtra u frekvencijskoj domeni dobivena u MatLab-u*

Promatranjem prijenosne karakteristike na temelju frekvencija koje propušta, prema slici 2.5, filtri se dijele u četiri tipa: niskopropusni, visokopropusni, pojasnopropusni te pojasna brana.




***Sl 2.5.*** *Vrste filtra u odnosu na frekvencijski odziv*

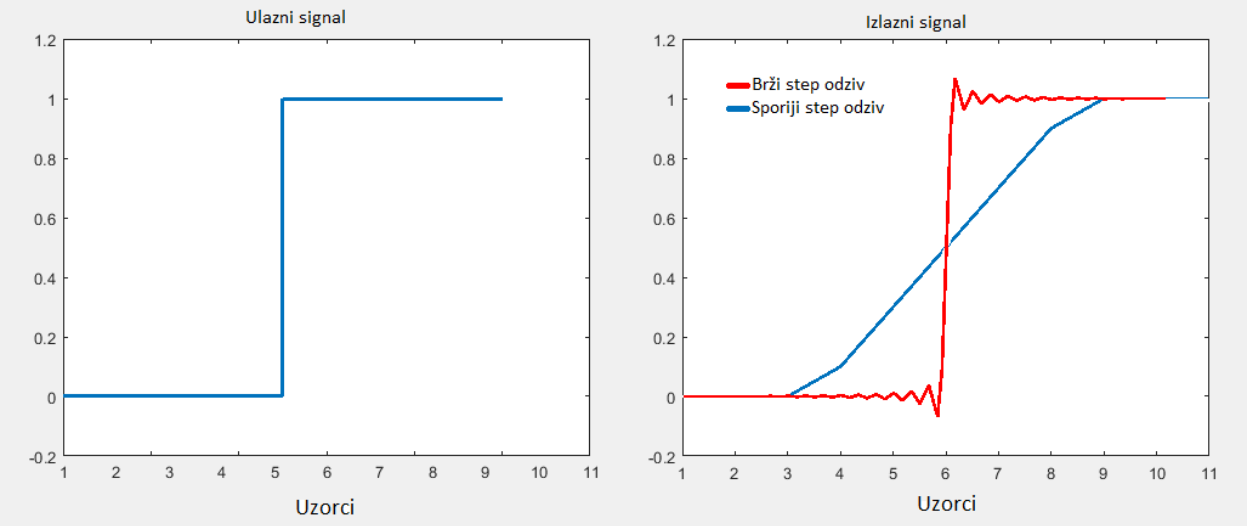
## 2.3 Karakteristike filtra u vremenskoj domeni

Analiza filtra u vremenskoj domeni se vrši promatranjem funkcije skokovitog odziva. Skokoviti odziv se mjeri tako što se na ulaz filtra postavi signal koji u vrlo kratkom vremenu promjeni stanje amplitude i pri tome se promatra izlazni signal. Ovisno o dizajnu filtra promjena vrijednosti amplitude traje određeni vremenski period. Slika 2.6 prikazuje ulaz i izlaz dvaju filtra s različitim skokovitim odzivima.

Trajanje skokovitog odziva treba trajati što kraće, kako bi bilo moguće razaznati događaje unutar signala. Vrijeme prijelaza skokovitog odziva je definirano je potrebnim brojem impulsa da se amplituda signala promjeni iz 10% u 90% amplitude. Slike 2.6 prikazuje dva izlazna signala različitog skokovitog odziva, bržem je potreban 1 uzorak za promjenu, dok je sporijem potrebno 4.

Potrebno je obratiti pažnju na istitravanje amplitude izlaznog signala. Na temelju odziva prikazanih slikom 2.6. vidi se karakteristika Chebyshevog i Butterworth. Chebysheov filtar sadrži valovitu krivulju koja opisuje odziv, no strmina je veća. Butterworth filtar nema valovitu krivulju, no strmina je manja, uz korištenje istog reda filtra.

Na temelju izgleda skokovitog odziva moguće je zaključiti linearnost faze. Ukoliko su gornja i donja polovina skokovite funkcije simetrične riječ je o linearnoj fazi. Ukoliko su asimetrične tada je faza nelinearna. [1]




***Sl 2.6.*** *Odaziv filtra na skokovitu pobudu u vremenskoj domeni*

## 2.4 Digitalni filtri

Najčešće korišteni digitalni filtar je linearan vremenski nepromjenjivi LTI (eng. Linear time invariant) filtar. LTI filtar temelji se na procesu linearne konvolucije gdje izlazni signal y, jednak konvoluciji ulaznog signala x i impulsnog odziva filtra f. [2]

$$y[n]=x[n]\bullet f[n]=\sum_k x[k]\,f[n-k]=\sum_k f[k]\,x[n-k] \tag{2-1}$$

LTI filtri se dijele na FIR (eng. Finitive impulse response), koji se sastoji od konačnog broja uzoraka, time se prethodno navedena konvolucijska suma svede na konačni broj zbrajanja. IIR (eng. Infinite impulse response), koji nema konačan broj uzoraka, zahtjeva da se provede beskonačan broj zbrajanja. [2]

## 2.5 FIR filtar

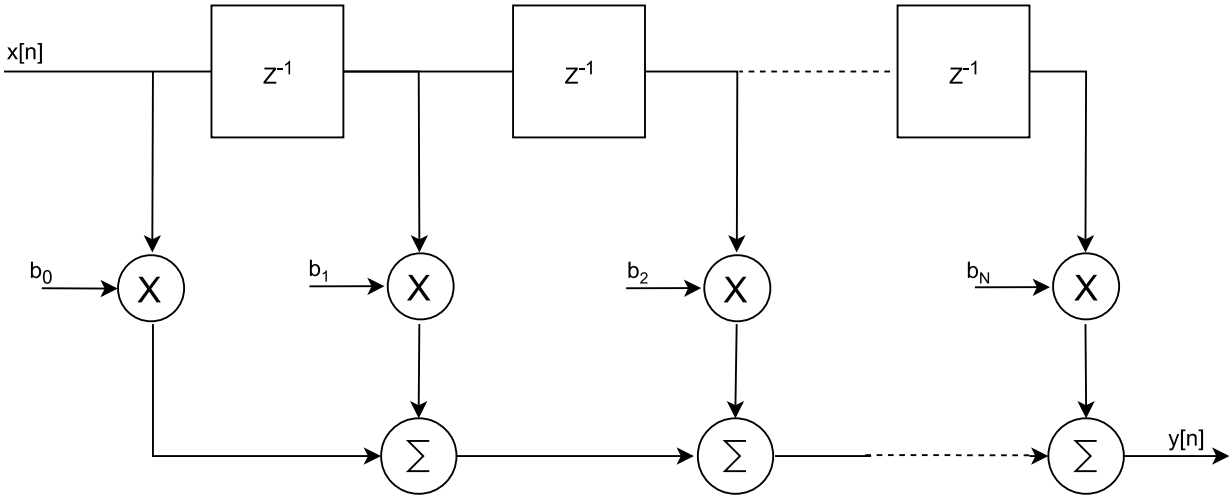
Princip rada FIR filtra temelji se na relaciji 2-2, izvođenjem konvolucije ulaznog signala te funkcije impulsnog odziva filtra.

$$y[n]=x[n]\bullet f[n]=\sum_{k=0}^{N-1} f[n]\,x[n-k] \tag{2-2}$$

$$y[n]=b_0x[n]+b_1x[n-1]+\ldots+b_Nx[n-N] \tag{2-3}$$

$$y[n]=\sum_{i=0}^{N}b_ix[n-i] \tag{2-4}$$

Izlaz filtra y jednak je sumi umnožaka N prošlih stanja uzoraka ulaznog signala i odgovarajućeg uzorka impulsnog odziva. Blokovska shema FIR filtra na slici 2.7 prikazuje kako FIR filtar sadrži blokove za kašnjenje, blokove za množenje te blokove za zbrajanje.[2]




***Sl 2.7.*** *Blok shema FIR filtra*

Princip rada FIR filtra može se demonstrirati na primjeru filtra pomične srednje vrijednosti gdje se uklanja neželjene frekvencije iz signala.

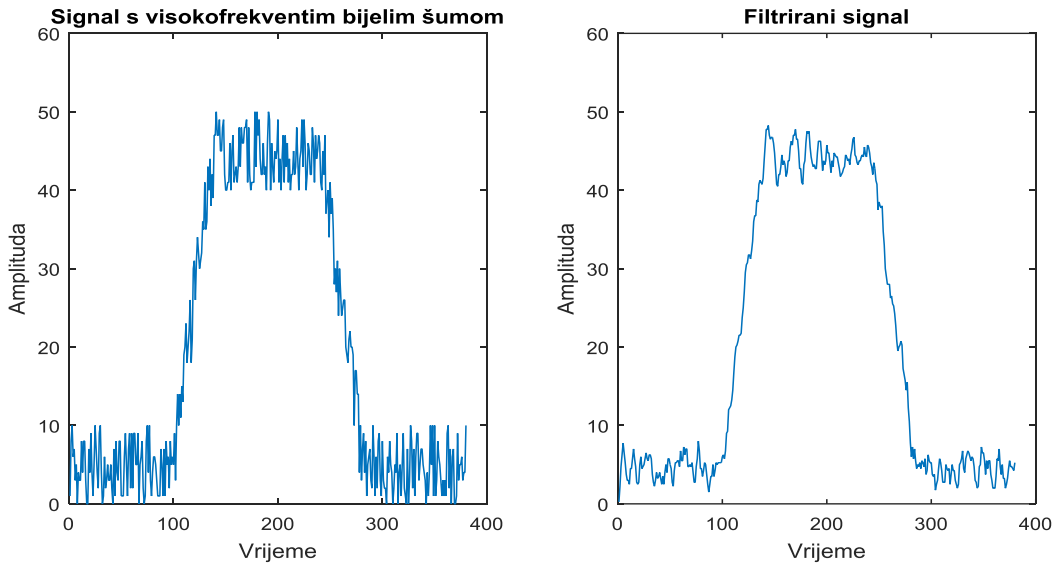
### 2.5.1 Filtar pomične srednje vrijednosti

Filtar pomične srednje vrijednosti kao izlazni signal daje srednju vrijednost N uzoraka ulaznog signala. Svaki od ulaznih uzoraka množi se sa koeficijentom koje je jednak 1/N, gdje je N red filtra.

$$y[i]=\frac{1}{N}\sum_{j=0}^{N-1}x[i+j] \tag{2-5}$$

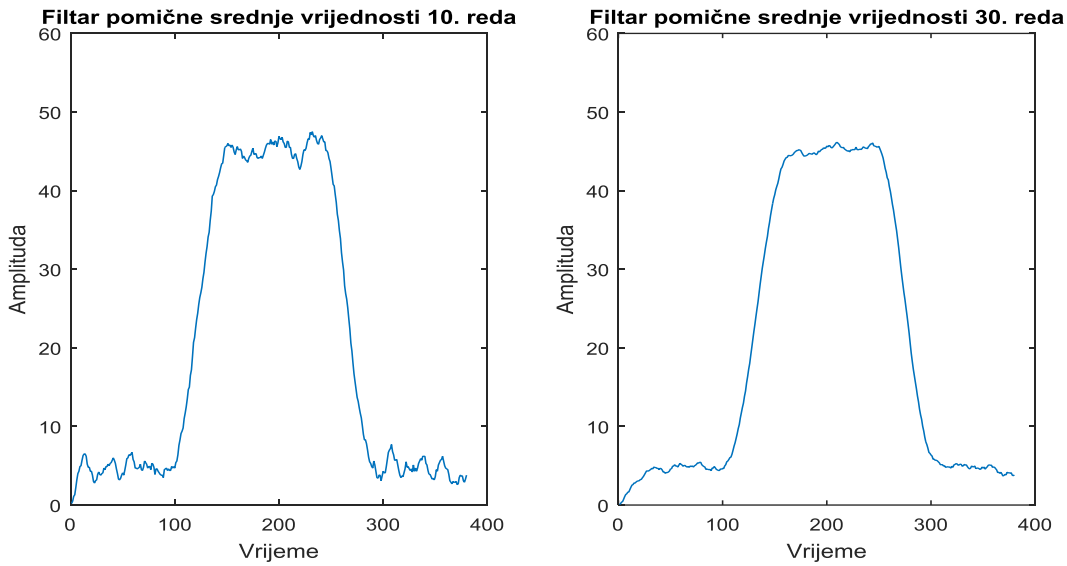
Kada bi se koristio filtar pomične vrijednosti 4 reda, formula prikazuje kako je izlazni signal u svakom trenutku jednak srednjoj vrijednosti trenutnog ulaza u filtar i 3 prethodne vrijednosti uzoraka.

Filtar pomične vrijednosti često se primjenjuje za uklanjanje visokofrekventnih komponenta bijelog šuma iz signala. Slika 2.8 prikazuje ulaz i izlaz signala filtra pomične srednje vrijednosti 4. reda. Filtrirani signal sadrži manji broj visokofrekventnih komponenta.



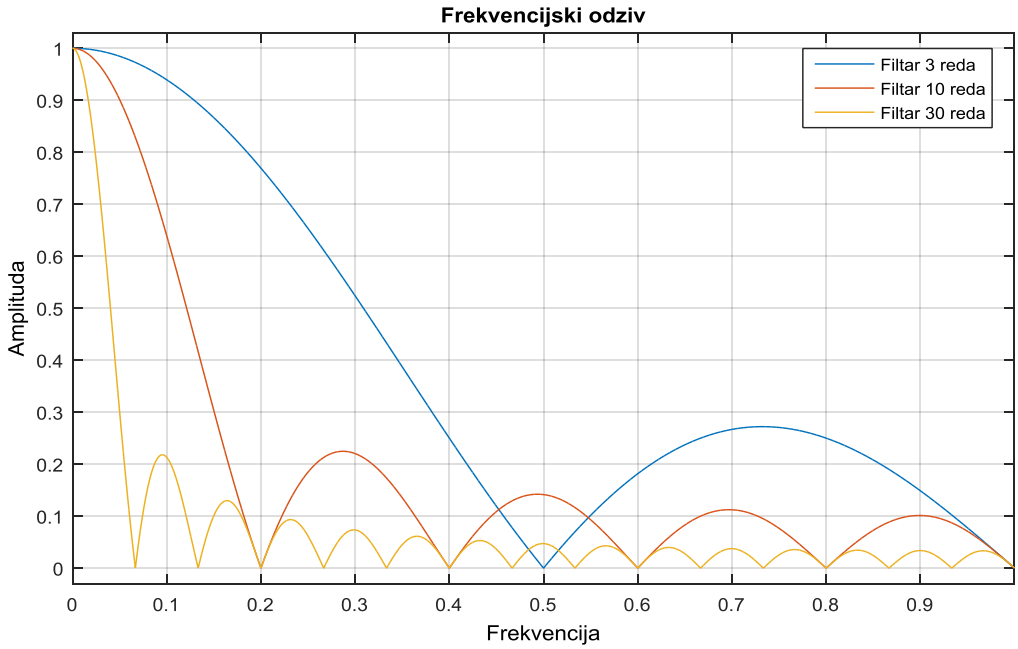

***Sl. 2.8*** *Signal prije i poslije upotrebe filtra pomične srednje vrijednosti prikazan u vremenskoj domeni*

Povećanjem reda filtra uklanja se više visokofrekventnih komponenta. Slika 2.9. prikazuje 10. I 30. Red filtra




***Sl. 2.9*** *Signal filtriran filtrom pomične srednje vrijednosti 10. i 30. reda*

Korištenjem filtra pomične srednje vrijednosti većeg reda, sinusoida poprima oblik sličniji originalnom signalu. Povećavanjem reda filtra, smanjuje se granična frekvencija kao što prikazuje slika 2.10.




***Sl. 2.10.*** *Frekvencijski odziv filtra pomične srednje vrijednosti u ovisnosti o redu filtra*

Filtar pomične srednje nije praktično koristiti ukoliko želimo specifičnu graničnu frekvenciju jer se granična frekvencija mijenja redom filtra. Odabir granične frekvencije izvodi se tako što se računaju koeficijenti filtra. Neovisno želi li se implementirati visoki propust, pojasni propust ili pojasna brana, potrebno je dizajnirati niskopropusni filtar.

## 2.6 Računanje koeficijenata filtra

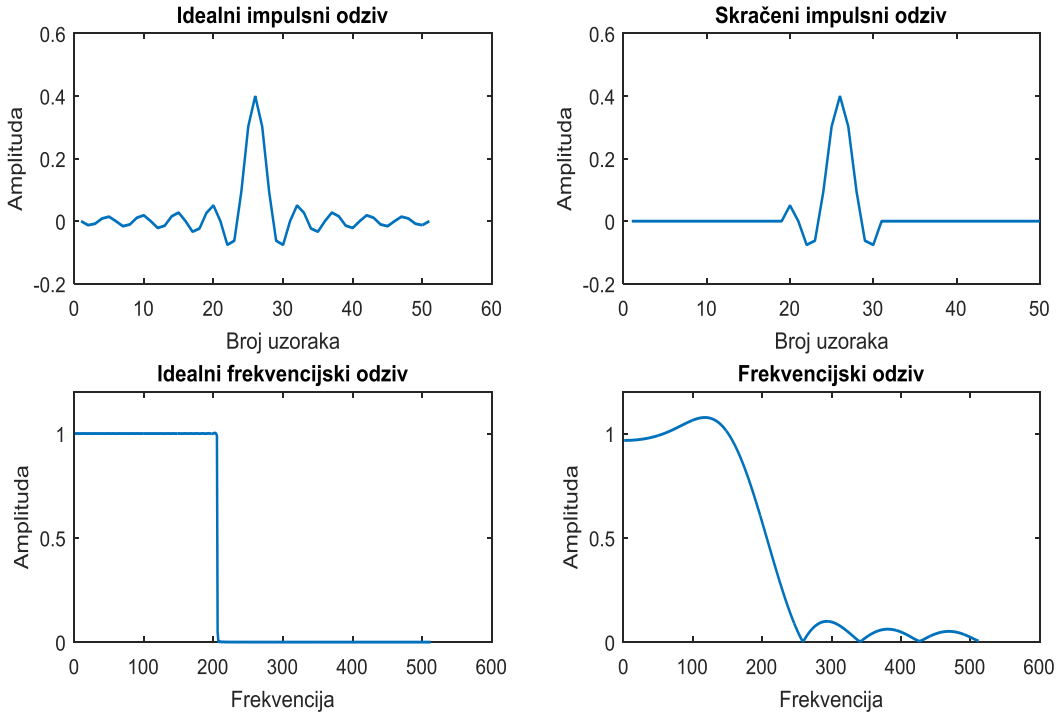
### 2.6.1 Koeficijenti niskopropusnog filtra

Za dizajn filtra potrebno je znati njegov impulsni odziv. Kako je impulsni odziv jednak prijenosnoj karakteristici, za dizajn nisko propusnog filtra potrebno je započeti tako što se koristi prijenosna karakteristika idealnog niskopropusnog filtra definirana relacijom 2-6.

$$h[i] = \frac{\sin(2\pi f_c i)}{i\pi} \tag{2-6}$$

Unosom željene granične frekvencije $f_c$ dobiva se impulsni odziv idealnog filtra. Problem kod dobivenog impulsnog odziva je što funkcija seže u beskonačnost, i to nije moguće implementirati. Ukoliko se za impulsni odziv koristi konačan broj uzoraka, tada je filtar moguće implementirati, no gubi se karakteristika idealnog filtra.

Impulsni i frekvencijski odziv pri korištenju impulsnog odziva s 10 uzoraka prikazan je slikom 2.11. Frekvencijski odziv gdje je korišten skraćeni impulsni odziv uvelike se razlikuje od frekvencijskog odziva idealnog filtra.




***Sl. 2.11.*** *Ovisnost impulsnog i frekvencijskog odziva*

Povećanjem reda filtra, dobiva se prijenosna karakteristika sličnija idealnom filtru. Drugi način kako poboljšati prijenosnu karakteristiku je korištenjem metode prozora.

### 2.6.2 Metoda prozora

Metoda prozora je postupak gdje se impulsni odziv filtra množi s definiranom funkcijom prozora i dobiva skraćeni impulsni odziv. Funkcija prozora je matematička funkcija koja je jednaka nuli za vrijednosti izvan definiranog intervala. Postupak skraćivanja impulsnog odziva izveden u poglavlju 2.6.1. naziva se još i postupak korištenja pravokutnog prozora s 10 uzoraka, jer

funkcija prozora sadrži konstantnu amplitudu 1 unutar definiranog intervala. Slika 2.14. prikazuje grafički oblik pravokutnog prozora.

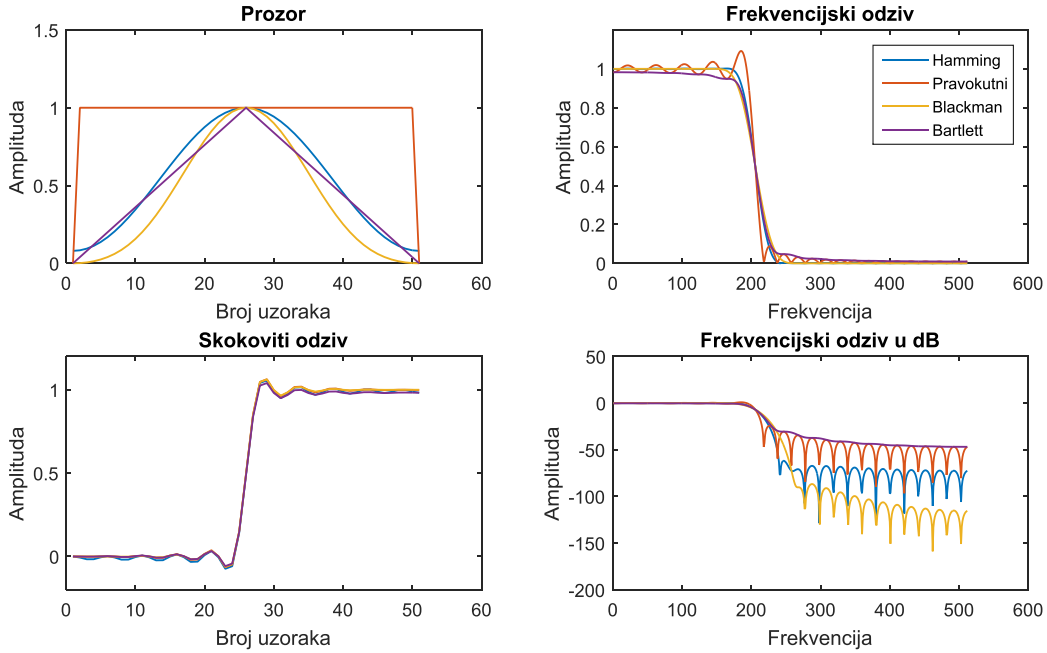
Korištenjem raznih funkcija prozora dobivaju se različiti impulsni odzivi. Tablica 2.2 prikazuje metode prozora te funkcije kojima su opisane. [1]

***Tab 2.2.*** *Metode prozora i funkcije koje ih opisuju*

| | |
|---|---|
| Pravokutni | $\omega(n)=1$ |
| Bartlett | $\omega(n)=1-\frac{2\left\lvert n-\frac{M}{2}\right\rvert}{M}$ |
| Hanning | $\omega(n)=0,5-0,5\cos\left(\frac{2\pi n}{M}\right)$ |
| Hamming | $\omega(n)=0,54-0,46cos\left(\frac{2\pi n}{M}\right)$ |
| Blackman | $\omega(n)=0,42-0,5cos\left(\frac{2\pi n}{M}\right)+0,08cos\left(\frac{4\pi n}{M}\right)$ |

Proveden je izračun funkcija pravokutnog, Hamming, Blackman i Bartlett prozora i prikazani frekvencijski odzivi filtra 50. reda. Prema slici 2.12. prikazana je ovisnost frekvencijskog odziva o korištenoj metodi prozora.

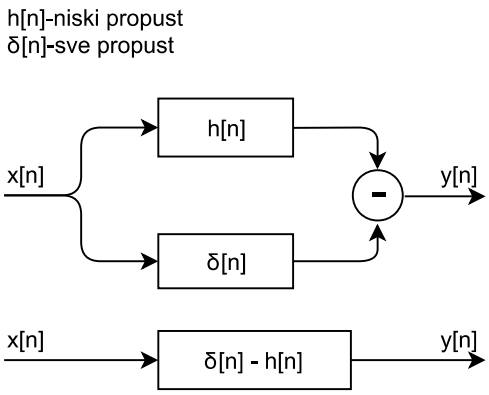
Frekvencijski odziv, pri korištenju pravokutnog prozora, opisan je krivuljom koja je valovita. Ovisno o odabiru prozora utječe se na valovitost frekvencijskog odziva, strminu na graničnoj frekvenciji i gušenje signala unutar pojasa gušenja i pojasa propuštanja, može se vidjeti prema slici 2.12.




***Sl. 2.12.*** *Ovisnost frekvencijskog odziva o korištenoj metodi prozora*

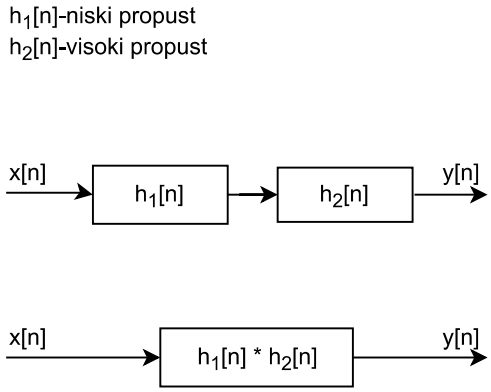
### 2.6.3. Koeficijenti visokopropusnog filtra, pojasnopropusnog filtra i pojasne brane

Visoko propusni filtar može se dobiti na dva načina, oba uključuju oduzimanje nisko propusnog i sve propusnog filtra. Slika 2.13 prikazuje oduzimanje u seriji i paraleli.



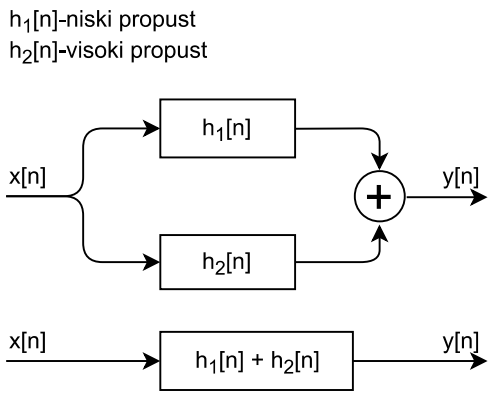

***Sl. 2.13.*** *Postupak izračuna koeficijenta visokopropusnog filtra*

Pojasni propust se izvodi spajanjem ulaznog signala na ulaz nisko propusnog filtra (slika 2.14). Izlaz niskopropusnog filtra spaja se na ulaz visokopropusnog filtra. Izlazni signal imati će prijenosnu karakteristiku pojasnog propusta. Drugi način je upotrebom konvolucije navedenih filtara.




***Sl. 2.14.*** *Postupak izračuna koeficijenta pojasnopropusnog filtra*

Pojasna brana se izvodi zbrajanjem izlaznih signala niskopropusnog i visokopropusnog filtra kao na slici 2.15.




***Sl. 2.15*** *Postupak izračuna koeficijenta filtra pojasne brane*

Izvođenjem postupka računanja koeficijenata filtra, te implementiranjem istih u FIR filtar moguće je bez promjene sklopovlja izvesti filtre raznih prijenosnih karakteristika.

## 2.7 Aritmetika za racionalne brojeve

Koeficijenti impulsnog odziva filtra racionalni su brojevi intervala normiranih vrijednosti od -1 do 1. Kako bi bilo moguće vršiti aritmetiku potrebnu za filtriranje signala, koeficijente filtra moguće je prikazati koristeći aritmetiku s fiksnim ili pomičnim zarezom.

Aritmetika s pomičnim zarezom podržava širok interval brojeva. Zbog toga se aritmetika s pomičnim zarezom koristi u sustavima koji sadrže velike i male brojeve a potrebno je brzo procesiranje. Kod aritmetike sa pomičnim zarezom pozicija decimalne točke se može mijenjati. Sadrži tri dijela za opis broja: eksponent, mantisa i predznak. Mantisa opisuje broj koji želimo prikazati, dok eksponent određuje položaj decimalne točke.

Aritmetika sa fiksnim zarezom ne sadrži pomičnu decimalnu točku, te je potrebno definirati interval brojeva koji želimo opisati. Korištenjem aritmetike fiksnog zareza s beskonačnim brojem bitova, moguće je prikazati sve racionalne brojeve. Kako se u praksi koristi konačan broj bitova, dobiva se aproksimacija najbližeg racionalnog broja. [3]

Uzevši u obzir način implementacije u FPGA u diplomskom radu korištena je aritmetika s fiksnim zarezom, jer je potrebno koristiti manje resursa FPGA i potrebna je manja latencija u odnosu na aritmetiku s pomičnim zarezom.[4]

Također interval koeficijenta filtra nalaze se u intervalu -1 do 1 pa nije potrebno koristiti aritmetiku pomičnog zareza koja pokriva interval svih brojeva.

Kada se cijeli brojevi prikazuju u binarnom sustavu, svaki bit ima težinu $2^N$, kod opisa racionalnih brojeva težina je $0{,}5^N$. Tablica 2.3 prikazuje kako ovisno o broju korištenih bitova, mijenja se težina bita. Korištenjem aritmetike fiksnog zareza broj kao što je 2.125 u binarnom obliku je 10.001. [3]

***Tab. 2.3*** *Težinske vrijednosti bita*

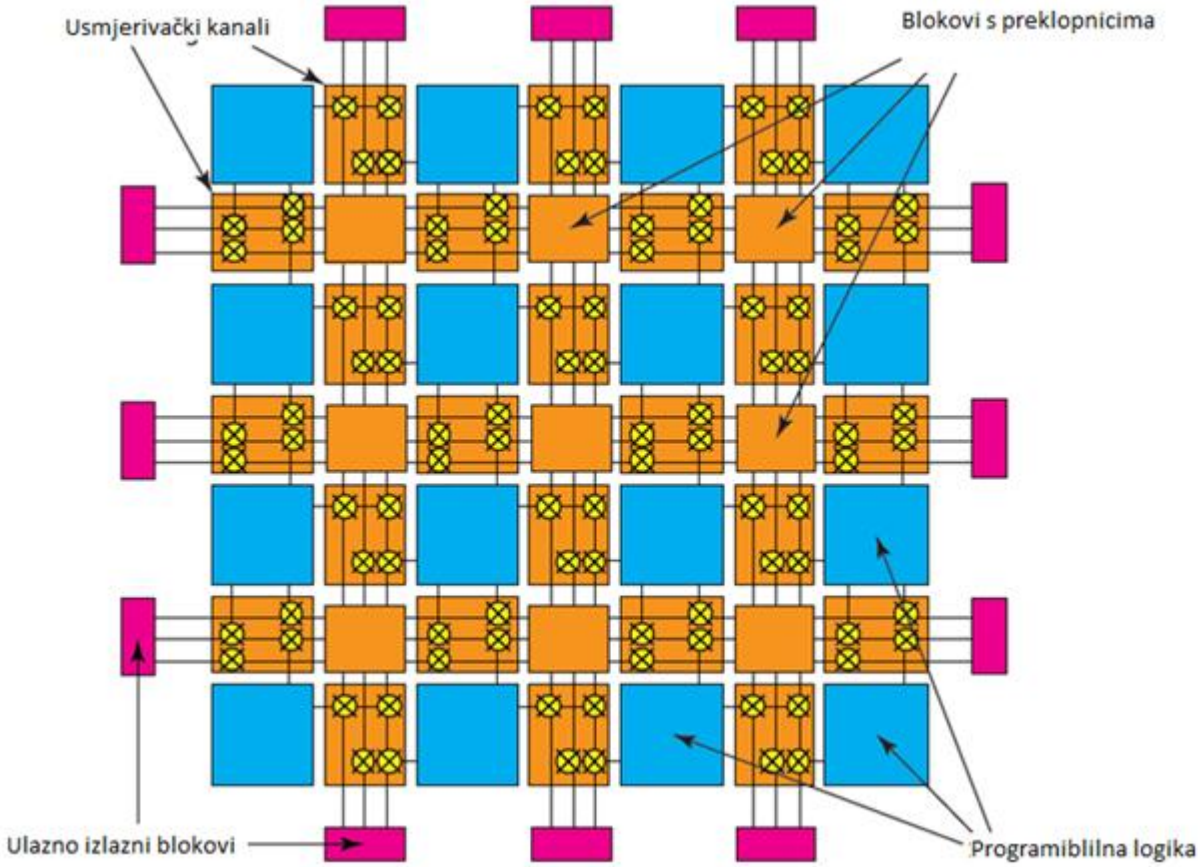
| Broj bita | N | 2 | 1 | 0 | | 0 | 1 | 2 | N |
|---|---|---|---|---|---|---|---|---|---|
| Težina | $2^N$ | 4 | 2 | 1 | . | 0,5 | 0,25 | 0,125 | $(0,5)^N$ |

U diplomskom radu korištena je aritmetika fiksnog zareza s 32 bita za opis broja, gdje je 1 bit predznak, 1 bit korišten za cjelobrojni dio, a 30 za racionalni. Takav zapis podržava interval vrijednosti od -2 do 1.9999999991 sa zadovoljavajućom preciznosti.

# 3. KORIŠTENE TEHNOLOGIJE

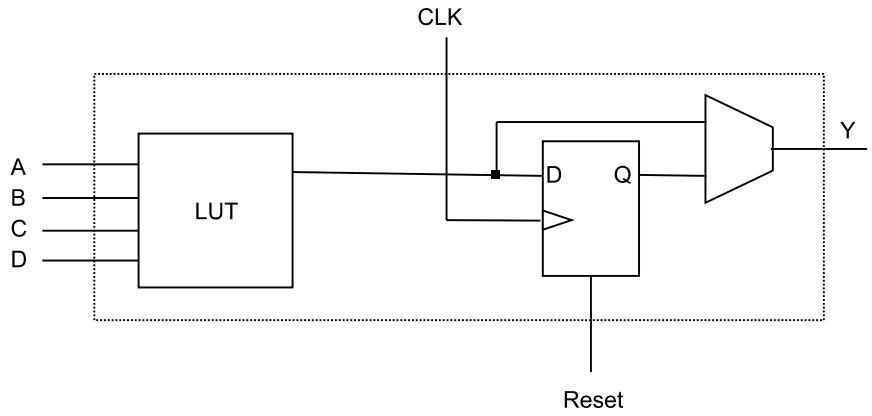
## 3.1 FPGA

FPGA (eng. Field Programable Gate Array) je polje programibilnih logičkih blokova i memorijskih elemenata koji su međusobno spojeni koristeći programibilne preklopnike. Programibilni dio FPGA omogućuje konfiguriranje i međusobno spajanje logičkih blokova na fizičkoj razini kako bi se implementirao željeni digitalni sustav. Arhitektura FPGA prikazana je prema slici 3.1. i sastoji se od međusobno povezanih blokova programibilne logike.



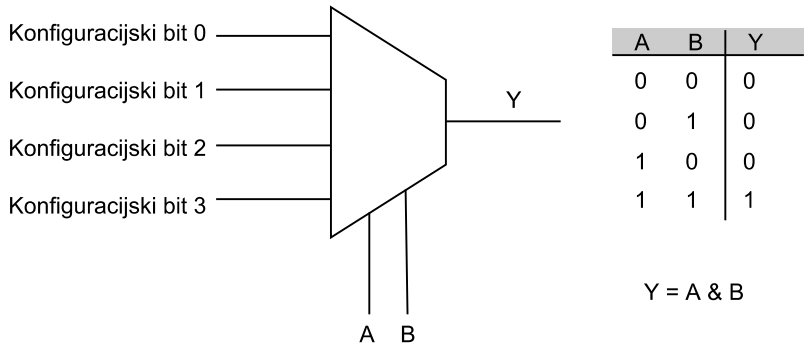

***Sl. 3.1*** *Blok shema arhitekture FPGA [5]*

Programibilni logički blokovi nazivaju se, ovisno o proizvođaču, CLB (eng. Configurable logic block) ili LAB (end. Logic array block). CLB se sastoji od nekoliko isječaka (eng. Slice). Isječak se sastoji od 1 ili više preglednih tablica (eng. Look up table, LUT), multipleksera te D bistabila kao što prikazuje slika 3.2. D bistabil služe za sinkronizaciju, dok multiplekser odabire sinkroni ili askinroni način rada.




***Sl. 3.2*** *Isječak CLB bloka koji se sastoji od: LUT-a, D bistabila i mulipleksera*

LUT je komponenta pomoću koje je se implementiraju Booloean funkcije. U memorijski element se upisuju moguća izlazna stanja. Slika 3.3 prikazuje konfiguraciju LUT-a s 2 ulaza, gdje je vrijednost Y spremljena u memoriji. Na temelju logičkih stanja signala A i B odabire se adresa i pristupa se memoriji LUT-a. Vrijednost spremljena na definiranoj adresi postavlja se na izlaz Y.[5]




***Sl. 3.3*** *LUT element i prikaz logičke tablice za konfiguraciju rada kao I logička vrata*

LUT sa 2 ulaza i 1 izlaznom prikazan je slikom 3.3. Svaki od četiri konfiguracijska bita mogu promijeniti funkciju LUT-a, čime se dobivaju potpuno programibilna logička vrata. Na slici 3.3. je prikazana konfiguracija LUT-a kao I logičkih vrata. Kombinacijom n-bitnih LUT tablica može se izvršiti svaka željena Boolean funkcija s n ulaza. [5]

CLB blokovi su međusobno povezani usmjerivačkim kanalima i preklopnim blokovima. Upotrebom programabilnih usmjerivačkih kanala i preklopnih blokova ostvarena je konfigurabilna mreža koja povezuje blokove FPGA. Usmjerivački kanali također su spojeni na ulazno izlaze blokove kako bi bilo moguće spajanje CLB blokova na ulazno izlaze priključke.[5]

Kako je rastao broj tranzistora unutar FPGA, počela je i implementacija takozvanih blokova specifične primjene, koji nisu potpuno programabilni. Korištenjem blokova specifične primjene gubi se na fleksibilnosti u usporedbi s CLB blokovima, no dobiva se bolju efikasnost.

Kako bi se olakšalo korištenje matematičkih funkcija kao što su zbrajanje ili množenje unutar FPGA dizajna, započela je implementacija DSP blokova. DSP je specijaliziran mikroprocesor koji je dizajniran s namjerom izvođenja operacija vezanih za procesiranje digitalnih signala. DSP blokovi u FPGA mogu biti reprogramirani do neke razine, no nisu fleksibilni kao CLB.

Također implementirani blokovi specifične primjene su memorijski elementi kao BRAM ili DRAM. BRAM je blokovska interna memorija malog kapaciteta i potrebna je mala latencija za pristup memoriji. DRAM je distribuirana eksterna memorija velikog kapaciteta, no potrebno je latencija od nekoliko taktova za pristup memoriji. Kao memorijski elementi se mogu koristiti D bistabili ili LUT-ovi iz isječaka. No spremanjem većeg broja podataka koristio bi se i velik broj CLB isječaka, čime je ograničen broj ostalih blokova koji se mogu implementirati u dizajnu. Spremanjem podatka u BRAM smanjuje broj iskorištenih CLB blokova, no upis podataka je sporiji. Latencija potrebna za upis podatka u D bistabil je 1 takt, dok je za BRAM 1 ili 2 takta kako bi se pristupilo jednoj memorijskoj adresi.

Podaci koji se upisuju u BRAM mogu imati varijabilnu širinu podatka, dok D bistabil sprema 1 bit podatka. Tipičan BRAM sadrži 32 Kbita podatka, koji mogu biti konfigurirani kao 32K x 1 bit, 16K x 2 bit, 8K x 4 bit, itd. BRAM –i se također mogu spojiti u kaskadu kako bi stvorili veće memorije. Ovisno o proizvođaču, FPGA može sadržavati i druge blokove specifične primjene. [5]

## 3.2 ZYNQ-7000

Zynq-7000 je SoC (eng System on Chip) razvijen od Xilinxa. Softverska programabilnost ARM procesora i hardverska programabilnost FPGA omogućuje stvaranje efikasnih dizajna visokih performansa, sve na jednoj platformi. Zbog toga Zynq-7000 SoC pronalazi primjenu u područjima kao što su: Automotiv, IP i pametne kamere, LTE radio, Multifunkcionalni printeri itd. [6]

Arhitektura Zynq-7000 sastoji se procesorskog sustava (eng. Processing System) i programabilne logike (eng. Programmable Logic).

Procesorski sustav sastoji se od četiri dijela: Jedinica za obradu aplikacije (eng. Applicatin Processor Unit, APU), Memorijskih sučelja (eng. Memory Interface), Ulazno izlazne periferije (eng. I/O Peripherals) i Preklopnika (eng. Interconect) kako prikazuje slika 3.4.

Glavni dio jedinice za obradu aplikacije je dvojezgreni ARM-Cortex A9 procesor. APU također sadrži:

- Kontrolor općih interupata (eng. General interrupt Controller, GIC),
- Tri Watchdog timera (eng. WDT)
- Dva Timera/Countera (eng. TTC)
- 8 kanalni DMA
- Dvokanalni RAM itd.

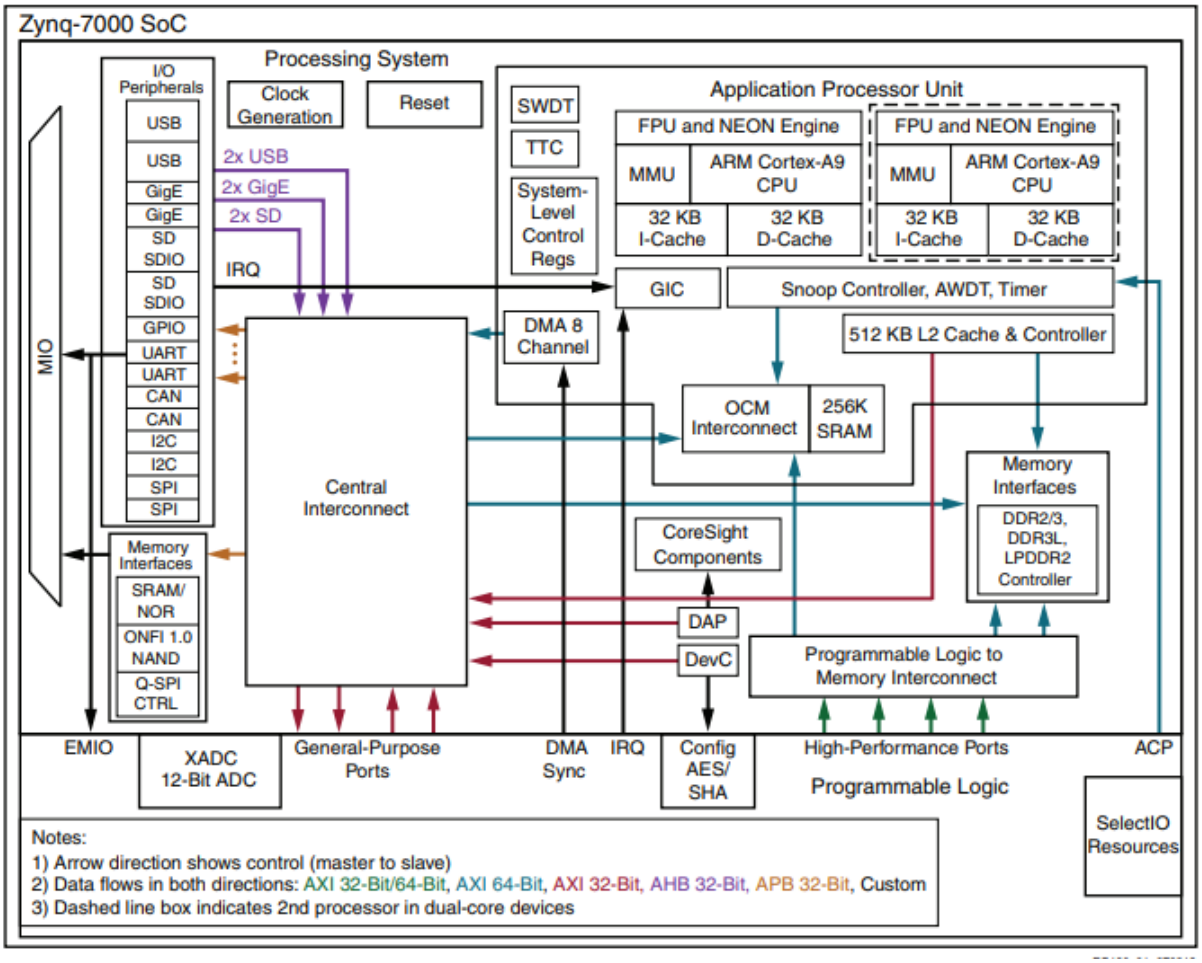
Memorijska sučelja sadrži dinamički upravljač memorije i module statičkog memorijskog sučelja koji podržavaju nekolika tipova memorija.

Ulazno izlazne periferije sadrže:

- Dva Ethernet priključka
- Dva USB 2.0 priključka
- Dva CAN 2.0 priključka
- Dva I2C sučelja
- Dva UART-a itd.

Preklopnik služi za komunikaciju između APU, Memorijskog sučelja, Ulazno izlazne periferije i programabilne logike. [6]

Programabilna logika sastoji se od Xilinx 7 serije Artix FPGA. Sadrži 28k logičkih blokova, 240k Blok RAM-a, 80 DSP blokova itd. Unutar programibilne logike CLB blokovi sadrže 8 LUT elementa i 16 bistabila. Memorijski LUT-ovi konfigurirani su kao 64x1, 32x2 ili kao pomični registri.




***Sl. 3.4*** *Arhitektura Zynq-7000[6]*

## 3.3 ZYBO bord

Zybo ploča je razvojna platforma tvrtke Digilent koja je zasnivana na  Xilinx Zynq-Z-7010 Soc-u (eng. System on Chip). Sastoji se od dual-core ARM Cortex-A9 procesora i Xilinx 7 serije FPGA logike. Također ploča sadrži periferiju pogodnu za razvoj embeded softwera i dizajniranje digitalne logike. [7]

Neke od karakteristika:

- 650Mhz dual-core Cortex-A9 procesor
- DDR3 memorijski kontrolor sa 8 DMA kanala
- Protokoli visokog širine pojasa :1G Ethernet, USB 2.0, SDIO
- Protokoli niske širine pojasa: SPI, UART, CAN, I2C
- Programibilna logika Artix-7 FPGA

- ZYNQ XC7Z010-1CLG400C
- Audio kodek SSM2603

## 3.4 VHDL

VHDL (eng. Very high speed integrated circuit hardware description language) je uz Verilog i System Verilog jedan od najpopularnijih jezika za opis fizičke arhitekture (eng. Hardware description language, HDL). Koristi se za opis hardvera prilikom modeliranja digitalnih sustava za potrebe dokumentiranja, simuliranja i sinteze. Također, koristi se za konfiguraciju programibilnih logičkih uređaja (eng. Programmable logic device, PLD) kao što su FPGA. Razvila ga je 1981. Američka vojska i jezik dijeli mnoge koncepte i karakteristike programskog jezika Ada, jedan od kojih je paralelno procesiranje zadataka. Jezik je strogo tipiziran, što znači kako je potrebno pratiti stroga pravila pisanja dizajna.

Ključna prednost HDL-a je što je moguće izvršiti modeliranje, verifikacija i simulacija dizajna prije implementacije na hardver.

Dizajn pisan VHDL-om se sastoji od dva dijela, entitet (eng. Entity) i arhitektura (eng. Architecture). Entity definira ulaze i izlaze iz dizajna, Architecture dio opisuje unutarnje operacije dizajna te definira funkcionalnost dizajna.

Postoje tri načina opisivanja sklopovlja VHDL-om:

Strukturalni VHDL (eng. Structural VHDL) je način opisivanja sklopovlja gdje se provodi međusobno povezivanje prethodno definiranih modula. Povezivanje se vrši sabirnicama koje se nazivaju signali. U praksi se koristi na vrhu hijerarhije dizajna, pri povezivanju više modula u jedan funkcionalan dizajn.

RTL (eng. Register transfer level) kodiranje se odnosi na opisivanje toka signala među registrima, te logičkih operacija koje se provode nad tim signalima. RTL kodiranje je viši nivo apstrakcije u usporedbi strukturalnim VHDL-om, jer se logičke funkcije opisuju relacijom umjesto povezivanjem logičkih vrata.

Funkcijsko kodiranje (eng. Behavioral coding) je najviša razina apstrakcije u VHDL-u. Opisivanje dizajna se ne provodi spajanjem logičkih operatora, već se piše algoritam koji opisuje kako se ulazi preslikavaju u izlaze. Opis dizajna ne opisuje građu sklopa već funkcionalnost i moguće je koristiti naredbe za proceduralno programiranje kao što u petlje, if i case uvjeti itd. Arhitektura dizajna pri korištenju funkcijskog kodiranja sastoji se od jednog ili više procesa.

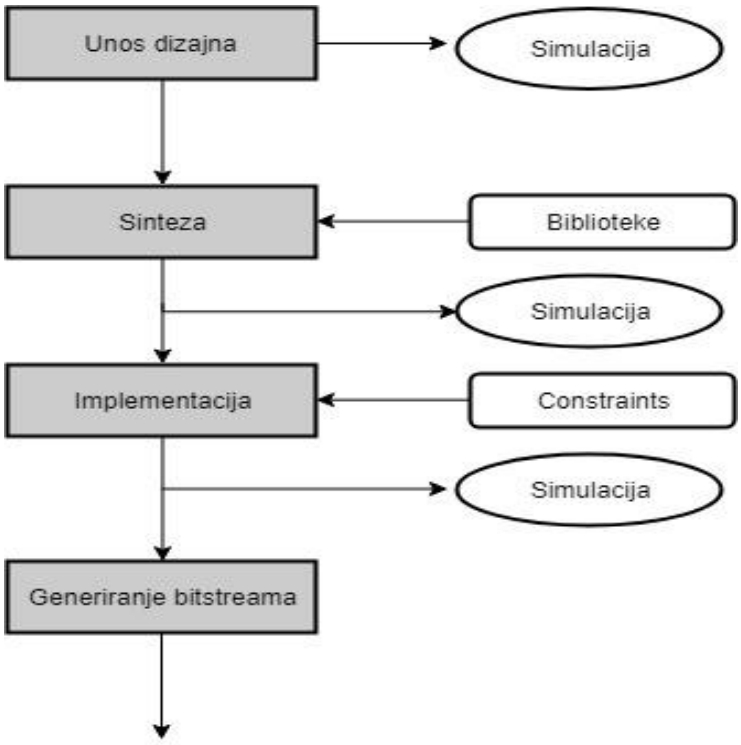
Proces omogućuje proceduralni opis sklopa i korištenje naredbi za kontrolu toka. Prilikom pokretanja dizajna proces se provede jednom te se zaustavi na kraju procesa. Ponovno izvršavanje procesa uvjetovano je listom osjetljivosti. Lista osjetljivosti definira ulaze ili signale pri čijim promjenama se pokreče proces. Unutar VHDL dizajna moguće je imati nekoliko procesa, no to su procesi dva tipa: sekvencijalni i kombinacijski.

Sekvencijalni proces na listi osjetljivosti najčešće sadrži samo takt sustava tako da se proces izvodi unutar jednakih vremenskih intervala, koji ovise o periodi takta. Kombinacijski proces može sadržavati nekolicinu signala na listi osjetljivosti i izvodi se trenutno prilikom promjene stanja signala na listi osjetljivosti.

Zbog popularnosti jezika 1987. IEEE je prihvatio VHDL kao standard. [8][9]

## 3.5 Dizajniranje s FPGA

Zbog kompleksnosti dizajniranja s FPGA, koristi se strukturni način dizajniranja. Gdje FPGA sastoji više međusobno povezanih blokova (entiteta). Entiteti mogu biti razvijeni od strane proizvođača FPGA ili razvijeni od strane korisnika korištenjem HDL-a. No proces dizajniranja s FPGA ne završava na unosu dizajna, slika 3.5. prikazuje proces koji je potrebno izvršiti kako bi FPGA radio u skladu s traženim zahtjevima.




***Sl. 3.5*** *Proces stvaranja dizajna za FPGA*

Prema slici 3.4., nakon unosa dizajna potrebno je izvršiti simulaciju kako bi testirali funkcionalnost dizajna. Radi provjere rada provodi se vrši se simulacija nakon sinteze i implementacije.

Sinteza je postupak u kojem se dizajn opisan HDL-om pretvara u mrežu međusobno povezanih logičkih vrata čija funkcionalnost odgovara zahtjevima definiranim HDL-om. Nakon sinteze potrebno je provesti implementaciju. Implementacija je postupak gdje se na temelju veza logičkih vrata dobivene sintezom, definira mreža fizički spojenih i konfiguriranih CLB blokova unutar FPGA, čime je ujedno definirana duljina propagacijskih kašnjenja. Tijekom implementacije definiraju se na koje fizičke ulaze i izlaze se spajaju ulazi i izlazi entiteta. Nakon što je dovršen postupak implementacije generira se bitstream datoteka kojim se programira FPGA.[5]

## 3.6 ModelSim

ModelSim je okruženje za verifikaciju i simulaciju dizajna HDL jezika kao što su VHDL, Verilog i SystemVerilog. Grafičko sučelje prikazuje sve ulaze, izlaze, signale i varijable unutar dizajna prikazane u ovisnosti o vremenu. Zbog toga se koristi za provjeru funkcionalnosti dizajna prije implementacije.

Potrebno je izvršiti detaljno testiranje dizajna kako bi uklonili sve pogreške prije procesa unosa dizajna u FPGA. Testiranje se izvodi tako što se definiraju stanja ulaza sustava kojeg se testira, te se promatra dali su izlazi sustava u skladu s očekivanjima. Ako se ne dobiva željeni izlaz u određenom slučaju, grafičko sučelje olakšava proces pronalaska pogreške. Grafičko sučelje prikazuje sve signale unutar dizajna te prikazuje detaljno unutarnje procese dizajna.

Također, bitan faktor zbog kojeg se provodi testiranje u ModelSimu je vrijeme potrebno za izvođenje sinteze i implementacije. Vrijeme sinteze i implementacije ovisi o kompleksnosti dizajna te performansama računala, no uvijek traje duže od vremena potrebnog za simulaciju. Upravo zbog tog razloga, kako bi se ubrzao proces izrade dizajna, provodi se testiranje simulacijom u ModelSimu umjesto provedbe sinteze i implementacije. Tek kada se simulacijom dobivaju zadovoljavajući rezultati, provodi se sinteza i implementacija.

Sve korisničke naredbe mogu biti unošene grafički ili uz pomoć skripte što olakšava proces testiranja. [10]

## 3.7 Vivado Design Suite

Vivado Desing Suite je razvojno okruženje za dizajn, integraciju i implementaciju HDL dizajna razvijen od Xilinx-a. Koristi se prilikom implementacije dizajna na Xilinx-ovom sklopovlju Xilinx UltraScale, Zync UltraScale+ MPSoC i Zync-7000SoC.

Korištenje Vivado Design Suite-a olakšava i ubrzava proces sinteze i implementacije dizajna, te nudi mogućnost analize performansa dizajna tijekom svake faze dizajniranja. Time je omogućeno rano otkrivanje pogrešaka i optimizacija sustava kako bi se smanjio broj iteracija tijekom dizajniranja. Podržava simulaciju prije sinteze, nakon sinteze te nakon implementacije gdje su uračunata propagacijska kašnjenja signala.

Vivado Design Suite sadrži brojne ugrađene alate kao što su projekt navigator, Xilinx tehnologija za sintezu, CORE generator, uređivanje vremenskih ograničenja, ISE simulator, ChipScope analizator, Xilinx analizator snage itd.

Svi alati mogu se koristiti uz pomoć Tlc (eng. Tool Command Language) skripte ili korištenjem grafičkog sučelja. Tcl skripta je dinamički programski jezik visoke razine korišten za opću primjenu. Pri korištenju s FPGA dizajnima moguće je uz pomoć Tlc skripte generirati i pokretati cijele dizajne, dodavati dizajne, podešavati svojstva dizajna itd.

Vizualizacija dizajna je olakšana jer svaki entitet može biti prikazan kao blok s definiranim ulazima i izlazima. Blokovi se grafički spajaju vezama gdje Vivado Design Suit generira kod kojim definira veze.

Vivado Design Suite također sadrži katalog IP-a (eng. Intelectual property) gdje je moguće koristiti entitete razvijene od strane Xilinxa. Unutar diplomskog rada entiteti korišteni iz IP kataloga su I2C kontoler te BRAM kontroler. [11]

## 3.8 Xilinx SDK

Xilinx SDK (eng Software development kit) je okolina koja se koristi za kreiranje aplikacija za Xilinxov procesore Zynq UltraScale+ MPSoC, Zynq-7000 te „soft core“ MicroBlaze procesor. Temeljen je na Eclipse 4.5.0 i CDT 8.8.0 i najčešće je uključen kao dio Vivado Design Suita. Sadrži potpunu integriranu okolinu za dizajn (eng. Integrated Design Environmen, IDE) koja vrši direktnu interakciju sa Vivado hardverskim okruženjem.

Podržava softversko te softver-hardversko traženje pogrešaka, uključujući više procesorske sustave. Sadrži sve standardne mogućnosti traženja pogrešaka kao što je postavljanje

„breakpointa“, analiza izvršenja programa korak po korak, uvid u varijable programa i stog te memoriju sustava. Postoji mogućnost istovremenog traženja pogrešaka programa koji se izvršavaju na različitim procesorima (višeprocesorski sustav).

Xilinx SDK sadrži pred definirane parametre sklopovlja definiranog sa Vivado Design Suit-om kako bi olakšao integraciju programskog i sklopovskog dijela sustava. Primjer preddefiniranih konfiguracija su konfiguracija memorijskih mapa, postavka registara periferije, putanje potrebnih biblioteka, postavka kompajliranja, postavke JTAG i flash memorije itd.

Detekcija „bottle neck“ u kodu je jednostavnija jer postoji alat za detekciju. Također alati za prikupljanje podataka performansa sustava te njihova vizualizacija omogućuje lakšu optimizaciju sustava. Neki od prikupljenih podataka su: postotak korištenosti procesora, koliko instrukcija se izvršava po ciklusu, latencija te širina spektra čitanja i pisanja između programskog i sklopovskog dijela sustava.[12]

## 3.9 Audio kodek SSM2603

Digitalni filtri vrše operacije nad diskretnim signalima te ulazni i izlazni signal treba biti diskretan. Kako je temom rada definirano, audio signal koji se filtrira je analogni signal i potrebno je provesti analogno digitalnu pretvorbu. Audio kodek obavlja pretvorbu ulaznog signala u diskretan signal pogodan za filtriranje. Također pretvara diskretan signal na izlazu filtra u audio signal pogodan za reprodukciju.

SSM2603 je audio kodek integriran na Zybo ploči i sadrži mikrofon ulaz, line in ulaz te izlaz. Karakteriziraju je 24-bitni analogno-digitalni te digitalno-analogni pretvornici koji podržavaju dvokanalni način rada. SSM2603 podržava master ili slave načinu rada i te frekvencije uzorkovanja od 8kHz do 96kHz. Konfiguracija audio kodeka se izvodi I2C protokolom tako što se podaci upisuju u različite 8 bitne registre prikazane tablicom 3.1. Upisom u registre prikazane tablicom može se podesiti frekvencija uzorkovanja, odabir ulaznih i izlaznih portova, konfiguriranje automatske kontrole amplitude audio signala itd. [13]

***Tab. 3.1*** *Popis registra audio kodeka SSM2603*

| Registar | Adresa | Naziv |
|---|---|---|
| R0 | 0x00 | ADC ulazni volumen lijevog kanala |
| R1 | 0x01 | ADC ulazni volumen desng kanala |
| R2 | 0x02 | DAC volumen lijevog kanala |
| R3 | 0x03 | DAC volumen desnog kanala |
| R4 | 0x04 | Analogna audio putanja |
| R5 | 0x05 | Digitalna audio putanja |
| R6 | 0x06 | Upravljanje napajanjem |
| R7 | 0x07 | Digitalni audio I/F |
| R8 | 0x08 | Frekvencija uzorkovanja |
| R9 | 0x09 | Aktivno |
| R15 | 0x0F | Softverski reset |
| R16 | 0x10 | ALC kontrola 1 |
| R17 | 0x11 | ALC kontrola 2 |
| R18 | 0x12 | Kontrola šuma |

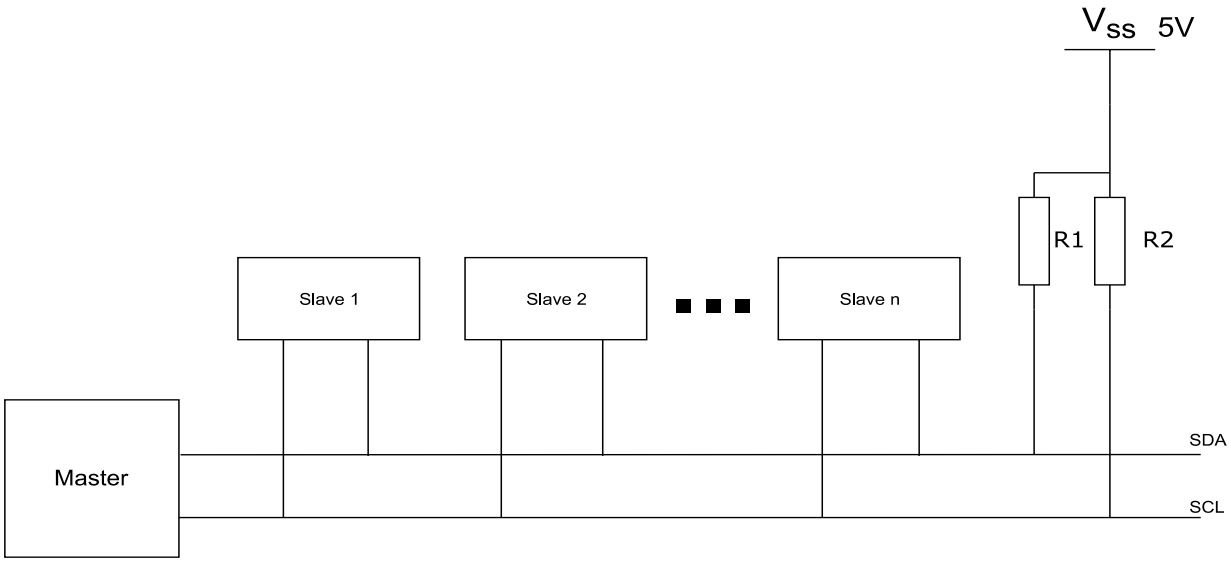
Audio kodek podržava četiri različita komunikacijska protokola za prijenos podataka: način rada s lijevim poravnavanjem, način rada s desnim poravnavanjem, I2S način rada i DSP način rada. Zbog dostupnosti I2S entiteta za FPGA, korišten je I2S način rada.

## 3.10 I2C protokol

Kako bi se audio kodek konfiguriran unutar Zybo ploče mogao koristiti, potrebno je izvršiti konfiguraciju I2C protokolom.

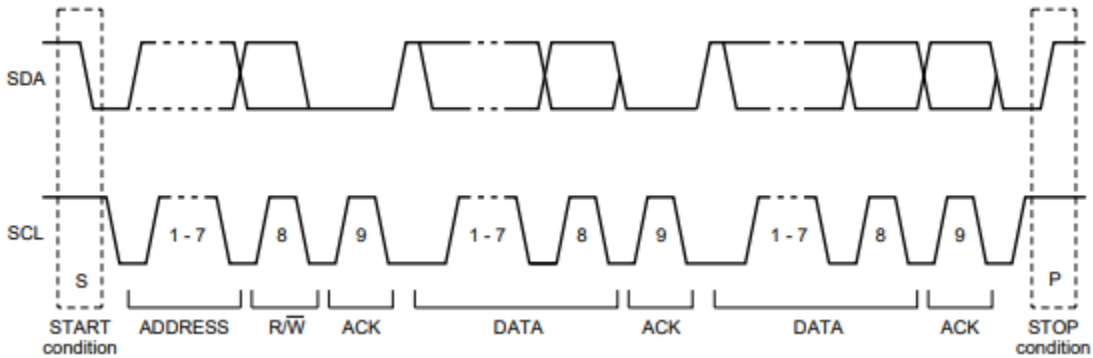
I2C je standard korišten za međusobno povezivanje periferije s procesorima ili mikroupravljačima na malim udaljenostima. Nastao je iz potrebe za hardverskom efikasnosti jer omogućuje povezivanje više perifernih uređaja koristeći dvije jednobitne sabirnice, SDA i SCL. SDA (serijska linija podataka) služi za prijenos podataka dok SCL (serijska linija takta) služi za sinkronizaciju. Svaki uređaj spojen na I2C posjeduju specifičnu 7 bitnu adresu i može raditi kao prijamnik ili predajnik. [14]

Podjela uređaja vrši se na master i slave uređaje. Master je uređaj koji započinje komunikaciju te šalje podatke i takt drugim uređajima. U tom trenutku svi ostali uređaji se smatraju slave-ovima kao što prikazuje slika 3.6. Na SDA i SCL su dvosmjerne sabirnice, koje su spojene kao otvoreni kolektor koje sadrže „pull-up“ otpornike spojene na 5V. Uređaji spojeni na I2C sabirnicu postavljaju stanje sabirnica u stanje '0' ili visoke impedancije 'Z'. Zbog „pull-up“ otpornika, u stanju 'Z' na sabirnici je stanje '1'.




***Sl. 3.6*** *Shema I2C protokola koja se sastoji od master uređaja koji ostvaruje komunikaciju sa slave-ovima, te dva „pull-up“ otpornika*

Signali koji se šalju linijama SDA i SCL prikazani su prema slici 3.7. Podaci se šalju paketima od 8 bitova te frekvencijom od 100kbit/s. Prilikom početka komunikacije master šalje 7 bitni podatak o adresi i R/W bit za upis ili čitanje podatka s uređaja kojim se želi povezati. Kada uređaj s navedenom adresom prihvati podatak, slave šalje ACK kao odgovor da je ostvarena komunikacija i počinje daljnji prijenos podataka. Veza se prekida tako što master šalje stop uvjet i nakon toga moguće je uspostaviti novu vezu. [14]




***Sl. 3.7*** *SDA i SCL linije prilikom komunikacije [14]*

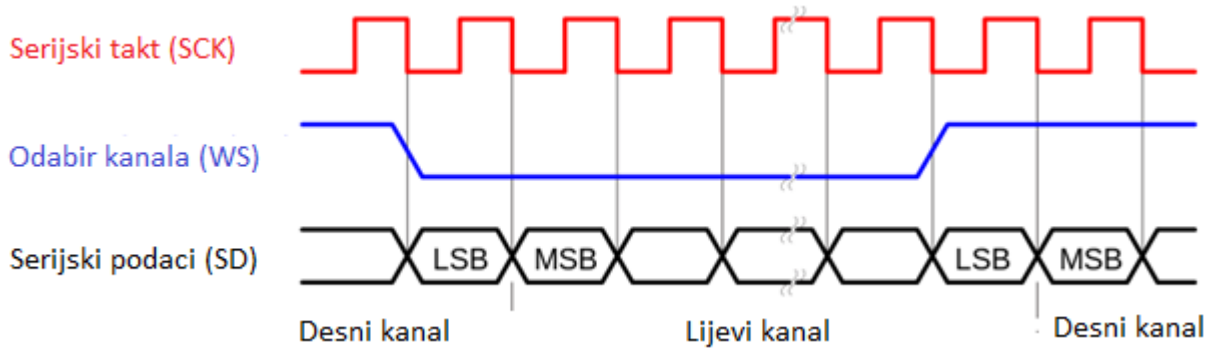
U diplomskom radu Zync je master, koji šalje podatke slave-u odnosno audio kodeku. Komunikacija se ostvaruje tako što se šalje paket koji sadrži adresu audio kodeka, zatim se šalje

paket koji definira registar prema tablici 3.1. te naposljetku se šalje paket koji sadrži konfiguracijske bitove koji se upisuju u registar.

## 3.11 I2S protokol

Audio kodek konfiguriran je da radi u I2S načinu rada, što znači kako se svi ulazni audio podaci prenose I2S protokolom. Svi podaci koje se žele poslati na izlaz audio kodeka također moraju koristiti I2S protokol.

I2S je standard serijske komunikacije za spajanje digitalnih audio uređaja. Koristi se za prijenos PCM audio podataka unutar nekog elektronskog uređaja. Kako bi se smanjio broj sabirnica potreban za prijenos audio podataka I2S protokol sastoji se od tri sabirnice: word takt, serijski takt te podaci. Word takt nosi podatak o kanalu gdje je '0' lijevi kanal a '1' desni kanal. Serijski takt, koji služi za sinkronizaciju tako što definira takt prijenosa podataka te posljednja linija data, gdje se vrši prijenos podataka. Podaci koji se šalju sadrže predznak, korištenjem drugog komplementa. Prilikom slanja podatka MSB (eng. Most significant bit, najvažniji bit) se šalje prvi. [15]




***Sl. 3.8*** *I2S protokol prilikom komunikacije*

Navedeni podaci spajaju se na entitet za I2S koji podatke pretvara u pakete od 24 bita koji se šalju dalje na filtriranje. Nakon postupka filtriranja, 24 bitni audio podaci šalju se na I2S entitet, koji pretvara paket u I2S format te šalje na audio kodek kako bi bilo moguće reproducirati zvuk.

## 3.12 AMBA AXI

AMBA AXI u (engl. Advanced Microcontroller Bus Architecture Advanced Extensible Interface) je protokol unutar koji omogućuje lako povezivanje procesora i entiteta FPGA unutar

SoC-a. AMBA AXI protokol je pogodan za prijenos podataka kada je potrebna visoka pojasna širina i niska latencija, te podržava rad na visokim frekvencijama. [11]

AMBA AXI ostvaruje komunikaciju uz pomoć pet kanala:

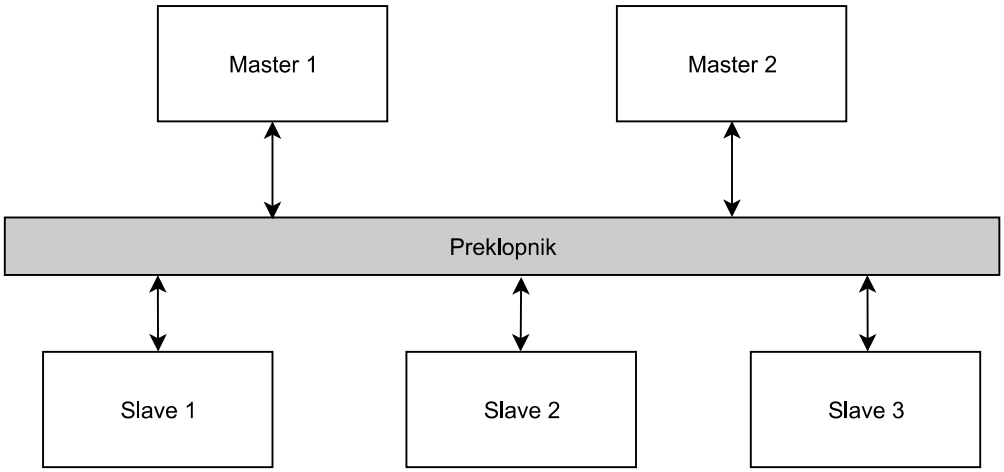
- Kanal za čitanje adrese
- Kanal za upis adrese
- Kanal za čitanje podataka
- Kanal za upis podataka
- Kanal za provjeru upisa [16]

Upis podataka AXI protokolom vrši se tako što master upisuje podatke na slave-a, dok se čitanje podataka vrši od slave-a prema masteru. Svaki od kanala sadrže unutar sebe signale Valid te Ready koji se koriste za dvosmjerni „handshake“. Predajnik podataka korištenjem signala Valid prikazuje kada je adresa, podatak ili kontrolna informacija dostupna na kanalu. Prijamnik podataka koristi Ready signal kako bi prikazao da je spreman primiti podatke. Kanali za čitanje i pisanje također sadrže signal Last, kako bi signalizirali kada se odvija transakcija zadnjeg podatka.

Kanali za čitanje i pisanje svaki sadrže kanal koji definira memorijske adrese. Kanali za upis i čitanje podataka nemaju fiksnu širinu kanala, te mogu biti 8, 16, 32, 64, 128, 256, 512 ili 1024 bita.

Svi slave-ovi koriste kanal za provjeru upisa za slanje odgovora prilikom upisa. Svaki postupak upisa zahtjeva potvrdu završetka upisa kanalom za provjeru upisa.

Tipičan sustav koji koristi AXI protokol sadrži nekoliko međusobno povezanih master i slave-ova. Takvo povezivanje se ostvaruje uz pomoć preklopnika (eng. Interconnect) kao što se vidi na slici 3.9.




***Sl. 3.9*** *AXI prijeklopnik koji se koristi za spajanje uređaja AXI protokolom*

Za komunikaciju Zynq procesora sa FPGA entitetima korišten je AMBA AXI 4 protokol, koji je 4. Generacija AMBA AXI protokola. Vrste AXI 4 protokola:

- AXI4 - pruža najbolje performanse od AXI protokola i koristi se za povezivanje memorijskih mapa.
- AXI4 Lite – Koristi se za prijenos jednog podatka gdje je podatak adresiran.
- AXI Stream – koristi se za prijenos većeg broja podataka gdje se definira početna adresa te količina podataka za prijenos.

## 3.13 BRAM

BRAM (Block RAM) je memorijski element integriran u FPGA. Koristi se uz pomoć Xilinx-ovog IP-a Block Memory Generator. Navedeni IP koristi se kao veza između dizajna te integriranih BRAM-a. Sadrži dva funkcionalna kanala koji se mogu koristiti za upis i ispis podataka s BRAM-a.

BRAM memorija može raditi kao:

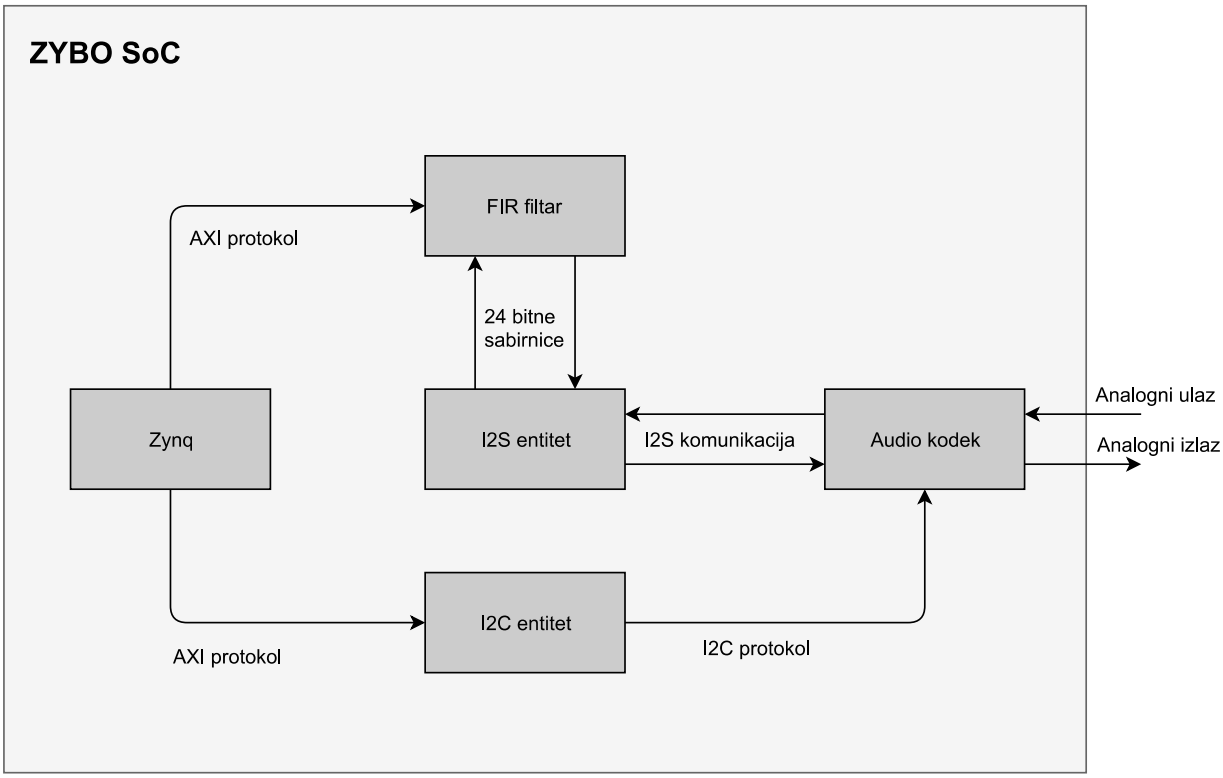
- Single Port RAM – koristi jedan kanal. Ne podržava istovremen upis i ispis podataka.
- Simple Dual Port RAM – moguće koristiti istovremeno dva kanala. Jedan kanal se koristi za upis podataka, drugi za ispis.
- True Dual Port Ram - moguće koristiti istovremeno dva kanala. Oba kanala mogu se koristiti za upis ili ispis podataka.
- Single Port ROM –koristi jedan kanal za čitanje podataka.
- Dual Port ROM – koristi dva kanala za čitanje podataka.

Širina podataka može biti od 1 do 4096 podataka, dok dubina može biti od 2 do 128k. Unutar diplomskog rada upis i ispis podataka vrši se AXI4 Lite protokolom.

Svaka memorijska adresa sadrži 32 bita podatka, latencija zapisa podatka je 2 takta, dok je čitanja 1 takt.[17]

# 4. IMPLEMENTACIJA FIR FILTRA U FPGA

Implementacija FIR filtra izvedena je unutar Zybo razvojnog okruženja gdje je idejni dizajn projekta prikazan na slici 4.1. Ulazni analogni signal koji je potrebno filtrirati dovodi se na ulaz audio kodeka, koji je prethodno konfiguriran za željeni način rada. Konfiguracija audio kodeka provodi se uz pomoć Zynq procesora i I2C entiteta. Zynq definira konfiguracijske podatke koji se šalju AXI protokolom na I2C entitet. I2C entitet navedene podatke šalje I2C sabirnicom na audio kodek te provodi konfiguraciju. Za potrebe diplomskog rada audio kodek je konfiguriran na rad sa 24 bitnom rezolucijom uzorkovanja, frekvencijom uzorkovanja od 48 kHz. Audio kodek ulazni analogni signal šalje I2S komunikacijom do I2S entiteta, koji audio podatke sprema u 24 bitni registar i šalje 24 bitnom sabirnicom do FIR filtra, koji vrši proces filtriranja. Proces filtriranja izvodi se na temelju koeficijenta filtra koji se unose u Zynq procesor te šalju AXI protokolom na FIR filtar. Nakon što se provede proces filtriranja 24 bitnog paketa audio podataka, paket se šalje na I2S entitet. I2S entitet šalje podatke I2S komunikacijom na audio kodek, gdje audio kodek provodi digitalno analognu pretvorbu signala. Filtrirani signal tada je moguće reproducirati na izlazu audio kodeka.

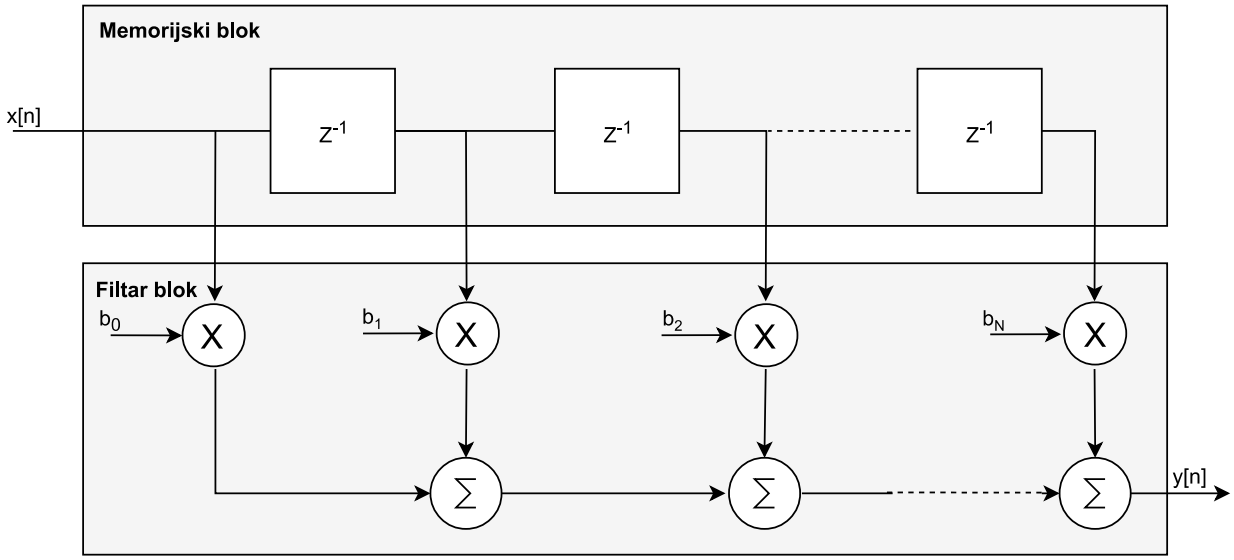



***Sl. 4.1*** *Idejna shema sustava a filtriranje audio signala*

Od blokova prikazanih slikom 4.1. Zynq i audio kodek implementirani su hardverski na Zybo razvojnom okruženju, dok su I2S i I2C entiteti dizajnirani od strane Xilinx-a i Digilenta. Potrebno je implementirati FIR filtar blok kako bi na temelju koeficijenata filtra dobivenih od Zynq procesora vršio proces filtriranja signala.

## 4.1 Idejno rješenje FIR filtra

U poglavlju 2. navedeno je kako je izlazni signal FIR filtra jednak zbroju umnožaka prethodnih stanja ulaza filtra te koeficijenta filtra. Zbog toga potrebno je implementirati blokove za zbrajanje, množenje te memorijske blokove koji spremaju prošla stanja audio signala. U poglavlju 2. Slika 2.7. prikazuje blokovsku shemu FIR filtra. Navedena shema koristi se za dizajniranje entiteta FIR filtra. Slika 4.1. prikazuje kako je blok shema FIR filtra iz poglavlja 2 podijeljena u dva bloka, odnosno dva entiteta koja će biti dizajnirana.




***Sl. 4.2*** *Blok shema FIR filtra unutar FPGA*

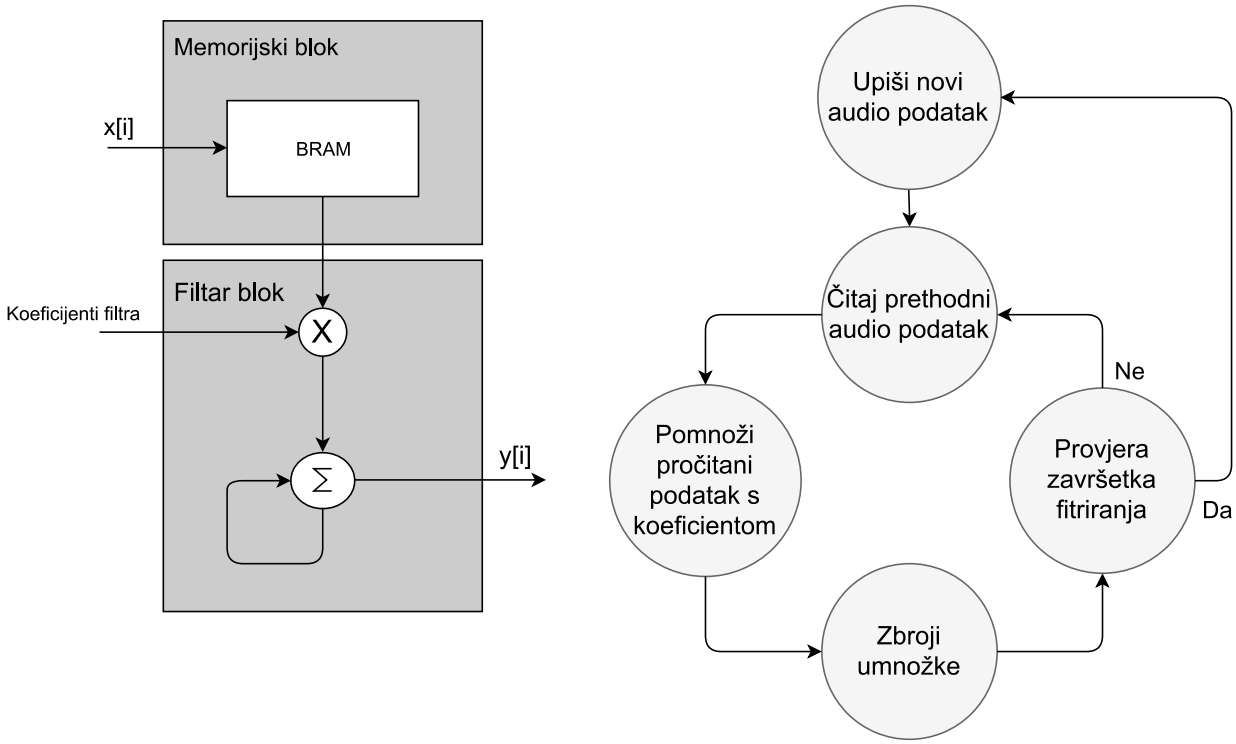
Prema slici 4.2. vidi se kako Memorijski blok izvršava funkciju spremanja podataka prošlih stanja audio ulaza. Filtar blok vrši funkciju množenja prošlih stanja audio ulaza sa koeficijentima filtra te zbrajanje svih umnožaka.

FIR filtar implementiran kao na slici 4.2. završio bi proces filtriranja s malom latencijom zbog toga što se sve aritmetičke funkcije potrebne za filtriranje izvršavaju paralelno. Negativna strana dizajna je što povećanjem reda filtra povećava se i potreban broj DSP blokova. Provedena je implementacija navedenog dizajna i ukupna latencija procesa filtriranja je 5 taktova. Zbog

ograničenja broja DSP elemenata unutar Zybo SoC-a, dolazi do ograničenja maksimalnog reda filtra, gdje je implementacija FIR filtra 20 reda koristila 50% DSP elemenata.

Kako bi se smanjio broj potrebnih DSP blokova implementiran je dizajn koji koristi manje DSP blokova, no potrebna je veća latencija pri procesu filtriranja. Dizajn je zamišljen na način da sva se sva zbrajanja i množenja ne provode paralelno, već da se potrebna aritmetika na podacima provodi sekvencijalno. Ovisno o redu filtra toliko puta će se provoditi postupak zbrajanja i množenja. Tim načinom rada ukupna latencija potrebna za provedbu filtriranja jednaka je umnošku reda filtra  i latencije potrebne za jedno množene i zbrajanje.

Osim uštede resursa, razlog za korištenjem navedenog dizajna je taj što je frekvencija uzorkovovanja višestruko manja od frekvencije rada FPGA. Najveća moguća frekvencija uzorkovanja audio kodeka je 96 kHz, dok je frekvencija rada FPGA 100 MHz . Kako FPGA dobiva novi uzorak frekvencijom 96 kHz, FPGA ima dovoljno perioda takta da se provede proces filtriranja većom latencijom. Slika 4.3. prikazuje blok shemu dizajna koji sadrži jedan element za zbrajanje i množenje, čime je smanjen potreban broj DSP blokova u usporedbi s dizajnom sa slike 4.2.



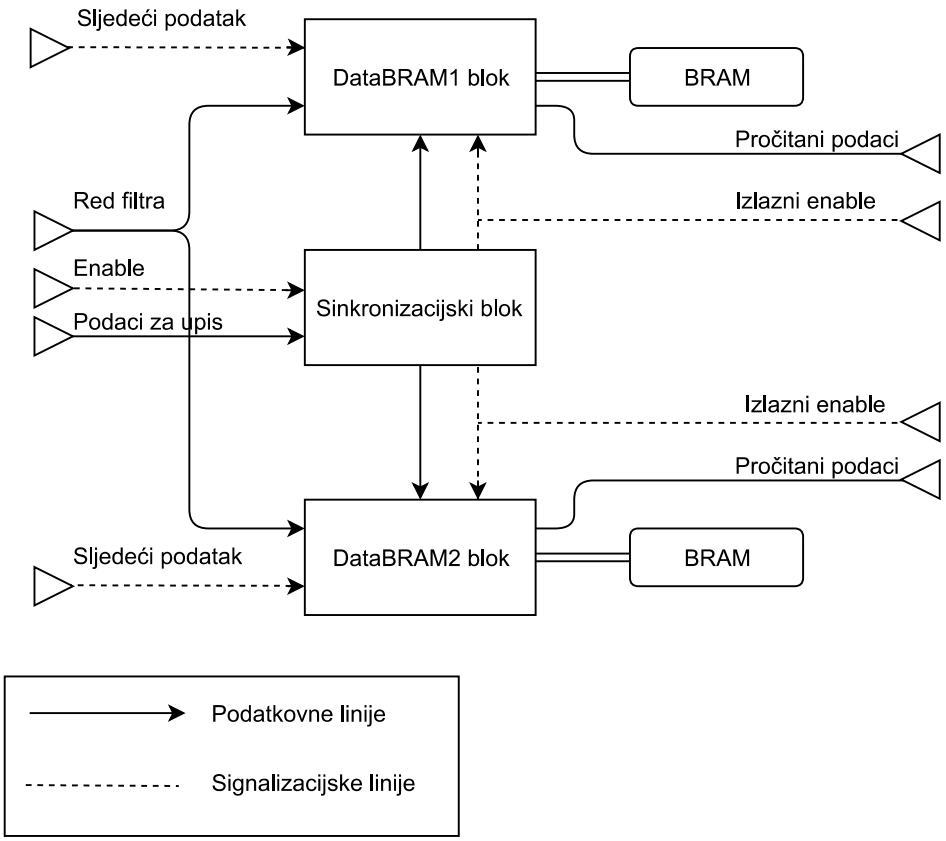

***Sl. 4.3*** *Izgled konačnog idejnog rješenja za implementaciju na FPGA i dijagram toka procesa filtriranja*

Dijagram toka prikazan je slikom 4.3. i počinje tako što se vrši upis audio podatka s ulaza u memoriju. Nakon toga započinje proces filtriranja, gdje se čitaju prethodni audio podaci iz memorije i vrši se množenje i zbrajanje. Kružni proces unutar slike 4.3. izvršit će se onoliko puta koliki je red filtra. Kako bi proces filtriranja bio uspješan, isti se mora završiti prije nego što na ulaz filtra dospije novi audio podatak. Time se dobiva ograničenje maksimalnog reda filtra. Prethodno je ograničenje bilo zbog resursa FPGA, no sada je ograničenje zbog trajanja procesa filtriranja.

## 4.2 Memorijski blok

Memorijski blok je entitet zadužen za upis i ispis audio podataka u memoriju. Podatke je moguće spremati u bistabile CLB isječaka, no time se povećava postotak iskorištenosti resursa FPGA. Kako bi se smanjila ukupna iskorištenost CLB resursa FPGA podaci se upisuju u BRAM-e.

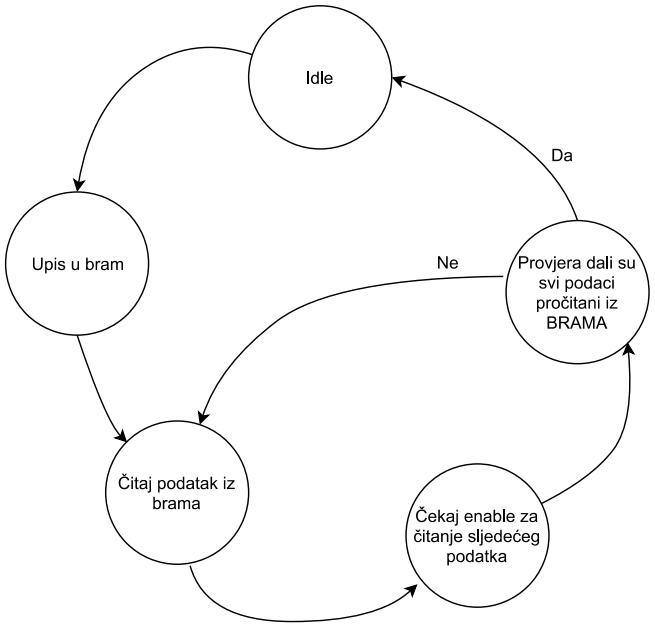
Blokovska shema (slika 4.4) prikazuje Memorijski blok koji je zadužen za upis i ispis audio podataka u BRAM. Sastoji se od Sinkronizacijskog bloka, DataBRAM bloka te BRAM-a. Sinkronizacijski blok je entitet zadužen za sinkronizaciju ulaznih audio podataka sa audio kodeka. Sadrži ulaz za enable, te ulaz za 24 bitne audio uzorke. U trenutku detekcije rastućeg ili padajućeg brida na enable ulazu, trenutni audio podaci se spremaju u registar i šalju u DataBRAM blok. Audio podaci spremaju se u dva različita DataBRAM bloka, u jedan BRAM se upisuju audio podaci lijevog kanala, dok u drugi od desnog kanala, ovisno o logičkom stanju enable signala.




***Sl. 4.4*** *Blok shema entiteta Memorijski blok*

Upis podataka u BRAM se izvodi po principu cirkularnog buffera. BRAM se popunjava brojem uzoraka koji je definiran redom filtra. Nakon što se u memorijske lokacije upiše broj podataka jednak redu filtra, svaki novi podatak se upisuje u lokaciju u BRAMU gdje se nalazi najstariji podatak.

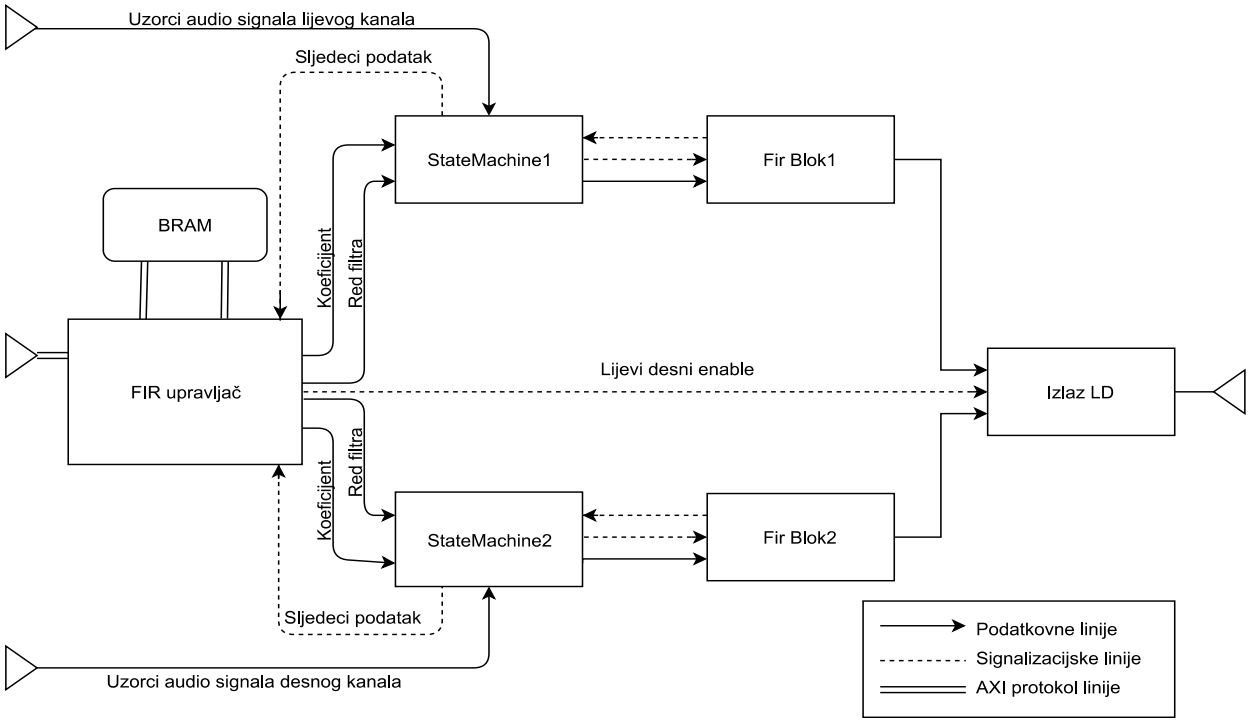
Pri procesu filtriranja Memorijski blok obavlja funkcionalnost prema slici 4.5. Pri detekciji novog audio podatka, isti se upisuje u BRAM. Nakon toga počinje proces čitanja svih audio podataka unutar BRAM-a te prosljeđivanje istih do bloka kojim se vrši filtriranje. Po završetku čitanja svih podataka iz BRAM-a, entitet čeka dolazak novog audio podatka za upis.




***Sl. 4.5*** *Dijagram toka Memorijskog bloka*

## 4.3 Filtar blok

Filtar blok je entitet koji vrši proces filtriranja. Izvršava proces spremanja koeficijenta filtra, koje prima od Zynq-a, u BRAM. Te tijekom procesa filtriranja čita potreban koeficijent iz BRAM-a. Također provodi operacije množenja i zbrajanja potrebnih za proces filtriranja.




***Sl. 4.6*** *Blok shema entiteta Filtar blok*

FirBlock se sastoji od sljedećih entiteta:

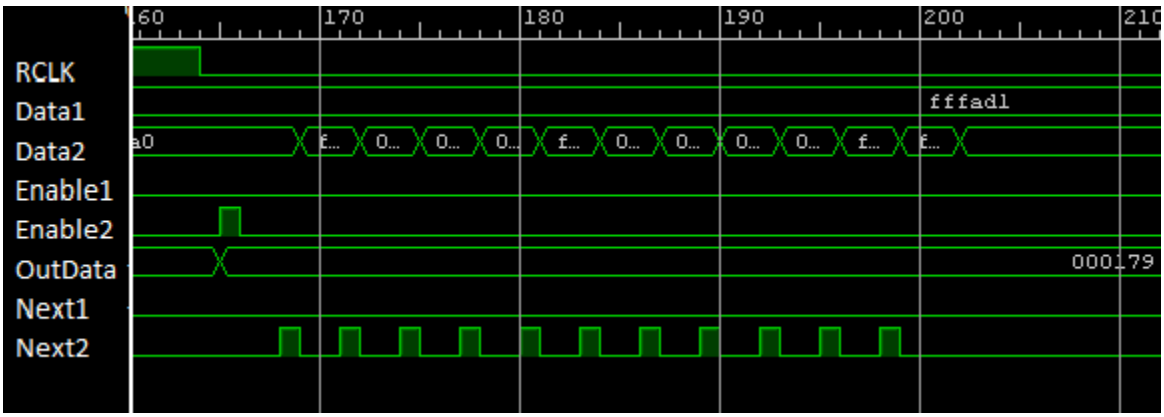
FIR upravljač entitet služi za upis i ispis koeficijenta filtra u BRAM. Upis podataka u BRAM izvodi se uz pomoć procesora, gdje se podaci šalju AXI lite sabirnicom. Ispis podataka iz BRAM-a provodi se kada State Machine entitet pošalje impuls signalom Sljedeći podatak.

State Machine entitet je konačni automat, odnosno glavni kontrolni blok koji započinje proces filtriranja. Kada State Machine entitet detektira impuls Enable ulaza, isti šalje potrebne impulse drugim entitetima kojima zahtjeva uzorke audio signala i koeficijente filtra. State Machine entitet navedene podatke prosljeđuje do FIR bloka.

Fir Blok je entitet unutar kojeg se vrše matematičke operacije potrebne za FIR filtriranje. Unutar sebe sadrži entitete koji provode množenje uzorka i koeficijenata, te zbrajanje svih umnoška.

Izlaz LD entitet služi kao demuliplexer, gdje u ovisnosti signalu Lijevi desni enable , šalje signal iz entiteta FIR blok1 ili FIR blok2 na izlaz. Drugim riječima šalje audio podatke lijevog ili desnog kanala na izlaz.

Prema slici 4.7 prikazani su signali unutar FIR block entiteta u slučaju filtra 11 reda. Nakon detektiranja impulsa signala Enable2, state machine entitet šalje impulse signalom Next2 (na slici 4.5. definiran kao signalizacijska linija Sljedeci podatak) kojima na ulaz dobiva podatke na ulaz Data2. Zbog toga što je filtar 11 reda, potrebno je 11 podataka.



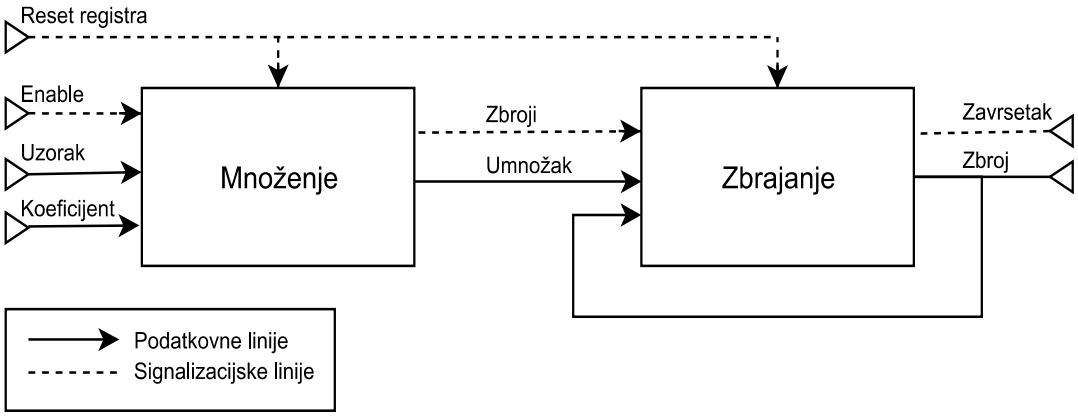

***Sl. 4.7*** *Signali unutar FIR blok entiteta*

## 4.4 FIR blok

Fir block entitet sastoji se od komponente koja množi koeficijent, koristeći aritmetiku fiksnog zareza s 32 bita, sa uzorkom audio signala te ga šalje u blok za zbrajanje kao što prikazuje slika 4.8.

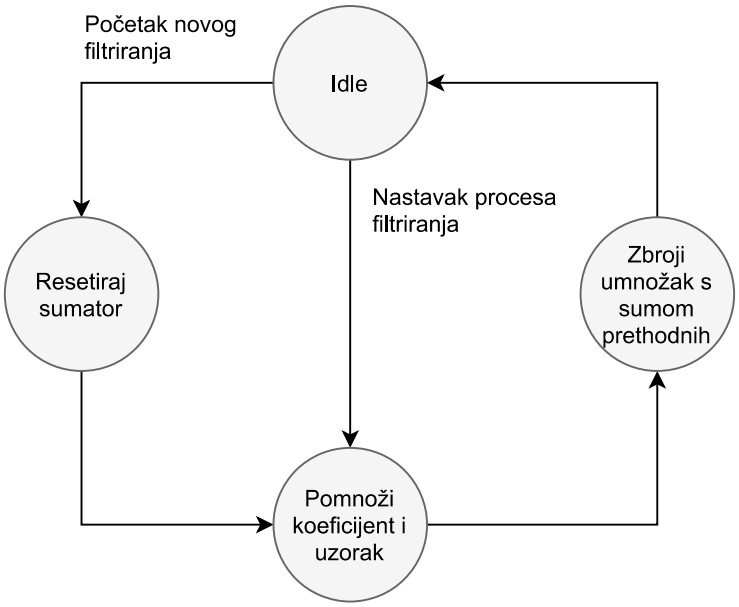
Pri početku procesa filtriranja uz pomoć signala Reset registra, svi registri unutar entiteta se postavljaju u 0, i moguće je započeti filtriranja. Ukoliko se ne provede resetiranje pri svakom novom računanju uzorka, dolazilo bi do miješanja trenutnog potrebnog stanja izlaza sa prošlim.

Nakon resetiranja, entitet Množenje na detekciju impulsa signala Enable provede množenje podataka koji se nalaze na ulazima Uzorak i Koeficijent. Nakon što se provede množenje, rezultat množenja šalje se signalom Umnožak do entieta Zbrajanje. Također se šalje impuls signalom Zbroji, kako bi zbrojio trenutni rezultat množenja sa prošlima.




**Sl. 4.8** Blok shema entiteta FIR blok

Zbroj svih umnožaka ostvaruje se zbog povratne veze entieta Zbrajanje, gdje se na signalu Zbroji nalazi rezultat dosadašnjih zbrajanja. Latencija FIR blok entiteta kako bi se izvršio proces množenja i zbrajanja jednog audio uzorka iznosi 3 takta.

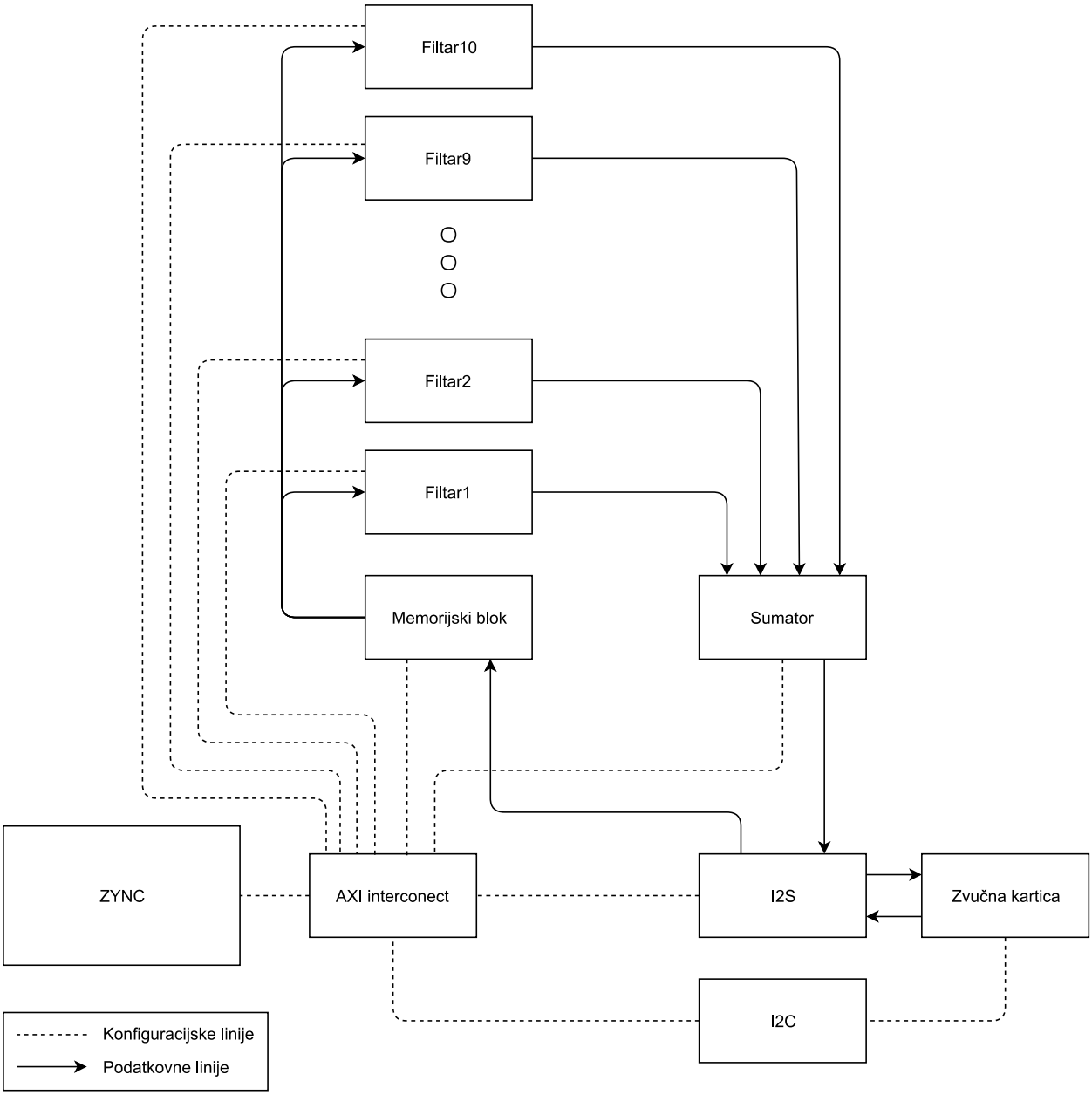



***Sl. 4.9*** *Dijagram toka za entitet FIR filtar*

## 4.5 Ekvilizator

Nakon implementacije FIR filtra tema rada proširena je na ekvilizator. Implementiran je ekvilizator koji podržava dvokanalni način rada te sadrži 10 FIR filtara kojima je moguće dinamički mijenjati red i koeficijente filtra.

Blokovski dijagram prema slici 4.10 prikazuje sve elemente ekvilizatora te kako su spojeni. Ispreakidane linije odnose se na konfiguracijske linije. Zync procesor šalje AXI protokolom podatke za konfiguraciju audio kodeka i I2S entiteta kako bi radili na željenoj frekvenciji uzorkovanja te sa 24 bitnom rezolucijom. Entitet Memorijski blok prima podatak o redu filtra kako bi definirao adrese i količinu uzoraka koje je potrebno spremiti u BRAM. Entiteti Filtar također primaju informaciju o redu filtra te podatke o svim koeficijentima filtra koje spremaju u BRAM. Sumator prima podatke o tome koliko je potrebno gušiti određene ulaze filtra.




***Sl. 4.10*** *Blok shema implementiranog ekvilizatora sa 10 FIR filtara u FPGA*

Podatkovne linije odnose se na linije protoka podataka uzoraka audio signala. Audio kodek prima podatke audio ulaza te ih šalje do I2S entiteta, koji šalje uzorke u entitet Memorijski blok. Memorijski blok sprema sve audio uzorke u BRAM, te ih čita iz BRAMA prilikom procesa filtriranja. Izlazi svih Filtar entiteta spajaju se na entitet Sumator, gdje se svaki ulaz dijeli s brojem definiranim konfiguracijskom linijom, kako bi ostvarila željena atenuacija signala. Zatim se svi ulazi sumiraju u jedan signal te šalju na ulaz I2S entiteta a zatim na audio kodek.

## 4.6 Računanje najvećeg mogućeg reda filtra

Glavno ograničenje za povećanje reda filtra u implementiranom dizajnu filtra je vrijeme i brzina rada sustava. Ograničenje vremena ovisi o frekvenciji uzorkovanja, jer proces filtriranja mora završiti prije dolaska novog uzorka.

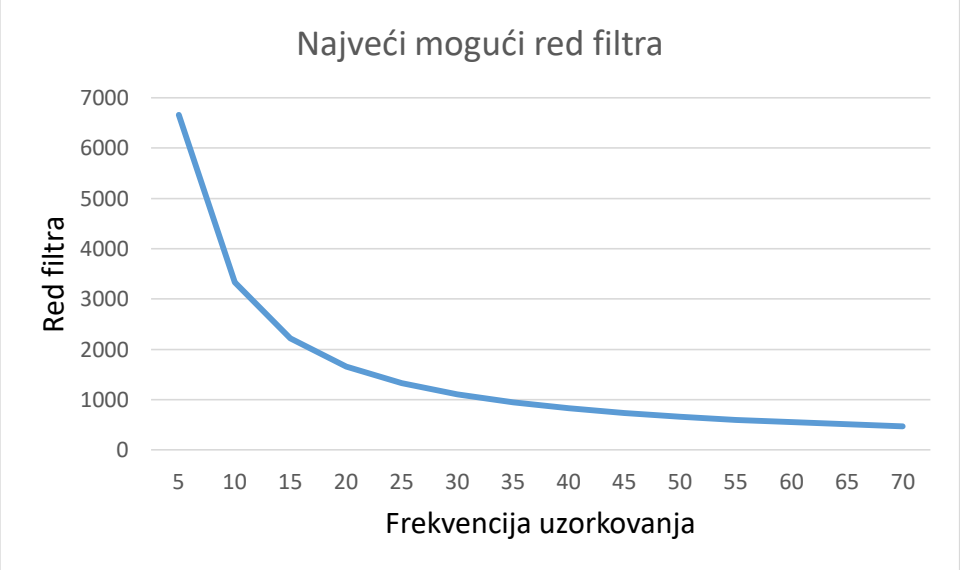
Testirani dizajn radi s frekvencijom uzorkovanja od 48kHz, dok je postavljena frekvencija rada FPGA 100 MHz. Dijeljenjem ta dva podatka dobiva se rezultat od 2083, koji govori kako FPGA ima 2083 taktova unutar kojeg je potrebno završiti proces filtriranja. Dizajn filtra je implementiran tako da pri početku i završetku filtriranja iskoristi 5 taktova, dok proces filtriranja jednog reda filtra zahtjeva 3 takta. Tim podacima dolazi se do relacije 4-1 kojom se računa maksimalni mogući red filtra.

$$N = \frac{Frekvencija\,FPGA}{Frekvencija\,uzorkovanja} * \frac{1}{3} - 5 \tag{4-1}$$

Uvrštavanjem  u formulu podatke implementiranog dizajna, gdje je 100 MHz frekvencija FPGA, te 48kHz frekvencija uzorkovanja audio signala, dobiva se da je najveći red filtra 689.

Želi li se veći red filtra moguće su dvije opcije, povećati frekvenciju FPGA ili smanjiti frekvenciju uzorkovanja audio signala.

Povećavanjem frekvencije FPGA, linearno će i rast najveći mogući red filtra. Promjenom frekvencije uzorkovanja red filtra ponaša se obrnuto proporcionalno kao što prikazuje slika 4.11.

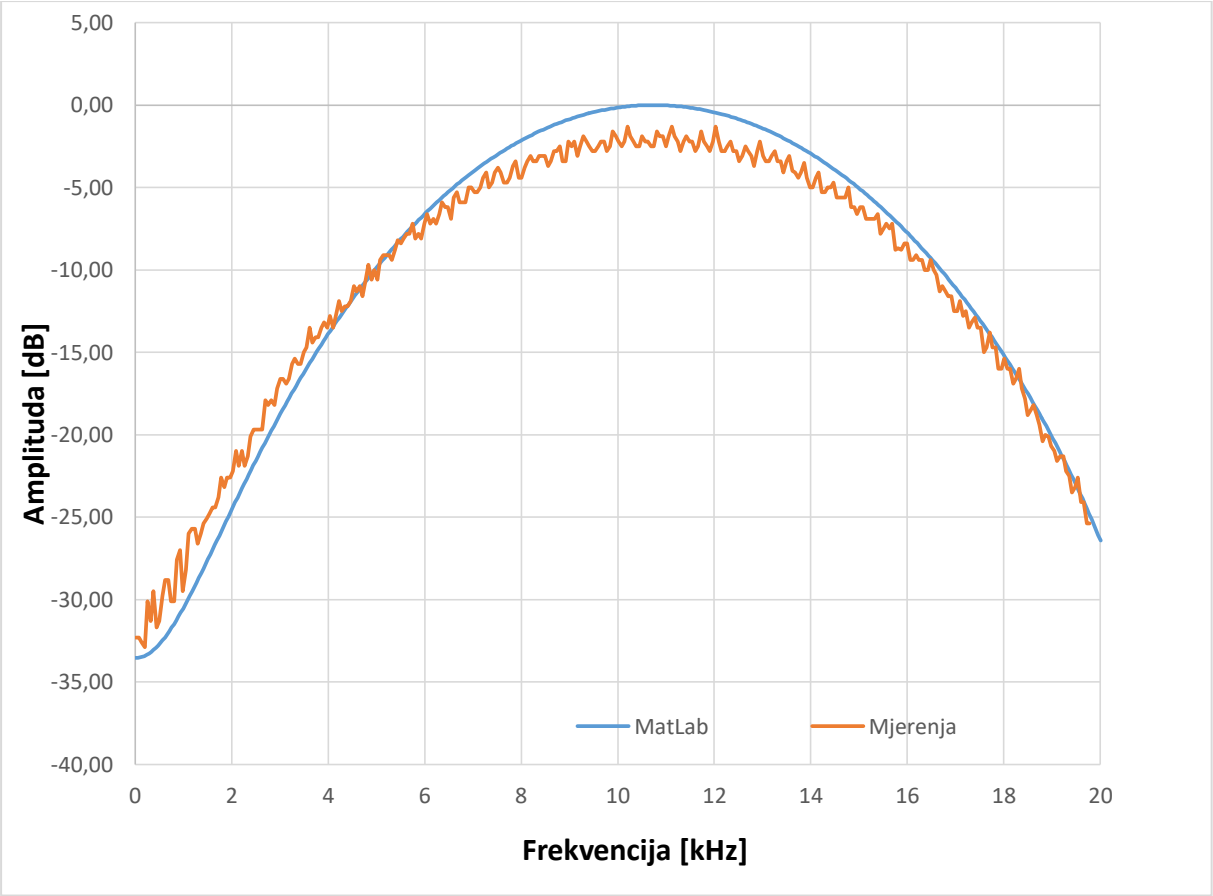



**Sl. 4.11** *Ovisnost najvećeg reda filtra o frekvenciji uzorkovanja audio signala*

## 4.7 Usporedba MatLab rezultata sa stvarnim rezultatima mjerenja

Test ispravnosti rada implementiranog filtra proveden je tako što su u MatLabu izračunati koeficijenti filtra proizvoljnih karakteristika. Zatim korištenjem prethodno izračunatih koeficijenata, u implementiranom dizajnu provedeno je mjerenje frekvencijskog odziva te usporedba s frekvencijskim odzivom MatLaba.

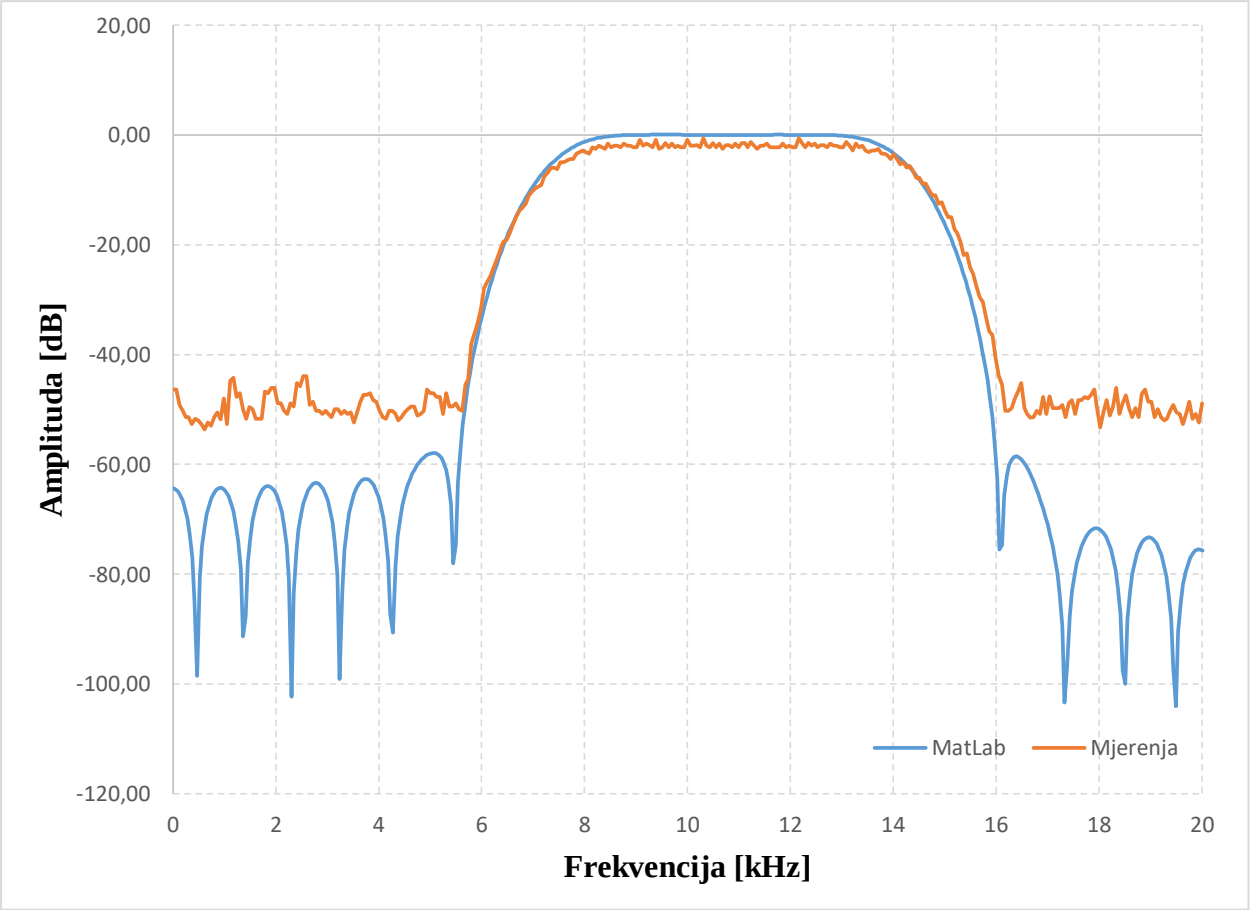
Odabrani filtar za testiranje je pojasnopropusni filtar gdje je niža granična frekvencija 7,2 kHz a viša 14,4 kHz. Prvo je provedeno testiranje filtra 10 reda. Slika 4.12 prikazuje frekvencijski odziv navedenog filtra u MatLabu te rezultati mjerenja.




***Sl. 4.12*** *MatLab simulacija frekvencijskog odziva pojasnopropusnog filtra 10 reda*

Usporedbom mjerenih rezultata s rezultatima iz MatLaba prema slici 4.12 može se vidjeti kako postoje odudaranja. Vrijednost najveće razlike amplitude je 3.88 dB na frekvenciji 1173 Hz, dok je standardna devijacija kroz cijeli mjereni spektar 1.5255 dB.

Provedena su daljina testiranja tako što je u MatLabu izveden nisko propusni filtar 50-og reda. Strmina filtra ovdje je veća od filtra 10-og reda kao što se vidi iz slike 4.13.




**Sl. 4.13** MatLab simulacija frekvencijskog odziva pojasnopropusnog filtra 50 reda

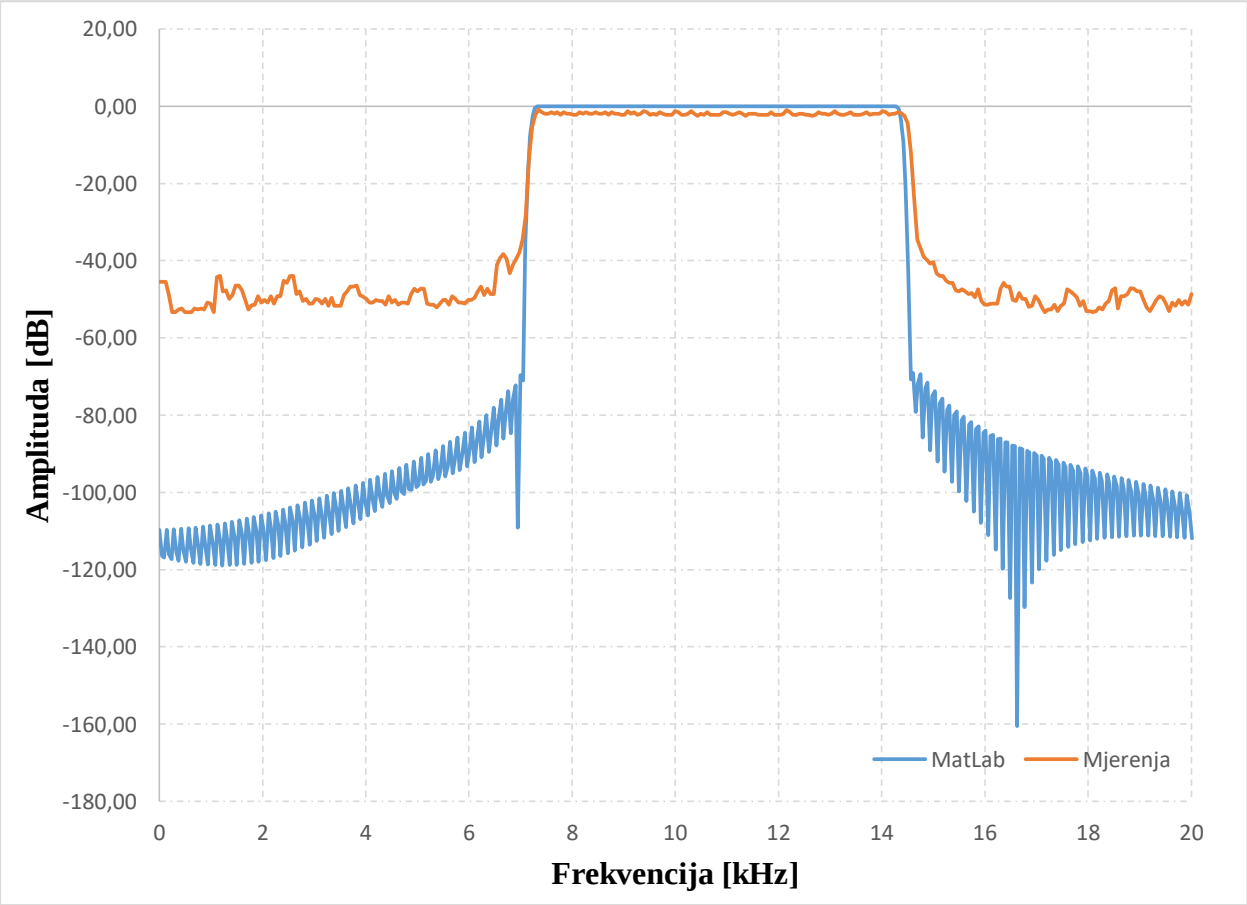
Mjerenje prijenosne karakteristike filtra izvedeno je istim postupkom kao kod filtra 10 reda. Frekvencijski odziv mjerenih rezultata na slici 4.13. podudara se s MatLab rezultatima.

Prema slici 4.13 može se zaključiti kako su najveća odstupanja mjerenih rezultata od rezultata MatLaba u području pojasa gušenja filtra gdje je amplituda signala ispod -45 dB. Navedena odstupanja se pojavljuju zbog kvantnog šuma te šuma analognih sklopova ADC i DAC audio codeca. Provedena je analiza za vrijednosti signala iznad -45dB, odnosno od 5.82 kHz do 15.96 kHz.

Maksimalno odstupanje unutar navedenog spektra je 15.04 dB no standardna devijacija kroz spektar je 1.944 dB. Provedena je i analiza odstupanja unutar propusnog pojasa filtra, unutar gornje i donje granične frekvencije. Unutar navedenog spektra standarda devijacija je 1.78 dB a

maksimalno odstupanje je 2.45 dB. Zaključuje se da su najveća odstupanja mjerenih rezultata od rezultata MatLaba nalaze u tranzicijskom pojasu i pojasu gušenja filtra.

Izvedeno je mjerenje 680-og reda filtra, jer je to maksimalni izračunati podržani red filtra. Kao što se vidi slikom 4.14. frekvencijski odziv ima znatno veću strminu od prijašnjih mjerenja.




**Sl. 4.145** *Rezultati mjerenja frekvencijskog odziva pojasnopropusnog filtra 680 reda*

Mjerenjem prijenosne karakteristike dobiven je frekvencijski odziv prikazan slikom 4.14. Prilikom mjerenja filtra 680-og, kao i kod filtra 50-og reda dolazi do značajnih odstupanja u rezultatima mjerenja na vrijednostima ispod -45 dB.

Usporedba mjerenih rezultata provedena je na  rezultata mjerenja s vrijednostima iznad -45 dB, odnosno u intervalu od 7.09 kHz do 14.69 kHz. Najveća odstupanja nalaze se na strminama kod donje i gornje granične frekvencije i najveće odstupanje iznosi 59.80. No standarda devijacija odstupanja je 5.4118.

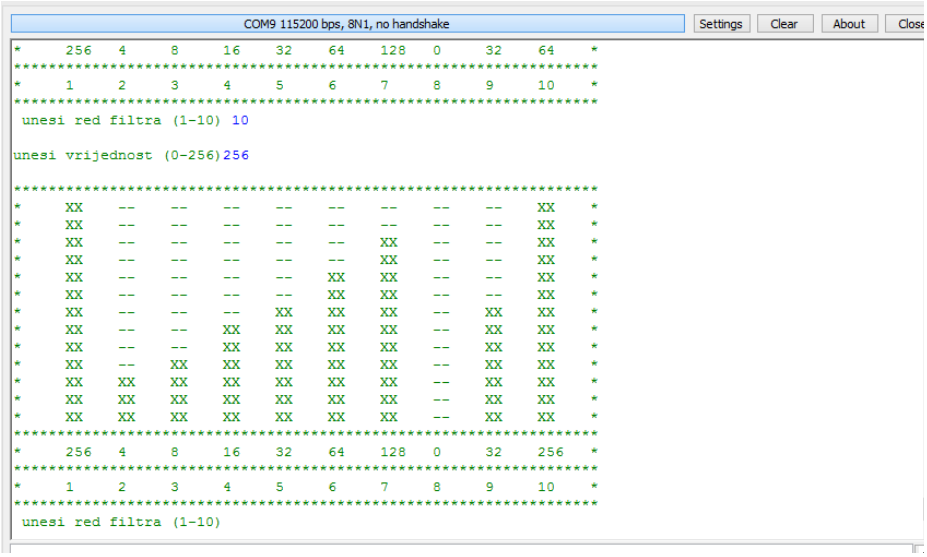
Provedena je analiza rezultata za frekvencijski spektar unutar definiranih graničnih frekvencija filtra. Standardna devijacija odstupanja je 1.92 dok je maksimalno odstupanje 2.44. Kao i kod slučaja filtra 50-og reda najveća odstupanja mjerenih rezultata s rezultatima MatLaba nalaze se unutar tranzicijskog pojasa i pojasa gušenja.

Iako postoje odstupanja mjerenih rezultata od rezultata simulacije MatLaba može se zaključiti kako dizajnirani filtar daje zadovoljavajuće rezultate. Odstupanja rezultata mjerenja u odnosu na simulaciju MatLaba pojavljuju se zbog nesavršenost mjerne opreme, nesavršenost elektroničkih komponenta, kvantizacijski šum, interferencija.

Mjerenjem filtra sa različitim iznosom njegovog reda zaključuje se kako je definiranje najvećeg filtra relacijom 4-1 zadovoljavajuće. Svi testirani filtri, uključujući i filtar maksimalnog podržanog reda ponašaju se u skladu sa MatLab predviđanjima.

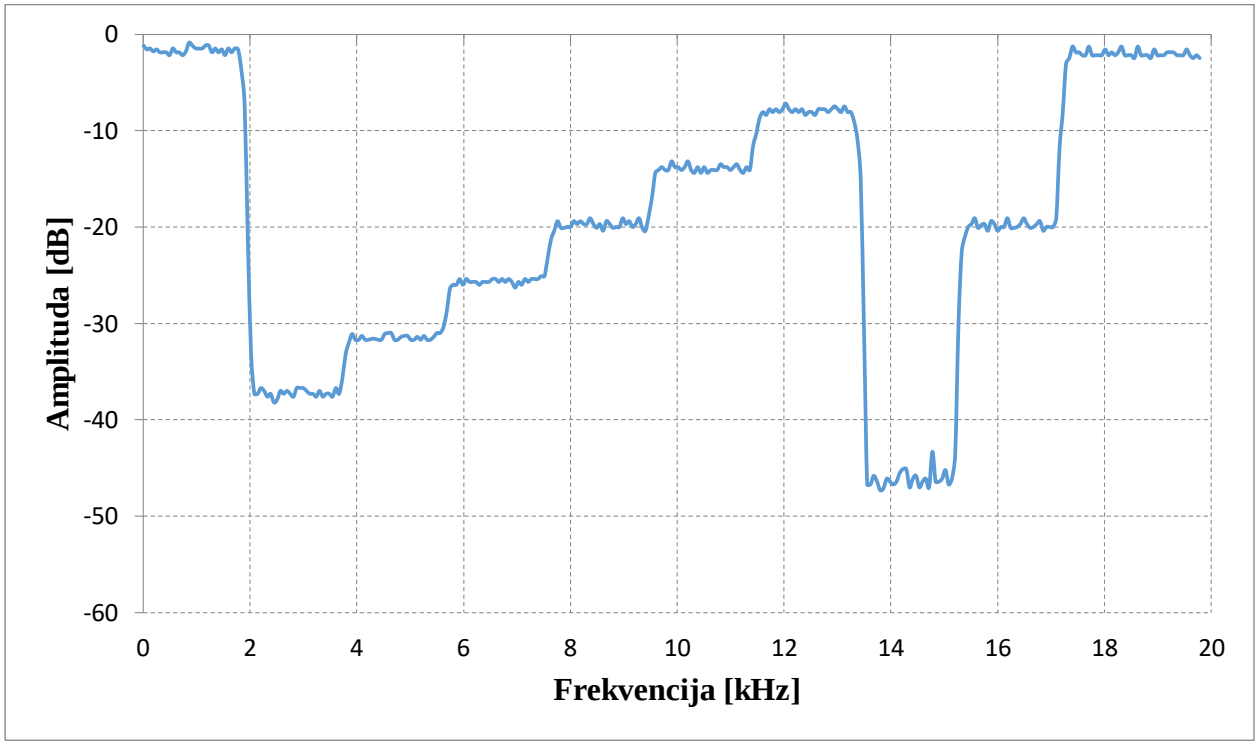
Posljednje mjerenje je frekvencijski odziv ekvilizatora. Ekvilizator se sastoji od 10 pojasno propusnih filtra spojenih paralelno, gdje se svakom filtru koeficijenti mogu mijenjati uz pomoć procesora. Za potrebe testiranja koeficijenti filtra su izvedeni tako da spektar od 20kHz dijeli u 10 pojaseva širine 2 kHz. Izlaz svakog filtra moguće je gušiti tako što uzorak izlaza filtra dijeli sa 8 bitnom rezolucijom.

Unos gušenja za pojedini filtar unosi se UART protokolom, gdje se prvo unosi redni broj filtra kojem se želi promijeniti gušenje, zatim se unosi broj od 0-256 koji definira gušenje. Nakon unosa gušenja ispisuje se predviđanje izgleda prijenosne karakteristike kao što prikazuje slika 4.15.




***Sl. 4.15*** *Unos gušenja pojedinog filtra i prikaz očekivanog frekvencijskog odziva UART terminalom*

Izvršeno je mjerenje prijenosne karakteristike ekvilizatora te usporediti sa predviđanjem iz slike 4.15, slika 4.16. prikazuje dobiveni frekvencijski odziv koji sadrži krivulju koja je u skladu sa predviđanjem.




***Sl. 4.16*** *Rezultati mjerenja prijenosne karakteristike ekvilizatora*

## 4.8 Mjerenje iskorištenosti resursa

Bitna stavka pri implementaciji dizajna na FPGA je količina iskorištenih resursa za postizanje funkcionalnosti. Tablica 4.1. prikazuje koliki je postotak iskorištenosti resursa kada je implementiran samo ekvilizator sa 1 FIR filtar. Tablica prikazuje su najveće iskorištenosti LUT elementa 16%, čime se zaključuje kako je moguće implementirati više filtra u dizajn.

***Tab. 4.1*** *Postotak iskorištenosti resursa pri implementaciji 1 filtra*

| | Iskorišteno | Dostupno | Postotak |
|---|---|---|---|
| LUT | 2856 | 17600 | 16,23 |
| LUTRAM | 74 | 6000 | 1,23 |
| FF | 4698 | 35200 | 13,35 |
| BRAM | 9,5 | 60 | 15,83 |
| DSP | 5 | 80 | 6,25 |
| IO | 12 | 100 | 12 |
| BUFG | 2 | 32 | 6,25 |
| PLL | 1 | 2 | 50 |

Drugi izvedeni dizajn je ekvilizator sa 10 filtra gdje tablica 4.2. prikazuje iskorištenost resursa U ovom slučaju očekivano je iskorištenost resursa veća, ovaj put je najveća iskorištenost DSP blokova 51,25%. Iskorištenost svih resursa je daleko od 100% što govori kako je moguće implementirati više od 10 filtra.

***Tab. 4.2*** *Postotak iskorištenosti resursa pri implementaciji ekvilizator sa 10 filtra*

| | Iskorišteno | Dostupno | Postotak |
|---|---|---|---|
| LUT | 6730 | 17600 | 38,24 |
| LUTRAM | 74 | 6000 | 1,23 |
| FF | 9450 | 35200 | 26,85 |
| BRAM | 18,5 | 60 | 30,83 |
| DSP | 41 | 80 | 51,25 |
| IO | 12 | 100 | 12 |
| BUFG | 2 | 32 | 6,25 |
| PLL | 1 | 2 | 50 |

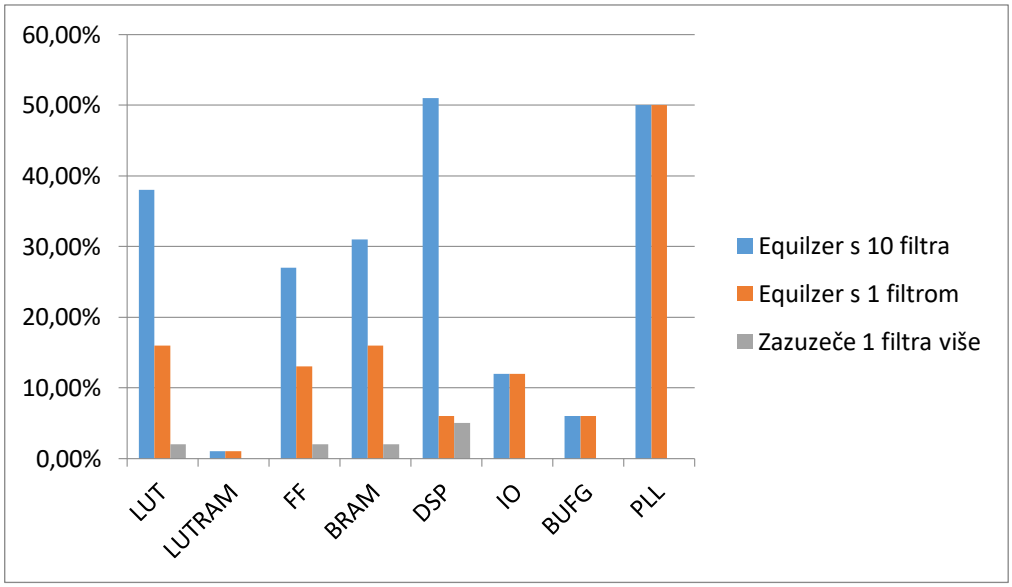
Izvršeno je mjerenje kojim se dobiva podatak o zauzeću FIR filtar entiteta. Tablica 4.3. prikazuje kako je unutar FIR filtar entiteta, najveća iskoristivost DSP elementa s 5%. To bi značilo kako je moguće implementirati ekvilizator sa više od 10 filtra. Potrebno je napomenuti kako je odabir reda filtra nema utjecaj na iskorištenost resursa. Entitet za spremanje uzoraka unutar FIR filtra

dizajniran je tako da alocira broj memorijskih lokacija koji je određen maksimalnim redom filtra, neovisno o redu filtra koji se trenutno koristi.

***Tab. 4.3*** *Postotak potrebnih resursa 1 filtra*

| | Iskorišteno | Dostupno | Postotak |
|---|---|---|---|
| LUT | 430,44 | 17600 | 2,45 |
| LUTRAM | 0 | 6000 | 0 |
| FF | 528 | 35200 | 1,5 |
| BRAM | 1 | 60 | 1,67 |
| DSP | 4 | 80 | 5 |
| IO | 0 | 100 | 0 |
| BUFG | 0 | 32 | 0 |
| PLL | 0 | 2 | 0 |

Uvid koliko se resursa zauzima implementacijom FIR filtra u FPGA može se promotriti prema slici 4.17. Za implementaciju filtra koriste se LUT elementi, bistabili, BRAM te DSP blokovi. Najveća je iskoristivost DSP blokova, i oni su ograničavajući faktor o kojem ovisi broj filtara koji je moguće implementirati.




***Sl. 4.17*** *Graf usporedbe iskorištenosti resursa*

# 5 ZAKLJUČAK

Rad demonstrira dizajn i implementacija FIR filtra uz pomoć FPGA. Konfiguracija filtra kojom se određuje red i koeficijenti filtra vrši se uz pomoć Zynq procesora. Dizajn podržava više frekvencija uzorkovanja audio kodeka, gdje se promjenom frekvencije uzorkovanja mijenja maksimalni mogući red filtra.

Prvobitno je implementiran jedan FIR filtar koji pri frekvenciji uzorkovanja od 48kHz podržava maksimalni red filtra od 689. Izvršene su simulacije frekvencijskog odziva u MatLabu, te usporedba sa rezultatima mjerenja. Dolazi se do zaključka kako je filtar dobro implementiran, jer je prijenosna karakteristika filtra za razne redove filtra i razne koeficijente u skladu s rezultatima simulacije.

Kako je implementiranjem jednog FIR filtra iskorišten mali udio resursa, izveden je ekvilizator koji sadrži deset FIR filtra spojenih paralelno. Testiranjem rada ekvilizatora dolazi se do zaključka kako je njegova prijenosna karakteristika u skladu s korisnički definiranim zahtjevima.

Na temelju svih rezultata mjerenja zaključuje se da su ispunjeni svi zahtjevi rada te da je implementacija filtra uspješna.

## POPIS SKRAĆENICA

| | |
|---|---|
| ADC (Analog to digital converter) | -Analogno digitalni pretvornik |
| DAC (Digital to analog converter) | -Digitalno analogni pretvornik |
| DSP (Digital signal processor) | -Procesor za digitalnu obradu signala |
| LTI (Linear time invariant) | -Linearni vremenski nepromjenjiv |
| FIR (Finitive impulse response) | -Konačni impulsni odziv |
| IIR (Infinite impulse response) | -Beskonačni impulsni odziv |
| FPGA (Field programable gate array) | -Polje programibilnih logičkih blokova |
| LUT (Look up table) | -Pregledna tablica |
| CLB (Configurable logic block) | -Konfigurabilni logički blok |
| LAB (Logic array block) | -Blok logičkih polja |
| FPGA (Field programable gate array) | -Polje programibilnih logičkih vrata |
| HDL (Hardware description langugage) | -Jezik za opis hardvera |
| APU (Applicatin Processor) | -Jedinica za obradu aplikacije |

# LITERATURA


[1] S. W. Smith, *The Scientist and Engineering's Guide to Digital Signal Processing*, California Techical Publishing, 1997

[2] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, third Edition, Springer, 2007

[3] R. Yates, *Fixed-Point Arithmetic: An Indtroduction*, Digital Signal Labs, 2007.

[4] Xilinx Inc., *Reduce Power and Cost by Converting from Floating Point to Fixed Point*, 2017.

[5] R. Kaster, J. Matai, S. Neuendorffer, *Parallel Programing for FPGA*, Arxiv, 2018.

[6] Xilinx Inc., *Zynq-7000 SoC Data Sheet: Overview*, 2018.

[7] Digilent INC., *ZYBO bord reference manual*, 2017

[8] W. Kafig, *VHDL 101 Everthing you need to know to get started*, Newnes, 2011.

[9] P. P. Chu, *Embeded SoPC design with Nios II processor and VHDL Examples*, Wiley, 2011.

[10] Mentor, *HDL Simulation ModelSim PE DataSheet*, 2018.

[11] Xilinx Inc., *Vivado Design Suite User Manual*, 2018.

[12] Xilinx Inc., *Embeded System Tools Reference Manual*, 2018.

[13] Analog Devices INC., *Low Power Audio Codec SSM2603*, 2013.

[14] NPX Semiconductors, *$I^2C$ – bus specification and user manual*, 2014.

[15] Philips Semiconductors, *$I^2S$ specifications*, 1996

[16] Xilinx Inc., *AXI reference guide*, 2011.

[17] Xilinx Inc., *Block Memory Generator v8.4 LogiCORE IP Product Guide*, 2017.

# SAŽETAK

Naslov: Implementacija FIR filtra zasnivanog na FPGA za digitalnu obradu zvuka u realnom vremenu

U radu opisan je postupak dizajniranja FIR digitalnog filtra, te njegova implementacija u FPGA. Pojašnjene su korištenih tehnologija te opisan rad svakog entiteta dizajna. Provedene su simulacije filtra u MatLabu te usporedba s mjerenim rezultatima implementiranog filtra. Dizajn je izveden u Vivado razvojnom okruženju na Zynq-7000 SoC-u, gdje se konfiguracija parametra filtra vrši uz pomoć procesora. Rad je proširen na ekvilizator s deset FIR filtara koji rade paralelno.

Ključne riječi : FIR filtar, FPGA, VHDL, Zybo, Zynq, Ekvilizator, BRAM, AXI, DSP.

## ABSTRACT

Title: FIR filter implementation based on FPGA for real-time digital signal sound processing

The paper explains the process of designing an FIR filter and his implementation in the FPGA. The used technologies and entities for the design are further explained. Simulations of the filter's behavior are made in MatLab and compared to the measured frequency response of the implemented filter. Vivado Design Suite is used to create the design on the Zynq-7000 SoC, where the configuration of the filter is done with the processor. The paper is expanded to the Equilizer that has 10 FIR filters working parallelly.



Key words: FIR filter, FPGA, VHDL, Zybo, Zynq, Equilizer, BRAM, AXI, DSP.

# ŽIVOTOPIS

Krunoslav Krajcar rođen je 22. Siječnja 1994. Godine u Bjelovaru od majke Blaženke i oca Josipa. Pohađa osnovnu školu u Šandrovcu i Veliko Trojstvu, nakon koje upisuje Tehničku školu Bjelovar, smjer Elektrotehničar. Po završetku srednje škole upisuje preddiplomski studij Elektrotehnike na Elektrotehničkom fakultetu Osijek. Na istom fakultetu upisuje diplomski studij Elektrotehnike, smjer Komunikacijske tehnologije. Tijekom posljednje godine studija započinje suradnju s institutom „RT-RK Osijek“ gdje radi na diplomskom radu vezanom za FPGA.